实验 1 环境变量和 Set-UID 程序

Task 1: Manipulating Environment Variables

使用 printenv 输出环境变量 PATH 的值，即文件路径：

```
[09/02/20]seed@VM:~$ printenv 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/andro
id-sdk-linux/tools:/home/seed/android/android-sdk-linux/platform-tools:/home/see
d/android/android-ndk/android-ndk-r8d:/home/seed/.local/bin
[09/02/20]seed@VM:~$
```

使用 export 新建并初始化环境变量 MILK，输出其值，使用 unset 取消环境变量，输出发现取消成功：

```
[09/02/20]seed@VM:~$ export MILK=123
[09/02/20]seed@VM:~$ printenv MILK
123
[09/02/20]seed@VM:~$ unset MILK
[09/02/20]seed@VM:~$ printenv MILK
[09/02/20]seed@VM:~$
```

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

分别编译原有程序，和注释掉子进程的情况下(①行)的 printenv()、添加父进程的情况下(②行)的 printenv()的程序，将输出结果分别存储到 a.out 和 b.out，使用 diff 命令比较两个文件的差异如下：

```
[09/02/20]seed@VM:~$ diff child1 child2
77c77
< _=./a.out
---
> _=./b.out
[09/02/20]seed@VM:~$
```

除了输出文件名，发现两文件输出完全相同。

Task 3: Environment Variables and execve()

在这个任务中，我们研究当通过 execve()执行一个新程序时环境变量是如何受到影响的。

函数 execve()调用一个系统调用来加载并执行一个新命令，这个函数永远不会返回。不创建新流程，并且调用进程的文本、数据、bss 和堆栈被加载的程序的文本覆盖。实际上，execve()在调用进程内部运行新程序。

改变 execve()的调用线为 execve(“/usr/bin/env”, argv,environ)，新程序输出结果为：

```
[09/02/20]seed@VM:~$ ./c.out
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:4fd59b52-db41-4183-95bb-5984cbc20d1e
CLUTTER_IM_MODULE=xim
SESSION=ubuntu
GIO_LAUNCHED_DESKTOP_FILE_PID=3796
ANDROID_HOME=/home/seed/android/android-sdk-linux
GPG_AGENT_INFO=/home/seed/.gnupg/S.gpg-agent:0:1
TERM=xterm
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/libboos
t_filesystem.so.1.64.0:/home/seed/lib/boost/libboost_system.so.1.64.0
WINDOWID=58720260
UPSTART_SESSION=unix:abstract=/com/ubuntu/upstart-session/1000/1449
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:ca=30;41:tw=30;42:ow=34;42:st=37;44:ex=01;32:*.tar=01;31:*.tgz=0
1;31:*.arc=01;31:*.arj=01;31:*.taz=01;31:*.lha=01;31:*.lz4=01;31:*.lzh=01;31:*.lzma=01;31:*.tlz
```

输出结果得，环境变量不会自动继承环境变量，需要通过参数传递。

Task 4: Environment Variables and system()

在这个任务中，我们研究了当通过 system()函数执行一个新程序时环境变量是如何受到影响的。该函数用于执行命令，但与直接执行命令的 execve()不同，system()实际上执行“/bin/sh -c 命令”，即执行/bin/sh，并要求 shell 执行该命令。

编译原程序得：

```
[09/02/20]seed@VM:~$ gcc test4.c -o d.out
[09/02/20]seed@VM:~$ ./d.out
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/l
ib:
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-nUVkPAOd8h
GIO_LAUNCHED_DESKTOP_FILE_PID=3796
COLORTERM=gnome-terminal
GNOME_KEYRING_CONTROL=
QT_QPA_PLATFORMTHEME=appmenu-qt5
MANDATORY_PATH=/usr/share/gconf/ubuntu.mandatory.path
```

system()使用 execl()来执行/bin/sh，execl()调用 execve()，并将环境变量数组传递给它，因此使用 system()，调用进程的环境变量将被传递给新的程序/bin/sh。

Task 5: Environment Variable and Set-UID Programs

Set-UID 是 Unix 操作系统中一种重要的安全机制。当 Set-UID 程序运行时，它假定所有者的特权。例如，如果程序的所有者是 root，那么当任何人运行时这个程序在执行过程中获得根用户的特权。Set-UID 允许我们做许多有趣的事情，但是它在执行时升级了用户的特权，使其风险很大，虽然 Set-UID 程序的行为是由它们的程序逻辑决定的，而不是由用户决定的，但用户确实可以影响

