TASK

Task1

查看指定环境变量`PWD`

```
_=/usr/bin/env
[10/31/22]seed@VM:~/.../Labsetup$ env | grep PWD
PWD=/home/seed/Desktop/share/Labsetup
[10/31/22]seed@VM:~/.../Labsetup$ echo $PWD
/home/seed/Desktop/share/Labsetup
```

export设置环境变量

```
[10/31/22]seed@VM:~/.../Labsetup$ export MYTEST=mytest
[10/31/22]seed@VM:~/.../Labsetup$ echo $MYTEST
mytest
[10/31/22]seed@VM:~/.../Labsetup$
```

unset取消环境变量

```
[10/31/22]seed@VM:~/.../Labsetup$ unset MYTEST
[10/31/22]seed@VM:~/.../Labsetup$ echo $MYTEST

[10/31/22]seed@VM:~/.../Labsetup$
```

Task2

1. 编译运行：`gcc myprintenv.c`
2. 将运行结果保存到文件中：`a.out > child`
3. 修改myprintenv.c文件，注释子进程中的printenv函数，并取消父进程中的注释，再次编译，将结果保存在另一个文件中：`a.out > parent`
4. 比较两个文件`diff child parent`，发现是相同的，即子进程完全继承父进程的环境变量。

```
[10/31/22]seed@VM:~/.../Labsetup$ a.out > child
[10/31/22]seed@VM:~/.../Labsetup$ vim myprintenv.c
[10/31/22]seed@VM:~/.../Labsetup$ gcc myprintenv.c
[10/31/22]seed@VM:~/.../Labsetup$ a.out > parent
[10/31/22]seed@VM:~/.../Labsetup$ diff child parent
[10/31/22]seed@VM:~/.../Labsetup$
```

Task3

myenv.c

```
execve("/usr/bin/env", argv, NULL);//修改前
execve("/usr/bin/env", argv, environ);//修改后
```

```
[10/31/22]seed@VM:~/.../Labsetup$ gcc myenv.c -o myenv1
[10/31/22]seed@VM:~/.../Labsetup$ vim myenv.c
[10/31/22]seed@VM:~/.../Labsetup$ gcc myenv.c -o myenv2
[10/31/22]seed@VM:~/.../Labsetup$ ./myenv1
[10/31/22]seed@VM:~/.../Labsetup$ ./mtenv2
bash: ./mtenv2: No such file or directory
[10/31/22]seed@VM:~/.../Labsetup$ ./myenv2
SHELL=/bin/bash
SESSION_MANAGER=local/VM:@/tmp/.ICE-unix/1910,unix/VM:/tmp/.ICE-unix/1910
QT_ACCESSIBILITY=1
COLORTERM=truecolor
```

修改前没有输出，修改后能输出环境变量。原因：

1. `extern char **environ;`获取环境变量表；
2. execve的第三个参数就是传入环境变量的位置；

Task4

编译运行，能输出环境变量。

```
[10/31/22]seed@VM:~/.../Labsetup$ gcc sys.c -o sys
[11/07/22]seed@VM:~/.../Labsetup$ ./sys
LESSOPEN=| /usr/bin/lesspipe %s
USER=seed
SSH_AGENT_PID=1873
XDG_SESSION_TYPE=x11
SHLVL=1
HOME=/home/seed
DESKTOP_SESSION=ubuntu
```

`system('/usr/bin/env')`调用execl()执行`/bin/sh`，execl()调用execve()并将环境变量传给它。

Task5

编译得到foo后，修改权限：

```
sudo chown root foo
sudo chmod 4755 foo
```

以普通用户修改环境变量`ANY_NAME`，执行foo后可以看到修改后的`ANY_NAME`，即父进程中设置的环境变量能进入到setuid(foo)的子进程中。

```
[11/07/22]seed@VM:~/.../Labsetup$ export ANY_NAME=setuidtest
[11/07/22]seed@VM:~/.../Labsetup$ ./foo
SHELL=/bin/bash
SESSION_MANAGER=local/VM:@/tmp/.ICE-unix/1910,unix/VM:/tmp/.ICE-unix/1910
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
ANY_NAME=setuidtest
GNOME_SHELL_SESSION_MODE=ubuntu
```

