Reverse Engineering MineSweeper Windows Application

Final report for the Software Reverse Engineering Project
San Jose State University
Fall 2019
Under guidance of
Prof Auston Davis

Name: Rekha Rani
SJSU ID: 013828828
e-mail: rekha.rani@sjsu.edu

Name: Lolitha Sresta Tupadha
SJSU ID: 014321307
e-mail: lolithasresta.tupadha@sjsu.edu

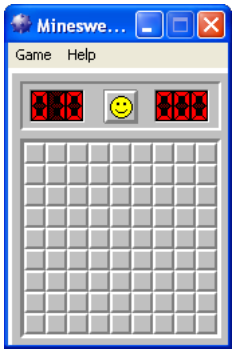
TABLE OF CONTENTS

ABSTRACT

Reverse Engineering is the process by which a product is understood by deconstructing it without having any prior information about products architecture, internal working details. As a Cryptology student, Reverse Engineering a product can help us in identifying any vulnerabilities in the product that hackers can take advantage of, and act on that accordingly.

MINESWEEPER VIDEO GAME (WINDOW APPLICATION)

- For this project, we chose to reverse engineer windows Minesweeper application.
- Minesweeper is a single-player puzzle computer game.
- It is based on the concept of Land mines. Land mine is an explosive mine laid on or just under the surface of the ground.
- The objective of the game is to clear a rectangular board containing hidden "mines" or bombs without detonating any of them, with help from clues about the number of neighboring mines in each field.
- How to win the game?
  - A player can with the game if he can identify all the squares without clicking the mines.



WHY MINESWEEPER FOR SOFTWARE REVERSE ENGINNERING PROJECT ?

- For this project, we chose to reverse engineer windows Video game application Minesweeper.
- Minesweeper: A game which we played a lot, but do not know much about the logic that goes behind it. We wanted to know about its logic.
- Games can often times be very frustrating. This frustration stems from the inherent fact that games, by design, present many unknowns to the player. Many develop cheats to obtain an unfair advantage. For us, however, have an entirely different motivation – the challenge it involves.
- Some bad features of a game need to be designed out.

- Resulting knowledge gained through the reverse-engineering this game can be applied to the design of similar games.
- We can learn from the shortcomings of existing designs.

TOOLS USED

- IDA Pro – For static analysis
- OllyDbg – For dynamic analysis

APPROACH

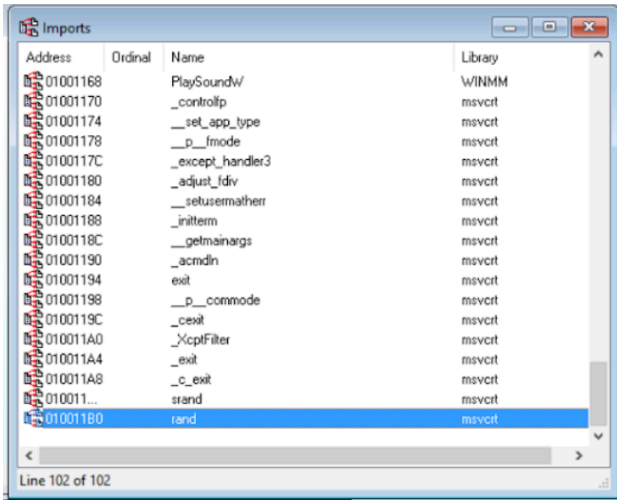
Suppose we have a minefield somewhere is the memory of the Minesweeper process. This minefield must contain the information regarding mine locations. If we manage to somehow change this information, so that a square with a mine will be marked with a flag we can know the location of mines and avoid clicking the mines.
In order to achieve this, we need to find information regarding mines in memory dump.

VULNERABILITIES FOUND

1. **Random Function:**

   Finding the Minefield:
   How would we determine which squares contain mines and which do not in a minefield? Since this decision should yield different results with every run of the game, we can assume that some randomization function is used to generate the minefield.

