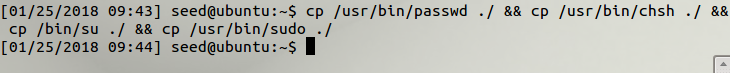
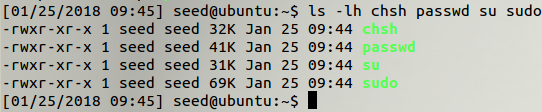
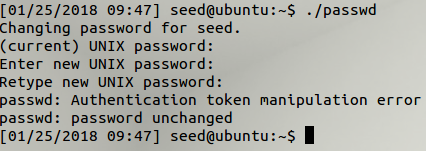
First, copied the listed executables into home directory:



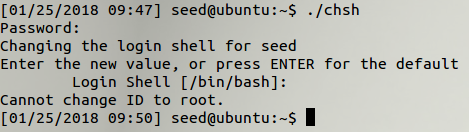
The copied files don’t have the Set-UID bit set:

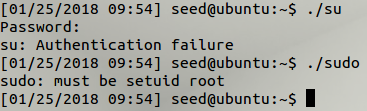


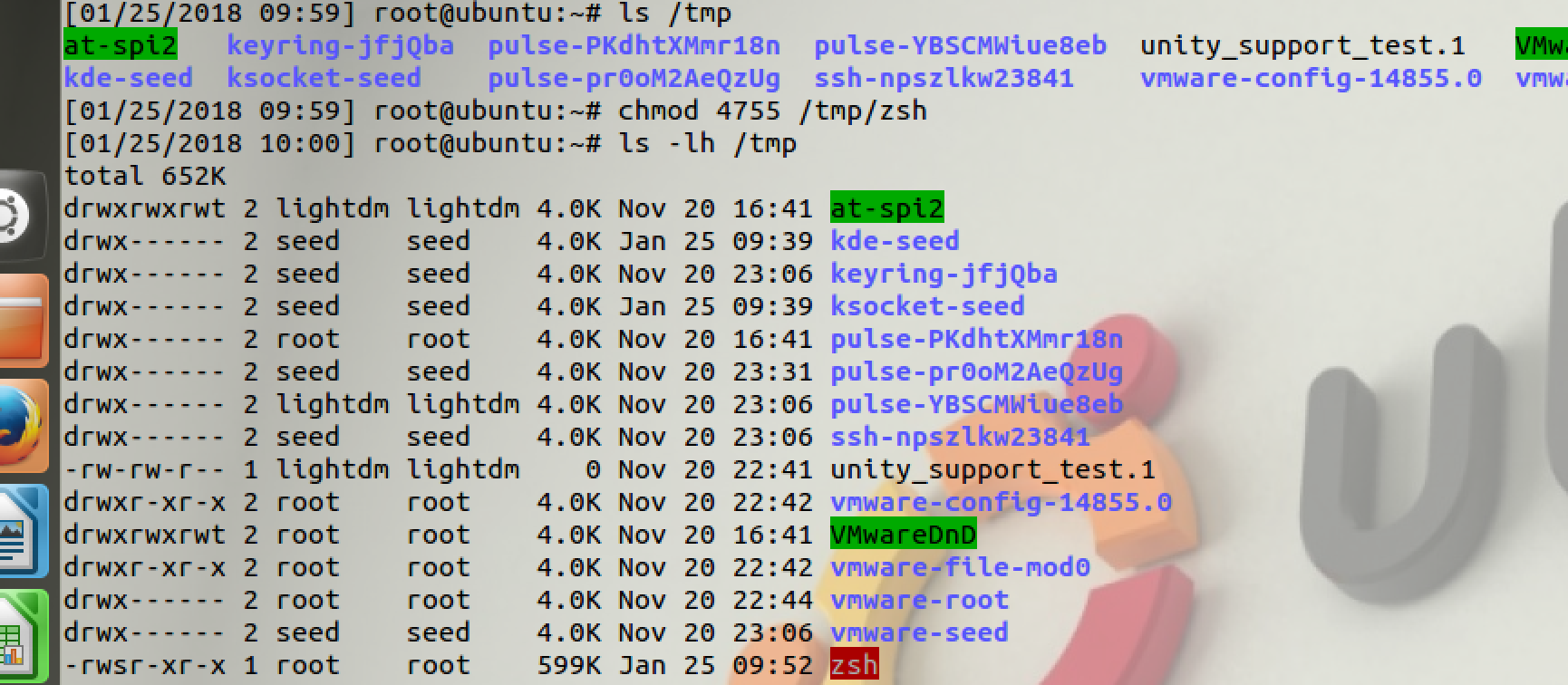
Running the local ./passwd executable gives the following error:



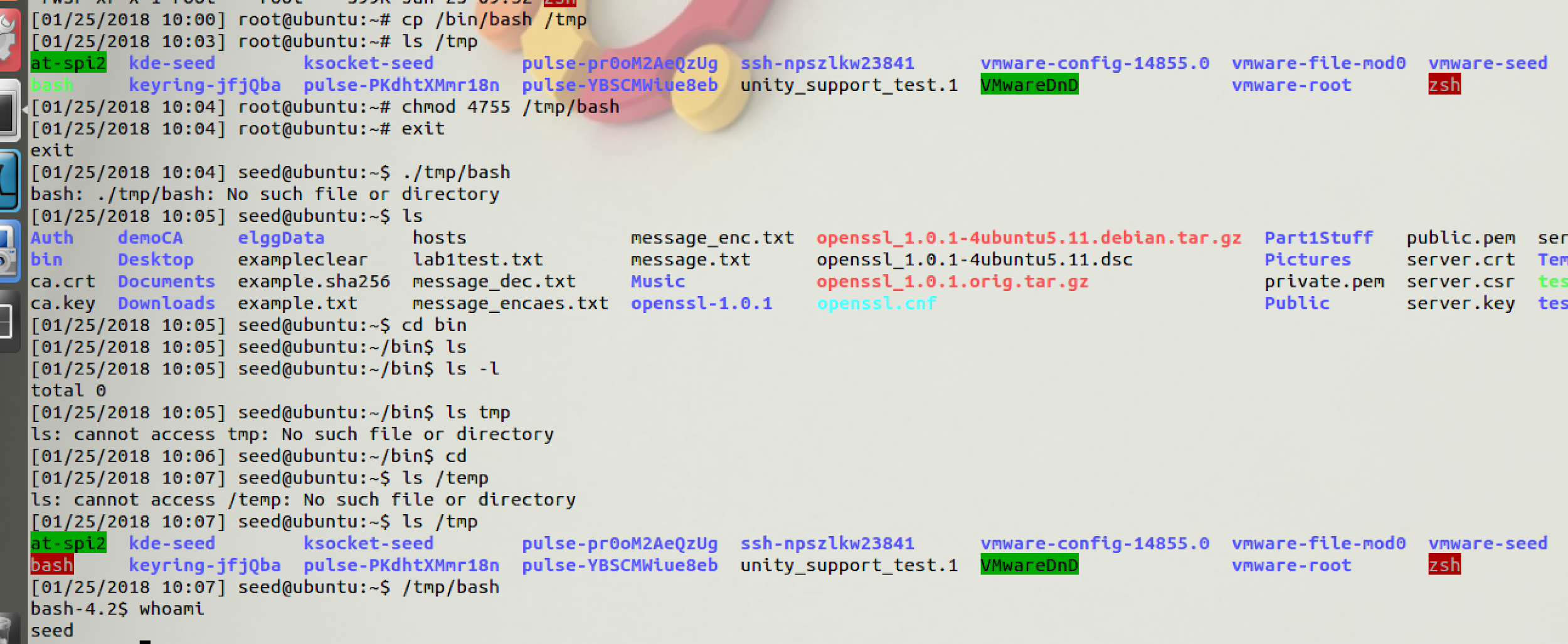
Similar errors are encountered running the other executables, with the “sudo” program explicitly showing that not having set-uid is the source of the error:





2.   


Will you get root privilege?  
Yes, root privilege was granted.



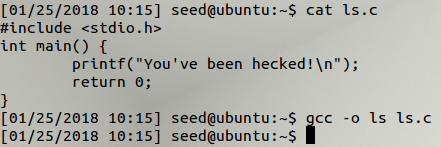
When you copy /bin/bash to /tmp and try to run /tmp/bash as a normal user, you will not get root privilege. This can be seen by typing the “whoami” command, which displayed the user as “seed” and not “root”. The root privilege was not achieved.

