Firejail : Le couteau suisse du confinement sous Linux

Vincent Autefage

▶ **To cite this version:**

Vincent Autefage. Firejail : Le couteau suisse du confinement sous Linux. JRES 2022 : 14èmes Journées Réseaux, RENATER, May 2022, Marseille, France. pp.Article 31. hal-03640803v2

HAL Id: hal-03640803
https://hal.science/hal-03640803v2

Submitted on 18 May 2022

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

Firejail : Le couteau suisse du confinement sous Linux

Vincent Autefage
IUT Informatique - Université de Bordeaux
15 Rue Naudet
33175 Gradignan Cedex, France

Résumé

L'isolation des processus sous Linux est un sujet bien connu des administrateurs systèmes qui peut porter sur un grand nombre d'éléments distincts (e.g. système de fichiers, réseau, espace mémoire, espace de PID). Du simple changement de racine à la virtualisation lourde, en passant par les conteneurs ou bien encore les systèmes de contrôle d'accès obligatoire, un grand nombre de techniques existe pour répondre à ce besoin de confinement (sans mauvais jeu de mots). Toute technique d'isolation a malheureusement un coût, que ce soit en espace de stockage, en perte de performance suite aux opérations induites par la surcouche du système d'isolation mais aussi en complexité de configuration et d'intégration.

Le noyau Linux propose un ensemble de fonctionnalités allant dans le sens d'une isolation sur mesure tout en minimisant le coût de surcharge. Firejail est un outil libre offrant une interface simplifiée et donnant accès à la manipulation de ces fonctionnalités.

Cet article donne un aperçu ainsi que des cas d'usages d'un tel outil dans le cadre d'un système d'information mais également dans un contexte d'enseignement. Nous utilisons Firejail comme moyen d'aborder la problématique de l'isolation des processus sous Linux dans sa globalité.

Mots-clefs

Firejail, Sandbox, Sécurité, Noyau Linux

1 Introduction

L'isolation de processus est une notion qui fait référence à un ensemble de techniques ayant pour but de limiter les accès au système pour un processus unique ou bien un groupe de processus. Cette isolation peut porter sur plusieurs aspects tels que la restriction d'accès à certaines parties du système de fichiers, la capacité d'utiliser ou non certains périphériques matériels, comme par exemple une carte réseau, ou bien encore la possibilité de fixer des limites sur l'utilisation des ressources comme la quantité de mémoire et le charge CPU.

Le terme *isolation de processus* ne doit pas être confondu avec son équivalent anglophone (*i.e. process isolation*) qui fait référence à un ensemble de techniques internes au système d'exploitation afin qu'un processus ne soit pas en mesure d'aller lire ou modifier l'espace mémoire d'un autre processus. Dans cet article, le terme *isolation* fait en réalité écho au concept de confinement (*i.e. sandboxing*) répondant au principe du moindre privilège recommandé par l'ANSSI [01][02].

L'exemple le plus ancien, si ce n'est le plus emblématique, est l'appel système *chroot* [03] apparu à la toute fin des années 70 sur UNIX puis au début des années 80 sur BSD. Cet appel système qui

permet de changer la racine du système de fichiers est à la base du fonctionnement des conteneurs mais aussi des images d'amorçage toujours utilisées à l'heure à laquelle cet article est écrit.

Les *conteneurs* sont apparus au début des années 2000 avec les *Jails BSD* [04] permettant ainsi de coupler l'isolation du système de fichier avec une isolation des utilisateurs, des processus ainsi que du réseau tout en reposant sur le même noyau. Les grands principes de ces *Jails* sont à la base des multiples solutions de conteneurisation aujourd'hui disponibles (*e.g. LXC, OpenVZ, RKT, runC, systemd-nspawn)* et par extension de leurs systèmes de management (e.g *Docker*, *LXD*, *Podman*) dont la popularité ne cesse de croître depuis plusieurs années [05]. La conteneurisation est souvent décrite comme un « chroot sous stéroïdes » [06], il faut néanmoins être prudent quand à l'aspect sécuritaire de cette technologie [07][08]. La plupart des failles révélées à ce jour se basent sur un défaut de mise à jour du système de fichiers propre à l'image conteneurisée ainsi qu'au partage du noyau entre le conteneur et l'hôte qui l'héberge.

