

Lab 3 – Set-UID

Gunnar Yonker

Task 1: Manipulating Environmental Variables

printenv

```
Terminal
[09/25/23]seed@VM:~$ printenv
XDG_VTNR=7
XDG_SESSION_ID=c1
XDG_GREETER_DATA_DIR=/var/lib/lightdm-data/seed
CLUTTER_IM_MODULE=xim
SESSION=ubuntu
ANDROID_HOME=/home/seed/android/android-sdk-linux
GPG_AGENT_INFO=/home/seed/.gnupg/S.gpg-agent:0:1
TERM=xterm-256color
VTE_VERSION=4205
SHELL=/bin/bash
DERBY_HOME=/usr/lib/jvm/java-8-oracle/db
QT_LINUX_ACCESSIBILITY_ALWAYS_ON=1
LD_PRELOAD=/home/seed/lib/boost/libboost_program_options.so.1.64.0:/home/seed/lib/boost/libboost_filesystem.so.1.64.0:/home/seed/lib/boost/libboost_system.so.1.64.0
WINDOWID=62914570
UPSTART_SESSION=unix:abstract=/com/ubuntu/upstart-session/1000/1161
GNOME_KEYRING_CONTROL=
GTK_MODULES=gail:atk-bridge:unity-gtk-module
USER=seed
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;
```

printenv PWD

```
Terminal
[09/25/23]seed@VM:~$ printenv PWD
/home/seed
```

env | grep PATH

```
[09/25/23]seed@VM:~$ env | grep PATH
LD_LIBRARY_PATH=/home/seed/source/boost_1_64_0/stage/lib:/home/seed/source/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-oracle/jre/bin:/home/seed/android/android-sdk-linux/tools:/home/seed/android/android-sdk-linux/platform-tools:/home/seed/android/android-ndk/android-ndk-r8d:/home/seed/.local/bin
MANDATORY_PATH=/usr/share/gconf/ubuntu.mandatory.path
COMPIZ_BIN_PATH=/usr/bin/
```

export MY_VAR="test"

unset MY_VAR

```
Terminal
[09/25/23]seed@VM:~$ export MY_VAR="test"
[09/25/23]seed@VM:~$ env | grep MY_VAR
MY_VAR=test
[09/25/23]seed@VM:~$ unset MY_VAR
[09/25/23]seed@VM:~$
```

