Justin Gallagher

CSC 482

Environment Variable and SetUID

Environment variables are a set of dynamic named values that can affect the way running processes will behave on a computer. Not all programmers are familiar with environment variables, which in turn can make them a possible vulnerability.

In this lab, we will combine modifying environment variables with SetUID programs to achieve local privilege escalation, an important step that attackers need to accomplish in most circumstances. In addition, it will help us understand how environment variables work, how they are propagated from parent process to child, and how they affect system/program behaviors.

Task 1: Manipulating Environment Variables

In the first task, we look at the commands used to set and unset environment variables. The commands introduced are `printenv` and `env` for printing out the variables, and `export` and `unset` which are used to set and unset environment variables. I was able to output the environment variables using both the `printenv` and `env` commands, along with the `sort` command to sort the output for easier reading. I then used `export` to create my own environment variable TEMPVAR and set it equal to the value 1. Then using `printenv`, I was able to confirm that the TEMPVAR environment variable had been created. Lastly, I used `unset` to delete TEMPVAR, because it was just being used for practice and has no practical use. Using the `printenv` command again, I confirmed TEMPVAR had been unset. The commands used and some of the output (output designated by '>') is listed below:

```
printenv | sort > task1_vars_printenv
env | sort > task1_vars_env
export TEMPVAR=1
printenv TEMPVAR
> 1
unset TEMPVAR
printenv TEMPVAR       (no output, TEMPVAR unset properly)
```

Task 2: Passing Environment Variables from Parent Process to Child Process

In this task, we study how a child process gets its environment variables from its parents and ultimately determine whether the parent's environment variables are inherited by the child process or not. A C program is given, and we are instructed to compile and run it. The output is then saved to a file. Afterwards, we are instructed to change part of the program (commenting out the portion that prints the child, and adding a portion the prints the parent), then compile and run it. The output is saved to a different file than the one from before. We are then asked to compare the two output files using the `diff` command. After doing so, it appears the files are identical. After doing some reading of the fork man pages, I was able to determine that the environment variables are passed from the parent to the child process, however there are a few things the child does not inherit (child has unique PID). The commands used in this task are below:

```
gcc -o task2 task2.c
./task2 | sort > task2_child_vars
gcc -o task2 task2.c                    (recompile task3.c after changing
code)
./task2 | sort > task2_parent_vars
diff task2_child_vars task2_parent_vars    (no output from this command)
```

Task 3: Environment Variables and `execve()`

For this task, we discover and learn about the `execve` command, which sends a system call to load a new command and execute it, but it has no return. No new process is created; instead, the calling process's text, data, bss, and stack are overwritten by that of the program loaded. We are to determine what happens to the environment variables when using this command. The first step of this task has us writing and compiling a given C program that includes the execve command and prints out the current processes to the screen. Upon compiling, I noticed a warning given by the compiler:

```
[08/31/20]seed@VM:~/.../lab2$ gcc -o task3 task3.c
task3.c: In function 'main':
task3.c:13:2: warning: implicit declaration of function 'execve' [-Wimplicit-fun
ction-declaration]
  execve("/usr/bin/env", argv, NULL);
  ^
```

