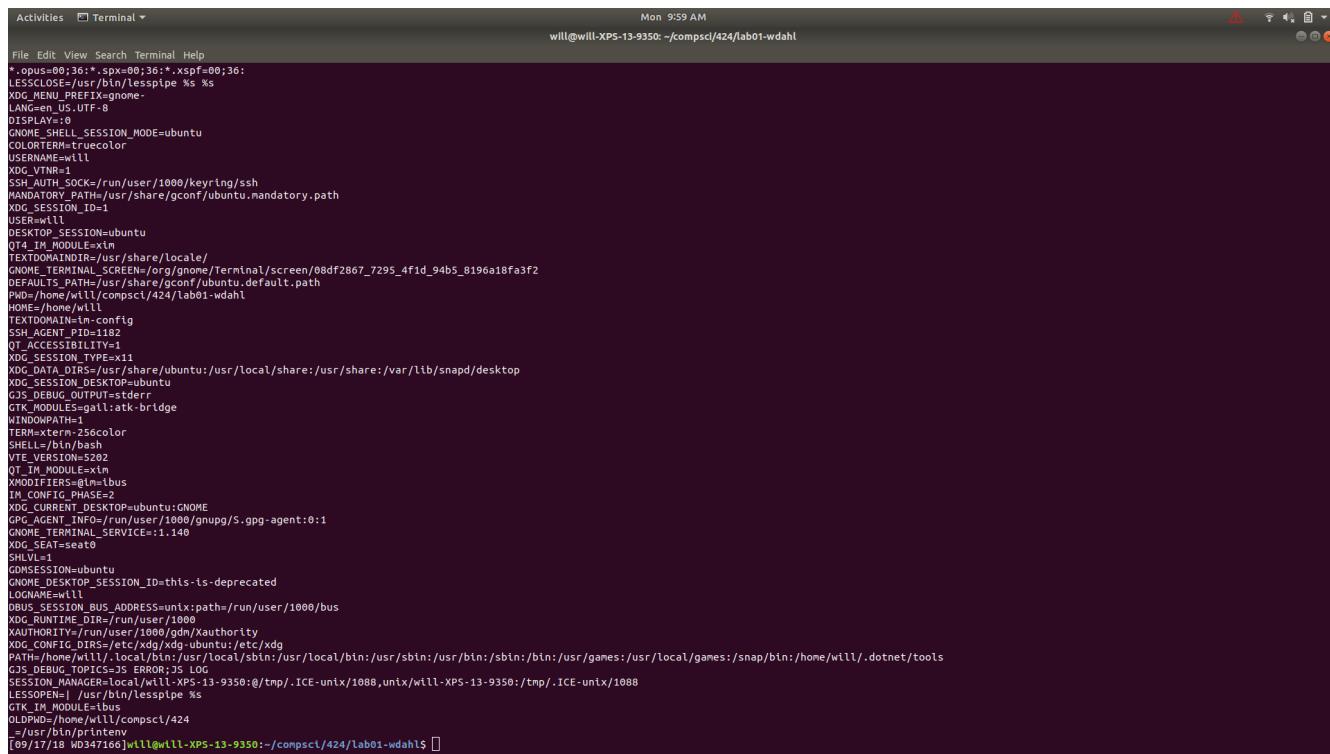


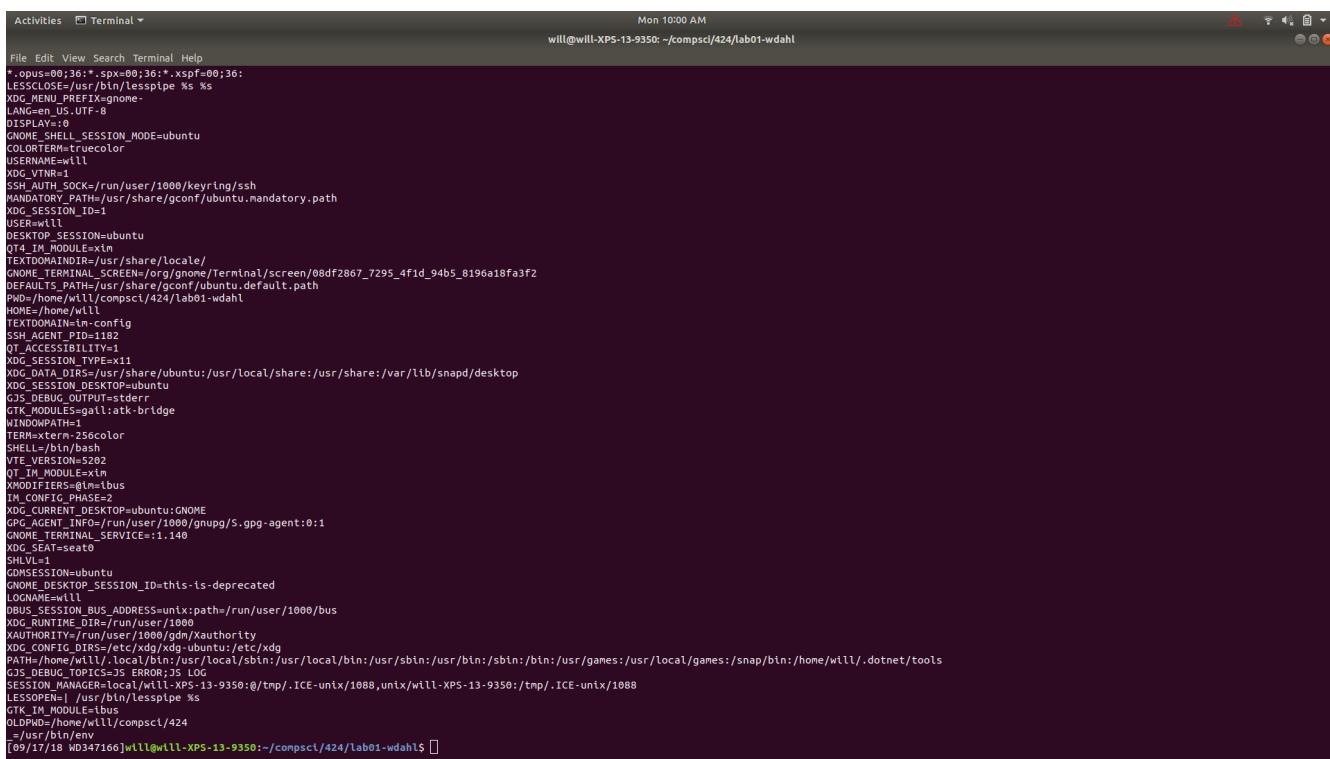
William Dahl
Professor Amir Masoumzadehtork
ICSI 424 Information Security – Lab 01
9/17/2018