   Import tab in IDA pro



rand_caller Function:

- rand_caller function is used to draw a flag, a mine or an empty square
- It receives one argument which IDA calls arg_0.
- It calls rand which returns a random integer in the EAX register.
- Then cdq is called. It expands the value in EAX such that this value is stored in EDX:EAX.
- EDX is moved to EAX right before the function returns implies that the relevant value is the division remainder .
- When idiv is called, it takes the value in EDX:EAX and divides it by the function's argument So, after this division, the quotient resides in EAX whereas the remainder (modulo) is in EDX.
- To sum up, rand_caller randomizes a number and then performs a modulo, which makes sure the result does not exceed the value in arg_0

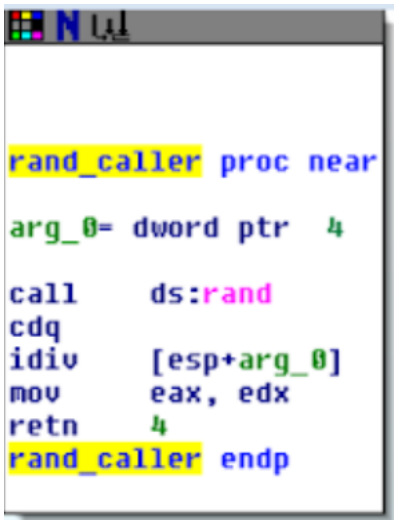


Fig. 1

2. Using IDA Pro, we can identify the rand function and see the references to the rand function. In Figure 2, we can see that rand_caller is being called twice.

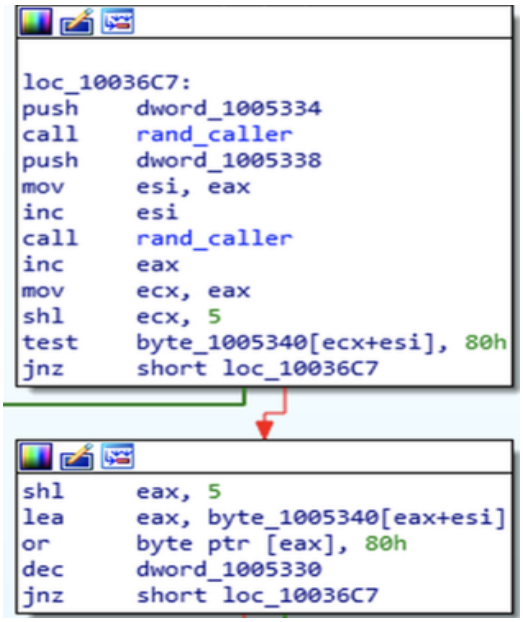
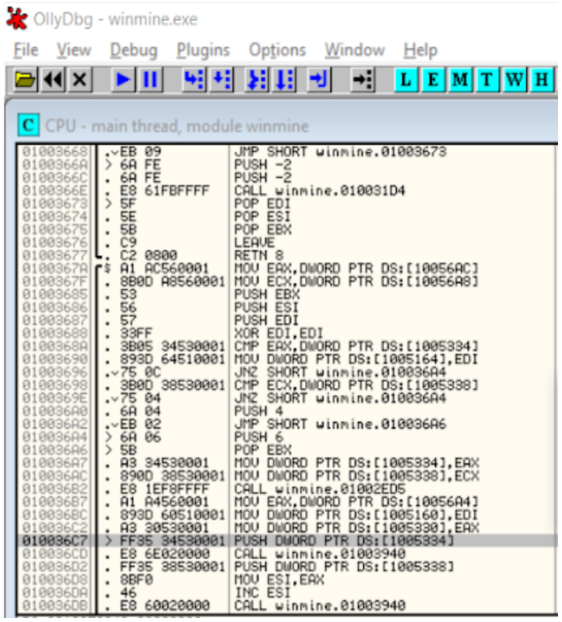


