第一周

1. TASK 1

```
➜  ~
➜  ~ # 1. printenv
➜  ~ printenv SHELL
/usr/bin/zsh
➜  ~
➜  ~ # 2.env
➜  ~ env | grep SHELL
SHELL=/usr/bin/zsh
➜  ~
➜  ~ # 3.export
➜  ~ export STUDENT=ZSC
➜  ~ printenv STUDENT
ZSC
➜  ~
➜  ~ # 4.unset
➜  ~ unset STUDENT
➜  ~ printenv STUDENT
```

2. TASK 2

为了方便说明，创建两个c文件，分别命名为foo1.c和foo2.c，foo1.c在子进程中调用printenv()，foo2.c在父进程中调用printenv():

```
➜  temp diff foo1.c foo2.c
20c20
<       printenv();
---
>       //printenv();
23c23
<         //printenv();
---
>         printenv();
```

分别编译运行，并将结果重定向至文本文件：

```
➜  temp gcc foo1.c -o foo1
➜  temp gcc foo2.c -o foo2
➜  temp ./foo1 > foo1.output
➜  temp ./foo2 > foo2.output
```

查看运行结果，发现foo1与foo2的输出完全一致（不考虑`$_`，为了方便说明，分别编译了foo1和foo2，必然导致`$_`不一致）

```
➜  temp diff foo1.output foo2.output
33c33
< _=/root/temp/./foo1
---
> _=/root/temp/./foo2
```

得出结论：通过fork()创建的子进程会继承父进程的环境变量。

3. TASK 3

3.1. 现象

当源文件中指定行的内容是`execve("/usr/bin/env", argv, NULL);`时，无输出；当源文件中指定行是`execve("/usr/bin/env", argv, environ);`时，输出了当前会话的环境变量。

3.2. 结论及分析

在Linux C中，environ是一个外部变量，存储着当前进程的环境变量。在本task中，创建子进程的方式是通过execve()调用，execve()的函数原型如下：

```
int execve(const char *pathname, char *const argv[], char *const
envp[]);
```

由函数原型可知，通过execve()创建的子进程，其环境变量由execve()的第三个参数决定。
第一次调用execve()时，第三个参数为NULL，因此无输出，第二次调用execve()时，第三个参数为父进程的environ，因此输出了父进程的环境变量。

4. Task 4: Environment Variables and system()

4.1. 现象

运行后，输出了当前会话的环境变量。

4.2. 结论及分析

system()会使用fork()创建子进程，并在子进程中调用execl("/bin/sh", "sh", "-c", command, (char *) NULL);执行指定的command，在这一过程中，父进程的环境变量会被传递给子进程。

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

5.1. 过程

```
ubuntu@VM-0-6-ubuntu:/root/temp$ # 切换为非root用户
ubuntu@VM-0-6-ubuntu:/root/temp$ whoami
ubuntu
ubuntu@VM-0-6-ubuntu:/root/temp$ # 修改环境变量
ubuntu@VM-0-6-ubuntu:/root/temp$ export PATH=/temp:$PATH
ubuntu@VM-0-6-ubuntu:/root/temp$ export LD_LIBRARY_PATH=/temp:$LD_LIBRARY_PATH
ubuntu@VM-0-6-ubuntu:/root/temp$ export STUDENT=ZSC
ubuntu@VM-0-6-ubuntu:/root/temp$
ubuntu@VM-0-6-ubuntu:/root/temp$ # 运行set-uid程序
ubuntu@VM-0-6-ubuntu:/root/temp$ ./t5 | grep PATH
PATH=/temp:/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin:/usr/games:/usr/local/games
ubuntu@VM-0-6-ubuntu:/root/temp$ ./t5 | grep LD_LIBRARY_PATH
ubuntu@VM-0-6-ubuntu:/root/temp$ ./t5 | grep STUDENT
STUDENT=ZSC
```

5.2. 现象

通过非特权用户设定的环境变量后，以同一个非特权用户运行的set-uid程序无法打印LD_LIBRARY_PATH，可以打印PATH和自定义变量。

5.3. 结论及分析

当运行程序的有效ID和真实ID不同时，环境变量中的LD_LIBRARY_PATH、LD_PRELOAD等将被忽略。

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

6.1. 过程

在/tmp目录下新建如下文件，命名为ls并赋予执行权限：

```
#!/bin/zsh
/usr/lib/klibc/bin/ls /root
```

修改PATH如下：

```
PATH=/temp:/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin:/usr/games:/usr/local/games
```

运行set-uid程序

6.2. 现象

```
ubuntu@VM-0-6-ubuntu:/root/temp$ ls
/root: Permission denied
```

bash

```
VM-0-6-ubuntu% ./t6
-rwxr-xr-x  1    0    0     13032 test
-rw-r--r--  1    0    0 108626482 朝代歌：一首菊花台唱完中华千年历史.mp4
```

zsh

可以运行自己的程序，这个程序具有root权限，但是这一权限会被Bash的安全措施阻拦。

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

7.1. 现象

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

- Make myprog a Set-UID root program, and run it as a normal user.
  - 睡眠
- Make myprog a Set-UID root program, export the `LD_PRELOAD` environment variable again in the root account and run it.
  - 不睡眠
- 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.
  - 睡眠

7.2. 结论及分析

同TAKS 5，当运行程序的有效ID和真实ID不同时，环境变量中的`LD_LIBRARY_PATH`、`LD_PRELOAD`等将被忽略。

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

8.1. system

可以，方法如下：

```
VM-0-6-ubuntu% head test
USER=root
LOGNAME=root
HOME=/root
PATH=/root/.nvm/versions/node/v13.2.0/bin:/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin:/usr/games:/usr/local/games
MAIL=/var/mail/root
SHELL=/usr/bin/zsh
SSH_CLIENT=221.179.160.96 2780 22
SSH_CONNECTION=221.179.160.96 2780 172.17.0.6 22
SSH_TTY=/dev/pts/0
TERM=xterm
VM-0-6-ubuntu% ./t8 "/dev/null > test"
VM-0-6-ubuntu% head test
VM-0-6-ubuntu%
```

8.2. execve

相同的方法无法生效

8.3. 结论及分析

由于`>`是一个独立的命令，在`/bin/cat /dev/null > test`这行命令中set-uid只能对`/bin/cat /dev/null`操作提权，而无法将`> test`操作提权。
考虑到system()会调用`execl("/bin/sh", "sh", "-c", command, (char *) NULL);`，由于存在`-c`参数，可以将`/dev/null > test`用引号引起，作为整体进行提权。
但是exceve()的函数原型中不包含`-c`参数，故以上方法无法生效。

9. Task 9: Capability Leaking

9.1. 现象

文件/etc/zzz确实被成功修改。

9.2. 结论及分析

由于程序在revoke权限时，存在一个尚未关闭的文件标识符，因此在revoke权限后，该程序对此文件仍然保有revoke之前的权限。