Name: Soundarya Poovaiah Kookanda

Environment Variable and Set-UID Program Lab

TASK 1: MANIPULATING ENVIRONMENT VARIABLES

Before starting, checking for the default shell configured for my account as shown below

```
[02/17/25]seed@VM:~/.../Labsetup$ cat /etc/passwd
root:x:0:0:root:/root:/bin/bash
daemon:x:1:1:daemon:/usr/sbin:/usr/sbin/nologin
bin:x:2:2:bin:/bin:/usr/sbin/nologin
sys:x:3:3:sys:/dev:/usr/sbin/nologin
sync:x:4:65534:sync:/bin:/bin/sync
games:x:5:60:games:/usr/games:/usr/sbin/nologin
man:x:6:12:man:/var/cache/man:/usr/sbin/nologin
lp:x:7:7:lp:/var/spool/lpd:/usr/sbin/nologin
mail:x:8:8:mail:/var/mail:/usr/sbin/nologin
news:x:9:9:news:/var/spool/news:/usr/sbin/nologin
uucp:x:10:10:uucp:/var/spool/uucp:/usr/sbin/nologin
proxy:x:13:13:proxy:/bin:/usr/sbin/nologin
www-data:x:33:33:www-data:/var/www:/usr/sbin/nologin
backup:x:34:34:backup:/var/backups:/usr/sbin/nologin
list:x:38:38:Mailing List Manager:/var/list:/usr/sbin/nologin
irc:x:39:39:ircd:/var/run/ircd:/usr/sbin/nologin
gnats:x:41:41:Gnats Bug-Reporting System (admin):/var/lib/gnats:/usr/sbin/nologin
nobody:x:65534:65534:nobody:/nonexistent:/usr/sbin/nologin
systemd-network:x:100:102:systemd Network Management,,,:/run/systemd:/usr/sbin/nologin
systemd-resolve:x:101:103:systemd Resolver,,,:/run/systemd:/usr/sbin/nologin
systemd-timesync:x:102:104:systemd Time Synchronization,,,:/run/systemd:/usr/sbin/nologin
messagebus:x:103:106::/nonexistent:/usr/sbin/nologin
syslog:x:104:110::/home/syslog:/usr/sbin/nologin
_apt:x:105:65534::/nonexistent:/usr/sbin/nologin
tss:x:106:111:TPM software stack,,,:/var/lib/tpm:/bin/false
uuidd:x:107:114::/run/uuidd:/usr/sbin/nologin
tcpdump:x:108:115::/nonexistent:/usr/sbin/nologin
avahi-autoipd:x:109:116:Avahi autoip daemon,,,:/var/lib/avahi-autoipd:/usr/sbin/nologin
usbmux:x:110:46:usbmux daemon,,,:/var/lib/usbmux:/usr/sbin/nologin
rtkit:x:111:117:RealtimeKit,,,:/proc:/usr/sbin/nologin
dnsmasq:x:112:65534:dnsmasq,,,:/var/lib/misc:/usr/sbin/nologin
cups-pk-helper:x:113:120:user for cups-pk-helper service,,,:/home/cups-pk-helper:/usr/sbin/nologin
speech-dispatcher:x:114:29:Speech Dispatcher,,,:/run/speech-dispatcher:/bin/false
avahi:x:115:121:Avahi mDNS daemon,,,:/var/run/avahi-daemon:/usr/sbin/nologin
kernoops:x:116:65534:Kernel Oops Tracking Daemon,,,:/:/usr/sbin/nologin
saned:x:117:123::/var/lib/saned:/usr/sbin/nologin
nm-openvpn:x:118:124:NetworkManager OpenVPN,,,:/var/lib/openvpn/chroot:/usr/sbin/nologin
hplip:x:119:7:HPLIP system user,,,:/run/hplip:/bin/false
whoopsie:x:120:125::/nonexistent:/bin/false
colord:x:121:126:colord colour management daemon,,,:/var/lib/colord:/usr/sbin/nologin
geoclue:x:122:127::/var/lib/geoclue:/usr/sbin/nologin
pulse:x:123:128:PulseAudio daemon,,,:/var/run/pulse:/usr/sbin/nologin
gnome-initial-setup:x:124:65534::/run/gnome-initial-setup/:/bin/false
gdm:x:125:130:Gnome Display Manager:/var/lib/gdm3:/bin/false
seed:x:1000:1000:SEED,,,:/home/seed:/bin/bash
systemd-coredump:x:999:999:systemd Core Dumper:/:/usr/sbin/nologin
telnetd:x:126:134::/nonexistent:/usr/sbin/nologin
ftp:x:127:135:ftp daemon,,,:/srv/ftp:/usr/sbin/nologin
sshd:x:128:65534::/run/sshd:/usr/sbin/nologin
[02/17/25]seed@VM:~/.../Labsetup$ cat /etc/passwd | grep seed
seed:x:1000:1000:SEED,,,:/home/seed:/bin/bash
[02/17/25]seed@VM:~/.../Labsetup$
```

The default sheel bash as seen in the output is
seed:x:1000:1000:SEED,,,:/home/seed:/bin/bash

To print all environment variables, used the command printenv. The output shows that environment variables are just variable =value pairs. Using env command, can observe the same output.

