Introduction à la sécurité informatique
Travaux pratiques : les droits d'accès dans les systèmes UNIX

Giuseppe Lipari

12 janvier 2021

Table des matières

1 Modalité de travail

Pour faire les exercices et les questions de ce TD/TP, il sera utile de travailler sur une machine virtuelle. L'Université de Lille met à disposition l'outil OpenStack à l'adresse `https://cloud.univ-lille.fr/`, pour créer et gérer des machines virtuelles avec Linux.

Attention : pour y accéder il faut être connecté en VPN.

— Allez sur `https://cloud.univ-lille.fr/` et utilisez votre login/mot de passe de l'Université pour vous connecter ;

— Repérez votre nom en haut à droite, cliquez et selectionnez "help/aide" pour télécharger un pdf avec les instructions pour préparer une machine virtuelle.

  — Il est pratique de se connecter en ssh avec une clé de sécurité (cela évite de taper son mot de passe à chaque connexion).

  — Je vous recommande d'installer au moins le compilateur C, make et cmake
  `sudo apt install g++ make cmake`

2 La gestion des droit d'accès dans Unix

2.1 User ID et Group IDs

Dans les systèmes Unix, chaque utilisateur a un User ID et un ou plusieurs Group IDs. Il est possible de les visualiser avec la commande `id` :

```
ubuntu@isi2:~$ id
uid=1000(ubuntu) gid=1000(ubuntu) groups=1000(ubuntu),4(adm),20(dialout),24(cdrom), ...
```

L'utilisateur `root` peut créer des nouveaux utilisateurs en utilisant la commande `adduser`, et des nouveaux groupes en utilisant `addgroup`. Il est possible de modifier les paramètres d'un utilisateur existant avec `usermod`.

Je vous invite à lire les pages du manuel pour ces commandes (`man adduser`).

Si un utilisateur appartient à plusieurs groupes, son premier groupe dans la liste est considéré comme son groupe principal, et les autres sont considérés comme les groupes *supplémentaires*.

2.2 Permissions de fichiers

Un fichier a :

— un User ID (id du propriétaire du fichier)

— un Groupe ID (id du groupe du fichier)

— des permissions d'accès en lecture, écriture et exécution pour le propriétaire, pour le groupe et pour les autres.

Pour voir ces information concernant un fichier, vous pouvez utiliser la commande `ls - al`.

2.2.1 Exemple

Dans le répertoire home de l'utilisateur `ubuntu`, on retrouve les fichiers suivants :

```
ubuntu@isi2:~$ ls -al
total 40
```

```
drwxr-xr-x 4 ubuntu ubuntu 4096 Jan  4 11:34 .
drwxr-xr-x 3 root   root   4096 Jan  4 11:05 ..
-rw------- 1 ubuntu ubuntu    8 Jan  4 11:08 .bash_history
-rw-r--r-- 1 ubuntu ubuntu  220 Feb 25  2020 .bash_logout
-rw-r--r-- 1 ubuntu ubuntu 3771 Feb 25  2020 .bashrc
drwx------ 2 ubuntu ubuntu 4096 Jan  4 11:08 .cache
-rw-r--r-- 1 ubuntu ubuntu  807 Feb 25  2020 .profile
drwx------ 2 ubuntu ubuntu 4096 Jan  4 11:05 .ssh
-rw-r--r-- 1 ubuntu ubuntu    0 Jan  4 11:34 .sudo_as_admin_successful
-rw------- 1 ubuntu ubuntu  708 Jan  4 11:33 .viminfo
-rw-rw-r-- 1 ubuntu ubuntu   35 Jan  4 11:33 file.txt
```

La deuxième colonne contient l'id du propriétaire du fichier ; la troisième colonne contient l'id du groupe ; la première colonne les permissions d'accès.

Le fichier `.viminfo` a les permissions suivantes :

```
-rw-------
```

Le premier caractère est le type du fichier (`d` pour directory, `-` pour un fichier normal). Les 3 caractères suivants sont les permissions de lecture (`r`), écriture (`w`) et exécution (`x`) pour le propriétaire (`ubuntu`). Le fichier est donc lisible et il peut être écrit par l'utilisateur `ubuntu`. Les trois caractères suivants représentent les mêmes permissions pour les utilisateurs qui appartiennent au groupe `ubuntu` ; et les trois derniers caractères représentent les permissions pour tous les autres utilisateurs. Ainsi, le fichier `.viminfo` n'est accessible qu'à l'utilisateur `ubuntu`.

