I Lost My Password

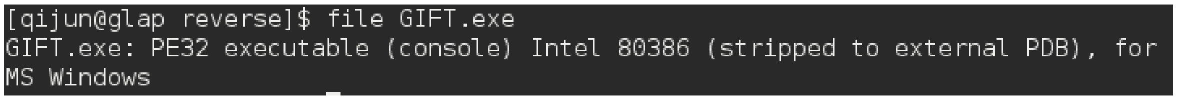
Problem: Can you find the flag in this file?

Given: GIFT.exe

Hint: Disassemblers are great

Introduction: I Lost My Password is an easy level challenge designed to teach students to perform basic reverse engineering analysis against a target binary. In this specific case we'll be reversing the binary to find the password needed to retrieve the flag.

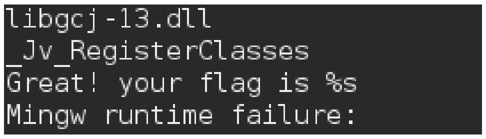
Steps:
(1) Executing the 'file' command gives us the following information:



```
[qijun@glap reverse]$ file GIFT.exe
GIFT.exe: PE32 executable (console) Intel 80386 (stripped to external PDB), for
MS Windows
```

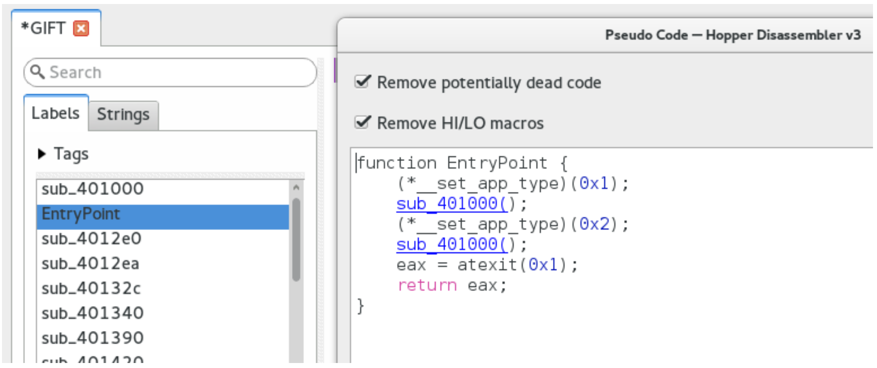
The binary we'll be reversing today is a PE32 executable based on the Intel 80386 architecture for MS Windows, or in layman's terms an EXE which will run on Windows. Which means we'll need access to a Windows machine if we want to run the binary.

(2) Executing the 'strings' command gives us the following information. The most important piece of info is highlighted: "Great! Your flag is %s". Which means that string is somewhere in the program.



```
libgcj-13.dll
_Jv_RegisterClasses
Great! your flag is %s
Mingw runtime failure:
```

(3) Then, we load the program into Hopper. It points to the "EntryPoint", the starting of the program. We then open the decompiler of Hopper and get the pseudo code of "EntryPoint". Inside, the program executes sub_40100, a sub routine.

