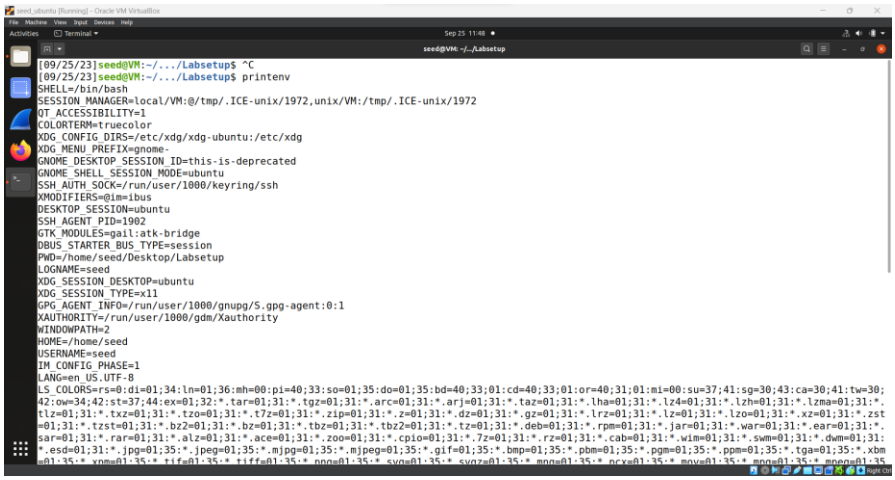
Environment Variable and Set-UID Program Lab

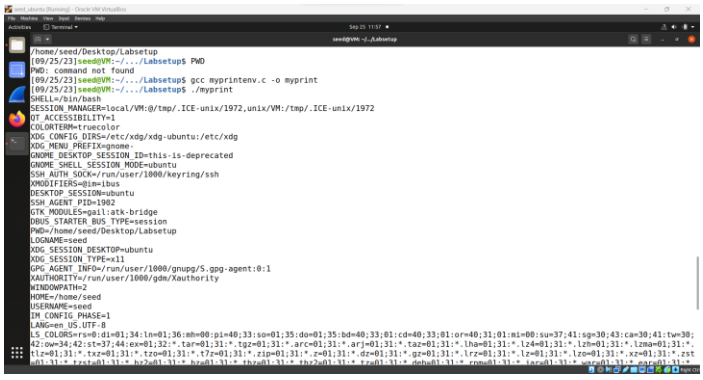
ANANTHANARAYANAN S

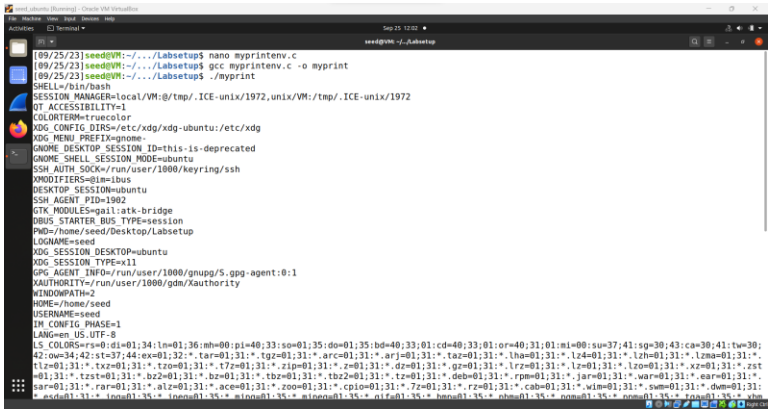
• Use printenv or env command to print out the environment variables



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

There is a difference in the environment variables of child and parent process





Comparing the out

```
_=./myprint1
[09/25/23]seed@VM:~/.../Labsetup$ diff myprint myprint1
Binary files myprint and myprint1 differ
[09/25/23]seed@VM:~/.../Labsetup$ myprint >file1
[09/25/23]seed@VM:~/.../Labsetup$ myprint1>fil2
[09/25/23]seed@VM:~/.../Labsetup$ ls
cap_leak.c  catall.c  fil2  file1  myenv.c  myprint  myprint1  myprintenv.c
[09/25/23]seed@VM:~/.../Labsetup$ diff file1 fil2
49c49
< _=./myprint
---
> _=./myprint1
[09/25/23]seed@VM:~/.../Labsetup$
```

