Sécurité des infrastructures de virtualisation

Conteneurs sous Linux : disponibilité, isolation et sécurité

Timothée Ravier

ANSSI

3e année cycle ingénieur STI, option 2SU
2015 – 2016

Plan global

Plan

rlimits : resource limits

- Limite les ressources disponibles pour l'ensemble des processus d'un utilisateur ou d'un groupe ;
- Affichage / manipulation des limites avec `ulimit` ;
- Deux types de limites : `hard` et `soft` ;
- Limite `soft` :
  - limite appliquée par le noyau ;
  - définissable entre 0 et <limite `hard`>.
- `CAP_SYS_RESOURCE` pour augmenter les limites `hard` ;
- Réduction des limites `hard` irréversibles pour `!root` ;
- Limites initiales appliquées lors du login par PAM, configurées dans `/etc/security/limits.conf`.

rlimits : resource limits

- cpu time (seconds) : unlimited
- file size (blocks) : unlimited
- data seg size (kbytes) : unlimited
- stack size (kbytes) : 8192
- core file size (blocks) : 0
- resident set size (kbytes) : unlimited
- processes : 1024
- file descriptors : 1024
- locked-in-memory size (kbytes) : 64
- address space (kbytes) : unlimited
- file locks : unlimited
- pending signals : 22949
- bytes in POSIX msg queues : 819200
- max nice : 0
- max rt priority : 0

Quota disque

- Limite les ressources disque par `user` et par `group` ;
- Limite `soft` & `hard` par système de fichier ;
- La limite `hard` ne peut pas être dépassée ;
- La limite `soft` peut être dépassée pour un temps donné ;
- Une fois ce temps dépassée, la limite `soft` devient `hard` ;
- L'utilisateur doit libérer de la place pour redescendre au dessous de la limite `soft` avant de pouvoir écrire à nouveau des données ;
- Quota disque par « projets » avec XFS [1].

Espace disque réservé pour les processus privilégiés

- Il est possible de réserver de l'espace sur un système de fichier pour permettre aux processus privilégiés (syslog / journald) de continuer à fonctionner lorsque le disque est presque plein ;
- `$ tun2fs -m reserved-blocks-percentage` ;
- Valeur par défaut : 5 % de la taille du système de fichier.

L'overcommit

- Chaque processus a l'impression d'être le seul à utiliser le matériel ;
- On lui propose donc plus de ressources que ce qui est réellement disponible sur la machine ;
- Exemple : l'espace d'adressage virtuel
  - $2^{32} \approx$ 4Go en 32bits. Combien pour 64bits ?
  - `/proc/sys/vm/{overcommit_memory,overcommit_ratio}` ;
  - `Documentation/vm/overcommit-accounting`.
- En tenir compte lorsque l'on atteint les limites.

Priorités, IO, et CPU

- nice level : Défini la priorité d'accès aux ressources CPU pour un processus ;
- ionice : Défini la priorité d'accès aux périphériques pour un processus ;
- cpu affinity : Oblige un processus à utiliser des cœurs définis ;

Limites du modèle statique

- Flexibilité très limitée ;
- Il faut gérer « à la main » les changements de limite : pas de répartition automatique entre les utilisateurs ou les processus ;
- Peu adapté au modèle où l'on va avoir beaucoup de machines virtuelles utilisées par un seul utilisateur ;
- Ou à des services constitués de plusieurs processus que l'on souhaite gérer comme un tout.

cgroups

- Introduit dans le noyau 2.6.24 par des ingénieurs de Google ;
- Permets de regrouper des processus par groupe avec un label ;
- Ces regroupements sont organisés hiérarchiquement et hérités par les processus fils ;
- On utilise alors les « controllers » pour définir un comportement particulier pour un groupe de tâches ;
- Représenté par un système de fichier virtuel : `/sys/fs/cgroup/`.

cgroups : controllers

- Les « controllers » disponibles sont :
  - block io controller : contrôle l'accès aux block devices ;
  - cpu accounting controller : mesure et limite l'utilisation CPU ;
  - cpusets : restreint les groupes à certains CPU ou nœuds ;
  - memory : restreint la consommation en mémoire ;
  - net_cls et net_prio : classer et définir des priorités pour les paquets réseaux ;
  - devices : restreins l'accès aux devices disponibles ;
  - ressource counter : mesure l'utilisation des autres ressources.

cgroups : systemd ?

- Fonctionnement différent suivant si l'on est sur un système avec ou sans `systemd` :
  - Sans systemd : il faut tout gérer à la main (ou avec des scripts). Aucun cgroup n'est utilisé par défaut. Pose des problèmes de type « race condition » ;
  - Avec systemd : répartition automatique des machines virtuelles et des services dans des cgroups ; gestion centralisée par systemd.
- Plus de précisions dans le cours sur systemd ;
- Exemple de répartition dans les cgroups avec libvirtd [2].

cgroups : hiérarchie unifiée

- Pour l'instant chaque `controller` a sa propre hiérarchie ;
- Pose de nombreux problèmes de gestion, performance, complexité ;
- Travail en cours pour utiliser une seule hiérarchie commune à tous les controllers [3].

