LDP

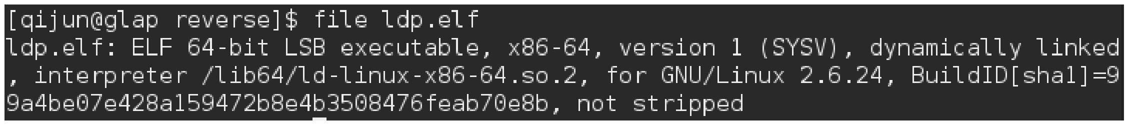
Problem: Can you get to the flag in this file?

Given: ldf.elf

Hint: GLIBC is weak

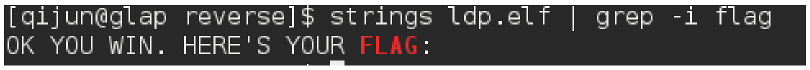
Introduction: LD_PRELOAD is a medium level reversing challenge designed to improve students reverse engineering skills by teaching what's commonly referred to as the "LD_Preload" trick. Which will allow you to heavily influence the way a program executes.

Steps:
(1) This writeup will be solved using GDB Debugger and Hopper disassembler, alongside the LD_Preload trick but you're free to use whatever tools you're comfortable with. Let's begin by performing a basic analysis against the binary using the 'file' and 'strings' command.



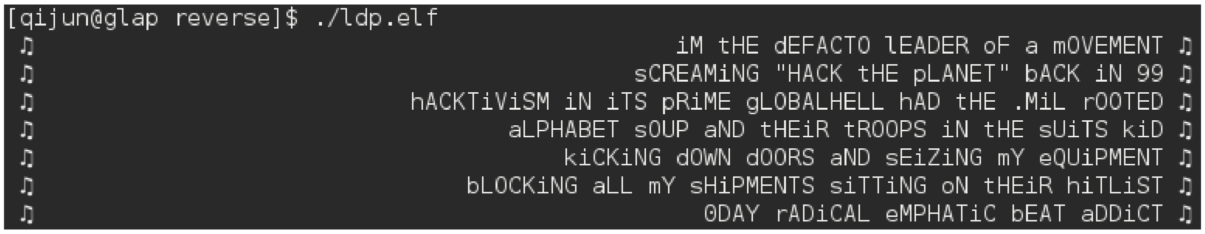
```
[qijun@glap reverse]$ file ldp.elf
ldp.elf: ELF 64-bit LSB executable, x86-64, version 1 (SYSV), dynamically linked
, interpreter /lib64/ld-linux-x86-64.so.2, for GNU/Linux 2.6.24, BuildID[sha1]=9
9a4be07e428a159472b8e4b3508476feab70e8b, not stripped
```

Nothing out of the ordinary here. I've omitted the output from strings because it was largely uninteresting other than a string that said "OK YOU WIN HERE'S YOUR FLAG!" with no output after that.



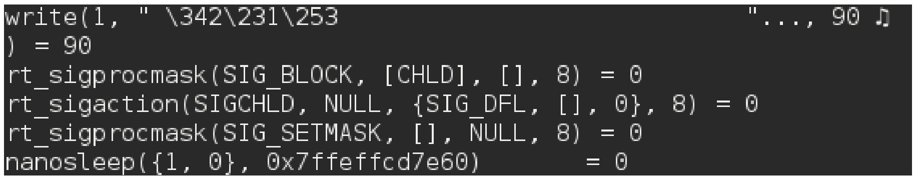
```
[qijun@glap reverse]$ strings ldp.elf | grep -i flag
OK YOU WIN. HERE'S YOUR FLAG:
```

(2) Let's move on to executing the program to see what happens, remember to chmod the binary if need be. All this program seems to do is output some song lyrics, if you wait long enough the program loops and continues repeating the lyrics.

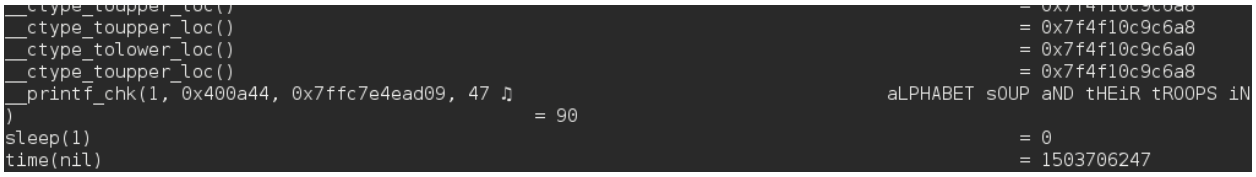


```
[qijun@glap reverse]$ ./ldp.elf
 ♫                                         iM tHE dEFACTO lEADER oF a mOVEMENT ♫
 ♫                                      sCREAMiNG "HACK tHE pLANET" bACK iN 99 ♫
 ♫                         hACKTiViSM iN iTS pRiME gLOBALHELL hAD tHE .MiL rOOTED ♫
 ♫                                  aLPHABET sOUP aND tHEiR tROOPS iN tHE sUiTS kiD ♫
 ♫                                      kiCKiNG dOWN dOORS aND sEiZiNG mY eQUiPMENT ♫
 ♫                             bLOCKiNG aLL mY sHiPMENTS siTTiNG oN tHEiR hiTLiST ♫
 ♫                                                0DAY rADiCAL eMPHATiC bEAT aDDiCT ♫
```

(3) Let's take a look at the output from both strace and ltrace and see what calls and libraries this program is using. Maybe that will give us a better idea of what's happening here. The output from strace is first, ltrace second. Both screenshots are trimmed due to repeated behavior between lyrics.