2.3 Task 3: Environment Variables and execve()

The new program must get its environment variables explicitly through the execve call. As we saw from the task, if no environment variables are passed through the call, the program will not have access to them

```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);     ①

  return 0 ;
}
```

```
[09/25/23]seed@VM:~/.../Labsetup$ gcc myenv.c -o myenv
[09/25/23]seed@VM:~/.../Labsetup$ ./myenv
[09/25/23]seed@VM:~/.../Labsetup$ nano myenv.c


Use "fg" to return to nano.

[1]+  Stopped                 nano myenv.c
[09/25/23]seed@VM:~/.../Labsetup$ nano myenv.c
[09/25/23]seed@VM:~/.../Labsetup$ mv myenv myenv1
[09/25/23]seed@VM:~/.../Labsetup$ gcc myenv.c -o myenv
[09/25/23]seed@VM:~/.../Labsetup$ ./myenv
SHELL=/bin/bash
SESSION_MANAGER=local/VM:@/tmp/.ICE-unix/1972,unix/VM:/tmp/.ICE-unix/1972
QT_ACCESSIBILITY=1
COLORTERM=truecolor
XDG_CONFIG_DIRS=/etc/xdg/xdg-ubuntu:/etc/xdg
XDG_MENU_PREFIX=gnome-
GNOME_DESKTOP_SESSION_ID=this-is-deprecated
GNOME_SHELL_SESSION_MODE=ubuntu
SSH_AUTH_SOCK=/run/user/1000/keyring/ssh
XMODIFIERS=@im=ibus
DESKTOP_SESSION=ubuntu
SSH_AGENT_PID=1902
GTK_MODULES=gail:atk-bridge
DBUS_STARTER_BUS_TYPE=session
PWD=/home/seed/Desktop/Labsetup
LOGNAME=seed
XDG_SESSION_DESKTOP=ubuntu
```

Task 4: Environment Variables and system()

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

int main()
{
  system("/usr/bin/env");
  return 0 ;
}
```

```
[09/25/23]seed@VM:~/.../Labsetup$ nano en.c
[09/25/23]seed@VM:~/.../Labsetup$ ls
cap_leak.c  catall.c  en.c  fil2  file1  myenv  myenv1  myenv.c  myprint  myprint1  myprintenv.c
[09/25/23]seed@VM:~/.../Labsetup$ gcc en.c -o en
[09/25/23]seed@VM:~/.../Labsetup$ ./en
SHELL=/bin/bash
SESSION_MANAGER=local/VM:@/tmp/.ICE-unix/1972,unix/VM:/tmp/.ICE-unix/1972
QT_ACCESSIBILITY=1
COLORTERM=truecolor
XDG_CONFIG_DIRS=/etc/xdg/xdg-ubuntu:/etc/xdg
XDG_MENU_PREFIX=gnome-
GNOME_DESKTOP_SESSION_ID=this-is-deprecated
GNOME_SHELL_SESSION_MODE=ubuntu
SSH_AUTH_SOCK=/run/user/1000/keyring/ssh
XMODIFIERS=@im=ibus
DESKTOP_SESSION=ubuntu
SSH_AGENT_PID=1902
GTK_MODULES=gail:atk-bridge
DBUS_STARTER_BUS_TYPE=session
PWD=/home/seed/Desktop/Labsetup
LOGNAME=seed
XDG_SESSION_DESKTOP=ubuntu
XDG_SESSION_TYPE=x11
GPG_AGENT_INFO=/run/user/1000/gnupg/S.gpg-agent:0:1
XAUTHORITY=/run/user/1000/gdm/Xauthority
```

By using the System() call, the environment variables are passed to the program because it uses exec1 internally, which provides the environment variables to execve automatically.

Task 5: Environment Variable and Set-UID Programs

In your shell (you need to be in a normal user account, not the root account), use the export command to set the following environment variables (they may have already exist):

- PATH
- LD LIBRARY PATH
- ANY NAME (this is an environment variable defined by you, so pick whatever name you want)

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

extern char **environ;
int main()
{
  int i = 0;
  while (environ[i] != NULL) {
    printf("%s\n", environ[i]);
    i++;
  }
}
```

```
[09/25/23]seed@VM:~/.../Labsetup$ nano foo.c
[09/25/23]seed@VM:~/.../Labsetup$ gcc foo.c -o foo
[09/25/23]seed@VM:~/.../Labsetup$ sudo chown root foo
[09/25/23]seed@VM:~/.../Labsetup$ sudo chmod 4755 foo
[09/25/23]seed@VM:~/.../Labsetup$ ls -l foo
-rwsr-xr-x 1 root seed 16768 Sep 25 12:54 foo
[09/25/23]seed@VM:~/.../Labsetup$ export PATH="/bin:/usr/bin"
[09/25/23]seed@VM:~/.../Labsetup$ printenv PATH
/bin:/usr/bin
[09/25/23]seed@VM:~/.../Labsetup$ export LD_LIBRARY_PATH="Mylibrarypath"
[09/25/23]seed@VM:~/.../Labsetup$ printenv LD_LIBRARY_PATH
Mylibrarypath
[09/25/23]seed@VM:~/.../Labsetup$  export MY_VAR_ANY="djksdfkjsdfkjhsdfkjh"
[09/25/23]seed@VM:~/.../Labsetup$  ./foo | grep "MY_VAR_ANY\|LD_LIBRARY_PATH\|PATH"
WINDOWPATH=2
MY_VAR_ANY=djksdfkjsdfkjhsdfkjh
PATH=/bin:/usr/bin
```

