Environment Variable and Set-UID Program Lab

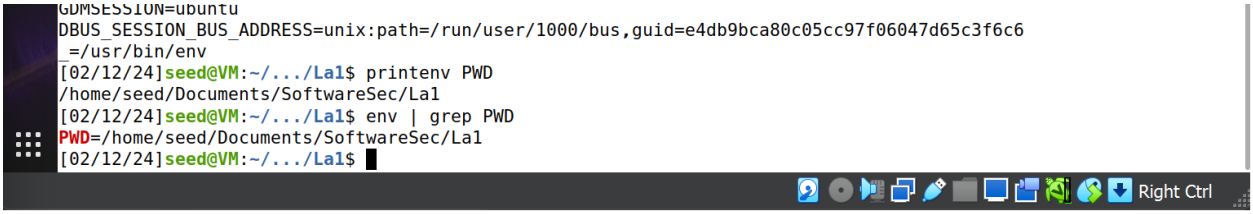
By Sebastian Medina

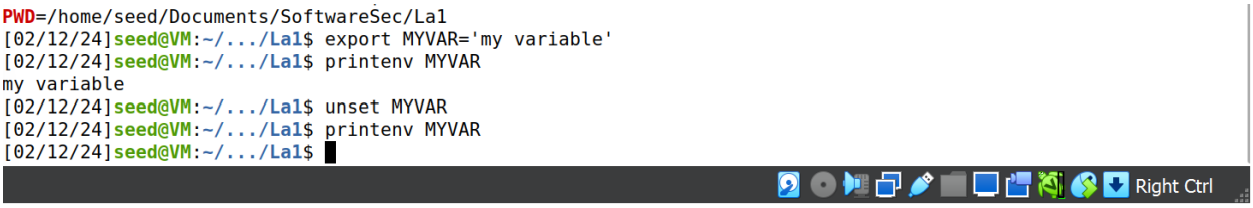
Contents:

1. **Environment Variable and Set-UID SEED Labs**

1.1 Task 1: Manipulating Environment Variables

In this first task we just learned some commands to use to set, unset, and view environment variables. The first step in task one was to use printenv or env command to print and view the environment variable of a certain program. An example of these being used is “printenv PWD” or “env | grep PWD”. The next step teaches you how to set or unset environment variables. You use export command to set an environment variable and unset command to unset an environment variable.