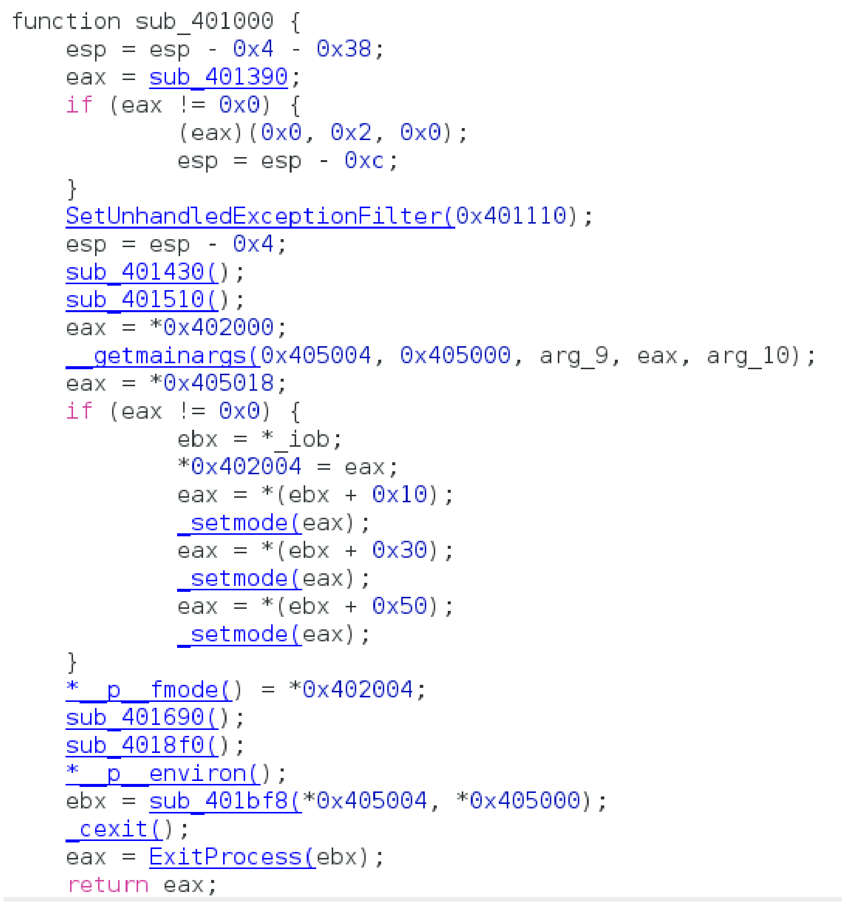


✓ Remove potentially dead code

✓ Remove HI/LO macros

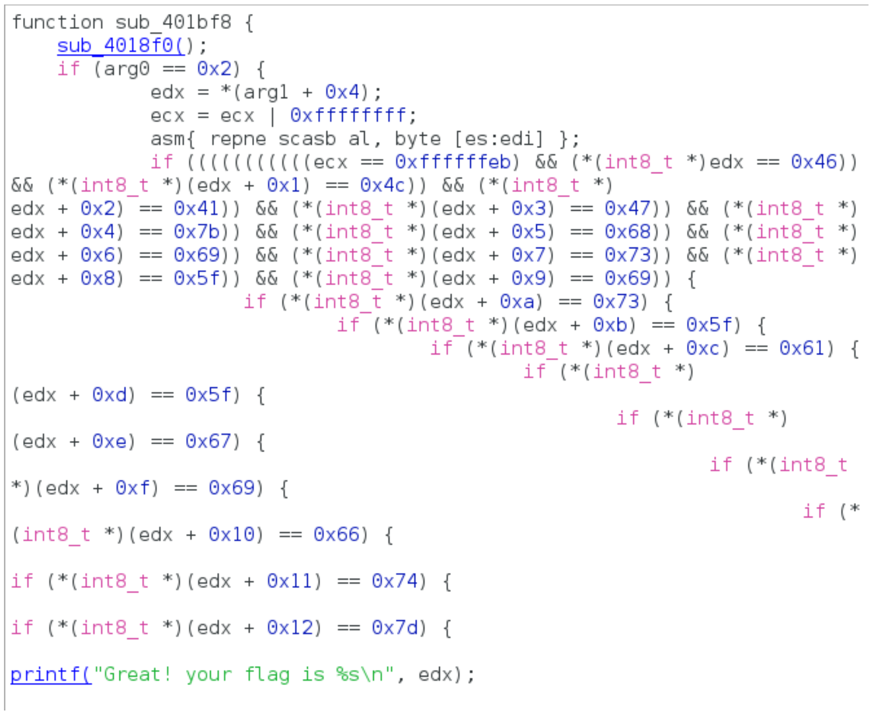
```
function EntryPoint {
    (*__set_app_type)(0x1);
    sub_401000();
    (*__set_app_type)(0x2);
    sub_401000();
    eax = atexit(0x1);
    return eax;
}
```

(4) We double click on the underlined "sub_40100", and Hopper brings us to the pseudo code of sub_40100. There are a few calls to other sub routines. But, "sub_401bf8" catches our eyes, because all other sub routines appear to be normal system or library routines in Windows. For example, __getmainargs is to get arguments. So, we inspect sub_401bf8 first by double clicking it.



```
function sub_401000 {
    esp = esp - 0x4 - 0x38;
    eax = sub_401390;
    if (eax != 0x0) {
            (eax)(0x0, 0x2, 0x0);
            esp = esp - 0xc;
    }
    SetUnhandledExceptionFilter(0x401110);
    esp = esp - 0x4;
    sub_401430();
    sub_401510();
    eax = *0x402000;
    __getmainargs(0x405004, 0x405000, arg_9, eax, arg_10);
    eax = *0x405018;
    if (eax != 0x0) {
            ebx = *_iob;
            *0x402004 = eax;
            eax = *(ebx + 0x10);
            _setmode(eax);
            eax = *(ebx + 0x30);
            _setmode(eax);
            eax = *(ebx + 0x50);
            _setmode(eax);
    }
    *__p__fmode() = *0x402004;
    sub_401690();
    sub_4018f0();
    *__p__environ();
    ebx = sub_401bf8(*0x405004, *0x405000);
    _cexit();
    eax = ExitProcess(ebx);
    return eax;
```

(5) We see the pseudo code of sub_401bf8 has a if statement where the condition is a sequence of byte comparisons. If all bytes comparisons pass, then it will print "Great! your flag is ...". OK, so we really need to figure out what these bytes are. Further more, we notice that the flag string is located at the address pointed by the register edx. Meanwhile, the comparisons are to compare bytes with the flag string pointed by edx too. So, the bytes in the comparisons are the bytes of the flag string.



```
function sub_401bf8 {
    sub_4018f0();
    if (arg0 == 0x2) {
            edx = *(arg1 + 0x4);
            ecx = ecx | 0xffffffff;
            asm{ repne scasb al, byte [es:edi] };
            if ((((((((((ecx == 0xffffffeb) && (*(int8_t *)edx == 0x46))
&& (*(int8_t *)(edx + 0x1) == 0x4c)) && (*(int8_t *)
edx + 0x2) == 0x41)) && (*(int8_t *)(edx + 0x3) == 0x47)) && (*(int8_t *)
edx + 0x4) == 0x7b)) && (*(int8_t *)(edx + 0x5) == 0x68)) && (*(int8_t *)
edx + 0x6) == 0x69)) && (*(int8_t *)(edx + 0x7) == 0x73)) && (*(int8_t *)
edx + 0x8) == 0x5f)) && (*(int8_t *)(edx + 0x9) == 0x69)) {
                    if (*(int8_t *)(edx + 0xa) == 0x73) {
                            if (*(int8_t *)(edx + 0xb) == 0x5f) {
                                    if (*(int8_t *)(edx + 0xc) == 0x61) {
                                            if (*(int8_t *)
(edx + 0xd) == 0x5f) {
                                                    if (*(int8_t *)
(edx + 0xe) == 0x67) {
                                                            if (*(int8_t
*)(edx + 0xf) == 0x69) {
                                                                    if (*
(int8_t *)(edx + 0x10) == 0x66) {

if (*(int8_t *)(edx + 0x11) == 0x74) {

if (*(int8_t *)(edx + 0x12) == 0x7d) {

printf("Great! your flag is %s\n", edx);
```

Let's write down the bytes to get the flag...