The environment variables are passed to the Set-UID child process. the variable LD_LIBRARY_PATH doesn't seem to have been passed

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

```c
int main()
{
  system("ls");
  return 0;
}
```

```
[09/25/23]seed@VM:~/.../Labsetup$ nano task.c
[09/25/23]seed@VM:~/.../Labsetup$ gcc task.c -o task
task.c: In function 'main':
task.c:3:1: warning: implicit declaration of function 'system' [-Wimplicit-function-declaration]
    3 | system("ls");
      | ^~~~~~
[09/25/23]seed@VM:~/.../Labsetup$ ./task
cap_leak.c  catall.c  en  en.c  fil2  file1  foo  foo.c  myenv  myenv1  myenv.c  myprint  myprint1  myprintenv.c  task  task.c
[09/25/23]seed@VM:~/.../Labsetup$ sudo chown root task
[09/25/23]seed@VM:~/.../Labsetup$ sudo chmod 4755 task
[09/25/23]seed@VM:~/.../Labsetup$ ls -l task
-rwsr-xr-x 1 root seed 16696 Sep 25 13:28 task
[09/25/23]seed@VM:~/.../Labsetup$ sudo ln -sf /bin/zsh /bin/sh
[09/25/23]seed@VM:~/.../Labsetup$ export PATH=/home/seed:$PATH
[09/25/23]seed@VM:~/.../Labsetup$ ./task
VM# exit
[09/25/23]seed@VM:~/.../Labsetup$ ls
ls: no such option: color=auto
```

By creating an executable file called "ls" in the /home/seed directory, and adding that directory to the PATH environment variable, we were able to make the Set-UID process run that executable instead of the "real" ls

Task 7: The LD PRELOAD Environment Variable and Set-UID Program

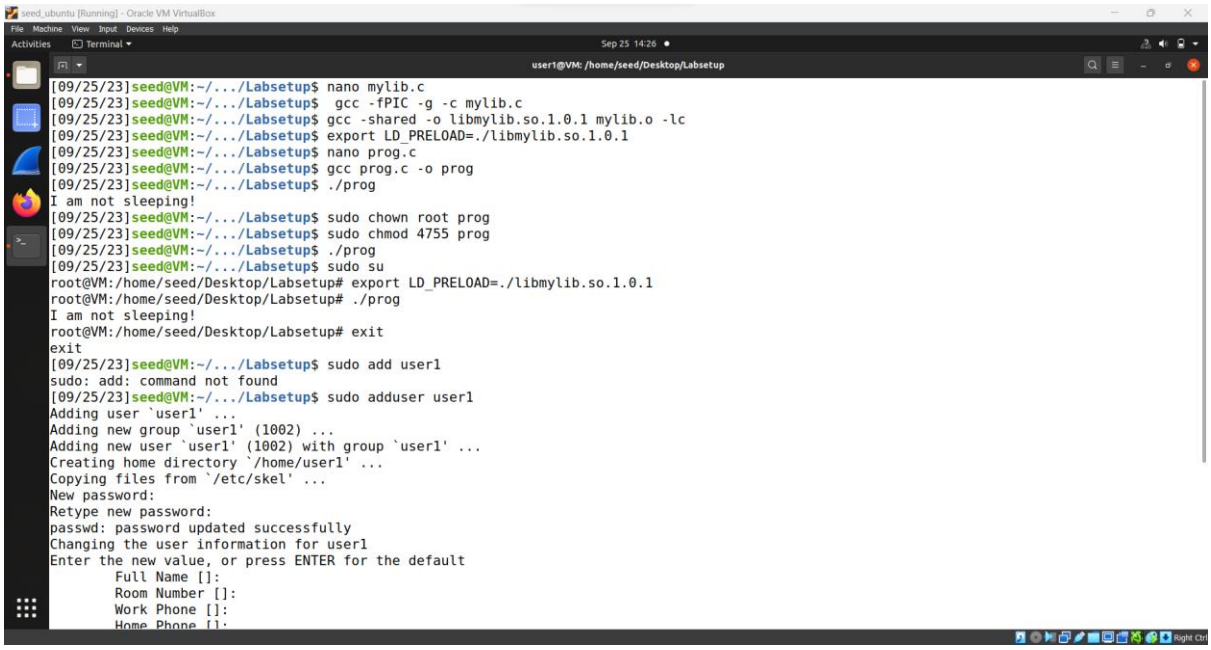
```c
#include <stdio.h>
void sleep (int s)
{
  /* If this is invoked by a privileged program,
     you can do damages here!  */
  printf("I am not sleeping!\n");
}
```