通过环境变量的行为。为了理解 Set-UID 程序是如何受到影响的，让我们首先弄清楚 Set-UID 程序的进程是否从用户的进程继承了环境变量。

步骤 3。在您的 shell(您需要在一个普通的用户帐户，而不是根帐户)，使用 export 命令来设置以下环境变量(它们可能已经存在)：

编写原程序，结果如下图：

```
[09/02/20]seed@VM:~$ ./e.out
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:d80b2b1a-39ac-4e24-81d6-58714f5498db
CLUTTER_IM_MODULE=xim
SESSION=ubuntu
GIO_LAUNCHED_DESKTOP_FILE_PID=5094
ANDROID_HOME=/home/seed/android/android-sdk-linux
GPG_AGENT_INFO=/home/seed/.gnupg/S.gpg-agent:0:1
TERM=xterm
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/li
b/boost/libboost_filesystem.so.1.64.0:/home/seed/lib/boost/libboost_system.so.1.
64.0
WINDOWID=58720260
UPSTART_SESSION=unix:abstract=/com/ubuntu/upstart-session/1000/1449
GNOME_KEYRING_CONTROL=
```

将程序所有权改为 root，并使其成为 Set-UID 程序：

```
[09/02/20]seed@VM:~$ sudo chown root e.out
[09/02/20]seed@VM:~$ sudo chmod 4755 e.out
[09/02/20]seed@VM:~$ ls -l e.out
-rwsr-xr-x 1 root seed 7396 Sep  2 07:23 e.out
[09/02/20]seed@VM:~$
```

在 shell(在一个普通的用户帐户，而不是根帐户)，使用 export 命令来设置环境变量 LD_LIBRARY_PATH 和 ANY_NAME:

```
[09/02/20]seed@VM:~$ env |grep LD_LIBRARY_PATH
LD_LIBRARY_PATH=/home/seed/source/boost_1_64_0/stage/lib:/home/seed/source/boost
_1_64_0/stage/lib:
[09/02/20]seed@VM:~$ ./e.out |grep LD_LIBRARY_PATH
[09/02/20]seed@VM:~$ ls -l e.out
-rwsr-xr-x 1 root seed 7396 Sep  2 07:23 e.out
[09/02/20]seed@VM:~$
```

```
[09/02/20]seed@VM:~$ export MY_PATH=/home/seed
[09/02/20]seed@VM:~$ env |grep MY_PATH
MY_PATH=/home/seed
[09/02/20]seed@VM:~$ ./e.out |grep MY_PATH
MY_PATH=/home/seed
[09/02/20]seed@VM:~$
```

```
[09/02/20]seed@VM:~$ ./e.out |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/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:/hom
e/seed/android/android-ndk/android-ndk-r8d:/home/seed/.local/bin
MANDATORY_PATH=/usr/share/gconf/ubuntu.mandatory.path
COMPIZ_BIN_PATH=/usr/bin/
MY_PATH=/home/seed
[09/02/20]seed@VM:~$
```

这些环境变量是在用户的 shell 进程中设置的。在 shell 中键入程序名称后，shell 将派生一个子进程，并使用该子进程运行程序。由结果所示，Set-UID 子进程可以继承自定义的环境变量，也可以继承 PATH，但是 shell 进程(父进程)中设置的环境变量并没有进入子进程。

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

由于调用的是 shell 程序，在 Set-UID 程序中调用 system()是非常危险的，这是因为 shell 程序的实际行为会受到环境变量的影响，比如 PATH，这些环境变量是由用户提供的，通过改变这些变量，恶意用户可以控制 Set-UID 程序的行为。

编译原程序，并将其所有者更改为 root，并将其设置为 Set-UID 程序，运行 ls：

```
[09/02/20]seed@VM:~$ sudo chown root f.out
[09/02/20]seed@VM:~$ sudo chmod 4755 f.out
[09/02/20]seed@VM:~$ ls
android  child2         d.out             get-pip.py  source     test4.c
a.out    c.out          Downloads         lib         Templates  test5.c
bin      Customization  e.out             Music       test2.cpp  test6.c
b.out    Desktop        examples.desktop  Pictures    test3.c    test.cpp
child1   Documents      f.out             Public      test3.cpp  Videos
[09/02/20]seed@VM:~$
```

上述程序是 root/bin 的 ls 程序，新建 myls：

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

int main()
{
  printf("myls");
  return 0;
}
```

将该文件 copy 到/home/bin/目录下，执行 myls，输出 myls 而不是该路径下文件，说明执行的是自己的 ls 程序：