L'*émulation* (*i.e.* hyperviseur de type 2), souvent dénommée par le terme *virtualisation* par abus de langage faisant référence aux *machines virtuelles*, a pour vocation d'émuler tout ou partie du matériel dans le but de faire tourner une image disque dédiée avec son propre noyau. Cette technique peut donc être considérée comme l'étape supérieure des conteneurs quant à son aspect isolant. Le supplément d'isolation fourni par une machine virtuelle a un coût significatif d'un point de vue performance (*e.g.* temps de démarrage, consommation CPU, utilisation mémoire, bande passante d'entrée-sortie) si on la compare à un conteneur équivalent [07][09] bien que ce surplus puisse être limité par l'utilisation d'un accélérateur matériel (*e.g. KVM, HyperV*). L'émulation n'est néanmoins pas exempte de failles de sécurité qui ont par exemple permis à des programmes contenus dans une machine virtuelle de pouvoir accéder à la mémoire de la machine hôte [10][11].

Depuis peu, certains projets comme *Kata* ou *gVisor* proposent une solution à mi-chemin entre conteneur et machine virtuelle [12]. Dans les grandes lignes, l'idée est de proposer des conteneurs exploitant leur propre noyau (ou pseudo-noyau dans le cas de *gVisor*) afin de renforcer la sécurité en limitant le surcoût de consommation induit par une machine virtuelle classique. Cela est possible grâce à l'introduction d'une couche de virtualisation allégée.

L'ensemble des solutions évoquées précédemment engendre un surplus de consommation des ressources du système, notamment sur le trafic des entrées-sorties, mais surtout en espace de stockage au regard de la nécessité de disposer d'une image disque complète par instance.

Afin de réduire le surcoût induit par une telle isolation, la solution la plus pertinente consiste à pouvoir décrire et configurer de façon la plus précise possible le type d'isolation recherchée. C'est en ce sens que les systèmes de contrôle d'accès obligatoire (*e.g. SELinux, AppArmor, Tomoyo*) ont été conçus. Ils permettent en effet de spécifier avec précision la manière dont un processus ou un ensemble de processus doit être isolé du système. Ces solutions reposent toutes sur un framework spécifique fourni par le noyau Linux appelé *Linux Security Modules* (*LSM*) [13]. Cette fonctionnalité permet d'obtenir une isolation avec un coût négligeable sur la performance de l'hôte. Néanmoins, la mise en place d'un tel système est largement reconnue comme complexe dans la mesure où elle doit s'appliquer à l'ensemble du système [14]. *LSM* n'est donc pas nécessairement adapté si l'on souhaite par exemple disposer d'une isolation ponctuelle ou ad hoc.

Cet article s'intéresse à la possibilité d'effectuer une isolation de processus la plus spécifique à un confinement recherché, tout en minimisant l'impact des coûts engendrés et en limitant la complexité de mise en place d'une telle solution. Dans un premier temps, nous détaillerons les fonctionnalités proposées par le noyau Linux allant dans le sens d'une isolation sur mesure. Par la suite, en section 3, nous présenterons l'outil *Firejail* permettant de manipuler avec simplicité les fonctionnalités présentées en section 2. Enfin, nous donnerons en section 4 quelques cas d'utilisation réels de cet outil dans un cadre industriel ainsi que dans le cadre d'un projet de recherche et développement.

2 Isolation des en processus sous Linux

Depuis le début des années 2000, le noyau Linux s'est vu doté de quatre briques de sécurité élémentaires allant dans le sens d'une isolation sur mesure permettent ainsi d'effectuer un confinement des plus précis. Ces fonctionnalités sont utilisables indépendamment les unes de autres.

2.1 Namespaces

Les *namespaces* [15] sont apparus en 2002 avec la version 2.4 du noyau Linux. Ils permettent de créer des groupes de processus partageant des espaces de ressources communes. Un même processus peut être dans plusieurs de ces espaces au même moment. L'idée sous-jacente est de créer des groupes isolés à la fois du système mais aussi des uns des autres. Par défaut, un processus hérite des *namespaces* de son processus parent.

La manipulation des différents *namespaces* est permise au travers d'appels systèmes dédiés :

- clone permet de créer un nouveau processus. À la différence de fork qui est l'appel système le plus connu pour cet usage, clone permet de spécifier un grand nombre de paramètres qui définissent les liens de partage entre le processus appelant et le nouveau processus créé. Le processus parent peut ainsi créer un nouveau *namespace* et placer son nouveau processus fils à l'intérieur.
- setns permet de rattacher un processus existant à un *namespace* créé au préalable.
- unshare permet à un processus de se détacher d'un *namespace* pour se rattacher à un nouveau.