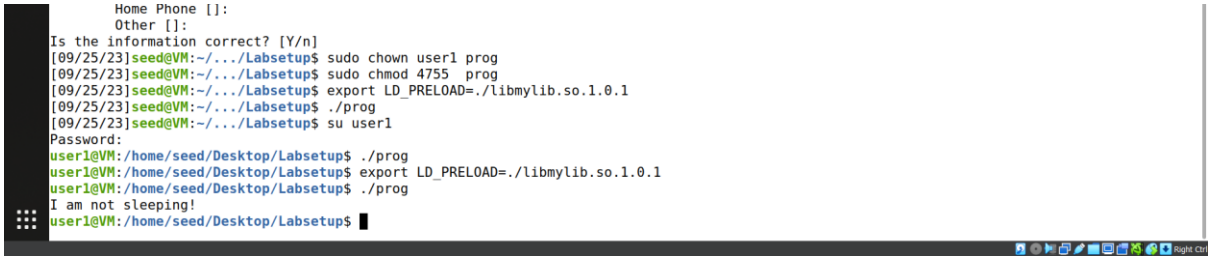
```c
/* myprog.c */
#include <unistd.h>
int main()
{
  sleep(1);
  return 0;
}
```

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

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

Make myprog a Set-UID root program, export the LD_PRELOAD environment variable again in the root account and run it.





Switching to new user, exporting library and running the program

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

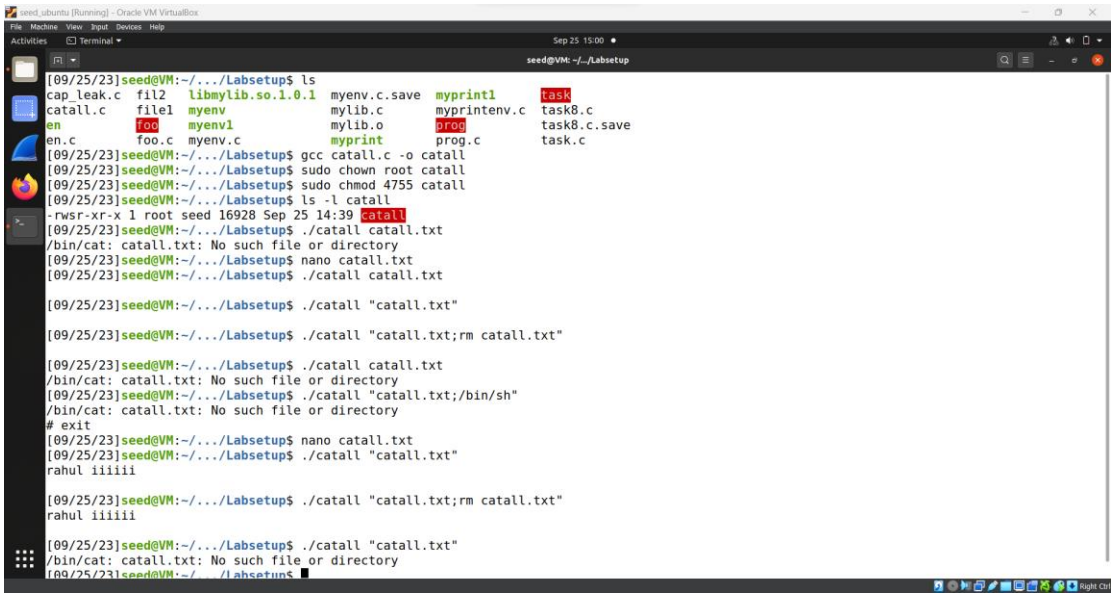
```
int main(int argc, char *argv[])
{
  char *v[3];
  char *command;

  if(argc < 2) {
    printf("Please type a file name.\n");
    return 1;
  }

  v[0] = "/bin/cat"; v[1] = argv[1]; v[2] = NULL;
  command = malloc(strlen(v[0]) + strlen(v[1]) + 2);
  sprintf(command, "%s %s", v[0], v[1]);

  // Use only one of the followings.
  system(command);
  // execve(v[0], v, NULL);
```

1:



2.

