

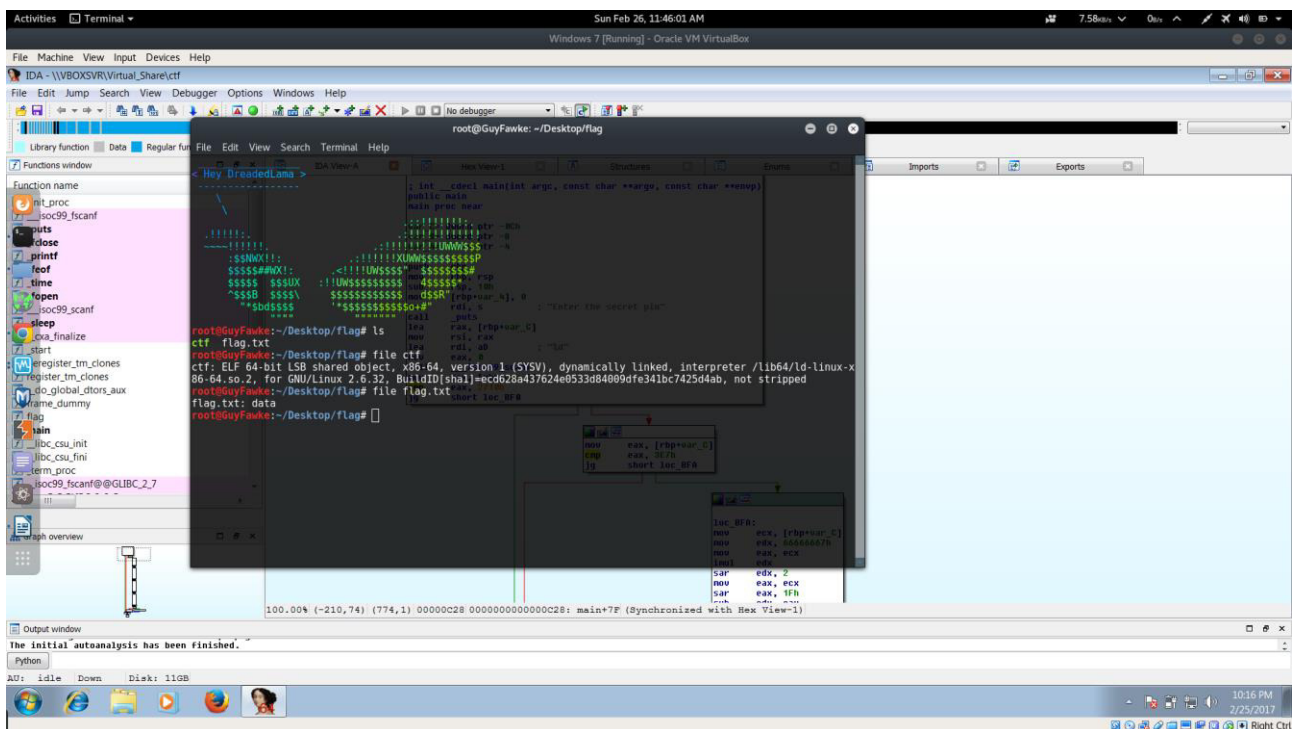
Challenge :

Calling the Uncalled

Even my girlfriend doesn't make me wait this long (UNIX library modification)

Note: Extract the zip into one folder i.e. keep both the files in the same folder

First let us run file command on both the files given to us.

