网络空间安全实训 实验报告一

57118107 任子悦

Task 1: Manipulating Environment Variables

```
/bin/bash
/bin/bash 63x15
[09/01/20]seed@VM:~$ printenv
XDG_VTNR=7
ORBIT_SOCKETDIR=/tmp/orbit-seed
```

```
/bin/bash
/bin/bash 63x15
[09/01/20]seed@VM:~$ env
XDG_VTNR=7
ORBIT_SOCKETDIR=/tmp/orbit-seed
XDG_SESSION_ID=c1
```

```
[09/01/20]seed@VM:~$ printenv USER
seed
[09/01/20]seed@VM:~$ env | grep SHELL
SHELL=/bin/bash
```

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

```
[09/01/20]seed@VM:~$ diff child parent
[09/01/20]seed@VM:~$
```

结论：子进程与父进程的环境变量完全相同，子进程继承了父进程的全部环境变量

Task 3: Environment Variables and execve()

① Step 1: execve 函数环境变量参数为空，运行程序没有结果输出

```
/bin/bash
/bin/bash 80x24
[09/02/20]seed@VM:~$ vim execve.c
[09/02/20]seed@VM:~$ gcc -o execve execve.c
[09/02/20]seed@VM:~$ ./execve
[09/02/20]seed@VM:~$
```

② Step 2: execve 函数环境变量为 environ，程序有输出

```
[09/02/20]seed@VM:~$ vim execve.c
[09/02/20]seed@VM:~$ gcc -o execve execve.c
[09/02/20]seed@VM:~$ ./execve
XDG_VTNR=7
ORBIT_SOCKETDIR=/tmp/orbit-seed
```

说明 execve 函数需要使用环境变量时必须显式传参，环境变量不会自动继承

Task 4: Environment Variables and system()

```
[09/02/20]seed@VM:~$ vim task4.c
[09/02/20]seed@VM:~$ gcc -o task4 task4.c
[09/02/20]seed@VM:~$ ./task4
LESSOPEN=| /usr/bin/lesspipe %s
GNOME_KEYRING_PID=
USER=seed
LANGUAGE=en_US
UPSTART_INSTANCE=
```

task4.cpp 源码:

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

int main()
{
        system("/usr/bin/env");
        return 0;
}
```

system()函数通过调用 execl()hanshu ， 启动 shell 来执行命令

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

修改 foo 程序所有者，改为 set-uid 程序:

```
[09/02/20]seed@VM:~$ vim foo.c
[09/02/20]seed@VM:~$ gcc -o foo foo.c
[09/02/20]seed@VM:~$ sudo chown root foo
[09/02/20]seed@VM:~$ sudo chmod 4755 foo
[09/02/20]seed@VM:~$ ls -l foo
-rwsr-xr-x 1 root seed 7396 Sep  2 04:04 foo
[09/02/20]seed@VM:~$
```

父进程的环境变量:

```
[09/02/20]seed@VM:~$ env | grep PATH
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
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/
[09/02/20]seed@VM:~$
```

修改了 PATH、LD_LIBRARY_PATH，新增环境变量 PATH_TEST:

```
[09/02/20]seed@VM:~$ export PATH=/home/seed:$PATH
[09/02/20]seed@VM:~$ export LD_LIBRARY_PATH=./:$LD_LIARBRY_PATH
[09/02/20]seed@VM:~$ export PATH_TEST=test
```

继承了 PATH 和自定义环境变量，没有继承 LD_LIBRARY_PATH:

```
[09/02/20]seed@VM:~$ ./foo | grep PATH
PATH_TEST=test
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:/usr/bin
:/sbin:/bin:/usr/games:/usr/local/games:.:/snap/bin:/usr/lib/jvm/java-8-oracle/b
in:/usr/lib/jvm/java-8-oracle/db/bin:/usr/lib/jvm/java-8-oracle/jre/bin:/home/se
ed/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
MANDATORY_PATH=/usr/share/gconf/ubuntu.mandatory.path
COMPIZ_BIN_PATH=/usr/bin/
```

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

编译运行 task6 程序，更改所有者为 root，改为 set-uid 程序：