Task 1: In this screenshot I ran printenv in my bash prompt which printed out all of my environmental variables



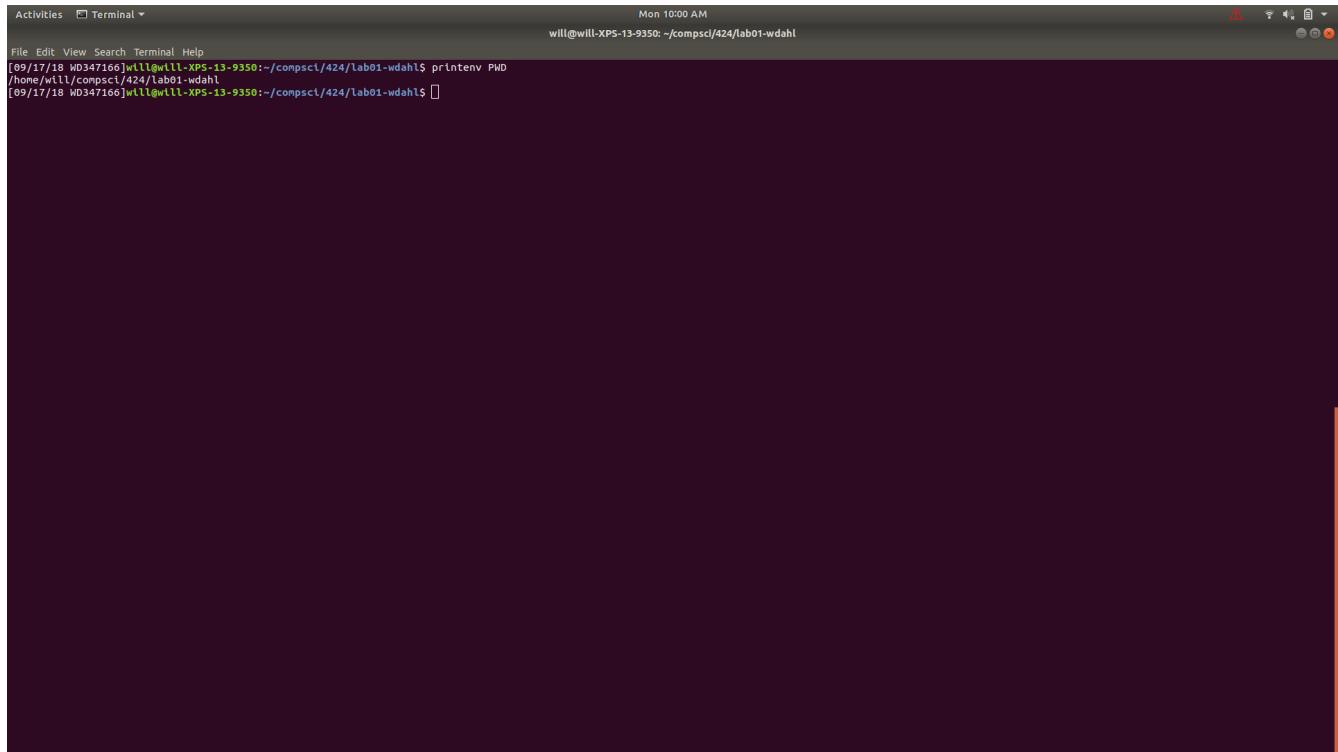
```
Activities Terminal Mon 9:59 AM will@will-XPS-13-9350: ~/compsc1/424/lab01-wdahl
File Edit View Search Terminal Help
*.opus=0;36.*.spx=0;36.*.xspf=0;36;
LESSCLOSE=/usr/bin/lesspipe %s %
XDG_MENU_PREFIX=gnome-
LANG=en_US.UTF-8
DISPLAY=:0
DISPLAY=:0
GNOME_SHELL_SESSION_MODE=ubuntu
COLORTERM=truecolor
USERNAME=will
XDG_VTNR=1
SSH_AUTH_SOCK=/run/user/1000/keyring/ssh
MANDATORY_PATH=/usr/share/gconf/ubuntu.mandatory.path
XDG_SESSION_ID=1
USER=will
DESKTOP_SESSION=ubuntu
QT_IM_MODULE=xim
TEXTDOMAIN=im-config
SSH_AGENT_PID=1182
QT_ACCESSIBILITY=1
XDG_SESSION_TYPE=x11
XDG_DATA_DIRS=/usr/share/ubuntu:/usr/local/share:/usr/share:/var/lib/snapd/desktop
XDG_SESSION_DESKTOP=ubuntu
GJS_DEBUG_OUTPUT=stderr
GTK_MODULES=gail:atk-bridge
WINDOWPATH=1
TERM=xterm-256color
SHELL=/bin/bash
VTE_VERSION=5202
QT_IM_MODULE=xim
XMODIFIERS=@imibus
IM_CONFIG_PHASE=2
XDG_CURRENT_DESKTOP=GNOME
GPG_AGENT_INFO=/run/user/1000/gnupg/S.gpg-agent:0:1
GNOME_TERMINAL_SERVICE=:1.140
XDG_SEAT=seat0
SHLVL=1
GDMSESSION=ubuntu
GNOME_DESKTOP_SESSION_ID=this-is-deprecated
LOGNAME=will
DBUS_SESSION_BUS_ADDRESS=unix:path=/run/user/1000/bus
XDG_RUNTIME_DIR=/run/user/1000
XAUTHORITY=/run/user/1000/gdm/Xauthority
XDG_CONFIG_DIRS=/etc/xdg/xdg-ubuntu:/etc/xdg
PATH=/home/will/.local/bin:/bin:/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/bin:/usr/games:/snap/bin:/home/will/.dotnet/tools
GJS_DEBUG_TOPICS=35 LOG=35
SESSION_MANAGER=local/will-XPS-13-9350:@/tmp/.ICE-unix/1088,unix/will-XPS-13-9350:/tmp/.ICE-unix/1088
LESSOPEN=- /usr/bin/lesspipe %
GTK_IM_MODULE=ibus
OLDPWD=/home/will/compsc1/424
_=~/bin/printenv
[09/17/18 WD347166] will@will-XPS-13-9350:~/compsc1/424/lab01-wdahl$
```

