

# Unix et Programmation Shell

Philippe Langevin

département d'informatique  
UFR sciences et technique  
université du sud Toulon Var

Automne 2013

## brouillon en révision

- site du cours :  
<http://langevin.univ-tln.fr/cours/UPS/upsh.html>
- localisation du fichier :  
<http://langevin.univ-tln.fr/cours/UPS/doc/perm.pdf>

## dernières modifications

upsh.tex	2024-01-28	21:05:32.653614398	+0100
tools.tex	2024-01-28	21:05:32.653614398	+0100
term.tex	2024-01-28	21:05:32.651614387	+0100
syntaxe.tex	2024-01-28	21:05:32.650614381	+0100
shell.tex	2024-01-28	21:05:32.649614376	+0100
prologue.tex	2024-01-28	21:05:32.648614371	+0100
proc.tex	2024-01-28	21:05:32.646614360	+0100
pipe.tex	2024-01-28	21:05:32.645614355	+0100
perm.tex	2024-01-28	21:05:32.645614355	+0100
man.tex	2024-01-28	21:05:32.643614344	+0100
part.tex	2024-01-28	21:05:32.644614349	+0100
file.tex	2024-01-28	21:05:32.642614338	+0100
direct.tex	2024-01-28	21:05:32.641614333	+0100
bash.tex	2024-01-28	21:05:32.639614322	+0100

## 5 - permission

- droit fichier
- changement
- droit répertoire
- droits spéciaux
- setuid bit
- faille

# permission

Un i-noeud est la propriété d'un utilisateur et d'un groupe :

```
~> chown toto:novice /tmp/canal
~> ls -l /tmp/canal
prw-rw-r--. 1 toto novice 0 ... /tmp/canal
~> stat /tmp/canal
  File: /tmp/canal
  Size: 0 Blocks: 0 IO Block: 4096 FIFO
Device: fd01h/64769d Inode: 787380 Links: 1
Access: (0664/prw-rw-r--)
  Uid: ( 1002/toto)   Gid: ( 1002/novice)
```

# drapeaux

Les permissions sont décrites par un nombre de 4 chiffres en base octale et forment 12 drapeaux de droit.

s	S	t		u	u	u		g	g	g		o	o	o
0	0	0		1	1	1		1	0	1		1	0	0
				r	w	x		r	-	x		r	-	-

TABLE – droit 0754

Le chiffre dominant concerne les droits spéciaux, les suivant les droits utilisateur, groupe et autre.

# affichage

s	S	t		u	u	u		g	g	g		o	o	o
1				r	w	s		r	w	x		r	w	x
	1			r	w	x		r	w	s		r	w	x
		1		r	w	x		r	w	x		r	w	t

TABLE – affichage des droits spéciaux

```
~> ls -l /bin/passwd
```

```
-rwsr-xr-x. 1 root root 2715 22 jui 02:51 /bin/
passwd
```

```
~> ls -l /usr/bin/wall
```

```
-r-xr-sr-x. 1 root tty 10588 18 fev 2013 /usr/
bin/wall
```

```
~> ls -ld /tmp
```

```
drwxrwxrwt 14 root root 480 18 sept. 18:22 /tmp
```

# droit fichier

Les drapeaux **rw**x sont définis par le propriétaire du fichier pour lui, le groupe et le reste du monde. Le propriétaire peut changer les permissions.

- **r** : fichier ouvert en lecture, copiable
- **w** : fichier ouvert en écriture, vidable
- **x** : exécutable

Les droits **suid** et **sgid** affectent ces droits.



