Software Security via Program Analysis

Protect Against hack (HW3)

Yu-Sheng Wang (bqk3dj)

Pre-Work

- Change the makefile: Add debug information (-g) and remove optimization

```
1   flappybird : flappybird.cpp
2       g++ -g -o $@ $^ -lncurses
3
4   clean:
5       rm -f flappybird
```

- Disable/Enable Randomize:

  echo 0 | sudo tee /proc/sys/kernel/randomize_va_space

- Change *collision* to *s_collision & r_collision* in flappybird.cpp via macros.

Pintool Program

Search *s_collision* in *flappyBird.S*. The return value %eax is stored in -0xe4(%rbp).

Trace instructions at 1c5d & 1d03.

```
936       s_collision = controlCollision(pipeCol1, birdCol, birdRow, crackStart1,
937     1c35:   8b bd d0 fe ff ff       mov    -0x130(%rbp),%edi
938     1c3b:   8b 8d cc fe ff ff       mov    -0x134(%rbp),%ecx
939     1c41:   8b 95 ec fe ff ff       mov    -0x114(%rbp),%edx
940     1c47:   8b b5 14 ff ff ff       mov    -0xec(%rbp),%esi
941     1c4d:   8b 85 f0 fe ff ff       mov    -0x110(%rbp),%eax
942     1c53:   41 89 f8                mov    %edi,%r8d
943     1c56:   89 c7                   mov    %eax,%edi
944     1c58:   e8 3d 09 00 00          call   259a <_Z16controlCollisioniiiii>
945     1c5d:   89 85 1c ff ff ff       mov    %eax,-0xe4(%rbp)
```

```
993       s_collision = controlCollision(pipeCol2, birdCol, birdRow, crackStart2,
994     1cdb:   8b bd d8 fe ff ff       mov    -0x128(%rbp),%edi
995     1ce1:   8b 8d d4 fe ff ff       mov    -0x12c(%rbp),%ecx
996     1ce7:   8b 95 ec fe ff ff       mov    -0x114(%rbp),%edx
997     1ced:   8b b5 14 ff ff ff       mov    -0xec(%rbp),%esi
998     1cf3:   8b 85 f4 fe ff ff       mov    -0x10c(%rbp),%eax
999     1cf9:   41 89 f8                mov    %edi,%r8d
1000    1cfc:   89 c7                   mov    %eax,%edi
1001    1cfe:   e8 97 08 00 00          call   259a <_Z16controlCollisioniiiii>
1002    1d03:   89 85 1c ff ff ff       mov    %eax,-0xe4(%rbp)
```

```
1051            // TOUCH THE GROUND
1052            isOver = true;
1053    1d92:   c7 85 e4 fe ff ff 01    movl   $0x1,-0x11c(%rbp)
1054    1d99:   00 00 00
```

Profile

For the located instructions, we monitor the values via the following code snippets.

```
165 VOID RecordMemWriteAfter_Profile(VOID * ip, VOID * addr, UINT32 size, ADDRINT* regRSP)
166 {
167     //ADDRINT* ipData = (ADDRINT*)ip;
168     ADDRINT offset = (ADDRINT)ip - g_addrLow;
169
170     //if (IsStackMem_Heuristic(*regRSP, (ADDRINT)addr)) return;   // If goes to stack memory, skip
171
172     g_accessMap[offset]++;
173
174     //log("[MEMWRITE(AFTER)] %p (stack: %p) -> ", offset, *regRSP);
175     log("[MEMWRITE(AFTER)] %p (hitcount: %d), mem: %p (sz: %d) (stack: %p) -> ", offset, g_accessMap[offset], addr, size, *regRSP);
176
177     LogData(addr, size);
178 }
```

```
219             if (offset == 0x1c5d || offset == 0x1d92) {
220                 UINT32 memOperands = INS_MemoryOperandCount(ins);
221                 // Iterate over each memory operand of the instruction
222                 for (UINT32 memOp = 0; memOp < memOperands; memOp++) {
223                     if (INS_OperandIsImplicit(ins, memOp)) {
224                         continue;
225                     }
226                     if (INS_MemoryOperandIsWritten(ins, memOp))
227                     {
228                         INS_InsertCall(
229                             ins, IPOINT_AFTER, (AFUNPTR)RecordMemWriteAfter_Profile,
230                             IARG_INST_PTR,
231                             IARG_MEMORYOP_EA, memOp,
232                             IARG_MEMORYWRITE_SIZE,
233                             IARG_REG_REFERENCE, REG_RSP,
234                             IARG_END);
235                     }
236                 }
237             }
```

