Information Security HandsOn Approach HW1

60947045s 呂昀修

Oct, 18, 2021

1. **SEED Lan (50 points)**

   Task1.

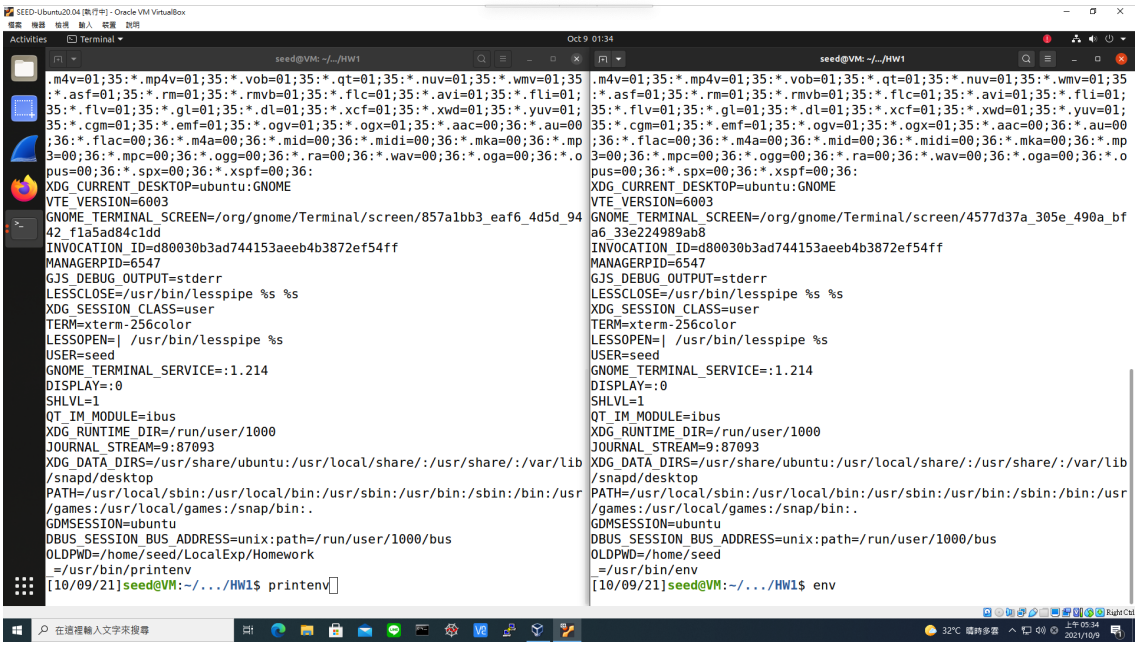

Figure 1: left window does printenv; right window does env

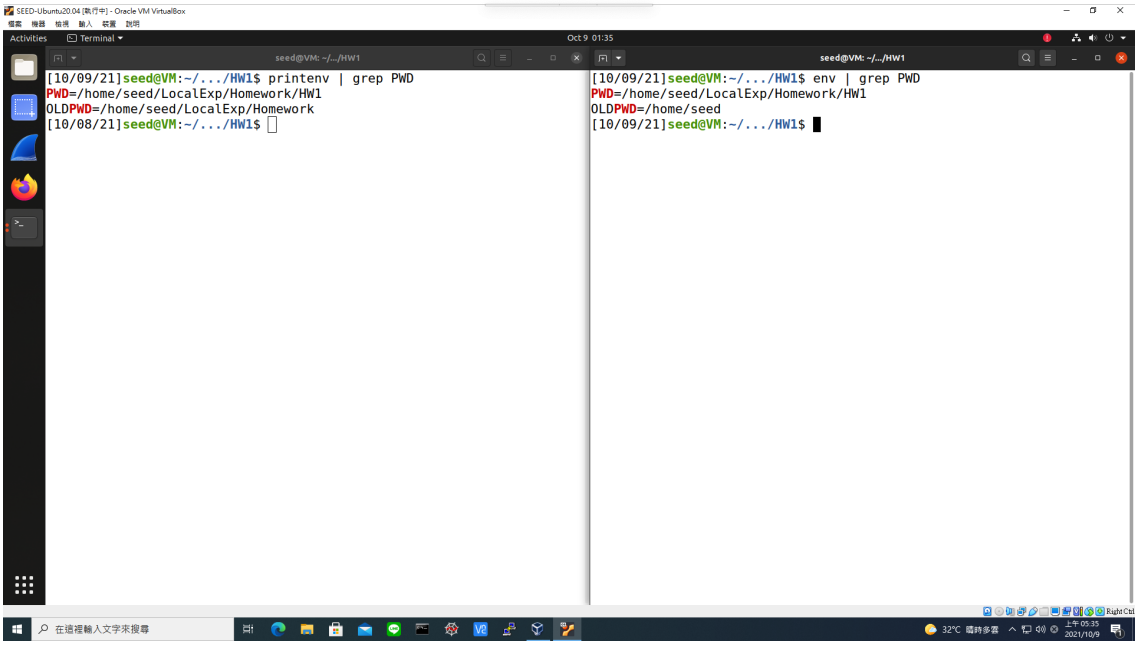

Figure 2: this shows that using grep to show the certain env variable

Next, I have export a new variable called "gohome" which equals "/home/seed" in left window. Clearly show that there isn't a variable called gohome in right window, because command "export" only export env variable in current bash.