# manipulation

La commande `chmod` permet d'ajuster les droits :

```
~> chmod u+x go-x /tmp/canal
```

de les sceller numériquement.

```
~> chmod 700 /tmp/canal
```

mode récursif :

```
~> chmod -R a-w /home/toto/pub
```

# priorité ugo

```
# useradd -g bar foo;useradd drm;useradd ken
# ls -l /home
drwx----- 4 drm    drm    4096 ... drm
drwx----- 4 foo    bar    4096 ... foo
drwx----- 4 ken    ken    4096 ... ken
# usermod -a -G bar drm ; groups drm
drm : drm bar
# echo world > /tmp/hello
# chown foo:bar /tmp/hello ; ls -l /tmp/hello
-rw-r--r-- 1 foo bar 6 ... /tmp/hello
# chmod ou-r /tmp/hello
```

```
# sudo -u ken cat /tmp/hello
cat: /tmp/hello: Permission non accordée
# sudo -u foo cat /tmp/hello
cat: /tmp/hello: Permission non accordée
# sudo -u drm cat /tmp/hello
world
```

# droit répertoire

- **r** : lecture du contenu du répertoire.
- **w** : permet la création, la suppression et le changement de noms
- **x** : autorise l'accès (ou ouverture) au répertoire.

Le **sticky bit** et le **sgid** affectent ces droits.

# x/dir

```
1 #!/bin/bash -v
2 mkdir -p /tmp/x/y
3 echo hello > /tmp/x/y/z.txt
4 cat /tmp/x/y/z.txt
5 hello
6 chmod u-x /tmp/x
7 cat /tmp/x/y/z.txt
8 cat: /tmp/x/y/z.txt: Permission non accordée
9 chmod u+x /tmp/x
```

# w/dir

```
~> su pat ; ls -ld /tmp/x/y
drwxrwxr-x 2 pl pl 4096 31 juil. 09:50 /tmp/x/y
~> yes | rm /tmp/x/y/z.txt
rm: impossible de supprimer /tmp/x/y/z.txt
```

# w/dir

```
~> su pat ; ls -ld /tmp/x/y
drwxrwxr-x 2 pl pl 4096 31 juil. 09:50 /tmp/x/y
~> yes | rm /tmp/x/y/z.txt
rm: impossible de supprimer /tmp/x/y/z.txt
```

```
~> chmod o+w /tmp/x/y
~> ls -al /tmp/x/y
drwxrwxrwx 2 pl pl 4096 31 juil. 09:58 .
drwxrwxr-x 3 pl pl 4096 31 juil. 09:40 ..
-rw-rw-r-- 1 pl pl 6 31 juil. 09:58 z.txt
```

# w/dir

```
~> su pat ; ls -ld /tmp/x/y
drwxrwxr-x 2 pl pl 4096 31 juil. 09:50 /tmp/x/y
~> yes | rm /tmp/x/y/z.txt
rm: impossible de supprimer /tmp/x/y/z.txt
```

```
~> chmod o+w /tmp/x/y
~> ls -al /tmp/x/y
drwxrwxrwx 2 pl pl 4096 31 juil. 09:58 .
drwxrwxr-x 3 pl pl 4096 31 juil. 09:40 ..
-rw-rw-r-- 1 pl pl 6 31 juil. 09:58 z.txt
```

```
$ echo boom > /tmp/x/y/z.txt
bash: /tmp/x/y/z.txt: Permission non accordée
$ rm -f /tmp/x/y/z.txt
$ ls -a /tmp/x/y (OK)
```

## masque de création

Un *masque* est appliqué par défaut lors de la création des fichiers. Il indique les droits à retirer ( masque ) ou à conserver (logique) :

```
~> umask ; umask -S
0002
u=rwx , g=rwx , o=rx
~> echo test > /tmp/test
~> ls -l test
-rw-rw-r--. 1 pl pl 5 3 sept. 19:45 /tmp/test
~> echo $(( 666 - 666 & 002 ))
664
~> stat --format="%a %A" /tmp/test
664 -rw-rw-r--
```



# masque de création

```
1 #!/bin/bash -v
2 echo 'int main(void) {return 0;}' > vrai.c
3 umask -S
4 u=rwx,g=rwx,o=rx
5 gcc vrai.c
6 ls -l a.out
7 -rwxrwxr-x 1 pl pl 4542 31 juil . 08:25 a.out
8 umask 077
9 gcc vrai.c
10 ls -l a.out
11 -rwx----- 1 pl pl 4542 31 juil. 08:25 a.out
```

