IVC-Chroot-CGroup : Techniques d'isolation avec Chroot, CGroup et Namespaces

Lom M. HILLAH

Table des matières

1 Introduction

Aide aux travaux pratiques sur les techniques d'isolation avec chroot, cgroup et namespaces.

1.1 Pré-requis

- Virtualbox
- Vagrant
- Fedora 33

1.2 Installation

Créez un répertoire pour travailler. Vous pourriez le nommer `devenv-chroot_cgroup`. Initialiser un Vagrantfile dans ce répertoire puis choisissez une box fedora 33 :

```
config.vm.box = "generic/fedora33"
```

Créez un `boostrap.sh` pour le provider Shell en y indiquant quelques packages à installer et dont on aura besoin :

```
#! /usr/bin/env bash

sudo dnf -y check-update
sudo dnf -y upgrade
sudo dnf -y install strace.x86_64
sudo dnf -y install busybox.x86_64
sudo dnf -y install tree.x86_64
```

Le gestionnaire de package sous Fedora est DNF : https://fedoraproject.org/wiki/DNF

Une fois les installations effectuées, ouvrez une session en vous connectant à la VM :

```
$> vagrant ssh
```

Puis passez root :

```
$> sudo su
```

1.3 Rappels

— Pour déterminer le chemin vers l'exécutable d'un utilitaire (s'il est référencé dans votre PATH), exemple avec `ls` :

```
$> which ls
```

— Pour visualiser une arborescence de fichiers (exemple pour le répertoire courant) :

```
$> tree .
```

2 Chroot de la commande ls

Cette section indique les étapes à effectuer pour réaliser un chroot de la commande ls.

2.1 Utilisation de ldd sur ls

ldd est un utilitaire qui liste les bibliothèques dont dépend un exécutable ou une bibliothèque partagée. C'est un utilitaire qui a peu d'options.

```
$> ldd /usr/bin/ls
```

Pour connaître les options possibles de ldd :

```
$> ldd  --help
```

Essayez ldd avec l'option -v.

2.2 Création d'un jail contenant ls et ses dépendances

En tant que root, créez un répertoire `jail`. Vous allez d'abord y dupliquer les chemins système vers ls et ses dépendances. Pour faciliter ce travail, écrivez un script Shell auquel vous passerez le chemin vers l'exécutable de la commande qui vous intéresse, ainsi que le répertoire `jail`.

```
$> vim copy_command_dependencies.sh
```

Vous pouvez utiliser le contenu suivant pour le script :

```
#! /usr/bin/env bash

E_ERROR=1
E_SUCCESS=0

if [ "$#" -ne 2 ] || ! [ -d "$2" ]
then
 echo "Usage: $0 PATH_TO_EXE DIRECTORY" >&2
 exit "$E_ERROR"
fi

jail=$2
echo "Commdand to copy: $1"
cp -f -v --parents "$1" "$jail"/

list="$(ldd $1 | egrep -o '/lib.*\.[0-9]')"
echo "Dependencies to copy: $list"
for i in $list
do
        cp -f -v --parents "$i" "$jail"/
done

echo "done"
exit "$E_SUCCESS"
```

Donnez le droit d'exécution à votre script avec la command `chmod` puis lancez-le avec les deux arguments suivants :

```
$> ./copy_command_dependencies.sh /usr/bin/ls jail
```

L'arborescence de ls ainsi que celle de ses dépendances seront dupliquées dans jail. Vérifiez :

```
$> tree jail
```

Lancez ensuite la commande chroot sur ls :

```
$> chroot jail /usr/bin/ls
```

ou

```
$> chroot jail ls
```

Vérifiez que sans ls ou ses dépendances dans jail, ls ne fonctionnera pas :

```
$> rm -rf jail/usr
$> chroot jail ls
```

ou

```
$> rm -rf jail/lib64
$> chroot jail ls
```

Vérifier ensuite que ls est bien **emprisonné** dans jail et ne peut donc pas lister les arborescences situées en dehors :

```
$> chroot jail ls /home/vagrant
```

2.3 Strace

Le programme `strace`[1] permet d'intercepter et de lister tous les appels système et signaux d'un processus. C'est un outil utile pour le débogage de programmes. Pour lister seulement les appels qui échouent, utiliser l'option -Z.

```
$> strace chroot jail ls
```

```
$> strace -Z chroot jail ls /home/vagrant
```

```
$> chroot jail $(strace ls)
```

Strace utilisé comme ci-dessus est celui que vous avez installé dans votre système. Pour tracer les appels de ls encapsulés dans le jail, il faut aussi emprisonner strace dans le jail, comme pour ls ci-dessus en utilisant le script ad hoc. Vouz pourrez ensuite tracer plus simplement les appels de ls dans le jail :

```
$> chroot jail strace -Z ls
```

Vous verrez alors plus clairement les autres fichiers (bibliothèques, locales) auxquels la commande ls n'arrive pas à accéder. Dupliquez ces fichiers et leurs arborescences dans jail.