Using Printenv command, the results are shown below:

```
[02/17/25]seed@VM:~/.../Labsetup$ printenv
SHELL=/bin/bash
SESSION_MANAGER=local/VM:@/tmp/.ICE-unix/2567,unix/VM:/tmp/.ICE-unix/2567
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=2525
GTK_MODULES=gail:atk-bridge
PWD=/home/seed/lab1/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
GJS_DEBUG_TOPICS=JS ERROR;JS LOG
WINDOWPATH=2
HOME=/home/seed
USERNAME=seed
IM_CONFIG_PHASE=1
LANG=en_US.UTF-8
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=4
```

Using env command,

```
[02/17/25]seed@VM:~/.../Labsetup$ env
SHELL=/bin/bash
SESSION_MANAGER=local/VM:@/tmp/.ICE-unix/2567,unix/VM:/tmp/.ICE-unix/2567
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=2525
GTK_MODULES=gail:atk-bridge
PWD=/home/seed/lab1/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
GJS_DEBUG_TOPICS=JS ERROR;JS LOG
WINDOWPATH=2
HOME=/home/seed
USERNAME=seed
IM_CONFIG_PHASE=1
LANG=en_US.UTF-8
LS_COLORS=rs=0:di=01;34:ln=01;36:mh=00:pi=40;33:so=01;35:do=01;35:bd=40;33
```

To observe the difference between the two output , performed the below task and got the below results

```
[02/17/25]seed@VM:~/.../Labsetup$ printenv > out1
[02/17/25]seed@VM:~/.../Labsetup$ env > out2
[02/17/25]seed@VM:~/.../Labsetup$ diff out1 out2
48c48
< _=/usr/bin/printenv
---
> _=/usr/bin/env
[02/17/25]seed@VM:~/.../Labsetup$
```

Now using **printenv PWD** which returns only the value of the variable PWD. In order to find out all variables that consists of substring PWD use the command **env | grep PWD** which gets all the variables and values that contains PWD as a substring within them. The output is in the format of variable = value.

```
[02/17/25]seed@VM:~/.../Labsetup$ printenv PWD
/home/seed/lab1/Labsetup
[02/17/25]seed@VM:~/.../Labsetup$ env | grep PWD
PWD=/home/seed/lab1/Labsetup
OLDPWD=/home/seed/lab1
[02/17/25]seed@VM:~/.../Labsetup$
```

The unset command helps to delete a particular environment variable as shown below, Once unset PWD and then try to find it using env command, it doesn't not return anything as there is no variable PWD. Using the export command, an environment variable and a value can be set. This command can be used to create or edit a particular environment variable.

```
[02/17/25]seed@VM:~/.../Labsetup$ export MYVAR='my variable'
[02/17/25]seed@VM:~/.../Labsetup$ env | grep MYVAR
MYVAR=my variable
[02/17/25]seed@VM:~/.../Labsetup$ unset MYVAR
[02/17/25]seed@VM:~/.../Labsetup$ env | grep MYVAR
[02/17/25]seed@VM:~/.../Labsetup$
```

TASK 2: PASSING ENVIRONMENT VARIABLES FROM PARENT PROCESS TO CHILD PROCESS.

This is the myprintenv.C program, Here the Printenv function is to print out the environment variable through the global window. In the main program, they have used fork function to create child process if the return value is zero then it child process. Otherwise, the parent process in both process we print out the environmental variable.

```
 7 void printenv()
 8 {
 9   int i = 0;
10   while (environ[i] != NULL) {
11       printf("%s\n", environ[i]);
12       i++;
13   }
14 }
15
16 void main()
17 {
18   pid_t childPid;
19   switch(childPid = fork()) {
20     case 0:  /* child process */
21       printenv();
22       exit(0);
23     default:  /* parent process */
24       // printenv();
25       exit(0);
```

case 1: Child process: comparing the child

```
[02/17/25]seed@VM:~/.../Labsetup$ gcc myprintenv.c -o child
[02/17/25]seed@VM:~/.../Labsetup$
```

Now commenting the parent

```
16 void main()
17 {
18   pid_t childPid;
19   switch(childPid = fork()) {
20     case 0:  /* child process */
21       printenv();
22       exit(0);
23     default:  /* parent process */
24       //printenv();
25       exit(0);
```

The ./child program

```
[02/17/25]seed@VM:~/.../Labsetup$ ./child
SHELL=/bin/bash
SESSION_MANAGER=local/VM:@/tmp/.ICE-unix/2567,unix/VM:/tmp/.ICE-unix/2567
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=2525
GTK_MODULES=gail:atk-bridge
PWD=/home/seed/lab1/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
GJS_DEBUG_TOPICS=JS ERROR;JS LOG
WINDOWPATH=2
HOME=/home/seed
USERNAME=seed
IM_CONFIG_PHASE=1
LANG=en_US.UTF-8
```

