Jorge Avila (**1001543128**)

Professor Trey Jones

Secure Programming

12 February 2021

Environment Variable and SET-UID

Task 1: Manipulating Environment Variables

The description of the command **printenv** is that it prints the values of the specified environment variable(s). If no variable is specified, print name and value pairs for them all. An example of the tasks given are shown below.

```
[02/03/21]seed@VM:~$ printenv PWD
/home/seed
[02/03/21]seed@VM:~$
```

When doing the other format: **env | grep PWD** the terminal yields as below.

```
[02/03/21]seed@VM:~$ env | grep PWD
PWD=/home/seed
[02/03/21]seed@VM:~$
```

The final commands for this section are **export** and **unset**. Export is a command that will display all the exported variables. A sample of the exportation is shown below.

```
[02/03/21]seed@VM:~$ export
declare -x ANDROID_HOME="/home/seed/android/android-sdk-linux"
declare -x CLUTTER_IM_MODULE="xim"
declare -x COMPIZ_BIN_PATH="/usr/bin/"
declare -x COMPIZ_CONFIG_PROFILE="ubuntu"
declare -x DBUS_SESSION_BUS_ADDRESS="unix:abstract=/tmp/dbus-OYA7zFHmzm"
declare -x DEFAULTS_PATH="/usr/share/gconf/ubuntu.default.path"
declare -x DERBY_HOME="/usr/lib/jvm/java-8-oracle/db"
declare -x DESKTOP_SESSION="ubuntu"
declare -x DISPLAY=":0"
declare -x GDMSESSION="ubuntu"
declare -x GDM_LANG="en_US"
declare -x GNOME_DESKTOP_SESSION_ID="this-is-deprecated"
declare -x GNOME_KEYRING_CONTROL=""
declare -x GNOME_KEYRING_PID=""
declare -x GPG_AGENT_INFO="/home/seed/.gnupg/S.gpg-agent:0:1"
declare -x GTK2_MODULES="overlay-scrollbar"
declare -x GTK_IM_MODULE="ibus"
declare -x GTK_MODULES="gail:atk-bridge:unity-gtk-module"
declare -x HOME="/home/seed"
declare -x IM_CONFIG_PHASE="1"
declare -x INSTANCE=""
declare -x J2REDIR="/usr/lib/jvm/java-8-oracle/jre"
declare -x J2SDKDIR="/usr/lib/jvm/java-8-oracle"
declare -x JAVA_HOME="/usr/lib/jvm/java-8-oracle"
declare -x JOB="unity-settings-daemon"
declare -x LANG="en_US.UTF-8"
declare -x LANGUAGE="en_US"
declare -x
LD_LIBRARY_PATH="/home/seed/source/boost_1_64_0/stage/lib:/home/seed/source/boost_
1_64_0/stage/lib:"
declare -x
LD_PRELOAD="/home/seed/lib/boost/libboost_program_options.so.1.64.0:/home/seed/lib/bo
ost/libboost_filesystem.so.1.64.0:/home/seed/lib/boost/libboost_system.so.1.64.0"
declare -x LESSCLOSE="/usr/bin/lesspipe %s %s"
declare -x LESSOPEN="| /usr/bin/lesspipe %s"
declare -x LOGNAME="seed"
declare -x
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:*.Z=01;31:*.dz=01;31:*.gz=01;31:*.lrz=01;31:*.lz=01;31:*.lzo=01;31:*.xz=01;31:*.b
z2=01;31:*.bz=01;31:*.tbz=01;31:*.tbz2=01;31:*.tz=01;31:*.deb=01;31:*.rpm=01;31:*.jar=0
1;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:*.jpg=01;35:*.jpeg=01;35:*.gif=01;35:*.bm
