# Environment Variable and SET-UID Lab

UFCFVN-30-M - Computer & Network Security

A Research Coursework

Prepared by

Kiran Kumar Mohanraj - 20054954

MSc Cyber Security

University of the West of England, Bristol

# TABLE OF CONTENTS

# 1 LAB TASK

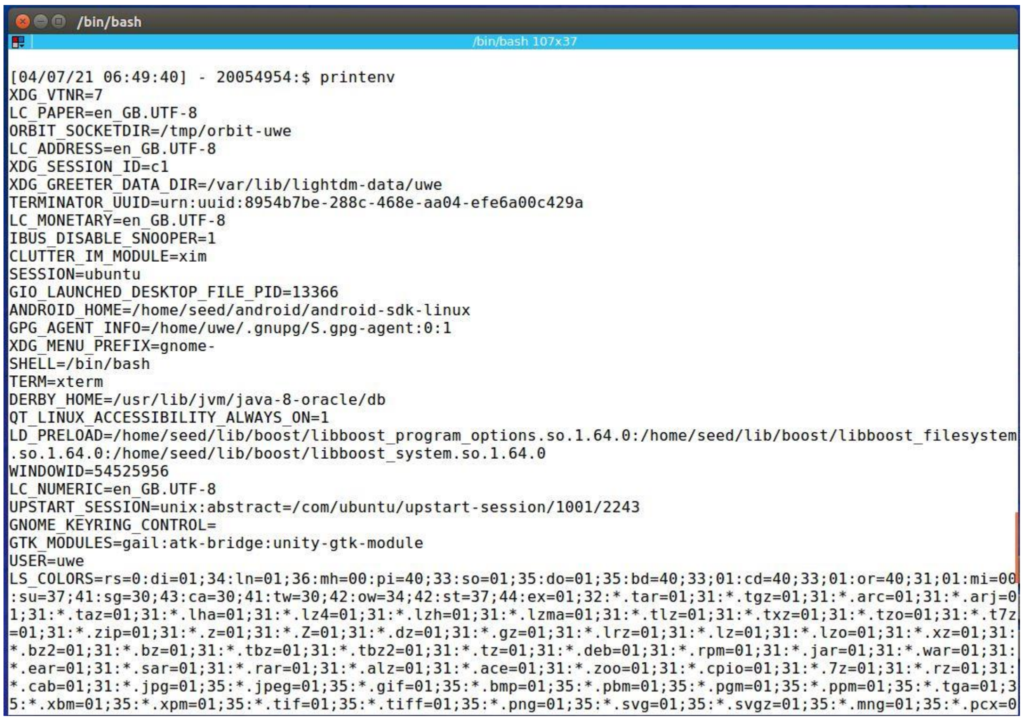
## 1.1 Task 1: Manipulating Environment Variables

a)

The functionality of both /usr/sbin/nologin and /bin/false is to deny shell access to a particular user account. When a particular user attempts to login into the machine, /bin/false denies the access and does nothing and just exits with a status code. Whereas /usr/sbin/nologin, exits with a message similar to this, “The account is currently unavailable”. It can display custom message present in /etc/nologin.txt.

b)

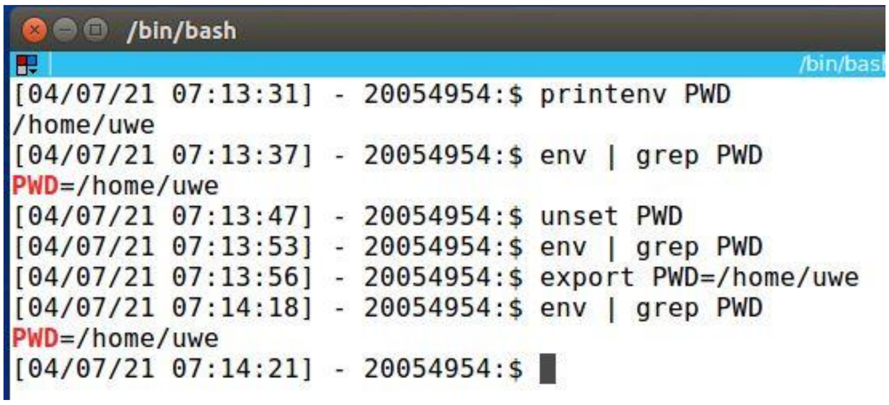
*printenv* command is used to print all the environment variables. Figure 1 shows the output the executed command. The *env* command too produces the same result. The output provides the environmental variable name and its value pairs.



```
[04/07/21 06:49:40] - 20054954:$ printenv
XDG_VTNR=7
LC_PAPER=en_GB.UTF-8
ORBIT_SOCKETDIR=/tmp/orbit-uwe
LC_ADDRESS=en_GB.UTF-8
XDG_SESSION_ID=c1
XDG_GREETER_DATA_DIR=/var/lib/lightdm-data/uwe
TERMINATOR_UUID=urn:uuid:8954b7be-288c-468e-aa04-efe6a00c429a
LC_MONETARY=en_GB.UTF-8
IBUS_DISABLE_SNOOPER=1
CLUTTER_IM_MODULE=xim
SESSION=ubuntu
GIO_LAUNCHED_DESKTOP_FILE_PID=13366
ANDROID_HOME=/home/seed/android/android-sdk-linux
GPG_AGENT_INFO=/home/uwe/.gnupg/S.gpg-agent:0:1
XDG_MENU_PREFIX=gnome-
SHELL=/bin/bash
TERM=xterm
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=54525956
LC_NUMERIC=en_GB.UTF-8
UPSTART_SESSION=unix:abstract=/com/ubuntu/upstart-session/1001/2243
GNOME_KEYRING_CONTROL=
GTK_MODULES=gail:atk-bridge:unity-gtk-module
USER=uwe
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=0
1;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:
*.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:*.jpg=01;35:*.jpeg=01;35:*.gif=01;35:*.bmp=01;35:*.pbm=01;35:*.pgm=01;35:*.ppm=01;35:*.tga=01;3
5:*.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=0
```

*Figure 1 Output of printenv command showing all environmental variables*

Figure 2 shows the execution for searching a particular environmental variable and modifying its value. *printenv PWD* command just displays the value of the environmental variable PWD whereas *env | grep PWD* displays the all the variables in env with a substring PWD. The *unset* command is used to delete the environmental variable. Figure 2 shows that once PWD is unset, it does not display any output for *env | grep PWD* command. The export command is used to set the environmental variable value as seen in Figure 2.

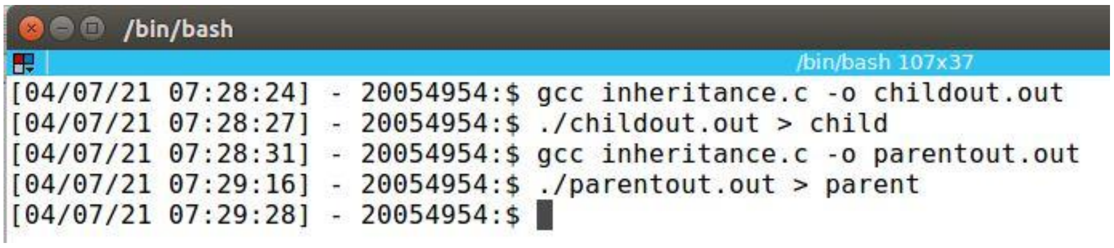


```
[04/07/21 07:13:31] - 20054954:$ printenv PWD
/home/uwe
[04/07/21 07:13:37] - 20054954:$ env | grep PWD
PWD=/home/uwe
[04/07/21 07:13:47] - 20054954:$ unset PWD
[04/07/21 07:13:53] - 20054954:$ env | grep PWD
[04/07/21 07:13:56] - 20054954:$ export PWD=/home/uwe
[04/07/21 07:14:18] - 20054954:$ env | grep PWD
PWD=/home/uwe
[04/07/21 07:14:21] - 20054954:$ 
```

*Figure 2 Demonstration of searching for a particular environmental variable and modifying its value*

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

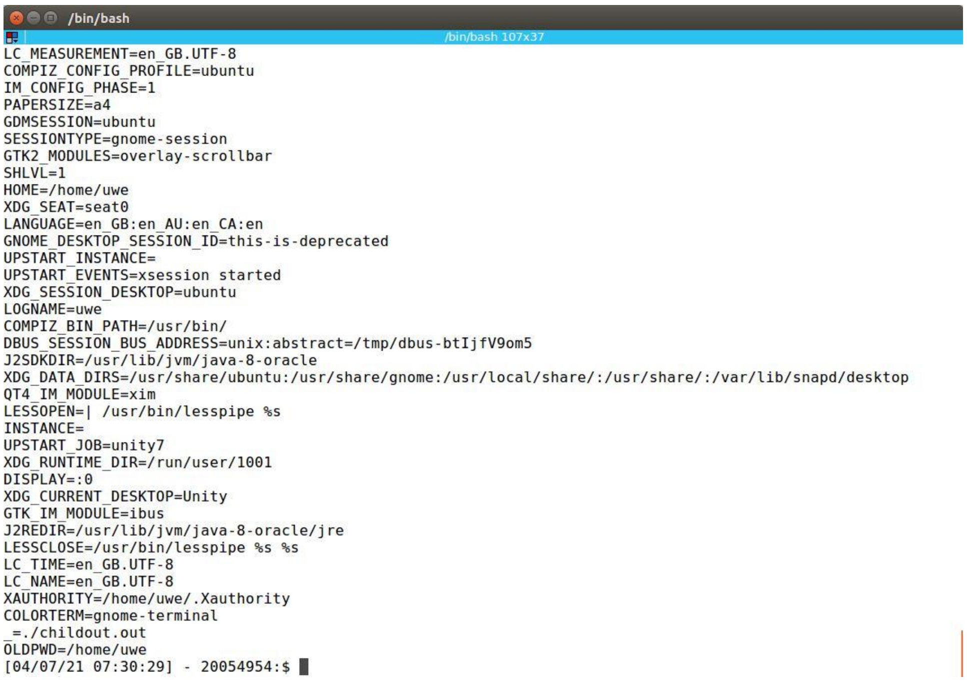
Figure 3 shows the execution of the provided program and their corresponding outputs are stored in child and parent files.



```
[04/07/21 07:28:24] - 20054954:$ gcc inheritance.c -o childout.out
[04/07/21 07:28:27] - 20054954:$ ./childout.out > child
[04/07/21 07:28:31] - 20054954:$ gcc inheritance.c -o parentout.out
[04/07/21 07:29:16] - 20054954:$ ./parentout.out > parent
[04/07/21 07:29:28] - 20054954:$ 
```

*Figure 3 Execution of C program and storing the corresponding output files.*

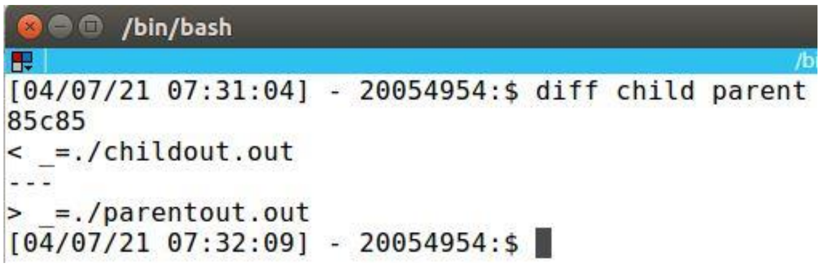
Figure 4 shows the output of the inheritance.c file containing child process with printenv. The file contains all the environment variables of the child process.



```
LC_MEASUREMENT=en_GB.UTF-8
COMPIZ_CONFIG_PROFILE=ubuntu
IM_CONFIG_PHASE=1
PAPERSIZE=a4
GDMSESSION=ubuntu
SESSIONTYPE=gnome-session
GTK2_MODULES=overlay-scrollbar
SHLVL=1
HOME=/home/uwe
XDG_SEAT=seat0
LANGUAGE=en_GB:en_AU:en_CA:en
GNOME_DESKTOP_SESSION_ID=this-is-deprecated
UPSTART_INSTANCE=
UPSTART_EVENTS=xsession started
XDG_SESSION_DESKTOP=ubuntu
LOGNAME=uwe
COMPIZ_BIN_PATH=/usr/bin/
DBUS_SESSION_BUS_ADDRESS=unix:abstract=/tmp/dbus-btIjfV9om5
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/1001
DISPLAY=:0
XDG_CURRENT_DESKTOP=Unity
GTK_IM_MODULE=ibus
J2REDIR=/usr/lib/jvm/java-8-oracle/jre
LESSCLOSE=/usr/bin/lesspipe %s %s
LC_TIME=en_GB.UTF-8
LC_NAME=en_GB.UTF-8
XAUTHORITY=/home/uwe/.Xauthority
COLORTERM=gnome-terminal
_=./childout.out
OLDPWD=/home/uwe
[04/07/21 07:30:29] - 20054954:$ 
```

Figure 4 Contants of the file child

To find the difference between the output of child process with printenv and parent process with printenv, we use diff command. “diff file1 file2”.



```
[04/07/21 07:31:04] - 20054954:$ diff child parent
85c85
< _=./childout.out
---
> _=./parentout.out
[04/07/21 07:32:09] - 20054954:$ 
```

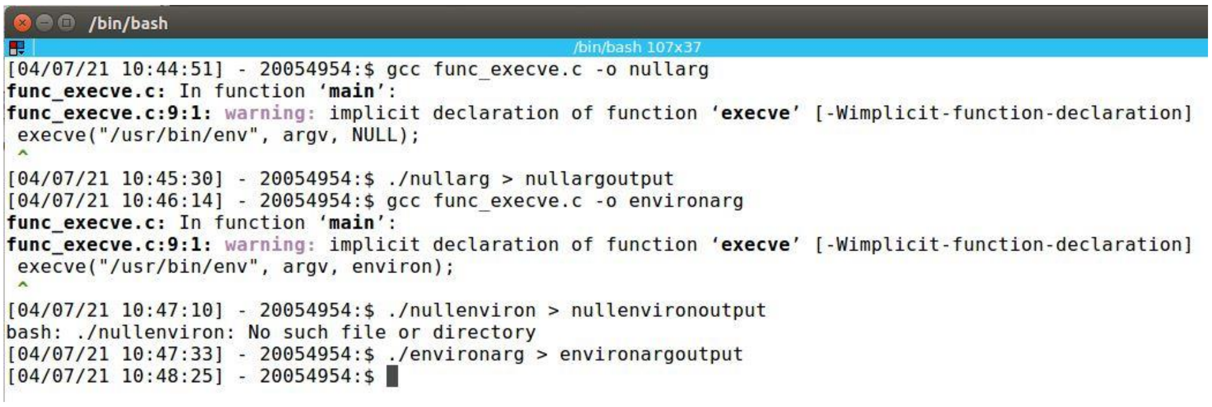
Figure 5 Demonstration of difference between the two files

In Figure 5, 85c85 means that line 85 of child and parent is changed. The < denotes the changes in the file1 which is child and > denotes the changes in file2 which is parent.

From this output, we can infer that _ environment variable stores the last command executed as its value. This shows that only _ variable is changed according to the compiled apart from that there is no change.

## 1.3 Task 3: Environment Variables and execve ()

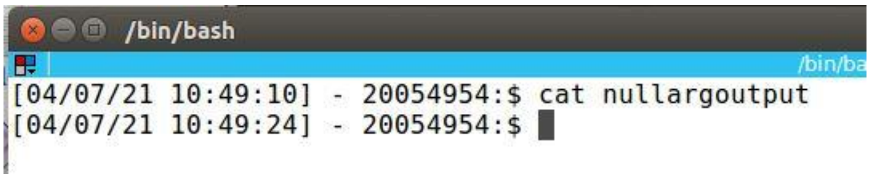
Figure 6 shows the execution of the program and the outputs were stored in respective files. The output of program with NULL as argument is stored in *nullargoutput* file and the output of program with environ as argument is stored in *environargoutput* file.



```
[04/07/21 10:44:51] - 20054954:$ gcc func_execve.c -o nullarg
func_execve.c: In function ‘main’:
func_execve.c:9:1: warning: implicit declaration of function ‘execve’ [-Wimplicit-function-declaration]
 execve("/usr/bin/env", argv, NULL);
 ^
[04/07/21 10:45:30] - 20054954:$ ./nullarg > nullargoutput
[04/07/21 10:46:14] - 20054954:$ gcc func_execve.c -o environarg
func_execve.c: In function ‘main’:
func_execve.c:9:1: warning: implicit declaration of function ‘execve’ [-Wimplicit-function-declaration]
 execve("/usr/bin/env", argv, environ);
 ^
[04/07/21 10:47:10] - 20054954:$ ./nullenviron > nullenvironoutput
bash: ./nullenviron: No such file or directory
[04/07/21 10:47:33] - 20054954:$ ./environarg > environargoutput
[04/07/21 10:48:25] - 20054954:$ █
```

*Figure 6 Execution of Task 3 program and storing the outputs in respective files*

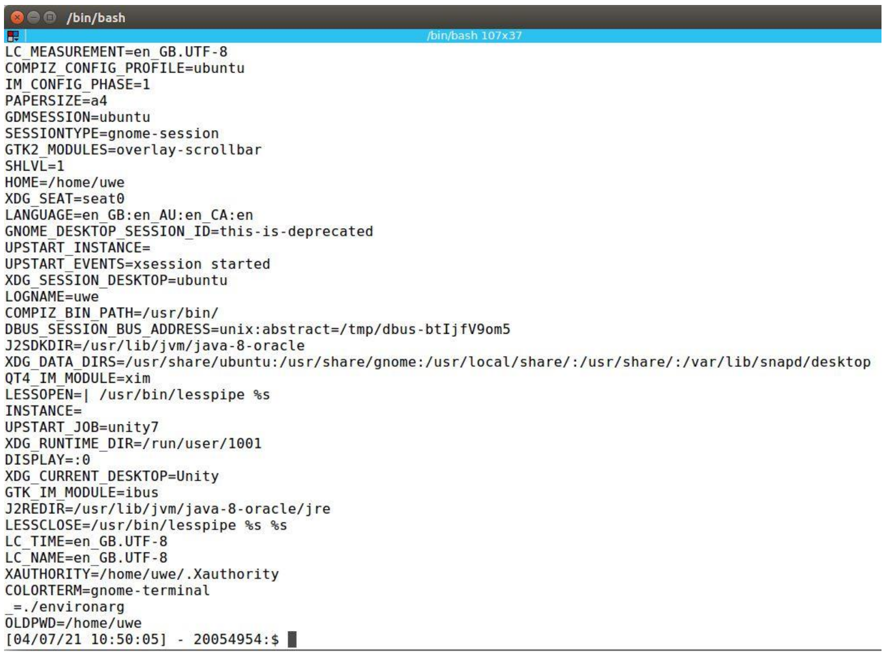
Figure 7 shows that the output of program with NULL as argument does not print any environment variable. Although the program had environ variable, the third argument is passed as NULL because of which no environment variable were associated with this process.



```
[04/07/21 10:49:10] - 20054954:$ cat nullargoutput
[04/07/21 10:49:24] - 20054954:$ █
```

*Figure 7 Output of Task 3 program with NULL as argument*

Figure 8 shows the output of the program with environ as third argument. Since the environ variable, which contained all environment variable, was passed the output produced contained the environment variable of the current process. The third argument is the reason for execve() function to get the environment variable.

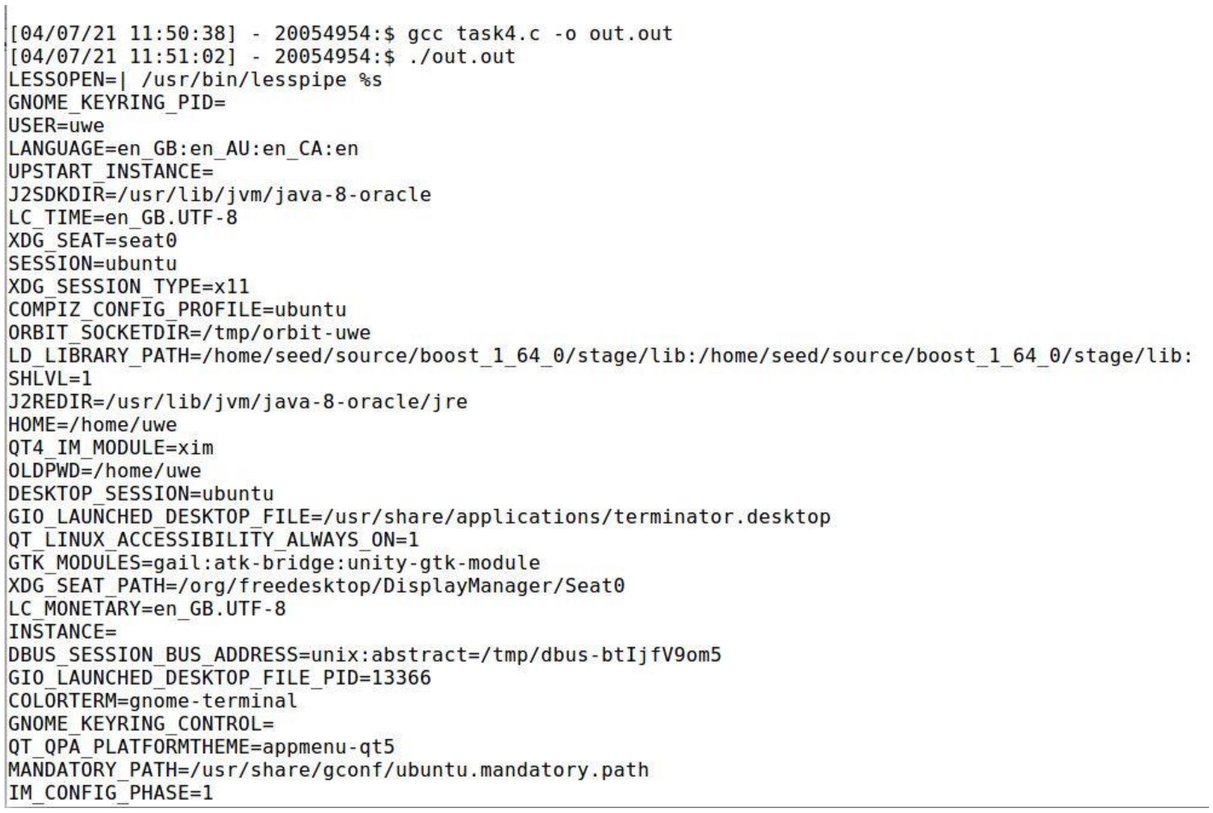


```
LC_MEASUREMENT=en_GB.UTF-8
COMPIZ_CONFIG_PROFILE=ubuntu
IM_CONFIG_PHASE=1
PAPERSIZE=a4
GDMSESSION=ubuntu
SESSIONTYPE=gnome-session
GTK2_MODULES=overlay-scrollbar
SHLVL=1
HOME=/home/uwe
XDG_SEAT=seat0
LANGUAGE=en_GB:en_AU:en_CA:en
GNOME_DESKTOP_SESSION_ID=this-is-deprecated
UPSTART_INSTANCE=
UPSTART_EVENTS=xsession started
XDG_SESSION_DESKTOP=ubuntu
LOGNAME=uwe
COMPIZ_BIN_PATH=/usr/bin/
DBUS_SESSION_BUS_ADDRESS=unix:abstract=/tmp/dbus-btIjfV9om5
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/1001
DISPLAY=:0
XDG_CURRENT_DESKTOP=Unity
GTK_IM_MODULE=ibus
J2REDIR=/usr/lib/jvm/java-8-oracle/jre
LESSCLOSE=/usr/bin/lesspipe %s %s
LC_TIME=en_GB.UTF-8
LC_NAME=en_GB.UTF-8
XAUTHORITY=/home/uwe/.Xauthority
COLORTERM=gnome-terminal
_=./environarg
OLDPWD=/home/uwe
[04/07/21 10:50:05] - 20054954:$ █
```

*Figure 8 Output of Task 3 program with environ as argument*

## 1.4 Task 4: Environment Variables and system()

Figure 9 shows that the Task 4 program displays the environment variables of the current process without explicitly providing the variables. This is because of the system() function which passes the environment variable to the called function /bin/sh.

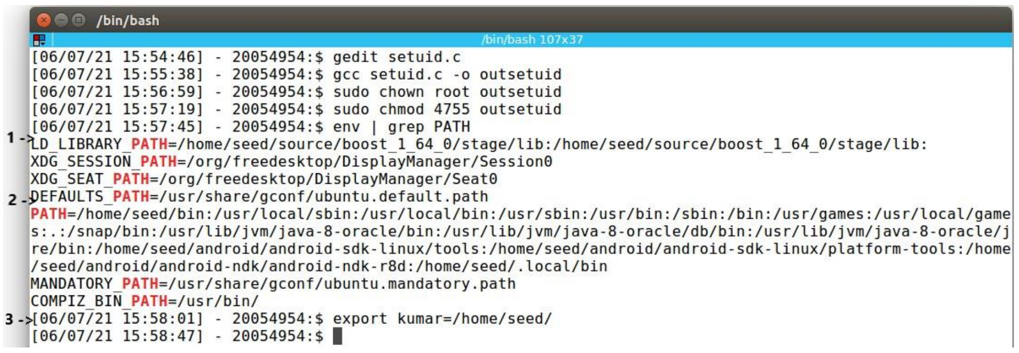


```
[04/07/21 11:50:38] - 20054954:$ gcc task4.c -o out.out
[04/07/21 11:51:02] - 20054954:$ ./out.out
LESSOPEN=| /usr/bin/lesspipe %s
GNOME_KEYRING_PID=
USER=uwe
LANGUAGE=en_GB:en_AU:en_CA:en
UPSTART_INSTANCE=
J2SDKDIR=/usr/lib/jvm/java-8-oracle
LC_TIME=en_GB.UTF-8
XDG_SEAT=seat0
SESSION=ubuntu
XDG_SESSION_TYPE=x11
COMPIZ_CONFIG_PROFILE=ubuntu
ORBIT_SOCKETDIR=/tmp/orbit-uwe
LD_LIBRARY_PATH=/home/seed/source/boost_1_64_0/stage/lib:/home/seed/source/boost_1_64_0/stage/lib:
SHLVL=1
J2REDIR=/usr/lib/jvm/java-8-oracle/jre
HOME=/home/uwe
QT4_IM_MODULE=xim
OLDPWD=/home/uwe
DESKTOP_SESSION=ubuntu
GIO_LAUNCHED_DESKTOP_FILE=/usr/share/applications/terminator.desktop
QT_LINUX_ACCESSIBILITY_ALWAYS_ON=1
GTK_MODULES=gail:atk-bridge:unity-gtk-module
XDG_SEAT_PATH=/org/freedesktop/DisplayManager/Seat0
LC_MONETARY=en_GB.UTF-8
INSTANCE=
DBUS_SESSION_BUS_ADDRESS=unix:abstract=/tmp/dbus-btIjfV9om5
GIO_LAUNCHED_DESKTOP_FILE_PID=13366
COLORTERM=gnome-terminal
GNOME_KEYRING_CONTROL=
QT_QPA_PLATFORMTHEME=appmenu-qt5
MANDATORY_PATH=/usr/share/gconf/ubuntu.mandatory.path
IM_CONFIG_PHASE=1
```

*Figure 9 Execution of Task 4 program*

## 1.5 Task 5: Environment Variable and Set-UID Programs

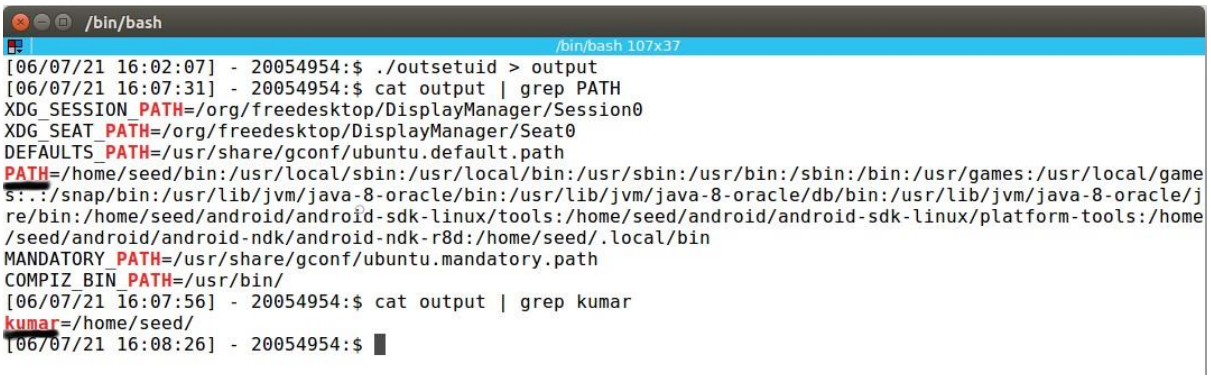
Firstly, the Task5 program was compiled and the output is stored in *outsetuid* file. The ownership and permission of this file is changed to root user. The program is changed to Set-UID program by setting the set-uid bit. Secondly, we set three environment variables for this task. Figure 10 shows that the PATH and LD LIBRARY PATH has already been set.



```
    [06/07/21 15:54:46] - 20054954:$ gedit setuid.c
    [06/07/21 15:55:38] - 20054954:$ gcc setuid.c -o outsetuid
    [06/07/21 15:56:59] - 20054954:$ sudo chown root outsetuid
    [06/07/21 15:57:19] - 20054954:$ sudo chmod 4755 outsetuid
    [06/07/21 15:57:45] - 20054954:$ env | grep PATH
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
2 ->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/game
    s:.:/snap/bin:/usr/lib/jvm/java-8-oracle/bin:/usr/lib/jvm/java-8-oracle/db/bin:/usr/lib/jvm/java-8-oracle/j
    re/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/
3 ->[06/07/21 15:58:01] - 20054954:$ export kumar=/home/seed/
    [06/07/21 15:58:47] - 20054954:$ █
```

*Figure 10 Demonstration of changing a program to Set-UID program*

Once the setup is done, we run the program and check whether the child process inherits all the environment variables of the parent process. From Figure 11, we can conclude that only PATH and kumar variable has been inherited. Since, the LD_LIBRARY_PATH variable's real user id and inherited user id are different, this variable is not inherited.

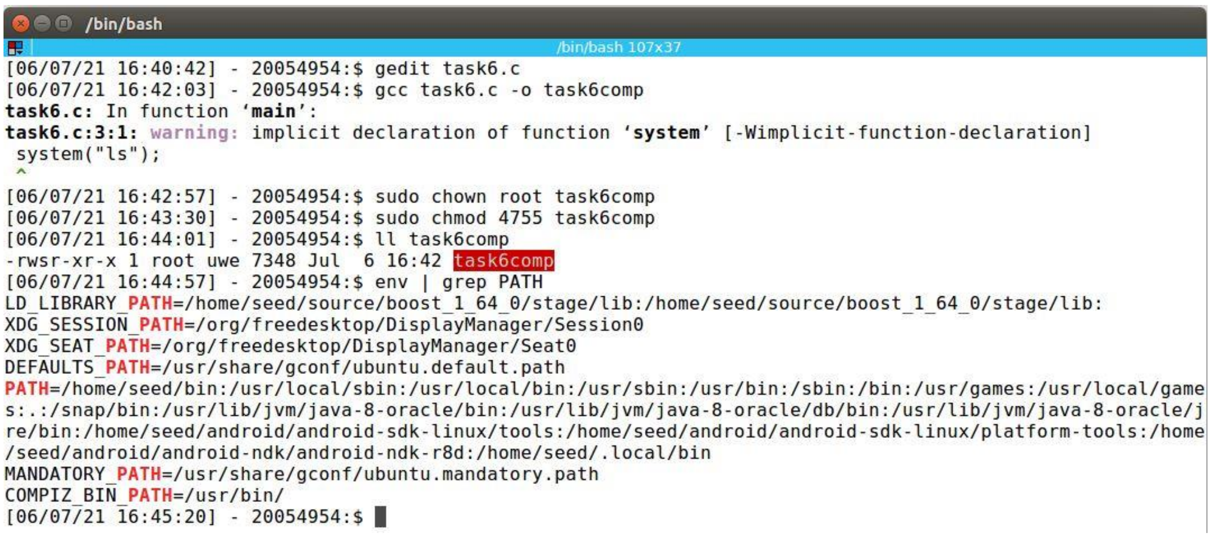


```
[06/07/21 16:02:07] - 20054954:$ ./outsetuid > output
[06/07/21 16:07:31] - 20054954:$ cat output | grep PATH
XDG_SESSION_PATH=/org/freedesktop/DisplayManager/Session0
XDG_SEAT_PATH=/org/freedesktop/DisplayManager/Seat0
DEFAULTS_PATH=/usr/share/gconf/ubuntu.default.path
PATH=/home/seed/bin:/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin:/usr/games:/usr/local/game
s:.:/snap/bin:/usr/lib/jvm/java-8-oracle/bin:/usr/lib/jvm/java-8-oracle/db/bin:/usr/lib/jvm/java-8-oracle/j
re/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/
[06/07/21 16:07:56] - 20054954:$ cat output | grep kumar
kumar=/home/seed/
[06/07/21 16:08:26] - 20054954:$ 
```

*Figure 11 Output of the Set-UID program*

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

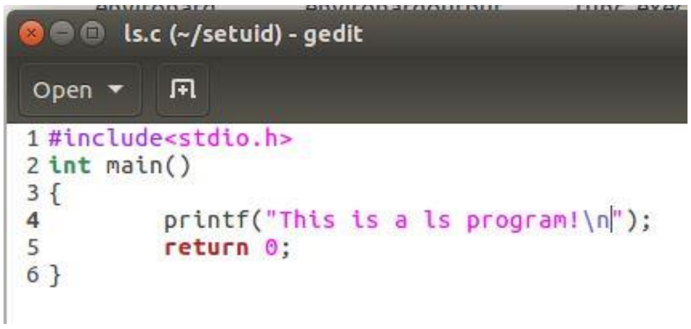
The task6 program is compiled and stored in task6comp. Figure 12 shows the method of converting the program to a Set-UID program by changing to root user and setting the set-uid bit. The figure also shows the present value for the PATH variable.



```
[06/07/21 16:40:42] - 20054954:$ gedit task6.c
[06/07/21 16:42:03] - 20054954:$ gcc task6.c -o task6comp
task6.c: In function 'main':
task6.c:3:1: warning: implicit declaration of function 'system' [-Wimplicit-function-declaration]
 system("ls");
 ^
[06/07/21 16:42:57] - 20054954:$ sudo chown root task6comp
[06/07/21 16:43:30] - 20054954:$ sudo chmod 4755 task6comp
[06/07/21 16:44:01] - 20054954:$ ll task6comp
-rwsr-xr-x 1 root uwe 7348 Jul  6 16:42 task6comp
[06/07/21 16:44:57] - 20054954:$ 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/game
s:.:/snap/bin:/usr/lib/jvm/java-8-oracle/bin:/usr/lib/jvm/java-8-oracle/db/bin:/usr/lib/jvm/java-8-oracle/j
re/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/
[06/07/21 16:45:20] - 20054954:$ 
```

*Figure 12 Demonstration of creating a Set-UID program*

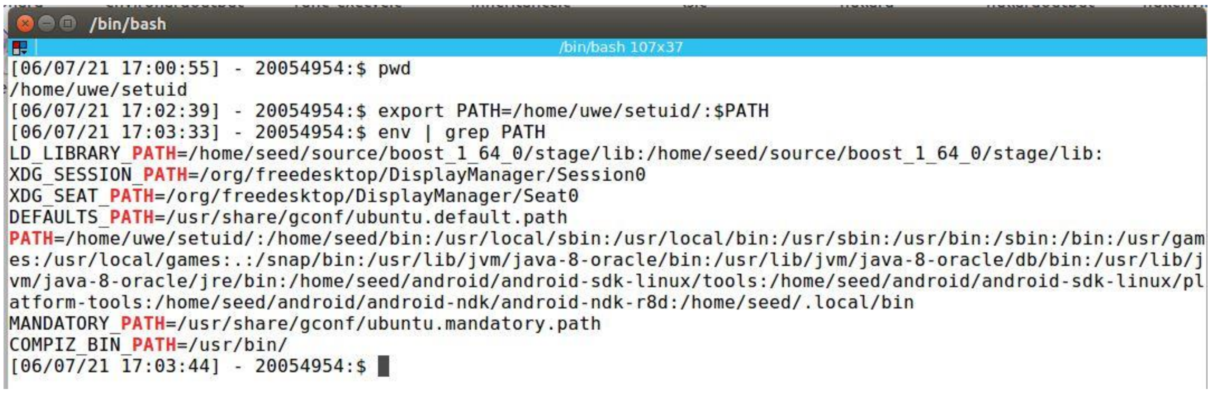
Figure 13 shows the ls program created for the Set-UID program to execute instead of /bin/ls.



```c
#include<stdio.h>
int main()
{
        printf("This is a ls program!\n");
        return 0;
}
```

Figure 13 ls program created for Set-UID program to execute

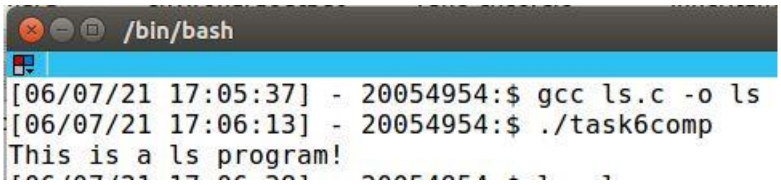
Figure 14 shows the steps to change the absolute path of the PATH variable to relative path where the ls program is present. The relative path is /home/uwe/setuid.



```
[06/07/21 17:00:55] - 20054954:$ pwd
/home/uwe/setuid
[06/07/21 17:02:39] - 20054954:$ export PATH=/home/uwe/setuid/:$PATH
[06/07/21 17:03:33] - 20054954:$ 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/uwe/setuid/:/home/seed/bin:/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin:/usr/gam
es:/usr/local/games:.:/snap/bin:/usr/lib/jvm/java-8-oracle/bin:/usr/lib/jvm/java-8-oracle/db/bin:/usr/lib/j
vm/java-8-oracle/jre/bin:/home/seed/android/android-sdk-linux/tools:/home/seed/android/android-sdk-linux/pl
atform-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/
[06/07/21 17:03:44] - 20054954:$ 
```

Figure 14 Demonstration of changing the PATH variable

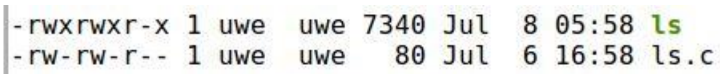
Once the task6comp is executed instead of executing the /bin/ls, the program produces the output of ls program which is "This is a ls program!" (as shown in Figure 15). This shows that a PATH environment variable can be changed to a particular directory to execute a malicious program. This is because of providing a relative path instead of absolute path and running system() which creates a shell to search for ls in the location provided in PATH variable.



```
[06/07/21 17:05:37] - 20054954:$ gcc ls.c -o ls
[06/07/21 17:06:13] - 20054954:$ ./task6comp
This is a ls program!
```

Figure 15 Output of task6 program with output of ls program

Figure 16 shows that the compilation of ls program contains normal user priviledge.



```
-rwxrwxr-x 1 uwe  uwe 7340 Jul  8 05:58 ls
-rw-rw-r-- 1 uwe  uwe   80 Jul  6 16:58 ls.c
```

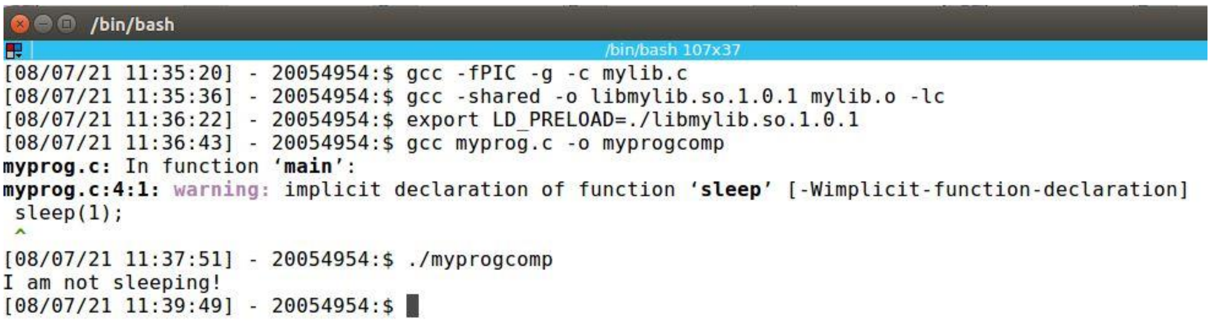
Figure 16 Privileges of ls program

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

Firstly, the mylib.c program which overrides the system's sleep function was created and compiled. The sleep function in this program just prints the statement as output. The executable output of this program is provided as value to LD_PRELOAD variable. Figure 17 shows the demonstration of these steps.

a) The myprog a regular program, executed as normal user.

When the program was run in normal user, the sleep function defined by us in the myprog.c was executed. Thus, printing the below shown statement.

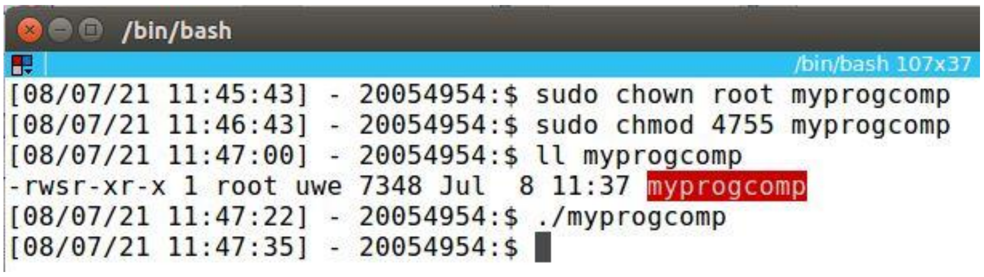


```
[08/07/21 11:35:20] - 20054954:$ gcc -fPIC -g -c mylib.c
[08/07/21 11:35:36] - 20054954:$ gcc -shared -o libmylib.so.1.0.1 mylib.o -lc
[08/07/21 11:36:22] - 20054954:$ export LD_PRELOAD=./libmylib.so.1.0.1
[08/07/21 11:36:43] - 20054954:$ gcc myprog.c -o myprogcomp
myprog.c: In function 'main':
myprog.c:4:1: warning: implicit declaration of function 'sleep' [-Wimplicit-function-declaration]
 sleep(1);
 ^
[08/07/21 11:37:51] - 20054954:$ ./myprogcomp
I am not sleeping!
[08/07/21 11:39:49] - 20054954:$ 
```

*Figure 17 The myprog compiled and executed as normal user*

b) The myprog a Set-UID root program, executed as normal user.

The owner of the myprogcomp file is changed to root user and the setuid bit is set as shown in Figure 18. Thus, converting the regular program to Set-UID root program. When the file is executed as normal user, the system's sleep function is run rather that our sleep function, because of which no output was printed.

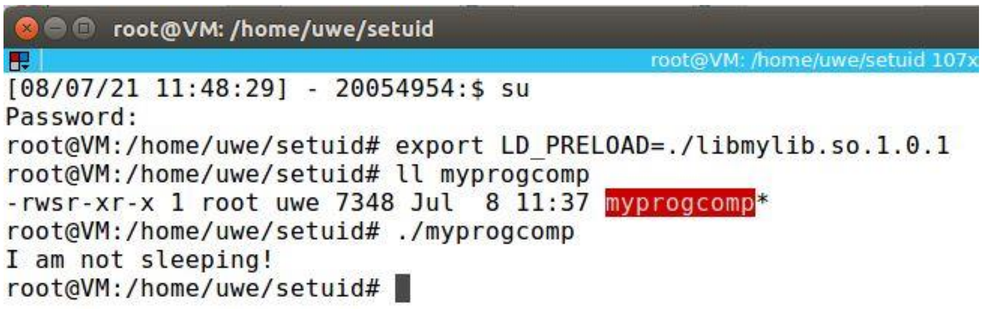


```
[08/07/21 11:45:43] - 20054954:$ sudo chown root myprogcomp
[08/07/21 11:46:43] - 20054954:$ sudo chmod 4755 myprogcomp
[08/07/21 11:47:00] - 20054954:$ ll myprogcomp
-rwsr-xr-x 1 root uwe 7348 Jul  8 11:37 myprogcomp
[08/07/21 11:47:22] - 20054954:$ ./myprogcomp
[08/07/21 11:47:35] - 20054954:$ 
```

*Figure 18 Conversion of myprogcomp to Set-UID program and executed*

c) The myprog a Set-UID root program, executed as root user.

The Set-UID program was executed as root user. Since the program is owned by the root user and is executed by it, the system's sleep() was overridden and the print statement was executed as shown in Figure 19.

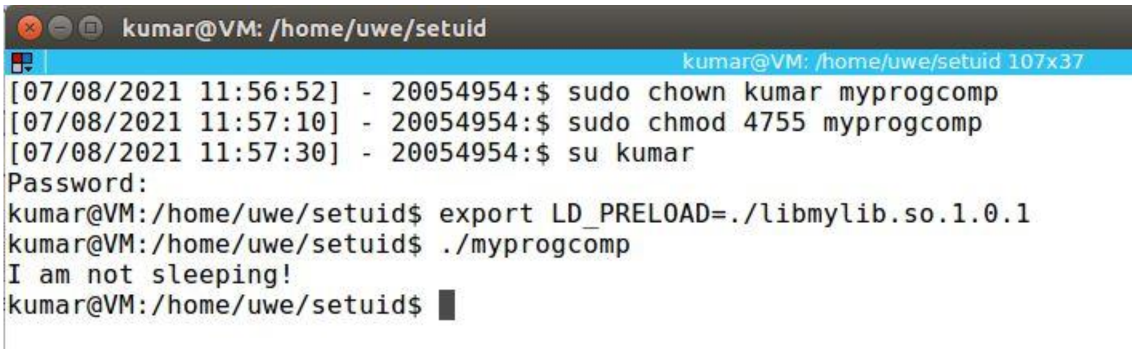


```
root@VM: /home/uwe/setuid
root@VM: /home/uwe/setuid 107x
[08/07/21 11:48:29] - 20054954:$ su
Password:
root@VM:/home/uwe/setuid# export LD_PRELOAD=./libmylib.so.1.0.1
root@VM:/home/uwe/setuid# ll myprogcomp
-rwsr-xr-x 1 root uwe 7348 Jul  8 11:37 myprogcomp*
root@VM:/home/uwe/setuid# ./myprogcomp
I am not sleeping!
root@VM:/home/uwe/setuid# 
```

*Figure 19 Executing myprogcomp as root user*

d) The myprog as Set-UID user1 program, executed as user1.

For this, the owner of the file is changed to kumar and the setuid bit was set asn shown in Figure 20. Similar to previous root user example, the system's sleep() was overridden.



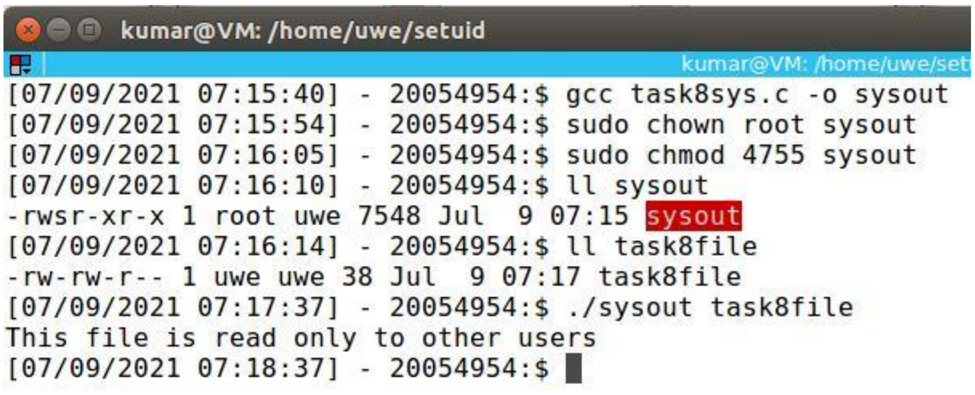
```
kumar@VM: /home/uwe/setuid
kumar@VM: /home/uwe/setuid 107x37
[07/08/2021 11:56:52] - 20054954:$ sudo chown kumar myprogcomp
[07/08/2021 11:57:10] - 20054954:$ sudo chmod 4755 myprogcomp
[07/08/2021 11:57:30] - 20054954:$ su kumar
Password:
kumar@VM:/home/uwe/setuid$ export LD_PRELOAD=./libmylib.so.1.0.1
kumar@VM:/home/uwe/setuid$ ./myprogcomp
I am not sleeping!
kumar@VM:/home/uwe/setuid$ 
```

*Figure 20 Executing myprogcomp as kumar user*

In the first, third and fourth cases, the system's sleep() was overridden because the effective user ID of the owner and the real user ID of the process was same. In the second case, because of Set-UID program's security mechanism, the effective user ID was root user's ID and the real user ID was normal user's ID (uwe user). Thus, the system defined sleep function was called.

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

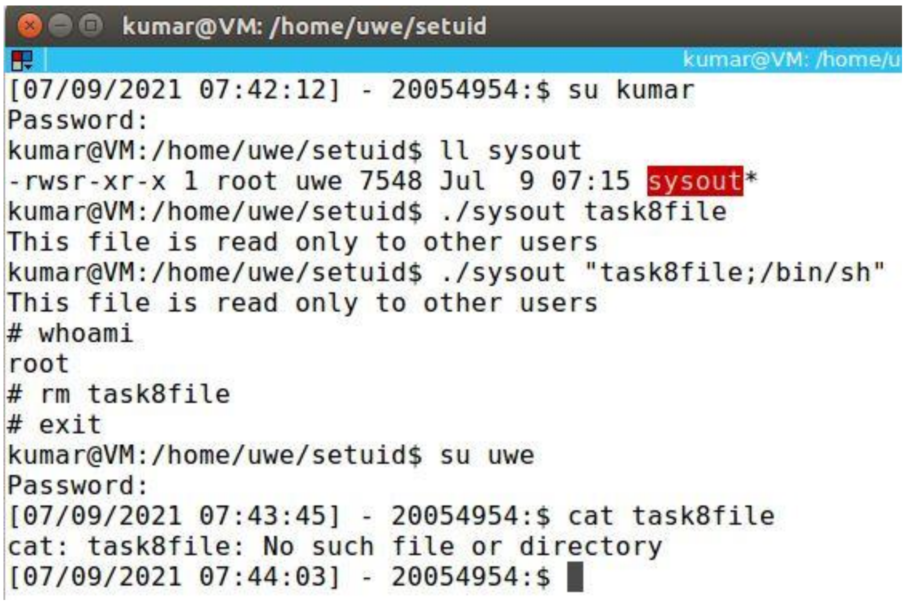
Firstly, the provided Task8 program was compiled and stored into the file named *sysout*. This file was converted into Set-UID root program as shown in Figure 21. Simultaneously, a file was created for other users who has read only access. Figure 21 shows the owner and permission of the sysout file and Task8file. When the Task8file was provided as argument to sysout, the contents of the file were displayed.



```
[07/09/2021 07:15:40] - 20054954:$ gcc task8sys.c -o sysout
[07/09/2021 07:15:54] - 20054954:$ sudo chown root sysout
[07/09/2021 07:16:05] - 20054954:$ sudo chmod 4755 sysout
[07/09/2021 07:16:10] - 20054954:$ ll sysout
-rwsr-xr-x 1 root uwe 7548 Jul  9 07:15 sysout
[07/09/2021 07:16:14] - 20054954:$ ll task8file
-rw-rw-r-- 1 uwe uwe 38 Jul  9 07:17 task8file
[07/09/2021 07:17:37] - 20054954:$ ./sysout task8file
This file is read only to other users
[07/09/2021 07:18:37] - 20054954:$ 
```

*Figure 21 Creation of Set-UID program with system() for other users to access read only files*

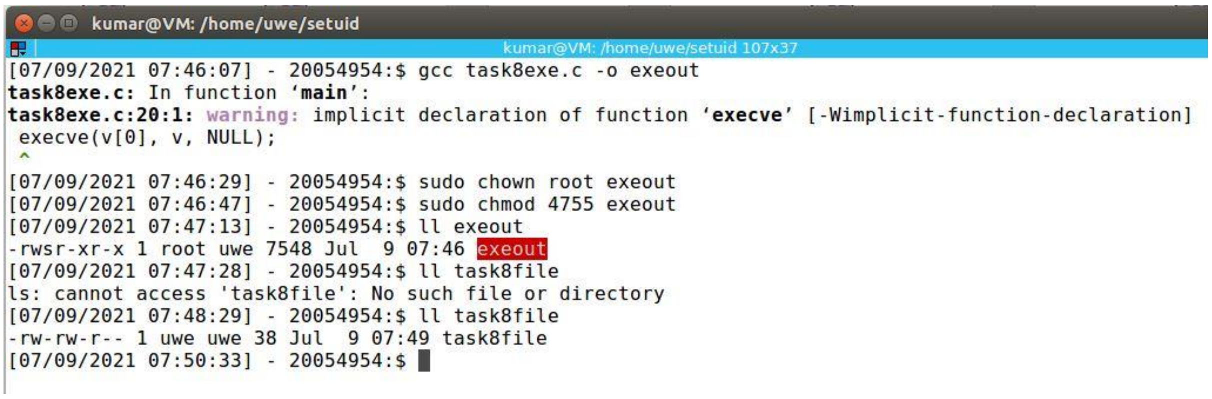
Secondly, considering Bob is using the kumar account, the program ran normally when the file was provided as argument. When “task8file;/bin/sh” was provided, the contents of the task8file were displayed and /bin/sh was executed as shell program with root privileges. Using this kumar user was able to delete any file for which the user did not have access to. Figure 22 shows the demonstration of the above steps. The reason for Bob deleting the file was the system() in the program does not differentiate the command and user input because of which the user input was considered as command instead of document name.



```
[07/09/2021 07:42:12] - 20054954:$ su kumar
Password:
kumar@VM:/home/uwe/setuid$ ll sysout
-rwsr-xr-x 1 root uwe 7548 Jul  9 07:15 sysout*
kumar@VM:/home/uwe/setuid$ ./sysout task8file
This file is read only to other users
kumar@VM:/home/uwe/setuid$ ./sysout "task8file;/bin/sh"
This file is read only to other users
# whoami
root
# rm task8file
# exit
kumar@VM:/home/uwe/setuid$ su uwe
Password:
[07/09/2021 07:43:45] - 20054954:$ cat task8file
cat: task8file: No such file or directory
[07/09/2021 07:44:03] - 20054954:$ 
```

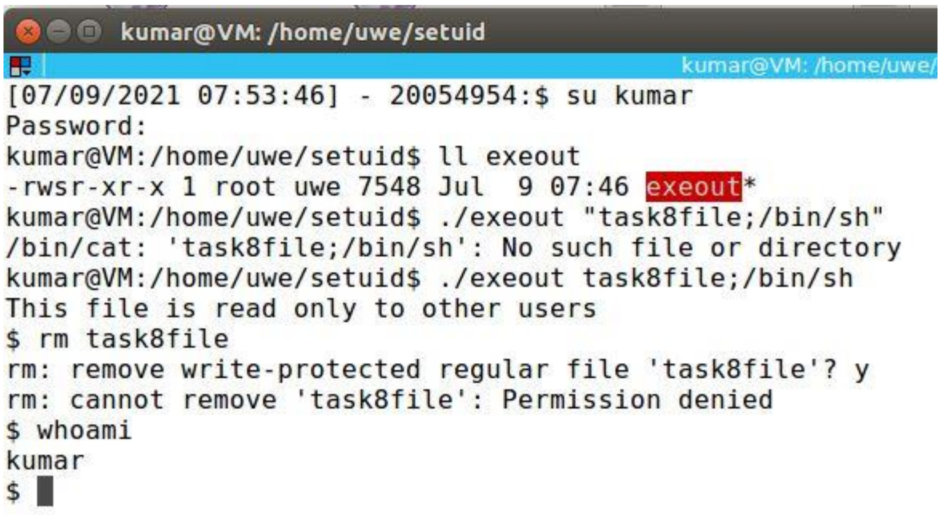
*Figure 22 Expoiting the Set-UID program to delete files.*

Similar to the previous steps, a Set-UID program was created with execve() as shown in Figure 23. When the kumar user ran the compiled file by providing the argument, the user wasn’t able to gain root privileges. In this case, we used execve() which considers the arguments as user input rather than a command. When task8file;/bin/sh was provided, the execve() considers ‘;’ as end of first input and considers the second input as command and runs the shell owned by the kumar user(as shown in Figure 24). Thus, the attack has been avoided by using execve() instead of system().



```
kumar@VM: /home/uwe/setuid
kumar@VM: /home/uwe/setuid 107x37
[07/09/2021 07:46:07] - 20054954:$ gcc task8exe.c -o exeout
task8exe.c: In function ‘main’:
task8exe.c:20:1: warning: implicit declaration of function ‘execve’ [-Wimplicit-function-declaration]
 execve(v[0], v, NULL);
 ^
[07/09/2021 07:46:29] - 20054954:$ sudo chown root exeout
[07/09/2021 07:46:47] - 20054954:$ sudo chmod 4755 exeout
[07/09/2021 07:47:13] - 20054954:$ ll exeout
-rwsr-xr-x 1 root uwe 7548 Jul  9 07:46 exeout
[07/09/2021 07:47:28] - 20054954:$ ll task8file
ls: cannot access 'task8file': No such file or directory
[07/09/2021 07:48:29] - 20054954:$ ll task8file
-rw-rw-r-- 1 uwe uwe 38 Jul  9 07:49 task8file
[07/09/2021 07:50:33] - 20054954:$
```

Figure 23 Creation of Set-UID program with execve() for other users to access read only files

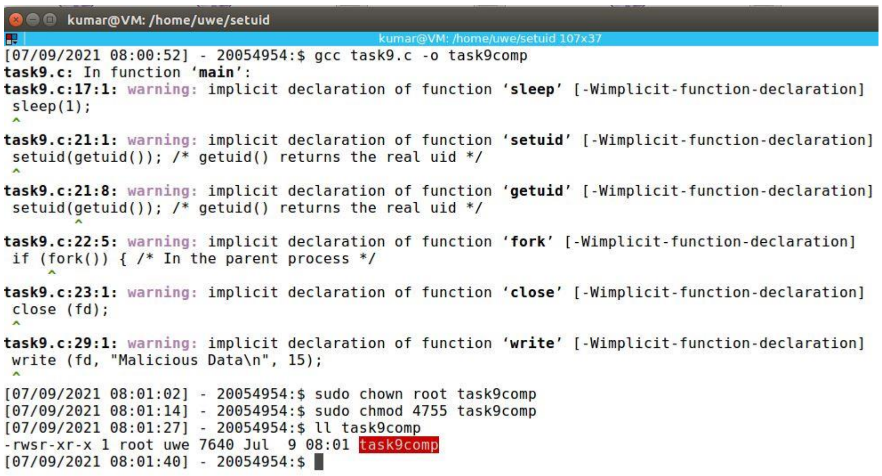


```
kumar@VM: /home/uwe/setuid
kumar@VM: /home/uwe/
[07/09/2021 07:53:46] - 20054954:$ su kumar
Password:
kumar@VM:/home/uwe/setuid$ ll exeout
-rwsr-xr-x 1 root uwe 7548 Jul  9 07:46 exeout*
kumar@VM:/home/uwe/setuid$ ./exeout "task8file;/bin/sh"
/bin/cat: 'task8file;/bin/sh': No such file or directory
kumar@VM:/home/uwe/setuid$ ./exeout task8file;/bin/sh
This file is read only to other users
$ rm task8file
rm: remove write-protected regular file 'task8file'? y
rm: cannot remove 'task8file': Permission denied
$ whoami
kumar
$
```

Figure 24 Demonstration of Set-UID program with execve() avoiding the attack

## 1.9 Task 9: Capability Leaking

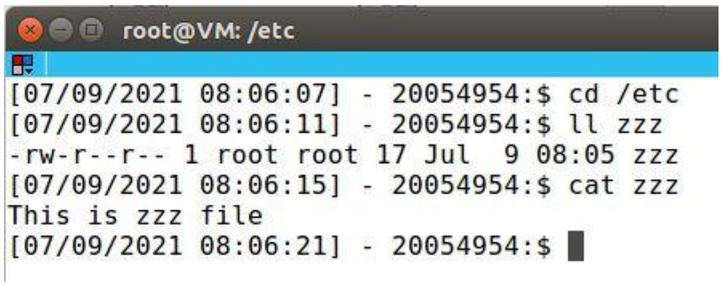
Firstly, we compile and make it a Set-UID root program as shown in Figure 25.



```
kumar@VM: /home/uwe/setuid
kumar@VM: /home/uwe/setuid 107x37
[07/09/2021 08:00:52] - 20054954:$ gcc task9.c -o task9comp
task9.c: In function ‘main’:
task9.c:17:1: warning: implicit declaration of function ‘sleep’ [-Wimplicit-function-declaration]
 sleep(1);
 ^
task9.c:21:1: warning: implicit declaration of function ‘setuid’ [-Wimplicit-function-declaration]
 setuid(getuid()); /* getuid() returns the real uid */
 ^
task9.c:21:8: warning: implicit declaration of function ‘getuid’ [-Wimplicit-function-declaration]
 setuid(getuid()); /* getuid() returns the real uid */
        ^
task9.c:22:5: warning: implicit declaration of function ‘fork’ [-Wimplicit-function-declaration]
 if (fork()) { /* In the parent process */
     ^
task9.c:23:1: warning: implicit declaration of function ‘close’ [-Wimplicit-function-declaration]
 close (fd);
 ^
task9.c:29:1: warning: implicit declaration of function ‘write’ [-Wimplicit-function-declaration]
 write (fd, "Malicious Data\n", 15);
 ^
[07/09/2021 08:01:02] - 20054954:$ sudo chown root task9comp
[07/09/2021 08:01:14] - 20054954:$ sudo chmod 4755 task9comp
[07/09/2021 08:01:27] - 20054954:$ ll task9comp
-rwsr-xr-x 1 root uwe 7640 Jul  9 08:01 task9comp
[07/09/2021 08:01:40] - 20054954:$
```

Figure 25 Making Task9 program to Set-UID program

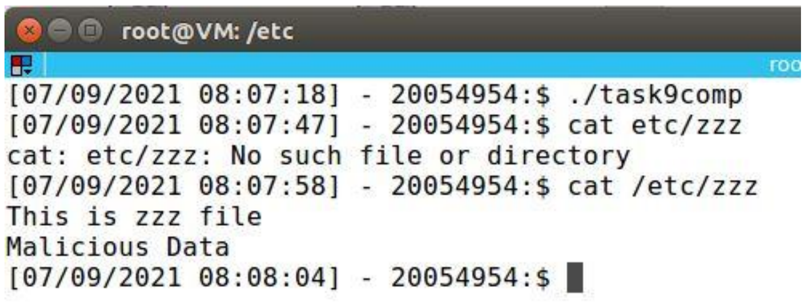
Figure 26 shows the zzz file and its contents in the /etc folder.



```
[07/09/2021 08:06:07] - 20054954:$ cd /etc
[07/09/2021 08:06:11] - 20054954:$ ll zzz
-rw-r--r-- 1 root root 17 Jul  9 08:05 zzz
[07/09/2021 08:06:15] - 20054954:$ cat zzz
This is zzz file
[07/09/2021 08:06:21] - 20054954:$ 
```

*Figure 26 Contents of zzz file*

The program was executed and the contents of the zzz file was checked. As shown in Figure 27, we can see that the contents have been modified. This is because we did not close the file at the right time and hence the file was still running with privileged permissions that allowed the data in the file to be modified, even without the right permissions. Here, after calling fork, the control is passed to the child process and hence the malicious user is successful in modifying the content of a privileged file. This shows that it is important to close the file descriptor after dropping privileges, in order for it to have the appropriate permissions.



```
[07/09/2021 08:07:18] - 20054954:$ ./task9comp
[07/09/2021 08:07:47] - 20054954:$ cat etc/zzz
cat: etc/zzz: No such file or directory
[07/09/2021 08:07:58] - 20054954:$ cat /etc/zzz
This is zzz file
Malicious Data
[07/09/2021 08:08:04] - 20054954:$ 
```

*Figure 27 Demonstration of capability leaking*

## 2 Further research and a real-world case study

a)

On all Unix systems, each process has an effective user ID, a real user ID, an effective group ID, and a real group ID. Normally when a process is executed, the effective, real, and saved user and group IDs are all set to the real user and group ID of the process's parent, respectively. However, when the setuid bit is set on an executable, the effective and saved user IDs are set to the user ID that owns the file. Likewise, when the setgid bit is set on an executable, the effective and saved group IDs are set to the group ID that owns the file (Viega and Messier, 2003).

For the most part, all privilege checks performed by the operating system are done using the effective user or effective group ID. The primary deviations from this rule are some of the system calls used to manipulate a process's user and group IDs. In general, the effective user or group ID for a process may be changed as long as the new ID is the same as either the real or the saved ID (Viega and Messier, 2003).

Taking all this into account, permanently dropping privileges involves ensuring that the effective, real, and saved IDs are all the same value. Temporarily dropping privileges requires that the effective and real IDs are the same value, and that the saved ID is unchanged so that the effective ID can later be restored to the higher privilege. These rules apply to both group and user IDs (Viega and Messier, 2003).

The first of two functions, spc_drop_privileges() drops any extra group or user privileges either permanently or temporarily, depending on the value of its only argument. If a nonzero value is passed, privileges will be dropped permanently; otherwise, the privilege drop is temporary. The second function, spc_restore_privileges() , restores privileges to what they were at the last call to spc_drop_privileges( ). If either function encounters any problems in attempting to perform its respective task, abort() is called, terminating the process immediately. If any manipulation of privileges cannot compete successfully, it's safest to assume that the process is in an unknown state, and you should not allow it to continue. spc_drop_privileges( ) is littered with preprocessor conditionals that test for the platform on which the code is being compiled. Dropping privileges involves a simple call to setegid( ) or seteuid( ), followed by a call to either setgid( ) or setuid( ) if privileges are being permanently dropped (Viega and Messier, 2003).

Thus, dash shell uses these functions to immediately change its effective UID to processor's UID, if it detects that it has been executed in a Set-UID process.

b)

## INTRODUCTION

Set-UID is an important mechanism which provides privilege escalation to the users. The privilege programs provide special permissions to the users which are not assigned to them. Set-UID is one of the privilege programs which provides users with root privilege while executing the program. The resources hosted by the web server are accessed by the users present in remote locations, this is one of the examples for privilege programs (Computer Hope, 2020). The Set-UID programs uses the concept of EUID (Effective User ID) and RUID (Real User ID). For a Set-UID program, the EUID is the ID of the owner of the program and RUID is the ID of the user of the program. There are a lot of vulnerabilities associated with Set-UID namely, hidden inputs through environment variable, invoking malicious programs, etc. This report will elaborate on two real-world cases of Set-UID program vulnerability ('SetUID, SetGID, and Sticky Bits in Linux File Permissions', 2019).

## JUNOS OS: MULTIPLE LOCAL PRIVILEGE ESCALATION VULNERABILITIES IN SUID BINARIES

Juniper Networks, Inc. is an American multinational corporation that develops and markets various networking products and software. Junos OS is one among them, it is a single operating system which powers the Juniper's networking and security products. Recently, a vulnerability has been found in Junos OS, where an authenticated user who has shell access can use local privilege escalation vulnerability in ethtraceroute to write to local files as root user. This is because the ethtraceroute which is owned by root user, has set-uid permissions, which allowed local users to run ethtraceroute with root privileges. This affected Junos OS versions from 17.3 to 20.4 (CVE-2021-0255). Similar to this vulnerability, the mosquito message broker had sensitive information disclosure vulnerability which allowed locally authenticated users to read sensitive files, which is master.passwd file. This is because mosquitto, which is owned by root user, allowed local user to run with root privileges. These issues were identified during external security research (CVE-2021-0256). The Juniper Networks's Security Incident Response Team (SIRT) stated that hasn't been any malicious exploitation using these vulnerabilities. According to Common Vulnerability Scoring System (CVSS) and Juniper's Security Advisories, this vulnerability has been stated with medium level of severity with 5.5 as CVSS score. To resolve these issues, software updates has been provided. To reduce the risk of malicious exploitation, we need to use access lists or firewall filters to limit CLI access to the device only from trusted, administrative networks or hosts. Additionally, we need to limit access to the Junos OS shell to only trusted system administrators (Juniper Networks, 2021).

## CONCLUSION

The Set-UID based vulnerabilities poses a security threat to organisations and the end users. As in the case of Junos OS, this vulnerability could have caused a malicious exploitation with the attacker accessing sensitive files which include the password file of the root user. To avoid these vulnerabilities, steps must be taken to frequently provide patching and software updates if any vulnerability has been found.

## 3 References

Computer Hope (2020) ‘Setuid’. Available at: https://www.computerhope.com/jargon/s/setuid.htm.

Juniper Networks (2021) ‘2021-04 Security Bulletin: Junos OS: Multiple Local Privilege Escalation vulnerabilities in SUID binaries’. Available at: https://kb.juniper.net/InfoCenter/index?page=content&id=JSA11175&actp=METADATA.

‘SetUID, SetGID, and Sticky Bits in Linux File Permissions’ (2019). Available at: https://www.geeksforgeeks.org/setuid-setgid-and-sticky-bits-in-linux-file-permissions/.

Viega, J. and Messier, M. (2003) ‘Secure Programming Cookbook for C and C++’. Available at: https://www.oreilly.com/library/view/secure-programming-cookbook/0596003943/ch01s03.html.