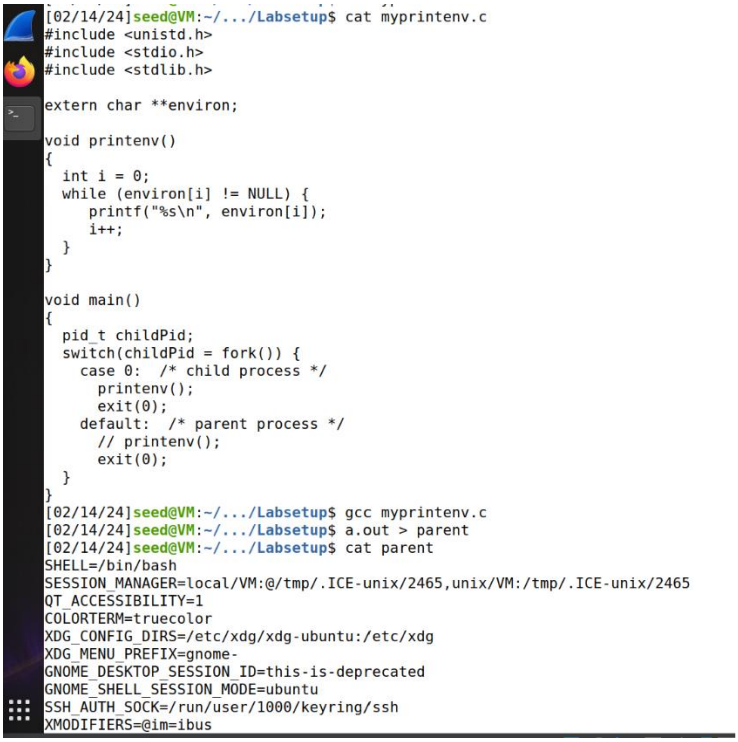




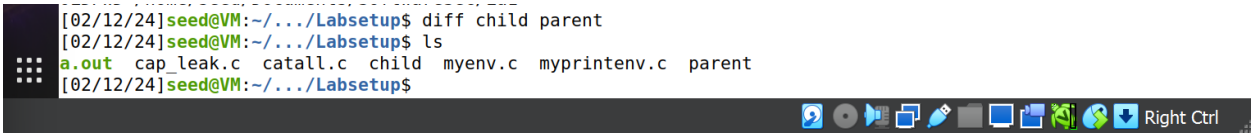
1.2 Task 2: Passing Environment Variables from Parent to Child Process

For the second task it explains how a child process gets its environment variables from the parent process. This task also explains the Unix command fork() and how when you use fork() it creates a new process called the child by duplicating whatever process is being called, the process that is being called is known as the parent. So, while doing this task it tells you to compile to compile the code myprintenv.c which lists all environment variables when run. The first time we compiled it, it ran as a child process, so we save the out put to a file named child.

In the next step of this task, you will have to go into the myprintenv.c code and change it so that this time when the code it runs it outputs the result form the parent program instead of the child program. Once this is done you compile the myprintenv.c and save this output to parent. Once again when you run this program it prints all the environmental variables.



For the last step you take the two file that have the parent output and the child output, so in my case the child and parent file. Then you use the diff command to compare the two files. When I ran this command like so “diff child parent” nothing was outputted because the two files are the same. They are the same because the child is a copy of the parent so nothing should be different.



1.3 Task 3: Environment Variables and execve()

Task 3 teaches you how a new program that is executed via execve() affects environment variables. Some background information about execve() is that this command calls a system call to load a new command and then execute it. With this command no new process is created, instead the new process copies and then overwrites the old process. For the first step all it wants you to do is compile the code named myenv.c and all it does is print out environment variables of the current process. However the code is not suited for this task so after you compile it you must change were it says Null to environ in order for execve() to

print out the all of the environmental variables. If you do not make this change, then none of the environmental variables will print off and you will never receive any of the environment variables.

```
[02/14/24]seed@VM:~/.../Labsetup$ cat myenv.c
#include <unistd.h>

extern char **environ;

int main()
{
  char *argv[2];

  argv[0] = "/usr/bin/env";
  argv[1] = NULL;

  execve("/usr/bin/env", argv, NULL);

  return 0 ;
}

[02/14/24]seed@VM:~/.../Labsetup$ gcc myenv.c -o myenv
[02/14/24]seed@VM:~/.../Labsetup$ ./myenv
[02/14/24]seed@VM:~/.../Labsetup$ vi myenv.c
[02/14/24]seed@VM:~/.../Labsetup$ cat myenv.c
#include <unistd.h>

extern char **environ;

int main()
{
  char *argv[2];

  argv[0] = "/usr/bin/env";
  argv[1] = NULL;

  execve("/usr/bin/env", argv, environ);

  return 0 ;
}

[02/14/24]seed@VM:~/.../Labsetup$ gcc myenv.c -o myenv
[02/14/24]seed@VM:~/.../Labsetup$ ./myenv
SHELL=/bin/bash
SESSION_MANAGER=local/VM:@/tmp/.ICE-unix/2465,unix/VM:/tmp/.ICE-unix/2465
QT_ACCESSIBILITY=1
COLORTERM=truecolor
XDG_CONFIG_DIRS=/etc/xdg/xdg-ubuntu:/etc/xdg
```