p=01;35:*.pbm=01;35:*.pgm=01;35:*.ppm=01;35:*.tga=01;35:*.xbm=01;35:*.xpm=01;35:*.t
```

```
if=01;35:*.tiff=01;35:*.png=01;35:*.svg=01;35:*.svgz=01;35:*.mng=01;35:*.pcx=01;35:*.m
ov=01;35:*.mpg=01;35:*.mpeg=01;35:*.m2v=01;35:*.mkv=01;35:*.webm=01;35:*.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:*.o
gv=01;35:*.ogx=01;35:*.aac=00;36:*.au=00;36:*.flac=00;36:*.m4a=00;36:*.mid=00;36:*.mi
di=00;36:*.mka=00;36:*.mp3=00;36:*.mpc=00;36:*.ogg=00;36:*.ra=00;36:*.wav=00;36:*.og
a=00;36:*.opus=00;36:*.spx=00;36:*.xspf=00;36:"
declare -x MANDATORY_PATH="/usr/share/gconf/ubuntu.mandatory.path"
declare -x OLDPWD
declare -x
PATH="/home/seed/bin:/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin:/usr/games:/u
sr/local/games:.:/snap/bin:/usr/lib/jvm/java-8-oracle/bin:/usr/lib/jvm/java-8-oracle/db/bin:/usr/l
ib/jvm/java-8-oracle/jre/bin:/home/seed/android/android-sdk-linux/tools:/home/seed/android/a
ndroid-sdk-linux/platform-tools:/home/seed/android/android-ndk/android-ndk-r8d:/home/seed/
.local/bin"
declare -x PWD="/home/seed"
declare -x QT4_IM_MODULE="xim"
declare -x QT_ACCESSIBILITY="1"
declare -x QT_IM_MODULE="ibus"
declare -x QT_LINUX_ACCESSIBILITY_ALWAYS_ON="1"
declare -x QT_QPA_PLATFORMTHEME="appmenu-qt5"
declare -x SESSION="ubuntu"
declare -x SESSIONTYPE="gnome-session"
declare -x SHELL="/bin/bash"
declare -x SHLVL="1"
declare -x SSH_AUTH_SOCK="/run/user/1000/keyring/ssh"
declare -x TERM="xterm-256color"
declare -x UPSTART_EVENTS="xsession started"
declare -x UPSTART_INSTANCE=""
declare -x UPSTART_JOB="unity7"
declare -x UPSTART_SESSION="unix:abstract=/com/ubuntu/upstart-session/1000/1337"
declare -x USER="seed"
declare -x VTE_VERSION="4205"
declare -x WINDOWID="54525962"
declare -x XAUTHORITY="/home/seed/.Xauthority"
declare -x XDG_CONFIG_DIRS="/etc/xdg/xdg-ubuntu:/usr/share/upstart/xdg:/etc/xdg"
declare -x XDG_CURRENT_DESKTOP="Unity"
declare -x
XDG_DATA_DIRS="/usr/share/ubuntu:/usr/share/gnome:/usr/local/share/:/usr/share/:/var/lib/
snapd/desktop"
declare -x XDG_GREETER_DATA_DIR="/var/lib/lightdm-data/seed"
declare -x XDG_RUNTIME_DIR="/run/user/1000"
declare -x XDG_SEAT="seat0"
declare -x XDG_SEAT_PATH="/org/freedesktop/DisplayManager/Seat0"
declare -x XDG_SESSION_DESKTOP="ubuntu"
declare -x XDG_SESSION_ID="c1"
declare -x XDG_SESSION_PATH="/org/freedesktop/DisplayManager/Session0"
declare -x XDG_SESSION_TYPE="x11"
declare -x XDG_VTNR="7"
declare -x XMODIFIERS="@im=ibus"
declare -x ls
[02/03/21]seed@VM:~$
```

Better yet said: "**Exporting**" a variable in the shell makes it available to all subshells **and** processes created by that shell. It **does** not make it available everywhere in the system, only by processes created from that shell. (AskUbuntu.com).