```
[09/02/20]seed@VM:~$ cp ls ~/bin/
[09/02/20]seed@VM:~$ cd ~/bin/
[09/02/20]seed@VM:~/bin$ ls
myls
[09/02/20]seed@VM:~/bin$
```

将自己的 ls 程序中的 printf 改为 root 才有权限执行的 system：

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

int main()
{
  system("cat /etc/shadow");
  return 0;
}
```

再次执行 ls 发现没有权限：

```
[09/02/20]seed@VM:~$ gcc mytest.c -o ls
[09/02/20]seed@VM:~$ cp ls ~/bin/
[09/02/20]seed@VM:~$ ls
cat: /etc/shadow: Permission denied
[09/02/20]seed@VM:~$
```

说明通过改变环境变量，恶意用户可以控制 Set-UID 程序的行为，但不能使用 root 权限。

Task 7: The LD PRELOAD Environment Variable and Set-UID Programs

在这个任务中，我们研究 Set-UID 程序如何处理一些环境变量。

将原程序 mylib 转换为动态链接库文件，并设置 LD_PRELOAD 环境变量：

```
[09/02/20]seed@VM:~$ gedit mylib.c
[09/02/20]seed@VM:~$ gcc -fPIC -c mylib.c
[09/02/20]seed@VM:~$ gcc -shared -o libmylib.so.1.0.1 mylib.o
-lc
[09/02/20]seed@VM:~$ export LD_PRELOAD=./libmylib.so.1.0.1
[09/02/20]seed@VM:~$
```

1. 将 myprog 作为一个常规程序，并以普通用户的身份运行它：

```
myprog.c: In function ‘main’:
myprog.c:6:3: warning: implicit declaration of function ‘sleep’ [-
Wimplicit-function-declaration]
   sleep(1);
   ^
[09/02/20]seed@VM:~$ ./myprog
[09/02/20]seed@VM:~$
```

2. 将 myprog 设置为一个 Set-UID 的 root 程序，并以普通用户的身份运行它：

```
[09/02/20]seed@VM:~$ sudo chown root myprog
[09/02/20]seed@VM:~$ sudo chmod 4755 myprog
[09/02/20]seed@VM:~$ ./myprog
[09/02/20]seed@VM:~$
```

2. 将 myprog 设置为一个 Set-UID 的 root 程序，在 root 帐户中再次导出 LD_PRELOAD 环境变量并运行它：

```
[09/02/20]seed@VM:~$ sudo su
root@VM:/home/seed# export LD_PRELOAD=./libmylib.so.1.0.1
root@VM:/home/seed# ./myprog
I am not sleeping!
root@VM:/home/seed#
```

4. 将 myprog 设置为一个 user1 的 Set-UID 的程序，在另一个用户的帐户(非根用户)中再次导出 LD_PRELOAD 环境变量并运行它：

```
[09/02/20]seed@VM:~$ sudo useradd newman -m
[09/02/20]seed@VM:~$ sudo chown newman myprog
[09/02/20]seed@VM:~$ sudo chmod 4755 myprog
```

```
[09/02/20]seed@VM:~$ export LD_PRELOAD=./libmylib.so.1.0.1
[09/02/20]seed@VM:~$ ./myprog
[09/02/20]seed@VM:~$
```

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

虽然 system()和 execve()都可以用于运行新程序，但如果在有特权的程序(如 Set-UID 程序)中使用 system()，则相当危险。我们已经看到了 PATH 环境变量如何影响 system()的行为，因为该变量影响 shell 的工作方式。execve()没有问题，因为它不调用 shell。调用 shell 还有另一个危险的后果，这一次，它与环境变量无关。让我们看一下下面的场景。

鲍勃在一家审计机构工作，他需要调查一家公司涉嫌欺诈。出于调查的目的，Bob 需要能够读取公司 Unix 系统中的所有文件;另一方面，为了保护系统的完整性，Bob 应该不能修改任何文件。为了实现这个目标，系统的超级用户 Vince 编写了一个特殊的 set-root-uid 程序(见下面)，然后给出了 Bob 的可执行权限。该程序要求 Bob 在命令行键入文件名，然后运行/bin/cat 来显示指定的文件。由于程序是以根用户的身份运行的，所以它可以显示 Bob 指定的任何文件。然而，由于程序没有写操作，Vince 非常确定 Bob 不能使用这个特殊的程序来修改任何文件。