Les informations concernant les *namespaces* auxquels un processus est rattaché sont accessibles dans /proc/<pid>/ns. À l'heure où nous écrivons cet article, la version actuelle du noyau Linux propose 8 *namespaces* différents : *pid*, *ipc*, *network*, *user*, *uts*, *time*, *mount* et *cgroup*.

2.1.1 PID

Les *pid_namespaces* permettent de créer des espaces de PID séparés. Un processus initial avec un PID de 1 se trouve dans chaque nouveau *pid_namespace* permettant ainsi de tuer l'ensemble des processus de l'espace de noms en tuant ce processus initial. Deux processus se trouvant dans deux *pid_namespaces* distincts peuvent partager le même PID ; la valeur de ces PIDs étant indépendante d'un *pid_namespace* à l'autre. Du point de vue du système (*i.e.* hors tout *pid_namespace*), les différents processus contenus dans ces espaces de noms restent visibles avec pour chacun un PID

unique et cohérent via une table de correspondance. Les processus contenus au sein d'un *pid_namespace* n'ont pas de visibilité sur les processus vivant hors de ce même espace ; les données visibles dans /proc varient donc d'un *pid_namespace* à l'autre.

2.1.2 IPC

Les *ipc_namespaces* se concentrent sur les communications inter-processus (*e.g.* pipes, mémoires partagées, signaux, sémaphores). Un tel espace de noms permet par exemple d'isoler les mémoires partagées du système et celles des processus appartenant à un *ipc_namespace* mais aussi entre *ipc_namespaces* distincts.

2.1.3 Network

Les *network_namespaces* permettent d'isoler l'ensemble des composants réseaux (*i.e* interfaces, adresses IP, tables de routage, règles de pare-feu, piles réseaux). Il est ainsi possible d'associer des interfaces virtuelles avec des adresses IP dédiées à chaque *network_namespace* et d'y réutiliser les même ports au sein de plusieurs espaces de noms distincts. Une interface réseau qu'elle soit physique ou virtuelle ne peut être rattachée qu'à un seul *network_namespace* au même moment. La communication entre ces différentes interfaces peut être établie par un tunnel (*i.e. point-à-point*) ou bien par un pont (*i.e. bridge*).

2.1.4 User

Les *user_namespaces* permettent à la fois de disposer d'identifiants d'utilisateurs et de groupes distincts mais aussi d'avoir la main sur la configuration des droits et privilèges associés. Il est ainsi possible de donner des privilèges factices à un utilisateur au sein d'un tel espace de noms sans que ses actions puissent aller au-delà des droits qu'il a réellement sur le système. On peut ainsi disposer d'un compte *root* (*i.e.* UID 0) au sein d'un *user_namespace* qui est en réalité un compte régulier voir anonyme du point de vue du système. La traduction entre UIDs internes au *user_namespace et* UIDs réels sur le système est ici encore possible grâce à une table de correspondance.

2.1.5 UTS

Les *uts_namespaces* permettent de spécifier un nom d'hôte ainsi qu'un nom de domaine différent pour les processus appartenant au dit espace de noms. De manière identique aux *namespaces* présentées précédemment, les propriétés internes à un *uts_namespace* ne sont visibles que pour les processus qui en font partie.

2.1.6 Time

Les *time_namespaces* sont récemment apparus avec la version 5.6 du noyau Linux. Ils fonctionnent de façon similaire aux *uts_namespaces* en se focalisant sur la manipulation du temps. Ainsi, les processus appartenant à un tel espace de noms peuvent disposer d'une date système différente ou encore d'une configuration d'horloge spécifique. Cela aura notamment des conséquences sur certains appels systèmes comme ceux permettant de récupérer le temps courant ou bien ceux qui ont pour but d'endormir un processus pendant une durée déterminée.

2.1.7 Mount

Les *mount_namespaces* sont dédiés à l'isolation du système de fichiers. Nous pouvons bien évidemment faire le lien avec l'appel *chroot* mais réduire les possibilités offertes par les *mount_namespaces* au simple changement de racine serait passer à coté de l'essentiel. L'objectif principal est de permettre une limitation d'accès et de visibilité des points de montages à un groupe de processus. Il est en outre autorisé de partager des points de montages entre plusieurs *mount_namespaces*. Il est ainsi possible d'avoir une arborescence en apparence similaire dans deux *mount_namespaces* mais qui est en réalité différente (*e.g.* un chemin identique vers le même fichier ou répertoire dans les deux *mount_namespaces* peut en réalité adresser un nœud différent). Cette mécanique permet par exemple de disposer d'un répertoire utilisateur factice (ou temporaire) pour les processus appartenant à un *mount_namespace* qui aurait été configuré dans ce sens.