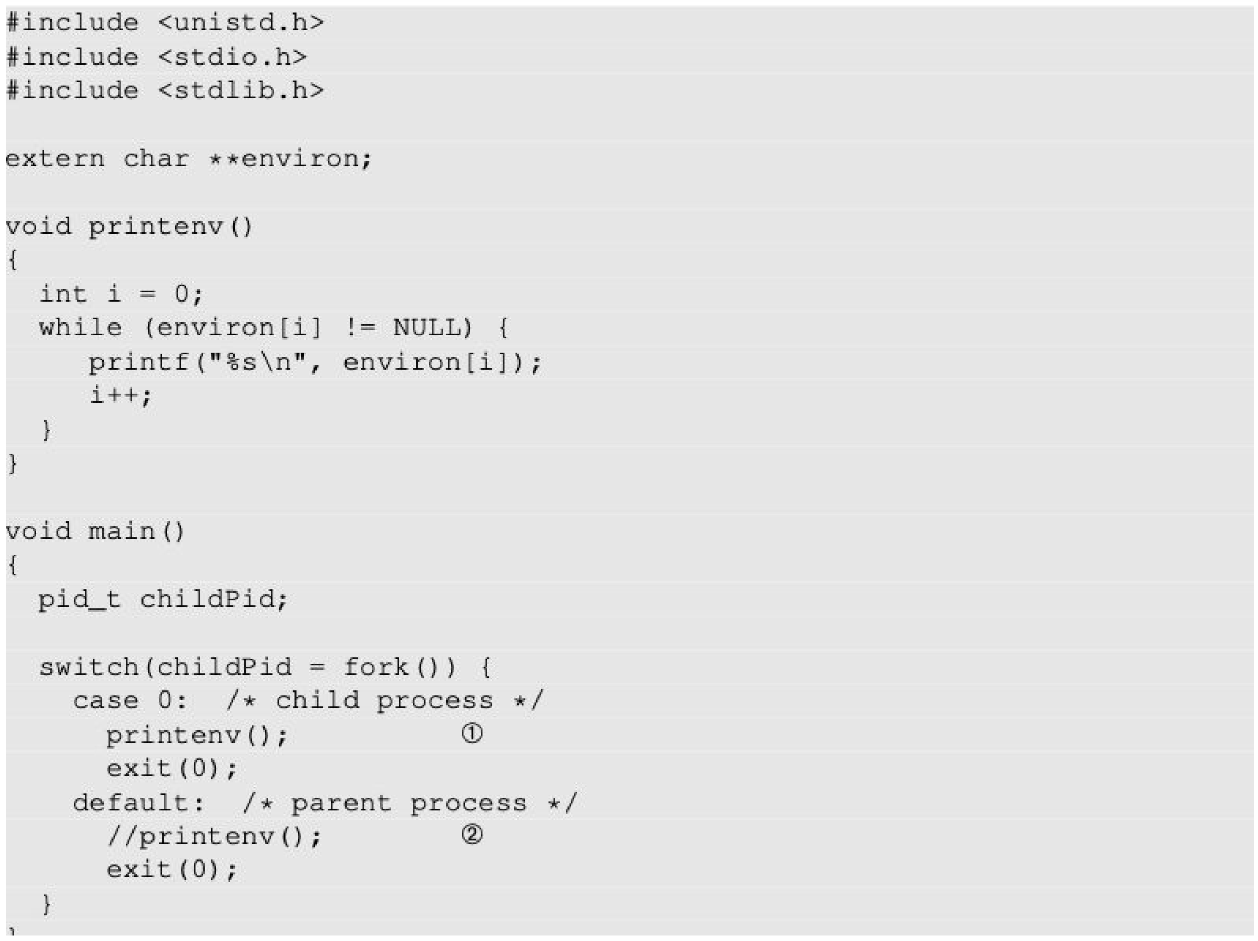
Unsetting or setting environmental variables are done by using the **set** command. When trying to clear variables. An example is shown below. (youtube link on export/set/unset)

```
[02/03/21]seed@VM:~$ myvar=10
[02/03/21]seed@VM:~$ myvar
myvar: command not found
[02/03/21]seed@VM:~$ export myvar
[02/03/21]seed@VM:~$ env | grep myvar
myvar=10
[02/03/21]seed@VM:~$ unset myvar
[02/03/21]seed@VM:~$ env | grep myvar
[02/03/21]seed@VM:~$
```

You can tell that in the last line, nothing was shown since I unset the variable.