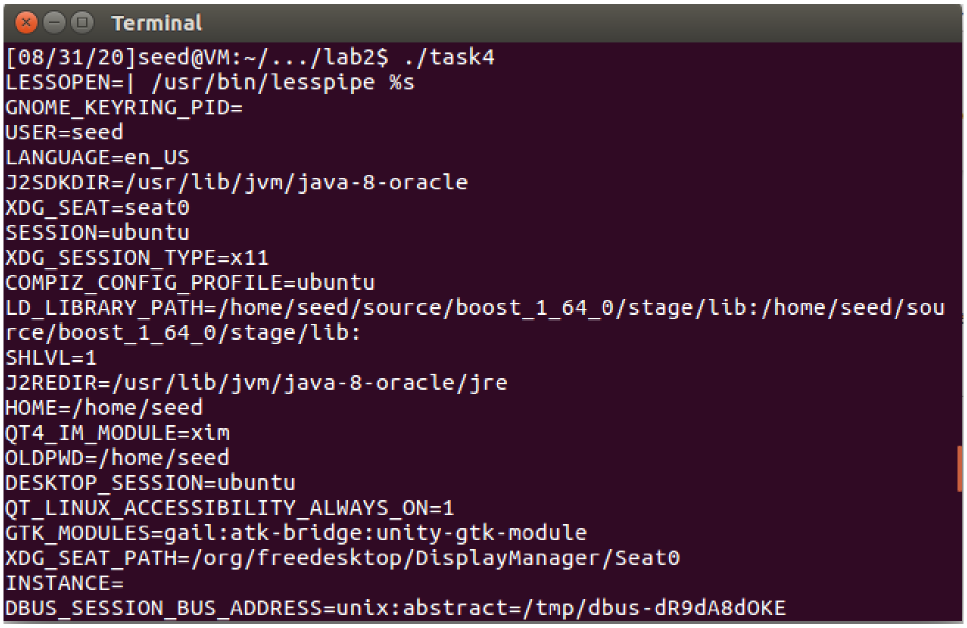
When the program is run, there is no output. My reasoning is that giving NULL for the parameter of the function (the environment variables) causes this command to not print out the current environment variables. Step 2 has us change a line of code where the execve command is located, specifically changing the parameter "NULL" to "environ." Upon compiling, the same error as before is given. When the program is run, this time it outputs the current processes. Based on my observations from running these two versions of this program, using NULL for the environment variables parameter doesn't store it in memory thus the process does not inherit the environment variables. Changing the program to include the environment variables ensures that they are stored in memory and then process inherits them. The following commands were used:

```
gcc -o task3 task3.c
./task3 | sort > task3_vars
```

Task 4: Environment Variables and `system()`

In this task, we look at another command called `system`, which is used to execute commands. Unlike execve which executes commands directly, system first executes /bin/sh and asks the shell to execute the command. Using system, the environment variables of the calling process are passed to the new program /bin/sh. The commands below helped to verify whether this was true or not:

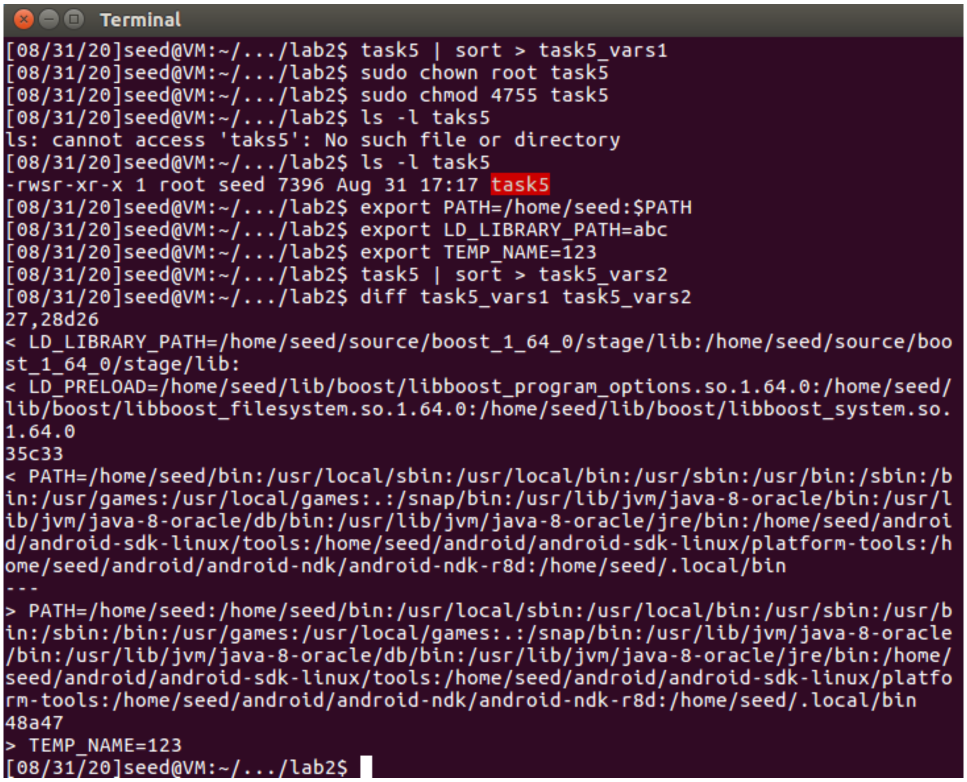
```
gcc -o task4 task4.c
./task4 | sort > task4_vars
```



```
Terminal
[08/31/20]seed@VM:~/.../lab2$ ./task4
LESSOPEN=| /usr/bin/lesspipe %s
GNOME_KEYRING_PID=
USER=seed
LANGUAGE=en_US
J2SDKDIR=/usr/lib/jvm/java-8-oracle
XDG_SEAT=seat0
SESSION=ubuntu
XDG_SESSION_TYPE=x11
COMPIZ_CONFIG_PROFILE=ubuntu
LD_LIBRARY_PATH=/home/seed/source/boost_1_64_0/stage/lib:/home/seed/sou
rce/boost_1_64_0/stage/lib:
SHLVL=1
J2REDIR=/usr/lib/jvm/java-8-oracle/jre
HOME=/home/seed
QT4_IM_MODULE=xim
OLDPWD=/home/seed
DESKTOP_SESSION=ubuntu
QT_LINUX_ACCESSIBILITY_ALWAYS_ON=1
GTK_MODULES=gail:atk-bridge:unity-gtk-module
XDG_SEAT_PATH=/org/freedesktop/DisplayManager/Seat0
INSTANCE=
DBUS_SESSION_BUS_ADDRESS=unix:abstract=/tmp/dbus-dR9dA8dOKE
```