3 Chroot de bash, busybox puis vim

Busybox[2] est un outil qui rassemble plusieurs utilitaires en un seul exécutable. C'est une sorte de couteau suisse qui se comporte comme tous les utilitaires qu'il agrège, même s'il présente moins d'options que chaque utilitaire original qu'il imite. Il a été conçu comme utilitaire pour les systèmes embarqués, où les contraintes de taille et de ressources sont primordiales. La liste des commandes que Busybox émule est disponible dans sa documentation[3].

1. https://strace.io/
2. https://busybox.net/about.html
3. https://busybox.net/downloads/BusyBox.html

Question : combien d'utilitaires Busybox émule-t-il ?

Localisez l'exécutable de Busybox (/usr/sbin/busybox) avec la commande `find`,s puis emprisonnez-le dans jail avec ses dépendances, comme pour la commande ls.

Installez ensuite toutes les commandes émulées par Busybox dans jail/bin :

```
$> chroot jail /usr/sbin/busybox --install /bin
```

3.1 Ajout des messages localisés

L'internationalisation [4] des interfaces logiciels [5] et messages système ainsi que la configuration pour la langue locale de l'utilisateur passent par les catalogues de messages [6], consignés dans les fichiers *.mo.

Pour trouver les fichiers *.mo :

```
$> find / -iname "*.mo"
```

Pour les copier dans jail, aidez-vous du script suivant (que vous pouvez nommer batch_copy.sh) :

```
#! /usr/bin/env bash

if [ "$#" -lt 2 ]
then
 echo "Usage: $0 DESTINATION_DIR PATH_TO_FILE [PATH_TO_FILE...]" >&2
 exit "$E_ERROR"
fi

DEST=$1
nb_args=$#
for (( f=2; f<=$nb_args; f++))
```

4. https://developer.ibm.com/technologies/linux/tutorials/l-lpic1-107-3/
5. https://www.gnu.org/software/libc/manual/html_mono/libc.html#Message-Translation
6. https://www.gnu.org/software/libc/manual/html_mono/libc.html#The-message-catalog-files

```
do
        cp -f -v --parents "${!f}" "$DEST"/
        echo "copied ${!f}"
done
```

Pour les copier dans jail :

```
$> find / -iname "*.mo" | xargs ./batch_copy.sh jail
```

Vérifiez :

```
$> tree jail/usr/share/locale/
```

Copier tous les fichiers LC_* :

```
$> find / -iname "LC_*"|xargs ./batch_copy.sh jail
```

Ceux qui sont déjà dans jail/usr/share seront omis.

3.2 Ajout des informations de terminal pour vi

Terminfo :

```
$> cp -r --parents /usr/share/terminfo jail
```

Tty[7][8][9] :

Figer les paths dans jail/root/.bashrc :

```
export PATH=/bin:/usr/bin:/usr/sbin:/sbin
```

7. https://www.howtogeek.com/428174/what-is-a-tty-on-linux-and-how-to-use-the-tty-command/
8. https://www.kernel.org/doc/Documentation/admin-guide/devices.txt
9. https://pubs.opengroup.org/onlinepubs/000095399/basedefs/xbd_chap10.html

4 Programme mon_app.c

Pour compiler le programme :

```
$> gcc mon_app.c -o mon_app
```

Utilisez ldd pour connaître les dépendances de mon_app et assurez-vous qu'elles sont bien présentes ou alors copiez-les vers l'emplacement approprié dans jail. Copier mon_app dans le répertoire bin de jail.

Le programme accède à trois emplacements :

- /dev/null
- /dev/random
- /proc/%d/cmdline, où %d désigne son PID

Pour connaître les systèmes de fichiers et leur point de montage, la commande `df` peut être utile. La configuration des montages est spécifiée dans le fichier /etc/fstab. Pour explorer plus précisément les points de montage, utiliser la command `findmnt`[10].

Avant de réaliser les montages nécessaires dans jail, l'exécution de mon_app doit échouer.

```
$> chroot jail mon_app
```

ou :

```
$> chroot jail /bin/bash
$> mon_app
```

Réaliser les montages des systèmes de fichier requis vers jail :

```
$> cd jail
$> mount --rbind /dev dev/
$> mount -t proc /proc proc/
$> mount --rbind /sys sys/
```

10. https://linuxhandbook.com/findmnt-command-guide/

Afficher les nouveaux montages avec la commande `findmnt`.

Relancer l'application mon_app :

```
$> cd ..
$> chroot jail /bin/bash
$> mon_app
```

À la fin, après avoir quitté l'environnement bash de jail, démonter les systèmes de fichier précédemment montés :

```
$> umount proc
$> umount -lf sys
$> umount -lf dev
```

5 Exercice - à faire plus tard

Créer un utilisateur (ex. vagrant2) à l'aide des commandes `adduser` et `groupadd` en l'attribuant à un groupe **jailedsshusers**.

