

Cyber forensics additional assignment1

(1) high-level descriptions of how it works

To make the avatar keep moving after a crash, I need to modify the return value of `crash_check`, which originally returns 1 if a crash is detected and 0 otherwise, to always 0.

Because of Address space layout randomization, I could not simply use the absolute address of the return instruction to identify the ret instruction.

So I first find the call instruction to `crash_check` by using the difference between the target address and the address of the call instruction. Having found the call instruction, it will then be easy to figure out where `crash_check` starts and returns by using the difference between the addresses.

Having found where the ret instruction is, I can now insert a call before the ret instruction to change of value of `eax` to 0, making it return 0.

(2) instructions and memory locations (i.e., variables) you identified by analyzing the

I found with IDA and EDB that the difference between the call and `crash_check` itself was 0xf17 and the difference between the start and the end of `crash_check` was 0x107, so I was able to locate the call instruction and then the ret instruction.

(3) strategies to implement your Pin tool, and

I first search in the source code for anything related to `crash` and then I found the function `crash_check`. I tried to modify the code function so it would always return 0. After recompiling and running code game, the avatar did keep moving after a crash, proving this approach works.

Knowing which section of the code to modify, all that was left to do was find the ret instruction in pin tool and change the value of `rax`, the return value, to 0. As stated above, I calculated the relative address from the call instruction to the ret instruction and then added a callback before the ret instruction to change the register value to 0.

(4) explanations of your code (pin tool's code) -- per basic block.

In the main function, I register `Instruction` to inspect and add callbacks to instructions.

```
195 // ...
196 INS_AddInstrumentFunction(Instruction, 0);
197 // RTN_AddInstrumentFunction(Routine, 0);
198
```

The Instruction function is as follows:

```

134 VOID Instruction(INS ins, VOID *v) {
135     ADDRINT addr = INS_Address(ins);
136     string insString = INS_Disassemble(ins);
137     // fprintf(stderr, "addr:%lx, ins:%s\n", addr, insString.c_str());
138
139     // Locate the crash_check call by the difference between the address of
140     // the instruction and the target address
141     if (INS_IsDirectControlFlow(ins) && INS_IsCall(ins) &&
142         INS_DirectControlFlowTargetAddress(ins) - addr == 0xf17) {
143         // fprintf(stderr, "found addr:%lx, ins:%s\n", addr, insString.c_str());
144         retAddress = addr + 0xf17 + 0x107;
145     }
146
147     if (retAddress != 0 && addr == retAddress) {
148         // fprintf(stderr, "found ret:%lx, ins:%s\n", addr, insString.c_str());
149         INS_InsertCall(ins, IPOINT::IPOINT_BEFORE, (AFUNPTR)setReg,
150             IARG_REG_REFERENCE, REG_RAX, IARG_UINT64, 0x0, IARG_END);
151     }
152
153     if (insString == "mov eax, dword ptr [rip+0x26300]") {
154         // fprintf(stderr, "found stakes:%lx, ins:%s\n", addr,
155             // insString.c_str());
156         INS_InsertCall(ins, IPOINT::IPOINT_AFTER, (AFUNPTR)setReg,
157             IARG_REG_REFERENCE, REG_RAX, IARG_UINT64, 5000,
158             IARG_END);
159     }
160     if (insString == "mov eax, dword ptr [rip+0x262e4]") {
161         // fprintf(stderr, "found stakes:%lx, ins:%s\n", addr,
162             // insString.c_str());
163         INS_InsertCall(ins, IPOINT::IPOINT_AFTER, (AFUNPTR)setReg,
164             IARG_REG_REFERENCE, REG_RAX, IARG_UINT64, 5000,
165             IARG_END);
166     }
167 }
168

```

On lines 141 and 142, I use the difference between the address of the target and the call instruction to determine if this is the call to the `crash_check` function. If it is, I calculate the ret address by adding 0xf17+0x107 to the current address and save it to a variable for later reference.

On line 146, I check if `retAddress` has been set. If yes, I check if it is equal to the address of the current instruction. If it is, we have found the ret instruction. Therefore, I add a callback function to the instruction with a reference to RAX and 0 as arguments.

Below is the setReg callback function.

```

72 VOID setReg(ADDRINT *regRef, UINT64 val) {
73     // return;
74     // fprintf(stderr, "original value: %ld\n", *regRef);
75     // fprintf(stderr, "change reg to %ld\n", val);
76     *regRef = val;
77 }

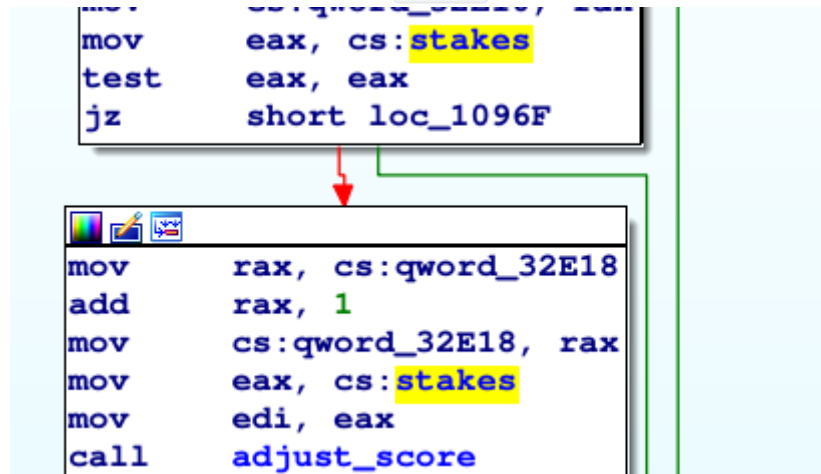
```

In this function I simply change the value of the register reference that has been passed in.

Extra Challenge

In game.c I found a function named `adjust_score` which as its names suggests, changes the score. Because it takes the value to add to the current score as a parameter, as long as we can change the value of the argument when `adjust_score` is called, we can make the score higher.

Looking in the function `scroll_handler`, we can see stakes both controls whether `adjust_score` will be called and how much the score will be adjusted. In IDA, we can see before both uses of `stakes`, the value will be loaded into `eax`.



So all I needed to do was modify the value of `eax` after the `mov` instructions.

With `ida` and `pintool`, I found the instruction in question were `mov eax, dword ptr [rip+0x26300]` and `mov eax, dword ptr [rip+0x262e4]`, so I inserted a callback function after each of them to change the value of the register.

```
if (insString == "mov eax, dword ptr [rip+0x26300]") {
    // fprintf(stderr, "found stakes:%lx, ins:%s\n", addr,
    // insString.c_str());
    INS_InsertCall(ins, IPOINT::IPOINT_AFTER, (AFUNPTR)setReg,
        IARG_REG_REFERENCE, REG_RAX, IARG_UINT64, 5000,
        IARG_END);
}

if (insString == "mov eax, dword ptr [rip+0x262e4]") {
    // fprintf(stderr, "found stakes:%lx, ins:%s\n", addr,
    // insString.c_str());
    INS_InsertCall(ins, IPOINT::IPOINT_AFTER, (AFUNPTR)setReg,
        IARG_REG_REFERENCE, REG_RAX, IARG_UINT64, 5000,
        IARG_END);
}
```