3. Set zsh to default shell by removing sh link:

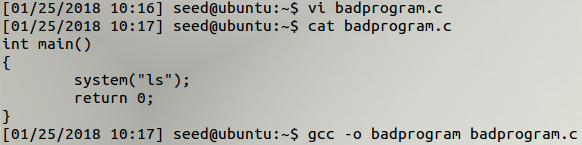


4. First, set the PATH directory to the home directory:  


Then, wrote a program and compiled it to “ls” in the current directory:



Copied and compiled the example code in part (4), and set the owner and permissions:





Running the program executes the code written by the attacker:



5. Difference between system() and execve()

1. **Set q=0 in program so it will use system() to invoke the command.**

**Is this program safe?**   
  
No, the program is not safe and is susceptible to a malicious user inputting commands as opposed to data instead.

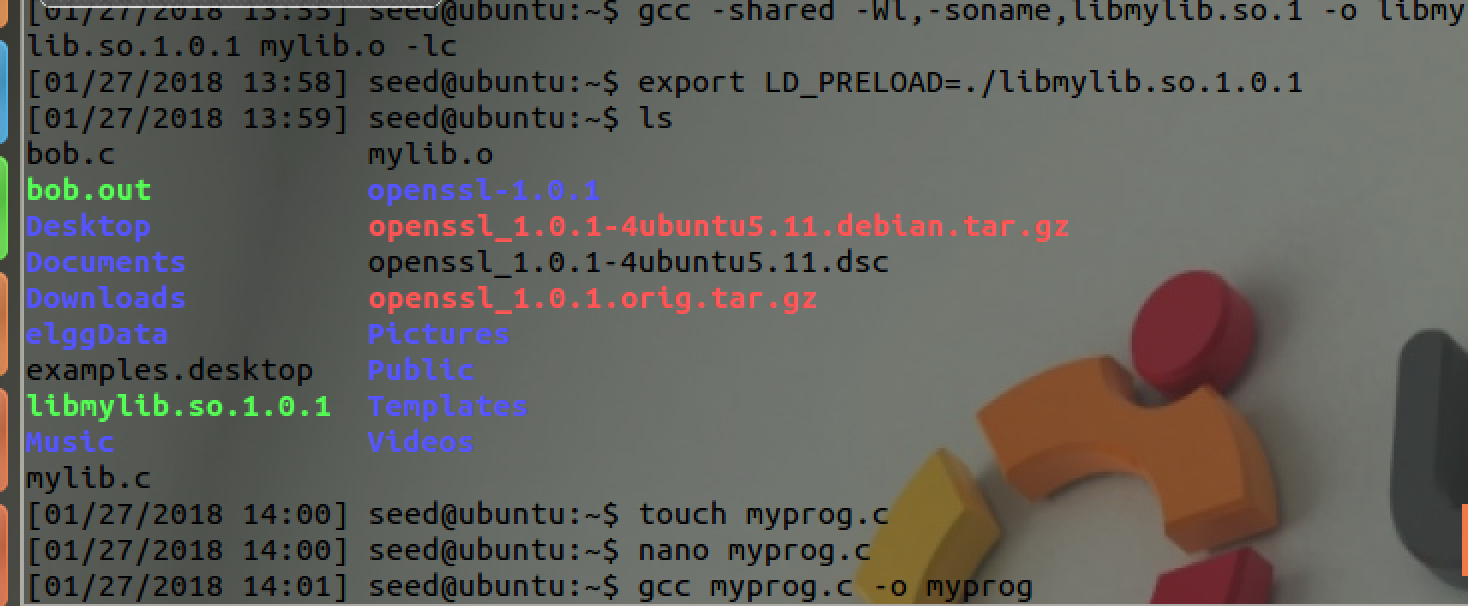
**If you were Bob, can you compromise the integrity of the system?**  
  
Yes; Bob can compromise the integrity of the of the system because since it invokes /bin/sh program and does not exit; Bob (or me) will have full privileges and access to shell since this program is set-root-uid. This would allow full modification and delete capabilities.