编译原程序：

```
[09/03/20]seed@VM:~$ gcc test8.c -o h.o
[09/03/20]seed@VM:~$ ls
android         Downloads          lib       Public      test9B.c
bin             examples.desktop   ma        source      test9.c
Customization   get-pip.py         ma.c      Templates   Videos
Desktop         h.o                Music     test8.c
Documents       i.o                Pictures  test9B
```

将该程序设置为一个 Set-UID 的 root 程序：

```
[09/03/20]seed@VM:~$ sudo chown root h.o
[09/03/20]seed@VM:~$ sudo chmod 4755 h.o
[09/03/20]seed@VM:~$ ls
android         Downloads          lib       Public      test9B.c
bin             examples.desktop   ma        source      test9.c
Customization   get-pip.py         ma.c      Templates   Videos
Desktop         h.o                Music     test8.c
Documents       i.o                Pictures  test9B
[09/03/20]seed@VM:~$
```

普通用户不可以删除 root 下的文件，但可以强制删除 seed 用户下 root 创建的文件：

```
[09/03/20]seed@VM:~$ ./h.o "i.o; /bin/rm /home/seed/i.o"
```

```
[09/03/20]seed@VM:~$ ls
android         Downloads          ma        source      test9.c
bin             examples.desktop   ma.c      Templates   Videos
Customization   get-pip.py         Music     test8.c
Desktop         h.o                Pictures  test9B
Documents       lib                Public    test9B.c
[09/03/20]seed@VM:~$
```

使用 execve 发现 i.o 没有被删除:

```
[09/03/20]seed@VM:~$ ./h.o ./i.o
```

```
[09/03/20]seed@VM:~$ ls
android         Downloads          lib       Public      test9B.c
bin             examples.desktop   ma        source      test9.c
Customization   get-pip.py         ma.c      Templates   Videos
Desktop         h.o                Music     test8.c
Documents       i.o                Pictures  test9B
[09/03/20]seed@VM:~$
```

Task 9: Capability Leaking

为了遵循最小权限原则，如果不再需要 root 权限，Set-UID 程序通常会永久放弃它们的 root 权限,且有时候程序需要把它的控制权交给用户,在这种情况下，必须撤销 root 特权,可以使用 setuid()系统调用来撤销特权。setuid()设置调用进程的有效用户 ID 如果调用者的有效 UID 是 root，则还设置了实际的 UID 和已保存的 set-user id，如果有效 UID 为 0，Set-UID 程序调用 setuid(n)，该进程将成为普通进程，其所有 UID 都被设置为 n。

撤销特权时，一个常见的错误是功能泄漏。这个过程可能有在它还享有特权的时候获得了一些特权能力;当特权被降级时，如果程序没有清除那些功能，那么非特权进程仍然可以访问它们。此时尽管进程的有效用户 ID 不再具有特权，但进程仍然具有特权，因为它拥有特权功能。

编译自己的程序 test9.c:

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

void main()
{
  printf("Test9\n");
}
```

将 test9.c 复制到/etc/zzz 文件中：

```
[09/03/20]seed@VM:~$ gcc test9.c -o i.o
[09/03/20]seed@VM:~$ sudo su
root@VM:/home/seed# cp test9.c /etc/zzz
root@VM:/home/seed# ls -l /etc/zzz
-rw-r--r-- 1 root root 77 Sep  3 13:38 /etc/zzz
root@VM:/home/seed#
```

编译题中所给程序，将其所有者更改为 root，并使其成为 Set-UID 程序，以普通用户的身份运行程序，发现文件/etc/zzz 已被修改：

```
[09/03/20]seed@VM:~$ gedit ma.c
[09/03/20]seed@VM:~$ gcc ma.c -o ma
[09/03/20]seed@VM:~$ sudo chown root ma
[09/03/20]seed@VM:~$ sudo chmod 4755 ma
[09/03/20]seed@VM:~$ ls
android         Downloads          ma         source     Videos
bin             examples.desktop   ma.c       Templates
Customization   get-pip.py         Music      test9B
Desktop         i.o                Pictures   test9B.c
Documents       lib                Public     test9.c
[09/03/20]seed@VM:~$ ./ma
[09/03/20]seed@VM:~$ sudo cat /etc/zzz
#include <stdio.h>
#include <stdlib.h>

void main()
{
  printf("Test9\n");
}
Malicious Data
[09/03/20]seed@VM:~$
```

虽然可以修改文件内容，，但用户没有 root 权限来删除该文件：

```
[09/03/20]seed@VM:~$ rm /etc/zzz
rm: remove write-protected regular file '/etc/zzz'? y
rm: cannot remove '/etc/zzz': Permission denied
[09/03/20]seed@VM:~$
```