In this screenshot I ran env in my bash prompt which printed put all my environmental variables



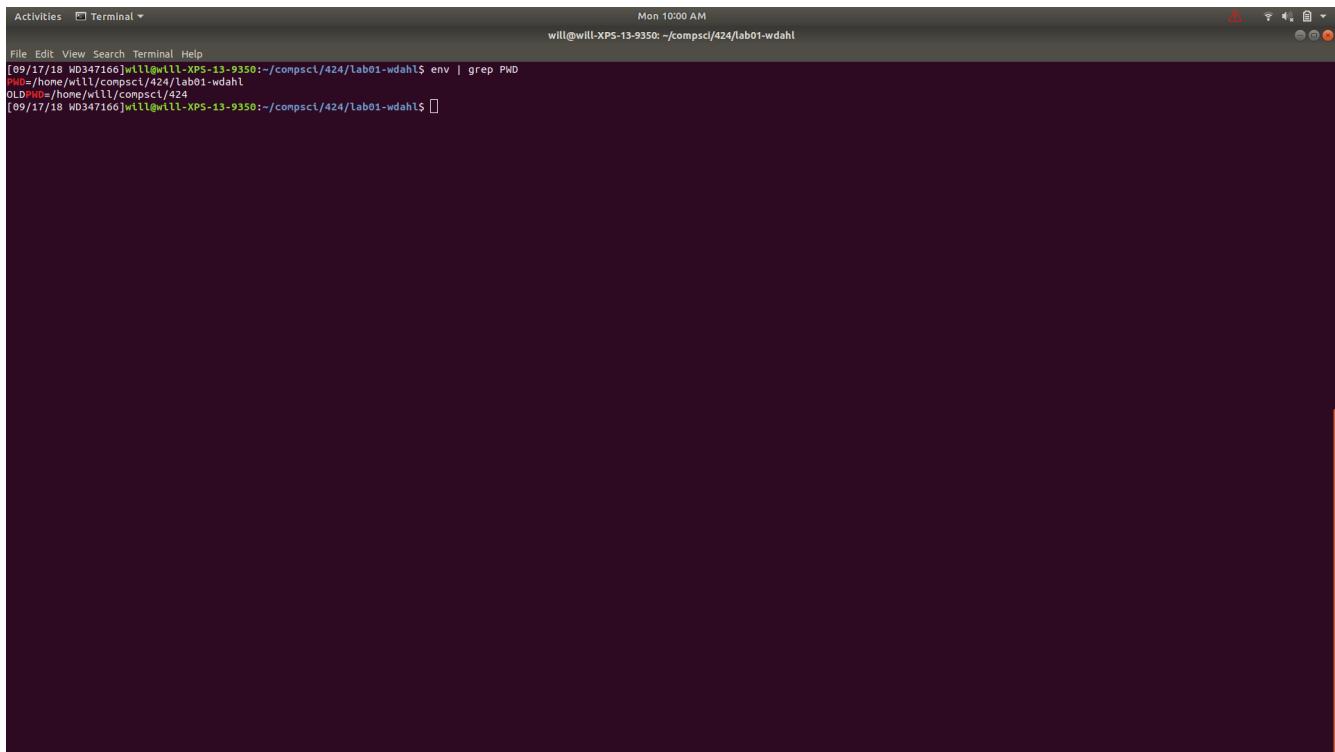
```
Activities Terminal Mon 10:00 AM will@will-XPS-13-9350: ~/compsc1/424/lab01-wdahl
File Edit View Search Terminal Help
*.opus=0;36.*.spx=0;36.*.xspf=0;36;
LESSCLOSE=/usr/bin/lesspipe %s %
XDG_MENU_PREFIX=gnome-
LANG=en_US.UTF-8
DISPLAY=:0
DISPLAY=:0
GNOME_SHELL_SESSION_MODE=ubuntu
COLORTERM=truecolor
USERNAME=will
XDG_VTNR=1
SSH_AUTH_SOCK=/run/user/1000/keyring/ssh
MANDATORY_PATH=/usr/share/gconf/ubuntu.mandatory.path
XDG_SESSION_ID=1
USER=will
DESKTOP_SESSION=ubuntu
QT_IM_MODULE=xim
TEXTDOMAIN=im-config
SSH_AGENT_PID=1182
QT_ACCESSIBILITY=1
XDG_SESSION_TYPE=x11
XDG_DATA_DIRS=/share/ubuntu:/usr/local/share:/usr/share:/var/lib/snapd/desktop
XDG_SESSION_DESKTOP=ubuntu
GJS_DEBUG_OUTPUT=stderr
GTK_MODULES=gail:atk-bridge
WINDOWPATH=1
TERM=xterm-256color
SHELL=/bin/bash
VTE_VERSION=5202
QT_IM_MODULE=xim
XMODIFIERS=@imibus
IM_CONFIG_PHASE=2
XDG_CURRENT_DESKTOP=GNOME
GPG_AGENT_INFO=/run/user/1000/gnupg/S.gpg-agent:0:1
GNOME_TERMINAL_SERVICE=:1.140
XDG_SEAT=seat0
SHLVL=1
GDMSESSION=ubuntu
GNOME_DESKTOP_SESSION_ID=this-is-deprecated
LOGNAME=will
DBUS_SESSION_BUS_ADDRESS=unix:path=/run/user/1000/bus
XDG_RUNTIME_DIR=/run/user/1000
XAUTHORITY=/run/user/1000/gdm/Xauthority
XDG_CONFIG_DIRS=/etc/xdg/xdg-ubuntu:/etc/xdg
PATH=/home/will/.local/bin:/bin:/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/bin:/usr/games:/snap/bin:/home/will/.dotnet/tools
GJS_DEBUG_TOPICS=35 ERROR=35 LOG=35
SESSION_MANAGER=local/will-XPS-13-9350:@/tmp/.ICE-unix/1088,unix/will-XPS-13-9350:/tmp/.ICE-unix/1088
LESSOPEN=- /bin/lesspipe %
GTK_IM_MODULE=ibus
OLDPWD=/home/will/compsc1/424
_=~/bin/env
[09/17/18 WD347166] will@will-XPS-13-9350:~/compsc1/424/lab01-wdahl$
```