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

Given the code below. We are to use **fork()** to create a child process from the parent process, which is an exact duplicate, but some things are not inherited such as *PIDs* (Process Identification).



```
#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);
  }
}
```

After commenting the **printenv()** in the child process and uncommenting the **printenv()** in the parent process, it is visible that everything seems to be printed out twice. In this link, you can see that it is all the same, until it seems to be duplicated again. Using the **diff** command you receive the output as **70c70** meaning that that is the same up until the right side of the 'c'.

Task 3: Environment Variables and `execve():`

Running the below code:

```
[02/03/21]seed@VM:~$
#include <stdio.h>
#include <stdlib.h>
#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;
}
```

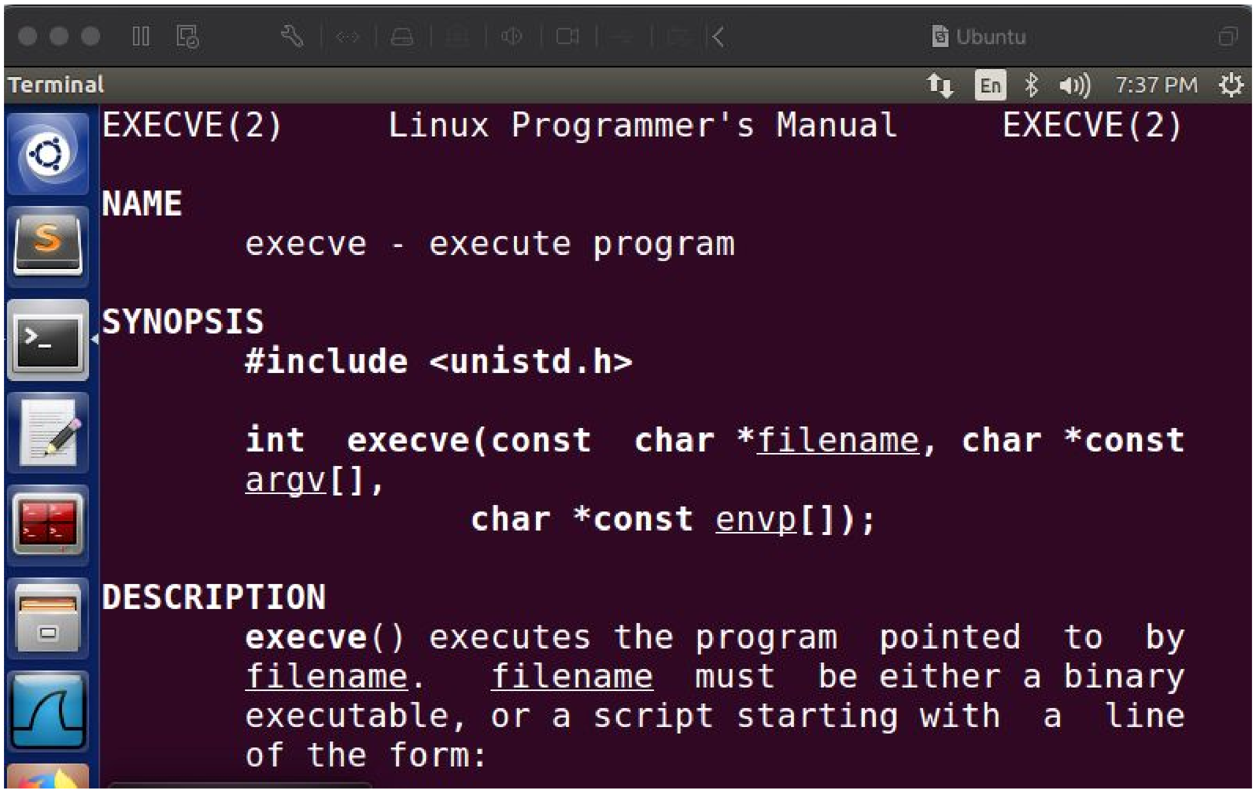
Gave out a null, blank output with no data. However when changing the invocation of execve():