Task 5: Environment Variables and `Set-UID` Programs

In this task, we look at Set UID. Set UID is a security mechanism in Unix, and when a Set-UID program runs, it inherits the owner's privileges. Set-UID can be affected by environment variables with this task having us figure out what environment variables are inherited by the Set-UID program's process from the user's process. In the first step, we write and compile C code that prints out all the environment variables of the current process. I ran the program and stored a sorted output in a file called "task5_vars1". Step 2 had us change the programs ownership to root and make it a Set-UID program. Step 3 then had us set new environment variables using export. I ran the program again, this time saving the sorted output to a file called "task5_vars2". I used the diff command to compare the two outputs and determined that the PATH and TEMP_NAME environment variables were inherited by the Set_UID program but the LD_LIBRARY_PATH is not. This is because LD_LIBRARY_PATH is a path shared by many libraries, so it is automatically not inherited, and a default path is set. Below is a picture containing the commands used in this task.



```
Terminal
[08/31/20]seed@VM:~/.../lab2$ task5 | sort > task5_vars1
[08/31/20]seed@VM:~/.../lab2$ sudo chown root task5
[08/31/20]seed@VM:~/.../lab2$ sudo chmod 4755 task5
[08/31/20]seed@VM:~/.../lab2$ ls -l taks5
ls: cannot access 'taks5': No such file or directory
[08/31/20]seed@VM:~/.../lab2$ ls -l task5
-rwsr-xr-x 1 root seed 7396 Aug 31 17:17 task5
[08/31/20]seed@VM:~/.../lab2$ export PATH=/home/seed:$PATH
[08/31/20]seed@VM:~/.../lab2$ export LD_LIBRARY_PATH=abc
[08/31/20]seed@VM:~/.../lab2$ export TEMP_NAME=123
[08/31/20]seed@VM:~/.../lab2$ task5 | sort > task5_vars2
[08/31/20]seed@VM:~/.../lab2$ diff task5_vars1 task5_vars2
27,28d26
< LD_LIBRARY_PATH=/home/seed/source/boost_1_64_0/stage/lib:/home/seed/source/boo
st_1_64_0/stage/lib:
< LD_PRELOAD=/home/seed/lib/boost/libboost_program_options.so.1.64.0:/home/seed/
lib/boost/libboost_filesystem.so.1.64.0:/home/seed/lib/boost/libboost_system.so.
1.64.0
35c33
< PATH=/home/seed/bin:/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/b
in:/usr/games:/usr/local/games:.:/snap/bin:/usr/lib/jvm/java-8-oracle/bin:/usr/l
ib/jvm/java-8-oracle/db/bin:/usr/lib/jvm/java-8-oracle/jre/bin:/home/seed/androi
d/android-sdk-linux/tools:/home/seed/android/android-sdk-linux/platform-tools:/h
ome/seed/android/android-ndk/android-ndk-r8d:/home/seed/.local/bin
---
> PATH=/home/seed:/home/seed/bin:/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/b
in:/sbin:/bin:/usr/games:/usr/local/games:.:/snap/bin:/usr/lib/jvm/java-8-oracle
/bin:/usr/lib/jvm/java-8-oracle/db/bin:/usr/lib/jvm/java-8-oracle/jre/bin:/home/
seed/android/android-sdk-linux/tools:/home/seed/android/android-sdk-linux/platfo
rm-tools:/home/seed/android/android-ndk/android-ndk-r8d:/home/seed/.local/bin
48a47
> TEMP_NAME=123
[08/31/20]seed@VM:~/.../lab2$
```