In this screenshot I ran printenv PWD in my prompt which returned my current working directory



```
Activities Terminal Mon 10:00 AM
will@will-XPS-13-9350: ~/compsc1/424/lab01-wdahl
File Edit View Search Terminal Help
[09/17/18 WD347166]will@will-XPS-13-9350:~/compsc1/424/lab01-wdahl$ printenv PWD
/home/will/compsc1/424/lab01-wdahl
[09/17/18 WD347166]will@will-XPS-13-9350:~/compsc1/424/lab01-wdahl$ 
```

In this screenshot I ran env | grep PWD which returned anything in my environment variables containing the string PWD, so it returned my current working directory as well as my old working directory which is the parent of my current working directory



```
Activities Terminal Mon 10:00 AM
will@will-XPS-13-9350: ~/compsc1/424/lab01-wdahl
File Edit View Search Terminal Help
[09/17/18 WD347166]will@will-XPS-13-9350:~/compsc1/424/lab01-wdahl$ env | grep PWD
PWD=/home/will/compsc1/424/lab01-wdahl
OLDPWD=/home/will/compsc1/424/lab01-wdahl
[09/17/18 WD347166]will@will-XPS-13-9350:~/compsc1/424/lab01-wdahl$ 
```

In this screenshot I set an Environmental Variable INFO to be a path, when I run cd \$INFO it takes me to that path. When I unset INFO and I ran cd INFO again it took me back to my home directory because INFO is empty.

```
Activities Terminal Mon 12:48 PM will@will-XPS-13-9350: ~
File Edit View Search Terminal Help
[09/17/18 WD347166]will@will-XPS-13-9350:~/compsc1/424/lab01-wdahls export INFO=~/compsc1/424
[09/17/18 WD347166]will@will-XPS-13-9350:~/compsc1/424/lab01-wdahls cd $INFO
[09/17/18 WD347166]will@will-XPS-13-9350:~/compsc1/424$ cd lab01*
[09/17/18 WD347166]will@will-XPS-13-9350:~/compsc1/424/lab01-wdahls unset INFO
[09/17/18 WD347166]will@will-XPS-13-9350:~/compsc1/424/lab01-wdahls cd $INFO
[09/17/18 WD347166]will@will-XPS-13-9350:~/compsc1/424$ printenv INFO
[09/17/18 WD347166]will@will-XPS-13-9350:~$ 
```

Task 2: This screen shot shows the output when the environmental variables of the child process are printed.

```
Activities Terminal Mon 1:26 PM will@will-XPS-13-9350: ~/compsc1/424/lab01-wdahl
File Edit View Search Terminal Help
[89/17/18 MD347160]will@will-XPS-13-9350:~/compsc1/424/lab01-wdahl$ cat child
CLUTTER_IM_MODULE=xim
SNAP_INSTANCE_USER_COMMON=/home/will/snap/vscode/common
SNAP_INSTANCE_USER_DATA=/home/will/snap/vscode/common
SNAP_USER_COMMON=/home/will/snap/vscode/common
SNAP_USER_DATA=/home/will/snap/vscode/54
LANDING_EPOCH=1528800000
DISPLAY=:0
OLDPWD=/home/will
GNOME_SHELL_SESSION_MODE=ubuntu
SNAP_REVISION=54
SNAP_ARCH=amd64
COLORTERM=truecolor
USERNAME=will
XDG_VTNR=1
SSH_AUTH_SOCK=/run/user/1000/keyring/ssh
MANDATORY_PATH=/usr/share/gconf/ubuntu.mandatory.path
XDG_SESSION_ID=1
SNAP_INSTANCE_USER_COMMON=/home/will/snap/vscode/common
USER=will
DESKTOP_SESSION=ubuntu
QT_IM=MODULE=xim
TEXTDOMAINDIR=/usr/share/locale/
GNOME_TERMINALSCREEN=/org/gnome/Terminal/screen/08df2867_7295_4fid_94b5_8196a18fa3f2
DESKTOP_HOME=/usr/share/gconf/ubuntu.default.path
PWD=$(cd /home/will/compsc1/424/lab01-wdahl;pwd)
HOME=/home/will
TEXTDOMAIN=im-config
SNAP_INSTANCE=/snap/vscode/54
SNAPS=/snap/vscode/54
SNAP_INSTANCE_USER_DATA=/home/will/snap/vscode/54
SSH_AGENT_PID=1182
TERM_PROGRAM=vscodium
SNAP_INSTANCE_USER_DATA=/var/snap/vscode/54
TERM_PROGRAM=vscode-terminal,version=0.27.2
QT_ACCESSIBILITY=1
SNAP_COMMON=/var/snap/vscode/common
XDG_SESSION_TYPE=x11
SNAP_NAME=vscode
XDG_DATA_DIRS=/usr/share/ubuntu:/usr/local/share:/usr/share:/var/lib/snapd/desktop
SNAP_INSTANCE_NAME=vscode
SNAP_INSTANCE=/snap/vscode/54
XDG_SESSION_DESKTOP=ubuntu
GJS_DEBUG_OUTPUT=stderr
SNAP_INSTANCE_COMMON=/var/snap/vscode/common
GTK_MODULES=gail+atk+bridge
SNAP_COOKIE=f1h75tQH0M6B7gZKJBwM2Lmt13WypV4i1h0rG9c17vnq
```

This screenshot shows the output when the environmental variables of the parent process are printed

This screenshot shows the output when diff is run against the outputs from the child process and the parent process.