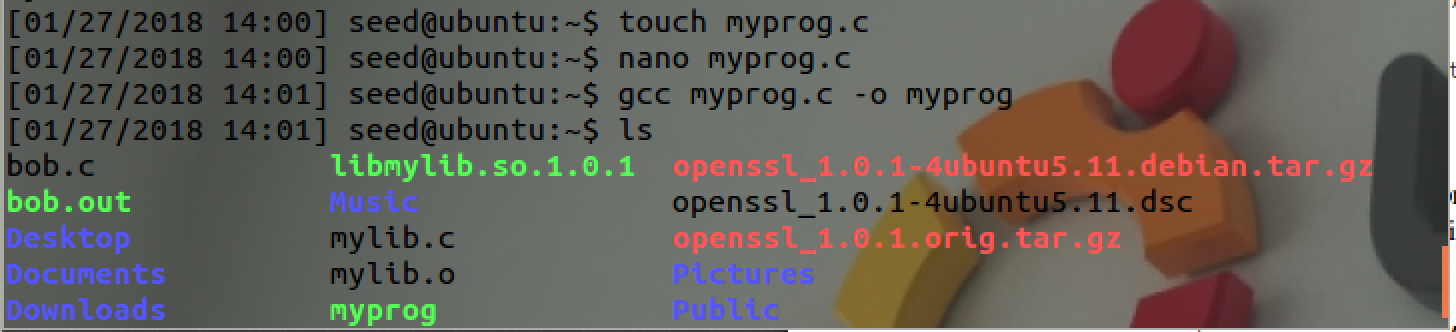
**B. Set q=1 in program so it will use execve() to invoke the command. Do the attacks from task a still work? Describe/explain observations.**

No, they do not work. This is because when you use execve, the command name and data are clearly separated and there is not a way for user data to become code.

6. The LD\_PRELOAD environmental variable

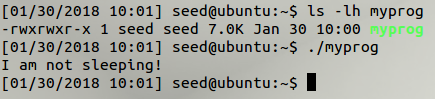
Created *mylib.c* as written, compiled and set LD\_PRELOAD environment variable.

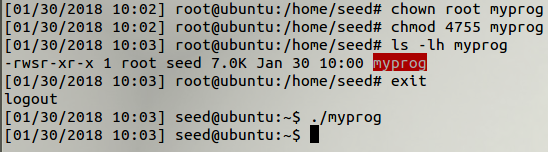


Created and compiled *myprog* in the same directory.  


Run myprog under the following conditions, and observe what happens.

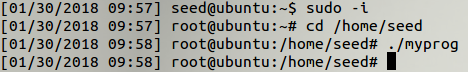
* Make *myprog* a regular program, and run it as a normal user.

  
  
This program runs successfully, because the program is owned and executed by seed, and the LD\_PRELOAD environment variable is recognized.

* Make myprog a Set-UID root program, and run it as a normal user  
    
  

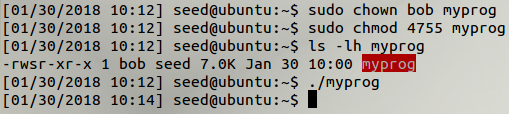
When running the Set-UID program as a normal user, the user-written dynamic library is not loaded, because the effective UID that the program executes under is the root account, which does not have the alternate LD\_PRELOAD environment variable set.

* Make myprog a Set-UID root program and run it in the root account:



Make myprog a Set\_UID root program, and run it in the root account. When running the program as the root user, the alternative dynamically linked library that we wrote wasn't loaded because the LD\_Preload environment variable was not set.

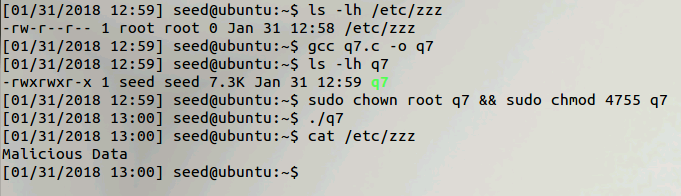
* Make myprog a Set-UID user1 program and run it as a different (non-root) user



Used user “bob” instead of user1, setting the file owner to “bob” and marking the program as a Set-UID program resulted in execution not loading the user-written dynamic library.

7. Relinquishing privileges and cleanup

Compiled the code, set permissions and executed the program with the Set-UID bit set and proper permissions in the /etc/zzz file:



The child process was able to access the file; the parent program that was executed had the Set-UID bit set, so the child process was also spawned as the owner’s UID (root).