Set-UID实验报告

57119101 王晨阳

2021年7月6日

TASK1

- 使用 `env` 查看环境变量

```
[07/06/21]seed@VM:~$ env
SHELL=/bin/bash
SESSION_MANAGER=local/VM:@/tmp/.ICE-unix/1990,unix/VM:/tmp/.ICE-unix/1990
QT_ACCESSIBILITY=1
COLORTERM=truecolor
XDG_CONFIG_DIRS=/etc/xdg/xdg-ubuntu:/etc/xdg
XDG_MENU_PREFIX=gnome-
GNOME_DESKTOP_SESSION_ID=this-is-deprecated
GNOME_SHELL_SESSION_MODE=ubuntu
SSH_AUTH_SOCK=/run/user/1000/keyring/ssh
XMODIFIERS=@im=ibus
DESKTOP_SESSION=ubuntu
SSH_AGENT_PID=1932
GTK_MODULES=gail:atk-bridge
PWD=/home/seed
LOGNAME=seed
XDG_SESSION_DESKTOP=ubuntu
XDG_SESSION_TYPE=x11
GPG_AGENT_INFO=/run/user/1000/gnupg/S.gpg-agent:0:1
XAUTHORITY=/run/user/1000/gdm/Xauthority
GJS_DEBUG_TOPICS=JS ERROR;JS LOG
WINDOWPATH=2
HOME=/home/seed
USERNAME=seed
IM_CONFIG_PHASE=1
LANG=en_US.UTF-8
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:*.dz=01;31:*.gz=01;31:*.lrz=01;31:*.lz=01;31:*.lzo=01;31:*.xz=01;31:*.zst
=01;31:*.tzst=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:*.wim=01;31:*.swm=01;31:*.dwm=01;31:
*.esd=01;31:*.jpg=01;35:*.jpeg=01;35:*.mjpg=01;35:*.mjpeg=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=0
```

然后查找 `PWD` 变量

```
[07/06/21]seed@VM:~$ env | grep PWD
PWD=/home/seed
```

- 使用 `export` 创建环境变量

```
[07/06/21]seed@VM:~$ export envvar=envvar1
[07/06/21]seed@VM:~$ env | grep envvar
envvar=envvar1
```

然后使用 `unset` 删除刚刚创建的变量

```
[07/06/21]seed@VM:~$ unset envvar
[07/06/21]seed@VM:~$ env | grep envvar
[07/06/21]seed@VM:~$
```

TASK2

- 编写程序 `myprintenv.c`

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

```
4
5   extern char **environ;
6
7   void printenv()
8   {
9       int i = 0;
10      while (environ[i] != NULL) {
11          printf("%s\n", environ[i]);
12          i++;
13      }
14  }
15
16  void main()
17  {
18      pid_t childPid;
19
20      switch(childPid = fork()) {
21          case 0: /* child process */
22          printenv();
23          exit(0);
24          default: /* parent process */
25          //printenv();
26          exit(0);
27      }
28  }
```

编译并保存结果到 `child`

```
1   $ gcc myprintenv.c -o child.out
2   $ child.out > child
```

- 修改程序 `myprintenv.c`

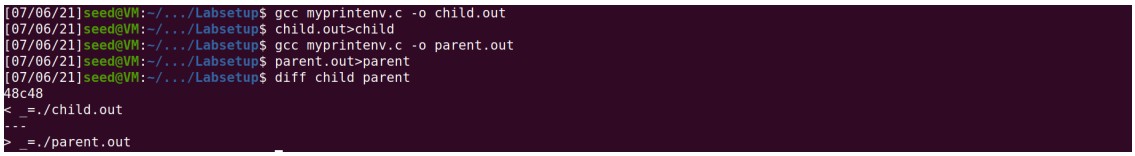
```
1   #include <unistd.h>
2   #include <stdio.h>
3   #include <stdlib.h>
4
5   extern char **environ;
6
7   void printenv()
8   {
9       int i = 0;
10      while (environ[i] != NULL) {
11          printf("%s\n", environ[i]);
12          i++;
13      }
14  }
15
16  void main()
17  {
18      pid_t childPid;
19
20      switch(childPid = fork()) {
21          case 0: /* child process */
22          //printenv();
23          exit(0);
24          default: /* parent process */
25          printenv();
```

```
            exit(0);
        }
    }
```

编译并保存结果到`parent`

```
$ gcc myprintenv.c -o parent.out
$ parent.out > parent
```

- 使用`diff`比较`child`和`parent`



`child`相较于`parent`，结果的第76行发生了改变。可以认为，**子进程和父进程除了pid几乎完全相同。**

TASK3

- 编写程序`myenv.c`

```
#include <stdio.h>
#include <stdlib.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, NULL);

    return 0 ;
}
```

编译并保存结果到`myenv_null`

```
$ gcc myenv.c -o myenv_null.out
$ myenv_null.out > myenv_null
```

