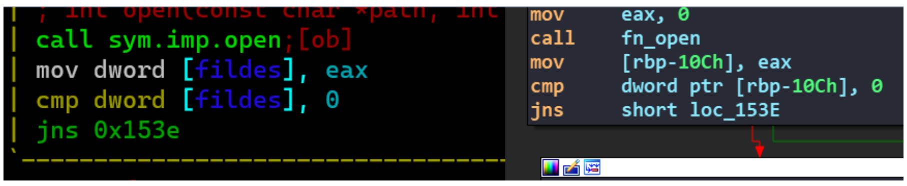
R09921A10 鄭翔予

CTF Account: stanleymusic

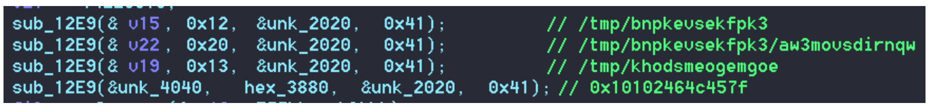
1. fifo
   - Solution
     - To find out the address of the entry point while debugging with gdb, first, type "starti" to stop the program at the beginning and then type "info file" to find out the entry point address
     - Use IDA to decompile the main function. While most of the functions are unclarified, I use radare2 to figure out the exact function name and modify the result in IDA



```
| call sym.imp.open;[ob]
| mov dword [fildes], eax
| cmp dword [fildes], 0
| jns 0x153e
```

```
mov     eax, 0
call    fn_open
mov     [rbp-10Ch], eax
cmp     dword ptr [rbp-10Ch], 0
jns     short loc_153E
```

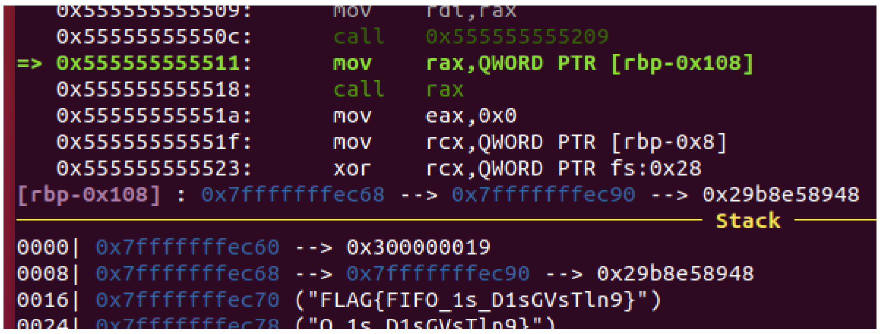
     - Function at offset 0x12E9
       - This function is modifying values by some arithmetic operations stored in the first parameter with other parameters and then storing the result back to the first parameter
       - Four parameters
         - The first parameter is the address of the input and output array
         - The second parameter is a constant which specify the length of the array which needs to be modified
         - The third parameter is the address of a constant which is used for arithmetic operations
         - The fourth parameter is a constant which is used for arithmetic operations



```
sub_12E9(& v15 , 0x12, &unk_2020, 0x41);          // /tmp/bnpkevsekfpk3
sub_12E9(& v22 , 0x20, &unk_2020, 0x41);          // /tmp/bnpkevsekfpk3/aw3movsdirnqw
sub_12E9(& v19 , 0x13, &unk_2020, 0x41);          // /tmp/khodsmeogemgoe
sub_12E9(&unk_4040,  hex_3880, &unk_2020,  0x41); // 0x10102464c457f
```

     - It then creates a file "/tmp/khodsmeogemgoe" and writes the value stored in offset 0x4040, which is modified by sub_12E9, with size 0x3880 into it. Finally, fork a child process and execute "/tmp/khodsmeogemgoe"
       - To trace into the new child process use gdb with the command "set follow-fork-mode child"
     - The parent process also creates a directory "/tmp/bnpkevsekfpk3" and a file "/tmp/bnpkevsekfpk3/aw3movsdirnqw" and then write the value stored in offset 0x2020 into it

