Software Security via Program Analysis

HW2 GodMode Minesweeper

Yu-Sheng Wang (bqk3dj)

1. With the understanding of the game's mechanism, we can assume that the program will create a global map in heap memory and assign values to each location. Therefore, we can first monitor the memory-write behavior.

   I insert a callback function *RecordMemWriteAfter()* that records the offset address every time a certain memory-write behavior is executed *(g_accessMap[offset]++)*. However, it turns out that lots of memory-write has been executed in this program.

2. Examine the first memory-write offset: 11d1 we find an operation to push value into stack. In fact, this kind of operations will not be our goals. Therefore, we can narrow down the search space by filtering out all the instructions with "push r".

```
const char* pszInst = strInst.c_str();
if (strstr(pszInst, "push r") == pszInst) {
    return;
}
```

3. Furthermore, since we have assumed that the locations of the mines should be stored in heap memory section, we can ignore all the memory-write operations that move the values into stacks.

4. Now, we have 7 possible candidates:

```
910   offset: 15a8, max-hitcount: 36
911   offset: 15cc, max-hitcount: 36
912   offset: 15ed, max-hitcount: 36
913   offset: 1616, max-hitcount: 36
914   offset: 1637, max-hitcount: 36
915   offset: 165c, max-hitcount: 36
916   offset: 16e5, max-hitcount: 36
```

   Using `grep <xxxx> log.txt` we grep more details about each instruction. Here we find out that instruction 1616 assigns value either 0 or 1, which can be consider assign mines to different locations.

5. From the last step, we have located the instruction of assigning the mine, whose offset location is 0x1616. To write down the values, we use `grep -n 1616 log.txt > cs6501_mine.txt` to record all the information, such as memory address and mine information.