Lab 3 – Set-UID

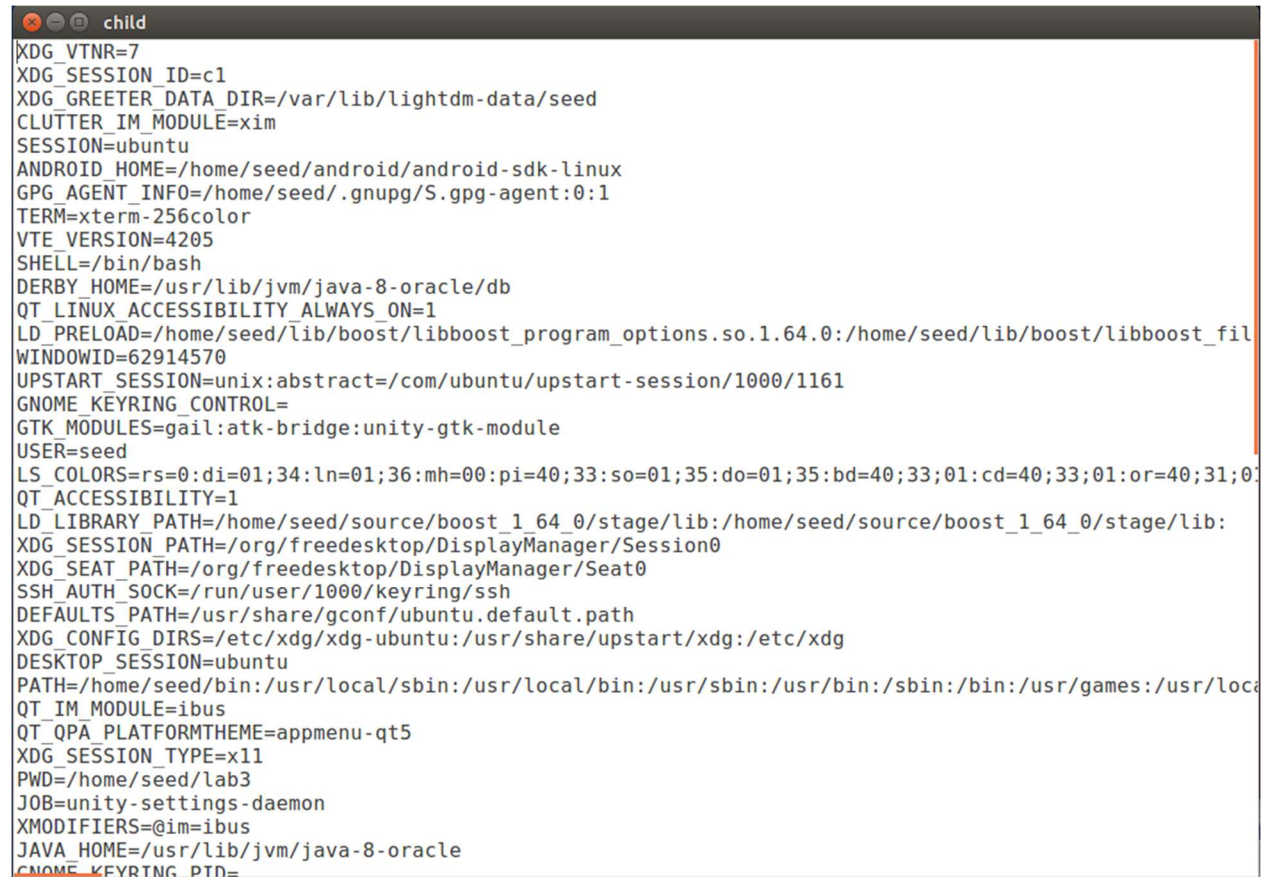
Gunnar Yonker

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

```
gcc env_variables.c -o a.out
```

```
a.out > child
```

```
child:
```



```
XDG_VTNR=7
XDG_SESSION_ID=c1
XDG_GREETER_DATA_DIR=/var/lib/lightdm-data/seed
CLUTTER_IM_MODULE=xim
SESSION=ubuntu
ANDROID_HOME=/home/seed/android/android-sdk-linux
GPG_AGENT_INFO=/home/seed/.gnupg/S.gpg-agent:0:1
TERM=xterm-256color
VTE_VERSION=4205
SHELL=/bin/bash
DERBY_HOME=/usr/lib/jvm/java-8-oracle/db
QT_LINUX_ACCESSIBILITY_ALWAYS_ON=1
LD_PRELOAD=/home/seed/lib/boost/libboost_program_options.so.1.64.0:/home/seed/lib/boost/libboost_filesystem.so.1.64.0
WINDOWID=62914570
UPSTART_SESSION=unix:abstract=/com/ubuntu/upstart-session/1000/1161
GNOME_KEYRING_CONTROL=
GTK_MODULES=gail:atk-bridge:unity-gtk-module
USER=seed
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;03
QT_ACCESSIBILITY=1
LD_LIBRARY_PATH=/home/seed/source/boost_1_64_0/stage/lib:/home/seed/source/boost_1_64_0/stage/lib:
XDG_SESSION_PATH=/org/freedesktop/DisplayManager/Session0
XDG_SEAT_PATH=/org/freedesktop/DisplayManager/Seat0
SSH_AUTH_SOCK=/run/user/1000/keyring/ssh
DEFAULTS_PATH=/usr/share/gconf/ubuntu.default.path
XDG_CONFIG_DIRS=/etc/xdg/xdg-ubuntu:/usr/share/upstart/xdg:/etc/xdg
DESKTOP_SESSION=ubuntu
PATH=/home/seed/bin:/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin:/usr/games:/usr/local/games
QT_IM_MODULE=ibus
QT_QPA_PLATFORMTHEME=appmenu-qt5
XDG_SESSION_TYPE=x11
PWD=/home/seed/lab3
JOB=unity-settings-daemon
XMODIFIERS=@im=ibus
JAVA_HOME=/usr/lib/jvm/java-8-oracle
GNOME_KEYRING_PID=
```

When the program was executed, the environment variables of the child process were printed. This output was saved to a file named child.

Gunnar Yonker

comment out `printenv()` and uncomment the other `printenv()`:

```
gcc env_variables.c -o a.out
```

```
a.out > parent
```

parent:

[illegible]

When the modified program was executed, the environment variables of the parent process were printed. This output was saved to a file named parent.

Comparing child and parent:

diff child parent

```
Terminal
[09/25/23]seed@VM:~/lab3$ diff child parent
[09/25/23]seed@VM:~/lab3$
```

No differences were observed between the two files, indicating that the environment variables for both the child and parent processes were the same.

Lab 3 – Set-UID

Gunnar Yonker

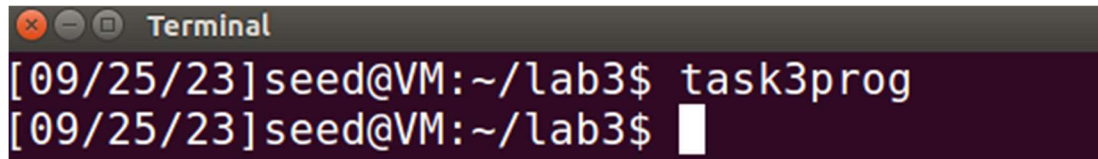
The child process inherits the environment variables of the parent process. This is evident from the absence of any differences between the two sets of environment variables printed by the child and parent processes. Thus, when a new process is created using `fork()`, the environment variables are shared and passed on from the parent to the child. This mechanism allows for consistency and continuity in the environment settings across process hierarchies.

