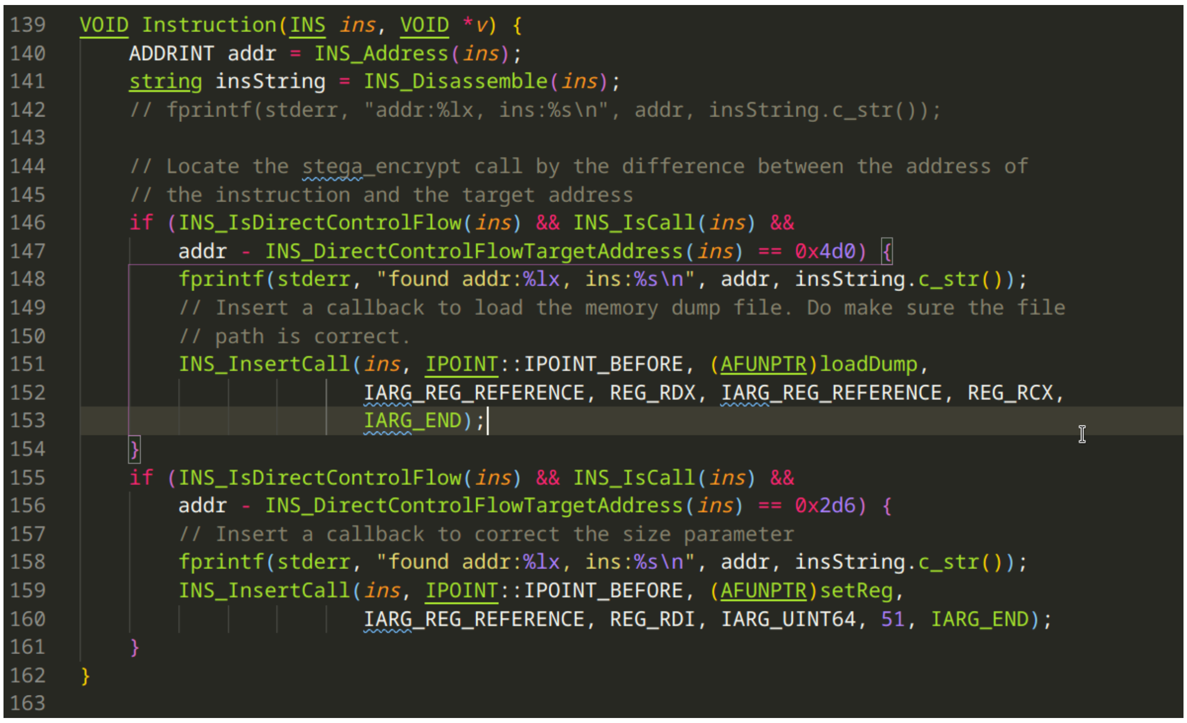
Cyber Forensics HW2

tags: `class`

How my pintool works

![(https://i.imgur.com/Ei5n6gm.png)



