CB.SC.P2CYS23017

Muhammed Nifal V

Environment Variable and Set-UID Program Lab

Task 1:

To print all environment variables

Cmd: printenv

In order to print all environment variables, I typed printenv. The output shows that environment variables are just variable = value pairs. Entering env would give a similar output.

```
[10/05/23]seed@VM:~/.../Labsetup$ printenv
SHELL=/bin/bash
SESSION_MANAGER=local/VM:@/tmp/.ICE-unix/1793,unix/VM:/tmp/.ICE-uni
x/1793
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=1746
GTK_MODULES=gail:atk-bridge
DBUS_STARTER_BUS_TYPE=session
PWD=/home/seed/Desktop/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
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:b
d=40;33;01:cd=40;33;01:or=40;31;01:mi=00:su=37;41:sg=30;43:ca=30;41
:tw=30;42:ow=34;42:st=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:*
.lzma=01;31:*.tlz=01;31:*.txz=01;31:*.tzo=01;31:*.t7z=01;31:*.zip=0
1;31:*.z=01;31:*.dz=01;31:*.gz=01;31:*.lrz=01;31:*.lz=01;31:*.lzo=0
```

```
=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:*.xspf=00;36:
XDG_CURRENT_DESKTOP=ubuntu:GNOME
VTE_VERSION=6003
GNOME_TERMINAL_SCREEN=/org/gnome/Terminal/screen/f01d634c_9817_41cb
_bdd7_7473aec87e55
INVOCATION_ID=81630c8a51194490b42c69f97a18ec1f
MANAGERPID=1536
LESSCLOSE=/usr/bin/lesspipe %s %s
XDG_SESSION_CLASS=user
TERM=xterm-256color
LESSOPEN=| /usr/bin/lesspipe %s
USER=seed
GNOME_TERMINAL_SERVICE=:1.98
DISPLAY=:0
SHLVL=1
QT_IM_MODULE=ibus
DBUS_STARTER_ADDRESS=unix:path=/run/user/1000/bus,guid=9db1732baeaa
a787277e4c1b651e54a6
XDG_RUNTIME_DIR=/run/user/1000
JOURNAL_STREAM=9:32830
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,guid=9db1732b
aeaaa787277e4c1b651e54a6
OLDPWD=/home/seed
_=/usr/bin/printenv
[10/05/23]seed@VM:~/.../Labsetup$
```

Cmd: printenv PWD

Which returns only the value of the variable PWD, which displays the current directory path.

```
OLDPWD=/home/seed
_=/usr/bin/printenv
[10/05/23]seed@VM:~/.../Labsetup$ printenv pwd
[10/05/23]seed@VM:~/.../Labsetup$ printenv PWD
/home/seed/Desktop/Labsetup
[10/05/23]seed@VM:~/.../Labsetup$
```

Cmd: unset

The unset command is used to remove or unset environment variables or shell variables.

```
[10/05/23]seed@VM:~/.../Labsetup$ nano myprintenv.c
[10/05/23]seed@VM:~/.../Labsetup$ export task=123
[10/05/23]seed@VM:~/.../Labsetup$ printenv task
123
[10/05/23]seed@VM:~/.../Labsetup$ env |grep task
task=123
[10/05/23]seed@VM:~/.../Labsetup$ unset task
[10/05/23]seed@VM:~/.../Labsetup$ printenv task
[10/05/23]seed@VM:~/.../Labsetup$
```

Cmd |grep used to search the environmental variables

Task 2:

To find the difference in the environment variables of child and parent process.

