TP3 Sécurité
IP Tables et `Setuid`

Wadii Hajji

1 IP Tables

1.1 Introduction aux pare-feux et vérification de l'installation

La commande suivante liste toutes les chaînes. Par défaut, la commande liste les chaînes de la table filter.

```
$ sudo iptables -L
Chain INPUT (policy ACCEPT)
target     prot opt source               destination

Chain FORWARD (policy ACCEPT)
target     prot opt source               destination

Chain OUTPUT (policy ACCEPT)
target     prot opt source               destination
```

Pour lister les chaînes de la table `nat` par exemple, il faut utiliser la commande suivante : `sudo iptables -t nat -n -L`.

— `-t` : permet d'indiquer la table
— `-n` : permet d'éviter les recherches de DNS inversées
— `-L` : liste les chaînes

1.2 Filtrage des ports

Question 1 La commande suivante liste les règles de la table `filter` (par défaut) et affiche le nombre de paquets ayant traversé ces règles ainsi que le nombre d'octets correspondant.

```
$ sudo iptables -L -v
Chain INPUT (policy ACCEPT 2878 packets, 2503K bytes)
pkts bytes target     prot opt in     out     source               destination

Chain FORWARD (policy ACCEPT 0 packets, 0 bytes)
pkts bytes target     prot opt in     out     source               destination

Chain OUTPUT (policy ACCEPT 2624 packets, 308K bytes)
pkts bytes target     prot opt in     out     source               destination
```

Question 2 On utilise la commande suivante : `$ iptables -A INPUT -p tcp -dport 22 -j DROP` afin d'empêcher une personnde externe de se connecter à notre machine en `SSH` sur le port 22.

— `-A INPUT` : ajoute en bas de la liste des règles de la chaîne INPUT.
— `-p tcp` : pour les paquets qui utilisent les protocoles TCP
— `-dport 22` : pour les paquets qui sont à destination du port 22.
— `-j DROP` : refuser le paquet.

En effet, on voit bien que la machine de mon voisin met un certain temps à lui annoncer que la connexion est impossible.

Question 3 La commande suivante va *ajouter* la règle donnant la possibilité à mon voisin (seulement lui) de se connecter à ma machine. Cette règle va être ajouter en première position car nous avons spécifié *1* après `INPUT`.

```
$ iptables -I INPUT 1 -p tcp --dport 22 -s 192.168.1.112 -j ACCEPT
$ sudo iptables -L -v --line-numbers
Chain INPUT (policy ACCEPT 25 packets, 2808 bytes)
pkts bytes target     prot opt in     out     source               destination
    0     0 ACCEPT     tcp  --  any    any     192.168.1.112        anywhere         tcp dpt:ssh
    0     0 DROP       tcp  --  any    any     anywhere             anywhere         tcp dpt:ssh
```

Question 4 La connexion marche à nouveau depuis le poste de mon voisin.

Question 5 Impossible de se connecter à ma machine depuis une autre machine.

Question 6 Soit la chaîne `INPUT` de la table IP suivante :

```
Chain INPUT (policy ACCEPT 21 packets, 1462 bytes)
num   pkts bytes target     prot opt in     out     source         ...
1        2   120 REJECT     tcp  --  any    any     10.0.2.15      ...
2        0     0 REJECT     tcp  --  any    any     192.168.1.2    ...
```

On tape la commande suivante : `$ sudo iptables -Z` et on voit que tous les compteurs ont été remis à 0.

```
Chain INPUT (policy ACCEPT 0 packets, 0 bytes)
pkts bytes target     prot opt in     out     source               ...
  2   120 REJECT     tcp  --  any    any     10.0.2.15            ...
  0     0 REJECT     tcp  --  any    any     192.168.1.2          ...

Chain FORWARD (policy ACCEPT 0 packets, 0 bytes)
pkts bytes target     prot opt in     out     source               ...

Chain OUTPUT (policy ACCEPT 0 packets, 0 bytes)
pkts bytes target     prot opt in     out     source               ...
```

Enfin, `$ iptables_save -c > regles` produit le fichier suivant :

```
# Generated by iptables-save v1.6.0 on Wed Jan  9 09:58:51 2019
*mangle
:PREROUTING ACCEPT [6825:9706597]
:INPUT ACCEPT [6825:9706597]
:FORWARD ACCEPT [0:0]
:OUTPUT ACCEPT [6730:802847]
:POSTROUTING ACCEPT [6732:802993]
COMMIT
# Completed on Wed Jan  9 09:58:51 2019
# Generated by iptables-save v1.6.0 on Wed Jan  9 09:58:51 2019
*nat
:PREROUTING ACCEPT [135:14538]
:INPUT ACCEPT [135:14538]
:OUTPUT ACCEPT [1365:91949]
:POSTROUTING ACCEPT [1365:91949]
COMMIT
# Completed on Wed Jan  9 09:58:51 2019
...
```

Ce fichier contient l'historique des opérations que l'on a effectué sur la table IP. On peut facilement restauré ces opérations avec la commande `$ iptables_restore < regles`.