- 修改程序`myenv.c`

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

extern char **environ;

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

```

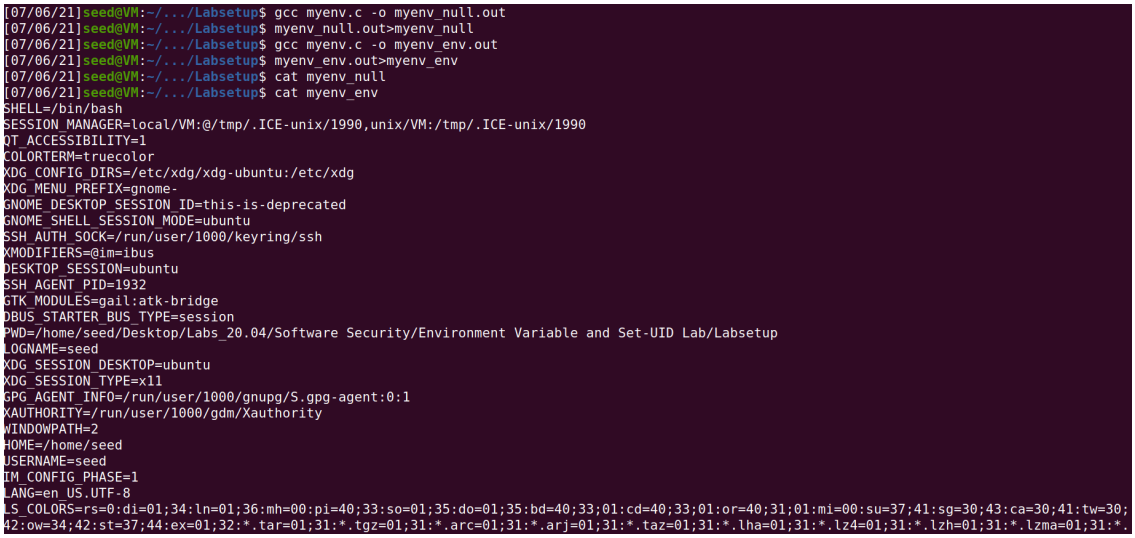
```
        argv[0] = "/usr/bin/env";
        argv[1] = NULL;

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

        return 0 ;
    }
```

编译并保存结果到`myenv_env`

```
$ gcc myenv.c -o myenv_env.out
$ myenv_env.out > myenv_env
```

- 观察结果

`3_null`为空，`3_environ`有内容



`execve()`函数的格式为

```
int execve(const char * filename, char * const argv[], char * const envp[])
```

在第一个程序中，我们没有向`envp[]`传入参数，故没有结果；

在第二个程序中，传入了环境变量，故能够打印出环境变量。

可见，`execve()`产生的新进程是被独立赋予环境变量的，相当于它是在已有进程上开启了新的进程。

TASK4

- 编写程序`4.c`

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

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

    return 0 ;
}
```

编译并保存结果到`4`

```
$ gcc 4.c -o 4.out
$ 4.out > 4
```

得到结果

```
[07/06/21]seed@VM:~/.../Labsetup$ gcc 4.c -o 4.out
[07/06/21]seed@VM:~/.../Labsetup$ 4.out>4
[07/06/21]seed@VM:~/.../Labsetup$ cat 4
LESSOPEN=| /usr/bin/lesspipe %s
USER=seed
SSH_AGENT_PID=1932
XDG_SESSION_TYPE=x11
SHLVL=1
HOME=/home/seed
DESKTOP_SESSION=ubuntu
GNOME_SHELL_SESSION_MODE=ubuntu
GTK_MODULES=gail:atk-bridge
MANAGERPID=1716
DBUS_STARTER_BUS_TYPE=session
DBUS_SESSION_BUS_ADDRESS=unix:path=/run/user/1000/bus,guid=a79fb70afc1ef63d1071f17e60e3f0e6
COLORTERM=truecolor
IM_CONFIG_PHASE=1
LOGNAME=seed
JOURNAL_STREAM=9:35354
_=./4.out
XDG_SESSION_CLASS=user
USERNAME=seed
TERM=xterm-256color
GNOME_DESKTOP_SESSION_ID=this-is-deprecated
WINDOWPATH=2
PATH=/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin:/usr/games:/usr/local/games:/snap/bin:.
SESSION_MANAGER=local/VM:@/tmp/.ICE-unix/1990,unix/VM:/tmp/.ICE-unix/1990
INVOCATION_ID=b1f6421a24414a97a63e6befa3414cc3
XDG_MENU_PREFIX=gnome-
GNOME_TERMINAL_SCREEN=/org/gnome/Terminal/screen/557bfeb7_7b08_4aff_ba5b_7217bc4bcbc2
XDG_RUNTIME_DIR=/run/user/1000
DISPLAY=:0
LANG=en_US.UTF-8
```

