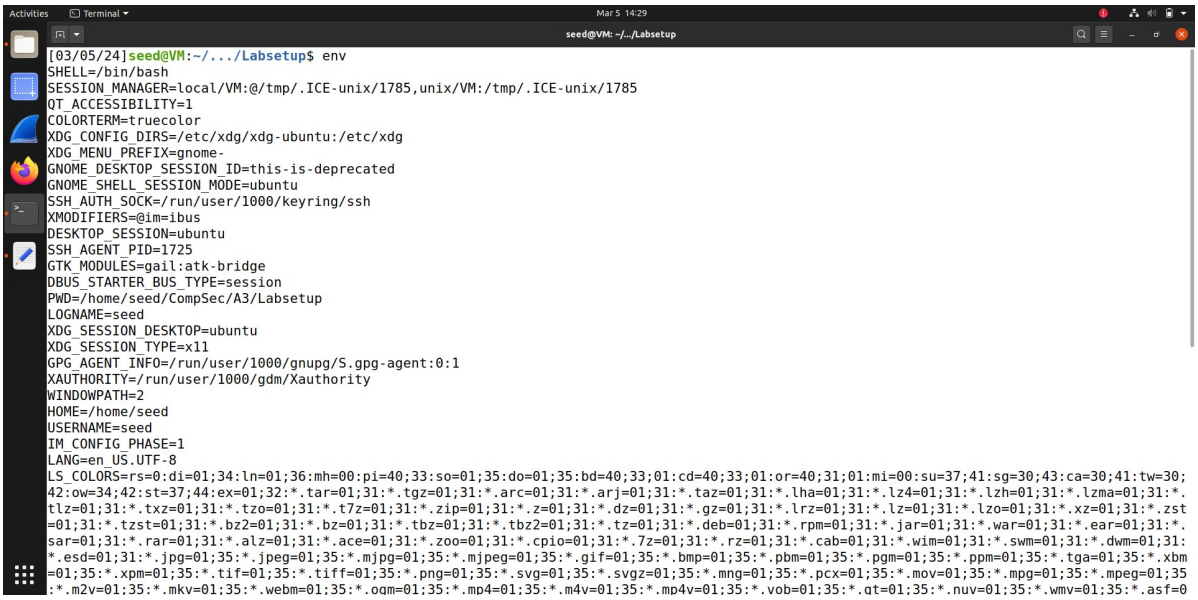
# Comp 4580 A3

Katrina Dotzlaw [redacted]

## Task 1: Manipulating Environment Variables

Using the shell command `env`, I printed out all the environment variables.




Then I set the `PATH` environment variable to `usr\bin\home` using `export`. Using `env | grep PATH`, I can see that the `PATH` variable has changed.


Next, I reset `PATH` back to its original value using `export`.

```
]seed@VM:~/.../Labsetup$ export PATH=/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin:/usr/games:/usr/local/games:/snap/bin:.
```

Since I didnt want to affect any existing environment variables, I created a new environment variable `TEST` and used `export TEST=1` to set it. Then I unset it using `unset TEST` and checked that it worked using `env | grep TEST`. As you can see, the environment variable `TEST` no longer exists.

```
[03/05/24]seed@VM:~/.../Labsetup$ TEST=1
[03/05/24]seed@VM:~/.../Labsetup$ export TEST
[03/05/24]seed@VM:~/.../Labsetup$ env | grep TEST
TEST=1
[03/05/24]seed@VM:~/.../Labsetup$ unset TEST
[03/05/24]seed@VM:~/.../Labsetup$ env | grep TEST
[03/05/24]seed@VM:~/.../Labsetup$
```

## Task 2: Passing Environment Variables Between Processes

First, I compiled `myprintenv.c` with `gcc myprintenv.c` which results in `a.out`.

```
[03/05/24]seed@VM:~/.../Labsetup$ gcc myprintenv.c
[03/05/24]seed@VM:~/.../Labsetup$ a.out > file1
[03/05/24]seed@VM:~/.../Labsetup$
```

Then, I ran `a.out` and piped the results into `file1`. I havent changed `myprintenv.c`, so `file1` contains all the environment variables from the **child** process.