Through profiling, we can find the memory addresses of variable *collision* (0x7fffffffd85c) and *isOver* (0x7fffffffd824).

```
28   [MEMWRITE(AFTER)] 0x1c5d (hitcount: 28), mem: 0x7fffffffd85c (sz: 4) (stack: 0x7fffffffd800) -> 0
29   [MEMWRITE(AFTER)] 0x1d92 (hitcount: 1), mem: 0x7fffffffd824 (sz: 4) (stack: 0x7fffffffd800) -> 1
```

Now, we modify the memories of *isOver* and *collision* via Pintool. The bird won't die when hitting the piles or the ground.

```
VOID RecordMemWriteAfter_Naive(VOID * ip, VOID * addr, UINT32 size, ADDRINT* regRSP)
{
    ADDRINT offset = (ADDRINT)ip - g_addrLow;

    // isOver
    if ((ADDRINT)0x7fffffffd824 == (ADDRINT)addr) {
            log("[MEMWRITE] isOver %p mem: %p (sz: %d) -> ",
        offset, addr, size);
        LogData(addr, size);

        // set to zero (force)
        memset(addr, 0, size);
    }

    // collision
    if ((ADDRINT)0x7fffffffd85c == (ADDRINT)addr) {
        log("[MEMWRITE] collision %p mem: %p (sz: %d) -> ",
        offset, addr, size);
        LogData(addr, size);

        // set to zero (force)
        memset(addr, 0, size);
    }
}
```

Method 1

At method 1, we keep changing the memory address of the critical variable via dynamically allocating memeories with random size.

```
28  #ifdef VER_METHOD1
29  int* g_pcollision = 0;
30  #define s_collision \
31          if (g_pcollision == 0) {\
32              g_pcollision = (int*)malloc(sizeof(int) * rand()%1000);\
33          } else {\
34              int* pnew = (int*)malloc(sizeof(int) * rand()%1000);\
35              *pnew = *g_pcollision;\
36              free(g_pcollision);\
37              g_pcollision = pnew;\
38          }\
39          *g_pcollision
40
41  #define r_collision *g_pcollision
42  #endif
```

Look into the dump file and trace the instruction at 0x1d6e.

```
1011        1d69:   e8 1b 0a 00 00          call   2789 <_Z16controlCollisioniiiii>
1012        1d6e:   89 03                   mov    %eax,(%rbx)
```

Profile the memory address via Pintool, we can see that the memory address changes every time it's being called.

```
1  [MEMWRITE(AFTER)] 0x1d6e (hitcount: 0), mem: 0x555555584f20 (sz: 4) (stack: 0x7fffffffd7e0) -> 0
2  [MEMWRITE(AFTER)] 0x1d6e (hitcount: 0), mem: 0x555555585200 (sz: 4) (stack: 0x7fffffffd7e0) -> 0
3  [MEMWRITE(AFTER)] 0x1d6e (hitcount: 0), mem: 0x555555585990 (sz: 4) (stack: 0x7fffffffd7e0) -> 0
4  [MEMWRITE(AFTER)] 0x1d6e (hitcount: 0), mem: 0x555555585e40 (sz: 4) (stack: 0x7fffffffd7e0) -> 0
5  [MEMWRITE(AFTER)] 0x1d6e (hitcount: 0), mem: 0x555555585ea0 (sz: 4) (stack: 0x7fffffffd7e0) -> 0
6  [MEMWRITE(AFTER)] 0x1d6e (hitcount: 0), mem: 0x5555555864a0 (sz: 4) (stack: 0x7fffffffd7e0) -> 0
```