Cmd: diff

Source Code:

```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);
  }
}
```

```
_=./a.out
[10/05/23]seed@VM:~/.../Labsetup$ gcc myprintenv.c
[10/05/23]seed@VM:~/.../Labsetup$ gcc myprintenv.c -o printenv
[10/05/23]seed@VM:~/.../Labsetup$ nano myprintenv.c
[10/05/23]seed@VM:~/.../Labsetup$ gcc myprintenv.c -o printenv1
[10/05/23]seed@VM:~/.../Labsetup$ printenv > file1
[10/05/23]seed@VM:~/.../Labsetup$ printenv1 > file2
[10/05/23]seed@VM:~/.../Labsetup$ diff file1 file12
diff: file12: No such file or directory
[10/05/23]seed@VM:~/.../Labsetup$ diff file1 file2
49c49
< _=/usr/bin/printenv
---
> _=./printenv1
[10/05/23]seed@VM:~/.../Labsetup$
```

The command diff file1 file2 which outputs the difference between to process highlighting where the lines have been modified, added or deleted.

49c49 means that in the 49th line (left) in left file is changed to the 49th line (right) in the right file, where c stands for changing and the left and right numbers indicate the line number. The < denotes lines in the left file and > indicates in the right file showing the changed content.

Task 3:

Cmd: **myenv**

The new program must get its environment variables explicitly through the execve call. As we saw from the task, if no environment variables are passed through the call, the program will not have access to them.

Source 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 ;
}
```

Output:

```
[10/15/23]seed@VM:~/.../Labsetup$ gedit myenv.c
[10/15/23]seed@VM:~/.../Labsetup$ gcc myenv.c -o myenv1
[10/15/23]seed@VM:~/.../Labsetup$ ./myenv1
[10/15/23]seed@VM:~/.../Labsetup$
```

Source Code:

```
#include <unistd.h>

extern char **environ;

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

  argv[0] = "/usr/bin/env";
  argv[1] = NULL;

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

  return 0 ;
}
```

Output:

```
[10/15/23]seed@VM:~/.../Labsetup$ gedit myenv.c
[10/15/23]seed@VM:~/.../Labsetup$ gcc myenv.c -o myenv1
[10/15/23]seed@VM:~/.../Labsetup$ ./myenv1
SHELL=/bin/bash
SESSION_MANAGER=local/VM:@/tmp/.ICE-unix/1896,unix/VM:/tmp/.ICE-uni
x/1896
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=1849
GTK_MODULES=gail:atk-bridge
DBUS_STARTER_BUS_TYPE=session
PWD=/home/seed/Desktop/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
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:b
d=40;33;01:cd=40;33;01:or=40;31;01:mi=00:su=37;41:sg=30;43:ca=30;41
```

Here, as seen, the Task 3 program is compiled and executed into respective output files and the output is stored in myenv (with NULL as the argument) and same myenv (with environ as the argument. The observation was that be myenv file was blank and next myenv had the output.

Task 4: Environment Variables and system()

Cmd: system()

Source code:

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

Output:

```
[10/06/23]seed@VM:~/.../Labsetup$ nano system.c
[10/06/23]seed@VM:~/.../Labsetup$ gcc system.c -o system1
[10/06/23]seed@VM:~/.../Labsetup$ ./system1
SHELL=/bin/bash
SESSION_MANAGER=local/VM:@/tmp/.ICE-unix/1786,unix/VM:/tmp/.ICE-uni
x/1786
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=1739
GTK_MODULES=gail:atk-bridge
DBUS_STARTER_BUS_TYPE=session
PWD=/home/seed/Desktop/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
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:b
d=40;33;01:cd=40;33;01:or=40;31;01:mi=00:su=37;41:sg=30;43:ca=30;41
:tw=30;42:ow=34;42:st=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:*
```

The program is compiled and executed and as seen, even though we don't explicitly send any environment variables in the program, the output shows the environment variable of the current process. This happens because the system function implicitly passes the environment variables to the called function /bin/sh.

Task 5:Environmental Variables and Set-UID PRograms

Environment Variable and Set-UID Programs:

When a Set-UID program runs, it assumes the owner's privileges.

Source Code:

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

extern char **environ;
int main()
{
int i = 0;
while (environ[i] != NULL)
{
printf("%s\n", environ[i]);
i++;
}
}
```

Change the ownership to root, and make it a Set-UID program.