Task 3: Environment Variables and `execve()`

Code was written into a file called `task3.c` and then compiled into `task3prog`

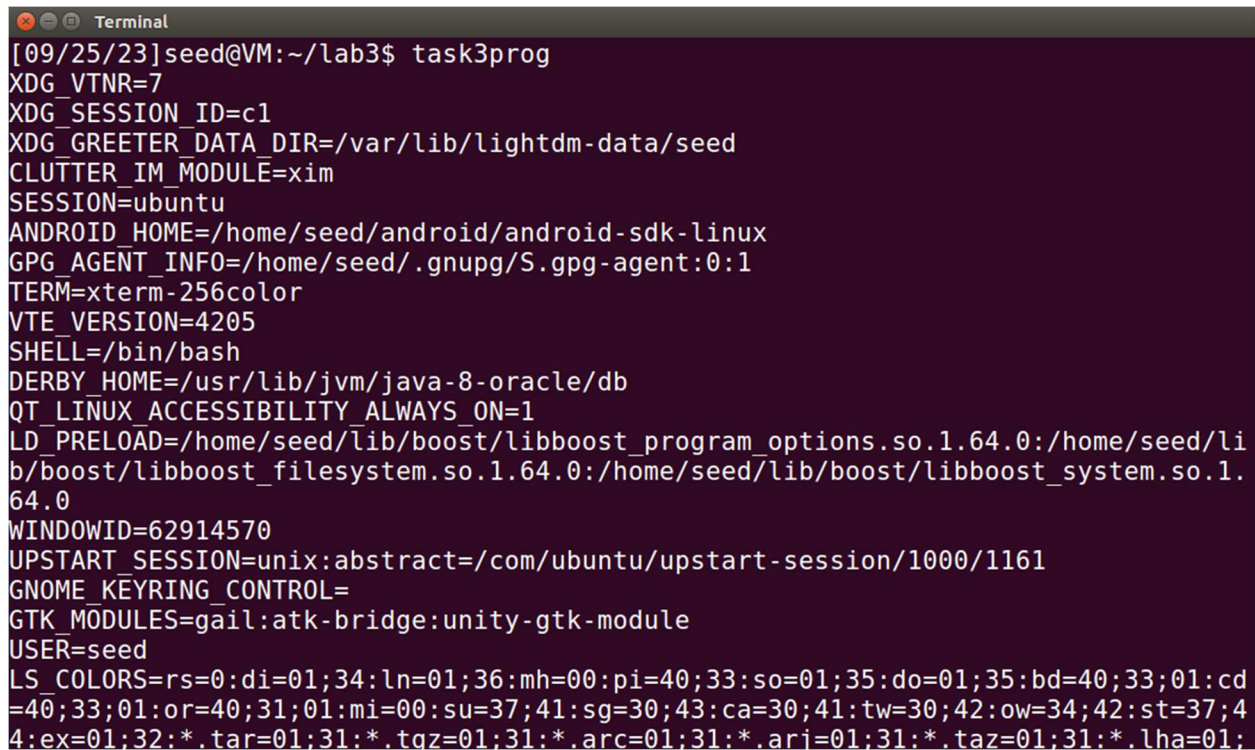
```
gcc task3.c -o task3prog
```

When the program was executed, no environment variables were printed. This is because `execve()` was called without passing any environment variables (the third argument was set to `NULL`).



```
Terminal
[09/25/23]seed@VM:~/lab3$ task3prog
[09/25/23]seed@VM:~/lab3$
```

The program was modified to pass the environment variables to the new program (`/usr/bin/env`). This time when the program was executed, the environment variables of the calling process were printed. This is because the `environ` variable, which contains all of the environment variables, was passed to `execve()`.



```
Terminal
[09/25/23]seed@VM:~/lab3$ task3prog
XDG_VTNR=7
XDG_SESSION_ID=c1
XDG_GREETER_DATA_DIR=/var/lib/lightdm-data/seed
CLUTTER_IM_MODULE=xim
SESSION=ubuntu
ANDROID_HOME=/home/seed/android/android-sdk-linux
GPG_AGENT_INFO=/home/seed/.gnupg/S.gpg-agent:0:1
TERM=xterm-256color
VTE_VERSION=4205
SHELL=/bin/bash
DERBY_HOME=/usr/lib/jvm/java-8-oracle/db
QT_LINUX_ACCESSIBILITY_ALWAYS_ON=1
LD_PRELOAD=/home/seed/lib/boost/libboost_program_options.so.1.64.0:/home/seed/lib/boost/libboost_filesystem.so.1.64.0:/home/seed/lib/boost/libboost_system.so.1.64.0
WINDOWID=62914570
UPSTART_SESSION=unix:abstract=/com/ubuntu/upstart-session/1000/1161
GNOME_KEYRING_CONTROL=
GTK_MODULES=gail:atk-bridge:unity-gtk-module
USER=seed
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;
```

Based on the observations from these two programs it is observed that:

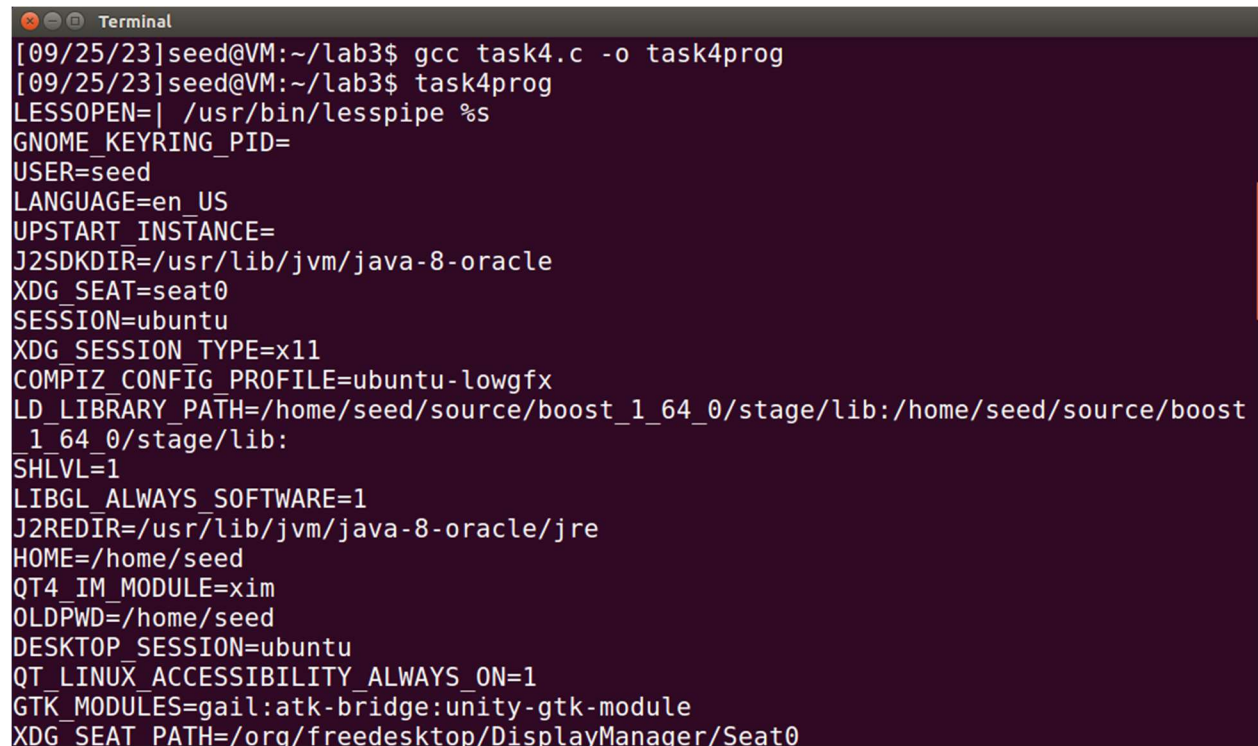
Lab 3 – Set-UID

Gunnar Yonker

When `execve()` is invoked without passing environment variables, the new program does not inherit any environment variables from the calling process. If you want the new program to inherit the environment variables, you must explicitly pass the environment variables to the `execve()` function. While the new program replaces the calling process's memory space, the inheritance of the environment variables is not automatic and depends on whether they are explicitly passed or not.

