Traced

Problem: try me

Given: insanity

Hint: Traces are great

Introduction:Traced is an easy challenge designed to teach students additional steps to perform during a basic analysis against a target binary to learn more about the binary. Using the commands ‘strace’, and ‘ltrace’. These commands should be some of your first go-tos when trying to learn more about a target executable.

Steps:
(1) The problem title states “Traced”. It indicates we use ‘strace’, and ‘ltrace’ for this problem. These two commands are helpful in determining a program's general behavior as it executes. strace is a program that will print out every system call or signal a program makes or receives during its execution. strace’s basic syntax is: “strace ./binaryToExecute”. Go ahead and run strace against the insanity binary.

```
[qijun@glap reverse]$ strace ./insanity
execve("./insanity", ["./insanity"], [/* 71 vars */]) = 0
strace: [ Process PID=23563 runs in 32 bit mode. ]
brk(NULL)                               = 0x8e93000
mmap2(NULL, 4096, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0xf7
792000
access("/etc/ld.so.preload", R_OK)      = -1 ENOENT (No such file or directory)
open("/opt/pps/lib/tls/i686/sse2/libc.so.6", O_RDONLY|O_CLOEXEC) = -1 ENOENT (No
 such file or directory)
stat64("/opt/pps/lib/tls/i686/sse2", 0xff8eb980) = -1 ENOENT (No such file or di
rectory)
open("/opt/pps/lib/tls/i686/libc.so.6", O_RDONLY|O_CLOEXEC) = -1 ENOENT (No such
 file or directory)
stat64("/opt/pps/lib/tls/i686", 0xff8eb980) = -1 ENOENT (No such file or directo
ry)
open("/opt/pps/lib/tls/sse2/libc.so.6", O_RDONLY|O_CLOEXEC) = -1 ENOENT (No such
 file or directory)
```

Each line output by strace represents either a system call or signal received by the ‘insanity’ program during execution. For instance we can see calls to ‘execve()’, ‘open()’, and ‘mmap2()’. Scroll down a little further in the output and you’ll see the system call to ‘write()’ that is responsible for printing out “Reticulating Splines….” to the screen.

(2) Now, we try the ltrace command. ltrace functions similarly to strace with the minor difference that ltrace prints out library calls made by a program during execution. Its syntax is: “ltrace ./binaryToExecute”. Run the command and we get the output below. If you’ve executed the binary normally before you should recognize some of these calls. ‘Sleep()’ is responsible for that boring 5 second delay before printing a message which is the value of the flag!. You can also see a call to the ‘puts()’ function responsible for printing “Reticulating splines”. Finally we can see information we normally can’t; there’s a call to puts() which should print out “Your ability to

hack is….” right before the program exits. Now while none of this information is particularly useful for this challenge. I wanted to demonstrate the power behind the strace and ltrace commands when learning about how to approach a target binary.

```
[qijun@glap reverse]$ ltrace ./insanity
__libc_start_main([ "./insanity" ] <unfinished ...>
puts("Reticulating splines, please wai"...Reticulating splines, please wait..
)       = 36
sleep(5)                                          = 0
time(nil)                                         = 1503680730
srand(1503680730)                                 = <void>
rand()                                            = 604523112
puts("rm -rf / : Permission denied"rm -rf / : Permission denied
)             = 29
+++ exited (status 0) +++
```