Fig. 2

- We went to the second location in the code where rand_caller is called
- The call appeared in a block which was a part of a loop
- This strengthened our assumption that the initial board was drawn using this function
- eax + esi is used as an offset of some memory location ***byte_1005340***

3. Using OllyDbg we can find out which values are passed as parameters to rand_caller



- In the disassembly window, the first push instruction is stored at address 0x10036C7
- Hit F2 to put a breakpoint
- Run minesweeper.exe until the breakpoint we just placed is hit
- The call which follows the push is the call to rand_caller
- we randomized a memory location and read the byte stored in this location

4. By examining the memory address in hex dump, we can easily find out mine location and empty location

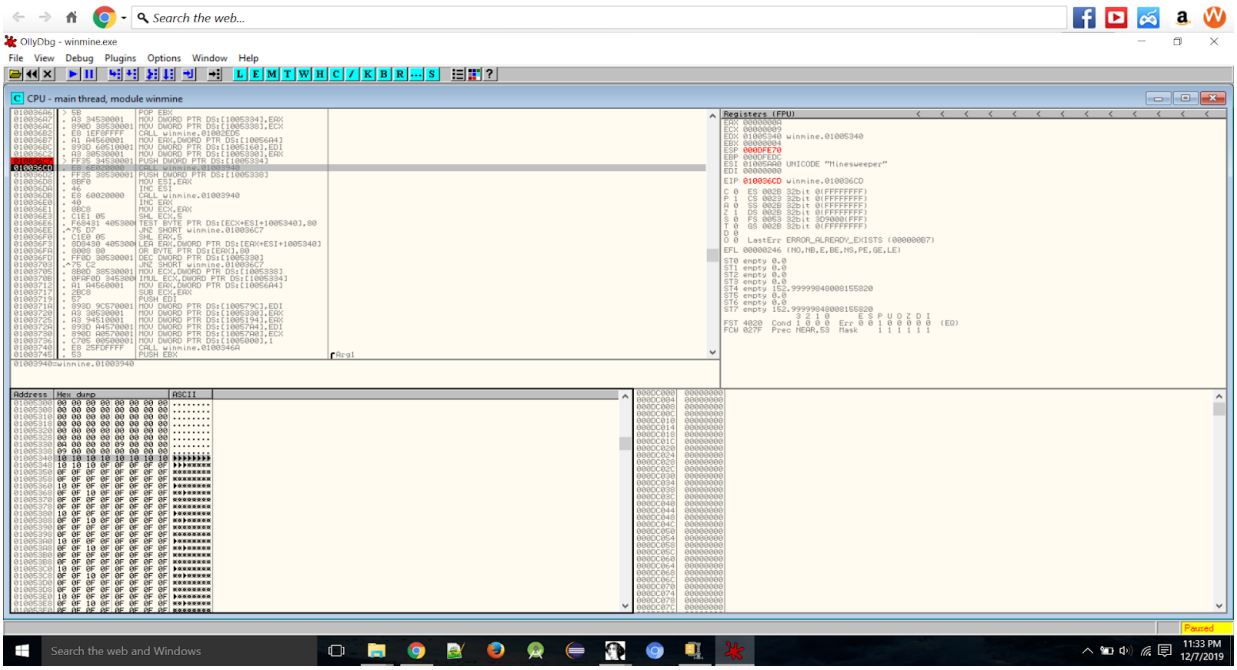


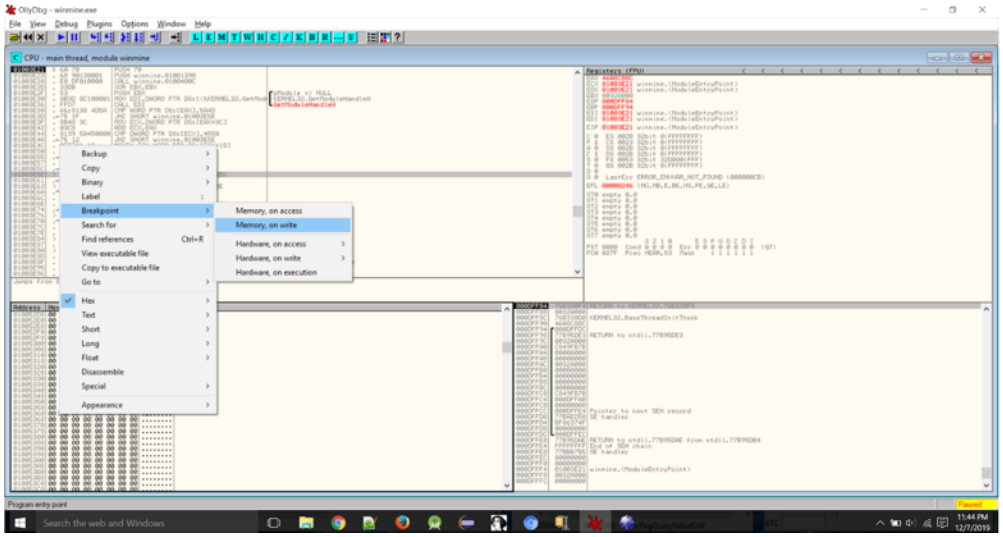
Figure 3

We examined the address 0x1005340 in memory and found something that looked a lot like a minefield