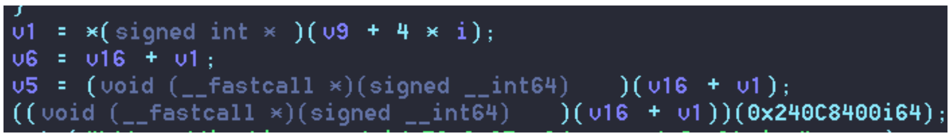
        - The child process then open the file "/tmp/bnpkevsekfpk3/aw3movsdirnqw" and read all the values in it
        - Inside the child process, there's a function at offset 0x1209(relative to child process image base) which is exactly the same function as sub_12E9 in the parent process. The third parameter points to the value which is read from "/tmp/bnpkevsekfpk3/aw3movsdirnqw"
        - After executing this function, the flag appears in the first parameter's address



```
   0x55555555550c:      call   0x555555555209
=> 0x555555555511:      mov    rax,QWORD PTR [rbp-0x108]
   0x555555555518:      call   rax
   0x55555555551a:      mov    eax,0x0
   0x55555555551f:      mov    rcx,QWORD PTR [rbp-0x8]
   0x555555555523:      xor    rcx,QWORD PTR fs:0x28
[rbp-0x108] : 0x7fffffffec68 --> 0x7fffffffec90 --> 0x29b8e58948
--------------------------------------------------------- Stack -----
0000| 0x7fffffffec60 --> 0x300000019
0008| 0x7fffffffec68 --> 0x7fffffffec90 --> 0x29b8e58948
0016| 0x7fffffffec70 ("FLAG{FIFO_1s_D1sGVsTln9}")
```

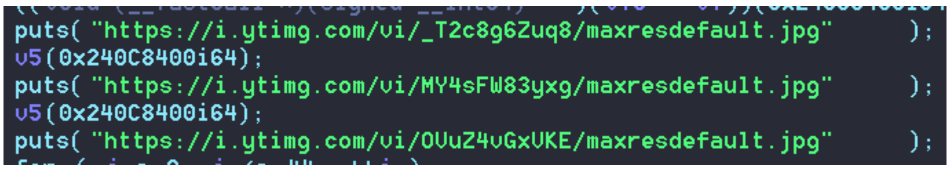
    - Flag
        - FLAG{FIFO_1s_D1sGVsTln9}
    - Reference
        - https://stackoverflow.com/questions/28789458/gdb-debugging-process-after-exec-call
        - https://stackoverflow.com/questions/9885545/how-to-find-the-main-functions-entry-point-of-elf-executable-file-without-any-s/9893169

2. giveUflag
    - Solution
        - Start from the entry point(offset 0x4E0) and trace through the functions to find out the main suspicious function
            - start -> sub_401180 -> sub_401870 -> sub_40184C -> sub_4015F3(main suspicious function)
        - The function pointer is assigned dynamically, which means IDA can’t figure which function is going to be executed



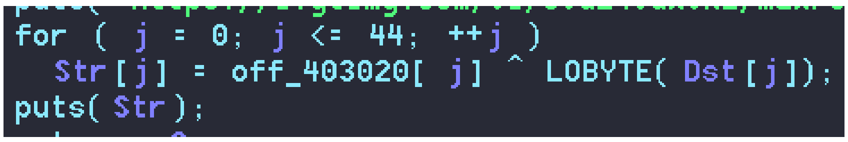
```
v1 = *(signed int * )(v9 + 4 * i);
v6 = v16 + v1;
v5 = (void (__fastcall *)(signed __int64)   )(v16 + v1);
((void (__fastcall *)(signed __int64)   )(v16 + v1))(0x240C8400i64);
```

        - These URLs have nothing to do with the challenge



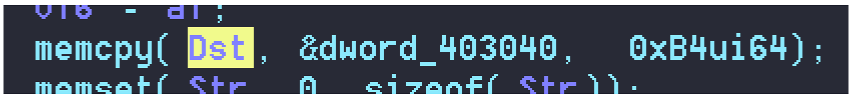
```
puts( "https://i.ytimg.com/vi/_T2c8g6Zuq8/maxresdefault.jpg"     );
v5(0x240C8400i64);
puts( "https://i.ytimg.com/vi/MY4sFW83yxg/maxresdefault.jpg"     );
v5(0x240C8400i64);
puts( "https://i.ytimg.com/vi/OVuZ4vGxUKE/maxresdefault.jpg"     );
```

