William Hupp and Ricardo Romero

**IDA Problem:** One of the problems encountered by IDA is when a malware writer intentionally interferes with the disassembly processes by introducing instructions to interfere with the offset of instruction addresses. In this project we created a script which handles two of such cases, a “jz” followed by a “jnz” and a unconditionally jump cause by an xor with the same two registers. Our environment for testing the code was a Windows 10 VM using Virtual Box and a demo version of IDA acquired from HexRays.

**Solution:** To counteract the problem of improper disassembly caused by the previously mentioned techniques, our group decided to write a script that would analyze the instructions of the program and print to the output screen if any of the mentioned cases were found. We used the IDC language, a scripting language built for IDA with syntactic similarities to C, and the functions found within the IDC API. To begin we created a for loop that would iterate through each instruction within the program that IDA disassembled. For every instruction we parse the instruction and check for the following: if the instruction if a “jz” followed by a “jnz” and viceversa, or if the instruction is an “xor” instruction with the same two registers as arguments followed by a “jz” or “jnz”. If either case is found we print the current instruction and its address to standard output and continue to the next instruction. The functions used from the IDC API are as follows: NextFunction(), next_addr(), substr(), GetDisasm(), Message(), get_operand_type(), and get_operand_value().

**Results**: We tested our code against the labs from the book that had this kind of problem. We were successful in finding the areas of code that required the user to change to instruction by use of no-op or changing from text to data.

```
#include <idc.idc>
static main() {
    auto addr;
    addr = 0;
    for (addr = NextFunction(addr); addr != BADADDR; addr = next_addr(addr)) {
        //jz is followed by jnz
        if((substr(GetDisasm(addr), 0, 2) == "jz") && (substr(GetDisasm(next_addr(addr)), 0, 3) ==
"jnz")){
            Message("Address: %x ... instruction:%s \n", addr-1, GetDisasm(addr));
        }
        //jnz is followed by jz
        if((substr(GetDisasm(addr), 0, 3) == "jnz" && substr(GetDisasm(next_addr(addr)), 0, 2) == "jz")){
            Message("Address: %x ... instruction:%s \n", addr-1, GetDisasm(addr));
        }
        //xor with same registers is followed by conditional jump
        if(substr(GetDisasm(addr), 0, 3) == "xor" && (substr(GetDisasm(next_addr(addr)), 0, 2) == "jz"
        || substr(GetDisasm(next_addr(addr)), 0, 3) =="jnz")){
            auto type = get_operand_type(addr-1, 0);
            if (type == 1) { //Is this a register indirect operand?
                if (get_operand_value(addr-1, 0) == get_operand_value(addr-1, 1)){
                    Message("Address: %x ... instruction:%s \n", addr-1,GetDisasm(addr));
```

```
                }
            }
        }
    }
}
```