Task6

- 在原有PATH变量首部添加`/home/seed`路径：`export PATH=/home/seed:$PATH`，寻找命令时会优先选择靠前的。
- 关闭保护措施：`sudo ln -sf /bin/zsh /bin/sh`
- 新建两个文件并编译。

```
//ls.c 真的调用ls命令，编译为ls，修改权限类似task5
int main()
{
    system("ls");
    return 0;
}
//fakels.c 恶意代码，编译为ls(放在/home/seed路径下)
int main(){
    printf("you are in fake ls!\n");
}
```

- 执行真的ls程序时，system调用首选了/home/seed下的ls程序，即调用了恶意代码。
```
[11/09/22]seed@VM:~/.../Labsetup$ ./ls
you are in fake ls!
```

Task7

```
gcc -fPIC -g -c mylib.c
gcc -shared -o libmylib.so.1.0.1 mylib.o -lc
export LD_PRELOAD=./libmylib.so.1.0.1
```

- 普通程序+普通用户：seed用户已经拥有LD_PRELOAD环境变量
```
[11/09/22]seed@VM:~/.../Labsetup$ gcc myprog.c -o pro1
[11/09/22]seed@VM:~/.../Labsetup$ ./pro1
I am not sleeping!
```
- setuid_root程序+普通用户：sleep 1秒；(进程ID与当前用户ID不一致)
- setuid_root程序+LD_PRELOAD环境(root)：(`sudo /bin/bash`进入root)
```
root@VM:/home/seed/Documents/Labsetup# ./pro2
I am not sleeping!
```
- setuid_user1程序+LD_PRELOAD环境(user2)：
    - seed用户编译程序后将权限改为sean用户；
    - seed用户导入环境变量后执行，睡眠1s。

结论：LD_PRELOAD环境变量让链接器将sleep()函数和用户的代码链接起来，将该环境变量加入新的共享库，程序会调用用户的sleep()函数，输出I am not sleeping! 但是当进程的真实用户ID和有效用户ID不一致时，进程忽略LD_PRELOAD环境变量。因此第二个和第四个的sleep没有被替换。

Task8

system()

1. 创建test.txt文件，修改权限

```
sudo chown root test.txt # 修改为root用户所有
sudo chmod 600 test.txt # 修改权限为所属用户能读写
```

2. 编译catall.c，此时仍不能读取test.txt，修改权限如task5后可以读取。

```
[11/11/22]seed@VM:~/.../Labsetup$ cat test.txt
cat: test.txt: Permission denied
[11/11/22]seed@VM:~/.../Labsetup$ ./cat test.txt
/bin/cat: test.txt: Permission denied
[11/11/22]seed@VM:~/.../Labsetup$ sudo chown root cat
[11/11/22]seed@VM:~/.../Labsetup$ sudo chmod 4755 cat
[11/11/22]seed@VM:~/.../Labsetup$ ./cat test.txt
iiiiiiiiiiiii
```

3. 执行`./cat "test.txt;rm test.txt"`，会先输出内容再删除该文件。(以分号分割两条命令)

execve()

1. 创建test.txt步骤类似；
2. 注释掉system函数，取消execve的注释，编译，修改cat权限，仍然没有权限读取test.txt；
3. 构建删除语句，execve无法识别为两条命令

```
[11/11/22]seed@VM:~/.../Labsetup$ ./cat test.txt
/bin/cat: test.txt: Permission denied
[11/11/22]seed@VM:~/.../Labsetup$ ./cat "test.txt;rm test.txt"
/bin/cat: 'test.txt;rm test.txt': No such file or directory
```

可以看出execve()比system()更安全。

Task9

1. 创建文件：`sudo vim /etc/zzz`

2. 注释execve函数，添加代码：

```
if(fork()){//创建进程
    close(fd);
    exit(0);
}else{//在子进程中，向已经打开的文件描述符中写入字符
    write(fd,"\ntask9 attack\n",15);
    close(fd);
}
```

3. 编译，修改权限，运行，发现指定字符串被写入文件

```
[11/11/22]seed@VM:~/.../Labsetup$ ll /etc/zzz
-rw-r--r-- 1 root root 15 Nov 11 07:10 /etc/zzz
[11/11/22]seed@VM:~/.../Labsetup$ cat /etc/zzz
iiiiiiiiiiiiii
[11/11/22]seed@VM:~/.../Labsetup$ leak
fd is 3
[11/11/22]seed@VM:~/.../Labsetup$ cat /etc/zzz
iiiiiiiiiiiiii

task9 attack
```

在此实验过程中，通过`setuid(getuid());`降低了程序的权限，但是由于文件已经被打开，降级后的进程仍可访问该文件。