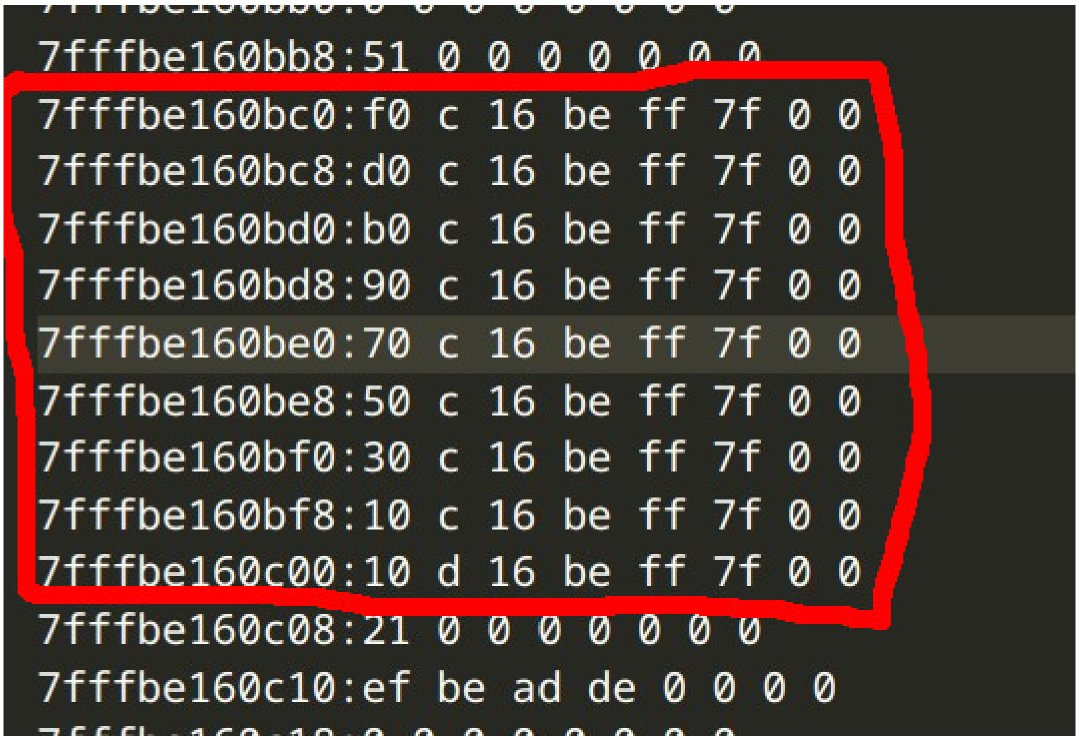
```
VOID Instruction(INS ins, VOID *v) {
    ADDRINT addr = INS_Address(ins);
    string insString = INS_Disassemble(ins);
    // fprintf(stderr, "addr:%lx, ins:%s\n", addr, insString.c_str());

    // Locate the stega_encrypt call by the difference between the address of
    // the instruction and the target address
    if (INS_IsDirectControlFlow(ins) && INS_IsCall(ins) &&
        addr - INS_DirectControlFlowTargetAddress(ins) == 0x4d0) {
        fprintf(stderr, "found addr:%lx, ins:%s\n", addr, insString.c_str());
        // Insert a callback to load the memory dump file. Do make sure the file
        // path is correct.
        INS_InsertCall(ins, IPOINT::IPOINT_BEFORE, (AFUNPTR)loadDump,
                       IARG_REG_REFERENCE, REG_RDX, IARG_REG_REFERENCE, REG_RCX,
                       IARG_END);
    }
    if (INS_IsDirectControlFlow(ins) && INS_IsCall(ins) &&
        addr - INS_DirectControlFlowTargetAddress(ins) == 0x2d6) {
        // Insert a callback to correct the size parameter
        fprintf(stderr, "found addr:%lx, ins:%s\n", addr, insString.c_str());
        INS_InsertCall(ins, IPOINT::IPOINT_BEFORE, (AFUNPTR)setReg,
                       IARG_REG_REFERENCE, REG_RDI, IARG_UINT64, 51, IARG_END);
    }
}
```

My pintool first locates the instruction that calls `stega_encrypt` and inserts a callback that overwrites register values. (the pointer and the size). To get the correct memory content, I also had to fix the pointers in the buffer containing the memory dump file. Then it looks for the `size_encrypt` function call and changes the argument value to the correct size.

How I injected the contents from memory buffer and find the corret offset that points to the buffer that holds instances of the BUFENC structure

The first thing I thought of was, because the structs are allocated with `malloc`, they should all be located in the heap. To find out which memory dump file was the heap, I printed out the values in the memory dump files went through each of them, and found that `2824-7fffbe15c000-7fffbe17d000` contains multiple addresses storing 0xdeadbeef(or EF BE AD DE in little endian), indicating this section was where the heap was.

Having found the heap, I needed to find the correct offset of the first BUFENC struct in the heap section.