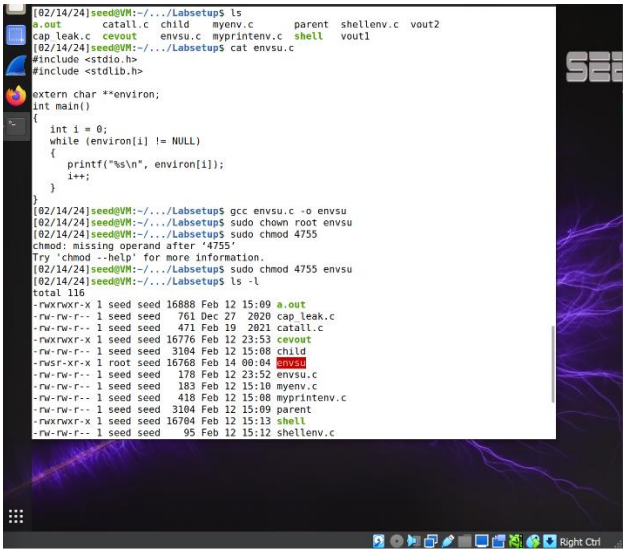
1.4 Task 4: Environment Variables and system()\

Task 4 is very similar to task 3 with one simple change. In this task you study how environmental variables are affected when a new program is executed via the system() function instead of the execve() function. The system() function has the same result as the execve() function how ever system() executes “/bin/sh -c command” witch means it executes a shell command then tells the shell to execute the command. For this task it uses the system() function in a piece of code to list all of the environment variables.

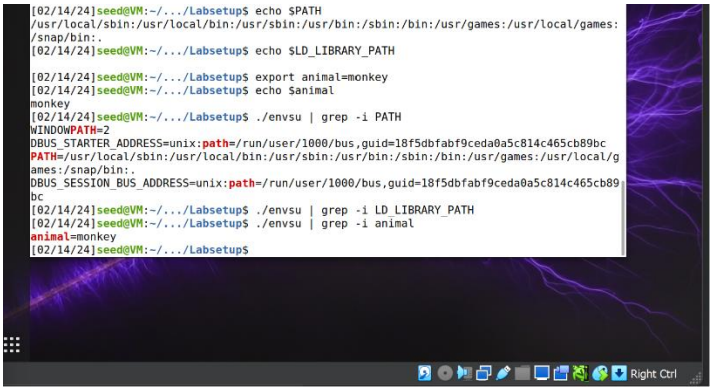
```
}
[02/12/24]seed@VM:~/.../Labsetup$ ls
a.out  cap_leak.c  catall.c  child  envvar  myenv.c  myprintenv.c  parent  vout1  vout2
[02/12/24]seed@VM:~/.../Labsetup$ vi shellenv.c
[02/12/24]seed@VM:~/.../Labsetup$ ls
a.out  cap_leak.c  catall.c  child  envvar  myenv.c  myprintenv.c  parent  shellenv.c  vout1  vout2
[02/12/24]seed@VM:~/.../Labsetup$ cat shellenv.c
#include <stdio.h>
#include <stdlib.h>

int main()
{
   system("/usr/bin/env");
   return 0;
}
[02/12/24]seed@VM:~/.../Labsetup$ gcc shellenv.c -o shell
[02/12/24]seed@VM:~/.../Labsetup$ ./shell
LESSOPEN=| /usr/bin/lesspipe %s
USER=seed
SSH_AGENT_PID=2424
XDG_SESSION_TYPE=x11
SHLVL=1
HOME=/home/seed
OLDPWD=/home/seed/Documents/SoftwareSec/La1
DESKTOP_SESSION=ubuntu
GNOME_SHELL_SESSION_MODE=ubuntu
GTK_MODULES=gail:atk-bridge
MANAGERPID=2218
DBUS_STARTER_BUS_TYPE=session
DBUS_SESSION_BUS_ADDRESS=unix:path=/run/user/1000/bus,guid=e4db9bca80c05cc97f06047d65c3f6c6
COLORTERM=truecolor
IM_CONFIG_PHASE=1
LOGNAME=seed
JOURNAL_STREAM=9:36309
_=./shell
XDG_SESSION_CLASS=user
USERNAME=seed
TERM=xterm-256color
GNOME_DESKTOP_SESSION_ID=this-is-deprecated
WINDOWPATH=2
PATH=/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin:/usr/games:/usr/local/games:/snap/bin:.
SESSION_MANAGER=local/VM:@/tmp/.ICE-unix/2465,unix/VM:/tmp/.ICE-unix/2465
INVOCATION_ID=63cac68d93414c3f85782ce05ff3260e
XDG_MENU_PREFIX=gnome-
GNOME_TERMINAL_SCREEN=/org/gnome/Terminal/screen/b97aeaa7_2ce7_4211_889e_31b8eae186d3
XDG_RUNTIME_DIR=/run/user/1000
DISPLAY=:0
LANG=en_US.UTF-8
XDG_CURRENT_DESKTOP=ubuntu:GNOME
XMODIFIERS=@im=ibus
XDG_SESSION_DESKTOP=ubuntu
XAUTHORITY=/run/user/1000/gdm/Xauthority
```