Activities Terminal ▾ Mon 1:22 PM
will@will-XPS-13-9350: ~/compsc1/424/lab01-wdahl

```
[09/17/18 WD347166]will@will-XPS-13-9350:~/compsc1/424/lab01-wdahl$ diff parent child
[09/17/18 WD347166]will@will-XPS-13-9350:~/compsc1/424/lab01-wdahl$ 
```

After running the diff command against the output from the child process there is no difference between the environmental variables in the parent and child processes. This is because no new Environmental variable were created within the program.

Task 3: In this screen shot we see that nothing is printed out because the source that the command execve is getting information from is NULL

```

Activities  Visual Studio Code *
lab01/lab01-env-vars-setuid.pdf at master · ualbany-cs1524-f18/lab01 - Mozilla Firefox
lab01/lab01-env-vars-setuid.pdf at master · ualbany-cs1524-f18/lab01 | University at Albany - Mozilla Firefox
MyUAlbany | University... | Yuzu | book.pdf-book.du... | Teaching Numerical A... | ualbany-cs1524-f18
Step 3. Compare the difference of these two files using the diff command. Please draw your conclusion.

2.3 Task 3: Environment Variables and execve()
In this task, we study how environment variables are affected when a new program is executed via execve(). The function execve() calls a system call to load a new command and execute it; this function never returns. No new process is created; instead, the calling process's text, data, bss, and stack are overwritten by that of the program loaded. Essentially, execve() runs the new program inside the calling process. We are interested in what happens to the environment variables; are they automatically inherited by the new program?

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



```

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

extern char **environ;

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

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

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

 return 0;
}

```

Step 2. Change the invocation of execve() in Line ① to the following; describe your observation.

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

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

2.4 Task 4: Environment Variables and system()
In this task, we study how environment variables are affected when a new program is executed via the system() function. This function is used to execute a command, but unlike execve(), which directly executes a command, system() actually executes "/bin/sh -c command", i.e., it executes /bin/sh, and asks the shell to execute the command.

```

In this screenshot we see that the processes environmental variables are printed out. This is because the source that execve was given was changed from NULL to environ which contained all the information regarding the environmental variables in the process.

```

Activities  Visual Studio Code *
lab01/lab01-env-vars-setuid.pdf at master · ualbany-cs1524-f18/lab01 - Mozilla Firefox
lab01/lab01-env-vars-setuid.pdf at master · ualbany-cs1524-f18/lab01 | University at Albany - Mozilla Firefox
MyUAlbany | University... | Yuzu | book.pdf-book.du... | Teaching Numerical A... | ualbany-cs1524-f18
Step 3. Compare the difference of these two files using the diff command. Please draw your conclusion.

2.3 Task 3: Environment Variables and execve()
In this task, we study how environment variables are affected when a new program is executed via execve(). The function execve() calls a system call to load a new command and execute it; this function never returns. No new process is created; instead, the calling process's text, data, bss, and stack are overwritten by that of the program loaded. Essentially, execve() runs the new program inside the calling process. We are interested in what happens to the environment variables; are they automatically inherited by the new program?

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.

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

Step 2. Change the invocation of execve() in Line ① to the following; describe your observation.

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

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

2.4 Task 4: Environment Variables and system()
In this task, we study how environment variables are affected when a new program is executed via the system() function. This function is used to execute a command, but unlike execve(), which directly executes a command, system() actually executes "/bin/sh -c command", i.e., it executes /bin/sh, and asks the shell to execute the command.

```

Without `environ` being passed to `eexecve` the process doesn't know of any environmental variables. That is why in order to see the environmental variables printed out the external environment needs to be given to the command so that It can print out the environmental variables.

Task 4: In this screenshot we see that the environment variables were printed out by running task4.c and no environ was needed. This is because the System call to “/bin/sh/env” uses execl() which calls execve() and passes to it the environment variables array.

Task 5: In this screen shot I changed the executable file for the code in task 5 to be owned by root and have a mode of 4755 to make it a Set-UID program.

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

```
// Assume the program's name is foo
$ sudo chown root foo
$ sudo chmod 4755 foo
```

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

- `PATH`

SEED Labs – Environment Variable and Set-UID Program Lab 5

- `LD_LIBRARY_PATH`
- `ANY_NAME` (this is an environment variable defined by you, so pick whatever name you want).

These environment variables are set in the user's shell process. Now, run the Set-UID program from Step 2 in your shell. After you type the name of the program in your shell, the shell forks a child process, and uses the child process to run the program. Please check whether all the environment variables you set in the shell process (parent) get into the Set-UID child process. Describe your observation. If there are surprises to you, describe them.

```
[09/17/18 WD347166@willwill-XPS-13-9350:~/compsci/424/lab01-wdahl$ ls
child parent task2.c task3.c task4.c
[09/17/18 WD347166@willwill-XPS-13-9350:~/compsci/424/lab01-wdahl$ code task5.c
[09/17/18 WD347166@willwill-XPS-13-9350:~/compsci/424/lab01-wdahl$ ls
child parent task2.c task3.c task4.c tasks5.c
[09/17/18 WD347166@willwill-XPS-13-9350:~/compsci/424/lab01-wdahl$ gcc -o task5 tasks5.c
[09/17/18 WD347166@willwill-XPS-13-9350:~/compsci/424/lab01-wdahl$ ls
child parent task2.c task3.c task4.c task5 task5.c
[09/17/18 WD347166@willwill-XPS-13-9350:~/compsci/424/lab01-wdahl$ sudo chown root task5
[09/17/18 WD347166@willwill-XPS-13-9350:~/compsci/424/lab01-wdahl$ sudo chmod 4755 task5
[09/17/18 WD347166@willwill-XPS-13-9350:~/compsci/424/lab01-wdahl$ ]
```

This screen shot shows the environment variables being set and the task5 code being run.

