Software Security:
Environment variable and Set-UID Program Lab

Batyi Amatov (amatobat@b-tu.de)

Task 1: Manipulating Environment Variables

```
root@vm:/etc# printenv PWD
/etc
root@vm:/etc# env | grep PWD
PWD=/etc
OLDPWD=/home/csl
root@vm:/etc# printenv
LC_PAPER=de_DE.UTF-8
XDG_VTNR=7
XDG_SESSION_ID=c2
LC_ADDRESS=de_DE.UTF-8
CLUTTER_IM_MODULE=xim
LC_MONETARY=de_DE.UTF-8
XDG_GREETER_DATA_DIR=/var/lib/lightdm-data/csl
SESSION=ubuntu
GPG_AGENT_INFO=/home/csl/.gnupg/S.gpg-agent:0:1
ANDROID_HOME=/home/csl/android/android-sdk-linux
SHELL=/bin/bash
XDG_MENU_PREFIX=gnome-
VTE_VERSION=4205
TERM=xterm-256color
DERBY_HOME=/usr/lib/jvm/java-8-oracle/db
QT_LINUX_ACCESSIBILITY_ALWAYS_ON=1
LD_PRELOAD=/home/csl/lib/boost/libboost_program_options.so.1.64.0:/home/csl/lib/boost/libboost_filesystem.so.1.64.0:/home/csl/lib/
boost/libboost_system.so.1.64.0
LC_NUMERIC=de_DE.UTF-8
WINDOWID=73400330
GNOME_KEYRING_CONTROL=
UPSTART_SESSION=unix:abstract=/com/ubuntu/upstart-session/1000/1645
GTK_MODULES=gail:atk-bridge:unity-gtk-module
USER=root
LD_LIBRARY_PATH=/home/seed/source/boost_1_64_0/stage/lib:/home/seed/source/boost_1_64_0/stage/lib:
LC_TELEPHONE=de_DE.UTF-8
QT_ACCESSIBILITY=1
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=3
0;41:tw=30;42:ow=34;42:st=37;44:ex=01;32:*.tar=01;31:*.tgz=01;31:*.arc=01;31:*.arj=01;31:*.taz=01;31:*.lha=01;31:*.lz4=01;31:*.lzh
=01;31:*.lzma=01;31:*.tlz=01;31:*.txz=01;31:*.tzo=01;31:*.t7z=01;31:*.zip=01;31:*.z=01;31:*.Z=01;31:*.dz=01;31:*.gz=01;31:*.lrz=01
;31:*.lz=01;31:*.lzo=01;31:*.xz=01;31:*.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;35:*.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=01;35:*.mov=01;35:*.mpg=01;35:*.mpeg=01;35:*.m2v=01;35:
*.mkv=01;35:*.webm=01;35:*.ogm=01;35:*.mp4=01;35:*.m4v=01;35:*.mp4v=01;35:*.vob=01;35:*.qt=01;35:*.nuv=01;35:*.wmv=01;35:*.asf=01;
35:*.rm=01;35:*.rmvb=01;35:*.flc=01;35:*.avi=01;35:*.fli=01;35:*.flv=01;35:*.gl=01;35:*.dl=01;35:*.xcf=01;35:*.xwd=01;35:*.yuv=01;
35:*.cgm=01;35:*.emf=01;35:*.ogv=01;35:*.ogx=01;35:*.aac=00;36:*.au=00;36:*.flac=00;36:*.m4a=00;36:*.mid=00;36:*.midi=00;36:*.mka=
00;36:*.mp3=00;36:*.mpc=00;36:*.ogg=00;36:*.ra=00;36:*.wav=00;36:*.oga=00;36:*.opus=00;36:*.spx=00;36:*.xspf=00;36:
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
SESSION_MANAGER=local/vm:@/tmp/.ICE-unix/1920,unix/vm:/tmp/.ICE-unix/1920
XDG_CONFIG_DIRS=/etc/xdg/xdg-ubuntu:/usr/share/upstart/xdg:/etc/xdg
MAIL=/var/mail/root
PATH=/home/seed/bin:/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin:/usr/games:/usr/local/games:.:/home/csl/android/a
ndroid-sdk-linux/tools:/home/csl/android/android-sdk-linux/platform-tools:/home/csl/android/android-ndk/android-ndk-r8d
DESKTOP_SESSION=ubuntu
QT_QPA_PLATFORMTHEME=appmenu-qt5
QT_IM_MODULE=ibus
LC_IDENTIFICATION=de_DE.UTF-8
JOB=dbus
PWD=/etc
XDG_SESSION_TYPE=x11
JAVA_HOME=/usr/lib/jvm/java-8-oracle
XMODIFIERS=@im=ibus
GNOME_KEYRING_PID=
LANG=en_US.UTF-8
MANDATORY_PATH=/usr/share/gconf/ubuntu.mandatory.path
GDM_LANG=en_US
```

Figure 1

printenv and *env* are used to print out the environment variables. Use *grep* in order to print particular environmental variable.