The ./parent program

```
[02/17/25]seed@VM:~/.../Labsetup$ ./parent
SHELL=/bin/bash
SESSION_MANAGER=local/VM:@/tmp/.ICE-unix/2567,unix/VM:/tmp/.ICE-unix/2567
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=2525
GTK_MODULES=gail:atk-bridge
PWD=/home/seed/lab1/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
GJS_DEBUG_TOPICS=JS ERROR;JS LOG
WINDOWPATH=2
HOME=/home/seed
USERNAME=seed
IM_CONFIG_PHASE=1
LANG=en_US.UTF-8
LS_COLORS=rs=0:di=01;34:ln=01;36:mh=00:pi=40;33:so=01;35:do=01;35:bd=40;33
```

Comparing the results of the child and the parent are below

```
[02/17/25]seed@VM:~/.../Labsetup$ ./child > cout.txt
[02/17/25]seed@VM:~/.../Labsetup$ ./parent> pout.txt
[02/17/25]seed@VM:~/.../Labsetup$ diff cout.txt pout.txt
49c49
< _=./child
---
> _=./parent
[02/17/25]seed@VM:~/.../Labsetup$
```

The output is interpreted as 49c49 which means 49th line (left) I the left file is changed to 49th line (right) in the right file, where c stands for changing and the left and right numbers indicating the line number. The “<”symbol denotes lines in the left file and “>” indicates in the right file showing the changed content.

This shows the environment variable takes on the value of the last command executed, here the command of the program execution. It is considered a special shell variable and contains different values depending on the scenario. This shows that the environment variable changed depending on the complied program being run but other than that there is no change in the environment variables, if both the programs were complied into a file with same name, there would not be any difference between the output of parent and child process.To view the results of both at the same time, we make small changes in the code , uncomment both the printenv present in the child and the parent. Also, add a print statement to view the output clearly.

```c
void main()
{
  pid_t childPid;
  switch(childPid = fork()) {
    case 0:  /* child process */
      printf("\n Child process environment variables\n==========");
      printenv();
      exit(0);
    default:  /* parent process */
      printf("\n Parent process environment variables\n==========");
      printenv();
      exit(0);
  }
}
```

Comparing the results, using the commands below

```
[02/17/25]seed@VM:~/.../Labsetup$ gcc myprintenv.c -o both
[02/17/25]seed@VM:~/.../Labsetup$ ./both

 Parent process environment variables
==========SHELL=/bin/bash
SESSION_MANAGER=local/VM:@/tmp/.ICE-unix/2567,unix/VM:/tmp/.ICE-unix/2567
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=2525
GTK_MODULES=gail:atk-bridge
PWD=/home/seed/lab1/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
GJS_DEBUG_TOPICS=JS ERROR;JS LOG
WINDOWPATH=2
HOME=/home/seed
```

```
 Child process environment variables
==========SHELL=/bin/bash
SESSION_MANAGER=local/VM:@/tmp/.ICE-unix/2567,unix/VM:/tmp/.ICE-unix/2567
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=2525
GTK_MODULES=gail:atk-bridge
PWD=/home/seed/lab1/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
GJS_DEBUG_TOPICS=JS ERROR;JS LOG
WINDOWPATH=2
HOME=/home/seed
USERNAME=seed
IM_CONFIG_PHASE=1
LANG=en_US.UTF-8
```

TASK 3: ENVIRONMENT VARIABLES AND execve()

Step 1: Running as NULL

Code:

```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 ;
}
```

Comparing it and observed that beforedit is blank as shown below

```
[02/17/25]seed@VM:~/.../Labsetup$ gcc myenv.c -o beforedit
[02/17/25]seed@VM:~/.../Labsetup$ ls
beforedit  both  cap_leak.c  catall.c  child  cout.txt  myenv.c  myprintenv.c  out1  out2  parent  pout.txt
[02/17/25]seed@VM:~/.../Labsetup$ ./beforedit
[02/17/25]seed@VM:~/.../Labsetup$
```

Step 2 : modify the NULL value to environ attaching the screenshot below

```c
#include <unistd.h>

extern char **environ;

int main()
{
   char *argv[2];

   argv[0] = "/usr/bin/env";
   argv[1] = NULL;
   // step 1
   // execve("/usr/bin/env", argv, NULL);
   // step 2
   execve("/usr/bin/env", argv, environ);
   return 0 ;
}
```

The output of afterdit is shown below

```
[02/17/25]seed@VM:~/.../Labsetup$ gcc myenv.c -o afterdit
[02/17/25]seed@VM:~/.../Labsetup$ ls
afterdit   both        catall.c  cout.txt  myprintenv.c  out2    pout.txt
beforedit  cap_leak.c  child     myenv.c   out1          parent
[02/17/25]seed@VM:~/.../Labsetup$ ./afterdit
SHELL=/bin/bash
SESSION_MANAGER=local/VM:@/tmp/.ICE-unix/2567,unix/VM:/tmp/.ICE-unix/2567
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=2525
GTK_MODULES=gail:atk-bridge
PWD=/home/seed/lab1/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
GJS_DEBUG_TOPICS=JS ERROR;JS LOG
WINDOWPATH=2
HOME=/home/seed
```

Even though the global environ variable was specified in the program, the beforedit program contained NULL as the third argument of the execve and the afterdit program contained the environ variable as the third argument of the execve. This change affected the output of the program because of the third argument to execve() function specifies the environment variable of the current process. Since the environ variable was not passed in the initial program and hence no environment variables were associated with this new process, the output was null. But after editing the program when passed the environ variable as the third argument to execve which contained all the environment variables as expected. In conclusion the third argument of the execve() command gets the program its environment variables.