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

to

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

The console then printed the environment variables. The explanation to this is the third parameter being passed to the execve is NULL and the other actually has the global environment. If you refer to **man execve** you can see that the third parameter is the environment known as **`*const envp[]`.**



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

NAME
       execve - execute program

SYNOPSIS
       #include <unistd.h>

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

DESCRIPTION
       execve() executes the program  pointed  to  by
       filename.   filename  must  be either a binary
       executable, or a script starting with  a  line
       of the form:
```

Task 4: Environment Variables and `system():`

The way this works is as opposed to execve() which runs the env command directly, system() first asks "/usr/bin/env", then executes it. Another method I tested this by typing in the command terminal interface the following, which verifies the given C code.

```
[02/03/21]seed@VM:~$ /bin/sh
$ env
   ●  The whole variables are then printed here
```

Task 5: Environment Variable and `Set-UID` **Programs:**

After running the code in step 1, the output illustrated the environment variables as usual from as before. After changing the ownership to root and making it a Set-UID program, the following occurred:

```
[02/03/21]seed@VM:~$ sudo chown root t5.c
[02/03/21]seed@VM:~$ sudo chmod 4755 t5.c
[02/03/21]seed@VM:~$ printenv PATH
/home/seed/bin:/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbi
n:/bin:/usr/games:/usr/local/games:.:/snap/bin:/usr/lib/jvm/java-8-or
acle/bin:/usr/lib/jvm/java-8-oracle/db/bin:/usr/lib/jvm/java-8-oracle
/jre/bin:/home/seed/android/android-sdk-linux/tools:/home/seed/androi
d/android-sdk-linux/platform-tools:/home/seed/android/android-ndk/and
roid-ndk-r8d:/home/seed/.local/bin
[02/03/21]seed@VM:~$ env | grep PATH
LD_LIBRARY_PATH=/home/seed/source/boost_1_64_0/stage/lib:/home/seed/s
ource/boost_1_64_0/stage/lib:
XDG_SESSION_PATH=/org/freedesktop/DisplayManager/Session0
XDG_SEAT_PATH=/org/freedesktop/DisplayManager/Seat0
DEFAULTS_PATH=/usr/share/gconf/ubuntu.default.path
PATH=/home/seed/bin:/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin
:/sbin:/bin:/usr/games:/usr/local/games:.:/snap/bin:/usr/lib/jvm/java
-8-oracle/bin:/usr/lib/jvm/java-8-oracle/db/bin:/usr/lib/jvm/java-8-o
racle/jre/bin:/home/seed/android/android-sdk-linux/tools:/home/seed/a
ndroid/android-sdk-linux/platform-tools:/home/seed/android/android-nd
k/android-ndk-r8d:/home/seed/.local/bin
```

```
MANDATORY_PATH=/usr/share/gconf/ubuntu.mandatory.path
COMPIZ_BIN_PATH=/usr/bin/
[02/03/21]seed@VM:~$ export jorge=/home/seed
[02/03/21]seed@VM:~$
```

And re-running the program again I noticed that the variable that I created was inherited and actually exported so that other shells could use it as well (*instead of locally in a sense*).