## bit collant

A l'origine, le **sticky bit** indique au système de conserver en mémoire le code. Cette terminologie historique est réutilisée pour qualifier les droits sur un répertoire :

```
~> ls -ld /tmp
drwxrwxrwt 13 root root 4096  5 sept. 12:36 /
tmp/

~> stat --format="%F %a %A" /tmp
repertoire 1777 drwxrwxrwt
```

### sticky bit du répertoire allumé

Un utilisateur peut manipuler une arborescence dans ce répertoire : créer et effacer les fichiers qui lui appartiennent.

# sticky bit

```
~> ls -ld /{tmp,TMP}
drwxrwxrwt. 103 root root 12288 ... /tmp
drwxrwxrwx   2 root root  4096 ... /TMP
~> echo hello > /tmp/msg.txt
~> echo hello > /TMP/msg.txt
~> ls -l /{tmp,TMP}/msg.txt
-rw-rw-r-- 1 pl pl 6 ... /tmp/msg.txt
-rw-rw-r-- 1 pl pl 6 ... /TMP/msg.txt
~> su pat
$ rm /TMP/msg.txt
$ rm /tmp/msg.txt
rm : operation non permise
```

## droit suid

Dans certaines circonstances, un utilisateur doit pouvoir exécuter des binaires avec des privilèges **root**. Typiquement, **toto** peut changer son mot de passe avec `passwd` et donc écrire dans `/etc/shadow` ! ?

```
~> ls -l /etc/shadow
-----. 1 root root 1164 ... /etc/shadow
~> ls -l /usr/bin/passwd
-rwsr-xr-x. 1 root root 25992 ... /usr/bin/passwd
~> stat --format="%F %a %A" /usr/bin/passwd
fichier 4755 -rwsr-xr-x
```

- `passwd` est un binaire **suid**
- ouvert en exécution
- fournit la capacité root (euid).

# binaires suid

```
↪ find /usr/bin -perm +4000  
/usr/bin/chsh  
/usr/bin/chage  
/usr/bin/passwd  
/usr/bin/mount  
/usr/bin/crontab  
/usr/bin/sudo  
/usr/bin/umount  
/usr/bin/at  
/usr/bin/su
```

# droit sgid répertoire

Le bit **sgid** sur un fichier exécutable s'applique au groupe : s masque les droits x du groupe, S sinon.

- Dans la cas d'un partage de dossier, tous les membres d'un groupe souhaitent modifier ou supprimer les contributions des membres du groupe.
- Tout fichier et répertoire créé dans un répertoire **sgid** appartient au groupe de celui-ci et non au groupe de l'utilisateur qui crée l'élément.
- Les sous-répertoires héritent du bit **sgid**, sans rétroaction !

```
# groupadd test ; mkdir /home/partage
# chown root:test /home/partage
# usermod -a -G test toto
~> groups
novice test
~> echo hello > /home/partage/toto
~> ls -al /home/partage *o*
-rw-r--r--. 1 toto novice 18 16:52 toto
# chmod g+s /home/partage
~> stat --format="%F %a %A" /home/partage
repertoire 2775 drwxrwsr-x
~> echo hello > /home/partage/world
~> ls -al /home/partage *o*
-rw-r--r--. 1 toto novice 18 16:52 toto
-rw-r--r--. 1 toto test 6 16:53 world
```

## uid, euid

Un processus possède un **uid** réel et un **uid** effectif sur lequel sont basé les privilèges du processus. En général

$$UID = EUID$$

- l'**uid** réel est celui du processus qui lance la commande.
- l'**uid** effectif est (sous condition) celui du propriétaire de la commande.

Les permissions sont basées sur l'**euid**.



## system vs execve

```
1 int main()
2 {
3 char *argv[2];
4 argv[0] = "id";
5 argv[1] = 0;
6
7 system("id");
8
9 execve("/usr/bin/id", argv, 0);
10
11 return 0;
12 }
```

- `system` exécution par le shell.
- `execve` appel système.