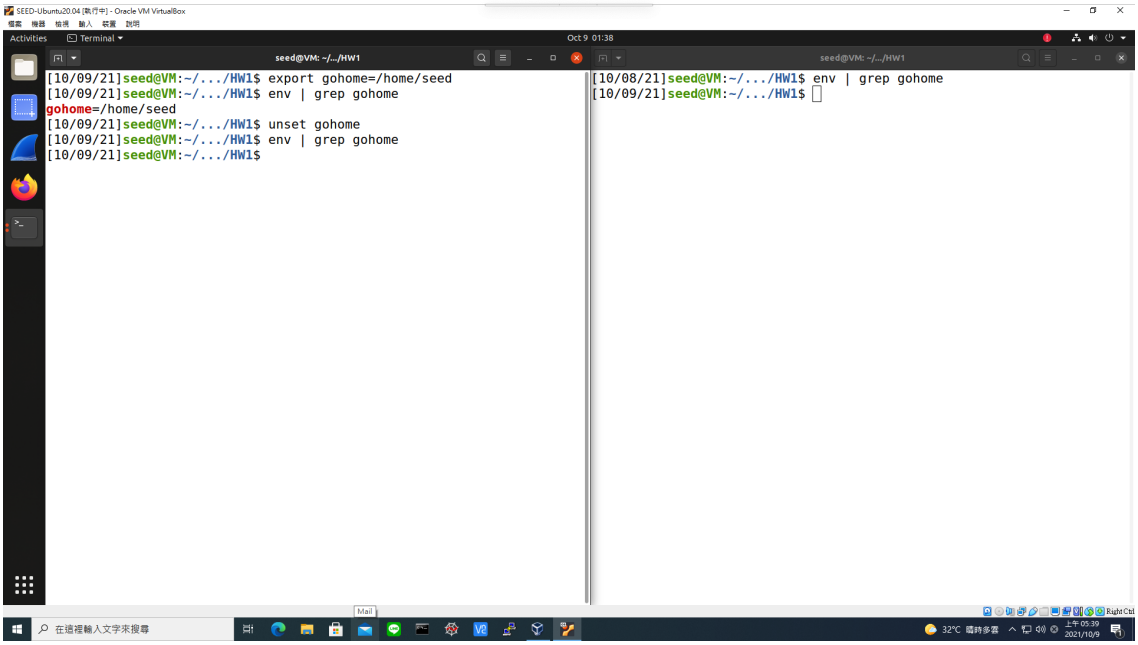

Figure 3: export a new variable "gohome=/home/seed"

Task2.
Step1. I have compared that file and command "env" (where file_env is a bin file contains env output)

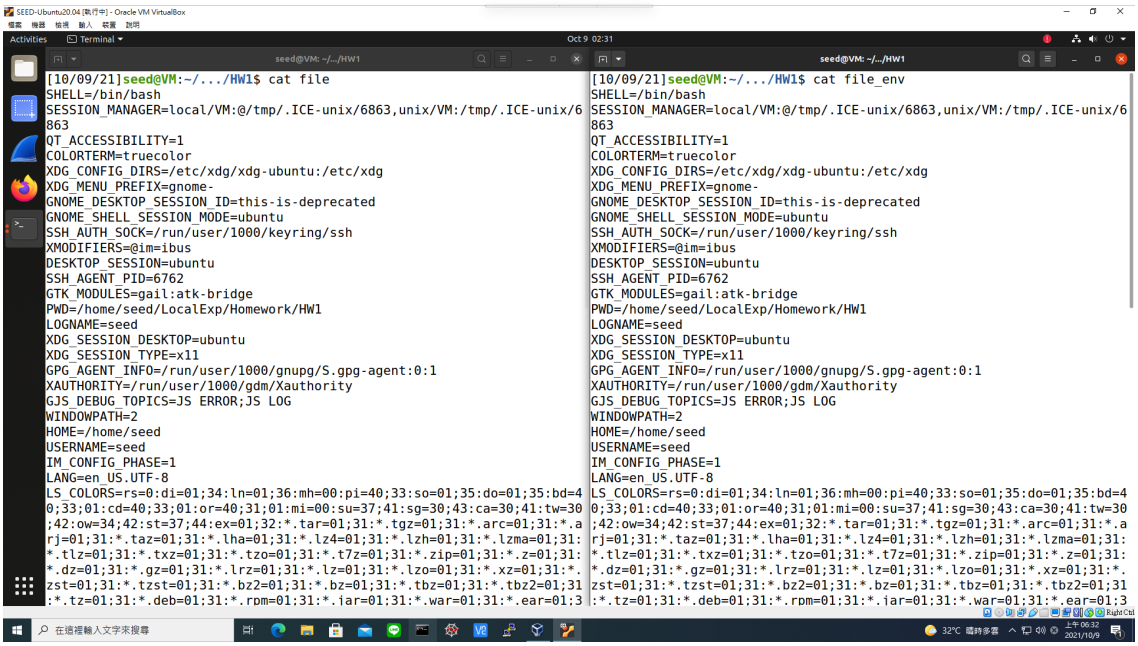

Figure 4: left window shows file contents; right window shows env output

Then the compared result:

```
[10/09/21]seed@VM:~/.../HW1$ diff file file_env
29c29
< GNOME_TERMINAL_SCREEN=/org/gnome/Terminal/screen/5565212c_301e_4bcd_
8e18_b8d854517b61
---
> GNOME_TERMINAL_SCREEN=/org/gnome/Terminal/screen/81097c70_9ef6_4ae0_
9edc_1b80565c62dc
38c38
< GNOME_TERMINAL_SERVICE=:1.249
---
> GNOME_TERMINAL_SERVICE=:1.214
49c49
< _=./myprintenv
---
> _=/usr/bin/env
```