Le fichier `file.txt` est accessible en lecture par tout le monde ; en écriture par l'utilisateur `ubuntu` et par les utilisateurs qui appartiennent au groupe `ubuntu` ; et n'est accessible en exécution par personne.

Pour changer les droit d'accès d'un fichier, vous pouvez utiliser la commande `chmod`. Pour changer le proprietaire et le groupe d'un fichier vous pouvez utiliser la commande `chown` (voir `man chmod` et `man chown` pour l'utilisation).

2.3 Identifiants de processus

Un processus est un programme en exécution, dont le code se trouve sur un fichier exécutable.

Plusieurs identifiants sont affectés à un processus. Le *Real UID* (RUID) est l'id de l'utilisateur qui a lancé le processus ; le *Real GID* (RGID) est l'id du groupe principal auquel appartient l'utilisateur qui a lancé le processus. L'*Effective UID* (EUID) et l'*Effective GID* (EGID) sont normalement égaux au RUID et RGID, et ils sont utilisés pour vérifier les droit d'accès du processus. Il est possible de modifier les EUID et UGID pour changer le niveau de privilège du processus.

Pour obtenir la valeur de ces identifiants, le processus peut utiliser les appels système `getuid() and getgid()` pour les real ids, et `geteuid() and getegid()` pour les ids effectifs. Pour obtenir la liste des groupes supplémentaires, il faut utiliser `getgroups()`.

Pour vérifier les droits d'accès à un fichier, le système utilise la procédure suivante :

— D'abord, il compare l'EUID du processus avec le propriétaire du fichier ; s'ils correspondent, il utilise le premier triplet de permissions pour vérifier les droits d'accès ;

— S'il ne correspondent pas, il compare la liste des groupes du processus (EGID et groupes supplémentaires) avec le groupe du fichier ; si un des groupes dans la liste correspond, il utilise le deuxième triplet ;

— S'ils ne correspondent pas, il utilise le troisième triplet.

Question 1. Pour cet exercice et les suivants, créez un utilisateur `toto` dans le système, et ajoutez-le au groupe `ubuntu`.

Supposons qu'un processus a été lancé par l'utilisateur `toto`. Le processus essaie d'ouvrir en écriture le fichier `myfile.txt` suivant :

```
-r--rw-r-- 1 toto ubuntu    5 Jan  5 09:35 titi.txt
```

— Dire si le processus peut écrire, et pourquoi.

□

Question 2. Le caractère `x` indique que le fichier est exécutable.

— Que signifie le caractère `x` pour un répertoire ?

Avec l'utilisateur `ubuntu` créez le répertoire `mydir`, et enlevez le droit d'exécution au groupe `ubuntu`. Maintenant, avec l'utilisateur `toto`, essayez d'entrer dans le répertoire avec `cd mydir`.

— Qu'est-ce qu'il se passe ? Pourquoi ?

Avec l'utilisateur `ubuntu`, créez un fichier `data.txt` dans le répertoire `mydir`. Maintenant, avec l'utilisateur `toto` essayez de lister le contenu de la directory avec `ls -al mydir`.

— Qu'est-ce qu'il se passe ? et pourquoi ?

□

2.4 Changer de privilège

Pour un fichier exécutable, il peut être utile de definir son flag `set-user-id` pour donner aux processur des privilèges plus élevés.

Quand ce flag est défini, le programme est lancé avec son EUID égal à l'id du propriétaire du fichier exécutable. Il existe aussi le flag `set-group-id` qui fait la même chose avec l'EGID. Pour changer la valeur de ce flag, il faut utiliser la commande `chmod`.