2.1.8 Cgroup

Les *cgroup_namespaces* permettent de masquer et d'isoler les *Control Groups* auxquels appartiennent un ensemble de processus. Les détails ainsi que le fonctionnement des *Control Groups* sont détaillés dans la section 2.2.

2.1.9 Utilisation des espaces de noms

Comme nous l'avons vu en introduction de cette section, les *namespaces* sont configurés au travers d'appels systèmes dédiés (*i.e.* au travers d'une API en langage C). Il existe néanmoins des commandes *shell* permettant une manipulation directe telle que la commande unshare qui ne doit pas être confondue avec l'appel système du même nom.

Nous donnons ci-dessous, un exemple de scénario dans lequel un nouveau *shell Bash* est lancé avec trois nouveaux espaces de noms *net*, *pid* et *uts*. Dans ces espaces, nous pouvons constater que notre *shell Bash* possède un PID de valeur 1, qu'aucune interface réseau autre que la boucle locale n'est visible et que la changement du nom de l'hôte est sans effet le système.

```
# System Side
# -----------
> ip -br addr
lo    UNKNOWN    127.0.0.1/8
eno1  UP         192.168.1.14/24
wlo1  DOWN
> ps -e
  PID TTY          TIME CMD
...
11456 pts/1    00:00:00 bash
13611 pts/1    00:00:00 ps
> hostname
my-linux
> unshare --uts --pid --net --fork \
--mount-proc /bin/bash
```

```
# Namespace Side
# --------------
> ps -e
  PID TTY          TIME CMD
    1 pts/1    00:00:00 bash
   89 pts/1    00:00:00 ps
> ip -br addr
lo   DOWN
> hostname linux-in-ns
> hostname
linux-in-ns
> exit

# System Side
# -----------
> hostname
my-linux
```

2.2 Control Groups

Les *Control Groups*, communément dénommé Cgroups [16], sont apparus en 2007 avec la version 2.6 du noyau Linux. Ils ont été mis à jour en version 2 lors du passage à la version 4.5 du noyau par souci de simplification dans la manipulation de cette fonctionnalité.

Un *Cgroup* n'est pas un système d'isolation à proprement parler a contrario des *namespaces*. Les *Cgroups* ont pour objectif de contrôler, prioriser et limiter la consommation des ressources du système par un processus ou un groupe de processus donné. Il est ainsi possible d'agir sur plusieurs types de ressources que l'on appelle *sous-systèmes* tels que la consommation CPU, la quantité de mémoire vive allouée, la bande passante d'entrées-sorties pour un périphérique donné ou encore le nombre de processus fils qu'un processus parent peut engendrer.

L'ensemble des informations concernant les *Cgroups* du système est consultable dans le répertoire /sys/fs/cgroup. Les informations concernant les *Cgroups* auxquels un processus est rattaché sont accessibles dans /proc/<pid>/cgroup. L'interaction avec les *Cgroups* s'effectue au travers du système de fichiers contenu dans ces deux répertoires.

Dans l'exemple suivant, nous créons un nouveau C*group* appelé jres ayant pour objectif de limiter le nombre de processus à 2 puis à 1 et de bloquer l'accès en lecture au périphérique /dev/random. Nous attachons ensuite notre *shell* courant à ce C*group* afin d'observer les effets de cette limitation.

```
> mkdir /sys/fs/cgroup/pids/jres
> echo 2 > /sys/fs/cgroup/pids/jres/pids.max
> echo $$ > /sys/fs/cgroup/pids/jres/cgroup.procs
> whoami
root
> echo 1 > /sys/fs/cgroup/pids/jres/pids.max
> whoami
sh: Cannot fork
> echo max > /sys/fs/cgroup/pids/jres/pids.max
> dd if=/dev/random of=/tmp/out bs=1M count=10
10485760 octets (10 MB, 10 MiB) copiés, 0,00719908 s, 1,5 GB/s
> ls -l /dev/random
crw-rw-rw- 1 root root 1, 8 nov.  24 20:46 /dev/random
> mkdir /sys/fs/cgroup/devices/jres
> echo c 1:8 r > /sys/fs/cgroup/devices/jres/devices.deny
> echo $$ > /sys/fs/cgroup/devices/jres/cgroup.procs
> dd if=/dev/random of=/tmp/out bs=1M count=10
dd: impossible d'ouvrir '/dev/random': Opération non permise
```