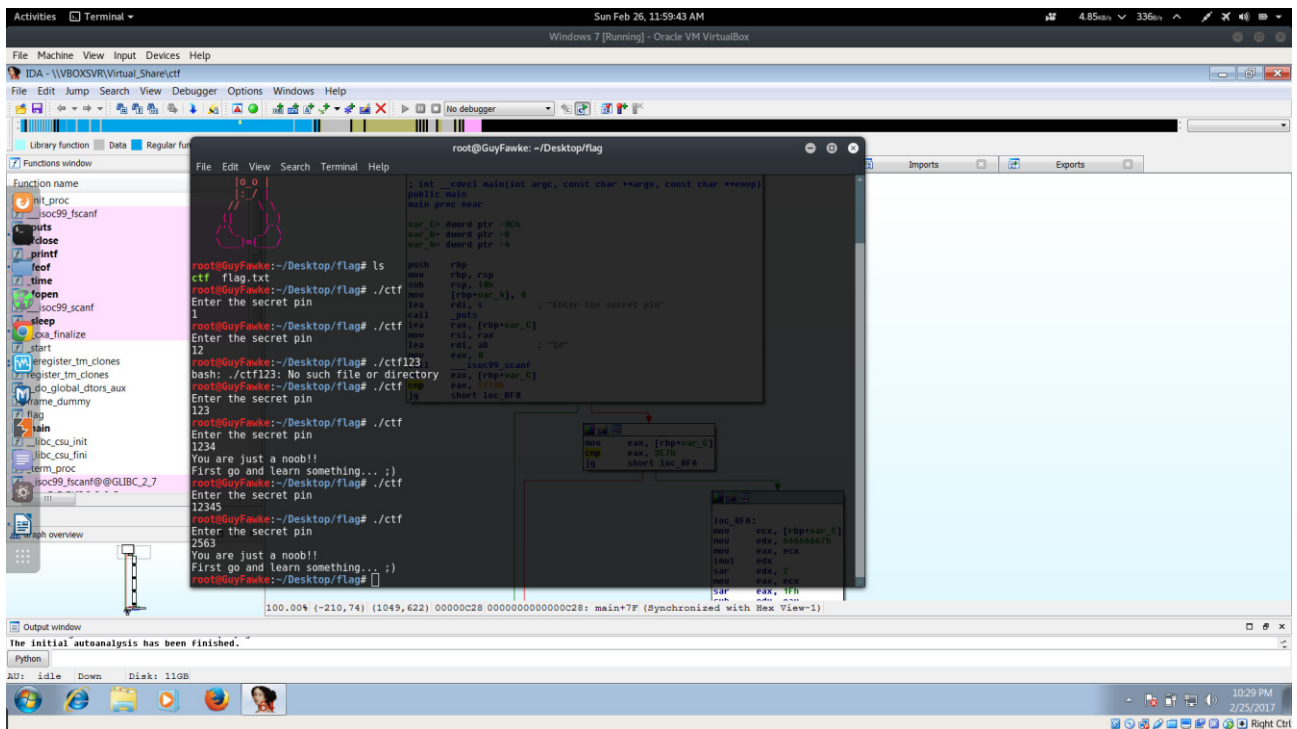


The file command says 'ctf' is ELF executable and 'flag.txt' is data file.

Running cat on flag.txt shows some random text, looks like some encrypted file.

Running strings command doesn't reveal much either.