Task 6: The PATH Environment Variable and `Set-UID` Programs

In this task, we are seeing how dangers can arise from using the system function in Set-UID programs. The actual behavior of the shell program can be affected by environment variables, which can be altered by malicious users. We changed the PATH environment variable to add the following directory, "/home/seed" to the beginning of the variable. We then wrote and compiled a Set-UID program that executes the ls command. We changed the programs ownership to root and set it to be a Set-UID program. After running the code, I found that it performs exactly as the ls command would. To test and see if I could alter this, I changed the environment variable PATH to the specific directory I had all the files for this lab in. I then created another C program that prints "Hello World!" to the console and named it 'ls.c' After compiling 'ls.c', I was left with a program called 'ls'. Running the task6 program again, we can see that since PATH has been altered, it prints "Hello World!" to the console. This is because we used a relative path in our task6 program, the program only searched the directory it's in for the 'ls' command and since the one I created was found that will be the one to execute. A picture of commands used is included below along with a picture of the 'ls.c' file.

ls.c

```c
#include <stdlib.h>
#include <stdio.h>

int main(){
    printf("Hello World!\n");
}
```

Terminal

```
[08/31/20]seed@VM:~$ cd Documents/lab2
[08/31/20]seed@VM:~/.../lab2$ gcc -o task6 task6.c
task6.c: In function 'main':
task6.c:3:2: warning: implicit declaration of function 'system' [-Wimplicit-func
tion-declaration]
  system("ls");
  ^
[08/31/20]seed@VM:~/.../lab2$ sudo chown root task6
[08/31/20]seed@VM:~/.../lab2$ sudo chmod 4755 task6
[08/31/20]seed@VM:~/.../lab2$ export PATH=/home/seed:$PATH
[08/31/20]seed@VM:~/.../lab2$ task6
ls                    task2              task3       task4.c      task5_vars2
ls.c                  task2.c            task3.c     task5        task6
task1_vars_env        task2_child_vars   task3_vars  task5.c      task6.c
task1_vars_printenv   task2_parent_vars  task4       task5_vars1
[08/31/20]seed@VM:~/.../lab2$ gcc -o ls ls.c
[08/31/20]seed@VM:~/.../lab2$ export PATH=/home/seed/Documents/lab2:$PATH
[08/31/20]seed@VM:~/.../lab2$ task6
Hello World!
[08/31/20]seed@VM:~/.../lab2$ 
```