2.4.1 Exemple

Considérons la situation suivante :

```
toto@isi2:~/exec$ ls -al
total 36
drwxrwxr-x 2 ubuntu ubuntu  4096 Jan  7 11:06 .
drwxr-xr-x 7 ubuntu ubuntu  4096 Jan  7 11:03 ..
-rw--w---- 1 ubuntu ubuntu     0 Jan  7 10:42 file_a.txt
-rw-rw-r-- 1 ubuntu ubuntu     0 Jan  7 11:06 file_b.txt
-rwxrwxr-x 1 ubuntu ubuntu 16872 Jan  7 11:05 myopen
-rw-rw-r-- 1 ubuntu ubuntu   361 Jan  7 11:04 myopen.c
```

Le code du programme `myopen.c` :

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

```c
int main(int argc, char *argv[])
{
  FILE *f;

  if (argc < 2) {
    printf("Missing argument\n");
    exit(EXIT_FAILURE);
  }
  printf("Hello world\n");
  f = fopen(argv[1], "r");
  if (f == NULL) {
    perror("Cannot open file");
    exit(EXIT_FAILURE);
  }
  printf("File opens correctly\n");
  fclose(f);
  exit(EXIT_SUCCESS);
}
```

Le programme `myopen` prendre en paramètre un nom de fichier et il essaie de l'ouvrir en lecture. Si l'utilisateur `toto` lance la commande :

```
toto@isi2:~/exec$ ./myopen file_a.txt
Hello world
Cannot open file: Permission denied
```

le programme n'arrive pas à ouvrir le fichier en lecture parce que `file_a.txt` est lisible seulement pas l'utilisateur `ubuntu`.

L'utilisateur `ubuntu` peut definir le set-user-id pour le fichier `myopen` comme ça :

```
ubuntu@isi2:~/exec$ chmod u+s myopen
ubuntu@isi2:~/exec$ ls -al myopen
-rwsrwxr-x 1 ubuntu ubuntu 16872 Jan  7 11:05 myopen
```

il faut remarquer le caractère `s` dans le premier triplet qui indique la présence d'un permission exécution avec set-user-id.

Maintenant, si `toto` lance la même commande :

```
toto@isi2:/home/ubuntu/exec$ ./myopen file_a.txt
Hello world
File opens correctly
```

2.4.2 Questions

Question 3. Écrire un programme en C qui imprime la valeur des ses ids (EUID, EGID, RUID, RGID) et le contenu du fichier `mydir/mydata.txt` (que vous avez crée à la question précédente).

Le fichier exécutable doit appartenir à l'utilisateur `ubuntu` et au groupe `ubuntu`, et il est lancé par l'utilisateur `toto`.

— Quel sont les valeurs de différents ids ? Est-ce que le processus arrive à ouvrir le fichier `mydir/mydata.txt` en lecture ?

Maintenant, activez le flag `set-user-id` du fichier exécutable, et relancez le programme.

— Quel sont les valeurs de différents ids ? Est-ce que le processus arrive à ouvrir le fichier `mydir/mydata.txt` en lecture ?

□

Question 4. Écrivez un script python qui imprime les valeurs des EUID et EGID. Le script doit appartenir à l'utilisateur `ubuntu`.

Activez le `set-user-id` et lancez le script avec l'utilisateur `toto`.

— Quel sont les valeurs de différents ids ?

□

Les *Saved set-user-id* et *saved set-group-id* sont utilisés pour sauvegarder les effective ids avant de le changer, pour pouvoir plus tard les récupérer.

Pour obtenir les valeurs de tous les id, vous pouvez utiliser les syscall :

```
int getresuid(uid_t *ruid, uid_t *euid, uid_t *suid);
int getresgid(gid_t *rgid, gid_t *egid, gid_t *sgid);
```

et pour les modifier :

```
int setresuid(uid_t ruid, uid_t euid, uid_t suid);
int setresgid(gid_t rgid, gid_t egid, gid_t sgid);
```

Un processus non privilégié ne peut pas affecter une valeur arbitraire aux effective ids.

Quel est l'utilité du flag `set-user-id` ? Ce flag permet à un utilisateur de lancer un programme avec de privilèges supérieures au siens. Cette fonctionnalité peut être utile pour accéder à des fichiers auxquels normalement l'utilisateur n'a pas le droit d'accéder.

Par exemple, considérons le fichier `/etc/passwd` :

```
-rw-r--r-- 1 root root 1802 Jan  4 18:04 /etc/passwd
```

Il contient les paramètres de tous les comptes utilisateurs du système, et il est donc lisible par tous le monde. Par contre, seul l'administrateur (`root`) a le droit de le modifier.