1.5 Task 5: Environment Variable and Set-UID Programs

For task 5 you will go over Set-UID how it is a very important security mechanism that Unix system have. Whenever there is a program that has Set-UID the program assumes the owners' privileges. With this tool there are many things that can be done however it is risky, and even though the Set-UID programs are influenced by the program logic not users, users can still influence the behaviors via environment variables. For the first two steps of this program, you must write and compile some code that prints out all of the environment variables for the current process. Once you have done that you have take the compile code and change the ownership to root and make it a Set-UID program with the commands, "sudo chown root (file name)", and "sudo chmod 4755 (file name)".

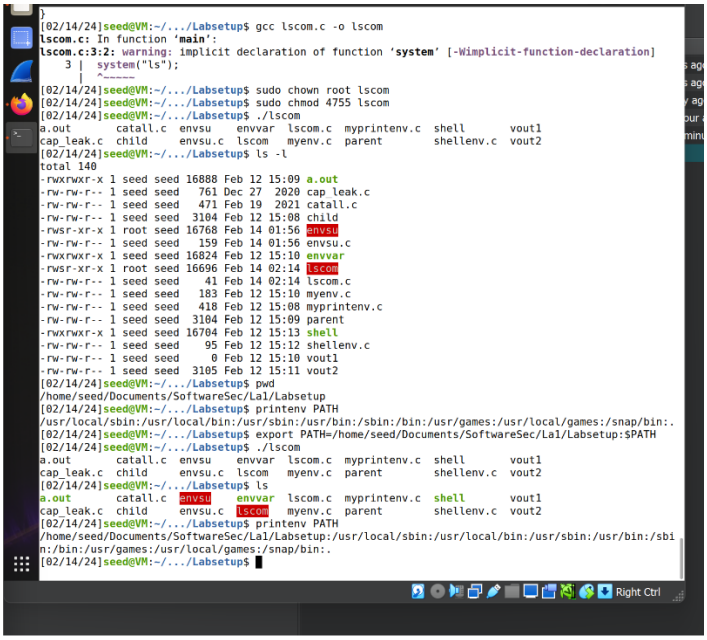


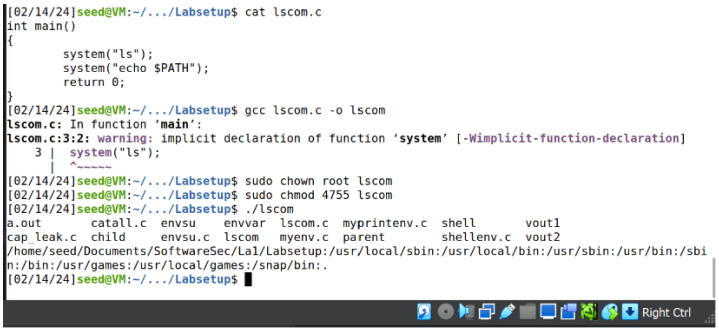
For step 3 you had to set the PATH, LD_LIBRARY_PATH, and ANY_NAME environment variable and then have the program run and see if they can see the environment variable and what you set them to. For this task I had the PATH set to it default, the LD_LIBRARY_PATH to nothing and then I made ANY_NAME into animal and set it to monkey. I then ran the program with grep to see if they can find the environment variables that I set. Just like the program was supposed to it found PATH and the animal environment variables but did not find LD_LUBRARY_PATH because I did not set it to anything.