TASK 4: ENVIRONMENT VARIABLES AND system()

In this task, executing a new program that is executed via the system() function. This command executes the /bin/sh -c command" that is it executes /bin/sh and asks the shell to execute the command. The program is executed as shown.

myprintenv.c

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

The program is complied and executed, Even though we don't explicity send any envronmnetal variables in the program, the output shows the environment variables of the cureent process. This happens due to system function implicity passes the environment variables to the called funtion /bin/sh.

```
[02/18/25]seed@VM:~/.../Labsetup$ gcc -o system env.c
[02/18/25]seed@VM:~/.../Labsetup$ ./env
GJS_DEBUG_TOPICS=JS ERROR;JS LOG
LESSOPEN=| /usr/bin/lesspipe %s
USER=seed
SSH_AGENT_PID=1853
XDG_SESSION_TYPE=x11
SHLVL=1
HOME=/home/seed
OLDPWD=/home/seed/lab1
DESKTOP_SESSION=ubuntu
GNOME_SHELL_SESSION_MODE=ubuntu
GTK_MODULES=gail:atk-bridge
MANAGERPID=1571
DBUS_SESSION_BUS_ADDRESS=unix:path=/run/user/1000/bus
COLORTERM=truecolor
IM_CONFIG_PHASE=1
LOGNAME=seed
JOURNAL_STREAM=9:34499
_=./env
XDG_SESSION_CLASS=user
USERNAME=seed
TERM=xterm-256color
```

The path link

```
[02/18/25]seed@VM:~/.../Labsetup$ ls -l /bin/sh
lrwxrwxrwx 1 root root 4 Nov 24  2020 /bin/sh -> dash
[02/18/25]seed@VM:~/.../Labsetup$
```

TASK5: ENVIRONMENT VARIABLE AND SET-UID PROBLEM

Step1: Printing all the environment variables in the current process.

```
[02/18/25]seed@VM:~/.../Labsetup$ gedit pritenv.c
[02/18/25]seed@VM:~/.../Labsetup$ ls
cap_leak.c  catall.c  env  env.c  myenv.c  myprintenv.c  pritenv.c  system
[02/18/25]seed@VM:~/.../Labsetup$ gcc pritenv.c -o pritenv
[02/18/25]seed@VM:~/.../Labsetup$ ls
cap_leak.c  catall.c  env  env.c  myenv.c  myprintenv.c  pritenv  pritenv.c  system
[02/18/25]seed@VM:~/.../Labsetup$ ./pritenv
SHELL=/bin/bash
SESSION_MANAGER=local/VM:@/tmp/.ICE-unix/1898,unix/VM:/tmp/.ICE-unix/1898
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=1853
GTK_MODULES=gail:atk-bridge
PWD=/home/seed/lab1/Labsetup
LOGNAME=seed
XDG_SESSION_DESKTOP=ubuntu
```

STEP2: Changing the ownership to root and making it a SET-UID program.

```
[02/18/25]seed@VM:~/.../Labsetup$ sudo chown root pritenv
[02/18/25]seed@VM:~/.../Labsetup$ sudo chmod 4755 pritenv
[02/18/25]seed@VM:~/.../Labsetup$ ls -l pritenv
-rwsr-xr-x 1 root seed 16768 Feb 18 14:26 pritenv
[02/18/25]seed@VM:~/.../Labsetup$
```