- 11 times 10 followed by 21 times 0F (32 bytes in total)
- a single 10 (row delimiters)
- a sequence of 9 times 0F (9 squares)
- another single 10
- a sequence of 21 times 0F

❑ the base address - 0x1005340 - is the address of our minefield
❑ First square of the board = 0x1005340 + 33 (0x21) = **0x1005361**
❑ **0F is an empty square**
❑ **8F is a mine**

5. We can place the flags at the location of mines



- Right-click on this address 0x1003E61 in the memory window and choose breakpoint > memory, on write.
- Now whenever a value is written to this location - the program will stop right before executing this write
- we placed a mine by:
  0xF or 0x80 -> 0x8F (10001111)

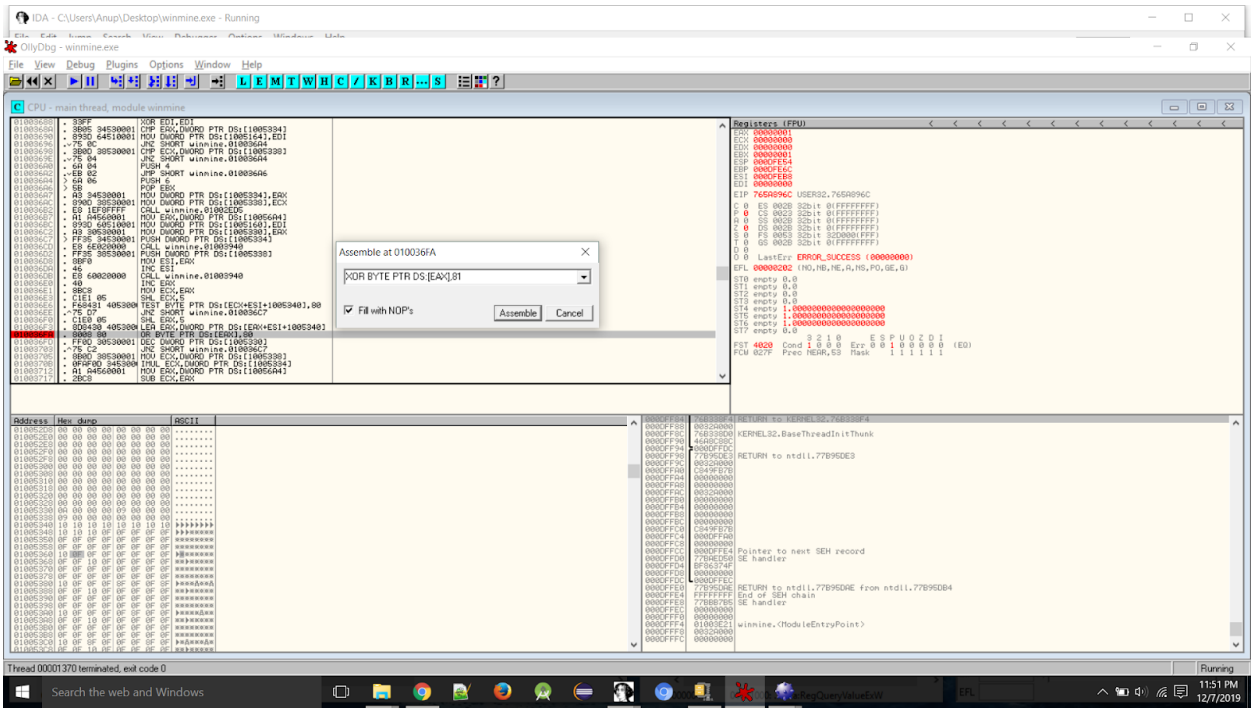
At this location we have to place a flag.

```
DEC EAX
MOV BYTE PTR DS:[EAX+1005340],0F
JNZ SHORT winmine.01002EDA
MOV ECX,DWORD PTR DS:[1005334]
MOV EDX,DWORD PTR DS:[1005338]
LEA EAX,DWORD PTR DS:[ECX+2]
TEST EAX,EAX
PUSH ESI
JE SHORT winmine.01002F11
MOV ESI,EDX
SHL ESI,5
LEA ESI,DWORD PTR DS:[ESI+1005360]
DEC EAX
```