```
[09/02/20]seed@VM:~$ vim task6.cpp
[09/02/20]seed@VM:~$ g++ -o task6 task6.cpp
[09/02/20]seed@VM:~$ ls -l task6
-rwxrwxr-x 1 seed seed 7636 Sep  2 03:15 task6
[09/02/20]seed@VM:~$ sudo chown root task6
[09/02/20]seed@VM:~$ sudo chmod 4755 task6
[09/02/20]seed@VM:~$ ls -l task6
-rwsr-xr-x 1 root seed 7636 Sep  2 03:15 task6
[09/02/20]seed@VM:~$ task6
android         Documents         get-pip.py  parent    task2.c  task6
bin             Downloads         lib         Pictures  task3    task6.cpp
```

task6 程序源码：

```
//task6.cpp
/*supposed to execute /bin/ls */
#include <stdlib.h>
#include <iostream>
int main()
{
        system("ls");
        return 0;
}
```

恶意 ls 程序源码：

```
/*malicious "ls" program*/
#include <iostream>
using namespace std;

int main()
{
        cout<<"This is a malicious program!"<<endl;
        return 0;
}
```

更改 PATH，恶意 ls 程序运行成功：

```
[09/02/20]seed@VM:~$ sudo ln -sf /bin/zsh /bin/sh
[09/02/20]seed@VM:~$ vim ls.cpp
[09/02/20]seed@VM:~$ g++ -o ls ls.cpp
[09/02/20]seed@VM:~$ export PATH=/home/seed:$PATH
[09/02/20]seed@VM:~$ echo $PATH
/home/seed:/home/seed/bin:/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbi
n:/bin:/usr/games:/usr/local/games:.:/snap/bin:/usr/lib/jvm/java-8-oracle/bin:/u
sr/lib/jvm/java-8-oracle/db/bin:/usr/lib/jvm/java-8-oracle/jre/bin:/home/seed/an
droid/android-sdk-linux/tools:/home/seed/android/android-sdk-linux/platform-tool
s:/home/seed/android/android-ndk/android-ndk-r8d:/home/seed/.local/bin
[09/02/20]seed@VM:~$ task6
This is a malicious program!
[09/02/20]seed@VM:~$
```

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

1) Step1

mylib.c 源码:

```
//mylib.c
#include <stdio.h>
void sleep(int s)
{
        printf("I am not sleeping!\n");
}
```

自定义 mylib 动态链接库，将 LD_PRELOAD 改为自定义库:

```
[09/02/20]seed@VM:~$ vim mylib.c
[09/02/20]seed@VM:~$ gcc -fPIC -g -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
```

myprog 程序源码:

```
/* myprog.c */
#include <unistd.h>
int main()
{
        sleep(1);
        return 0;
}
```

```
[09/02/20]seed@VM:~$ vim myprog.c
[09/02/20]seed@VM:~$ gcc -o myprog myprog.c
```

2) Step2

① myprog 为常规程序，以普通用户身份运行；使用了自定义 sleep 动态库，说明子进程继承了 LD_PRELOAD

```
[09/02/20]seed@VM:~$ myprog
I am not sleeping!
[09/02/20]seed@VM:~$
```

② myprog 为 set-uid 程序，作为普通用户运行；使用了系统自带的 sleep，因为动态连接的防御机制，当进程为 set-uid 时，EUID 和 RUID 不同，系统忽略了 LD_PRELOAD 环境变

量。

```
[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:~$
```

③ myprog 为 set-uid 程序，以 root 用户身份 export LD_PRELOAD，运行程序；使用了自定义的动态库；用户特权提升，没有防御机制。