Step3: Using the export command to set the following environmental variable

```
[02/18/25]seed@VM:~/.../Labsetup$ echo $PATH
/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin:/usr/games:/usr/local/games:/snap/bin
:.
[02/18/25]seed@VM:~/.../Labsetup$ echo $LD_LIBRARY_PATH

[02/18/25]seed@VM:~/.../Labsetup$ export MYVAR ='my variable'
bash: export: `=my variable': not a valid identifier
[02/18/25]seed@VM:~/.../Labsetup$ export MYVAR='my variable'
[02/18/25]seed@VM:~/.../Labsetup$ LD_LIBRARY_PATH='.'
[02/18/25]seed@VM:~/.../Labsetup$ printenv PATH
/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin:/usr/games:/usr/local/games:/snap/bin
:.
[02/18/25]seed@VM:~/.../Labsetup$ printenv LD_LIBRARY_PATH
[02/18/25]seed@VM:~/.../Labsetup$ export LD_LIBRARY_PATH
[02/18/25]seed@VM:~/.../Labsetup$ printenv LD_LIBRARY_PATH
.
[02/18/25]seed@VM:~/.../Labsetup$ printenv MYVAR
my variable
```

```
[02/18/25]seed@VM:~/.../Labsetup$ ./pritenv | grep PATH
WINDOWPATH=2
PATH=/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin:/usr/games:/usr/local/games:/sna
p/bin:.
[02/18/25]seed@VM:~/.../Labsetup$ ./pritenv | grep LD_LIBRARY PATH
grep: PATH: No such file or directory
[02/18/25]seed@VM:~/.../Labsetup$ ./pritenv | grep LD_LIBRARY_PATH
[02/18/25]seed@VM:~/.../Labsetup$
```

```
[02/18/25]seed@VM:~/.../Labsetup$ ./printenv | grep MYVAR
bash: ./printenv: No such file or directory
[02/18/25]seed@VM:~/.../Labsetup$ ./pritenv | grep MYVAR
MYVAR=my variable
[02/18/25]seed@VM:~/.../Labsetup$ ./pritenv | grep PATH
WINDOWPATH=2
PATH=/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin:/usr/games:/usr/local/games:/sna
p/bin:.
[02/18/25]seed@VM:~/.../Labsetup$ ./pritenv | grep LD_LIBRARY PATH
grep: PATH: No such file or directory
[02/18/25]seed@VM:~/.../Labsetup$ ./pritenv | grep LD_LIBRARY_PATH
[02/18/25]seed@VM:~/.../Labsetup$
```

After compiling the given program, changed the ownership and permissions of the file using the following commands:
sudo chown root filename [making the root as the owner of the filename]
sudo chmod 4755 filename [making the program a SET-UID program by setting set-uid bit]

This makes the program a SET UID root program. Looking at the environment variables, since the PATH and LD_LIBRARY_PATH are already present, only initialize a new variable with name MYVAR= 'my variable' using the export command allowing the other environment values to be same. This shows that the SET-UID program may not inherit all the environment variables of the parent process. For example: LD_LIBRARY_PATH this is a security mechanism implemented by the dynamic linker. The LD_LIBRARY_PATH is ignored here because the real user id and effective user id is different. This is the reason two other environment variables are seen in the output.

TASK 6: THE PATH ENVIRONMENT VARIABLE AND set-UID Programs

The given program is written in the file named task 6.c and the compiled into task 6. The compiled owner is changed into root and is converted into SET-UID program. Check the current value of the environment variable PATH and also the working directory of the program.

```
[02/19/25]seed@VM:~/.../Labsetup$ gcc -o task6 task6.c
[02/19/25]seed@VM:~/.../Labsetup$ ls
cap_leak.c  env     ls     myenv.c        pritenv     system   task6.c    task7.c
catall.c    env.c   ls.c   myprintenv.c   pritenv.c   task6    task7a.c
[02/19/25]seed@VM:~/.../Labsetup$ task6
cap_leak.c  env     ls     myenv.c        pritenv     system   task6.c    task7.c
catall.c    env.c   ls.c   myprintenv.c   pritenv.c   task6    task7a.c
[02/19/25]seed@VM:~/.../Labsetup$ gedit ls.c
[02/19/25]seed@VM:~/.../Labsetup$ gcc -o ls ls.c
[02/19/25]seed@VM:~/.../Labsetup$ ls
cap_leak.c  env     ls     myenv.c        pritenv     system   task6.c    task7.c
catall.c    env.c   ls.c   myprintenv.c   pritenv.c   task6    task7a.c
[02/19/25]seed@VM:~/.../Labsetup$ ./ls
My malicious ls program is called1
[02/19/25]seed@VM:~/.../Labsetup$ gedit ls.c
[02/19/25]seed@VM:~/.../Labsetup$ /bin/ls
cap_leak.c  env     ls     myenv.c        pritenv     system   task6.c    task7.c
catall.c    env.c   ls.c   myprintenv.c   pritenv.c   task6    task7a.c
[02/19/25]seed@VM:~/.../Labsetup$ ls -l ls
-rwxrwxr-x 1 seed seed 16696 Feb 19 14:53 ls
[02/19/25]seed@VM:~/.../Labsetup$ sudo chown root ls
[02/19/25]seed@VM:~/.../Labsetup$ ls -l ls
-rwxrwxr-x 1 root seed 16696 Feb 19 14:53 ls
[02/19/25]seed@VM:~/.../Labsetup$ sudo chmod 4755 ls
[02/19/25]seed@VM:~/.../Labsetup$ ls -l ls
```

Confirming that it is set to the SET UID with root as the owner and the program which is compiled is shown below

task6.c

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

int main(){
 printf("My malicious ls program is called 1\n");
 return 0;
 }
```

To run my program instead of the standard “ls” program, I changed the value of the environment variable PATH and provided the path to my file as the first value of the variable. This makes the program search for the file in my directory first before any other directory and sine I have the file with same name as ls, the current program will execute my program.