1.6 Taks 6: The PATH Environment Variable and Set-UID Programs

Within this task it explains how calling system() and having the shell execute the programs within a Set-UID program is dangerous. The reason it is dangerous is because the behavior of the shell can be influenced by environment variables such as PATH. This can cause users to change environment variables to have malicious behavior and attack the system. For explain in this task they have you go into Bash and use the export command to add the file path of the code given in this task that can have malicious code into PATH. Then when you run the code that this task gives you it will trigger PATH and an attack can happen on the system.





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

In task 7 you study how Set-UID programs deal with some of the environment variables. Many environment variables have an influence on the behavior of dynamic loader/linker which can be dangerous because a dynamic loader/linker is a part of an (OS) that loads from ram and links to shared libraries to execute commands at run time or shut down. Being able to influence means that malicious code can be ran in the background or prevent safety guidelines from launching. In this task you write two pieces of code. The code is called

mylib.c, a program that can overwrite the sleep() function. Once you have created this code you compile it as a dynamic link library to build a link library. Then you will set the LD_PRELOAD environment variable to the dynamic link library that you just created. After that is all done you create another program that uses the sleep() function called myprog.c.

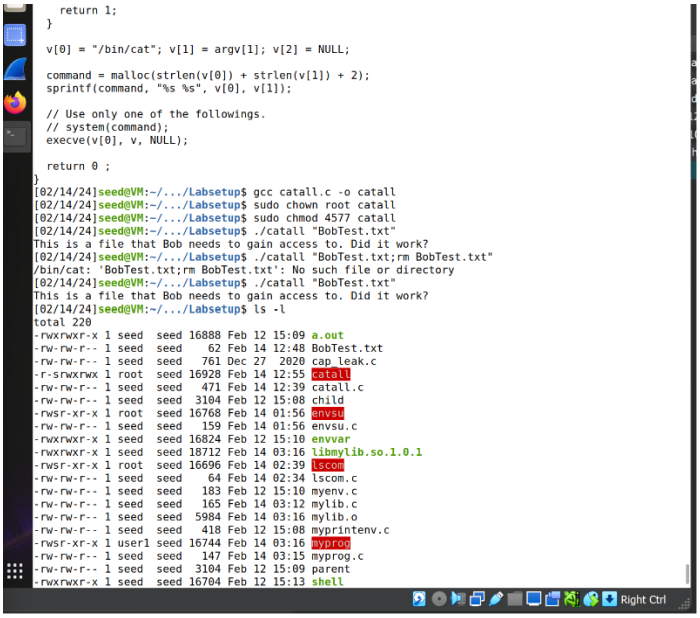
```
[02/14/24]seed@VM:~/.../Labsetup$ cat mylib.c
#include <stdio.h>
void sleep (int s)
{
        /* If this is invoked by a privileged program,
        you can do damages here! */
        printf("Just kidding I am not sleeping!\n");
}
[02/14/24]seed@VM:~/.../Labsetup$ gcc -fPIC -g -c mylib.c
[02/14/24]seed@VM:~/.../Labsetup$ gcc -shared -o libmylib.so.1.0.1 mylib.o -lc
[02/14/24]seed@VM:~/.../Labsetup$ export LD_PRELOAD=./libmylib.so.1.0.1
[02/14/24]seed@VM:~/.../Labsetup$ cat myprog.c
#include <unistd.h>
#include <stdio.h>
int main()
{
        printf("Im going to sleep for 1 sec \n");
        sleep(1);
        printf("I'm awake now\n");
        return 0;
}
[02/14/24]seed@VM:~/.../Labsetup$ gcc myprog.c -o myprog
[02/14/24]seed@VM:~/.../Labsetup$ ./myprog
Im going to sleep for 1 sec
Just kidding I am not sleeping!
I'm awake now
[02/14/24]seed@VM:~/.../Labsetup$
```