```
[09/02/20]seed@VM:~$ su
Password:
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 myprog 为 set-uid 程序，以 user1 用户身份 export LD_PRELOAD，运行程序；使用了系统自带的 sleep；防御机制，LD_PRELOAD 被忽略。

```
root@VM:/home/seed# useradd user1
root@VM:/home/seed# passwd user1
Enter new UNIX password:
Retype new UNIX password:
passwd: password updated successfully
root@VM:/home/seed# su user1
user1@VM:/home/seed$ export LD_PRELOAD=./libmylib.so.1.0.1
user1@VM:/home/seed$ myprog
user1@VM:/home/seed$
```

3) 验证

```
[09/02/20]seed@VM:~$ cp /usr/bin/env ./myenv
[09/02/20]seed@VM:~$ sudo chown root myenv
[09/02/20]seed@VM:~$ sudo chmod 4755 myenv
[09/02/20]seed@VM:~$ ls -l myenv
-rwsr-xr-x 1 root seed 30460 Sep  2 05:26 myenv
[09/02/20]seed@VM:~$ export LD_PRELOAD=./libmylib.so.1.0.1
[09/02/20]seed@VM:~$ export LD_LIBRARY_PATH=.
[09/02/20]seed@VM:~$ export LD_TEST="testld"
[09/02/20]seed@VM:~$ export TEST_ENVIRON="testenviron"
[09/02/20]seed@VM:~$ env | grep LD_
LD_PRELOAD=./libmylib.so.1.0.1
LD_LIBRARY_PATH=.
LD_TEST=testld
[09/02/20]seed@VM:~$ myenv | grep LD_
LD_TEST=testld
[09/02/20]seed@VM:~$ env | grep TEST
TEST_ENVIRON=testenviron
LD_TEST=testld
[09/02/20]seed@VM:~$ myenv | grep TEST
TEST_ENVIRON=testenviron
LD_TEST=testld
[09/02/20]seed@VM:~$
```

拷贝 env 程序，更名为 myenv，并将其修改为 set-uid 程序，运行 myenv 时 LD_PRELOAD 和 LD_LIBRARY_PATH 被忽略，自定义的 LD_TEST 没有被忽略

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

audit.c 源码：

```
//audit.c
#include <string.h>
#include <stdio.h>
#include <stdlib.h>
#include <unistd.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;
}
```

编写 audit 程序，修改为 root 权限 set-uid 程序:

```
[09/02/20]seed@VM:~$ touch audit.c
[09/02/20]seed@VM:~$ vim audit.c
[09/02/20]seed@VM:~$ vim audit.c
[09/02/20]seed@VM:~$ gcc -o audit audit.c
[09/02/20]seed@VM:~$ sudo chown root audit
[09/02/20]seed@VM:~$ sudo chmod 4755 audit
```

```
[09/03/20]seed@VM:~$ ls -l audit
-rwsr-xr-x 1 root seed 7544 Sep  2 11:50 audit
[09/03/20]seed@VM:~$
```

在/root 目录下新增文件 file:

```
[09/03/20]seed@VM:~$ su
Password:
root@VM:/home/seed# cd /root
root@VM:~# ls
root@VM:~# vim file
root@VM:~#
```

正常情况普通用户无法写或删除 file 文件:

```
[09/03/20]seed@VM:~$ rm /root/file
rm: cannot remove '/root/file': Permission denied
[09/03/20]seed@VM:~$
```

①Step 1: system()函数

使用 audit 程序可以删除/root/file:

```
[09/03/20]seed@VM:~$ ./audit "aa; rm /root/file"
/bin/cat: aa: No such file or directory
[09/03/20]seed@VM:~$ cd /root
bash: cd: /root: Permission denied
[09/03/20]seed@VM:~$ su
Password:
root@VM:/home/seed# cd /root
root@VM:~# ls
root@VM:~#
```

② Step 2: execve()