```
[02/19/25]seed@VM:~/.../Labsetup$ ls -l ls
-rwxrwxr-x 1 seed seed 16696 Feb 19 14:53 ls
[02/19/25]seed@VM:~/.../Labsetup$ sudo chown root ls
[02/19/25]seed@VM:~/.../Labsetup$ ls -l ls
-rwxrwxr-x 1 root seed 16696 Feb 19 14:53 ls
[02/19/25]seed@VM:~/.../Labsetup$ sudo chmod 4755 ls
[02/19/25]seed@VM:~/.../Labsetup$ ls -l ls
-rwsr-xr-x 1 root seed 16696 Feb 19 14:53 ls
[02/19/25]seed@VM:~/.../Labsetup$ pwd
/home/seed/lab1/Labsetup
[02/19/25]seed@VM:~/.../Labsetup$ printenv PATH
/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin:/usr/games:/usr/local/games:/snap/bin:.
[02/19/25]seed@VM:~/.../Labsetup$ export PATH=/home/seed/lab1/Labsetup:$PATH
[02/19/25]seed@VM:~/.../Labsetup$ printenv PATH
/home/seed/lab1/Labsetup:/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin:/usr/games:/usr/local/games:/snap/bin:.
[02/19/25]seed@VM:~/.../Labsetup$ LS
LS: command not found
[02/19/25]seed@VM:~/.../Labsetup$ ls
My malicious ls program is called1
[02/19/25]seed@VM:~/.../Labsetup$ /bin/ls
cap_leak.c  env     ls     myenv.c        pritenv     system   task6.c    task7.c
catall.c    env.c   ls.c   myprintenv.c   pritenv.c   task6    task7a.c
[02/19/25]seed@VM:~/.../Labsetup$ ls
My malicious ls program is called1
[02/19/25]seed@VM:~/.../Labsetup$
```

The following shows that I have compiled my program with file name ls, on running the task6m it runs my ls program instead of the system ls.

```
[02/19/25]seed@VM:~/.../Labsetup$ ls
My malicious ls program is called1
[02/19/25]seed@VM:~/.../Labsetup$ /bin/ls
cap_leak.c  env    ls    myenv.c       pritenv    system  task6.c   task7.c
catall.c    env.c  ls.c  myprintenv.c  pritenv.c  task6   task7a.c
[02/19/25]seed@VM:~/.../Labsetup$ ls
My malicious ls program is called1
```

The following shows that task 6 consists of the ls program with normal privileges.

```
[02/19/25]seed@VM:~/.../Labsetup$ ls -l ls
-rwsr-xr-x 1 root seed 16696 Feb 19 14:53 ls
```

This shows the way in which PATH environment variable can be changed to a desired folder and execute the user defined programs which could be malicious. Since we are using system() it is potentially dangerous due to inclusion of the shell and the environment variables. Also, instead of specifying the abolute path, we have specified the relative path of the process. Due to this, the system() will spawn a shell which will look for a ls program in the location specidied by the PATH environment variable. Hence by changing the PATH value to the folder containing a malicious code with root privileges because it is root owned SET UID program. Hence using relative path and system function in a SET-UID program could lead to severe attacks.

In order to overcome the dash shell security mechanism of dropping SET -UID program's privileges on being called from one, we link the /bin/sh to another shell.

```
[02/19/25]seed@VM:~/.../Labsetup$ sudo rm /bin/sh
[02/19/25]seed@VM:~/.../Labsetup$ sudo ln -s /bin/zsh /bin/sh
[02/19/25]seed@VM:~/.../Labsetup$
```

TASK 7: THE LD_PRELOAD ENVIRONMENT VARIABLE AND SET-UID PROGRAMS

First created a program named mylib.c that has the sleep function overriding the system's sleep function as given. This function is just printing a statement on the standard output. After this, we compile the program using the following command: gcc -fPIC -g -c mylib.c where fPIC means emit position independent code, suitable for dynamic linking and avoiding any limit on the size of the global offset table, -g means producing debugging information and -c means compiling the file but not linking it.

Gcc -shared -o filename mylib.o -lc (where -shared produces a shared object that can be linked to other objects to form a executable, -o file stores the output in the file)

This executable file is the value of LD_PRELOAD variable. This makes any program to load this library before executing the program.

After this a program with sleep function in the same directory and it compiles

myprog.c

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

On running this program as a normal user we see that program calls the sleep function defined by us and prints the statement that is defined.