Let us try executing the file

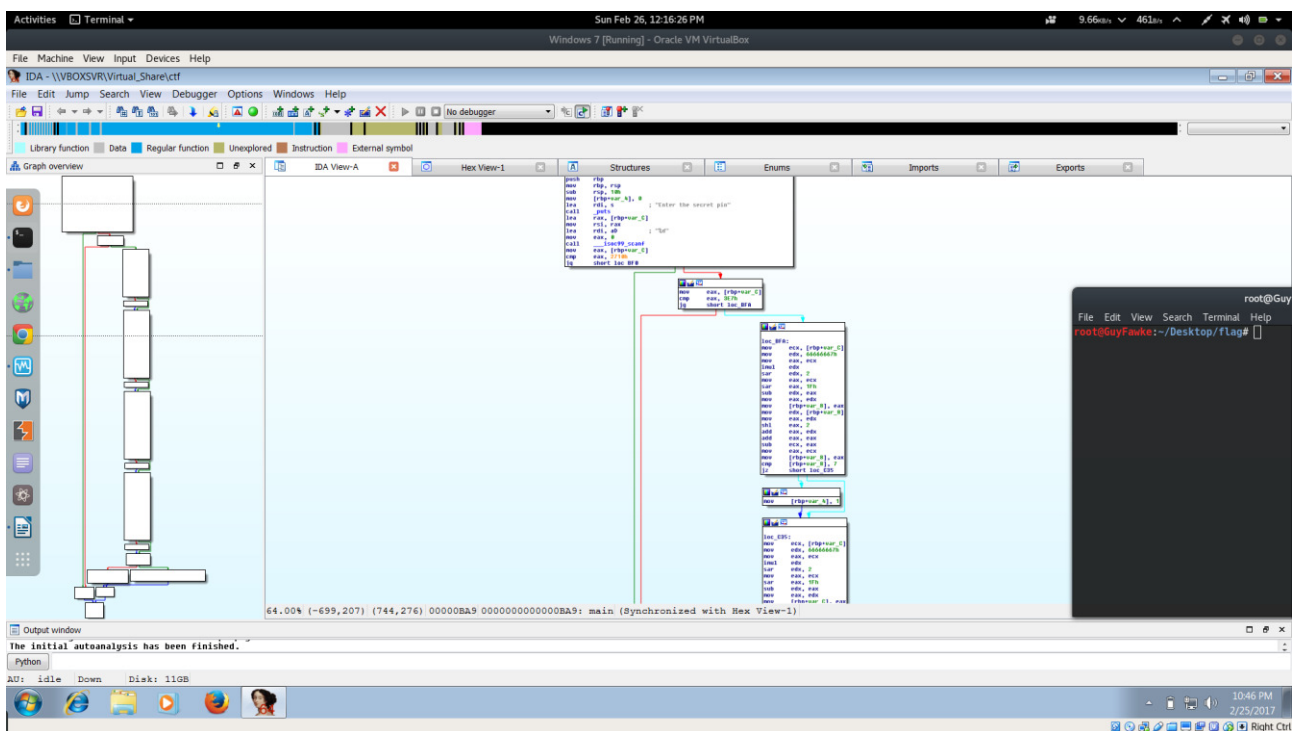


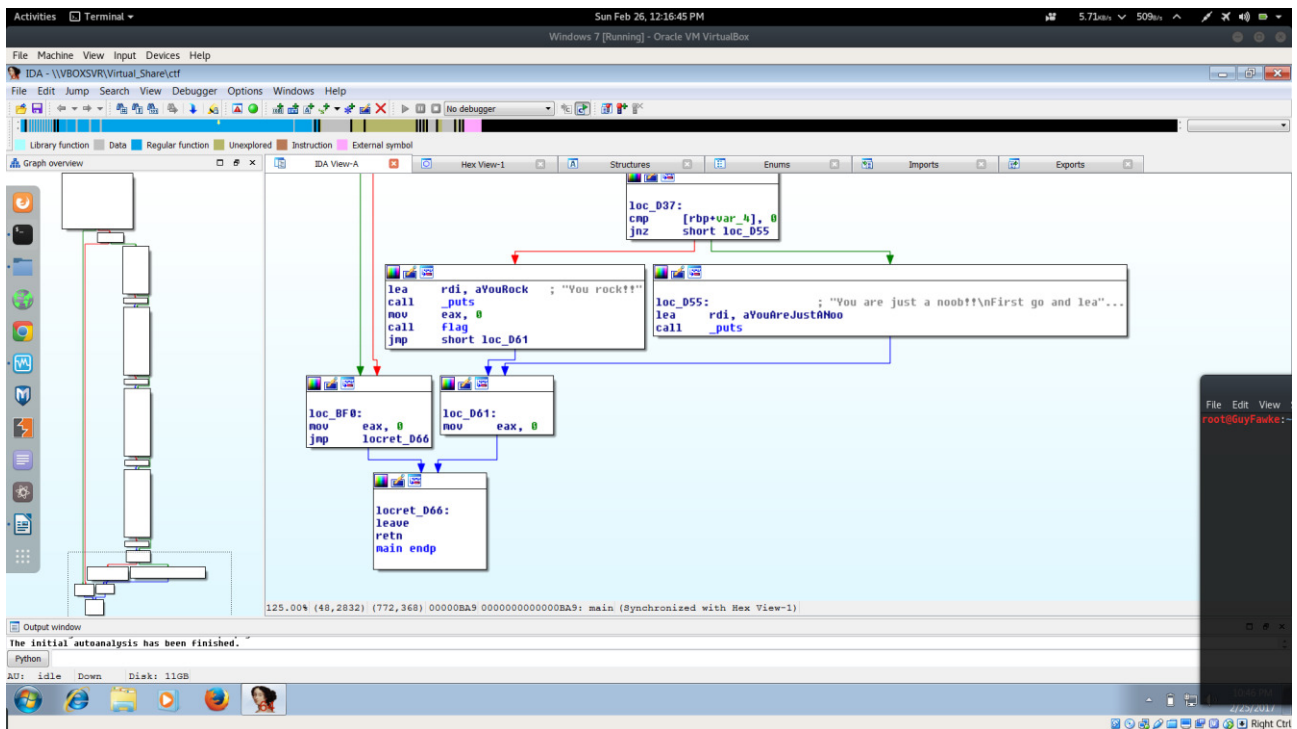
On executing the file its asks for a secret pin.