修改 audit.c 文件，编译，更改权限和属性，再次运行 audit 程序，不能删除 file 文件:

```
[09/03/20]seed@VM:~$ vim audit.c
[09/03/20]seed@VM:~$ gcc -o audit audit.c
[09/03/20]seed@VM:~$ ls -l audit
-rwxrwxr-x 1 seed seed 7544 Sep  3 20:44 audit
[09/03/20]seed@VM:~$ sudo chown root audit
[09/03/20]seed@VM:~$ sudo chmod 4755 audit
[09/03/20]seed@VM:~$ ls -l audit
-rwsr-xr-x 1 root seed 7544 Sep  3 20:44 audit
[09/03/20]seed@VM:~$ ./audit "aa; /root/file"
/bin/cat: 'aa; /root/file': No such file or directory
[09/03/20]seed@VM:~$ ./audit "aa; rm /root/file"
/bin/cat: 'aa; rm /root/file': No such file or directory
```

攻击失效，说明 execve()通过设置参数，避免了特权泄露，不能将数据转变为命令。

Task 9: Capability Leaking

编辑/etc/zzz 文件，文件内容为“zzz system file”:

```
[09/02/20]seed@VM:~$ vim /etc/zzz
[09/02/20]seed@VM:~$ cat /etc/zzz
zzz system file
```

编辑 leak 源文件，更改所有者为 root，将 leak 程序变为 set-uid 程序:

```
[09/02/20]seed@VM:~$ vim leak.c
[09/02/20]seed@VM:~$ gcc -o leak leak.c
[09/02/20]seed@VM:~$ sudo chown root leak
[09/02/20]seed@VM:~$ sudo chmod 4755 leak
[09/02/20]seed@VM:~$ ls -l leak
-rwsr-xr-x 1 root seed 7640 Sep  2 08:30 leak
```

leak 源文件:

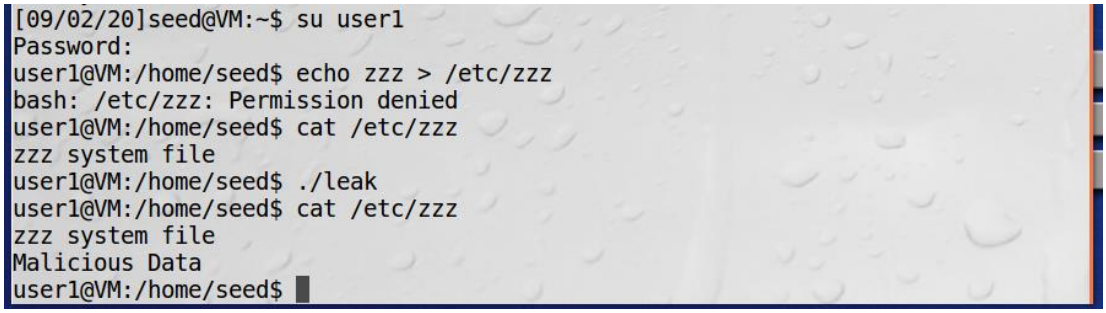
```
#include <stdio.h>
#include <stdlib.h>
#include <fcntl.h>
#include <unistd.h>
void main()
{
        int fd;
        /* Assume that /etc/zzz is an important system file,
```

```
 * 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);
}

/* Stimulate 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);
}
}
```

运行 leak 程序，发现/etc/zzz 文件被更改：

```
[09/02/20]seed@VM:~$ ./leak
[09/02/20]seed@VM:~$ cat /etc/zzz
zzz system file
Malicious Data
[09/02/20]seed@VM:~$
```

切换到另一个普通用户 user1，正常情况是不能够更改 zzz 文件的，但运行 leak 程序后文件可以被更改，说明 leak 是 set-uid 程序，发生了特权泄露：


[09/02/20]seed@VM:~$ su user1
Password:
user1@VM:/home/seed$ echo zzz > /etc/zzz
bash: /etc/zzz: Permission denied
user1@VM:/home/seed$ cat /etc/zzz
zzz system file
user1@VM:/home/seed$ ./leak
user1@VM:/home/seed$ cat /etc/zzz
zzz system file
Malicious Data
user1@VM:/home/seed$