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

```
// Assume the program's name is foo
$ sudo chown root foo
$ sudo chmod 4755 foo
```

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

- PATH

SEED Labs – Environment Variable and Set-UID Program Lab

These environment variables are set in the user's shell process. Now, run the Set-UID program from Step 2 in your shell. After you type the name of the program in your shell, the shell forks a child process, and uses the child process to run the program. Please check whether all the environment variables you set in the shell process (parent) get into the Set-UID child process. Describe your observation. If there are surprises to you, describe them.

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

Because of the shell program invoked, calling `system()` within a Set-UID program is quite dangerous. This is because the actual behavior of the shell program can be affected by environment variables, such as `PATH`; these environment variables are provided by the user, who may be malicious. By changing these variables, malicious users can control the behavior of the Set-UID program. In `Bash`, you can change the `PATH` environment variable in the following way (this example adds the directory `/home/seed` to the

In the environment variables outputted by the code, the only exported variable that did not appear in the child processes environment variables was LD_LIBRARY_PATH. Both Path and my made up variable appeared in the child processes environment variables. I was surprised to see that LD_LIBRARY_PATH did not appear but ANY_NAME did.

Task 6: In this screenshot I compile the code for task6.c and change the executable task6 to be owned by root and to be a SET_UID program. Then I run the code and it performs as expected. However after I compile the code in ls.c and name the executable ls, task6 will instead execute my code which is a system call using the command “whoami” which this displays that the program thinks I’m root thus showing that my code is running with root privileges.

These environment variables are set in the user's shell process. Now, run the Set-UID program from Step 2 in your shell. After you type the name of the program in your shell, the shell forks a child process, and uses the child process to run the program. Please check whether all the environment variables you set in the shell process (parent) get into the Set-UID child process. Describe your observation. If there are surprises to you, describe them.

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

Because of the shell program invoked, calling `system()` within a Set-UID program is quite dangerous. This is because the actual behavior of the shell program can be affected by environment variables, such as `PATH`; these environment variables are provided by the user, who may be malicious. By changing these variables, malicious users can control the behavior of the Set-UID program. In Bash, you can change the `PATH` environment variable in the following way (this example adds the directory `/home/seed` to the beginning of the `PATH` environment variable):

```
$ export PATH=/home/seed:$PATH
```

The Set-UID program below is supposed to execute the `/bin/ls` command; however, the programmer only uses the relative path for the `ls` command, rather than the absolute path:

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

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

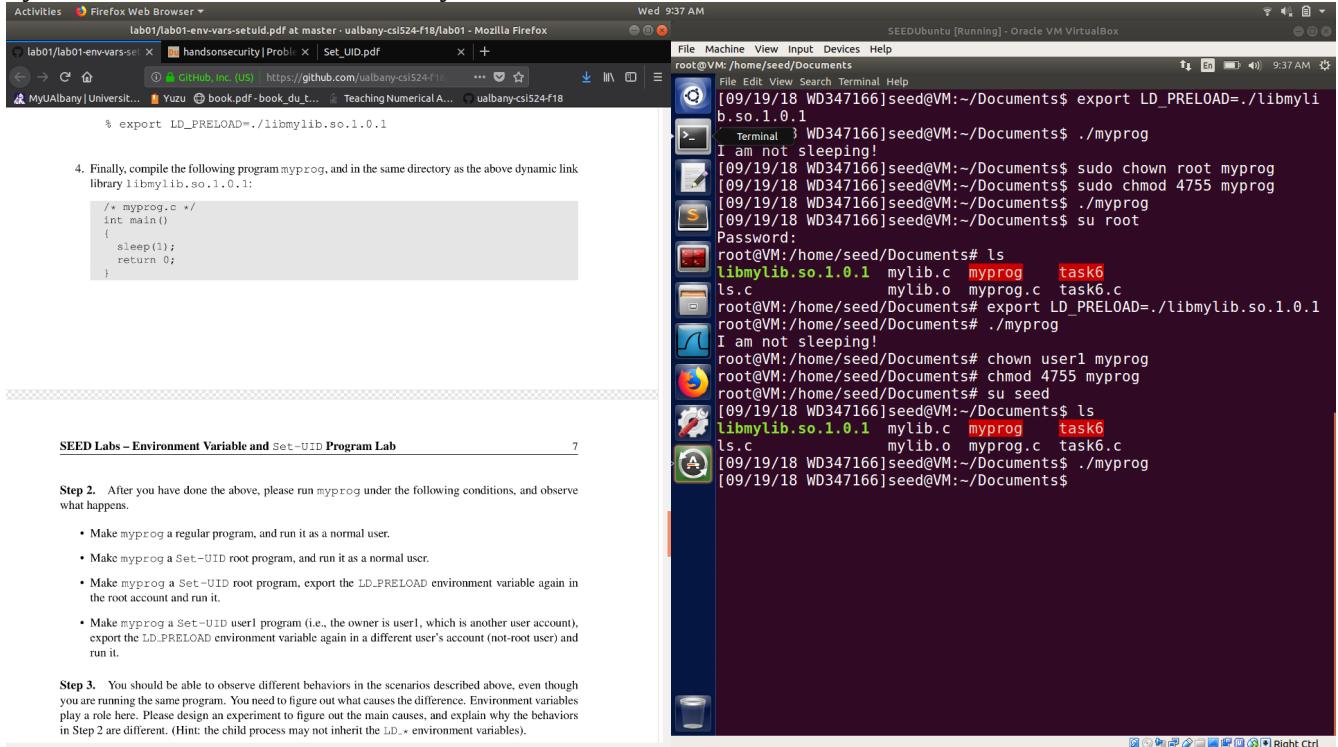
Note (Ubuntu 16.04 VM only): The `system(cmd)` function executes the `/bin/sh` program first, and then asks this shell program to run the `cmd` command. In both Ubuntu 12.04 and Ubuntu 16.04 VMs, `/bin/sh` is actually a symbolic link pointing to the `/bin/dash` shell. However, the dash program in these two VMs have an important difference. The dash shell in Ubuntu 16.04 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. The dash program in Ubuntu 12.04 does not have this behavior.