2.3 Capabilities

Les *Capabilities* [17] ont commencé à être introduites dans le noyau Linux 2.2 afin de limiter ou d'augmenter les privilèges donnés à un processus particulier. On parle d'opération privilégiée pour des actions nécessitant d'être *root* sur le système. Ce mécanisme permet ainsi d'autoriser le programme *ping* à créer des *raw sockets* ou encore d'autoriser un serveur web à utiliser un port réservé (*e.g.* 80 et 443) plutôt que de les lancer avec l'utilisateur *root* ou bien de leur attribuer un *setuid-bit* qui leur donnerait de fait, les pleins pouvoirs sur le système dans son ensemble. Il est bien évidemment possible d'effectuer l'opération inverse, c'est à dire retirer une capacité à un programme qui est exécuté avec les droits administrateurs.

Les *Capabilities* peuvent être manipulées grâce à des appels systèmes spécifiques comme prctl ou bien directement grâce aux commandes setcap et getcap et setpriv. Dans l'exemple ci-dessous nous ajoutons au programme *busybox* lancé avec un utilisateur régulier les capacités à effectuer un ping (*i.e.* création de *raw sockets*) ainsi qu'à utiliser les ports réseaux réservés dans le but de fournir un service web sur le port TCP 80.

```
> whoami
reguser
> busybox ping -q -c1 127.0.0.1
PING 127.0.0.1 (127.0.0.1): 56 data bytes
ping: permission denied (are you root?)
> busybox httpd -f -vv -p 80
httpd: bind: Permission denied
> sudo bash
> whoami
root
> setcap "cap_net_raw+ep cap_net_bind_service+ep" /bin/busybox
> exit
> whaomi
reguser
> busybox ping -q -c1 127.0.0.1
PING 127.0.0.1 (127.0.0.1): 56 data bytes
--- 127.0.0.1 ping statistics ---
1 packets transmitted, 1 packets received, 0% packet loss
round-trip min/avg/max = 0.070/0.070/0.070 ms
> busybox httpd -f -vv -p 80
Listening on :80
```

2.4 Secure Computing Modes

Les *Secure Computing Modes* [18] plus communément appelés *Seccomp* sont apparues avec la version 2.6 du noyau Linux en 2005. Cette fonctionnalité permet à un processus de supprimer sa capacité à appeler certains appels systèmes dans le but de renforcer sa propre sécurité. *Seccomp* peut être utilisé de deux façons :

- le mode *strict* qui supprime la quasi totalité des appels systèmes pouvant être appelés.
- Le mode *filter* qui permet de spécifier explicitement les appels à supprimer.

Le mode *filter* est beaucoup plus communément utilisé que le mode *strict*. Il en résulte la dénomination *seccomp-bpf* pour *SECure COMPuting with Berkeley Packet Filters*.

La manipulation de S*eccomp* se fait exclusivement via des appels systèmes tels que seccomp, prctl et bpf.

3 Firejail

Comme nous avons pu le constater dans la section précédente, l'utilisation conjointe des quatre techniques d'isolation proposées par le noyau Linux n'est pas chose aisée d'autant plus que le scénario de confinement recherché est complexe.

Firejail [19] est un outil libre supportant les versions 3 et ultérieures du noyau Linux ayant pour but d'offrir un accès simplifié à l'ensemble des ces fonctionnalités sans pour autant nécessiter de système de fichiers dédiés comme c'est le cas pour les conteneurs. L'interface de cet outil permet de décrire un scénario d'isolation avancé avec un niveau d'abstraction supérieure. Écrit en langage C, il ne nécessite ni démon en arrière plan ni configuration particulière du système. Il offre une solution de confinement simplifiée, sur mesure et sans coût résiduel.