Next I commented out `printenv()` in case 0 of `myprintenv.c` and uncommented `printenv()` in case 1. Now running this program will print out all the environment variables of the **parent** process.

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

I compiled the program again with `gcc myprintenv.c` and ran it, piping the results into `file2`.

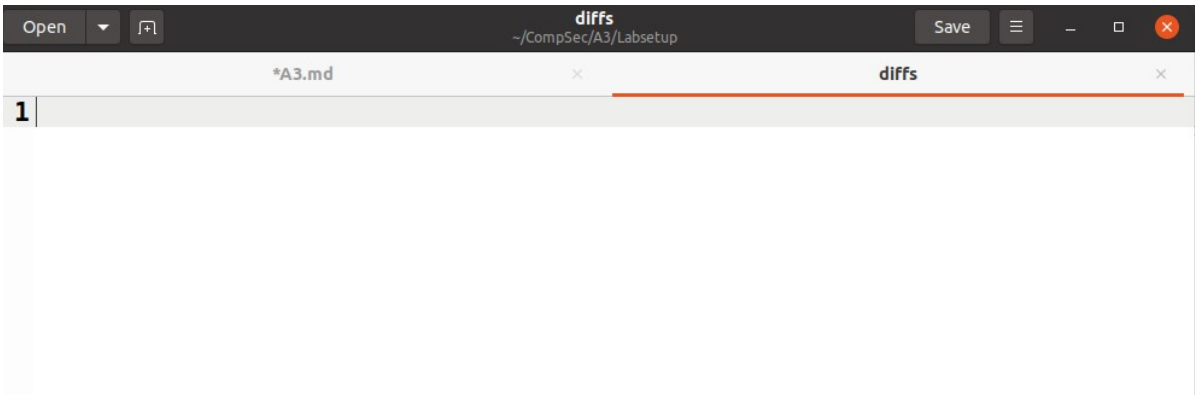
```
[03/05/24]seed@VM:~/.../Labsetup$ gcc myprintenv.c
[03/05/24]seed@VM:~/.../Labsetup$ a.out > file2
```

Looking at the image below (file2) and file1, they appear to be the same. I cant see any observable differences. The order of the environment variables listed appear the same, and the values of the environment variables also appear to be the same.


To make sure my hypothesis about the 2 files being the same holds, I used the shell command `diff file1 file2 > diffs` to find any differences in the 2 files and store the output in a textfile.

```
[03/05/24]seed@VM:~/.../Labsetup$ diff file1 file2 > diffs
```

`diffs` is blank indicating that there are no differences between file1 and file2. This means that the environment variables of the child process and the parent process are the same. It appears that the child process inherits its environment variables from its parent.




## Task 5: Environment Variables & SUID Programs

Using the code provided in the lab setup document, I created `foo.c`.


Then I compiled `foo.c` using `gcc foo.c`. To change `foo.c` into a SUID program, I first changed its ownership to root using `sudo chown root a.out` and then made it into a SUID program with `sudo chmod 4755 a.out`.

```
[03/05/24]seed@VM:~/.../Labsetup$ gcc foo.c
[03/05/24]seed@VM:~/.../Labsetup$ sudo chown root a.out
[03/05/24]seed@VM:~/.../Labsetup$ sudo chmod 4755 a.out
```