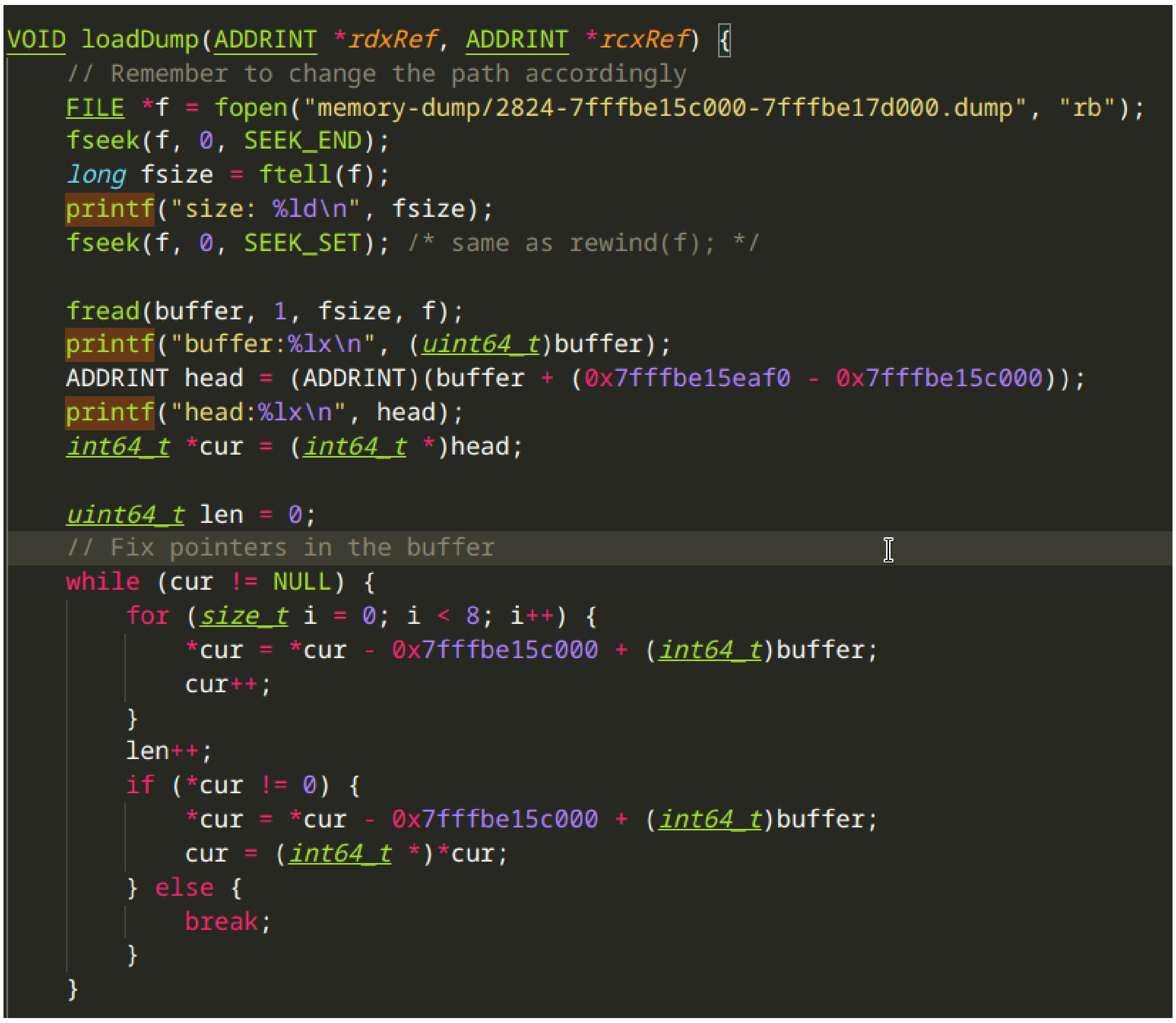


```
7fffbe160bb8:51 0 0 0 0 0 0 0
7fffbe160bc0:f0 c 16 be ff 7f 0 0
7fffbe160bc8:d0 c 16 be ff 7f 0 0
7fffbe160bd0:b0 c 16 be ff 7f 0 0
7fffbe160bd8:90 c 16 be ff 7f 0 0
7fffbe160be0:70 c 16 be ff 7f 0 0
7fffbe160be8:50 c 16 be ff 7f 0 0
7fffbe160bf0:30 c 16 be ff 7f 0 0
7fffbe160bf8:10 c 16 be ff 7f 0 0
7fffbe160c00:10 d 16 be ff 7f 0 0
7fffbe160c08:21 0 0 0 0 0 0 0
7fffbe160c10:ef be ad de 0 0 0 0
```

Notice the pattern in the values at there addresses. Since each `BUFENC` contains 9 pointers, I figured the addresses in the picture were probably where a `BUFENC` struct was located. Having found 1 node, all that was left to do was trace back and find the first node in the linked list, which turned out to be at 7fffbe15eaf0.

Why the program crashes after you inject the memory and how I fixed it

The reason the program would crash was that there are pointers in the memory dump(in the linked list), and those pointers weren't pointing to the correct addresses because the base address will change every time we allocate a buffer to store the memory content. So I needed to fix the pointers for the injection to work.



```
VOID loadDump(ADDRINT *rdxRef, ADDRINT *rcxRef) {
    // Remember to change the path accordingly
    FILE *f = fopen("memory-dump/2824-7fffbe15c000-7fffbe17d000.dump", "rb");
    fseek(f, 0, SEEK_END);
    long fsize = ftell(f);
    printf("size: %ld\n", fsize);
    fseek(f, 0, SEEK_SET); /* same as rewind(f); */

    fread(buffer, 1, fsize, f);
    printf("buffer:%lx\n", (uint64_t)buffer);
    ADDRINT head = (ADDRINT)(buffer + (0x7fffbe15eaf0 - 0x7fffbe15c000));
    printf("head:%lx\n", head);
    int64_t *cur = (int64_t *)head;

    uint64_t len = 0;
    // Fix pointers in the buffer
    while (cur != NULL) {
        for (size_t i = 0; i < 8; i++) {
            *cur = *cur - 0x7fffbe15c000 + (int64_t)buffer;
            cur++;
        }
        len++;
        if (*cur != 0) {
            *cur = *cur - 0x7fffbe15c000 + (int64_t)buffer;
            cur = (int64_t *)*cur;
        } else {
            break;
        }
    }
```

The fixing of the pointers is done in the while loop. For each `BUFENC` struct, which has 9 pointers, I fix all the nine pointers and then proceed to fix the next struct on the linked list by going to the address in the last pointer in the current struct.

what is the secret message and how did you recover?

The secret message is

> Everything's amazing right now, and nobody's happy.

I recovered the message by calling `ptr2str`. I passed the pointer to the head of the linked list as an argument to the function and got back the original secret message. (code in loadMemory.cpp)