Figure 5: we can igore GNOME_TERMINAL_SCREEN... and GNOME_TERMINAL_SERVICE..., because they mean the different terminals, and _ means the previous execute command

From the result, I found that there's no different between using "myprint.c" and "env" step2. and step3.

```
[10/09/21]seed@VM:~/.../HW1$ diff file1 file2
[10/09/21]seed@VM:~/.../HW1$
```

Figure 6: file1 is step1's result; file2 is step2's results; The result shows nothing, it means that file1 and file2 are same

Task3.
step1. As the picture, it shows nothing for argument is NULL.

```
[10/09/21]seed@VM:~/.../HW1$ gcc myenv.c -o myenv
[10/09/21]seed@VM:~/.../HW1$ ./myenv
[10/09/21]seed@VM:~/.../HW1$
```

Figure 7: There shows nothing when using the argument is NULL

step2. after changing the argument into environ

```
seed@VM: ~/.../HW1
[10/09/21]seed@VM:~/.../HW1$ ./myenv2
SHELL=/bin/bash
SESSION_MANAGER=local/VM:@/tmp/.ICE-unix/6863,unix/VM:/tmp/.ICE-unix/6863
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=6762
GTK_MODULES=gail:atk-bridge
PWD=/home/seed/LocalExp/Homework/HW1
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
```

Figure 8: it shows the env variables correctly

step3. Conclusion:
From step1 and step2, the difference is only the third argument of execve (NULL and environ). When using execve(), we can execute the process (the second argument) with the certain shellscript (the first argument).
The third argument is the environment vector, which means the environment variables. In step1,

we use NULL for env variables, on the other hand, there's no env variable. In step2, we use environ, that include environment variables at the beginning (extern char **environ;)

Task4.

Figure 9: it shows the env variables correctly

The main difference of system() to execve() is whether doing fork(). execve() does the program (first argument) and the command (second argument) with given environment variables; system() does fork() first, and does the command and sends the original environment variables to the child process.

Task5.

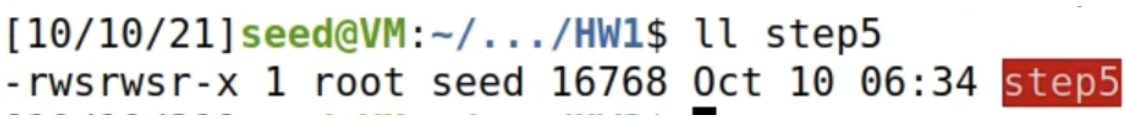

Figure 10: step5 Set-UID first

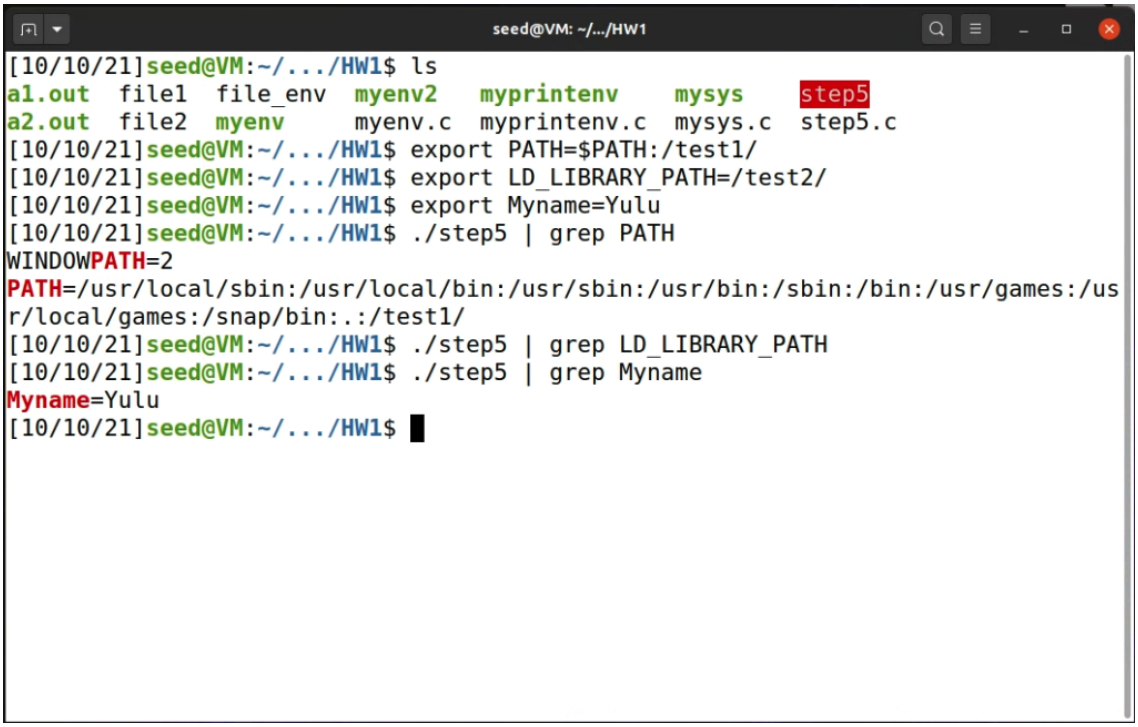

Figure 11: it shows that only LD_LIBRARY_PATH isn't changed

When we execute a program with different owner and executor (root/not-root), dynamic linker will ignore LD_LIBRARY_PATH (and also LD_PRELOAD). So the result shows that I can only edit PATH and Myname env.