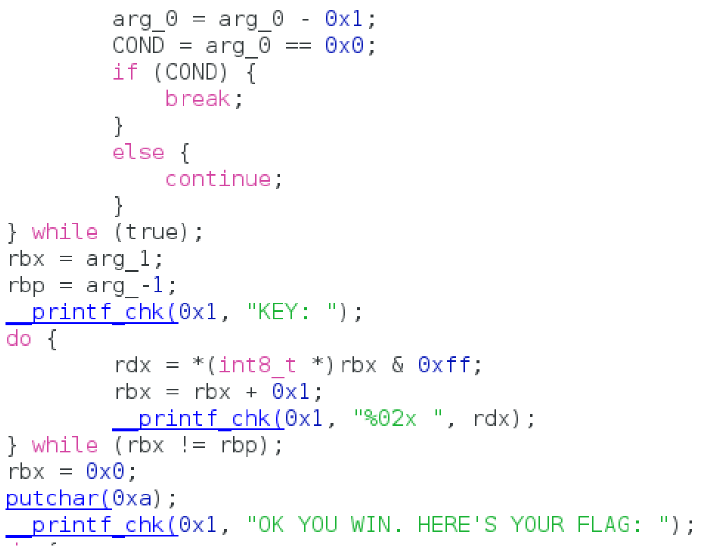
```
write(1, " \342\231\253                                  "..., 90 ♫
) = 90
rt_sigprocmask(SIG_BLOCK, [CHLD], [], 8) = 0
rt_sigaction(SIGCHLD, NULL, {SIG_DFL, [], 0}, 8) = 0
rt_sigprocmask(SIG_SETMASK, [], NULL, 8) = 0
nanosleep({1, 0}, 0x7ffeffcd7e60)        = 0
```



```
__ctype_toupper_loc()                                                     = 0x7f4f10c9c6a8
__ctype_toupper_loc()                                                     = 0x7f4f10c9c6a8
__ctype_tolower_loc()                                                     = 0x7f4f10c9c6a0
__ctype_toupper_loc()                                                     = 0x7f4f10c9c6a8
__printf_chk(1, 0x400a44, 0x7ffc7e4ead09, 47 ♫                     aLPHABET sOUP aND tHEiR tROOPS iN
)                                        = 90
sleep(1)                                                                  = 0
time(nil)                                                                 = 1503706247
```

We observe that, between each lyric output, the program makes a call to the c functions sleep() and time(). Calls to these functions are responsible for the pause between each lyric output. Now that we have a better picture of how this program executes let's open the program in the disassembler of your choice and see if we don't notice any glaring items.

(4) We load the program into Hopper and decompile its main function. In the pseudo code, we see a big while loop, after which the flag may be printed "OK YOU WIN. HERE'S YOUR FLAG:". In the while loop, we see a counter arg_0. Each iteration decrements the counter. When the counter reaches 0, the loop breaks and the flag is printed.



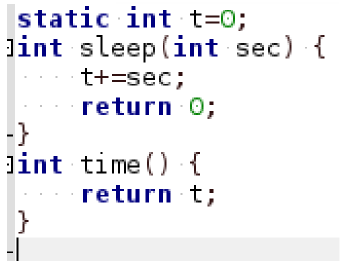
```
        arg_0 = arg_0 - 0x1;
        COND = arg_0 == 0x0;
        if (COND) {
            break;
        }
        else {
            continue;
        }
} while (true);
rbx = arg_1;
rbp = arg_-1;
__printf_chk(0x1, "KEY: ");
do {
        rdx = *(int8_t *)rbx & 0xff;
        rbx = rbx + 0x1;
        __printf_chk(0x1, "%02x ", rdx);
} while (rbx != rbp);
rbx = 0x0;
putchar(0xa);
__printf_chk(0x1, "OK YOU WIN. HERE'S YOUR FLAG: ");
```

(5) After getting around in the binary, we cannot figure out the initial value of arg_0. It also appears the program just prints lyrics for very long time. So, let's try to fake the sleep call, tricking the program into thinking it has actually slept for the required number of seconds. We see the program calls the time function too. If we only fake the sleep call, the time function will not actually catch the elapsed time. So, we fake the time call too.

We can do that by overriding GLIBC's sleep() and time() functions very easily: create a C file that contains the functions you want to override, and make them have the same signature. Just increment the time with one second on every sleep() call, without invoking the underlying sleep() function. Then compile it into a shared object and preload it using the LD_PRELOAD trick. When you'd run the binary, the sleep function will execute our implementation, while it still increases the elapsed time.

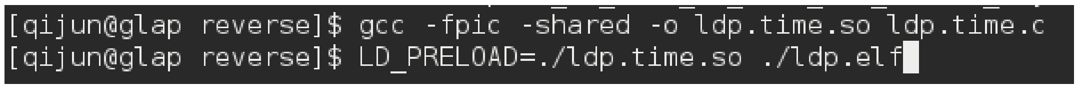
(6) We write our own C code for the sleep and time functions. We can see the sleep function increase the time variable t with the requested sleep seconds and immediately returns. Then,

the time function returns the time variable t. So, when the program runs with the faked sleep and time function, it will not perceive anything abnormal.



```
static int t=0;
int sleep(int sec) {
    t+=sec;
    return 0;
}
int time() {
    return t;
}
```

(7) Now all that’s left to do is compile this piece of code into a shared object’ and execute the binary with LD_Preload.



```
[qijun@glap reverse]$ gcc -fpic -shared -o ldp.time.so ldp.time.c
[qijun@glap reverse]$ LD_PRELOAD=./ldp.time.so ./ldp.elf
```