Comment un utilisateur peut changer un de ses attributs sans demander à l'administrateur ?

Question 5. Visualisez le contenu du fichier `/etc/passwd`.

— À quoi sert la commande `chfn` ? Donnez les résultat de `ls -al /usr/bin/chfn`, et expliquez les permissions.

— Lancez la commande `chfn` en tant que utilisateur `toto`, répondez au questions. Visualisez à nouveau le contenu du fichier `/etc/passwd` et vérifiez que les informations ont été mis à jour correctement.

□

Question 6. En effet, le fichier `/etc/passwd` ne contient aucun mot de passe.

— Ou ils sont gardez les mots de passe des utilisateurs ? Pourquoi ?

□

2.5 Règles pour `access`

La commande `access` vérifie les droit d'accès à un fichier, mais elle utilise le RUID et RGID au lieu du EUID et EGID.

De `man access` :

> The check is done using the calling process's real UID and GID, rather than the effective IDs as is done when actually attempting an operation (e.g., open(2)) on the file. Similarly, for the root user, the check uses the set of permitted capabilities rather than the set of effective capabilities ; and for non-root users, the check uses an empty set of capabilities.
>
> This allows set-user-ID programs and capability-endowed programs to easily determine the invoking user's authority. In other words, access() does not answer the "can I read/write/execute this file ?" question. It answers a slightly different question : "(assuming I'm a setuid binary) can the user who invoked me read/write/execute this file ?", which gives set-user-ID programs the possibility to prevent malicious users from causing them to read files which users shouldn't be able to read.

2.6 Règles spéciales pour les directories

2.6.1 Setgid pour directories

Si on utilise le `setgid` sur une directory, les fichiers qui seront créé dans la directory auront comme group ID ce de la directory.

Par exemple, supposons que la directory mydir a les permissions suivantes :

```
drwxrwxr-x 2 ubuntu ubuntu 4096 Jan  4 18:02 mydir
```

Supposons que l'utilisateur `toto` est dans le groupe `ubuntu`.

Si toto crée un fichier dans mydir, il aura comme permissions :

```
toto@isi2:/home/ubuntu$ touch mydir/tata.txt
toto@isi2:/home/ubuntu$ ls -al mydir/tata.txt
-rw-rw-r-- 1 toto toto 0 Jan  4 18:06 mydir/tata.txt
```

Maintenant, on ajoute un setgid à la directory `mydir` :

```
ubuntu@isi2:/home/ubuntu$ chmod g+s mydir/
ubuntu@isi2:~$ ls -al mydir/
...
drwxrwsr-x 2 ubuntu ubuntu 4096 Jan  4 18:06 mydir
```

À la place du caractère `x` on voit un `s` pour le triplet correspondant au groupe.

Si `toto` crée un fichier dans la directory :

```
toto@isi2:/home/ubuntu$ touch mydir/titi.txt
toto@isi2:/home/ubuntu$ ls -al mydir/
total 8
drwxrwsr-x 2 ubuntu ubuntu 4096 Jan  4 18:11 .
drwxr-xr-x 5 ubuntu ubuntu 4096 Jan  4 18:02 ..
-rw-rw-r-- 1 toto   toto      0 Jan  4 18:07 tata.txt
-rw-rw-r-- 1 toto   ubuntu    0 Jan  4 18:11 titi.txt
```

Le nouveau fichier a comme groupe `ubuntu`.

2.6.2 Sticky bit

Le *sticky-bit* est un flag qu'on peut affecter à une directory. Si le flag est activé pour une directory `mydir`, un fichier qui se trouve dans l'arborescence qui a comme racine `mydir` peut être renommé ou effacé que par le proprietaire du répertoire ou par le propriétaire du fichier.

Pour affecter le sticky bit, on utilise la commande `chmod` en spécifiant le caractère `t` :

```
ubuntu@isi2:/home/ubuntu$ chmod +t mydir
ubuntu@isi2:~$ ls -l
total 12
drwxrwSr-t 2 ubuntu ubuntu 4096 Jan  5 09:36 mydir
```

3 Exercices

3.1 Serveur de fichiers partagés

On voudrait mettre en place un serveur où des utilisateurs peuvent partager des fichiers, avec des restrictions selon le groupe d'appartenance.