Firejail se présente sous la forme d'un exécutable en ligne de commande disposant du *setuid-bit* lui conférant ainsi l'accès complet aux fonctionnalités introduites dans la section précédente. Le fonctionnement général de l'outil s'articule autour des notions de *sandbox* et de *profils*. Dans le vocabulaire de *Firejail*, un *profil* représente une configuration de sécurité particulière tandis qu'une *sandbox* représente une instance du programme héritant d'un ou plusieurs *profils*. Pour imager plus simplement ces notions, une *sandbox* peut être vue comme un environnement restreint dans lequel évolue un ensemble de processus. Le cahier des charges décrivant l'ensemble des restrictions appliquées à cet environnement est matérialisé par un *profil*. Un profil peut être formalisé par un fichier de configuration ou bien défini dynamiquement grâce à un ensemble d'options proposées par la ligne de commande de *Firejail*. L'outil est fourni avec plusieurs centaines de profils différents conçus sur mesure pour des programmes spécifiques. Par défaut, *Firejail* se base sur le nom de l'exécutable afin d'appliquer un profil particulier. Par exemple la commande firejail firefox cherchera si un fichier firefox.profile définissant des règles de restrictions existe dans le répertoire de profils par défaut (dans la plus part des cas /etc/firejail). Il est bien évidemment possible de préciser des règles spécifiques au lancement en surcharge ou bien en remplacement de la configuration définie dans un profil. Un appel à *Firejail* sans spécifier de programme lance une nouvelle instance du shell utilisateur dans sa configuration restreinte telle que définie dans son profil dédié.

Un exemple trivial est l'interdiction d'accès au répertoire personnel de l'utilisateur (option private) ainsi qu'aux cartes réseaux du système (option net=none). L'exemple ci-dessous illustre une telle utilisation en désactivant toute mesure d'isolation supplémentaire (option no-profile).

```
> ip -br addr
lo    UNKNOWN    127.0.0.1/8
eno1  UP         192.168.1.14/24
> ps -e
  PID TTY          TIME CMD
...
11456 pts/1    00:00:00 bash
11740 pts/1    00:00:00 ps
> ls -a $HOME
. .. .bash_history .bashrc .cache .config
.gnupg .local .Xauthority desktop doc work tmp
> firejail --no-profile --private –net=none
Parent pid 11752, child pid 11753
Child process initialized in 45.11 ms
> ps -e
  PID TTY          TIME CMD
    1 pts/1    00:00:00 firejail
    8 pts/1    00:00:00 bash
   12 pts/1    00:00:00 ps
> ip -br addr
lo    UNKNOWN    127.0.0.1/8
> ls $HOME
.bashrc .Xauthority
> tree -a $HOME |wc -l > $HOME/files.jail.count
> ls $HOME
.bashrc .Xauthority files.jail.count
> cat files.jail.count
6
> exit
> ls -a $HOME
. .. .bash_history .bashrc .cache .config
.gnupg .local .Xauthority desktop doc work tmp
> tree -a $HOME |wc -l
90000
```

Comme nous l'avons évoqué précédemment, les profils de confinement peuvent être consignés dans un fichier. La quantité d'options disponibles pour définir un profil étant trop importante pour toutes les présenter dans cet article, nous proposons ci-dessous une analyse succincte d'un extrait du profil fourni par les développeurs de *Firejail* pour le lecteur vidéo libre VLC.

```
include disable-common.inc
include disable-devel.inc
include disable-exec.inc
include disable-interpreters.inc
include disable-passwdmgr.inc
include disable-programs.inc

mkdir ${HOME}/.cache/vlc
mkdir ${HOME}/.config/vlc
mkdir ${HOME}/.local/share/vlc
whitelist ${HOME}/.cache/vlc
whitelist ${HOME}/.config/vlc
whitelist ${HOME}/.config/aacs
whitelist ${HOME}/.local/share/vlc

include whitelist-common.inc
include whitelist-player-common.inc
include whitelist-var-common.inc
```

```
caps.drop all
seccomp
netfilter
protocol unix,inet,inet6,netlink

nogroups
nonewprivs
noroot
nou2f

private-bin cvlc,nvlc,qvlc,rvlc,svlc,vlc
private-dev
private-tmp

shell none
dbus-user none
dbus-system none
memory-deny-write-execute
```

L'instruction include permet d'inclure des fichiers de profils annexes. Les fichiers ainsi inclus par les 6 premières lignes font appel aux instructions blacklist, read-only et noexec qui permettent pour un fichier ou un répertoire de respectivement interdire l'accès, l'écriture et l'exécution. Ces instructions sont ici appelées sur prêt de 2000 entrées interdisant ainsi l'accès à un grand nombre de données privées de l'utilisateur (*e.g.* ~/,gnupg) et du système (*e.g.* /etc/ssh) mais également l'exécution de programmes sensibles (*e.g.* situés dans /sbin).