Trying some random pins with different digits, we notice that on entering 4 digit pin “You are just a noob!!” is displayed.

We get this output only on 4 digits pin, so maybe the pin is a 4 digit pin.

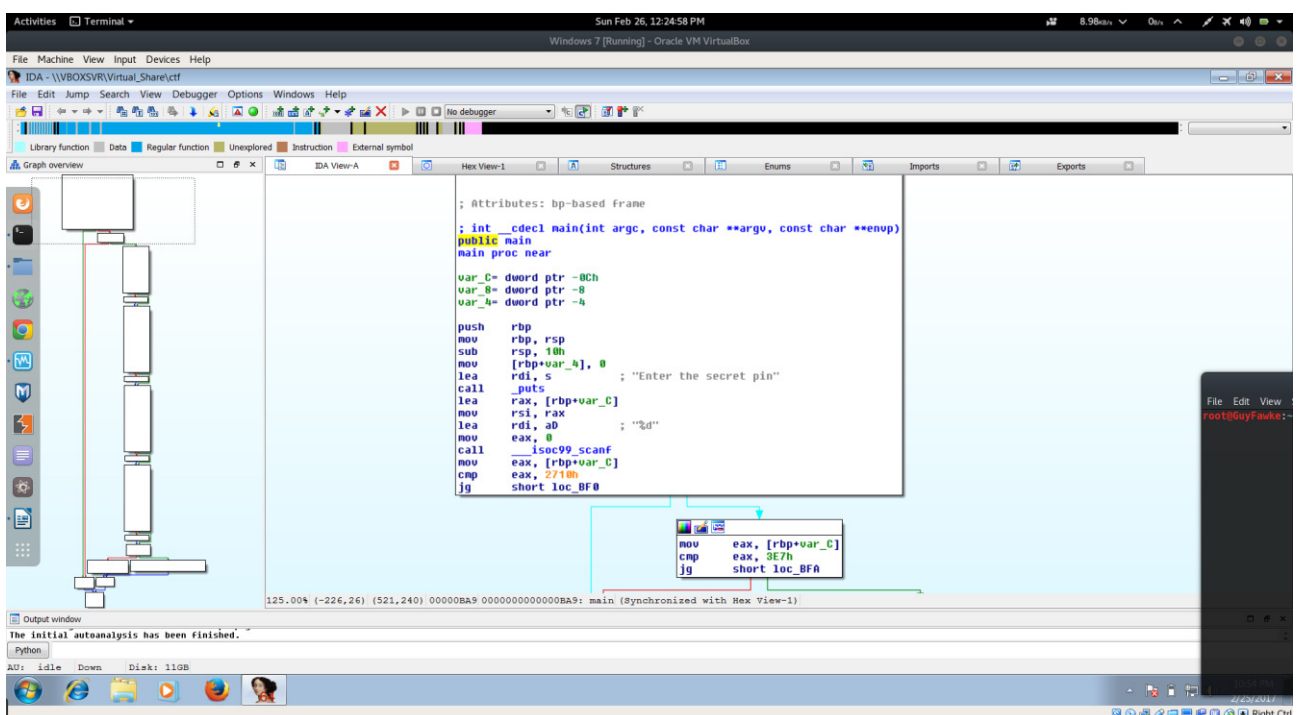
Now let us put this file in IDA (to those who don’t know what it is – its a disassembler used to disassemble executables to assembly codes).





We see few comparisons, few branching but at the end of the graph we notice an interesting branching, showing the string we get when we enter a 4 digit pin "You are just a noob!!". But there is another string in the other branch saying "You Rock!!", that must be the branch that gives us the flag (hope so). On seeing the other branch, we notice that there is an instruction 'call flag' which calls a function flag. That must be the function that prints our flag. So we just need to call it.