```
[02/03/21]seed@VM:~$ env | grep jorge
jorge=/home/seed
```

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

In this task I had to change the ownership and make it a Set-UID program, once I did that by using the following:

```
$ sudo chown root t6.c
$ sudo chmod 4755 t6.c
```

The following happened when I printed the information on that file

```
[02/03/21]seed@VM:~$ ls -l t6.c
-rwsr-xr-x 1 root seed 65 Feb  3 20:19 t6.c
[02/03/21]seed@VM:~$
```

This illustrates how it is possible to modify the PATH environment variable to point to a desired folder and run user-defined programs that may be malicious. It's risky because of the inclusion of shell and environment variables because we use the function **system()**. We have defined the relative direction of the method instead of defining the absolute path, because of this, a shell is spawned by the **system()** which looks for a "ls" program that I have created at the position defined by the PATH environment variable.

In order to reverse what we did we use then the following commands:

```
$ sudo rm /bin/sh
$ sudo ln -s /bin/zsh /bin/sh
```

Task 7: The LD_PRELOAD Environment VAriable and `Set-UID` programs:

mylib.c is below:

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

Then compiling the above program using the lines below:

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

Once that is all done, we run the following exportation:

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

Now once compiling the **`myprog.c`** that contains the following:

```
//my prog
void int main (void)
{
   sleep(1);
   return 0;
}
```

When running **myprog.c** as a regular usar with no modifications, it defaults to the message we wrote and then finishes. Once I make **myprog.c** a Set-UID root program and run it as a normal user it happens where it is now using the **sleep()** function that we did not create. Once we export the LD_PRELOAD environment variable again in the root account and run it, the code executed is the one that we created since we are in the root and we are able to allow other shells to inherit the environment variable. Hence, the **LD_PRELOAD** is present in the other shells. That means that the process is the effective and real user ID as we talked about in class. Finally setting the Set-ID to a different name and exporting the LD_PRELOAD environment variable made the code run the user-defined library once again. We can see that the first,third and fourth time, the user defined code was executed, but the second time it was not since the **euid** and the **ruid** were not the same. Also, we can tell that LD_PRELOAD was not inherited by the child processes.

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

Running the code below:

```
#include <string.h>
#include <stdio.h>
#include <stdlib.h>
#include <unistd.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 ;
}
```

Which just shows the file that you are passing since **/bin/cat** shows the contents of a file. In this step we are trying to invoke the command that uses ***system()***. The question is now if I was Bob (another user), could I compromise the integrity of the system (i.e. remove a file that is not writable to me?) After doing the following:

```
02/03/21]seed@VM:~$ gcc t8.c
[02/03/21]seed@VM:~$ sudo chown root ./a.out
[02/03/21]seed@VM:~$ sudo chmod 4755 ./a.out
[02/03/21]seed@VM:~$ ll
total 1772
drwxrwxr-x 4 seed seed    4096 May  1  2018 android
-rwsr-xr-x 1 root seed    7544 Feb  3 23:02 a.out
drwxrwxr-x 2 seed seed    4096 Jan 14  2018 bin
drwxrwxr-x 2 seed seed    4096 Jan 14  2018 Customization
drwxr-xr-x 2 seed seed    4096 Jul 25  2017 Desktop
-rw-rw-r-- 1 seed seed    8234 Feb  3 18:46 difference
drwxr-xr-x 2 seed seed    4096 Jul 25  2017 Documents
drwxr-xr-x 2 seed seed    4096 Jan 30 18:44 Downloads
-rw-r--r-- 1 seed seed    8980 Jul 25  2017 examples.desktop
-rw-rw-r-- 1 seed seed 1661676 Jan  2  2019 get-pip.py
drwxrwxr-x 3 seed seed    4096 May  9  2018 lib
-rwxrwxr-x 1 seed seed    7912 Feb  3 21:27 libmylib.so.1.0.1
-rw-rw-r-- 1 seed seed     121 Feb  3 21:01 ls.c
drwxr-xr-x 2 seed seed    4096 Jul 25  2017 Music
-rw-rw-r-- 1 seed seed     154 Feb  3 21:27 mylib.c
-rw-rw-r-- 1 seed seed    2572 Feb  3 21:27 mylib.o
-rwsr-xr-x 1 root seed      38 Feb  3 21:32 myprog.c
drwxr-xr-x 3 seed seed    4096 Jan 14  2018 Pictures
drwxr-xr-x 2 seed seed    4096 Jul 25  2017 Public
drwxrwxr-x 4 seed seed    4096 May  9  2018 source
-rw-rw-r-- 1 seed seed     489 Feb  3 22:52 t8.c
drwxr-xr-x 2 seed seed    4096 Jul 25  2017 Templates
-rw-rw-r-- 1 seed seed      24 Feb  3 22:53 test.txt
-rw-rw-r-- 1 seed seed    8234 Feb  3 18:46 third
drwxr-xr-x 2 seed seed    4096 Jul 25  2017 Videos
[02/03/21]seed@VM:~$
```

So after some investigating and messing with the terminal as shown below, you can tell there was a vulnerability and indeed Bob can change files that he has no permission.