Output:

```
[10/06/23]seed@VM:~/.../Labsetup$ nano environ.c
[10/06/23]seed@VM:~/.../Labsetup$ gcc environ.c -o environ
[10/06/23]seed@VM:~/.../Labsetup$ sudo chown root environ
[10/06/23]seed@VM:~/.../Labsetup$ sudo chmod 4755 environ
[10/06/23]seed@VM:~/.../Labsetup$ ./environ
SHELL=/bin/bash
SESSION_MANAGER=local/VM:@/tmp/.ICE-unix/1786,unix/VM:/tmp/.ICE-uni
x/1786
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=1739
GTK_MODULES=gail:atk-bridge
DBUS_STARTER_BUS_TYPE=session
PWD=/home/seed/Desktop/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
WINDOWPATH=2
HOME=/home/seed
USERNAME=seed
IM_CONFIG_PHASE=1
```

```
QT_IM_MODULE=ibus
DBUS_STARTER_ADDRESS=unix:path=/run/user/1000/bus,guid=7e804b2b659b
49295ec42a5765202004
XDG_RUNTIME_DIR=/run/user/1000
JOURNAL_STREAM=9:32865
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,guid=7e804b2b
659b49295ec42a5765202004
OLDPWD=/home/seed
_=./environ
[10/06/23]seed@VM:~/.../Labsetup$ nano environ.c
[10/06/23]seed@VM:~/.../Labsetup$ export PATH="bin/:/usr/bin"
[10/06/23]seed@VM:~/.../Labsetup$ printenv PATH
bin/:/usr/bin
[10/06/23]seed@VM:~/.../Labsetup$ export LD_LIBRARY_PATH="Mylibrary
 path"
[10/06/23]seed@VM:~/.../Labsetup$ printenv Ld-LIBRARY_PATH
[10/06/23]seed@VM:~/.../Labsetup$ printenv LD-LIBRARY_PATH
[10/06/23]seed@VM:~/.../Labsetup$ printenv LD_LIBRARY_PATH
Mylibrary path
[10/06/23]seed@VM:~/.../Labsetup$ export MY_VAR_ANY="VARIABLES"
[10/06/23]seed@VM:~/.../Labsetup$ printenv MY_VAR_ANY
VARIABLES
[10/06/23]seed@VM:~/.../Labsetup$ ./environ|grep "MY_VAR_ANY\|LD_LI
BRARY_PATH\|PATH"
WINDOWPATH=2
MY_VAR_ANY=VARIABLES
PATH=bin/:/usr/bin
```

On running the above compiled program and storing the output in a file named print_env, it's seen that the child process inherits the PATH and MY_VAR_ANY environment variable but there is no LD environment variable, as can be seen in the screenshot (on searching for LD in the file, it does not return any values).

This shows that the SET-UID program's child process may not inherit all the environment variables of the parent process, LD_LIBRARY_PATH being one of them over here.

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. That is why only the other two environment variables are seen in the output.

LD_LIBRARY_PATH is used to extend the search path for shared libraries on a system, which can be useful for custom or non-standard library locations.

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

By creating an executable file called “ls” in the /home/seed directory, and adding that directory to the PATH environment variable, we were able to make the Set-UID process run that executable instead of the “real” ls.

Source Code:

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

Output:

```
[10/15/23]seed@VM:~/.../Labsetup$ gedit ls.c
[10/15/23]seed@VM:~/.../Labsetup$ gcc ls.c -o ls
ls.c: In function ‘main’:
ls.c:3:1: warning: implicit declaration of function ‘system’ [-Wimp
licit-function-declaration]
    3 | system("ls");
      | ^~~~~~
[10/15/23]seed@VM:~/.../Labsetup$ ./ls
a.out       environ            ls.c          myprog     system1
cap_leak.c  environ.c          lsp           myprog.c   system.c
catall      file1              ls_program.c  print1     test.txt
catall.c    file2              myenv1        print2     zzz
cleak       libmylib.so.1.0.1  myenv.c       printenv
env1        ls                 mylib.c       printenv1
env2        ls1.c              mylib.o       set1
env2.c      ls2                myprintenv.c  set1.c
[10/15/23]seed@VM:~/.../Labsetup$ sudo chown root ls
[10/15/23]seed@VM:~/.../Labsetup$ sudo chmod 4755 ls
[10/15/23]seed@VM:~/.../Labsetup$ ls -al ls
-rwsr-xr-x 1 root seed 16696 Oct 15 13:37 ls
[10/15/23]seed@VM:~/.../Labsetup$ sudo ln -sf /bin/zsh /bin/sh
[10/15/23]seed@VM:~/.../Labsetup$ export PATH=/home/seed:$PATH
[10/15/23]seed@VM:~/.../Labsetup$ ./ls
VM# exit
[10/15/23]seed@VM:~/.../Labsetup$ ls
ls: no such option: color=auto
```