Task 4: Environment Variable and `system()`

`task4.c` compiled to `task4prog` and then executed



```
Terminal
[09/25/23]seed@VM:~/lab3$ gcc task4.c -o task4prog
[09/25/23]seed@VM:~/lab3$ task4prog
LESSOPEN=| /usr/bin/lesspipe %s
GNOME_KEYRING_PID=
USER=seed
LANGUAGE=en_US
UPSTART_INSTANCE=
J2SDKDIR=/usr/lib/jvm/java-8-oracle
XDG_SEAT=seat0
SESSION=ubuntu
XDG_SESSION_TYPE=x11
COMPIZ_CONFIG_PROFILE=ubuntu-lowgfx
LD_LIBRARY_PATH=/home/seed/source/boost_1_64_0/stage/lib:/home/seed/source/boost_1_64_0/stage/lib:
SHLVL=1
LIBGL_ALWAYS_SOFTWARE=1
J2REDIR=/usr/lib/jvm/java-8-oracle/jre
HOME=/home/seed
QT4_IM_MODULE=xim
OLDPWD=/home/seed
DESKTOP_SESSION=ubuntu
QT_LINUX_ACCESSIBILITY_ALWAYS_ON=1
GTK_MODULES=gail:atk-bridge:unity-gtk-module
XDG_SEAT_PATH=/org/freedesktop/DisplayManager/Seat0
```

When the program is executed, it prints the environment variables of the current process. This is because the `system()` function internally invokes `/bin/sh` to execute the given command and this shell inherits the environment variables of the calling process.

The `system()` function by design ensures that the called command inherits the environment variables of the calling process. This behavior is expected since `system()` executes the command via a shell, which in turn inherits the environment of its parent process. Thus, any program or command invoked through `system()` will have access to the parent's environment variables, making the propagation of environments settings consistent when using this function.

Lab 3 – Set-UID

Gunnar Yonker

Task 5: Environment Variable and Set-UID Programs

```
gcc task5.c -o task5prog
```

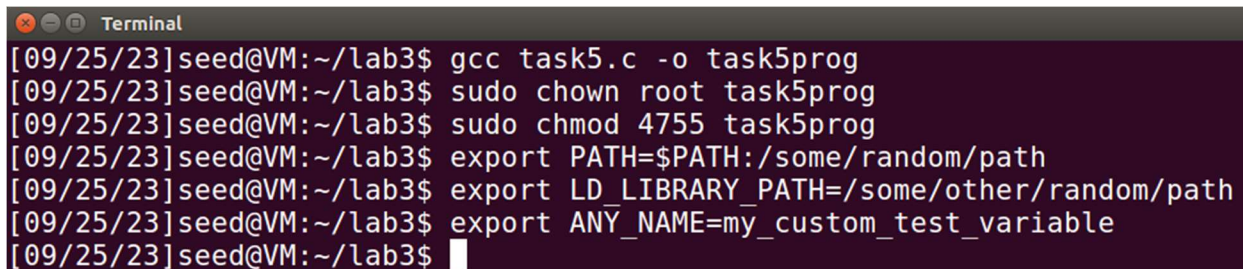
```
sudo chown root task5prog
```

```
sudo chmod 4755 task5prog
```

```
export PATH=$PATH:/some/random/path
```

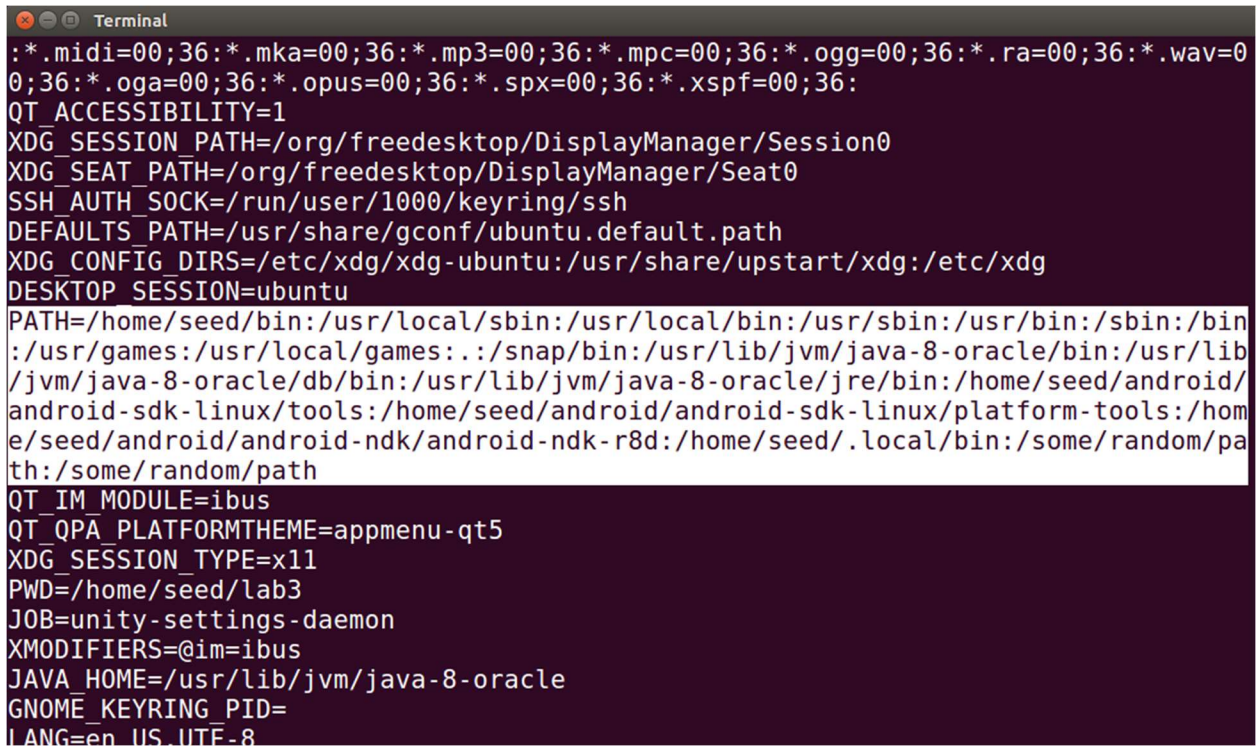
```
export LD_LIBRARY_PATH=/some/other/random/path
```

```
export ANY_NAME=my_custom_test_variable
```



```
Terminal
[09/25/23]seed@VM:~/lab3$ gcc task5.c -o task5prog
[09/25/23]seed@VM:~/lab3$ sudo chown root task5prog
[09/25/23]seed@VM:~/lab3$ sudo chmod 4755 task5prog
[09/25/23]seed@VM:~/lab3$ export PATH=$PATH:/some/random/path
[09/25/23]seed@VM:~/lab3$ export LD_LIBRARY_PATH=/some/other/random/path
[09/25/23]seed@VM:~/lab3$ export ANY_NAME=my_custom_test_variable
[09/25/23]seed@VM:~/lab3$
```