Les instructions de type whitelist permettent de spécifier explicitement l'ensemble des fichiers et des répertoires accessibles en écriture. Toute modification du système de fichiers en dehors des entrées spécifiées en liste blanche est invisible hors de la *sandbox*. Cette instruction permet ainsi de fournir une liste exhaustive des entrées du système de fichiers dont les modifications seront pérennes.

L'instruction caps.drop précise les C*apabilities* qui doivent être abandonnées au sein de la *sandbox*. Toutes les *capabilities* fournies par le système sont ici concernées. L'instruction seccomp permet quant à elle d'interdire un ensemble d'appels systèmes considérés comme dangereux (e.*g.* execve, ptrace). L'instruction netfilter configure un ensemble de règles de pare-feu au sein de la *sandbox* afin que cette dernière ne puisse ni transmettre de paquet (*i.e.* désactivation du *forwarding*) ni jouer le rôle de serveur (*i.e.* désactivation de la chaîne *input* sauf pour les connexions établies au préalable). L'instruction protocol permet de spécifier les protocoles réseaux autorisés.

L'instruction nogroup supprime les groupes UNIX auxiliaires pour l'utilisateur courant, nonewprivs empêche l'acquisition de privilèges supérieurs, noroot supprime l'utilisateur root de la *sandbox* et nou2f supprime le support des périphériques U2F (*i.e.* périphérique d'authentification).

Les instructions private-bin, private-dev et private-tmp permettent de restreindre à la fois l'accès mais aussi la visibilité des entrées présentes dans les répertoires /etc, /tmp et /bin ainsi que toutes ses variantes (*e.g.* /usr/bin, /usr/local/bin). Un exception est ici faite pour une liste d'exécutables nécessaires au bon fonctionnement de VLC.

L'instruction shell none désactive le lancement d'un shell de commandes. Les instructions dbus-user none et dbus-system none permettent d'interdire le dialogue avec le démon *dbus*. Enfin, memory-deny-write-execute empêche la création de zones mémoires dynamiques contenant du code exécutable. Cette dernière option devra néanmoins être supprimée du profil dans le cas où VLC doit décoder des flux matériellement (*i.e.* accélération GPU pour le décodage vidéo).

Cet exemple permet d'illustrer la facilité de conception d'un profil de sécurité sur mesure. *Firejail* propose ainsi une interface intuitive permettant d'appliquer des règles de confinements complexes à l'aide d'instructions simples. Il permet ainsi de s'abstraire de la manipulation directe des fonctionnalités de sécurité du noyau Linux présentées en section 2.

4 Exemples d'utilisation réelle

4.1 Isolation de serveurs VNC dans un contexte industriel

Dans un contexte industriel, nous avons utilisé *Firejail* afin de pouvoir créer des instances de serveurs VNC restreintes. Les applications tournant dans ces serveurs VNC ayant besoin d'accéder à haute fréquence à du matériel spécifique (*e.g.* cartes d'acquisition), à certaines fonctionnalités des cartes graphiques (*e.g.* calculs GPU, rendu 3D) ainsi qu'à certaines parties du système de fichiers de l'hôte, *Firejail* nous a semblé être un compromis intéressant permettant de durcir la protection du système contre d'éventuels détournements tout en limitant l'impact sur les performances générales.

Firejail nous a ainsi permis d'appliquer un certain nombre de restrictions telles que la suppression de certains appels systèmes ainsi que la grande majorité des *capabilities*, le durcissement d'accès au serveur X de l'hôte, la limitation de certaines ressources (*e.g.* bande passante réseau, mémoire), la limitation d'accès à certaines parties du système de fichiers (*e.g.* /var, /home, /etc), la suppression d'accès au compte *root* ou encore la mise en place d'un répertoire personnel factice.

4.2 Plateforme d'expérimentation réseau

Dans le cadre d'un projet de recherche et développement, nous devions simuler un réseau de plusieurs instances d'un même programme nécessitant ici encore des accès particuliers au système (*i.e.* matériel spécifique, cartes graphiques, parties du système de fichiers). L'objectif était de tester la robustesse de l'application aux fluctuations des connexions réseaux. *Firejail* nous a permis de mettre en place ces instances isolées les unes des autres en simulant un réseau au propriétés mouvantes (*e.g.* bande passante, latence, gigue, taux d'erreurs de transmission).