Even if we try to modify the value in memory address 0x555555584f20, the hacking fails and the bird dies when hitting the ground/piles. It's because the critical variables keep changing their memory addresses via malloc and free.

```
174     // collision (Method 1)
175     if ((ADDRINT)0x555555584f20 == (ADDRINT)addr) {
176         log("[MEMWRITE] collision %p mem: %p (sz: %d) -> ",
177         offset, addr, size);
178         LogData(addr, size);
179
180         // set to zero (force)
181         memset(addr, 0, size);
182     }
```

The Pintool locates the correct memory address but loses track after.

```
1   [MEMWRITE] collision 0x1e9b mem: 0x555555584f20 (sz: 4) -> 0
2
```

Method 2

In method 2, we record the critical value to another global variable when the value is called/modified. The program can detect the hacking behavior by checking whether the value is the same between the local and global ones.

```
44  #ifdef VER_METHOD2
45  int g_s_collision;
46
47  #define s_collision collision=g_s_collision
48  #define r_collision get_collision(collision)
49  #define PRINT_SCOLLISION printf("%d\n", r_collision );
50
51  int __attribute__((always_inline)) inline get_collision(int v)
52  {
53      if( g_s_collision != v ) {
54          printf("corrupted!\n");
55          exit(-1);
56      }
57      return v;
58  }
59  #endif
```

Now, we profile again with Pintool and identify the instruction at 0x1ca9 where the value of *collision* is modified. In a real application, the modification of g_s_collision should happen somewhere else so that the hacker would not notice. The memory is located at 0x7fffffffd84c.

```
962      1c98:   e8 43 0a 00 00          call   26e0 <_Z16controlCollisioniiii>
963      1c9d:   89 05 89 33 00 00       mov    %eax,0x3389(%rip)        # 502c <g_s_collision>
964      1ca3:   8b 05 83 33 00 00       mov    0x3383(%rip),%eax        # 502c <g_s_collision>
965      1ca9:   89 85 0c ff ff ff       mov    %eax,-0xf4(%rbp)
```

```
1  [MEMWRITE(AFTER)] 0x1ca9 (hitcount: 0), mem: 0x7fffffffd84c (sz: 4) (stack: 0x7fffffffd7f0) -> 0
2  [MEMWRITE(AFTER)] 0x1ca9 (hitcount: 0), mem: 0x7fffffffd84c (sz: 4) (stack: 0x7fffffffd7f0) -> 0
3  [MEMWRITE(AFTER)] 0x1ca9 (hitcount: 0), mem: 0x7fffffffd84c (sz: 4) (stack: 0x7fffffffd7f0) -> 0
4  [MEMWRITE(AFTER)] 0x1ca9 (hitcount: 0), mem: 0x7fffffffd84c (sz: 4) (stack: 0x7fffffffd7f0) -> 0
```

Now, we launch the attack via Pintool.

```
185  VOID RecordMemWriteAfter_Naive2(VOID * ip, VOID * addr, UINT32 size, ADDRINT* regRSP)
186  {
187      ADDRINT offset = (ADDRINT)ip - g_addrLow;
188
189      // collision
190      if ((ADDRINT)0x7fffffffd84c == (ADDRINT)addr) {
191          log("[MEMWRITE] collision %p mem: %p (sz: %d) -> ",
192          offset, addr, size);
193          LogData(addr, size);
194
195          // set to zero (force)
196          memset(addr, 0, size);
197      }
198  }
```

Run the game, and we can find out that the program exits and report corrupted when hitting the piles.