Task 7: The LD_PRELOAD Environment Variable and Set-UID Programs

In this task, we study how Set-UID programs deal with some of the environment variables. Several environment variables influence the behavior of dynamic loader/linker. A dynamic loader/linker is the part of the OS that loads and links the shared libraries needed by an executable at run time. In Linux, ld.so or ld-linux.so, are the dynamic loader/linker. Among the environment variables that affect their behaviors, LD_LIBRARY_PATH and LD_PRELOAD are the two that we are concerned about. In Linux, LD_LIBRARY_PATH is a colon-separated set of directories where libraries should be searched for first, before the standard set of directories. LD_PRELOAD specifies a list of additional, user-specified, shared libraries to be loaded before all others. In step 1, we want to see how environment variables affect the dynamic loader/linker. We start by building a dynamic link library. We programed and compiled 'mylib.c' to create the library 'libmylib.so.1.0.0.' After, I set the LD_PRELOAD environment to the file location of the dynamic link library. Then I coded and compiled another program called 'myprog.c'. Upon running it the first time, it printed "I am not sleeping!". I then changed 'myprog' ownership to root and made it Set-UID. I checked to see if it was a Set-UID program, and afterwards ran 'myprog' again. This time, the system slept for 1 second, then the program ended. I then entered root and set LD_PRELOAD to the dynamic link list again, and after running 'myprog' for a third time it printed out "I am not sleeping!" again. To explain why these things happened, the LD_PRELOAD variable is always ignored if Set-UID programs attempt to access it. In the first case of running 'myprog', it is a regular file run by seed. We can see that LD_PRELOAD is not ignored, and the dynamic link list is accessed. In the second case, it is a Set_UID root program ran by seed, so LD_PRELOAD is ignored, and the default library is used. Therefore, we don't see the output from the first run. In the third run, my prog is a Set-UID program run by root, so it searches for user ID and since it's root, it will run the dynamic link list and there is output. On the final run, I created a new user 'user1' and set 'myprog' ownership to user1 and set it as a Set-UID program. I changed LD_PRELOAD to the dynamic link list using user1's account. I then ran myprog from another user and like the second run, the system slept for 1 second, then the program ended. The pictures below show all the commands used, except for the command to create the new user and the command used to set LD_PRELOAD as user1 (I used `useradd user1` and the same export command)

```
root@VM: /home/seed/Documents/lab2
[08/31/20]seed@VM:~$ cd Documents/lab2
[08/31/20]seed@VM:~/.../lab2$ gcc -fPIC -g -c mylib.c
[08/31/20]seed@VM:~/.../lab2$ gcc -shared -o libmylib.so.1.0.1 mylib.o -lc
[08/31/20]seed@VM:~/.../lab2$ export LD_PRELOAD=./libmylib.so.1.0.1
[08/31/20]seed@VM:~/.../lab2$ gcc -o myprog myprog.c
myprog.c: In function 'main':
myprog.c:2:2: warning: implicit declaration of function 'sleep' [-Wimplicit-func
tion-declaration]
  sleep(1);
  ^
[08/31/20]seed@VM:~/.../lab2$ myprog
I am not sleeping!
[08/31/20]seed@VM:~/.../lab2$ sudo chown root myprog
[08/31/20]seed@VM:~/.../lab2$ sudo chmod 4755 myprog
[08/31/20]seed@VM:~/.../lab2$ ls
libmylib.so.1.0.1  myprog.c            task2_parent_vars  task5
ls                 task1_vars_env      task3              task5.c
ls.c               task1_vars_printenv task3.c            task5_vars1
mylib.c            task2               task3_vars         task5_vars2
mylib.o            task2.c             task4              task6
myprog             task2_child_vars    task4.c            task6.c
[08/31/20]seed@VM:~/.../lab2$ myprog
[08/31/20]seed@VM:~/.../lab2$ sudo su root
root@VM:/home/seed/Documents/lab2# export LD_PRELOAD=./libmylib.so.1.0.1
root@VM:/home/seed/Documents/lab2# myprog
I am not sleeping!
root@VM:/home/seed/Documents/lab2# sudo su seed
[08/31/20]seed@VM:~/.../lab2$
```

```
[08/31/20]seed@VM:~/.../lab2$ ls -l myprog
-rwsr-xr-x 1 user1 seed 7348 Aug 31 19:25 myprog
[08/31/20]seed@VM:~/.../lab2$ myprog
[08/31/20]seed@VM:~/.../lab2$
```

Task 8: Invoking External Programs using `system()` versus `execve()`

In this task, we are given a C program that we coded and compiled or order to test some of the differences between the commands system and execve. The first step shows us some of the dangers of using the system command specifically. After compiling the program, I changed its ownership to root and made it a Set-UID program.

```
[08/31/20]seed@VM:~$ cd Documents/lab2
[08/31/20]seed@VM:~/.../lab2$ gcc -o task8 task8.c
[08/31/20]seed@VM:~/.../lab2$ sudo chown root task8
[08/31/20]seed@VM:~/.../lab2$ sudo chmod 4755 task8
```