This shows the way in which PATH environment variable can be changed to point 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 the inclusion of shell and the environment variables.

Task 7: LD PRELOAD Environmental variable and SET-UID program

LD_PRELOAD is an environment variable in Unix-like operating systems, such as Linux, that allows you to specify a list of shared libraries to be loaded before all other libraries when a program starts.

Source Code:

Open ▾ mylib.c ~/Desktop/Labsetup Save

```
1 #include<stdio.h>
2 void sleep (int S)
3 {
4 printf("I am not sleeping!\n");
5 }
6
```

Open ▾ myprog.c ~/Desktop/Labsetup Save

```
1 #include<unistd.h>
2 int main()
3 {
4 sleep(1);
5 return 0;
6 }
7
```

Running as regular user:

```
[10/09/23]seed@VM:~/.../Labsetup$ gedit myprog.c
[10/09/23]seed@VM:~/.../Labsetup$ gcc myprog.c -o myprog
[10/09/23]seed@VM:~/.../Labsetup$ ./myprog
I am not sleeping!
```

Making it root owned and setuid, then run as normal user:

```
[10/09/23]seed@VM:~/.../Labsetup$ gedit myprog.c
[10/09/23]seed@VM:~/.../Labsetup$ sudo chown root myprog
[10/09/23]seed@VM:~/.../Labsetup$ sudo chmod 4755 myprog
[10/09/23]seed@VM:~/.../Labsetup$ ./myprog
[10/09/23]seed@VM:~/.../Labsetup$
```

Exporting the ld library and running as root:

```
[10/09/23]seed@VM:~/.../Labsetup$ gedit myprog.c
[10/09/23]seed@VM:~/.../Labsetup$ sudo chown root myprog
[10/09/23]seed@VM:~/.../Labsetup$ sudo chmod 4755 myprog
[10/09/23]seed@VM:~/.../Labsetup$ ./myprog
[10/09/23]seed@VM:~/.../Labsetup$ sudo su
root@VM:/home/seed/Desktop/Labsetup# export LD_PRELOAD=./libmylib.s
o.1.0.1
root@VM:/home/seed/Desktop/Labsetup# ./myprog
I am not sleeping!
```

Switching to new user, exporting library and running the program.

```
root@VM:/home/seed/Desktop/Labsetup# sudo chown nifal myprog
root@VM:/home/seed/Desktop/Labsetup# whoami
root
root@VM:/home/seed/Desktop/Labsetup# su nifal
nifal@VM:/home/seed/Desktop/Labsetup$ export LD_PRELOAD=./libmylib.
so.1.0.1
nifal@VM:/home/seed/Desktop/Labsetup$ ./myprog
I am not sleeping!
nifal@VM:/home/seed/Desktop/Labsetup$
```

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

On running the same program in different scenarios as specified in the lab document, I noticed that in certain situations, the library containing my sleep

function was not called and instead the Environment Variable and Set-UID Program Lab seed system defined sleep function was executed.

The LD_PRELOAD allows you to override or interpose the standard dynamic linking behaviour of the system's dynamic linker, which is typically provided by the ld.so or ld-linux.so libraries. This can be useful for various purposes, including debugging, profiling, or modifying the behaviour of programs without actually altering their source code.