On a deux groupes d'utilisateurs, le `groupe_a` et le `groupe_b`. On a a un utilisateur spécial qu'on appel *administrateur* : `admin`.

Chaque groupe a une directory partagée à tous les membres du groupe, mais qu'il n'est pas accessible au membres de l'autre groupe : une directory `dir_a` et une directory `dir_b`. Il y a une directory `dir_c` qui est partagé par les utilisateurs de deux groupes.

Les membres du `groupe_a` :

— Il peuvent lire tous les fichiers et sous-directories contenus dans `dir_a` et `dir_c` ;
— Ils peuvent lire, mais ils ne peuvent pas modifier les fichiers dans `dir_c`, ni les renommer, ni les effacer, ni créer des nouveaux fichiers.
— Il peuvent modifier tous les fichiers contenus dans l'arborescence à partir de `dir_a`, et il peuvent créer de nouveaux fichiers et directories dans `dir_a` ;
— Ils n'ont pas le droit d'effacer, ni de renommer, des fichiers dans `dir_a` qui n'appartient pas à eux ;
— Ils ne peuvent pas ni lire, ni modifier, ni effacer les fichiers dans `dir_b`, et ils ne peuvent pas créer des nouveaux fichiers dans `dir_b`.

Des règles symétriques sont valides pour les membres du `groupe_b`.

L'utilisateur `admin` suivi les même règles que les autres membres des groupes, exception faite pour les règles suivantes :

— Il peut modifier des fichiers où en créer des nouveaux dans `dir_c` ;
— il peut effacer (où renommer) des fichiers dans `dir_a`, `dir_b` et `dir_c`.

Question 7. Mettre en place la structure décrite ci-dessus. En particulier, il faut créer au moins :
— les utilisateurs `lambda_a` qui appartient au `groupe_a` et `lambda_b` qui appartient au `groupe_b` ;
— l'utilisateur `admin` ;
— créer les directories `dir_a`, `dir_b` et `dir_c`, et des fichiers avec les permissions correctes.

Il est possible (nécessaire ?) de créer d'autres groupes et utilisateurs.

Validez les règles d'accessibilité avec des script bash, un script pour les utilisateurs lambda de chaque groupe, et un script pour les admin. □

Maintenant, on veut donner la possibilité aux utilisateur du `groupe_a` (ou du `groupe_b`) de pouvoir effacer des fichiers dans la directory `dir_a` (`dir_b`, respectivement) s'il sont dans le `groupe_a` (`groupe_b`, respectivement). Pour faire ça, on met en place un mécanisme d'authentification avec mot de passe.

Question 8. Écrire un programme `rmg` qui prend en paramètre un nom de fichier à effacer. Le programme doit d'abord demander à l'utilisateur un mot de passe. Chaque utilisateur a son propre mot de passe (qui est différent de son mot de passe pour se logger dans le système). Ces mots de passe sont gardés dans un fichier `passwd`, qui doit pouvoir être lu et écrit seulement par `admin`. Le fichier se trouve dans `/home/admin/passwd`.

Le programme `rmg` :

— il doit être exécutable pour tous les utilisateurs du système ;

— avant de demander le mot de passe, il vérifie que l'utilisateur qui l'a lancé appartient au même groupe que le fichier (`groupe_a` ou `groupe_b` respectivement), en cas contraire, il affiche un message d'erreur ;

— il doit avoir le privilège minimal nécessaire à effacer un fichier dans `dir_a` ou dans `dir_b` ;

Écrire le programme. Valider son fonctionnement avec un script bash.

Remarque : Il est recommandé de séparer l'implémentation dans plusieurs modules. En particulier, on vous recommande de mettre les déclarations des fonctions pour vérifier le mot de passe dans un fichier `check_pass.h`, et leur implémentation dans `check_pass.c`. □

Question 9. On voudrait empêcher l'éviter `admin` de connaître les mots de passe des autres utilisateurs. Pour faire ça, on permet aux utilisateurs d'établir son propre mot de passe avec une programme `pwg`.