I created a file called 'temp.txt' in a text editor between these two pictures and the next picture picks up with modifying the temp file to have root permissions. I then ran the program with the temp file as its argument. The program just outputs the file and is seen working. I then added an echo to see if the code could be manipulated. After seeing that Hello had printed to the screen, I re-tried the original command but this time I added a remove command to remove the temp file. After running it, I listed the files in the current directory and the temp file had been removed. This shows how vulnerable the system command is.

```
[08/31/20]seed@VM:~/.../lab2$ sudo chown root:root temp.txt
[08/31/20]seed@VM:~/.../lab2$ task8 temp.txt
This is a temp file to be read then deleted.
[08/31/20]seed@VM:~/.../lab2$ task8 "temp.txt;echo Hello"
This is a temp file to be read then deleted.
Hello
[08/31/20]seed@VM:~/.../lab2$ task8 "temp.txt;rm temp.txt"
This is a temp file to be read then deleted.
[08/31/20]seed@VM:~/.../lab2$ ls
libmylib.so.1.0.1  myprog.c             task2_parent_vars  task5         task8
ls                 task1_vars_env       task3              task5.c       task8.c
ls.c               task1_vars_printenv  task3.c            task5_vars1
mylib.c            task2                task3_vars         task5_vars2
mylib.o            task2.c              task4              task6
myprog             task2_child_vars     task4.c            task6.c
[08/31/20]seed@VM:~/.../lab2$
```

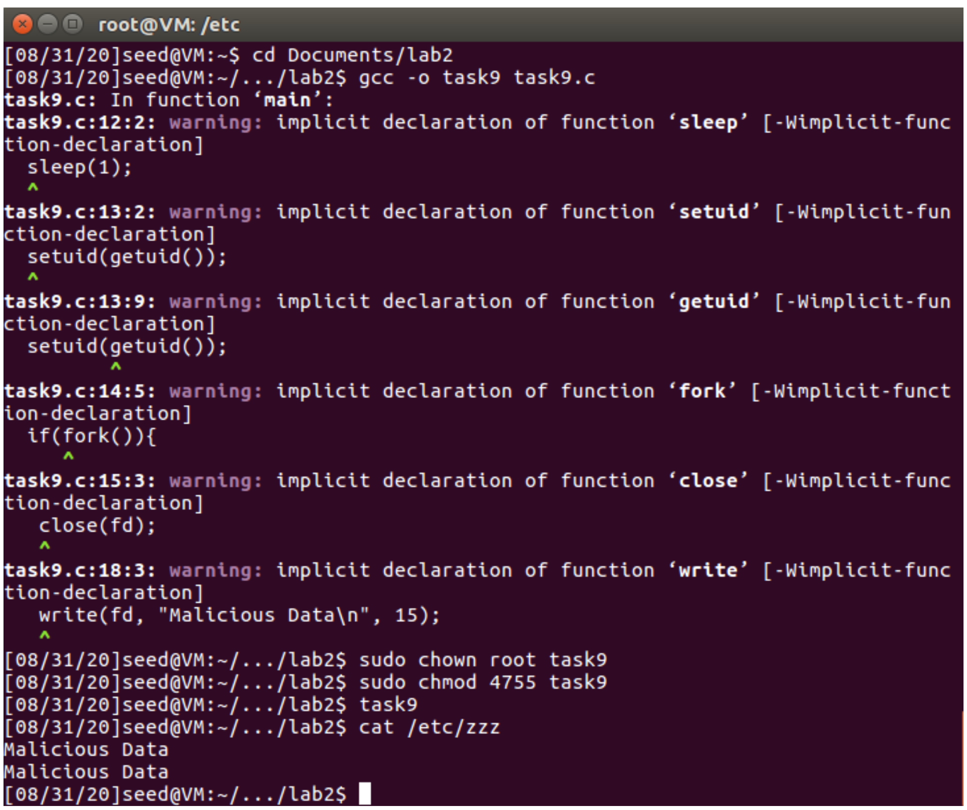
The second step has us modifying the code to use the execve command as opposed to system. After doing so, I compiled the code. I then changed the program ownership to root and set it to be a Set-UID program. I gave my temporary file root permissions, then I attempted the same attacks as from before. I noticed they did not work the same as before.

```
user1@VM: /home/seed/Documents/lab2
[08/31/20]seed@VM:~/.../lab2$ gcc -o task8 task8.c
task8.c: In function 'main':
task8.c:18:5: warning: implicit declaration of function 'execve' [-Wimplicit-fun
ction-declaration]
     execve(v[0], v, NULL);
     ^
[08/31/20]seed@VM:~/.../lab2$ sudo chown root task8
[08/31/20]seed@VM:~/.../lab2$ sudo chmod 4755 task8
[08/31/20]seed@VM:~/.../lab2$ ls
libmylib.so.1.0.1  task1_vars_env       task3.c      task5_vars2
ls                 task1_vars_printenv  task3_vars   task6
ls.c               task2                task4        task6.c
mylib.c            task2.c              task4.c      task8
mylib.o            task2_child_vars     task5        task8.c
myprog             task2_parent_vars    task5.c      temp.txt
myprog.c           task3                task5_vars1
[08/31/20]seed@VM:~/.../lab2$ sudo chmod root:root temp.txt
chmod: invalid mode: 'root:root'
Try 'chmod --help' for more information.
[08/31/20]seed@VM:~/.../lab2$ sudo chown root:root temp.txt
[08/31/20]seed@VM:~/.../lab2$ task8 temp.txt
This is a temp file to be deleted later.
[08/31/20]seed@VM:~/.../lab2$ task8 "temp.txt echo Hello"
/bin/cat: 'temp.txt echo Hello': No such file or directory
[08/31/20]seed@VM:~/.../lab2$ task8 "temp.txt;echo Hello"
/bin/cat: 'temp.txt;echo Hello': No such file or directory
[08/31/20]seed@VM:~/.../lab2$ task8 "temp.txt;rm temp.txt"
/bin/cat: 'temp.txt;rm temp.txt': No such file or directory
[08/31/20]seed@VM:~/.../lab2$
```