可以看到，程序输出了环境变量。

TASK5

- 编写程序 `5.c`

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

extern char **environ;

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

- 编译程序到 `5.out`

```
$ gcc 5.c -o 5.out
```

修改权限，然后使其成为Set-UID程序

```
[07/06/21]seed@VM:~/.../Labsetup$ gcc 5.c -o 5.out
[07/06/21]seed@VM:~/.../Labsetup$ sudo chown root 5.out
[07/06/21]seed@VM:~/.../Labsetup$ sudo chmod 4755 5.out
```

- 检查 PATH 和 LD_LIBRARY_PATH 环境变量是否存在

```
[07/06/21]seed@VM:~/.../Labsetup$ env | grep PATH
WINDOWPATH=2
PATH=/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin:/usr/games:/usr/local/games:/snap/bin:.
```

新建 `ANY_NAME` 环境变量

```
[07/06/21]seed@VM:~/.../Labsetup$ export ANY_NAME=ANYNAME
[07/06/21]seed@VM:~/.../Labsetup$ env | grep ANY_NAME
ANY_NAME=ANYNAME
```

使用刚刚的程序打印这三个环境变量

```
[07/06/21]seed@VM:~/.../Labsetup$ 5.out | grep PATH
WINDOWPATH=2
PATH=/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin:/usr/games:/usr/local/games:/snap/bin:.
[07/06/21]seed@VM:~/.../Labsetup$ 5.out | grep ANY_NAME
ANY_NAME=ANYNAME
```

尽管`5.out`设置为root所有，但因为其设置了SUID权限，故可以通过它产生拥有特殊权限地子进程，打印环境变量。

TASK6

- 编写程序`6.c`

```
#include<stdlib.h>

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

编译程序到`6.out`

```
$ gcc 6.c -o 6.out
```

修改权限，然后使其成为Set-UID程序

```
$ sudo chown root 6.out
$ sudo chmod 4755 6.out
```

使用`6.out`实现`ls`的功能

```
[07/06/21]seed@VM:~/.../Labsetup$ gcc 6.c -o 6.out
[07/06/21]seed@VM:~/.../Labsetup$ sudo chown root 6.out
[07/06/21]seed@VM:~/.../Labsetup$ sudo chmod 4755 6.out
[07/06/21]seed@VM:~/.../Labsetup$ 6.out
4    4.out  5.out  6.out       catall.c  child.out  myenv_env      myenv_null      myprintenv.c  parent.out
4.c  5.c    6.c    cap_leak.c  child     myenv.c    myenv_env.out  myenv_null.out  parent
```

SUID程序成功执行了`ls`指令。

TASK8

- 新建文件`tmp`

```
$ touch tmp
$ vim tmp
```

编辑内容为

```
tmp file
```

更改权限发现不能访问

```
[07/06/21]seed@VM:~/.../Labsetup$ touch tmp
[07/06/21]seed@VM:~/.../Labsetup$ vim tmp
[07/06/21]seed@VM:~/.../Labsetup$ cat tmp
tmp file
[07/06/21]seed@VM:~/.../Labsetup$ sudo chmod 000 tmp
[07/06/21]seed@VM:~/.../Labsetup$ cat tmp
cat: tmp: Permission denied
```

编写程序`catall.c`

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

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

```
9
10      if(argc < 2) {
11          printf("Please type a file name.\n");
12          return 1;
13      }
14
15      v[0] = "/bin/cat"; v[1] = argv[1]; v[2] = NULL;
16      command = malloc(strlen(v[0]) + strlen(v[1]) + 2);
17      sprintf(command, "%s %s", v[0], v[1]);
18
19      // Use only one of the followings.
20      system(command);
21      // execve(v[0], v, NULL);
22
23      return 0 ;
24  }
```

编译程序到 `catall`

```
$ gcc catall.c -o catall
```

将 `catall` 设置为Set-UID root程序并运行

```
[07/06/21]seed@VM:~/.../Labsetup$ gcc catall.c -o catall
[07/06/21]seed@VM:~/.../Labsetup$ sudo chown root catall
[07/06/21]seed@VM:~/.../Labsetup$ sudo chmod 4755 catall
[07/06/21]seed@VM:~/.../Labsetup$ catall tmp
tmp file
```

- 修改程序 `catall.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 ;
}
```

编译程序到 `catall`

```
$ gcc catall.c -o catall
```

将 `catall` 设置为Set-UID root程序并运行

```
[07/06/21]seed@VM:~/.../Labsetup$ gcc catall.c -o catall
[07/06/21]seed@VM:~/.../Labsetup$ sudo chown root catall
[07/06/21]seed@VM:~/.../Labsetup$ sudo chmod 4775 catall
[07/06/21]seed@VM:~/.../Labsetup$ catall tmp
/bin/cat: tmp: Permission denied
```

`system()` 可以成功攻击，而 `execve()` 不能