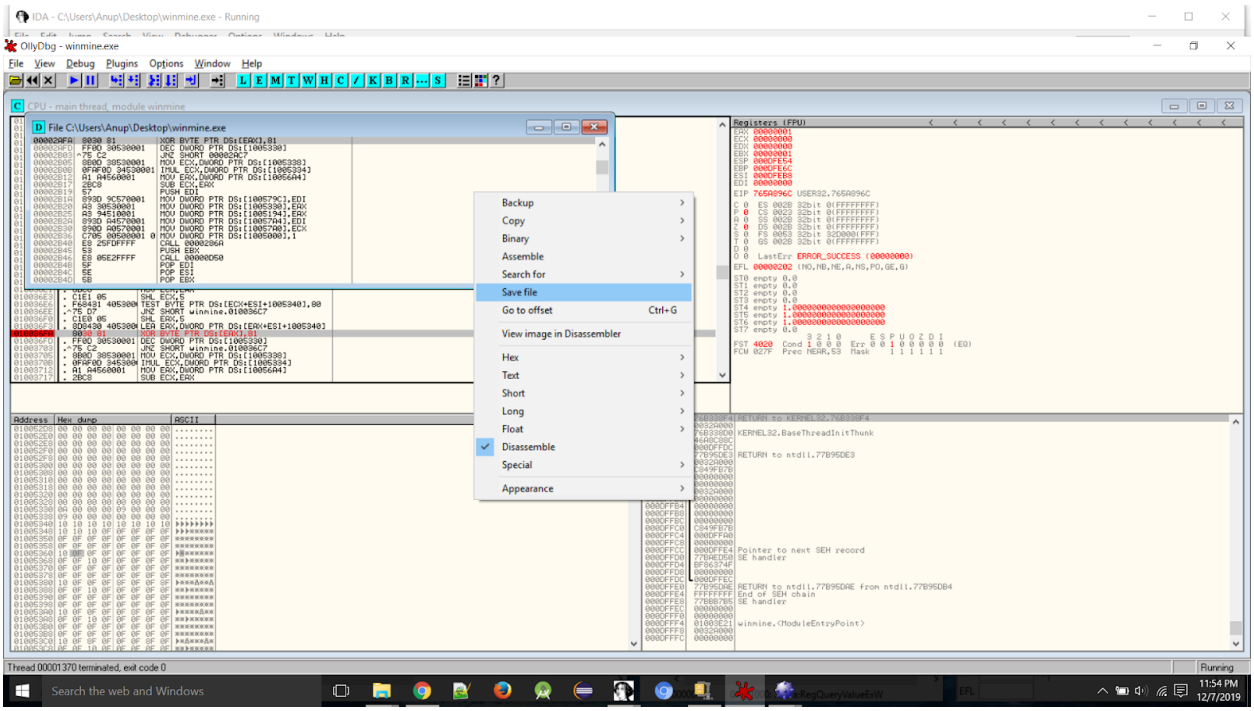
- Run the Minesweeper and Right-click on your chosen square to put a flag on it