I made sure I wasnt in a root account by checking the `USERNAME` environment variable. Then using `export`, I set the environment variables `PATH`, `LD_LIBRARY_PATH`, and `TEST` as seen in the image below.

```
[03/05/24]seed@VM:~/.../Labsetup$ env | grep USERNAME
USERNAME=seed
[03/05/24]seed@VM:~/.../Labsetup$ export PATH=/bin:/usr/bin/home
[03/05/24]seed@VM:~/.../Labsetup$ env | grep LD_LIBRARY_PATH
[03/05/24]seed@VM:~/.../Labsetup$ export LD_LIBRARY_PATH=1
[03/05/24]seed@VM:~/.../Labsetup$ export TEST=1
[03/05/24]seed@VM:~/.../Labsetup$ env | grep LD_LIBRARY_PATH
LD_LIBRARY_PATH=1
[03/05/24]seed@VM:~/.../Labsetup$ env | grep TEST
TEST=1
```

Then I compiled `foo.c`, which created `a.out` and ran it. Looking at the image below, I can see that `PATH` is set to `/bin:/usr/bin/home` and `TEST` is set to 1, but I cant find `LD_LIBRARY_PATH`.


When I investigated further using `./a.out | grep <env var>`, I can see that the environment variable `LD_LIBRARY_PATH` that was set in the shell is NOT transfered into the SUID program `foo.c`. `PATH` and `TEST` are confirmed to be passed to the SUID process.

```
[03/05/24]seed@VM:~/.../Labsetup$ ./a.out | grep PATH
WINDOWPATH=2
PATH=/bin:/usr/bin/home
[03/05/24]seed@VM:~/.../Labsetup$ ./a.out | grep LD_LIBRARY_PATH
[03/05/24]seed@VM:~/.../Labsetup$ ./a.out | grep TEST
TEST=1
```

## Task 6: PATH Environment Variable & SUID Programs

The code in `suid.c` is exactly the same as in SEED Labs Task 6, shown below.


I compiled `suid.c` into an object file, `goodls`. Then I made `goodls`'s owner root and made it a SUID program as shown below.

```
[03/05/24]seed@VM:~/.../Labsetup$ gcc suid.c -o goodls
[03/05/24]seed@VM:~/.../Labsetup$ sudo chown root goodls
[03/05/24]seed@VM:~/.../Labsetup$ sudo chmod 4755 goodls
```