## system vs execve

```
sudo gcc -Wall setuidexec.c -o setuidexec.exe
./setuidexec.exe | cut -d ',' -f1,2
uid=501(pl) gid=501(pl) groupes=501(pl),10(wheel)
uid=501(pl) gid=501(pl) groupes=501(pl),10(wheel)
sudo chmod u+s setuidexec.exe
./setuidexec.exe | cut -d ',' -f1,2
uid=501(pl) gid=501(pl) groupes=501(pl),10(wheel)
uid=501(pl) gid=501(pl) euid=0(root) groupes=0(
root),10(wheel)
```

- **system** euid  $\rightsquigarrow$  uid ( **bash** n'est pas **suid** ).
- **execve** euid = root. (appel système)

# eid.c

```
1 #include <unistd.h>
2 #include <stdlib.h>
3 int main( void ) {
4 FILE *src = NULL;
5 system("ls -alt eid.exe");
6 printf ( "\nuid=%d eid=%d\n",getuid (),geteuid ());
7 system("whoami");
8 system("wc -L /etc/shadow");
9                                     // -L : plus longue ligne ?
10 src = fopen("/etc/shadow", "r");
11 printf ( "%p", src );
12 fclose ( src );
13 return 0;
14 }
```

## system, execve

```
~> sudo gcc -Wall euid.c -o euid.exe
~> sudo chmod u+s euid.exe
~> ./euid.exe
-rwsr-xr-x 1 root root 5326 21:37 euid.exe

uid=501 euid=0
pl
wc: /etc/shadow: Permission non accordée
0x859a008
```

- **bash** : perte de **euid**.
- **fopen** : exécuté avec **euid** root.

# uid, euid

```
1 #include <sys/types.h>
2 #include <stdio.h>
3 #include <unistd.h>
4 #include <stdlib.h>
5 int main( void )
6 {
7 FILE *src;
8 printf ( "\nuid=%d euid=%d\n",getuid (),geteuid ());
9 src = fopen( "uid.txt", "r" );
10 if ( ! src )
11     perror ( "interdit " );
12 fclose (src );
13 return 0;
14 }
```

# uid, euid

```
~> gcc -Wall uid.c -o uid.exe
~> sudo chown nobody:nobody uid.exe
~> sudo cp -p uid.exe uid.sss
~> sudo chmod u+s uid.sss
~> touch uid.txt ; ls -l uid.[est]*
-rwxrwxr-x  1 nobody nobody 5237 ... uid.exe
-rwsrwxr-x  1 nobody nobody 5237 ... uid.sss
-rw-w----- 1 pl      pl      0 ... uid.txt
~> id
uid=501(pl) gid=501(pl) groupes=501(pl),10(wheel)
~> ./uid.exe
uid=501 euid=501
~> ./uid.sss
uid=501 euid=99
interdit: Permission denied
```

# setuid.c

```
1 #include <sys/types.h>
2 #include <stdio.h>
3 #include <unistd.h>
4 #include <stdlib.h>
5 int main( void ) {
6     system("ls -alt setuid.exe");
7     printf ( "\nuid=%d euid=%d\n",getuid (),geteuid ());
8     system("whoami");
9     system("ls /root");
10    setuid(0);
11    printf ( "\nuid=%d euid=%d\n",getuid (),geteuid ());
12    system("whoami");
13    system("ls /root");
14    return 0;
15 }
```

# setuid.exe

```
~> sudo gcc -Wall setuid.c -o setuid.exe
~> sudo chmod u+s setuid.exe
./setuid.exe
-rwsr-xr-x 1 root root 5187 ... setuid.exe
uid=501 euid=0
pl
ls: impossible d'ouvrir le repertoire
root: Permission non accordee
uid=0 euid=0
root
anaconda-ks.cfg Desktop install.log
```

`setuid` : le processus utilise son **euid** pour changer **uid** !