Task6. First, I have compile the example code myself and called "myls"
execute it and get results below:

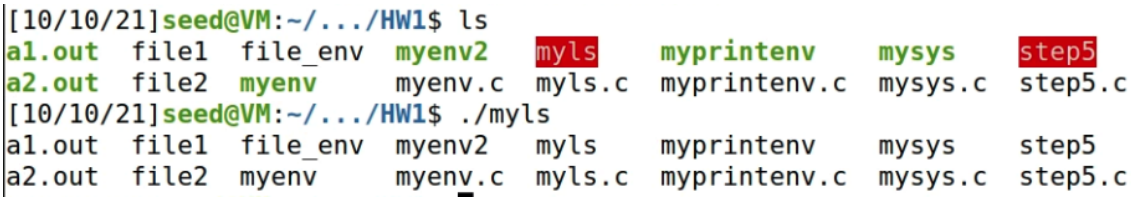

Figure 12: it does only original "ls"

Next, I export a new path (/tmp) in PATH env variable with a file called "ls" copied from "/bin/sh". Then execute "myls" again:

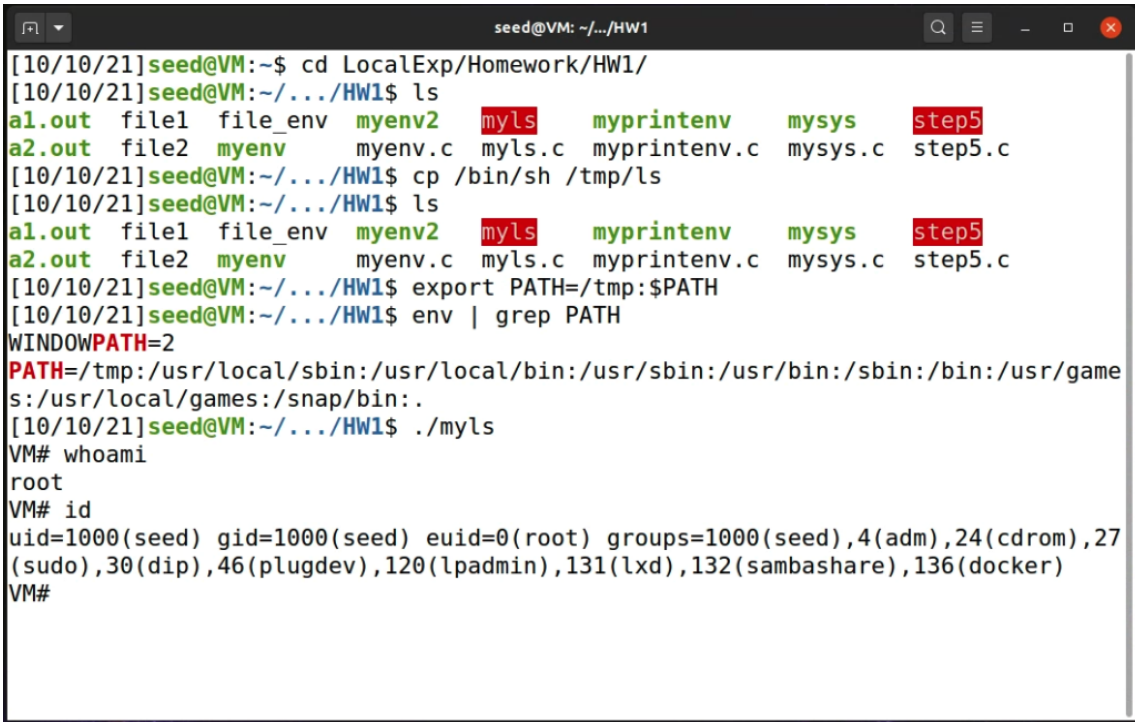

Figure 13

Clearly, it gets the root permission.(I have linked sh to zsh at first)

Task7. Run in normal mode first:

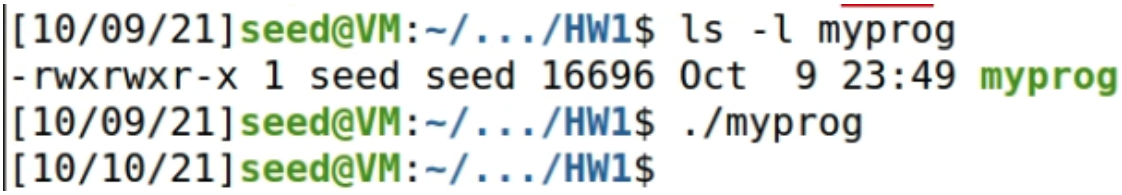

Figure 14: it normally sleeps for 1s

Next, make it a Set-UID program and execute again:

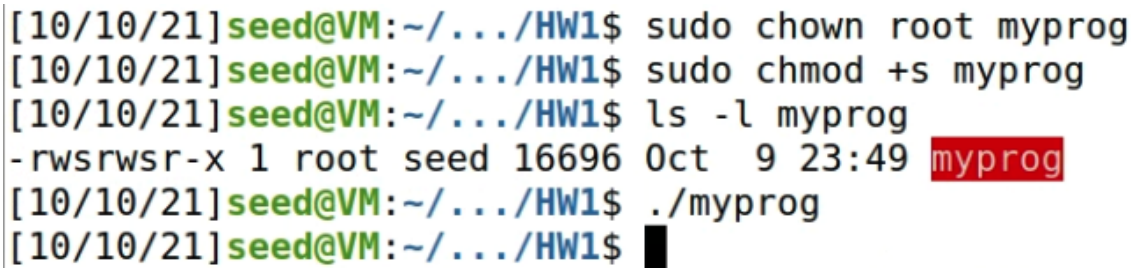

Figure 15: it sleeps for 1s same

Third, export the libmylin.so.1.0.1 in LD_PRELOAD:

```
[10/10/21]seed@VM:~/.../HW1$ export LD_PRELOAD=./libmylib.so.1.0.1
[10/10/21]seed@VM:~/.../HW1$ ls -l myprog
-rwsrwsr-x 1 root seed 16696 Oct  9 23:49 myprog
[10/10/21]seed@VM:~/.../HW1$ ./myprog
[10/10/21]seed@VM:~/.../HW1$
```

Figure 16: it sleeps for 1s same

And also execute it in normal mode:

```
[10/10/21]seed@VM:~/.../HW1$ sudo chown seed myprog
[10/10/21]seed@VM:~/.../HW1$ ls -l myprog
-rwxrwxr-x 1 seed seed 16696 Oct  9 23:49 myprog
[10/10/21]seed@VM:~/.../HW1$ ./myprog
I am not sleeping!!!
[10/10/21]seed@VM:~/.../HW1$
```

Figure 17: it shows that runs incorrectly!

Same as task5, dynamic linker will ignore a program that in different owner and executor, so it runs correctly in Set-UID mode. But in normal mode, we can link our-own lib.
Last, I add a new user called "user1" and run in normal mode and Set-UID mode:

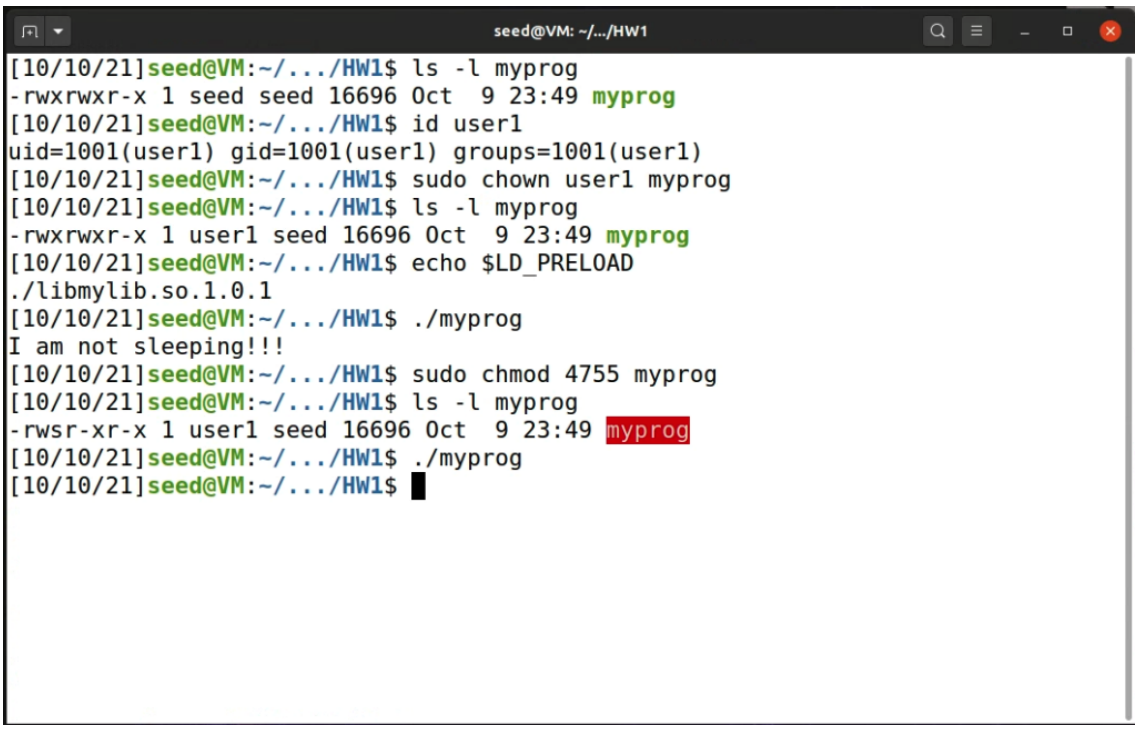

Figure 18

The results show that even we are not the owner, we can run others program in our-own lib if we have their execution permission.

