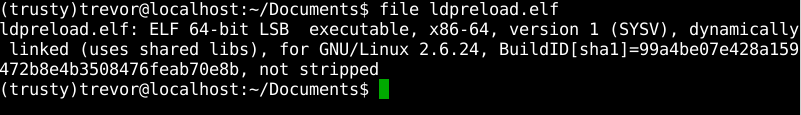
**Intro To Reverse - LD\_PRELOAD Solution**

**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.

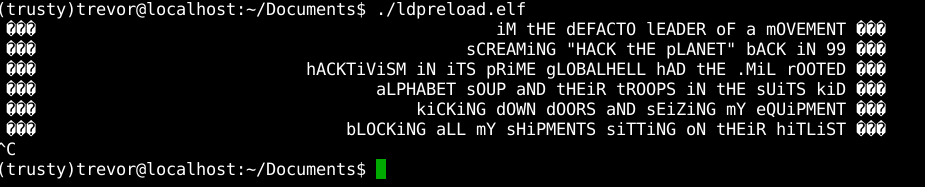
**Task:** Using the LD\_Preload trick, and the disassembler / debugger of your choice retrieve the flag from the provided binary.

**Solving:** 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.



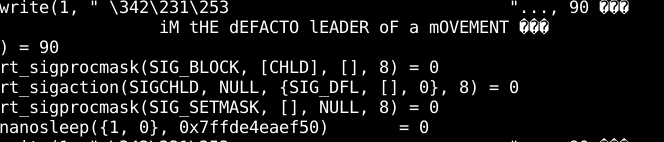
Nothing out of the ordinary here. I’ve omitted the output from strings because it was largely uninteresting other than a string that said “OK YOU WIN HERE’S YOUR FLAG!” with no output after that.

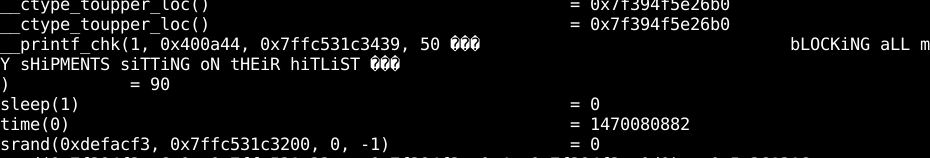
Let’s move on to executing the program to see what happens, remember to chmod the binary if need be.



All this program seems to do is output some song lyrics, if you wait long enough the program loops and continues repeating the lyrics. Also, am I crazy or is the delay between each output getting longer as the program executes?

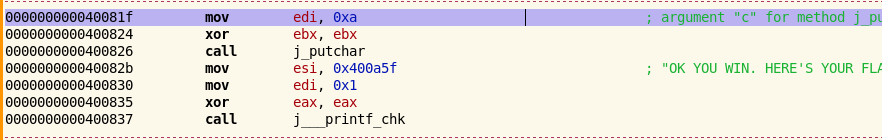
Let’s take a look at the output from both strace and ltrace and see what calls and libraries this program is using. Maybe that will give us a better idea of what’s happening here. The output from strace is first, ltrace second. Both screenshots are trimmed due to repeated behavior between lyrics.

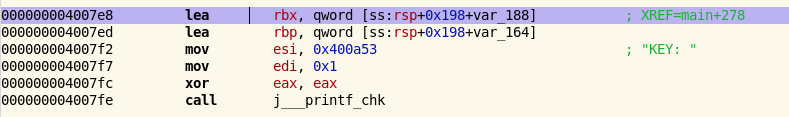




Between each lyric output the program makes a call to the c functions sleep() and time(). Calls to these functions are responsible for the pause between each lyric output. Now that we have a better picture of how this program executes let's open the program in the disassembler of your choice and see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

Inside of hopper there are two strings of interest, the first is “OK YOU WIN. HERE’S YOUR FLAG:” and the “Key: “ string. The first string is located at address 0x40081f, and the second at address 0x4007e8. I’ve put the assembly instructions provided by Hopper below.



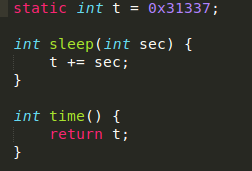


Open up the debugger of your choice and try jumping to those addresses manually and see what happens!

Hopefully you noticed that if you try to jump to the addresses the outputted flag is still garbage when we jump to 0x4007E8. Jumping to 0x400803 immediately segfaults. Not what we want. It was a longshot but it was worth the try.

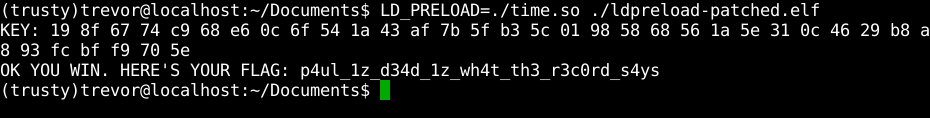
Now it’s time to get into the meat of this challenge. What if you could fake both the sleep calls and elapsed time, tricking the program into thinking it has actually slept for the required number of seconds? You can do that by overriding GLIBC’s sleep() and time() functions very easily: create a C file 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](https://rafalcieslak.wordpress.com/2013/04/02/dynamic-linker-tricks-using-ld_preload-to-cheat-inject-features-and-investigate-programs/). When you’d run the binary, the sleep function will execute our implementation, while it still increases the elapsed time.

What we need to do now is write our own implementations of time() and sleep() to achieve this desired effect. To save time I’ve attached a copy of the code, it is also presented in the screenshot below.



Now all that’s left to do is compile this piece of code into a shared object ‘gcc -fpic -shared -o time.so time.c’ and execute the binary with LD\_Preload! ‘LD\_PRELOAD=./time.so ./ldpreload-patched’

I’ve provided a patched binary that doesn’t print out the song lyrics because that takes time. If you would like to test the LD\_Preload trick against the original binary you’re more than welcome to, it’ll just take some time to print every iteration of the lyrics.



Congratulations! You now know how to use the LD\_Preload trick!