Question 7 La commande suivante va supprimer la deuxième règle de la chaîne `INPUT`.

```
$ sudo iptables -D 2 INPUT --line-numbers
$ sudo iptables -L -v --line-numbers
Chain INPUT (policy ACCEPT 6 packets, 420 bytes)
num   pkts bytes target     prot opt in     out     source               destination
1        0     0 ACCEPT     tcp  --  any    any     192.168.1.2          anywhere        tcp dpt:ssh

Chain FORWARD (policy ACCEPT 0 packets, 0 bytes)
num   pkts bytes target     prot opt in     out     source               destination

Chain OUTPUT (policy ACCEPT 6 packets, 420 bytes)
num   pkts bytes target     prot opt in     out     source               destination
```

La commande suivante permet d'afficher le numéro de chaque règle dans chaque liste de règles.

```
$ sudo iptables -L -v --line-numbers
Chain INPUT (policy ACCEPT 6 packets, 420 bytes)
num   pkts bytes target     prot opt in     out     source               destination
1        0     0 ACCEPT     tcp  --  any    any     192.168.1.2          anywhere        tcp dpt:ssh
2        0     0 DROP       tcp  --  any    any     192.168.1.3          anywhere        tcp dpt:ssh

Chain FORWARD (policy ACCEPT 0 packets, 0 bytes)
num   pkts bytes target     prot opt in     out     source               destination

Chain OUTPUT (policy ACCEPT 6 packets, 420 bytes)
num   pkts bytes target     prot opt in     out     source               destination
```

2 Setuid

Task 1 : Manipulating environment variables

- `$ env | grep PWD` et `printenv PWD` sont équivalents : ils affichent les variables d'environnement `PWD`.
- `export` et `unset` sont des directives `bash` qui permettent respectivement de changer une variable d'environnement et de supprimer la définition d'une variable d'environnement particulière.

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

Step 1 On compile le programme proposé : `$ gcc setuid.c`. On *output* la sortie dans un fichier texte : `$ ./setuid > child`.

Step 2 On décommente la ligne concernant le processus parent et on commente la ligne concernant le processus fils. On exécute ensuite un `diff parent child` et on remarque que les deux fichiers sont identiques. Le processus parent et le processus fils ont les mêmes variables d'environnement. Les variables d'environnement ont donc été passés par le processus parent au processus fils à l'exécution du programme.

Task 3 : Environement Variables and `execve`

Step 1 On compile le programme proposé. Celui-ci ne produit rien pour le moment car le programme `/usr/bin/env` attend un paramètre.

Step 2 On compile le programme proposé ajoutant cette fois `execve ("/usr/bin/env", argv, environ)`. Celui-ci va afficher les variables d'environnement du processus. Le programme affiche ce qui suit à l'exécution.

```
XDG_VTNR=7
ORBIT_SOCKETDIR=/tmp/orbit-seed
XDG_SESSION_ID=c1
XDG_GREETER_DATA_DIR=/var/lib/lightdm-data/seed
IBUS_DISABLE_SNOOPER=1
TERMINATOR_UUID=urn:uuid:9cc930f2-6796-416b-b63a-d508c49cbe8f
CLUTTER_IM_MODULE=xim
```

```
SESSION=ubuntu
GIO_LAUNCHED_DESKTOP_FILE_PID=3114
ANDROID_HOME=/home/seed/android/android-sdk-linux
GPG_AGENT_INFO=/home/seed/.gnupg/S.gpg-agent:0:1
TERM=xterm
SHELL=/bin/bash
USER=seed
...
_=./printEnvVar
```

Step 3 La variable `char **environ` est une variable externe qui pointe vers un tableau de pointeurs vers des chaînes de caractères qui représentent les variables d'environnement. Cette variable est fournie par l'entête `<unistd.h>`. Le troisième paramètre de `execve` est un tableau de chaînes de caractères (de la forme *clé=valeur*). En l'occurence, nous passons les variables d'environnement du processus actuel au processus fils à ce moment-là sous forme de paramètre.

Task 4 : Environement Variables and `system()`

Soit le programme proposé `system.c` qui est compilé en `system`. On lance tout d'abord le programme proposé et on sauvegarde l'output du programme dans un fichier `environ.txt`. Ensuite, on lance `env ./system > myenv.txt` : dans `myenv` se trouve les variables d'environnement du processus appellant et dans `environ` se trouve les variables d'environnement du nouveau processus.

On lance `$ diff myenv environ` :

```
< _=/usr/bin/env
---
>_=./system
```

Les deux fichiers sont identiques.

Task 5 : Environement Variable and `Setuid` Programs

Dans cette tâche, nous allons créer un programme `Setuid` qui va output les variables d'environnement du processus courant. On change l'accés à ce fichier, le fichier appartient au `root` à présent. À présent, nous allons modifier les variables d'environnement en temps qu'utilisateur à l'aide des commandes suivantes.

```
$ export PATH=$PATH:/home/seed/Downloads/setuid
$ export LD_LIBRARY_PATH=/home/seed/Desktop:$LD_LIBRARY_PATH
$ export ANY_NAME=foo
```

On lance à présent le programme `Setuid`. Les variables d'environnement que nous avons exporté précédemment se trouve toutes au niveau du processus fils mis à part la variable `LD_LIBRARY_PATH`. Cette variable correspond aux dossiers que `gcc` va chercher pour trouver les loaders/linkers. Elle n'est pas transmise aux processus fils.