4.3 Durcissement d’un système d’intégration continue

Firejail nous a été utile afin de limiter l’empreinte d’un système d’intégration continue (*i.e.* CI) sur le système hôte qui l’hébergeait. La majorité des systèmes d’intégration continue en mode *natif* (*i.e.* exécutés dans un shell sans conteneur) laisse des fichiers temporaires ainsi que des traces diverses sur le système de fichier de l’hôte pouvant notamment poser des problèmes de confidentialité des données. *Firejail* nous a ici permis de supprimer intégralement ces traces par l’instanciation de répertoires factices. En outre, il nous a permis de limiter l’accès du CI au système ainsi qu’au réseau de l’hôte sans pour autant devoir mettre en place un système à base de conteneurs.

5 Conclusion

L’écosystème qui gravite autour de l’isolation des processus sous Linux regroupe plusieurs solutions répondant à des critères distincts. *Firejail* fournit un accès simplifié aux fonctionnalités de sécurité du noyau Linux proposant ainsi une isolation sur mesure. Ce outil se concentre sur l’isolation de processus *ad hoc* se différenciant ainsi des systèmes d’isolation d’ordre global tels que les systèmes de contrôle d’accès obligatoire. Il se différencie des machines virtuelles ainsi que des conteneurs par sa granularité d’isolation mais aussi par sa légèreté et sa porosité au système hôte ; bien que cette dernière puisse être inférieure à celle d’un conteneur classique via un profil adéquate. Les briques de sécurité du noyau Linux ainsi que *Firejail* ne se limitent pas à des restrictions *ad hoc* et peuvent être utilisés pour durcir la sécurité d’un conteneur ou encore d’un service système.

Bibliographie

[01] ANSSI, Recommandations pour la mise en place de cloisonnement système, version 1, décembre 2017

[02] ANSSI, Recommandations de configuration d'un système GNU/Linux, version 1.2, février 2019

[03] The Linux Foundation, *A Brief Look at the Roots of Linux Containers*, mai 2017, https://www.linuxfoundation.org/blog/a-brief-look-at-the-roots-of-linux-containers/

[04] P.H. Kamp et R.N.M. Watson, *Jails: Confining the omnipotent root*, The FreeBSD Project, 2000

[05] Cloud Native Computing Foundation, *Cloud Native Survey 2020: Use of containers in production has increased by 300% since 2016*, novembre 2020

[06] L. Poettering, S*ystemd-Nspawn is Chroot on Steroids*, LinuxCon Europe, 2013

[07] J. Hertz, *Abusing Privileged and Unprivileged Linux Containers*. Whitepaper, NCC Group, volume 48, 2015

[08] S. Sultan, I. Ahmad, T. Dimitriou, *Container Security: Issues, Challenges, and the Road Ahead*, IEEE Access, volume 7, pages 52976-52996, 2019

[09] W. Felter, A. Ferreira, R. Rajamony and J. Rubio, *An Updated Performance Comparison of Virtual Machines and Linux Containers*, IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS), pages 171-172, avril 2015

[10] G. Pék, L. Buttyán and B. Bencsáth, *A Survey of Security Issues in Hardware Virtualization*, ACM Computer Survey, volume 45, 2013.

[11] Irazoqui G., Inci M.S., Eisenbarth T., Sunar B. *Wait a Minute! A fast, Cross-VM Attack on AES*, Srpinger Research in Attacks, Intrusions and Defenses, Lecture Notes in Computer Science, volume 8688, 2014

[12] Wonderfall, *gVisor & Kata : des Sandbox pour Renforcer l'Isolation des Conteneurs*, avril 2021, https://wonderfall.space/gvisor-kata-containers

[13] C. Wright, C. Cowan, S. Smalley, J. Morris and G. Kroah-Hartman, *Linux Security Modules: General Security Support for the Linux Kernel*, ACM USENIX Security Symposium, 2002

[14] Red Hat, *What is SELinux?*, août 2019. https://www.redhat.com/en/topics/linux/what-is-selinux

[15] Linux Man Pages, N*amespaces*, section 7, 2021, https://man7.org/linux/man-pages/man7/namespaces.7.html

[16] Linux Man Pages, *Control Groups*, section 7 2021, https://man7.org/linux/man-pages/man7/cgroups.7.html

[17] Linux Man Pages, C*apabilities*, section 7, 2021, https://man7.org/linux/man-pages/man7/capabilities.7.html

[18] Linux Kernel Documentation, *Seccomp BPF (SECure COMPuting with filters)*, 2021, https://www.kernel.org/doc/html/latest/userspace-api/seccomp_filter.html

[19] Firejail, 2021, https://firejail.wordpress.com