- The last part of this function is interesting since it's doing xor decryption and output the result



```
for ( j = 0; j <= 44; ++j )
  Str[j] = off_403020[ j] ^ LOBYTE( Dst[j]);
puts( Str );
```

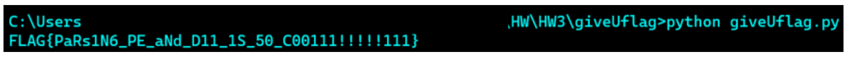
- The result can be known using the debugger and set the breakpoint here. In addition, it can also be manually calculated since Dst and off_403020 are fixed values stored in the binary file
  - off_403020 = YOU_USE_HAIYA_WHEn_YOU'RE_DISAPPOINTED_MMSSGG
  - Dst is copied from dword_403040



```
memcpy( Dst, &dword_403040, 0xB4ui64);
```

[31, 3, 20, 24, 46, 3, 36, 13, 59, 112, 7, 111, 30, 15, 18, 23, 36, 32, 59, 6, 11, 100, 22, 13, 116, 12, 27, 124, 99, 30, 19, 96, 127, 120, 127, 101, 100, 101, 126, 108, 108, 98, 98, 118, 58]

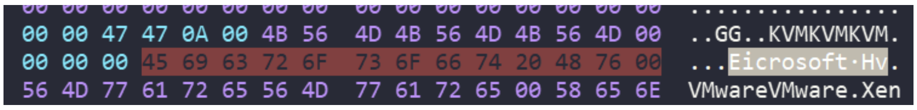
- Write a simple python script to get the flag



```
C:\Users[redacted]HW\HW3\giveUflag>python giveUflag.py
FLAG{PaRs1N6_PE_aNd_D11_1S_50_C00111!!!!!111}
```

- Flag
  - FLAG{PaRs1N6_PE_aNd_D11_1S_50_C00111!!!!!111}

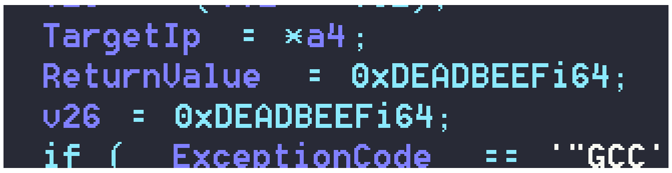
3. nani
- Solution
  - Use "Detect It Easy" to analyze the file and find out it's using UPX(3.96) packer. Thus, I download UPX(3.96) and unpack it with the command "upx -d nani.exe"
  - Inside function 0x4019a3, it's using "IsDebuggerPresent()" for anti-debugging. Thus, I use x64dbg with the plugin ScyllaHide to bypass the limitation
  - Inside function 0x401869, it's comparing the result of "cpuid" with several constant strings for anti-vm. Thus, I patch the constant string to avoid successful comparison. In my case, the result of "cpuid" is "Microsoft Hv" and I patch it to "Eicrosoft Hv"



```
00 00 47 47 0A 00 4B 56  4D 4B 56 4D 4B 56 4D 00  ..GG..KVMKVMKVM.
00 00 00 45 69 63 72 6F  73 6F 66 74 20 48 76 00  ...Eicrosoft·Hv.
56 4D 77 61 72 65 56 4D  77 61 72 65 00 58 65 6E  VMwareVMware.Xen
```

  - Inside function 0x40c9d0, there's a "RaiseException" function call. However, it won't call any exception handling function and just end the program

- There are two exception handling functions that I found using IDA pro, 0x4016FB and 0x4A0F10 respectively. Among them, the latter one is suspicious since its subfunction is using constant 0xdeadbeef which is obviously written by the programmer rather than the system



```
TargetIp = *a4;
ReturnValue = 0xDEADBEEFi64;
v26 = 0xDEADBEEFi64;
if ( ExceptionCode == '"GCC'
```

- Since there's no normal way to direct control flow to this function, I set a breakpoint on "RaiseException" and modify the value of rip to 0x4A0F10
- After jumping to 0x4A0F10, it then executes to 0x4015AF and after the xor decryption the flag will show up in the memory

- Flag
  - FLAG{r3v3rs3_Ma5T3R}