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 16.04 VM. We use the following commands to link `/bin/sh` to `zsh` (there is no need to do this in Ubuntu 12.04):

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

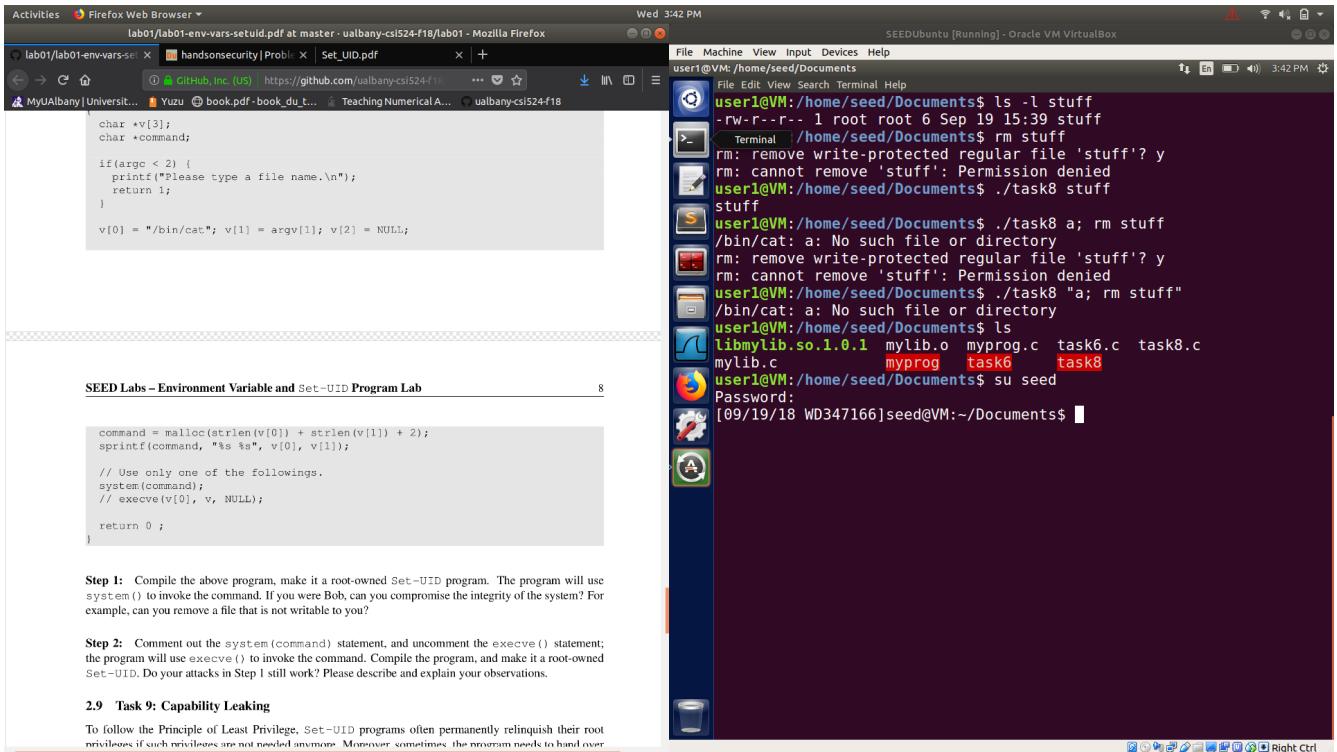
Altering the PATH variable by putting a “.” at the beginning (referencing the current directory) caused the ls file in the current directory to execute instead of the shell command ls. My program also shows that it is being run with root privilege as the shell thinks that It is a root privilege program. This shows that I could write my own malicious code and have it execute with root privileges.

Task 7: In this screenshot I ran myprog as seed owned by seed, as seed owned by root, as root owned by root, and then as seed owned by user1.