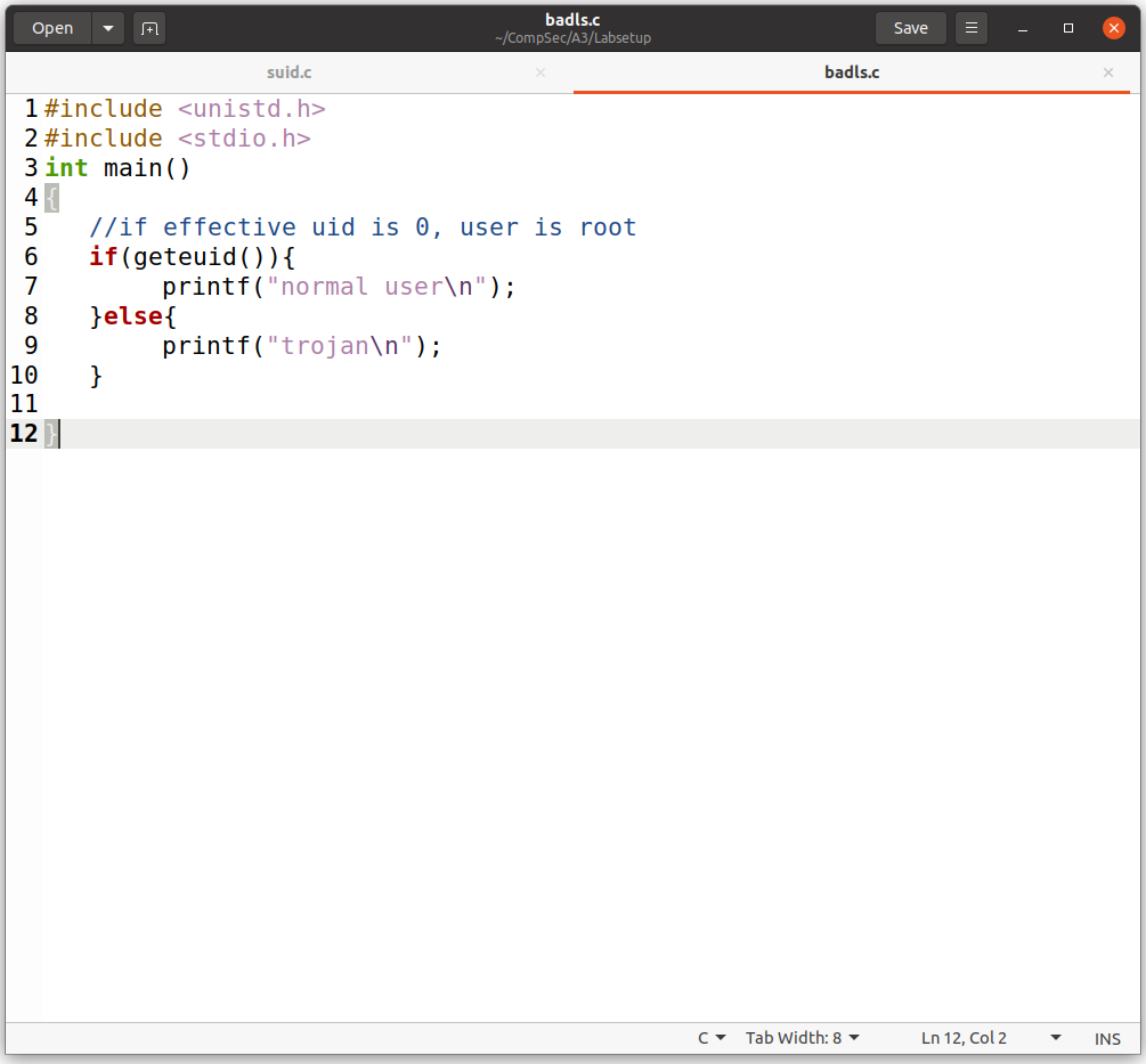
If I run `./goodls`, it searches the correct path and runs `ls`.

```
[03/05/24]seed@VM:~/.../Labsetup$ ./goodls
a.out    cap_leak.c  diffs  file2  goodls  myenv.c       suid.c
badls.c  catall.c    file1  foo.c  ls      myprintenv.c
```

To get `suid.c` to run malicious code, I created another file called `badls.c` and compiled it into `ls`.

```
[03/05/24]seed@VM:~/.../Labsetup$ gcc badls.c -o ls
[03/05/24]seed@VM:~/.../Labsetup$ ./ls
normal user
```

In `badls.c`, it returns "normal user" if the effective user id is not 0 (ie is not root user), and "trojan" if the effective user has root access. It uses `geteuid()` to get the effective user id as shown in the Linux documentation found here.




I also changed the `PATH` variable to be `PATH=/home/seed:$PATH` so that the home directory will be searched before `$PATH`. Since this file ( `ls` ) has the same name as the relative path used in `suid.c`, placing `ls` into a directory that is searched before `/ls` will allow the exploit to be done.

```
[03/05/24]seed@VM:~/.../Labsetup$ export PATH=/home/seed:$PATH
```

Then I put `ls` into the directory `/home/seed`. If I run `./ls`, "normal user" is returned which indicates that `ls` (compiled from `badls.c` ) does NOT have root access.

```
[03/05/24]seed@VM:~/.../Labsetup$ gcc badls.c -o ls
[03/05/24]seed@VM:~/.../Labsetup$ ./ls
normal user
```

But if I run the compiled version of `suid.c` with `./goodls`, "trojan" is returned, proving that I can execute 'malicious' code when I change the `PATH` environment variable. Since `goodls` is compiled from `suid.c` and I changed the owner of `suid.c` to root and made it a SUID program, `goodls` is also root owned and a SUID program.

```
[03/05/24]seed@VM:~/.../Labsetup$ ./goodls
trojan
```