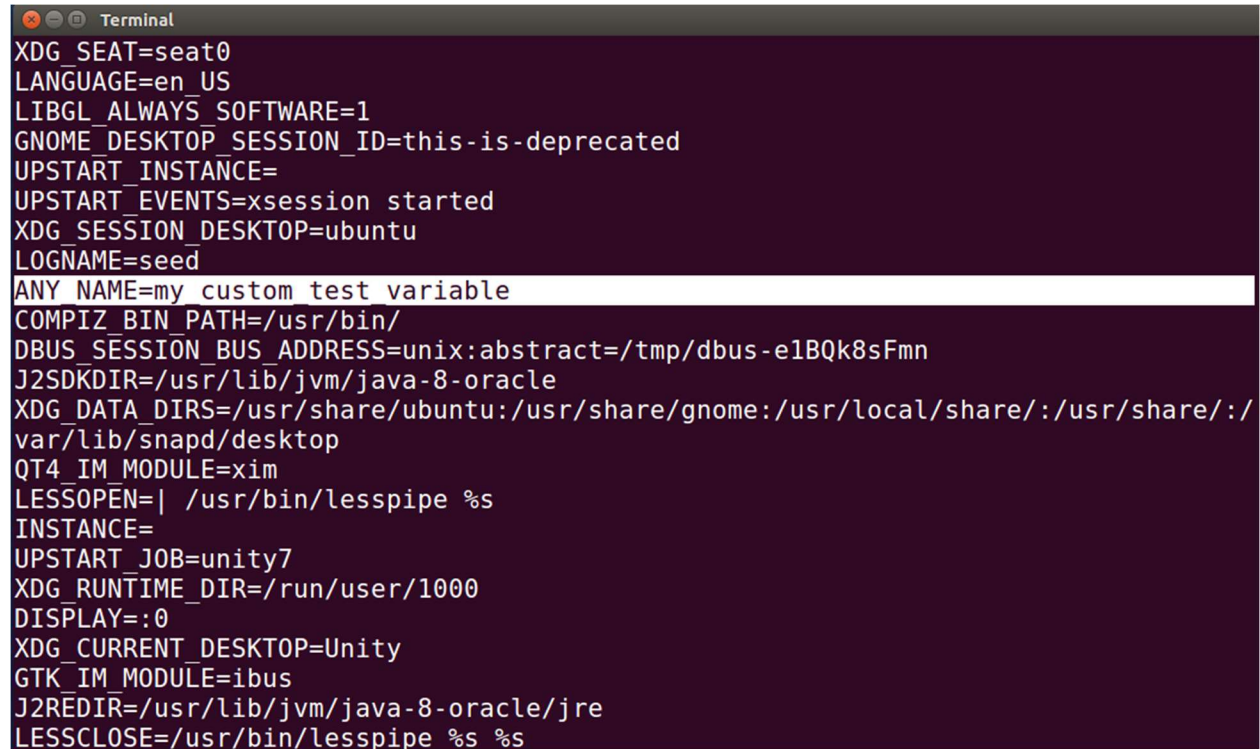
When executed:



```
Terminal
:*.midi=00;36:*.mka=00;36:*.mp3=00;36:*.mpc=00;36:*.ogg=00;36:*.ra=00;36:*.wav=0
0;36:*.oga=00;36:*.opus=00;36:*.spx=00;36:*.xspf=00;36:
QT_ACCESSIBILITY=1
XDG_SESSION_PATH=/org/freedesktop/DisplayManager/Session0
XDG_SEAT_PATH=/org/freedesktop/DisplayManager/Seat0
SSH_AUTH_SOCK=/run/user/1000/keyring/ssh
DEFAULTS_PATH=/usr/share/gconf/ubuntu.default.path
XDG_CONFIG_DIRS=/etc/xdg/xdg-ubuntu:/usr/share/upstart/xdg:/etc/xdg
DESKTOP_SESSION=ubuntu
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-oracle/jre/bin:/home/seed/android/
android-sdk-linux/tools:/home/seed/android/android-sdk-linux/platform-tools:/hom
e/seed/android/android-ndk/android-ndk-r8d:/home/seed/.local/bin:/some/random/pa
th:/some/random/path
QT_IM_MODULE=ibus
QT_QPA_PLATFORMTHEME=appmenu-qt5
XDG_SESSION_TYPE=x11
PWD=/home/seed/lab3
JOB=unity-settings-daemon
XMODIFIERS=@im=ibus
JAVA_HOME=/usr/lib/jvm/java-8-oracle
GNOME_KEYRING_PID=
LANG=en_US.UTF-8
```