This is because, unlike the system command, the execve command takes the entire argument as the filename and does not split it between the "" or ';'. Also, if quotes are forgotten, then the user doesn't gain root permission. This makes for a more secure program.

Task 9: Capability Leaking

In the final task we learn about Capability Leaking. When revoking the privilege, it is one of the most common mistakes. The process may have gained some privileged capabilities when it was still privileged; then when the privilege is downgraded, if the program does not clean up those capabilities, they may still be accessible by the non-privileged process. In other words, although the effective user ID of the process becomes non-privileged, the process is still privileged because it possesses privileged capabilities. I started by coding and compiling the given code, I changed its ownership to root and set it as a Set-UID program. I then created a file named 'zzz' in the etx directory. I ran the program, and it was able to find the file and write malicious data to it. To avoid attacks like this, one would have to close the file checking for the fork. All commands used are pictured below.



```
root@VM: /etc
[08/31/20]seed@VM:~$ cd Documents/lab2
[08/31/20]seed@VM:~/.../lab2$ gcc -o task9 task9.c
task9.c: In function 'main':
task9.c:12:2: warning: implicit declaration of function 'sleep' [-Wimplicit-func
tion-declaration]
  sleep(1);
  ^
task9.c:13:2: warning: implicit declaration of function 'setuid' [-Wimplicit-fun
ction-declaration]
  setuid(getuid());
  ^
task9.c:13:9: warning: implicit declaration of function 'getuid' [-Wimplicit-fun
ction-declaration]
  setuid(getuid());
         ^
task9.c:14:5: warning: implicit declaration of function 'fork' [-Wimplicit-funct
ion-declaration]
  if(fork()){
     ^
task9.c:15:3: warning: implicit declaration of function 'close' [-Wimplicit-func
tion-declaration]
   close(fd);
   ^
task9.c:18:3: warning: implicit declaration of function 'write' [-Wimplicit-func
tion-declaration]
   write(fd, "Malicious Data\n", 15);
   ^
[08/31/20]seed@VM:~/.../lab2$ sudo chown root task9
[08/31/20]seed@VM:~/.../lab2$ sudo chmod 4755 task9
[08/31/20]seed@VM:~/.../lab2$ task9
[08/31/20]seed@VM:~/.../lab2$ cat /etc/zzz
Malicious Data
Malicious Data
[08/31/20]seed@VM:~/.../lab2$
```