Task 8: Invoking external programs using system() versus execve():

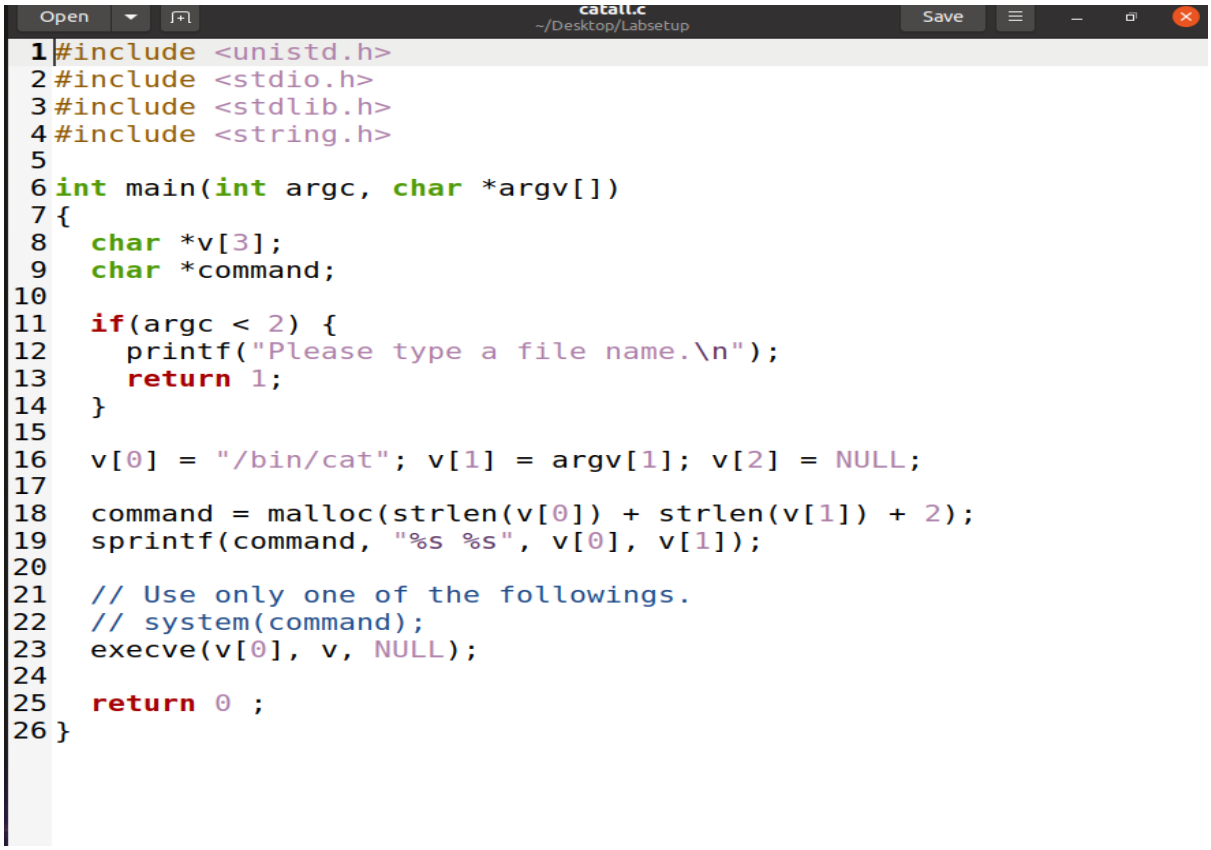
Open catall.c ~/Desktop/Labsetup Save

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

int main(int argc, char *argv[])
{
  char *v[3];
  char *command;

  if(argc < 2) {
    printf("Please type a file name.\n");
    return 1;
  }

  v[0] = "/bin/cat"; v[1] = argv[1]; v[2] = NULL;

  command = malloc(strlen(v[0]) + strlen(v[1]) + 2);
  sprintf(command, "%s %s", v[0], v[1]);