ADDING A PATCH

- Goal – to print flags where there are mines
- Find in OllyDbg the address **0x10036FA** where the OR instruction is executed and click on it
- Hit the spacebar while the line is highlighted. A popup titled "Assemble at 010036FA" will appear.
- **0x8F in binary** - 10001111
  0x0F in binary - 00001111
- We wanted to turn 0x0F into 0x8E (10001110)
- Type in XOR instead of OR, and 81 instead of 80
- 0x0F **XOR** 0x81 -> **0x8E**
  00001111 XOR 10000001 =10001110

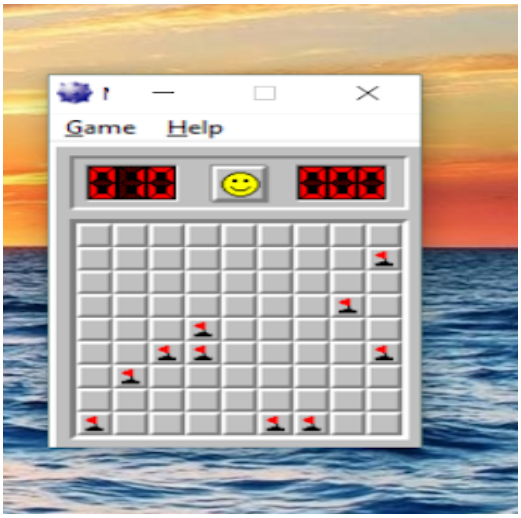


- Right-click on the disassembly window and choose "Copy to executable"
- Click on "Copy all".
- Right-click again, choose "Save file" and specify location and name of the newly patched file.
- Open the new patch exe. file from its location



PATCHED MINESWEEPER

In the patched version of minesweeper, mines are highlighted with flags. Thus, we can know the location of mines in the game before only and avoid clicking them to win the game.



HOW THE ORIGINAL DEVELOPER CAN PATCH THESE VULNERABILITIES ?

As we have seen the Vulnerabilities and they can be taken advantage of, in the previous section, now we can think about what a developer can do so that his software is more secure to such vulnerabilities.

- As we remember, we started with finding rand keyword in the code. This is because the developer has used previously defined random function in his logic. Instead of that a developer can implement his **own random function** that does not use any keywords.

- Also, we were able to decode a lot of information from the memory window as explained above. So, developer can be careful and use **Address Space Layout randomization (ASLR)** for the memory protection process.

- Since, the location of mines was determined when the game is loaded only, we were able to see the mines location in the hex memory window. Here, developer can implement it in such a way that mines position is determined dynamically during the runtime not statically. That is the location of mine might change as the game progress or as player keeps identifying plane grids. This way, the attacker cannot determine about the mine location, prior to the game.

- Developer could obfuscate the various functions which give the hacker the hint of program workflow.

CONCLUSION

Throughout this document the many concepts required to successfully locate, comprehend, and manipulate the Minesweeper playing grid has been exposed. As such, many details surrounding these concepts were neglected for the sake of brevity. In order to obtain a more holistic view of the covered concepts, it is encouraged to read those items articulated in the reference section and seek out additional works.

REFERENCES

- https://www.hex-rays.com/products/ida/
- http://www.ollydbg.de/
- https://en.wikipedia.org/wiki/Microsoft_Minesweeper
- https://www.i-programmer.info/news/150-training-a-education/12772-a-reverse-engineering-workshop-for-beginners.html