```
[02/19/25]seed@VM:~/.../Labsetup$ gedit mylib.c
[02/19/25]seed@VM:~/.../Labsetup$ gcc -fPIC -g -c mylib.c
[02/19/25]seed@VM:~/.../Labsetup$ gcc -shared -o libmylib.so.1.0.1 mylib.o -lc
[02/19/25]seed@VM:~/.../Labsetup$ export LD_PRELOAD=./limylib.so.1.0.1
ERROR: ld.so: object './limylib.so.1.0.1' from LD_PRELOAD cannot be preloaded (cannot open shared
object file): ignored.
[02/19/25]seed@VM:~/.../Labsetup$ export LD_PRELOAD=./libmylib.so.1.0.1
[02/19/25]seed@VM:~/.../Labsetup$ gedit myprog.c
[02/19/25]seed@VM:~/.../Labsetup$ gcc -o myprog myprog.c
[02/19/25]seed@VM:~/.../Labsetup$ ./myprog
I am not sleeping[02/19/25]seed@VM:~/.../Labsetup$
[02/19/25]seed@VM:~/.../Labsetup$ ./myprog
I am not sleeping[02/19/25]seed@VM:~/.../Labsetup$
[02/19/25]seed@VM:~/.../Labsetup$
```

On running the same program in different scenarios as specified in the lab document, noticed that in certain situations the library containing my sleep function was not called and instead the system defined sleep function was executed.

Next, I made the program a SET UID root program and ran the program again. The output shows that my library containing the sleep function was not called and also shows that the environment variable of that process did not contain the LD_PRELOAD variable. This showed that the SET-UID child process that was created did not inherit LD_PRELOAD variable and hence it did not load my library function but the system defined sleep function causing the program to sleep.

```
[02/19/25]seed@VM:~/.../Labsetup$ sudo chown root myprog
[02/19/25]seed@VM:~/.../Labsetup$ sudo chmod 4755 myprog
[02/19/25]seed@VM:~/.../Labsetup$ ./myprog
[02/19/25]seed@VM:~/.../Labsetup$ sudo su
root@VM:/home/seed/lab1/Labsetup# export LD_PRELOAD=./libmylib.so.1.0.1
root@VM:/home/seed/lab1/Labsetup# ./myprog
I am not sleepingroot@VM:/home/seed/lab1/Labsetup#
root@VM:/home/seed/lab1/Labsetup# exit
exit
```

Since the program was already a SET UID root program , I just logged into the root user account and defined the LD_PRELOAD variable. On running the program, see that the user defined sleep function is executed and LD_PRELOAD variable is present. This happens because the root account and the function's owner is root as well. This makes the process have the same real ID and effective ID and hence LD_PRELOAD variable is not dropped.

```
[02/19/25]seed@VM:~/.../Labsetup$ sudo adduser user1
Adding user `user1' ...
Adding new group `user1' (1001) ...
Adding new user `user1' (1001) with group `user1' ...
Creating home directory `/home/user1' ...
Copying files from `/etc/skel' ...
New password:
Retype new password:
passwd: password updated successfully
Changing the user information for user1
Enter the new value, or press ENTER for the default
        Full Name []: soundarya
        Room Number []: 3210
        Work Phone []: 9999999
        Home Phone []: 8888888
        Other []:
Is the information correct? [Y/n] Y
[02/19/25]seed@VM:~/.../Labsetup$ ls -l myprog
My malicious ls program is called1
[02/19/25]seed@VM:~/.../Labsetup$ sudo chown user1 myprog
[02/19/25]seed@VM:~/.../Labsetup$ sudo chmod 4755 myprog
[02/19/25]seed@VM:~/.../Labsetup$ export LD_PRELOAD=./libmylib.so.1.0.1
[02/19/25]seed@VM:~/.../Labsetup$ ./myprog
[02/19/25]seed@VM:~/.../Labsetup$
```

Making the file owner as user 1 another user account other than root and making it a SET UID program. Go into this user account and set the LD _PRELOAD variable again. On running the program again the user defined sleep function is called and also the LD_PRELOAD variable is present in the current process.

This behavior indicates that the LD_PRELOAD variable is present if the effective and the real ID are the same and is dopped if they are different. This is due to the SET-UID program's security mechanisms.

TASK 8: INVOKING EXTERNAL PROGRAMS USING system() versus execve()

```
[02/19/25]seed@VM:~/.../Labsetup$ gcc -o catall catall.c
[02/19/25]seed@VM:~/.../Labsetup$ sudo chown root catall
[02/19/25]seed@VM:~/.../Labsetup$ sudo chown 4755 catall
[02/19/25]seed@VM:~/.../Labsetup$ ls -l catall
-rwxrwxr-x 1 4755 seed 16928 Feb 19 19:51 catall
[02/19/25]seed@VM:~/.../Labsetup$ gedit catall.txt
[02/19/25]seed@VM:~/.../Labsetup$ ./catall catall.txt
This is the file which Bob can read, cannot modify this file.
[02/19/25]seed@VM:~/.../Labsetup$ ./catall "catall.txt;rm catall.txt"
This is the file which Bob can read, cannot modify this file.
[02/19/25]seed@VM:~/.../Labsetup$ ./catall catall.txt
/bin/cat: catall.txt: No such file or directory
[02/19/25]seed@VM:~/.../Labsetup$
[02/19/25]seed@VM:~/.../Labsetup$ gcc -o catall catall.c
[02/19/25]seed@VM:~/.../Labsetup$ sudo chown root  catall
[02/19/25]seed@VM:~/.../Labsetup$ sudo chmod 4755 catall
[02/19/25]seed@VM:~/.../Labsetup$ ls -l catall
-rwsr-xr-x 1 root seed 16928 Feb 19 19:56 catall
```

The program is first compiled and then converted into the root owned SET-UID program with executable permission to other users.

Considering bob is using the user account treating bob as others. Here we can see the programs runs normally when provide the file to read. But, if we provide a malicious input such as “txt;bin/sh”, the program first read the contents of the document and then run /bin/sh as the command. This command allows Bob to run the shell program which has root privileges and bob then runs the rm command to remove a file on which it did not have the writ permission. The root terminal is indicated by #. This shows that even though Bob did not have permission to write, it could remove a file easily by assuming the privileges of the root user.

The problem here is the system call inside the program which does not separate the command and user input.