For step 3 in this task they have you run the program that has the sleep() function in two other ways than what is demonstrated in the snapshot above. Step 3 has you run the code myprog.c as a regular program and as a normal user and the sleep function well and as myprog.c as a Set-UID root program and a normal user and they will both be overwritten by mylib.c as demonstrated with the snapshot above. Then you have to make myprog.c a Set-UID program and export the LD_PRELOAD environment variable again in the root account and when you run this myprog.c does not get overwritten by mylib.c and outputs what it needs to output because root does not get influced by environment variables as easily. Lasty for the last point you must exit root account and create a user named user1, make user 1 the owner of myprog and also a Set_UID program and run it. This way will also not be overwritten because you are not the owner of the program so myprog will run normally.

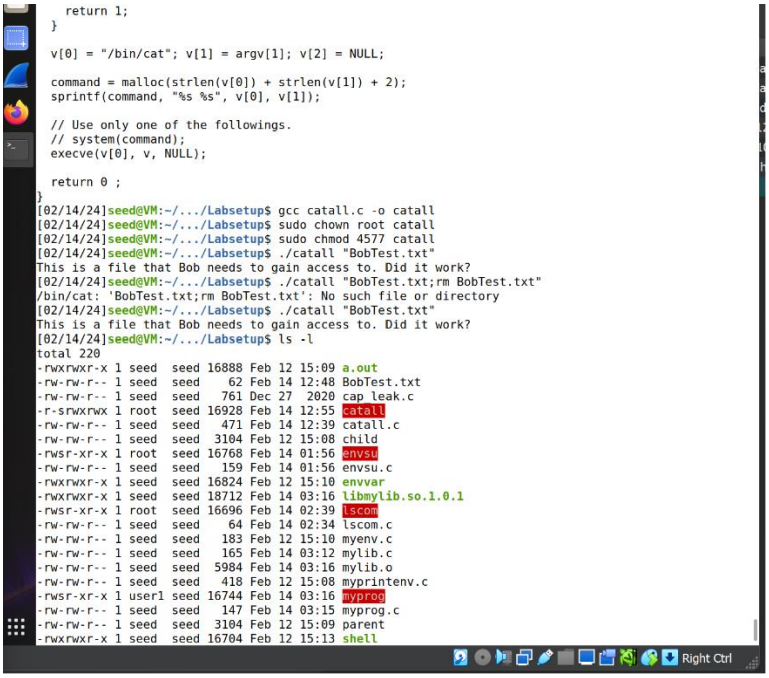
```
[02/14/24]seed@VM:~/.../Labsetup$ sudo su
root@VM:/home/seed/Documents/SoftwareSec/Lal/Labsetup# export LD_PRELOAD=./libmylib.so.1.0.1
root@VM:/home/seed/Documents/SoftwareSec/Lal/Labsetup# ./myprog
Im going to sleep for 1 sec
I'm awake now
root@VM:/home/seed/Documents/SoftwareSec/Lal/Labsetup# exit
exit
[02/14/24]seed@VM:~/.../Labsetup$ sudo chown user1 myprog
[02/14/24]seed@VM:~/.../Labsetup$ sudo chmod 4755 myprog
[02/14/24]seed@VM:~/.../Labsetup$ export LD_PRELOAD=./libmylib.so.1.0.1
[02/14/24]seed@VM:~/.../Labsetup$ ./myprog
Im going to sleep for 1 sec
I'm awake now
[02/14/24]seed@VM:~/.../Labsetup$
```

1.8 Task 8: Invoking External Programs Using system() versus execve()

Task 8 teaches you that even though system() and execve() can both be used to run new programs system() is dangerous to use with Set-UID programs because of the vulnerabilities. In this task you must pretend to be a person named bob who must have the ability to view all the files of a company. The company gives you a program that they think will only allow bob to just view files and that is not the case. However since the program is using system() and using the shell to call the commands it can be manipulated so that bob can remove files from the company when he is only suppose to view them.