Configurer `etc/ssh/sshd_config` de manière à ce que chaque fois que le nouvel utilisateur se connecte, son répertoire d'arrivée soit le répertoire jail sur lequel vous avez travaillé jusqu'alors. Redémarrer sshd.

Sans quitter votre session actuelle de vagrant, ouvrez un autre terminal dans le répertoire de travail de votre TP (là où vous avez fait vagrant ssh) pour vous connecter avec vagrant2.

Vérifiez que le nouvel utilisateur vagrant2 n'a accès qu'au système de fichiers limité, enraciné dans jail.

6 Cgroup

Documentation :

- https://facebookmicrosites.github.io/cgroup2/
- https://www.kernel.org/doc/html/latest/admin-guide/cgroup-v2.html

Fixer la mémoire de memory.high à 32 Mo

(dans le sous-répertoire **jail/cgroupv2/mon_conteneur**) :

```
$> echo 33554432 > memory.high
```

Avant d'exécuter chroot avec votre bash, refaites les montages vus plus haut (dev, proc et sys). Puis exécuter chroot dans jail avec votre bash.

Trouver le PID du processus bash :

```
$> echo $$
```

Associer le processus à votre nouveau cgroup :

```
$> echo $$ > ./cgroupv2/mon_conteneur/cgroups.procs
```

7 Namespaces

Pour lister les namespaces existants :

```
$> lsns
```

Alternativement :

```
$> ls /proc/*/ns
```

Le processus bash a comme identifiant self :

```
$> ls /proc/self/ns
```

Chaque processus par défaut a un namespace distinct des autres. Les processus dans un même namespace peuvent communiquer, interagir. Les processus isolés (dans un conteneur) dans un namespace ne peuvent pas accéder à de l'information dans un namespace différent.

7.1 Création d'un namespace

Dans cet example, nous allons créer un PID namespace (isolation de processus à travers une nouvelle arborescence).

Il est préférable d'utiliser un processus qui ne s'exécute pas sur votre machine. L'objectif est de pouvoir vérifier son isolation après l'avoir lancé et pendant que ce processus tourne. Puisque vous utilisez déjà bash, installer zsh sur votre système. Pour ce faire, quittez la VM, puis mettez à jour votre bootstrap.sh avec l'installation du package :

```
$> sudo dnf -y install zsh.x86_64
```

Mettez à jour la configuration de l'OS dans la VM :

```
$> vagrant provision
```

Si cela ne fonctionne pas, forcez l'arrêt puis redémarrez :

```
$> vagrant halt
$> vagrant up --provision
```

Rouvrez ensuite une session dans la VM redémarrée (`vagrant ssh`), puis passez root.

Tout d'abord assurez-vous que le processus zsh ne tourne pas. La commande suivante ne fera aucun affichage :

```
$> pidof zsh
```

Créez ensuite le nouveau namespace en y exécutant zsh :

```
$> unshare --fork --pid --mount-proc zsh
```

Vous pouvez changer de répertoire de travail à la création du namespace ci-dessus en spécifiant l'option –wd :

```
$> unshare --fork --pid --wd=jail --mount-proc zsh
```

Trouvez le PID de zsh :

```
$> pidof zsh
```

Le PID devrait être 1. Or, habituellement, seul systemd a pour PID 1. Le PID de zsh est 1 dans ce contexte car c'est un processus isolé dans son namespace. Tous les processus fils de zsh auront des PID supérieurs à 1, dans ce namespace.

Pour vérifier les deux observations précédentes, ouvrez une autre session dans votre VM (`vagrant ssh`) via un autre terminal. Puis cherchez le pid de zsh dans le namespace racine :

```
$> pidof zsh
```

Son pid dans le namespace racine est bien supérieur à 1.

Cherchez le pid de systemd :

```
$> ps 1
```

Dans le namespace de zsh, tapez :

```
$> ps aux
```

Vous verrez que le pid de zsh est 1, celui de son processus fils ps est supérieur à 1. Les namespaces, comme les cgroup, fournissent une brique de base pour conteneuriser des processus.

Pour quitter le namespace, faites Ctrl-D.

À faire : Créez de nouveau un namespace pour bash et testez-y les commandes `id` / `ps` / `ls`. Testez les mêmes commandes dans l'autre session que vous avez ouverte pour chercher le pid de zsh dans le namespace racine. Lancer l'application `mon_app` en tâche de fond dans le nouveau namespace de bash :

```
$> ./bin/mon_app > mon_app.txt &
```

Cherchez le pid de son processus avec :

```
$> $!
```

Dans l'autre session, cherchez le pid avec :

```
$> ps aux
```

Enfin, toujours dans le nouveau namespace de bash, essayer la commande `poweroff`, puis essayez la même commande après avoir quitté le namespace. Observez que votre session dans l'autre terminal aura été fermée. Une fois sorti de la session, vérifiez que la VM est bien éteinte :

```
$> vagrant status
```

---

8 Feedback

Lom M. Hillah (lhillah@parisnanterre.fr)

9 Licence