Task8. Here we compare system() and execve()
First, I create a read-only file "secret" contains "SECRET!!!". I can use Set-UID program to remove the secret, which I am not the owner neither writable.

```
[10/10/21]seed@VM:~/.../HW1$ ll secret
-r--r--r-- 1 root seed 10 Oct 10 01:24 secret
[10/10/21]seed@VM:~/.../HW1$ ./catall secret
SECRET!!!
[10/10/21]seed@VM:~/.../HW1$ ./catall "secret;rm secret"
SECRET!!!
[10/10/21]seed@VM:~/.../HW1$ ll secret
ls: cannot access 'secret': No such file or directory
[10/10/21]seed@VM:~/.../HW1$
```

Figure 19: remove the secret with nothing happened

In normal situation, it should not be remove from me, but here it still needs a confirm.

```
[10/10/21]seed@VM:~/.../HW1$ ll secret
-r--r--r-- 1 root seed 10 Oct 10 09:25 secret
[10/10/21]seed@VM:~/.../HW1$ rm secret
rm: remove write-protected regular file 'secret'?
```

Figure 20

Then, we can access in "/bin/sh" although we don't have the "secret".

```
[10/10/21]seed@VM:~/.../HW1$ ./catall "secret;/bin/sh"
/bin/cat: secret: No such file or directory
# whoami
root
# id
uid=1000(seed) gid=1000(seed) euid=0(root) groups=1000(seed),4(adm),24(cdrom),27
(sudo),30(dip),46(plugdev),120(lpadmin),131(lxd),132(sambashare),136(docker)
#
```

Figure 21

But when using execve() to access /bin/sh, I failed.

```
[10/10/21]seed@VM:~/.../HW1$ ll catall_execve
-rwsrwsr-x 1 root seed 16928 Oct 10 09:30 catall_execve
[10/10/21]seed@VM:~/.../HW1$ ./catall_execve "secret;/bin/sh"
/bin/cat: 'secret;/bin/sh': No such file or directory
[10/10/21]seed@VM:~/.../HW1$
```

Figure 22

As above, system() does three things: (1) fork a child. (2) exevce() in child. (3) parent wait until child done. In (2), there starts a shell, so I can do multi-command; also, it's a Set-UID program, so I have the root permission.
But in catall_execve, the second argument is seen as a file name, so the error message says "secret;/bin/sh: No such file or directory" instead of "secret: No such file or directory".

Task9.

```
[10/10/21]seed@VM:~/.../HW1$ ll /etc/zzz
-rw-r--r-- 1 root root 7 Oct 10 10:23 /etc/zzz
[10/10/21]seed@VM:~/.../HW1$ cat /etc/zzz
SECRET
[10/10/21]seed@VM:~/.../HW1$ echo /etc/zzz
/etc/zzz
[10/10/21]seed@VM:~/.../HW1$ echo "test" >> /etc/zzz
bash: /etc/zzz: Permission denied
[10/10/21]seed@VM:~/.../HW1$
```

Figure 23: create a secret file is own to root, the result shows that we are not writable

```
[10/10/21]seed@VM:~/.../HW1$ ./cap_leak
fd is 3
$ echo AAAAAAAA >> /etc/zzz
zsh: permission denied: /etc/zzz
$ echo AAAAAAAA >& 3
$ cat /etc/zzz
SECRET
AAAAAAAA
$ exit
[10/10/21]seed@VM:~/.../HW1$
```

Figure 24: Same as what we did in class, I can write in /etc/zzz

```
[10/10/21]seed@VM:~/.../HW1$ ./cap_leak
fd is 3
$ echo BBBB >& 3
zsh: 3: bad file descriptor
$
```

Figure 25: Here close file after setuid(), then I can't write in "/etc/zzz"

I try another way to write in "/etc/zzz":

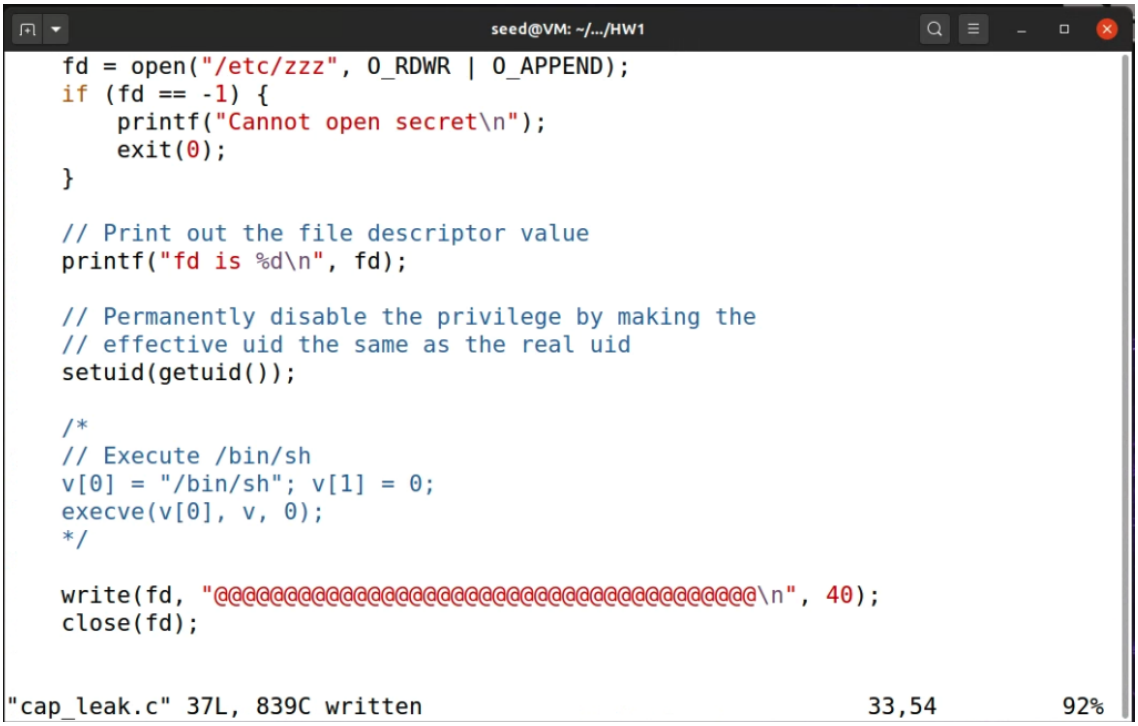

Figure 26

```
[10/10/21]seed@VM:~/.../HW1$ ./cap_leak
fd is 3
[10/10/21]seed@VM:~/.../HW1$ cat /etc/zzz
SECRET
AAAAAAAA
@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
[10/10/21]seed@VM:~/.../HW1$
```