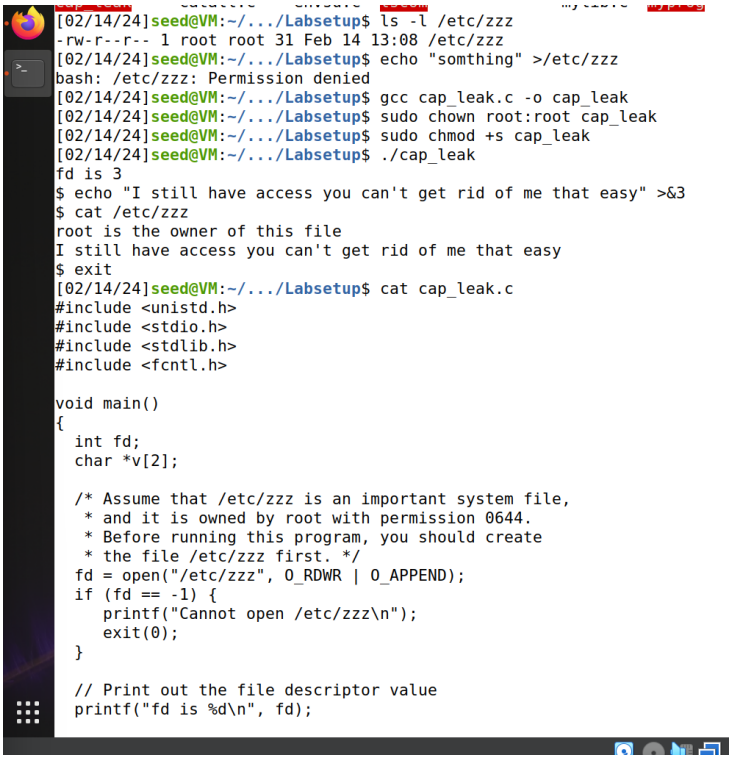


In step two of task 8 you use the same command however you replace where they call the system() function with the execve() command. When you replace system() with execve() when it runs a new program is does not use the shell instead just call the new program witch allows bob to just view the files and nothing else.



1.9 Task 9: Capability Leaking

For the last task of the lab, you learn that when you revoke root privileges a common mistake is capability leaking which means even though a program technically does not have privileges it still does have privileges. In the task it asks you to exploit the capability leaking vulnerability witch I did by making the file root and then used chmod +s to make groups and user inherit the ownership of both and allow me to run the program. Once I ran the program, I was able to add and change /etc/zzz/ directory even though I did not have the privileges to do that.



```
[02/14/24]seed@VM:~/.../Labsetup$ ls -l /etc/zzz
-rw-r--r-- 1 root root 31 Feb 14 13:08 /etc/zzz
[02/14/24]seed@VM:~/.../Labsetup$ echo "somthing" >/etc/zzz
bash: /etc/zzz: Permission denied
[02/14/24]seed@VM:~/.../Labsetup$ gcc cap_leak.c -o cap_leak
[02/14/24]seed@VM:~/.../Labsetup$ sudo chown root:root cap_leak
[02/14/24]seed@VM:~/.../Labsetup$ sudo chmod +s cap_leak
[02/14/24]seed@VM:~/.../Labsetup$ ./cap_leak
fd is 3
$ echo "I still have access you can't get rid of me that easy" >&3
$ cat /etc/zzz
root is the owner of this file
I still have access you can't get rid of me that easy
$ exit
[02/14/24]seed@VM:~/.../Labsetup$ cat cap_leak.c
#include <unistd.h>
#include <stdio.h>
#include <stdlib.h>
#include <fcntl.h>

void main()
{
  int fd;
  char *v[2];

  /* Assume that /etc/zzz is an important system file,
   * and it is owned by root with permission 0644.
   * Before running this program, you should create
   * the file /etc/zzz first. */
  fd = open("/etc/zzz", O_RDWR | O_APPEND);
  if (fd == -1) {
     printf("Cannot open /etc/zzz\n");
     exit(0);
  }

  // Print out the file descriptor value
  printf("fd is %d\n", fd);
```