So we can either do the time consuming work of understanding the assembly code to get the secret pin or we can change the assembly code to directly call the function flag without asking us for pin.



The screenshot shows a Windows 7 desktop environment. A virtual machine (Oracle VM VirtualBox) is running, displaying a Windows 7 desktop. The main application window is IDA Pro, which is showing the assembly code of a program. The code is in C and assembly, showing a loop that reads a secret pin and prints it. The output window shows the initial autoanalysis has been finished. The taskbar at the bottom shows various applications and the system clock.

IDA Pro Interface:

- File Machine View Input Devices Help
- File Edit Jump Search View Debugger Options Windows Help
- Library function Data Regular function Unexplored Instruction External symbol
- Functions window: Function name, int_proc, soc99_fscanf, puts, close, printf, scanf, time, fopen, soc99_scanf, sleep, soc_finalize, start, register_tm_clones, register_tm_clones, do_global_ctors_aux, frame_dummy, flag, main, libc_csu_init, libc_csu_fini, term_proc, soc99_fscanf@GLIBC_2.7, ...
- Hex View-1:


```

; int __cdecl main(int argc, const char **argv, const char **envp)
public main
main proc near

var_C= dword ptr -0Ch
var_8= dword ptr -8
var_4= dword ptr -4

push    rbp
mov     rbp, rsp
sub     rsp, 10h
mov     [rbp+var_4], 0
lea     rdi, 5 ; "Enter the secret pin"
call    puts
lea     rax, [rbp+var_C]
mov     rsi, rax
lea     rdi, a0 ; "%d"
mov     eax, 0
call    flag
mov     eax, [rbp+var_C]
cmp     eax, 2710h
jg      short loc_BF0

```
- Output window: The initial autoanalysis has been finished. Python
- Taskbar: AU: idle, Down, Disk: 11GB, system clock: 2/25/2017, 11:23 PM

Hell yeah, we do jump to the flag function without entering any pin ;)

But wait, what the heck is this??

Here it just keeps on printing “Ready to get the flag” and the number at starting keeps on increasing.

Moreover to print each instruction the program takes about 1 second (indicates sleep function has been used in the program to delay the output)

So, all we need to do is to wait till we get the flag??

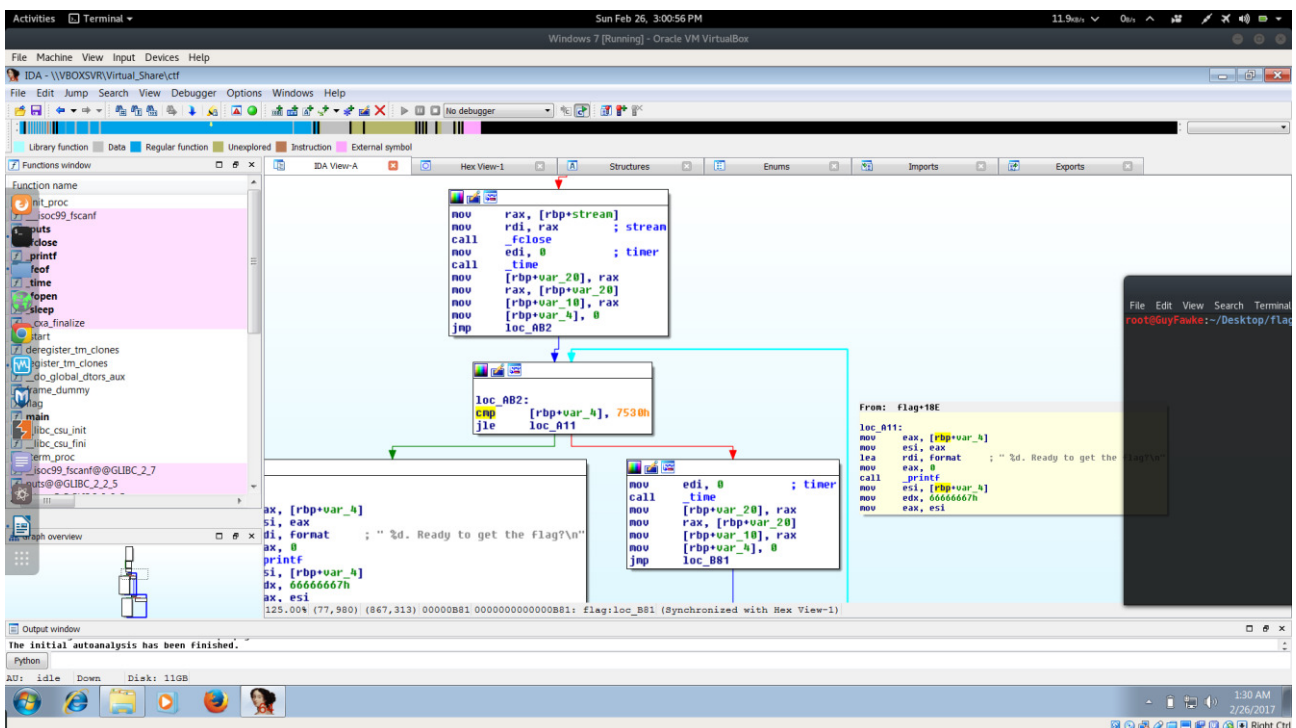
Obviously not, there must be a way around!! :)