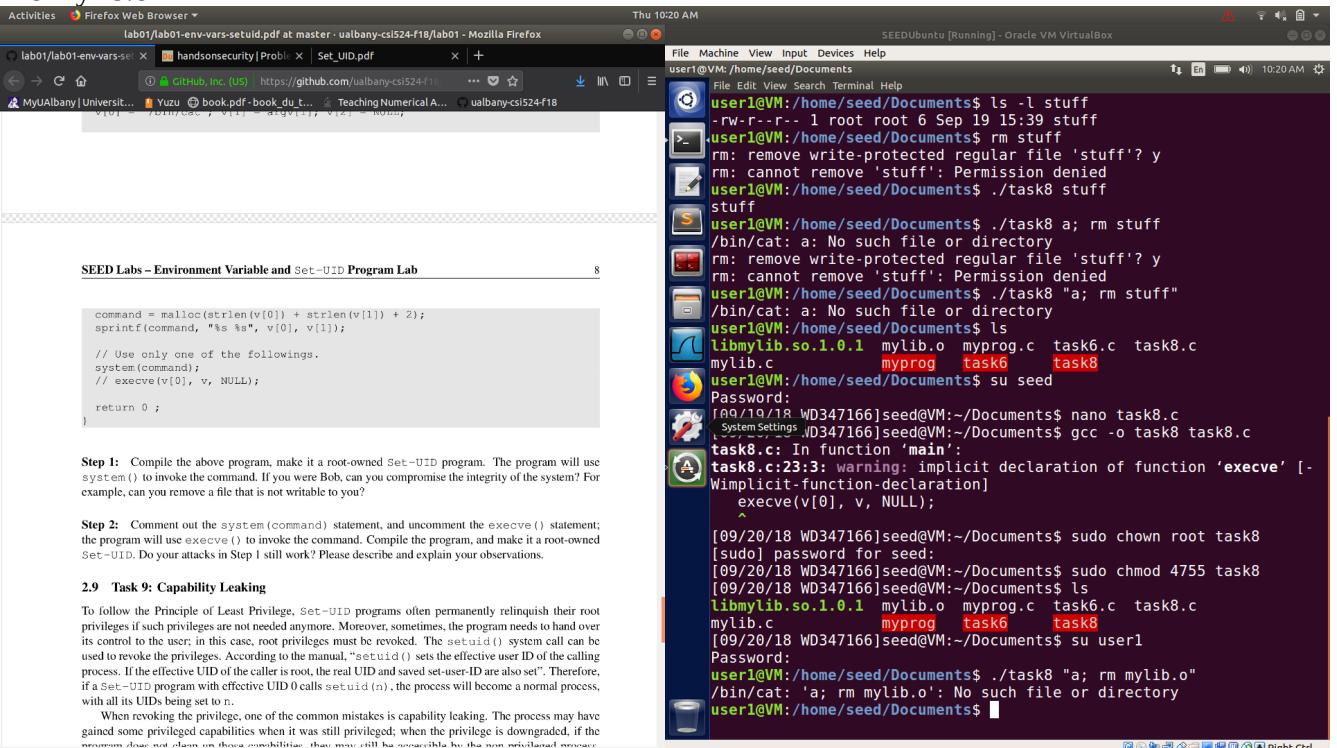


The output from `myprog` is different because in the cases where “I am not sleeping” printed out, the EUID and the RUID matched so It used the library specified in `LD_PRELAOD`. However in the other times it didnt work was because the EUID and the RUID did not match so as a countermeasure the system ignores the `LD_PRELOAD` variable.

Task 8: In this screen shot I use the user1 account I created for task 7 to try and remove a file called stuff which is owned by root and is write protected.



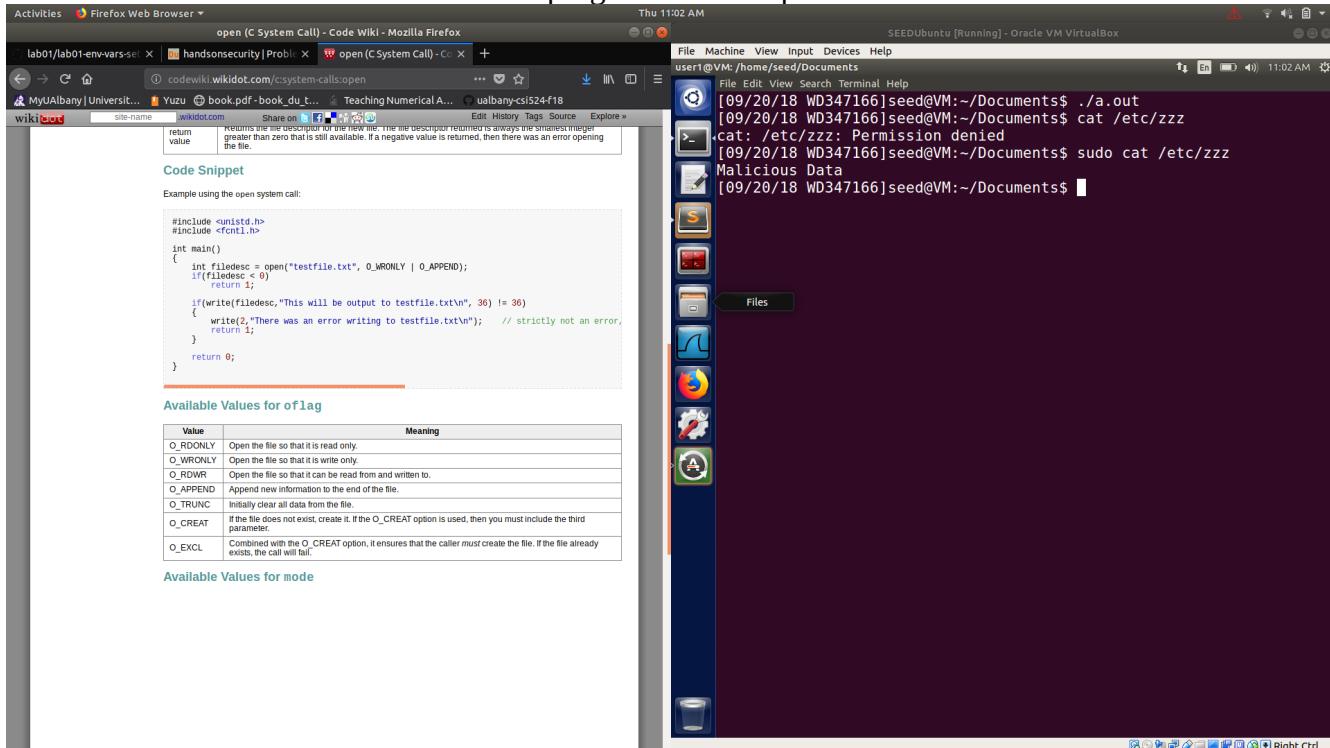
In this screen shot I run task8 code again but this time using the execve function and try to remove the file mylib.o



When using the system call I was able to remove a write protected file because the system() function makes a call to bin/sh which then executes the command passed to it, in this case "a; rm stuff". Thus the command a; rm stuff was executed with root privileges. A is not a file that command doesn't work

but the ; allows for us to put in more than one command thus rm stuff was executed with root privileges and I was able to remove the write protected file. However, when using execve() the entire string “a; rm mylib.o” was passed to bin/cat and sense there is no directory called that the command did not work.

Task 9: In this screen shot I run the task 9 program and then print the continents of ect/zzz out.



The continents of ect/zzz were changed because we are able to open the file thanks to the root privileges and even though the uid was changed later in the program we were still able to write to the file because it was already opened.