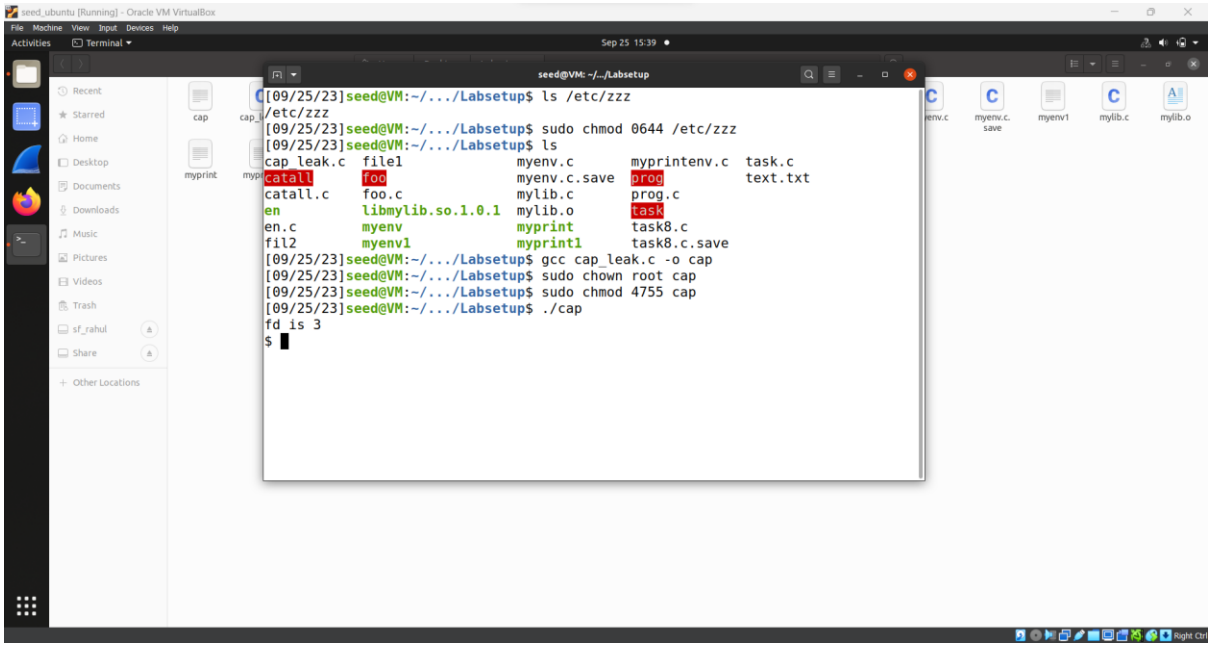
```
[09/25/23]seed@VM:~/.../Labsetup$ nano  catall.c
[09/25/23]seed@VM:~/.../Labsetup$ nano  catall.c
[09/25/23]seed@VM:~/.../Labsetup$ gcc catall.c -o catall
[09/25/23]seed@VM:~/.../Labsetup$ sudo chown root catall
[09/25/23]seed@VM:~/.../Labsetup$ sudo chmod 4755 catall
[09/25/23]seed@VM:~/.../Labsetup$ nano text.txt
[09/25/23]seed@VM:~/.../Labsetup$ ./catall "text.txt"
rahul king

[09/25/23]seed@VM:~/.../Labsetup$ ./catall "text.txt;rm text.txt"
/bin/cat: 'text.txt;rm text.txt': No such file or directory
[09/25/23]seed@VM:~/.../Labsetup$ ./catall "text.txt"
rahul king

[09/25/23]seed@VM:~/.../Labsetup$
```

The problem here is the system call inside the program which does not separate the command and user input. The user input is eventually treated as a command instead of data/document name.

Task 9: Capability Leaking



We run the program and again see the content of the zzz file, and we see that the file content is modified. This happens because even though in the program, we dropped the privileges, we did not close the file at the right time and hence the file was still running with privileged permissions that allowed the data in the file to be modified, even without the right permissions. Here, after calling fork, the control is passed to the child process and hence the malicious user is successful in modifying the content of a privileged file. This shows that it is important to close the file descriptor after dropping privileges, in order for it to have the appropriate permissions.