Let us jump to the flag function in IDA and see what it has to say.

We see there is a loop which iterates 7530h times (30,000 in decimal). What we need to wait 30,000 sec ?? (As each instruction takes 1 sec , which we can see by running the program)

But wait, this wait isn't over here yet, there is one more such loop which iterates 2710h times (10,000 in decimal).

So in total of 40,000 seconds. Are you f**king kidding me!! :0

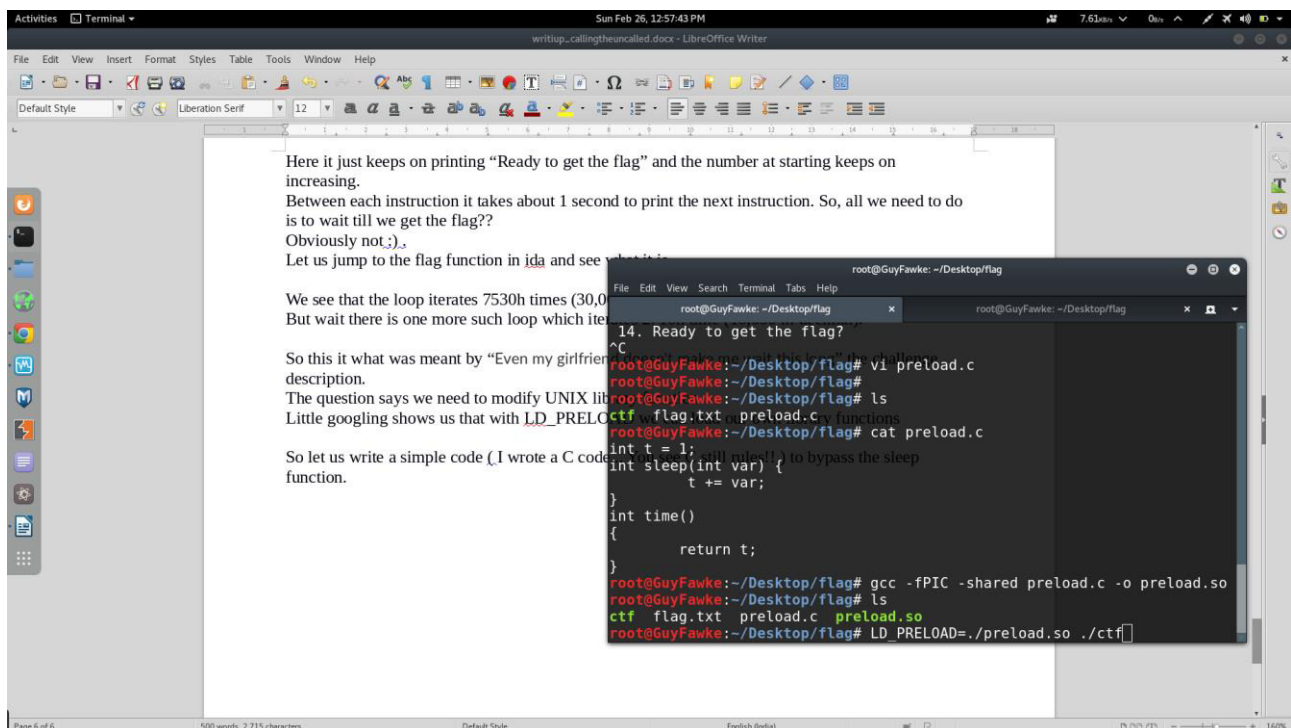


So this is what was meant by “Even my girlfriend doesn't make me wait this long” in the challenge description.