  // Use only one of the followings.
  system(command);
  // execve(v[0], v, NULL);

  return 0 ;
}
```

```
[10/09/23]seed@VM:~/.../Labsetup$ gcc catall.c -o catall
[10/09/23]seed@VM:~/.../Labsetup$ sudo chown root catall
[10/09/23]seed@VM:~/.../Labsetup$ sudo chmod 4755 catall
[10/09/23]seed@VM:~/.../Labsetup$ ./catall
Please type a file name.
[10/09/23]seed@VM:~/.../Labsetup$ ./catall test.txt
hello
[10/09/23]seed@VM:~/.../Labsetup$ sudo su
root@VM:/home/seed/Desktop/Labsetup# su nifal
nifal@VM:/home/seed/Desktop/Labsetup$ ./catall test.txt
hello
nifal@VM:/home/seed/Desktop/Labsetup$ ./catll "test.txt;/bin/sh"
bash: ./catll: No such file or directory
nifal@VM:/home/seed/Desktop/Labsetup$ ./catall "test.txt;/bin/sh"
hello
# rm test.txt
# exit
nifal@VM:/home/seed/Desktop/Labsetup$ ./catall test.txt
/bin/cat: test.txt: No such file or directory
nifal@VM:/home/seed/Desktop/Labsetup$
```

Now using execve command

Source Code:



Output:

```
[10/09/23]seed@VM:~/.../Labsetup$ nano catall.c
[10/09/23]seed@VM:~/.../Labsetup$ gcc catall.c -o catall
[10/09/23]seed@VM:~/.../Labsetup$ sudo chown root catall
[10/09/23]seed@VM:~/.../Labsetup$ sudo chmod 4755 catall
[10/09/23]seed@VM:~/.../Labsetup$ ./catall test.txt
/bin/cat: test.txt: No such file or directory
[10/09/23]seed@VM:~/.../Labsetup$ nano test.txt
[10/09/23]seed@VM:~/.../Labsetup$ ./catall test.txt
hello
[10/09/23]seed@VM:~/.../Labsetup$ ./catall "test.txt;/bin/sh"
/bin/cat: 'test.txt;/bin/sh': No such file or directory
[10/09/23]seed@VM:~/.../Labsetup$
```

The problem here is the system call inside the program which does not separate the command and user input. The user input is eventually treated as a command instead of data/document name.

TASK 9: Capability Leaking

Capabilities can include permissions, privileges, or sensitive information that should only be accessible to authorized users or components. When a capability leak occurs, it can lead to security vulnerabilities or privacy breaches, potentially allowing malicious actors to gain unauthorized access or control over a system, its data, or its resources.

Source Code:

```
 GNU nano 4.8                                                cap_leak.c
#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("/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);
}
```

```
[11/02/23]seed@VM:~/.../Labsetup$ gcc -o cap_leak cap_leak.c
[11/02/23]seed@VM:~/.../Labsetup$ sudo chown root cap_leak
[11/02/23]seed@VM:~/.../Labsetup$ sudo chmod 4755 cap_leak
[11/02/23]seed@VM:~/.../Labsetup$ ls -l cap_leak
-rwsr-xr-x 1 root seed 17008 Nov  2 02:59 cap_leak
[11/02/23]seed@VM:~/.../Labsetup$ cat /etc/zzz
[11/02/23]seed@VM:~/.../Labsetup$ cat /etc/zzz
bbbbbbbb
[11/02/23]seed@VM:~/.../Labsetup$ echo aaaaaa > /etc/zzz
bash: /etc/zzz: Permission denied
[11/02/23]seed@VM:~/.../Labsetup$ ./cap_leak
fd is 3
$ echo ffffff >& 3
$ exit
[11/02/23]seed@VM:~/.../Labsetup$ ./cap_leak
fd is 3
$ exit
[11/02/23]seed@VM:~/.../Labsetup$ cat /etc/zzz
bbbbbbbb
ffffff
[11/02/23]seed@VM:~/.../Labsetup$
```

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 file was still 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 a privileged file. This shows that it is important to close the file descriptor after dropping privileges, in order for it to have the appropriate permissions