```
[02/19/25]seed@VM:~/.../Labsetup$ ./catall /etc/shadow
root:!:18590:0:99999:7:::
daemon:*:18474:0:99999:7:::
bin:*:18474:0:99999:7:::
sys:*:18474:0:99999:7:::
sync:*:18474:0:99999:7:::
games:*:18474:0:99999:7:::
man:*:18474:0:99999:7:::
lp:*:18474:0:99999:7:::
mail:*:18474:0:99999:7:::
news:*:18474:0:99999:7:::
uucp:*:18474:0:99999:7:::
proxy:*:18474:0:99999:7:::
www-data:*:18474:0:99999:7:::
backup:*:18474:0:99999:7:::
```

Running shadow command to catch all the values in this path different from when we run the catall function.

This can be avoided by segregating the user input and the command in the program. Since the system call requires constructing the command using the input , we should avoid using system functions in the program and instead execve function which treats anything input from the user as the input string does not allow to run as a command. We edit the program and then re-compile making it a root owned set UID program.

```
[02/19/25]seed@VM:~/.../Labsetup$ gedit catall.txt
[02/19/25]seed@VM:~/.../Labsetup$ ./catall catall.txt
This is the file Bob can read, cannot modify this file.
[02/19/25]seed@VM:~/.../Labsetup$ ./catall "catall.txt; rm catall.txt"
/bin/cat: 'catall.txt; rm catall.txt': No such file or directory
[02/19/25]seed@VM:~/.../Labsetup$ ./catall catall.txt
This is the file Bob can read, cannot modify this file.
[02/19/25]seed@VM:~/.../Labsetup$
```

When the system function executes it does not execute the command directly, it calls the shell instead and executes the command. So, if the program is a SET UID program. The user will have temporary root privileges and can remove any file he wants with the root privileges. Multiple commands can be passed together using semicolon signs. System commands call the shell and the shell passes the string which handles the semicolon. Whereas execve function command replaces the program with called program and passes the argument strings exactly as specified and doesn't interpret quotes so when we pass something after the semicolon sign it is treated as a new command and root privileges would have been lost. So the rm command is executed using user privileges which is why it cant delete the file.

TASK 9: CAPABILITY LEAKING

Complied the given program and make it a root owned SET-UID program, we create a file name zzz in the /etc folder containing print in the main. 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 files was 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 the privileged file. This shows that it is important to close the descriptor after dropping the privileges, for it to have the appropriate permissions.

```
[02/19/25]seed@VM:~/.../Labsetup$ gedit cap_leak.c
[02/19/25]seed@VM:~/.../Labsetup$ gcc -o cap_leak cap_leak.c
[02/19/25]seed@VM:~/.../Labsetup$ sudo chown root cap_leak
[02/19/25]seed@VM:~/.../Labsetup$ sudo chmod 4755 cap_leak
[02/19/25]seed@VM:~/.../Labsetup$ ls -l cap_leak
-rwsr-xr-x 1 root seed 17008 Feb 19 20:48 cap_leak
[02/19/25]seed@VM:~/.../Labsetup$ stat -c %a cap_leak
4755
[02/19/25]seed@VM:~/.../Labsetup$ sudo su
root@VM:/home/seed/lab1/Labsetup# cd /etc
root@VM:/etc# nano zzz
root@VM:/etc# cat zzz
Task 9 capability Leaks
root@VM:/etc# ls -l zzz
-rw-r--r-- 1 root root 24 Feb 19 20:50 zzz
root@VM:/etc# exit
exit
[02/19/25]seed@VM:~/.../Labsetup$ ./cap_leak
fd is 3
$ cat zx zz      \
zsh: command not found: c
$ cat zzz
cat: zzz: No such file or directory
$ cat /etc/zzz
Task 9 capability Leaks
$ exit
[02/19/25]seed@VM:~/.../Labsetup$
```

```
[03/11/25]seed@VM:~/Labsetup$ gcc -o cap_leak cap_leak.c
[03/11/25]seed@VM:~/Labsetup$ sudo chown root cap_leak
[03/11/25]seed@VM:~/Labsetup$ sudo chmod 4755 cap_leak
[03/11/25]seed@VM:~/Labsetup$ sudo touch /etc/zzz
[03/11/25]seed@VM:~/Labsetup$ sudo chmod 0644 /etc/zzz
[03/11/25]seed@VM:~/Labsetup$ ./cap_leak
fd is 3
$ echo "malicious Entry" >&3
$ cat /etc/zzz
malicious Entry
$
```