But the question also says we need to modify UNIX library functions.

Little googling shows us that with LD_PRELOAD we can load our own library, overwriting default library functions.

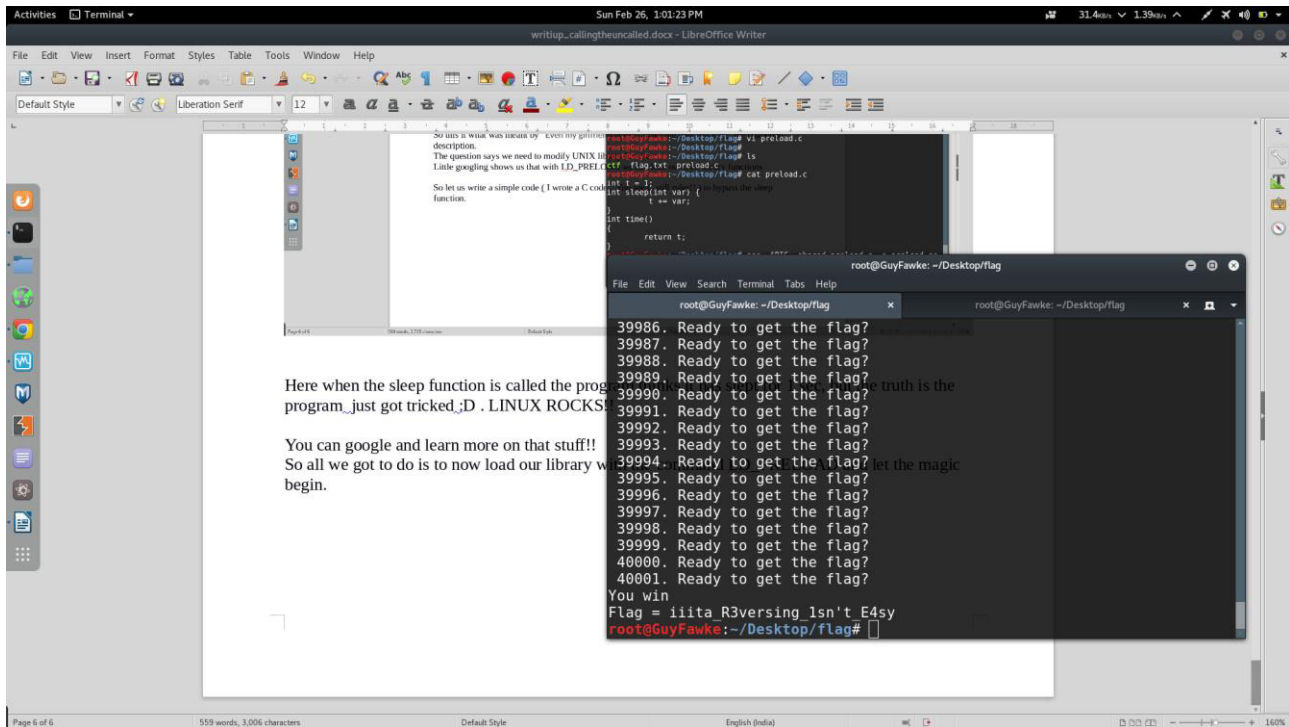
So let us write a simple code (I wrote a C code... You see C still rules!!) to overwrite the sleep and time functions.



Here when the sleep function is called the program thinks it has slept for 1 sec, but the truth is the program just got tricked :D . LINUX ROCKS!!

You can google and learn more on that stuff!!

So all we got to do is to now load our library with the command LD_PRELOAD and let the magic begin.



And Voila!!

Our 40,000 seconds of wait was computed in milliseconds revealing the flag
“iiita_R3versing_1sn’t_E4sy”.

Well that seems true, isn’t it ?? :p

Note: There are many other ways to do this challenge such as converting the assembly code back to C code, and yeah the secret pin was 1337 (Try understanding the assembly code, hope you can figure it out).

-DreadedLama