Task 6 : The PATH Environment Variable and `Setuid` Programs

Soit le programme `foo.c` suivant :

```
1 #include <stdio.h>
2 #include <stdlib.h>
3
4 int main()
5 {
6   printf("%s\n", "Hello world");
7   return 0;
8 }
```

On compile ce programme et on l'appelle `ls`, on exporte le dossier courant dans le `PATH`. On lance ensuite le programme `myls` fourni par l'énoncé et on obtient :

```
$ ./myls
Hello world
```

On peut donc lancer notre propre code à partir du programme `Setuid`. En effet `myls.c` utilise un chemin relatif vers la commande `ls`. Ce chemin doit se trouver dans le `PATH` car la fonction `system` utilise cette variable pour localiser `ls` et autre. Si notre chemin d'accès est dans le `PATH` alors `system` va appeler notre `ls`.

Task 7 : The `LD_PRELOAD` Environment Variable and `Setuid` Programs

1. `myprog` normal, utilisateur normal :

   ```
   $ ./myprog
   I am not sleeping!
   ```

   Le programme affiche la chaîne de caractère retourné par `void sleep()`. La fonction `sleep` que nous avons écrit a pris le dessus sur la fonction `sleep` de la libraire standard.

2. `myprog Setuid`, utilisateur normal :

   ```

   ```

   Rien ne s'affiche après une seconde de latence, le programme fait appel à la fonction `sleep` de la librairie standard.

3. `myprog Setuid`, utilisateur `root` : on a exporté à nouveau `LD_PRELOAD`, cette fois en `root`

   ```
   $ su
   $ export LD_PRELOAD=./libmylib.so.1.0.1
   ```

   ```
   $ ./myprog
   I am not sleeping!
   ```

   La fonction `sleep` que nous avons écrit est appelée ici.

4. On crée cette fois un nouvel utilisateur `user1`. On se connecte en tant que `user1` et on exporte à nouveau la variable `LD_PRELOAD`, puis on lance le programme à partir d'un compte différent de `root` et de `user1`.

   ```
   $ sudo su
   root# useradd -d /usr/user1 -m user1
   root# chown user1 myprog
   root# chmod 4755 myprog
   root# exit
   exit
   $ export LD_PRELOAD=./libmylib.so.1.0.1
   $ ./myprog
   I am not sleeping!
   ```

   La fonction `sleep` a été remplacé par notre fonction `sleep`.

On peut déduire de ces situations que `LD_PRELOAD` est seulement utilisé lorsque l'utilisateur lance un programme que lui même a crée.

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

Step 1 On compile le programme proposé et on l'élève au rang de programme `Setuid root`. Ce programme est censé permettre à Bob de lire un fichier du système à l'aide de `/bin/cat`. Cependant, il peut faire ce que bon lui semble car le programme prend l'entrée de Bob et appelle la fonction `system()` avec l'entrée de ce dernier en paramètre. Il est alors possible de causer des dégâts.

`foo` est un fichier arbitraire et `mycat` le programme de Vince :

```
$ ./mycat "foo; rm -v foo"
Hello world!
removed 'foo'
# ou encore
$ ./mycat "foo; echo 'Je peux écrire' > foo"
```

Le caractère *;* permet de concaténer des instructions `bash`.

Step 2 `system()` invoque un `shell` qui va parcourir la chaîne de caractère donnée en entrée tandis que `execve()` remplace simplement le processus en cours par le programme spécifié et lui passe les paramètres indiqués. C'est pourquoi l'attaque précédente ne fonctionnera pas. En effet :

```
$ ./mycat "foo; rm -v foo"
/bin/cat: 'foo; rm -v foo': No such file or directory
```

Le programme `mycat` cherche un fichier qui s'appelle `'foo; rm -v foo'`. Il ne le trouve pas.

Task 9 : Capability Leaking

Ce programme illustre le concept de *capability leaking*. Dans certains cas, des programmes priviliégiés réduisent leurs privilèges au cours de leur exécution. Cependant, il arrive que cela ne soit pas fait correctement. Cela conduit à la *capability leaking*.

Dans l'exemple donné, le programme va réinitialiser le `uid` à celui du processus parent à la ligne suivante :

```
setuid(getuid());
```

et ferme bien le fichier après avoir réduit les privilèges, dans le processus parent et le processus fils :

```
close(fd);
```

Cependant le problème réside ici dans le fait que la fonction `setuid(getuid())` est utilisé avant que le fichier `/etc/zzz` ne soit ouvert. Il est possible de déplacer l'appel à `setuid()` avant l'ouverture de ce fichier afin de résoudre le problème.

```
void main()
{
    int fd;
    setuid(getuid()); // ici
    ...
}
```