Lab 3 – Set-UID

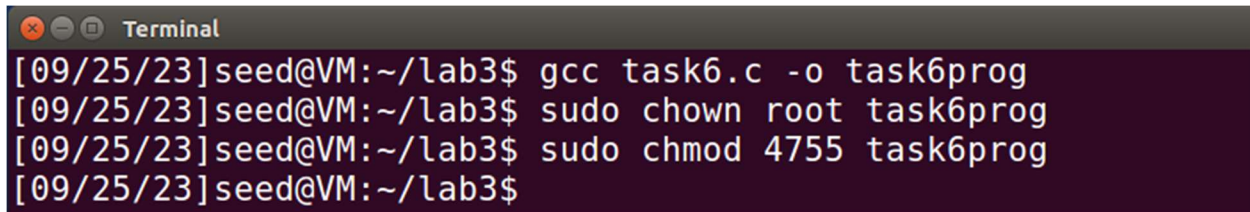
Gunnar Yonker

A terminal window titled "Terminal" with a dark background and light text. It displays a list of environment variables and their values. The variables include XDG_SEAT, LANGUAGE, LIBGL_ALWAYS_SOFTWARE, GNOME_DESKTOP_SESSION_ID, UPSTART_INSTANCE, UPSTART_EVENTS, XDG_SESSION_DESKTOP, LOGNAME, ANY_NAME (highlighted in white), COMPIZ_BIN_PATH, DBUS_SESSION_BUS_ADDRESS, J2SDKDIR, XDG_DATA_DIRS, QT4_IM_MODULE, LESSOPEN, INSTANCE, UPSTART_JOB, XDG_RUNTIME_DIR, DISPLAY, XDG_CURRENT_DESKTOP, GTK_IM_MODULE, J2REDIR, and LESSCLOSE.

```
Terminal
XDG_SEAT=seat0
LANGUAGE=en_US
LIBGL_ALWAYS_SOFTWARE=1
GNOME_DESKTOP_SESSION_ID=this-is-deprecated
UPSTART_INSTANCE=
UPSTART_EVENTS=xsession started
XDG_SESSION_DESKTOP=ubuntu
LOGNAME=seed
ANY_NAME=my custom test variable
COMPIZ_BIN_PATH=/usr/bin/
DBUS_SESSION_BUS_ADDRESS=unix:abstract=/tmp/dbus-e1BQk8sFmn
J2SDKDIR=/usr/lib/jvm/java-8-oracle
XDG_DATA_DIRS=/usr/share/ubuntu:/usr/share/gnome:/usr/local/share:/usr/share:/var/lib/snapd/desktop
QT4_IM_MODULE=xim
LESSOPEN=| /usr/bin/lesspipe %s
INSTANCE=
UPSTART_JOB=unity7
XDG_RUNTIME_DIR=/run/user/1000
DISPLAY=:0
XDG_CURRENT_DESKTOP=Unity
GTK_IM_MODULE=ibus
J2REDIR=/usr/lib/jvm/java-8-oracle/jre
LESSCLOSE=/usr/bin/lesspipe %s %s
```

When the Set-UID program task5prog is executed, it will print the environment variables of the current process. For most environment variables, including the ANY_NAME one that I created, they will be inherited by the Set-UID program as seen above in the screenshots. The LD_LIBRARY_PATH is not seen above because it is a special environment variable that can change the program behavior drastically. I think that it is not seen here because the system automatically cleared that variables for Set-UID binaries to prevent potential misuse.

Set-UID programs do inherit environment variables, but there are mechanisms in place to prevent potential misuse, especially when it comes to variables that can alter the behavior of a program in a way that could be abused for privilege escalation.

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

```
Terminal
[09/25/23]seed@VM:~/lab3$ gcc task6.c -o task6prog
[09/25/23]seed@VM:~/lab3$ sudo chown root task6prog
[09/25/23]seed@VM:~/lab3$ sudo chmod 4755 task6prog
[09/25/23]seed@VM:~/lab3$
```

Creating a malicious ls program:

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

```
void main()
```

```
{
```

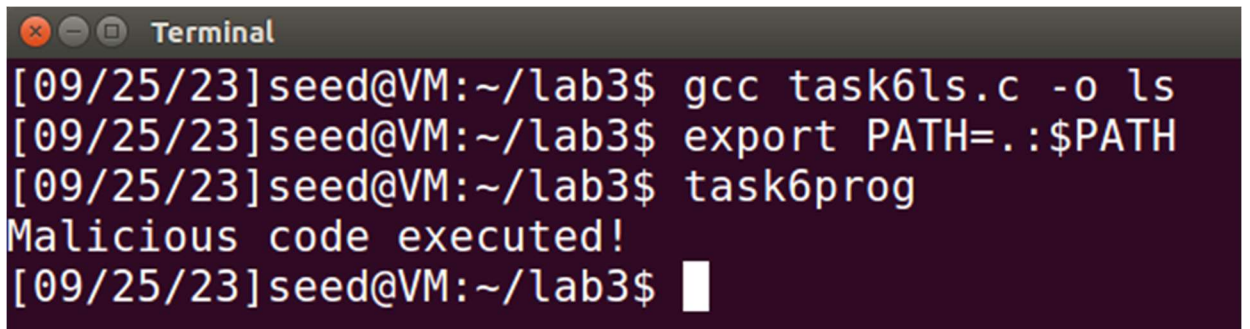
```
    printf("Malicious code executed!\n");
```

```
}
```

Manipulate path variable to run malicious ls before real ls

```
export PATH=.:$PATH
```

When executing task6prog



```
Terminal
[09/25/23]seed@VM:~/lab3$ gcc task6ls.c -o ls
[09/25/23]seed@VM:~/lab3$ export PATH=.:$PATH
[09/25/23]seed@VM:~/lab3$ task6prog
Malicious code executed!
[09/25/23]seed@VM:~/lab3$
```

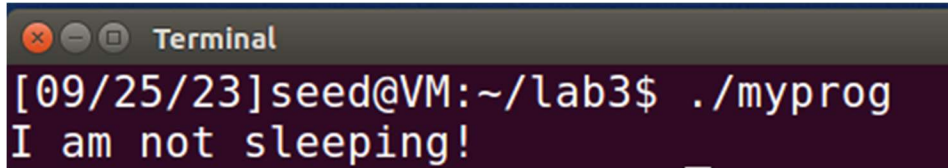
When task6prog is executed, instead of the actual ls command running, my malicious ls command is run which displays the “Malicious code executed” message. When the Set-UID program uses system(ls), it looks for the ls command in directories specified in the PATH variable. I placed my current directory containing the malicious ls at the beginning of PATH so that the system executes my ls instead of the real one. Since task6prog is Set-UID root, the malicious ls runs with root privileges.

Using relative paths in Set-UID programs, especially with the system() function, is hazardous due to the potential manipulation of the PATH variable. Malicious users can exploit this to execute arbitrary code with elevated privileges. Developers should always use absolute paths in such programs and be cautious about potential environment variable manipulations.

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

Programs and code were created and compiled as outlined in the lab.

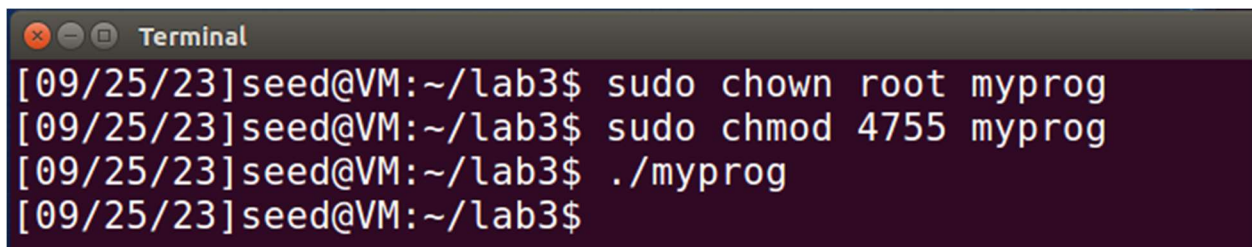
myprog as regular program and normal user:

A terminal window titled "Terminal" with a dark background. It shows the command `./myprog` being executed, and the output is `I am not sleeping!`.

```
[09/25/23]seed@VM:~/lab3$ ./myprog
I am not sleeping!
```

“I am not sleeping!” is printed, indicating that the custom library’s `sleep()` function is invoked.

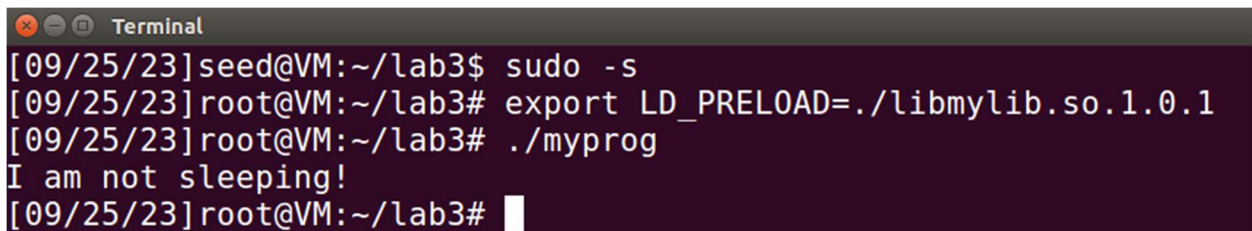
myprog as Set-UID root program and run as normal user:

A terminal window titled "Terminal" with a dark background. It shows the user `seed@VM` running `sudo chown root myprog`, `sudo chmod 4755 myprog`, and then `./myprog`. The output is blank, indicating the program is sleeping for 1 second.

```
[09/25/23]seed@VM:~/lab3$ sudo chown root myprog
[09/25/23]seed@VM:~/lab3$ sudo chmod 4755 myprog
[09/25/23]seed@VM:~/lab3$ ./myprog
[09/25/23]seed@VM:~/lab3$
```

The program appears to sleep for 1 second without printing anything, indicating that the custom `sleep()` function is not called.

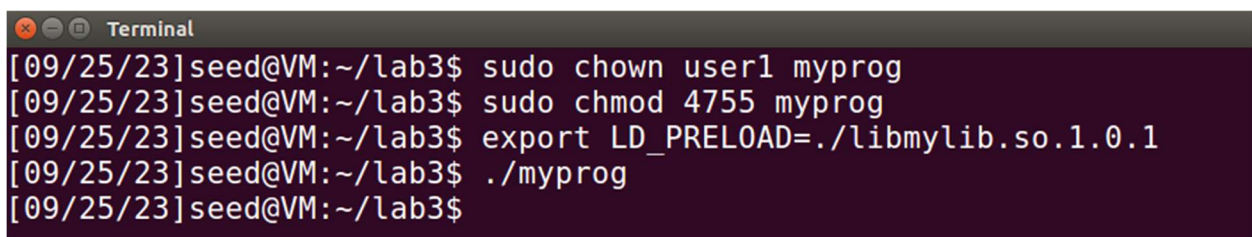
myprog as Set-UID root program, export LD_PRELOAD environment variable in the root account and run it:

A terminal window titled "Terminal" with a dark background. It shows the user `seed@VM` running `sudo -s` to become root. Then, as root, the user runs `export LD_PRELOAD=./libmylib.so.1.0.1` and `./myprog`. The output is `I am not sleeping!`.

```
[09/25/23]seed@VM:~/lab3$ sudo -s
[09/25/23]root@VM:~/lab3# export LD_PRELOAD=./libmylib.so.1.0.1
[09/25/23]root@VM:~/lab3# ./myprog
I am not sleeping!
[09/25/23]root@VM:~/lab3#
```

For the Set-UID root program executed as root with the `LD_PRELOAD` environment variable set under the root account, “I am not sleeping!” is printed. Which indicates that the custom library’s `sleep()` function is called.

myprog as a Set-UID user1 program, export LD_PRELOAD environment variable to a different user’s account and run it:

A terminal window titled "Terminal" with a dark background. It shows the user `seed@VM` running `sudo chown user1 myprog`, `sudo chmod 4755 myprog`, and then `export LD_PRELOAD=./libmylib.so.1.0.1`. Finally, the user runs `./myprog`, and the output is blank.

```
[09/25/23]seed@VM:~/lab3$ sudo chown user1 myprog
[09/25/23]seed@VM:~/lab3$ sudo chmod 4755 myprog
[09/25/23]seed@VM:~/lab3$ export LD_PRELOAD=./libmylib.so.1.0.1
[09/25/23]seed@VM:~/lab3$ ./myprog
[09/25/23]seed@VM:~/lab3$
```

Lab 3 – Set-UID

Gunnar Yonker

For the Set-UID user1 program executed by another user with LD_PRELOAD set, the program sleeps for 1 second and there is nothing printed out. This indicates that the malicious sleep() is not called.

Regular programs inherit the LD_PRELOAD environment variable and can be influenced by it. This can be exploited for malicious activities. For Set-UID programs, the LD_PRELOAD environment variable from the user's environment is not inherited when the program is executed. This is a security feature to prevent malicious library injection attacks on Set-UID programs. When executing the Set-UID program as the actual owner (root in this test), the LD_PRELOAD set in the owner's environment does not affect execution. For the last case, the behavior is consistent with the security measure against LD_PRELOAD in Set-UID programs. The main takeaway is that the dynamic loader/linker treats LD_* environment variables specifically for Set-UID programs to prevent privilege escalation attacks. Normal users cannot influence the behavior of Set-UID programs with LD_PRELOAD, but the actual owner of the program can.

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

```
gcc task8.c -o task8prog
```

```
touch testfile.txt
```

The first step can be tested by the creation of a file called testfile.txt and seeing if it can be removed using the program. First checking if the program can read the file:

```
Terminal
[09/25/23]seed@VM:~/lab3$ ls -l | grep testfile.txt
-rw-rw-r-- 1 seed  seed  22 Sep 25 16:06 testfile.txt
[09/25/23]seed@VM:~/lab3$ ./task8prog testfile.txt
This is a test file.

[09/25/23]seed@VM:~/lab3$
```

Now attempting to remove the file using:

```
./task8prog; rm testfile.txt
```

```
Terminal
[09/25/23]seed@VM:~/lab3$ ./task8prog; rm testfile.txt
Please type a file name.
[09/25/23]seed@VM:~/lab3$ ls -l | grep testfile.txt
[09/25/23]seed@VM:~/lab3$
```

The file was successfully removed. Bob would be able to remove a file that is not writable to me (Bob). Bob is able to do this because he can exploit the fact that system() invokes a shell and could interpret shell metacharacters. When system() is invoked, it calls /bin/sh -c command, where command is the string passed to system(). This context, ; allows us to execute multiple commands sequentially. So, after the cat command, the rm command is executed, deleting the target file.

execve():

The same command did not have any effect on the target file name when run. This is because `execve()` directly executes a given program and does not interpret any shell metacharacters. Hence, there is no risk of injection attacks as seen with `system()`. When Bob tries to pass the filename; `rm` as an argument, the program would try to display a file name named as “filename; `rm`” and would not interpret the `;` as a command separator like the shell would. In this case, `execve()` will treat everything as part of the filename.

In conclusion, using `system()` in Set-UID programs is dangerous because it can interpret shell metacharacters, making it vulnerable to command injection attacks. An attacker can chain commands to carry out malicious activities, potentially leading to privilege escalation. Using `execve()` is safer because it does not interpret shell metacharacters. When invoking external programs, it is always better to use a more direct function like `execve()` rather than a function like `system()` that invokes a shell. Vince’s intention to restrict Bob to only read files and not modify them failed when using the `system()` function.

Task 9: Capability Leaking

```
sudo touch /etc/zzz/
```

```
sudo chmod 0644 /etc/zzz
```

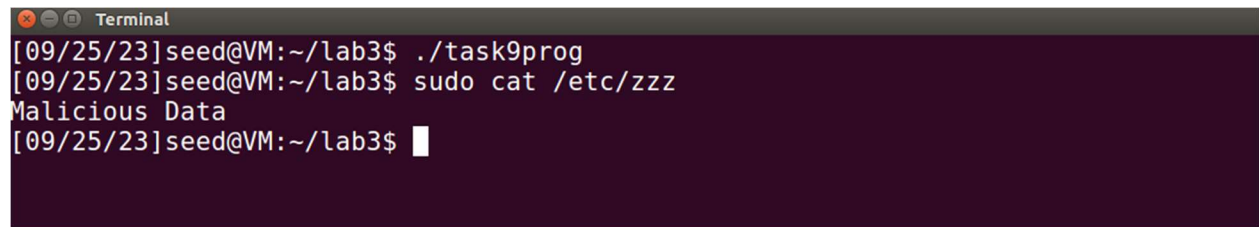
Compile Set-UID program called `task9.c` and `task9prog`

```
gcc task9.c -o task9prog
```

```
sudo chown root task9prog
```

```
sudo chmod 4755 task9prog
```

Run program and observe:



```
Terminal
[09/25/23]seed@VM:~/lab3$ ./task9prog
[09/25/23]seed@VM:~/lab3$ sudo cat /etc/zzz
Malicious Data
[09/25/23]seed@VM:~/lab3$
```

The program was able to open `/etc/zzz` successfully and got a file descriptor “`fd`”. The program opened the file and then relinquished the root privilege by calling “`setuid(getuid())`”. This action should typically make the process lose all root capabilities. However, the child process, even after relinquishing the root privileges, was still able to write “`Malicious Data`” to the file `/etc/zzz` using the file descriptor “`fd`”. This indicates that although the effective user ID was changed to a non-root user, the file descriptor, which

Lab 3 – Set-UID

Gunnar Yonker

was obtained while the process had root privileges, still retained its capability. Using `sudo cat /etc/zoo` we can confirm that “Malicious Data” has been appended to it.

Capability leaking is a major security concern in Set-UID programs. In this case, the file descriptor is the “capability: that leaked from a privileged context to a non-privileged one. Even though the effective user ID was set to a non-root user, the child process was still able to operate with the file descriptor that was opened while the program had root privileges. The key takeaway is that it’s essential not only to drop privileges appropriately but also to ensure that any capabilities (like file descriptors) that were obtained during privileged operations are handled appropriately, cleaned up, or closed before transferring control to a potentially untrusted part of the code or relinquishing privileges.