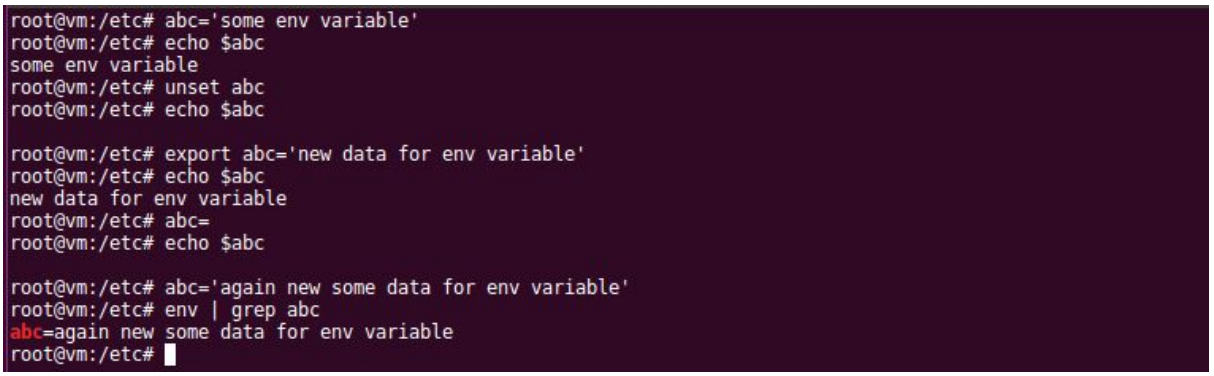

Figure 2

Using *export* and *unset* we can set or unset environment variables. It should be noted that these two commands are two of the Bash's internal commands.

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

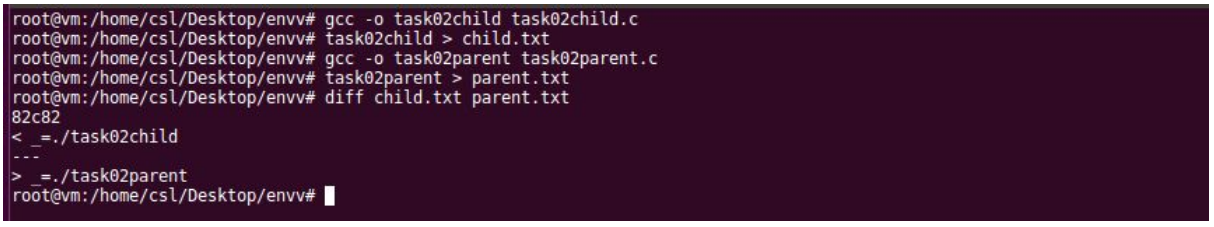

Figure 3

Almost all environmental variables are the same except one $_ which shows the last argument of the shell. So, we can conclude that child process inherits all environmental variables from the parent process.

Task 3: Passing Environment Variables and execve()

```
root@vm:/home/csl/Desktop/envv# gcc -o task3 task3.c
root@vm:/home/csl/Desktop/envv# ./task3
root@vm:/home/csl/Desktop/envv# cat task031.c
#include <stdio.h>
#include <unistd.h>

extern char **environ;

int main() {
        char *argv[2];
        argv[0] = "/usr/bin/env";
        argv[1] = NULL;
        execve("/usr/bin/env", argv, environ);
        return 0;
}root@vm:/home/csl/Desktop/envv# gcc -o task031 task031.c
root@vm:/home/csl/Desktop/envv# ./task031
LC_PAPER=de_DE.UTF-8
XDG_VTNR=7
XDG_SESSION_ID=c2
LC_ADDRESS=de_DE.UTF-8
CLUTTER_IM_MODULE=xim
LC_MONETARY=de_DE.UTF-8
XDG_GREETER_DATA_DIR=/var/lib/lightdm-data/csl
SESSION=ubuntu
GPG_AGENT_INFO=/home/csl/.gnupg/S.gpg-agent:0:1
ANDROID_HOME=/home/csl/android/android-sdk-linux
SHELL=/bin/bash
XDG_MENU_PREFIX=gnome-
VTE_VERSION=4205
TERM=xterm-256color
DERBY_HOME=/usr/lib/jvm/java-8-oracle/db
QT_LINUX_ACCESSIBILITY_ALWAYS_ON=1
LD_PRELOAD=/home/csl/lib/boost/libboost_program_options.so.1.64.0:/home/csl/lib/boost/libboost_filesystem.so.1.64.0:/home/csl/lib/
boost/libboost_system.so.1.64.0
LC_NUMERIC=de_DE.UTF-8
WINDOWID=73400330
OLDPWD=/home/csl/Desktop
GNOME_KEYRING_CONTROL=
UPSTART_SESSION=unix:abstract=/com/ubuntu/upstart-session/1000/1645
GTK_MODULES=gail:atk-bridge:unity-gtk-module
USER=root
LD_LIBRARY_PATH=/home/seed/source/boost_1_64_0/stage/lib:/home/seed/source/boost_1_64_0/stage/lib:
LC_TELEPHONE=de_DE.UTF-8
QT_ACCESSIBILITY=1
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=3
0;41:tw=30;42:ow=34;42:st=37;44:ex=01;32:*.tar=01;31:*.tgz=01;31:*.arc=01;31:*.arj=01;31:*.taz=01;31:*.lha=01;31:*.lz4=01;31:*.lzh
=01;31:*.lzma=01;31:*.tlz=01;31:*.txz=01;31:*.tzo=01;31:*.t7z=01;31:*.zip=01;31:*.z=01;31:*.Z=01;31:*.dz=01;31:*.gz=01;31:*.lrz=01
;31:*.lz=01;31:*.lzo=01;31:*.xz=01;31:*.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;35:*.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=01;35:*.mov=01;35:*.mpg=01;35:*.mpeg=01;35:*.m2v=01;35:
*.mkv=01;35:*.webm=01;35:*.ogm=01;35:*.mp4=01;35:*.m4v=01;35:*.mp4v=01;35:*.vob=01;35:*.qt=01;35:*.nuv=01;35:*.wmv=01;35:*.asf=01;
35:*.rm=01;35:*.rmvb=01;35:*.flc=01;35:*.avi=01;35:*.fli=01;35:*.flv=01;35:*.gl=01;35:*.dl=01;35:*.xcf=01;35:*.xwd=01;35:*.yuv=01;
35:*.cgm=01;35:*.emf=01;35:*.ogv=01;35:*.ogx=01;35:*.aac=00;36:*.au=00;36:*.flac=00;36:*.m4a=00;36:*.mid=00;36:*.midi=00;36:*.mka=
00;36:*.mp3=00;36:*.mpc=00;36:*.ogg=00;36:*.ra=00;36:*.wav=00;36:*.oga=00;36:*.opus=00;36:*.spx=00;36:*.xspf=00;36:
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
SESSION_MANAGER=local/vm:@/tmp/.ICE-unix/1920,unix/vm:/tmp/.ICE-unix/1920
XDG_CONFIG_DIRS=/etc/xdg/xdg-ubuntu:/usr/share/upstart/xdg:/etc/xdg
```

Figure 4

We are using *execve()* function which takes three arguments: the command to input, arguments for the command, the env variables passed to the new program.
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.

Step 1: we passed NULL as a third parameter for function *execve()* and ofcourse we was not able to print any environmental variables.
Step 2: we can passed global variable environ which points to the beginning of environmental variables, and we managed to print them all.

Task 4: Environment Variables and system()

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. Environmental variables are passing to the shell, and shell execute the command with environmental variables as arguments.

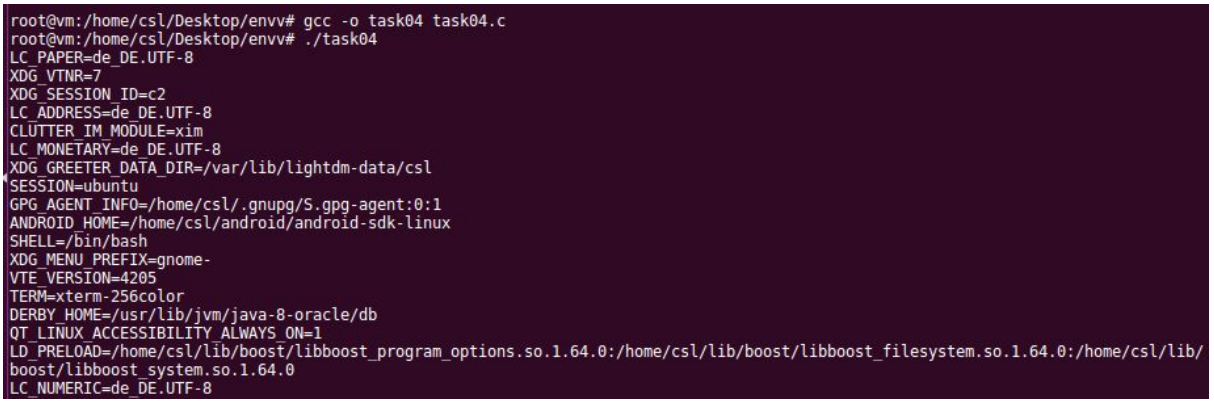

Figure 5

As a result environmental variables are printed out.

Task 5: Environment Variables and Set-UID Programs

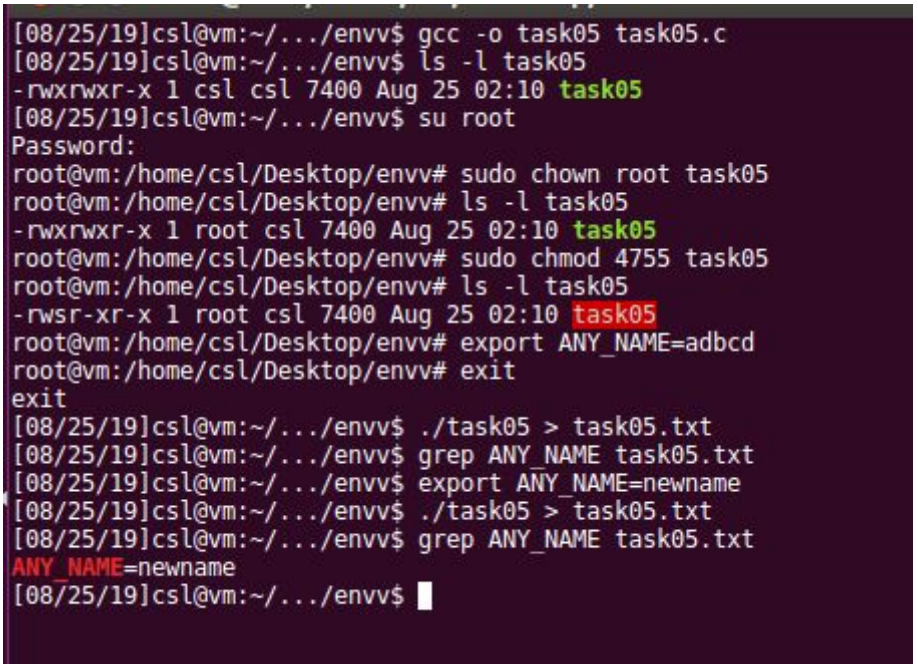

Figure 6

We made a program owner - root, and made it Set_UID program. After the execution , the shell forks a child process, and uses the child process to run the program. These environment variables which were in the user's shell process, passed to the Set-UID child process.

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

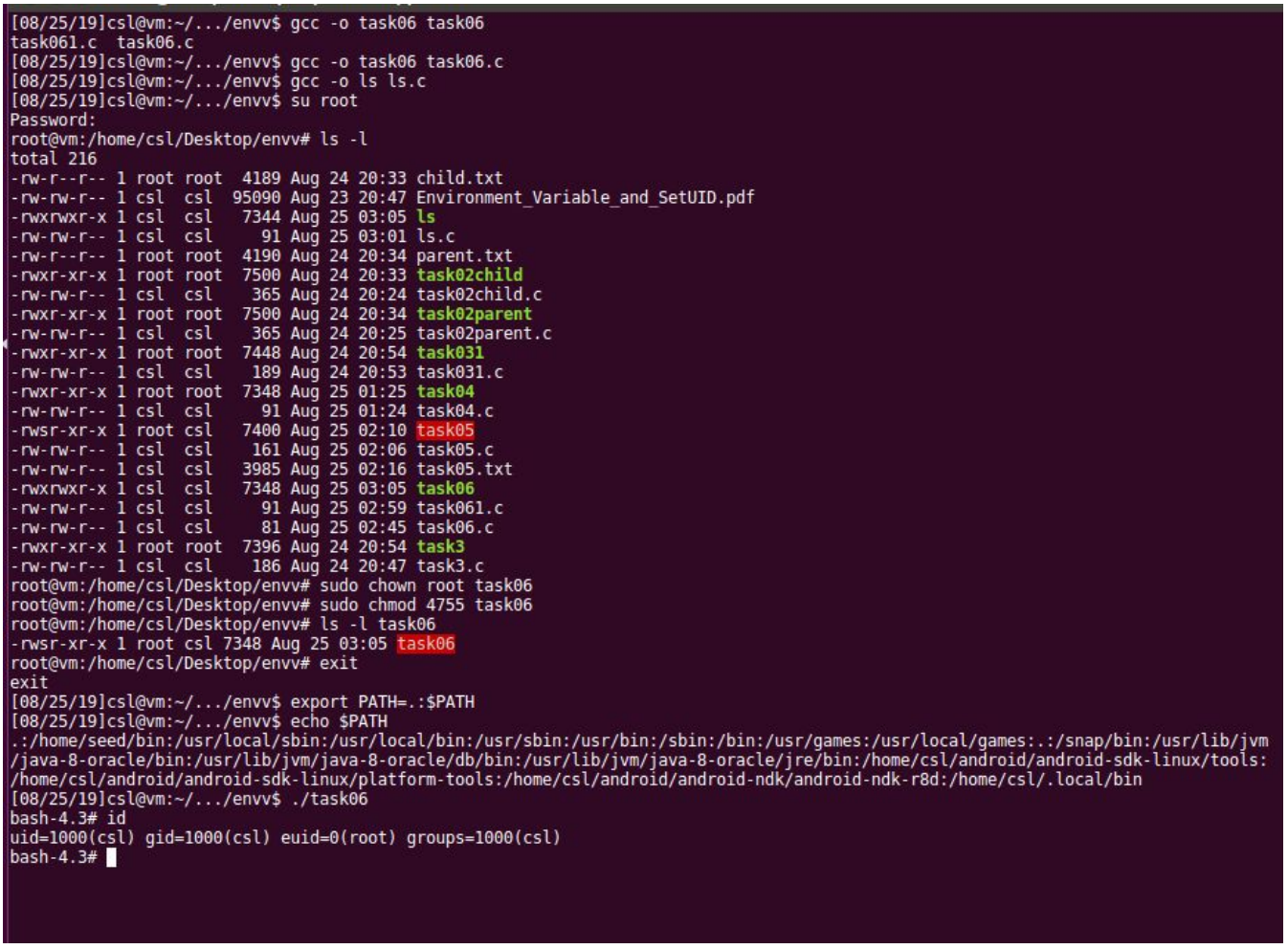

Figure 7

If we are using Set-UID vulnerable program we can get access to data or privileges which we are not allowed to have. For example, if system() function uses not a full length, it possible to change a path to a malicious file. In our case, we are invoking ls command, we can create ls executable file which itself invokes bash root shell. Since, our vulnerable program is a Set-UID program we are able to get a root shell.

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

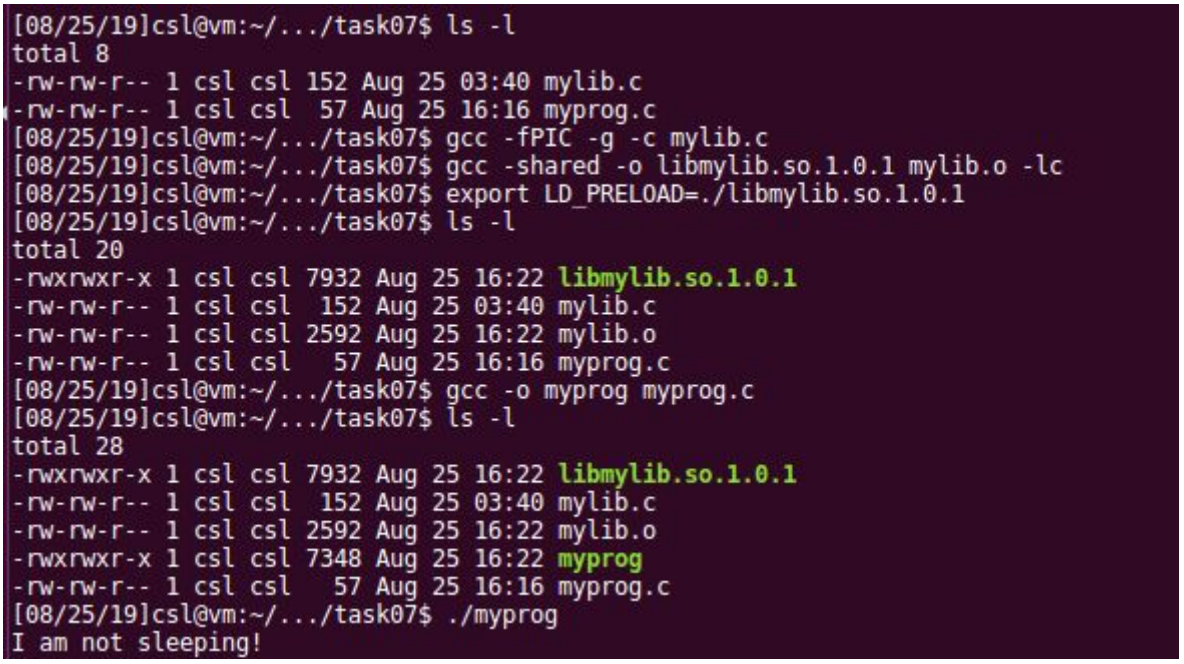

Figure 8

In Linux, ld.so or ld-linux.so, are the dynamic loader/linker. Among the environment variables that affect their behaviors, LD PRELOAD is the one that we are concerned in this lab. LD PRELOAD specifies a list of additional, user-specified, shared libraries to be loaded before all others. We created our malicious program and name it mylib.c. It basically overrides the sleep() function in libc library.
We compiled it and exported a new path in LD_PRELOAD environmental variable which started to point to our malicious program. We running without Set-UID privilege and we able to execute our malicious program.

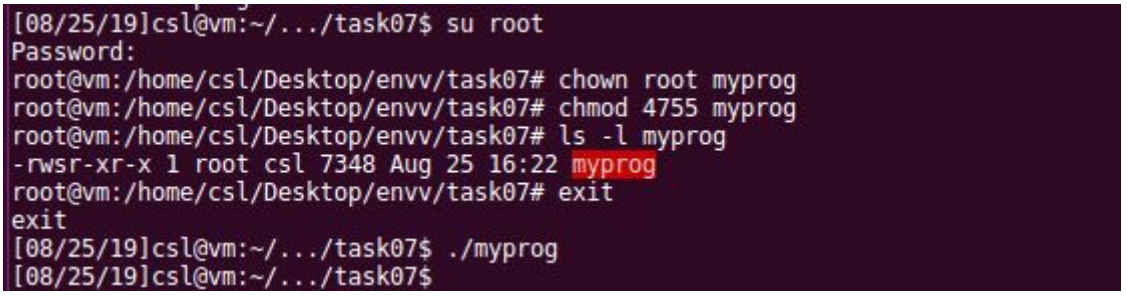

Figure 9

After making myprog a Set-UID root program, and run it as a normal user we were not able to run our malicious program. Instead a correct function sleep() from trustfull library was executed. This is due to countermeasure implemented by the dynamic linker, which ignores LD_PRELOAD env variable when the process's real and effective user or group IDs differ.

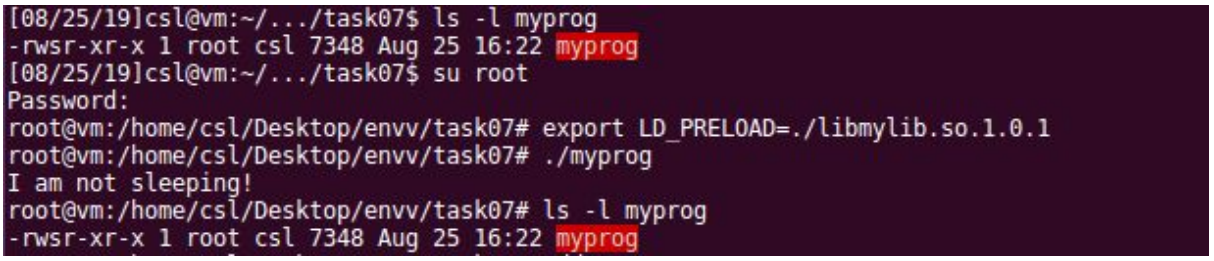

Figure 10

After logging in root, we were able to run our malicious program, because child process inherits path from LD_PRELOAD and countermeasure of dynamic linking does not work, because our owner is the root and after execution effective UID also becomes root.

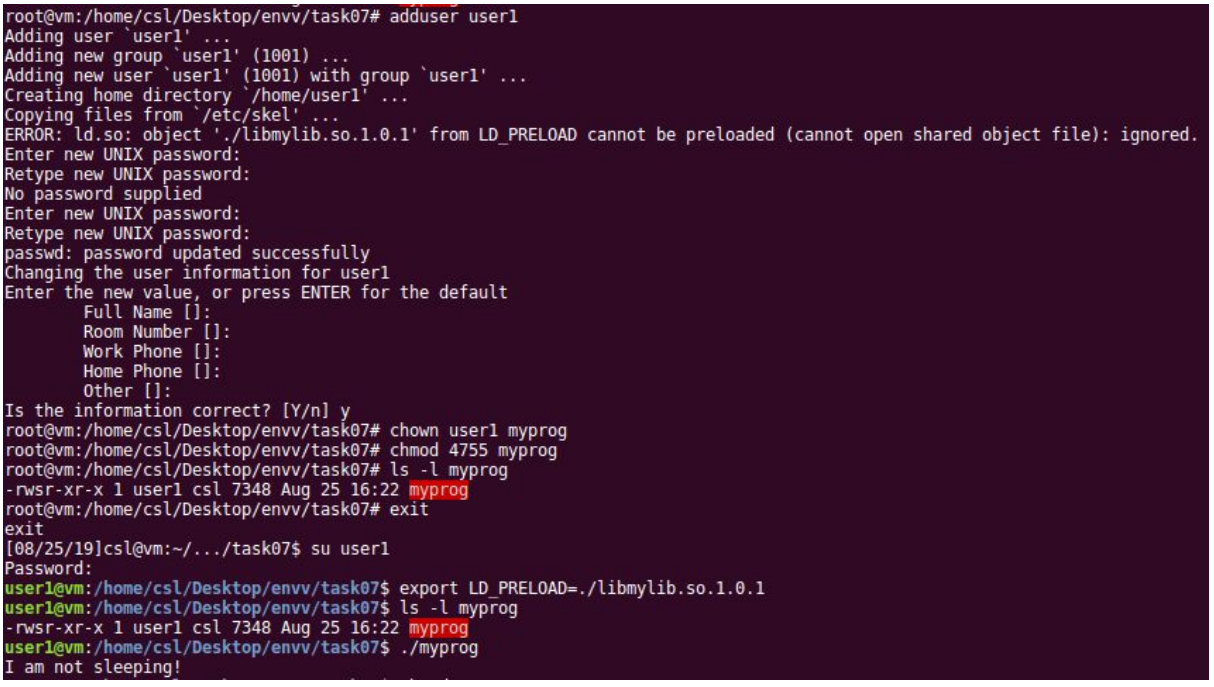

Figure 11

After creating a new user1 account, we were managed to execute our malicious program, because it has Set-UID privilege and real and effective user IDs are the same.

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

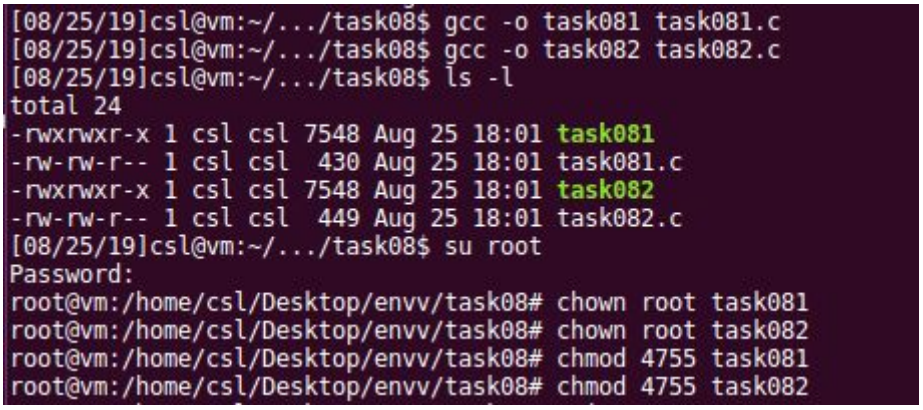

Figure 12

We prepared our executable files with root owner and Set-UID priviledge.

Execve() function is safer than *system()*, because it is directly and explicitly pass command string as an input to system call. In comparison, *system()* function execute it using a shell, and it is possible to inject another command using semicolon as separator.

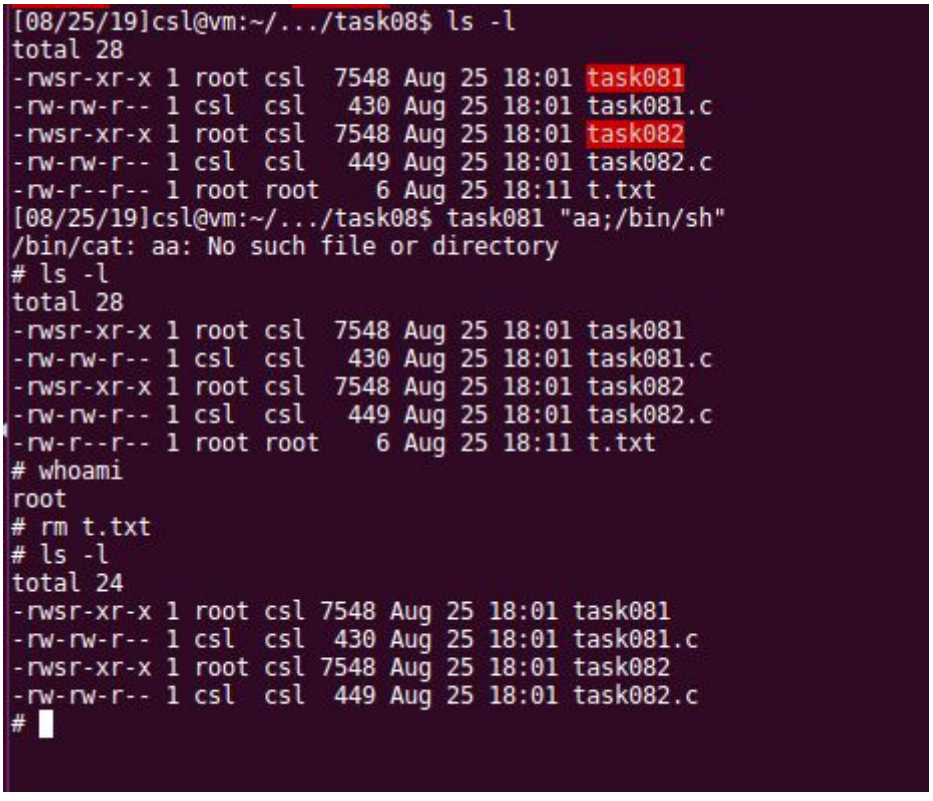

Figure 13

That's why we were managed to inject the system() function and invoke a root shell. With root shell we can do almost everything.

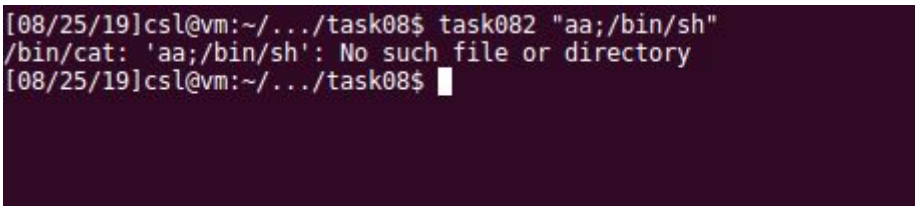

Figure 14

Execve() function did not permitted poisoning because it does have a man-in-the-middle shell, which executes a command. Instead it directly takes input as one atomic string and perform system call.

Task 9: Capability Leaking

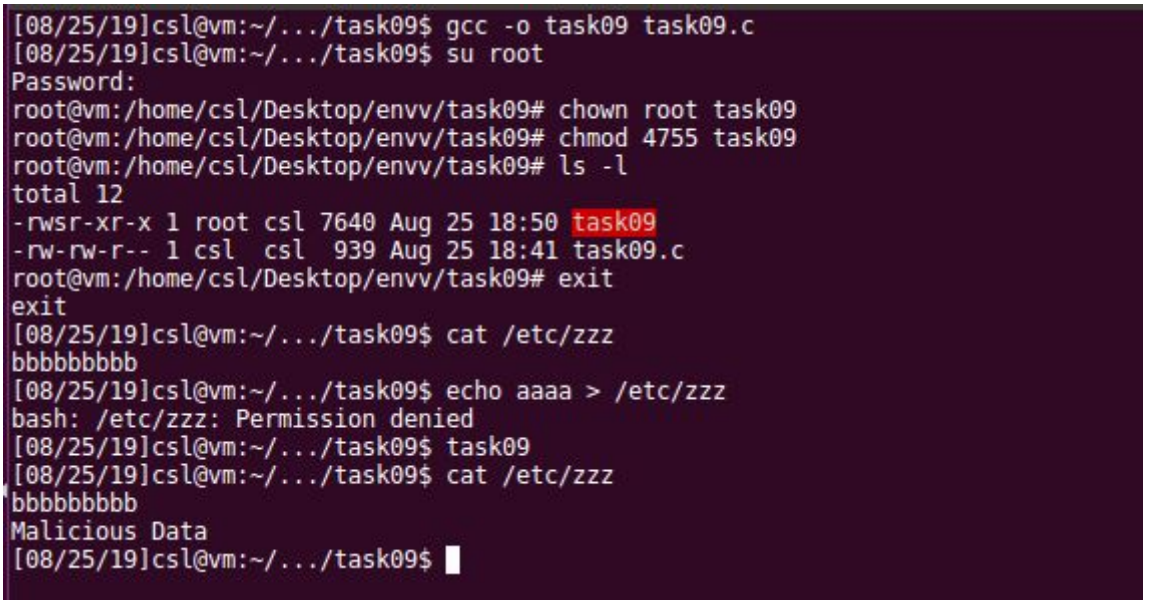

Figure 15

To revoke root privileges the setuid() system call can be used. According to the manual, “setuid() sets the effective user ID of the calling process. If the effective UID of the caller is root, the real UID and saved set-user-ID are also set”. Therefore, if a Set-UID program with effective UID 0 calls setuid(n), the process will become a normal process, with all its UIDs being set to n.

When revoking the privilege, one of the common mistakes is capability leaking. In our case program does not clean up a fd (file descriptor) which is a form of capability, which makes the file zzz still accessible by the non-privileged process.