```
[02/03/21]seed@VM:~$ ./a.out t8.c; /bin/sh
```

```
#include <string.h>
#include <stdio.h>
#include <stdlib.h>
#include <unistd.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 ;
}
-------------------------------------------------------------------
[02/03/21]seed@VM:~$ ls
android        examples.desktop   myprog.c  t8.c
a.out          get-pip.py         Pictures  Templates
bin            lib                Public    test.txt
Customization  libmylib.so.1.0.1  source    third
Desktop        ls.c               t3.c      Videos
difference     Music              t4.c
Documents      mylib.c            t5.c
Downloads      mylib.o            t7.c
$ rm t7.c
$ exit
[02/03/21]seed@VM:~$ ls
android        examples.desktop   myprog.c  t8.c
a.out          get-pip.py         Pictures  Templates
bin            lib                Public    test.txt
```

```
Customization  libmylib.so.1.0.1  source    third
Desktop        ls.c               t3.c      Videos
difference     Music              t4.c
Documents      mylib.c            t5.c
Downloads      mylib.o            t6.c
[02/03/21]seed@VM:~$
```

Essentially, attaching to the back of the command line arguments and making **system()** run directly the /bin/sh and then allowing to *rm* (remove) a file, which in my case t7.c was then removed.

In this second step of the lab uncommenting the **execve()** and commenting the **system()** produces the same output and the same Set-UID modifications. My observations now are that you cannot break into it as easily as we did when using the previous functions. The reason why is because there is some protection in execve. This function does not parse it for you, instead says that there is an error finding it rather than the **system()** function we previously used.

Task 9: Capability Leaking:

```
#include <stdio.h>
#include <stdlib.h>
#include <fcntl.h>
void main()
{
    int fd;


/* Assume that /etc/zzz is an important system file,
 * and it is owned by root with permission 0644.
* Before running this program, you should creat
 * the file /etc/zzz first. */
fd = open("/etc/zzz", O_RDWR | O_APPEND);
if (fd == -1) {
   printf("Cannot open /etc/zzz\n");
   exit(0);
}
/* Simulate the tasks conducted by the program */
sleep(1);
/* After the task, the root privileges are no longer needed,
it's time to relinquish the root privileges permanently. */
setuid(getuid());  /* getuid() returns the real uid */
if (fork()) { /* In the parent process */
  close (fd);
  exit(0);
} else { /* in the child process */
  /* Now, assume that the child process is compromised, malicious
     attackers have injected the following statements
     into this process */
  write (fd, "Malicious Data\n", 15);
close (fd); }
}
```

This was the given code above, after running it and changing the appropriate permissions. I see that the code was successful and you can see that “Malicious data” was presented due to the files not being closed in time. Like in class today, Professor Jones stated that many of the reasons that these vulnerabilities occur are due to simple things such as not being able to close a file before doing other things to it. Hence why we need to drop privileges and close specific file pointers.