# faill

```
1 #include <sys/types.h>
2 #include <stdio.h>
3 #include <unistd.h>
4 #include <stdlib.h>
5
6 int main( int argc, char *argv[] )
7 {
8     char cmd[64];
9
10    sprintf ( cmd, "wc -l %s", argv[1] );
11    setuid(0);
12    system( cmd );
13    return 0;
14 }
```

# exploit

```
~> ./boom.exe /etc/passwd  
35 /etc/passwd
```

# exploit

```
~> ./boom.exe /etc/passwd  
35 /etc/passwd
```

```
~> ./boom.exe /etc/shadow  
35 /etc/shadow
```

# exploit

```
~> ./boom.exe /etc/passwd  
35 /etc/passwd
```

```
~> ./boom.exe /etc/shadow  
35 /etc/shadow
```

```
~> ./boom.exe "/etc/passwd; grep pl /etc/shadow"  
35 /etc/passwd  
pl:$6$uHL8DbWBehKEl0Rb$dZZ/3O2uhDIMkkrnjIG/q8Lo.  
J8  
CZKPqZAc7j7EFFBo6o4QTW/UIWK.YWhjvgKRqquMo.  
aKJEtIG3  
YEYPyg5T.:15439:0:99999:7: : :  
passwd
```

# Injection de code

**faible** L'absence de contrôle sur les données passées au exécutable pose toujours des soucis, dans le cas des *suid* c'est un sérieux problème de sécurité !

# Injection de code

**faible** L'absence de contrôle sur les données passées au exécutable pose toujours des soucis, dans le cas des *suid* c'est un sérieux problème de sécurité!

```
~> ./boom.exe 'boom.c;/bin/sh'
```

# Injection de code

**faible** L'absence de contrôle sur les données passées au exécutable pose toujours des soucis, dans le cas des *suid* c'est un sérieux problème de sécurité!

```
~> ./boom.exe 'boom.c;/bin/sh'
```

```
15 boom.c
sh-4.2#
```

# Injection de code

**faible** L'absence de contrôle sur les données passées au exécutable pose toujours des soucis, dans le cas des *suid* c'est un sérieux problème de sécurité!

```
~> ./boom.exe 'boom.c;/bin/sh'
```

```
15 boom.c  
sh-4.2#
```

```
sh-4.2# whoami  
root
```

- shell root!



# attaque sushi

```
#!/bin/bash
echo $UID $EUID
~> sudo chown root:root suid.sh
~> sudo chmod u+s suid.sh
~> ls -l suid.sh
-rwsrwxr-x 1 root root 28 19 sept. 06:10 suid.sh
~> ./suid.sh
501 501
```

Les possibilités de failles dans un script **suid** seraient encore plus nombreuses : attaques **sushi**. Pour des raisons de sécurité :

- dans un script **shell** : **eid := uid**
- modification de fichier  $\implies$  perte du **suid-bit**

# shellshock

```
1 #include <sys/types.h>
2 #include <stdio.h>
3 #include <unistd.h>
4 #include <stdlib.h>
5 // shellshockable : facile
6 int main( void )
7 {
8 char tmp[128], cmd[128];
9 printf (tmp, "sed -n '/:%d:/s/:.*//p' /etc/passwd", getuid
    () );
10 printf (cmd, "grep $( %s ) /etc/shadow", tmp);
11 puts("cryptogramme:");
12 setuid(0);
13 system( cmd );
14 return 0;
```

# shellshock

SYSTEM(3) Manuel du programmeur Linux

## NOM

system – Executer une commande shell

## SYNOPSIS

```
#include <stdlib.h>
int system(const char *command);
```

## DESCRIPTION

La fonction `system()` exécute la commande indiquée dans `command` en appelant `/bin/sh -c command`, et revient après l'exécution complète de la commande.

## NOTES

N'utilisez **jamais** `system()` dans un programme avec les privilèges `Set-UID` ou `Set-GID`. Des variables d'env. avec des valeurs étranges peuvent être utilisées pour corrompre le système.