Figure 27: Write in "/etc/zzz" successfully

2. **ping.c (15 points)**

1. After checking icmp all information of each round, I found one thing interesting: why I receive the request packet? And also, why I can receive the same timestamp packet, one for request packet, the other for reply packet?

```
ping 192.168.1.162 (0.0.0.0) : 24 bytes of data.

icmp all info
type: 8; code: 0; checksum: 49057
id: 6288; sequence: 1; timestamp: 1633925353
icmp-type/ICMP_ECHOREPLY: 8 0
ICMP packets are not send by us
unpack() error

icmp all info
type: 0; code: 0; checksum: 49065
id: 6288; sequence: 1; timestamp: 1633925353
icmp-type/ICMP_ECHOREPLY: 0 0
ICMP packets are not send by us
unpack() error

icmp all info
type: 8; code: 0; checksum: 48377
id: 6288; sequence: 2; timestamp: 1633925354
icmp-type/ICMP_ECHOREPLY: 8 0
ICMP packets are not send by us
unpack() error

icmp all info
type: 0; code: 0; checksum: 48385
id: 6288; sequence: 2; timestamp: 1633925354
icmp-type/ICMP_ECHOREPLY: 0 0
ICMP packets are not send by us
unpack() error

icmp all info
type: 8; code: 0; checksum: 46934
id: 6288; sequence: 3; timestamp: 1633925355
"ping_test.log" 38L, 1003C
```

Figure 28: ping's log file

so there are two reasons why it keeps unpack error:
(1) we should receive the packet which id is 0 (reply) instead of 8 (request)
(2) icmp-id is 8 bits, on the other hand, zero for each 65536 rounds, but process id is zero for each 4194304 rounds. There might not be synchronized for a long time.
The number 4194304 is from pid_max ("cat /proc/sys/kernel/pid_max")

```
ping www.google.com (74.125.24.106) : 24 bytes of data.

icmp all info
type: 0; code: 0; checksum: 9214
id: 10254; sequence: 1; timestamp: 1633926248
icmp-type/ICMP_ECHOREPLY: 0 0
icmp-id/getpid(): 10254 141326
ICMP packets are not send by us
unpack() error

icmp all info
type: 0; code: 0; checksum: 42808
id: 10254; sequence: 2; timestamp: 1633897429
icmp-type/ICMP_ECHOREPLY: 0 0
icmp-id/getpid(): 10254 141326
ICMP packets are not send by us
unpack() error

icmp all info
type: 0; code: 0; checksum: 23860
id: 10254; sequence: 3; timestamp: 1633926250
icmp-type/ICMP_ECHOREPLY: 0 0
icmp-id/getpid(): 10254 141326
"ping_test.log" 43L, 1170C
```

Figure 29: ping's log file

2. and 3.
Because ping need to be able to create a raw network, without capabilities, we will need to be set-uid.
After checking the permission of ping (ls -l /bin/ping), it's not a set-uid program, but with capabilities "ep" (effective + permitted):

```
[10/11/21]seed@VM:~/.../2$ ls -l /bin/ping
-rwxr-xr-x 1 root root 72776 Jan 30  2020 /bin/ping
[10/11/21]seed@VM:~/.../2$ getcap /bin/ping
/bin/ping = cap_net_raw+ep
```

Figure 30

3. **setuid and seteuid (15 points)**
Here I design a code for setuid() and seteuid():

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

int main() {
    printf("Real user id: %d, Effective user id: %d\n", getuid(), geteuid());
    setuid(1200);
    //seteuid(1200);
    printf("Real user id: %d, Effective user id: %d\n", getuid(), geteuid());
    setuid(1000);
    //seteuid(1000);
    printf("Real user id: %d, Effective user id: %d\n", getuid(), geteuid());
    return 0;
}
```

Figure 31

Then compile it and make it set-uid and run:

```
[10/11/21]seed@VM:~/.../3$ gcc set.c -o set
[10/11/21]seed@VM:~/.../3$ sudo chown root set
[10/11/21]seed@VM:~/.../3$ sudo chmod +s set
[10/11/21]seed@VM:~/.../3$ ll set
-rwsrwsr-x 1 root seed 16832 Oct 11 14:24 set
[10/11/21]seed@VM:~/.../3$ ./set
Real user id: 1000, Effective user id: 0
Real user id: 1200, Effective user id: 1200
Real user id: 1200, Effective user id: 1200
```

Figure 32

The result shows that once it becomes not root uid, it can't change to other uid, because it's no more root.
Next, I modify the code (comment out setuid() and un-comment seteuid()), and do as above:

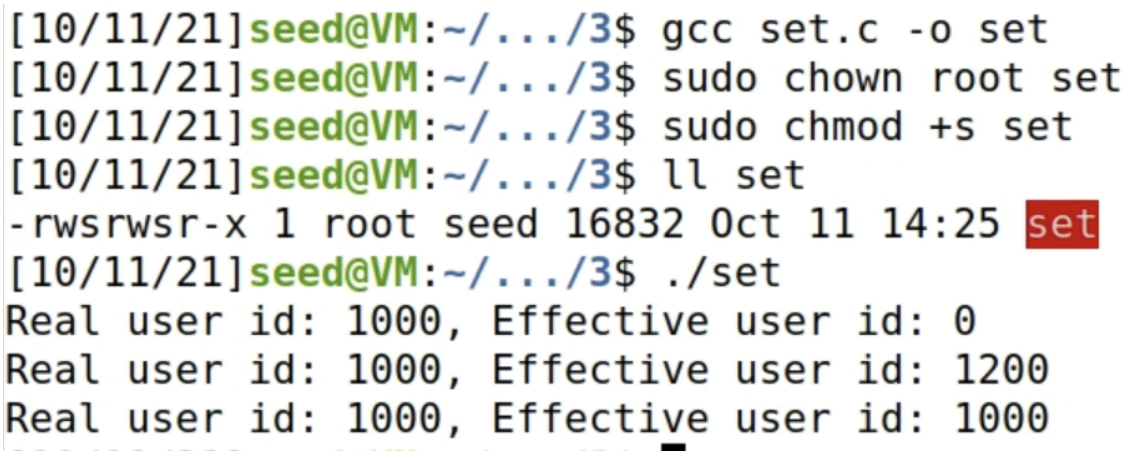

Figure 33

The result shows that it only changes euid instead of changes both, so when it wants to change to other euid, it is still a root and successfully changes.

4. **execve (15 points)**
   here I design a shellscript to show the environment variables in new shell: (with title line: "#! /bin/sh")

```
#! /usr/bin/sh

echo "Starting script . . ."

<<com
echo "Command line parameters:"

for var in $@
    do
        echo "    $var"
    done

echo "Environment variables:"
echo "    SHELL=$SHELL"
echo "    USER=$USER"
echo "    TERM=$TERM"
echo "    Myname=$Myname"
com

echo "Show env:"
env
echo "++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++"

echo "End of script"
```

Figure 34

And this is myexecve.c with no environment variable input:

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

extern char **environ;
int main() {
    char cmd[] = "./userscript.sh";
    char *argv[] = {"userscript", /*"arg1", "arg2", "3",*/ NULL};
    char *envp[] = {NULL};

    printf("Start of execve call %s:\n", cmd);
    printf("========================================\n");


    execve(cmd , argv, envp);
    //printf("Debug here!\n");

    return 0;
}
```

Figure 35

And the result is as below:

```
[10/11/21]seed@VM:~/.../4$ run myexecve.c
Start of execve call ./userscript.sh:
========================================
Starting script . . .
Show env:
LOGNAME=seed
SHLVL=1
PWD=/home/seed/LocalExp/Homework/HW1/4
OLDPWD=/home/seed/LocalExp/Homework/HW1/4
_=/bin/env
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
End of script
[10/11/21]seed@VM:~/.../4$
```

Figure 36: The environment variables are inherited

The result shows that there are five environment variables inherited from parent process (LOGNAME, SHLVL, PWD, OLDPWD, _)

But if I delete the title line, here comes nothing:

```
[10/15/21]seed@VM:~/.../4$ run myexecve.c
Start of execve call ./userscript.sh:
========================================
[10/15/21]seed@VM:~/.../4$
```

Figure 37

5. **ld-linux (15 points)**
LD_AUDIT is a variable that load before everything, even bfore LD_PRELOAD:

```
[10/11/21]seed@VM:~/.../5$ LD_PRELOAD=./preloadlib.so.1.0.1 LD_AUDIT=./auditlib.
so.1.0.1 whoami
I'm loaded from LD_AUDIT
I'm loaded from LD_PRELOAD
seed
```

Figure 38

For preventing malicious preload, we can disable it; we can also detect it if it is running:
(But here I don't have the running malicious program...)
(ref: *Leveraging LD_AUDIT to Beat the Traditional Linux Library Preloading Technique*)

```
[10/11/21]seed@VM:~/.../5$ ./myprog
[10/11/21]seed@VM:~/.../5$ export LD_PRELOAD=./mysleep.so.1.0.1
[10/11/21]seed@VM:~/.../5$ ./myprog
I am not sleeping!!!
[10/11/21]seed@VM:~/.../5$ export LD_AUDIT=./audit_sleep.so.1.0.1
Disabling the loading of a 'preload' library: ./mysleep.so.1.0.1
ERROR: ld.so: object './mysleep.so.1.0.1' from LD_PRELOAD cannot be preloaded (c
annot open shared object file): ignored.
[10/11/21]seed@VM:~/.../5$ ./myprog
Disabling the loading of a 'preload' library: ./mysleep.so.1.0.1
ERROR: ld.so: object './mysleep.so.1.0.1' from LD_PRELOAD cannot be preloaded (c
annot open shared object file): ignored.
Disabling the loading of a 'preload' library: ./mysleep.so.1.0.1
ERROR: ld.so: object './mysleep.so.1.0.1' from LD_PRELOAD cannot be preloaded (c
annot open shared object file): ignored.
[10/11/21]seed@VM:~/.../5$
```

Figure 39

So if LD_AUDIT can load before everything, I think attack can be forcing an user to preload a malicious library
LD_DEBUG_OUTPUT: