Environment Variable and SetUID Program Lab

61517304 刘余文

September 4, 2020

1 Task 1: Manipulating Environment Variables

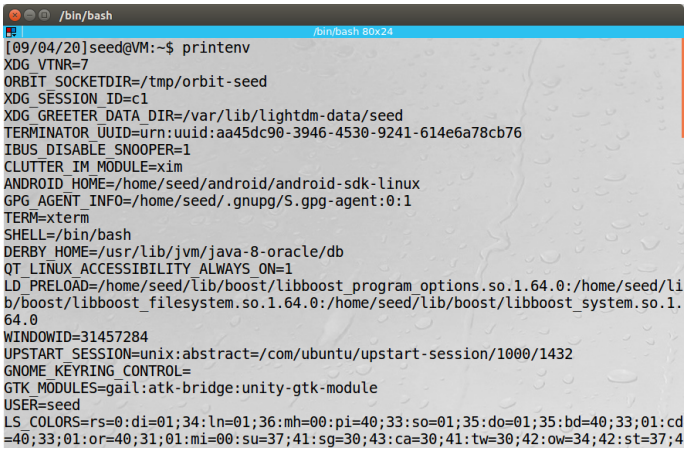
1.1 Requirements

1. Use `printenv` or `env` command to print out the environment variable.

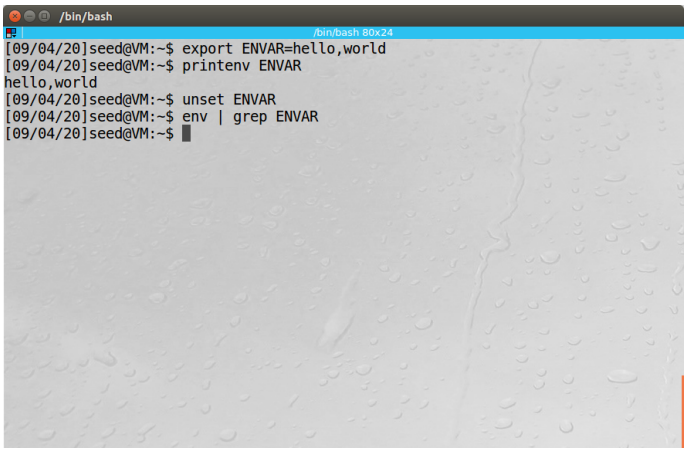
2. Use `export` and `unset` to set or unset environment variables

1.2 Steps and results

1. To req1, just type `printenv` or `env` is ok. Or type `printenv xxx` or `env | grep xxx` to get specific environment variable.



2. To req2, type `export ENVAR=value` to set environment variable and `unset ENVAR` to unset that. Note that when setting environment variable with `export`, value is a must, otherwise it will not be recorded in environment variable list.



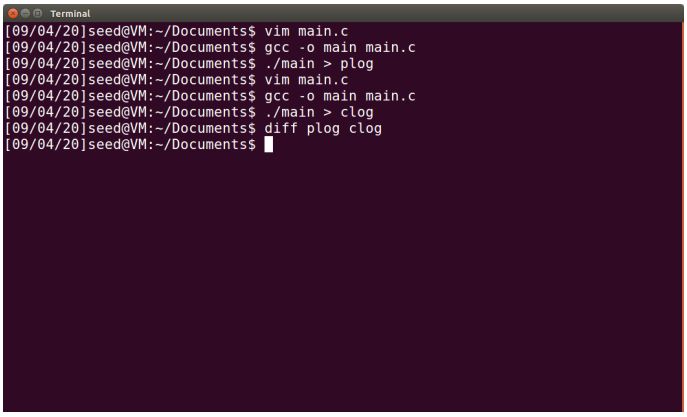
2 Task 2: Passing Environment Variables from Parent Process to Child Process

2.1 Requirements

1. Compare the environment variables between parent and child process, find out the difference and draw the conclusion.

2.2 Steps and result

1. Comment child process output function and record parent process's outputs into plog
2. Comment parent process output function and record child process's outputs into clog
3. Use command `diff` to find the differece between plog and clog. Result is that there are no difference between the two process.



2.3 Conclusion

From the result, we can assume that the child process inherit the environment variables from parent process.

3 Task 3: Environment Variables and `execve()`

3.1 Requirements

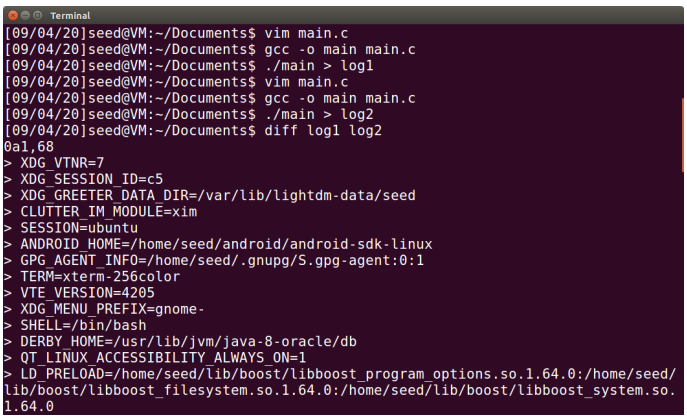
1. Find out how function `execve()` get environment variables and draw the conclusion.

3.2 Steps and result

1. Run the provided code and set the third parament as `NULL`, record the outputs as log1

2. Run the provided code and set the third parament as `environ`, record the outputs as log2

3. Compare the two outputs. Result is that there are environment variables in log2, while log1 is empty.



```
Terminal
[09/04/20]seed@VM:~/Documents$ vim main.c
[09/04/20]seed@VM:~/Documents$ gcc -o main main.c
[09/04/20]seed@VM:~/Documents$ ./main > log1
[09/04/20]seed@VM:~/Documents$ vim main.c
[09/04/20]seed@VM:~/Documents$ gcc -o main main.c
[09/04/20]seed@VM:~/Documents$ ./main > log2
[09/04/20]seed@VM:~/Documents$ diff log1 log2
0a1,68
> XDG_VTNR=7
> XDG_SESSION_ID=c5
> XDG_GREETER_DATA_DIR=/var/lib/lightdm-data/seed
> CLUTTER_IM_MODULE=xim
> SESSION=ubuntu
> ANDROID_HOME=/home/seed/android/android-sdk-linux
> GPG_AGENT_INFO=/home/seed/.gnupg/S.gpg-agent:0:1
> TERM=xterm-256color
> VTE_VERSION=4205
> XDG_MENU_PREFIX=gnome-
> 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
```

3.3 Conclusion

From the result, we can assume that function `execve()` accept and set environment variables by the third parament. And it it true when referring to `man execve`.

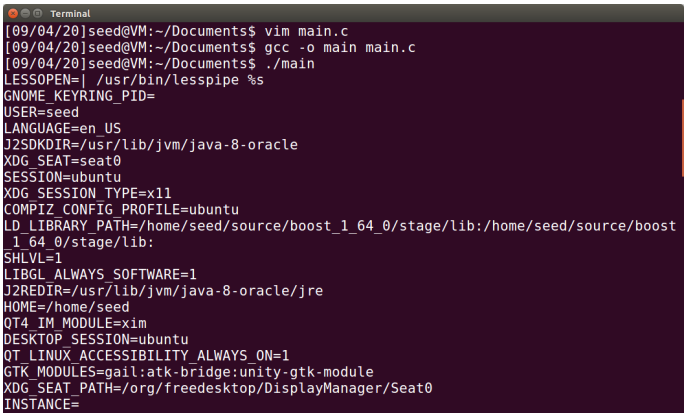
4 Task 4: Environment Variables and `system()`

4.1 Requirements

1. Study how environment variables are affected when a new program is executed via the `system()` function.

4.2 Steps and result

1. Run the provided code, record the outputs.



```
Terminal
[09/04/20]seed@VM:~/Documents$ vim main.c
[09/04/20]seed@VM:~/Documents$ gcc -o main main.c
[09/04/20]seed@VM:~/Documents$ ./main
LESSOPEN=| /usr/bin/lesspipe %s
GNOME_KEYRING_PID=
USER=seed
LANGUAGE=en_US
J2SDKDIR=/usr/lib/jvm/java-8-oracle
XDG_SEAT=seat0
SESSION=ubuntu
XDG_SESSION_TYPE=x11
COMPIZ_CONFIG_PROFILE=ubuntu
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
QT_LINUX_ACCESSIBILITY_ALWAYS_ON=1
GTK_MODULES=gail:atk-bridge:unity-gtk-module
XDG_SEAT_PATH=/org/freedesktop/DisplayManager/Seat0
INSTANCE=
```

4.3 Conclusion

From the result, we can assume that function `system()` passes environment variables of the old process to the bash it invokes.

4.4 Some comments

I think the example in Lab Manual is not so good to draw the conclusion. The example should use `execve()` and pass some specific environment variables, then invoke `system()`, it will be clearer to understand that.

5 Task 5: Environment Variable and `Set-UID` Programs

5.1 Requirements

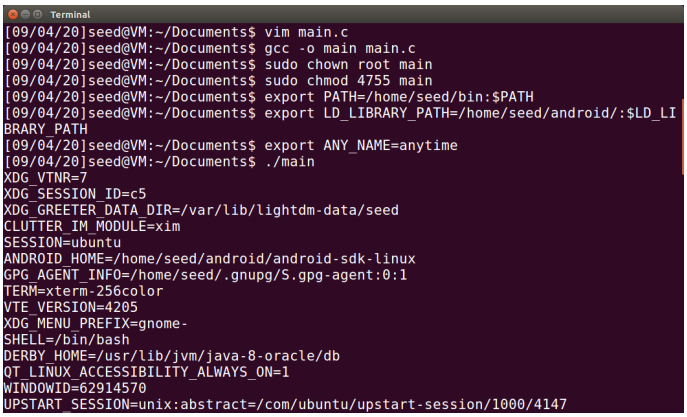
1. Study figure out whether environment variables are inherited by the `Set-UID` program' s process from the user' s process.

5.2 Steps and result

1. Run the provided code.

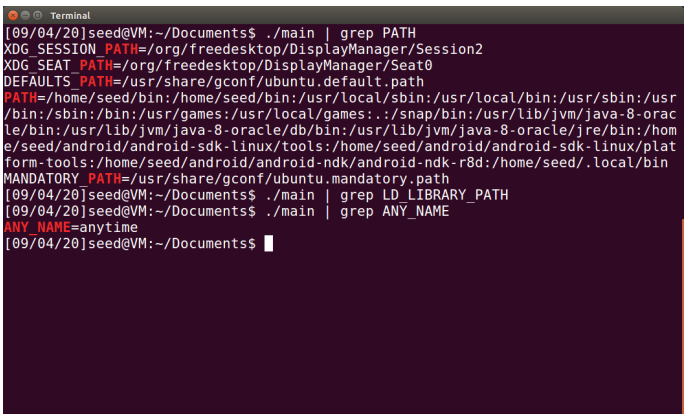
2. Use `chown` to change owner and `chmod` to set as Set-UID program

3. Set the environment variables in user' s shell, then run the program and observe the output about environment variables



```
Terminal
[09/04/20]seed@VM:~/Documents$ vim main.c
[09/04/20]seed@VM:~/Documents$ gcc -o main main.c
[09/04/20]seed@VM:~/Documents$ sudo chown root main
[09/04/20]seed@VM:~/Documents$ sudo chmod 4755 main
[09/04/20]seed@VM:~/Documents$ export PATH=/home/seed/bin:$PATH
[09/04/20]seed@VM:~/Documents$ export LD_LIBRARY_PATH=/home/seed/android/:$LD_LI
BRARY_PATH
[09/04/20]seed@VM:~/Documents$ export ANY_NAME=anytime
[09/04/20]seed@VM:~/Documents$ ./main
XDG_VTNR=7
XDG_SESSION_ID=c5
XDG_GREETER_DATA_DIR=/var/lib/lightdm-data/seed
CLUTTER_IM_MODULE=xim
SESSION=ubuntu
ANDROID_HOME=/home/seed/android/android-sdk-linux
GPG_AGENT_INFO=/home/seed/.gnupg/S.gpg-agent:0:1
TERM=xterm-256color
VTE_VERSION=4205
XDG_MENU_PREFIX=gnome-
SHELL=/bin/bash
DERBY_HOME=/usr/lib/jvm/java-8-oracle/db
QT_LINUX_ACCESSIBILITY_ALWAYS_ON=1
WINDOWID=62914570
UPSTART_SESSION=unix:abstract=/com/ubuntu/upstart-session/1000/4147
```

4. Result is that



```
Terminal
[09/04/20]seed@VM:~/Documents$ ./main | grep PATH
XDG_SESSION_PATH=/org/freedesktop/DisplayManager/Session2
XDG_SEAT_PATH=/org/freedesktop/DisplayManager/Seat0
DEFAULTS_PATH=/usr/share/gconf/ubuntu.default.path
PATH=/home/seed/bin:/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-orac
le/bin:/usr/lib/jvm/java-8-oracle/db/bin:/usr/lib/jvm/java-8-oracle/jre/bin:/hom
e/seed/android/android-sdk-linux/tools:/home/seed/android/android-sdk-linux/plat
form-tools:/home/seed/android/android-ndk/android-ndk-r8d:/home/seed/.local/bin
MANDATORY_PATH=/usr/share/gconf/ubuntu.mandatory.path
[09/04/20]seed@VM:~/Documents$ ./main | grep LD_LIBRARY_PATH
[09/04/20]seed@VM:~/Documents$ ./main | grep ANY_NAME
ANY_NAME=anytime
[09/04/20]seed@VM:~/Documents$
```

5.3 Conclusion

From the result, we can conclude that to `PATH` and `ANY_NAME`, the environment variables can be changed but `LD_LIBRARY_PATH` cannot.

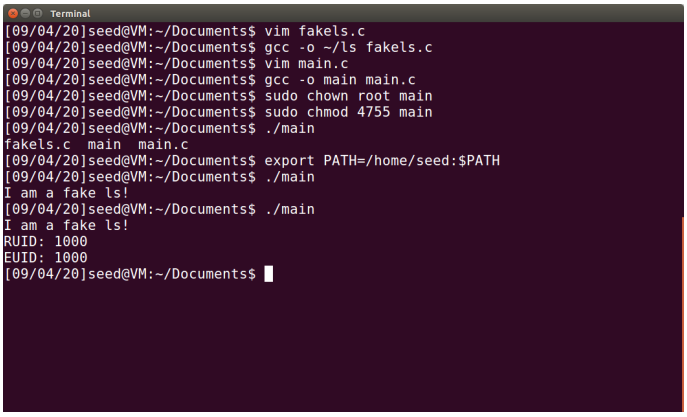
6 Task 6: The PATH Environment Variable and Set-UID Programs

6.1 Requirements

1. Find out how the PATH affect a Set-UID program

6.2 Steps and result

1. Add a path in the front of the PATH
2. Produce my own 'ls' program at the above-mentioned path.
3. Run the provided code and observe the result.



6.3 Conclusion

From the result, we can assume that PATH can mislead /bin/sh to incorrect path, but in the ls program, you cannot run as root, which is a protection.

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

7.1 Requirements

1. Study how Set-UID programs deal with some of the environment variables.

7.2 Steps and result

1. Make a DLL file and export the environment variable, then write a program invoke the DLL.

2. Run the program (myprog) in different condition and observe the result, here are conditions:

   (a) Make myproga regular program, and run it as a normal user.

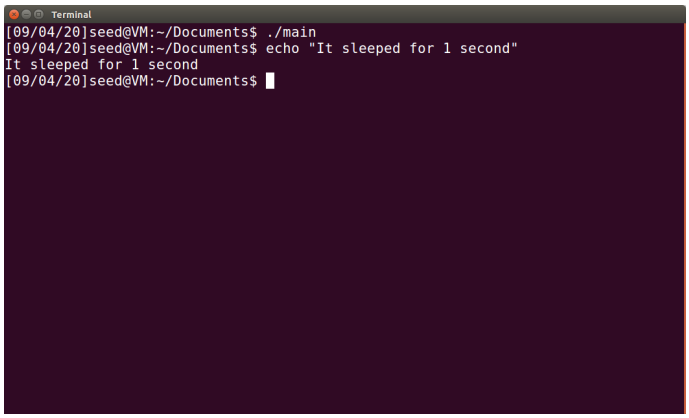
   (b) Make myproga root program, and run it as a normal user.

   (c) Make myproga root program, export the LDPRELOAD environment variable again inthe root account and run it.

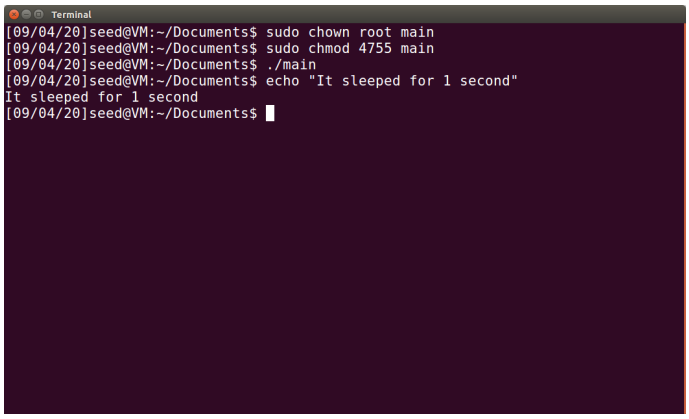
   (d) Make myproga Set-UID user1 program, export the LD_PRELOAD environment variable again in a different user' s account (not-root user) andrun it.

3. Results are that:

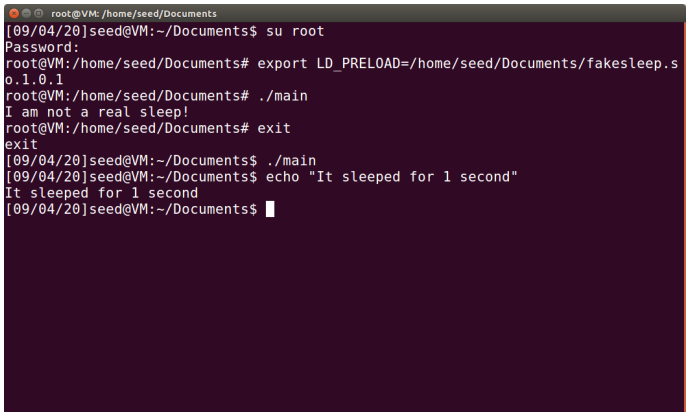
   (a) Nothing happens



(b) Nothing happens



(c) Lead to incorrect ls file



7.3 Conclusion

From the result, we can assume that if `euid` not equals to `ruid`, program will not take the path.

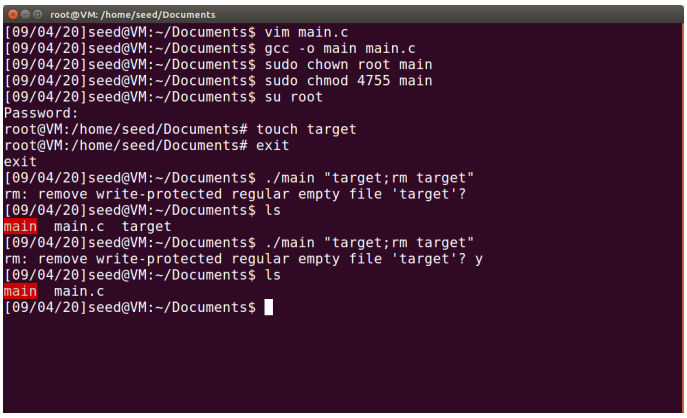
8 Task 8: Invoking External Programs Using `system()` versus `execve()`

8.1 Requirements

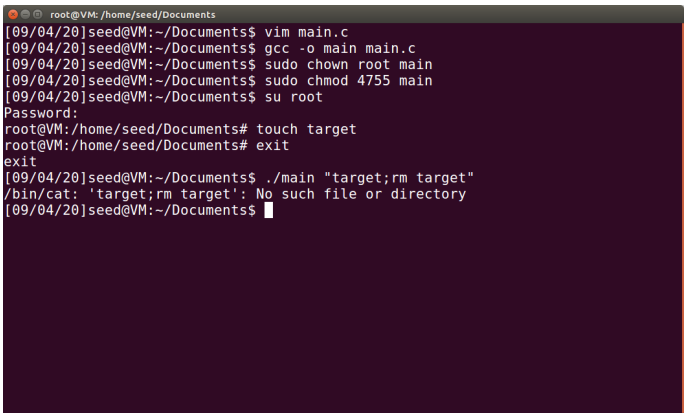
1. Study how to use `system()` or `execve()` to do something 'intersecting'.

8.2 Steps and result

1. Run the provided program and try to change input to remove an unprivileged file with `system()`



2. Run the provided program and try to change input to remove an unprivileged file with `execve()`



8.3 Conclusion

From the result, we can assume that `system()` can do something privileged with privileged program, but `execve()` cannot.

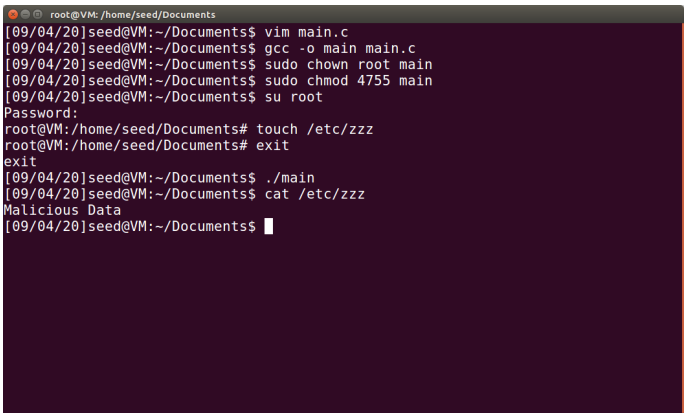
9 Task 9: Capability Leaking

9.1 Requirements

1. Study the capability leaking

9.2 Steps and result

1. Following step provided in the manual and observe the result.



2. Result is that the privileged file has still been written, even after setting uid.

9.3 Conclusion

From the result, we can find that if we do not close the file handle, user, without privilege, can still change privileged file.