Le programme est utilisable par tous les utilisateurs du `groupe_a` ou du `groupe_b`. Si l'utilisateur a déjà un mot de passe dans le fichier `passwd`, il demande l'ancien mot de passe avant de pouvoir le modifier. Si la vérification est correct, ou si l'utilisateur n'a pas déjà un mot de passe, il demande le nouveau mot de passe et il le mémorise dans le fichier `passwd` de manière crypté.

L'algorithme pour crypter le mot de passe, utilisez la fonction `crypt()` disponible dans la libc [1]. Plus d'informations sont disponibles ici :

`https://ftp.gnu.org/old-gnu/Manuals/glibc-2.2.3/html_chapter/libc_32.html`

— Écrire le programme `pwg` ;

— Modifier le programme `rmg` pour prendre en compte cette modification (noter que si l'implémentation a été séparé dans des modules, il suffit de modifier les fichiers `check_pass.h` et `check_pass.c`). ;

— Valider le fonctionnement des deux programmes avec des script bash.

□

3.2 Client/Serveur

On donne la possibilité aux utilisateurs du `groupe_a` et du `groupe_b`, et à l'`admin`, d'accéder aux fichiers à distance.

Un programme `group_server` se met en écoute sur le port `4000` attendant les connexions des clients. Le programme s'exècute avec de privilèges de `root`.

Quand un client se connecte, il envoie son nom d'utilisateur et son mot de passe. Le serveur verifie la couple utilisateur / mot de passe dans le fichier `/home/admin/passwd`, et si elle est correcte, il fait un

1. Même s'il ne pas consideré comme un algorithme très sur, on utilisera cette solution pour simplicité.

`fork()` pour lancer un processus fils qui va continuer l'interaction avec le client. Le processus père se remet en écoute d'un prochaine connexion.

Le processus fils, avant d'intéragir avec le client, *incarne* l'utilisateur qui vient de se connecter en changeant son EUID et EGID.

Le client peut demander de :

— lister le contenu d'un répertoire parmi `dir_a`, `dir_b` et `dir_c` avec la commande :
`list <dirname>`
le serveur répond avec la sortie de la commande `ls <dirname>` (la liste, ou un message d'erreur).

— lire le contenu d'un fichier contenu dans une des répertoires, avec la commande :
`read <filename>`
ou `filename` est le chemin relatif à la directory `/home/admin` (par exemple, la commande `read dir_a/file_a1.txt` li le fichier `/home/admin/dir_a/file_a1.txt`. Le serveur répond avec la sortie de la commande `cat <filename>` (le contenu du fichier ou un message d'erreur).

— terminer la connexion avec la commande
`close`

Pour simplifier l'exercice, on considère que tous les fichiers contiennent du texte en format ASCII.

Question 10.

1. Écrire le programme serveur `group_server`.
2. Écrire le programme client `group_client <command_file>` qui prend en paramètre le nome d'un fichier texte qui contient :
   — le nom d'utilisateur et le mot de passe, séparé par un espace, sur la première ligne ;
   — sur chaque ligne suivante, une commande à envoyer au serveur.
   Le client, après connexion au serveur, envoie les commandes contenues dans `command_file` une par une, attends la réponse, et imprime les résultats sur le standard output.
3. Tester le bon fonctionnement de la couple client/serveur en lançant le client avec des fichiers différents comme paramètre d'entrée. En particulier, il faut tester que :
   — Le serveur ne se plante pas ;
   — Toutes permissions sont correctement mis en place : l'utilisateur `lambda_a` n'a pas le droit de lire un fichier dans la directory `dir_b`, etc.

□

4 Instructions pour le rendu

Forkez le dépôt et **ajoutez votre intervenant comme membre développeur**.

Répondez aux questions 1-10 dans le fichier `rendu.md`. Le code va dans le sous-répertoire correspondant à la question `questionX`.

Pour les programmes en C, pensez à fournir un fichier `Makefile` pour compiler et tester votre code. On vous rappelle que le dépôt doit contenir seulement les sources (`.c` et `.h`), des `makefile`, des scripts bash et/ou python. Il ne doit pas contenir ni des fichiers objet (`.o`) ni des fichiers exécutables.

N'oubliez pas de faire `git push` !