Plan

Rappel sur les processus et les threads

- Différence entre un processus et un thread ?
- Implémentation dans le noyau :
  - Linux ?
  - Windows ?
  - Darwin ?
- Appels système `fork` et `clone` ;
- Autres combinaisons que les threads et les processus ?

Isolation avec les namespaces I

Principe général :

- Crée un ou plusieurs espace de nom (namespaces ou ns) pour le fils ;
- Isole les objets créés dans un namespace des éléments du namespace parent ;
- Lorsque tous les processus d'un namespace ont terminé leur exécution, les objets restant sont détruits ou renvoyés dans le namespace parent.

Isolation avec les namespaces II

- Fonctionnalité ancienne (début dans la branche 2.4 !) mais étendue récemment ;
- Précédemment principalement utilisée avec PAM : pam_namespace.so ;
- Ajouté comme FLAGS à l'appel système `clone` :
  - int clone(int (*fn)(void *), void *child_stack, int flags, void *arg, /* pid_t *ptid, struct user_desc *tls, pid_t *ctid */) ;
  - CLONE_NEWNET, CLONE_NEWNS, CLONE_NEWPID, CLONE_NEWUTS, CLONE_NEWIPC, CLONE_NEWUSER.

CLONE_NEWNS

Crée un namespace pour les points de montage :

- Présent depuis le noyau 2.4.19 ;
- Permet au fils de monter/démonter des systèmes de fichier sans impacter le père ;
- Nécessite `CAP_SYS_ADMIN`.

CLONE_NEWUTS

Crée un namespace UTS :

- Présent depuis le noyau 2.6.19 ;
- Contient le domain name et le host name ;
- Permets au fils de changer ces informations sans impacter le père ;
- Nécessite `CAP_SYS_ADMIN`.

CLONE_NEWIPC

Crée un namespace pour les IPC :

- Introduit dans le noyau 2.6.19, complété dans le 2.6.30 ;
- Vue isolée des IPC System V : seuls les processus de ce namespace peuvent accéder à ces objets ;
- Destruction des objets à la fin du namespace ;
- Nécessite `CAP_SYS_ADMIN`.

CLONE_NEWNET

Crée un namespace pour les interface réseaux :

- Introduit dans le noyau 2.6.24, complété dans le 2.6.29 ;
- Le fils dispose de sa propre stack réseau : interfaces, stacks IPv4/IPv6, tables de routage IP, règles de parfeu, sockets...
- Une interface physique ne peut être que dans un seul namespace à la fois : rattachée au namespace initial à la destruction ;
- Il faut donc créer des interfaces virtuelles pour pouvoir les utiliser dans les namespaces ;
- Nécessite `CAP_SYS_ADMIN`.

CLONE_NEWPID

Crée un namespace pour les PID :

- Présent depuis le noyau 2.6.24 ;
- Les PIDs recommencent à 1 dans ce namespace ;
- Les processus du namespace parent voient les processus du namespace fils avec des PIDs différents ;
- Les processus dans un namespace ne peuvent voir que ceux du même namespace et des namespaces créés par des processus de leur namespace ;
- Nécessite `CAP_SYS_ADMIN`.

CLONE_NEWUSER

Crée un namespace pour les UID/GID :

- Introduit dans le noyau 2.6.24 (mal documenté, encore partiellement expérimental !) ;
- Les processus peuvent avoir un uid/gid quelconque, indépendamment de ceux du namespace parent (utilise un sorte de table de correspondance) ;
- En revanche le noyau « s'assure » qu'il n'y as pas d'élévation de privilèges par rapport au couple uid/gid initial ;
- Donne l'impression au processus fils qu'il est root sans l'être réellement ;
- Rien à partir du 3.8. Nécessitait `CAP_SYS_ADMIN`, `CAP_SETUID` and `CAP_SETGID`. Comportement variable suivant les distributions.

Vulnérabilités autours des namespaces

- Beaucoup de vulnérabilités trouvées ces dernières années : CVE-2013-1858, [4], [5], [6], [7], [8], [9] ;
- Noyau : cible privilégiée car commun à tous les namespaces ;
- Présentations sur l'exploitation du noyau : [12] et [13].
- Cas non prévus lorsque l'on combine plusieurs namespaces ;
- Failles noyau donc très souvent critiques (« local root ») ;
- User namespace particulièrement touchés.

Plan

Pourquoi utiliser les conteneurs ?

- On souhaite garder les propriétés suivantes :
  - Isolation vis à vis des autres processus / services ;
  - Contrôle de la consommation en ressources.
- Tous nos programmes sont compilés pour Linux et ne dépendent que rarement de la version du noyau utilisée ;
- Pas de besoin de virtualisation plus lourde (KVM, Xen...) :
  - Il n'est pas nécessaire de virtualiser un autre système d'exploitation ;
  - Il n'est pas nécessaire de virtualiser le noyau.

Principe des conteneurs sous Linux

- Technique de « virtualisation » légère ne nécessitant pas de support matériel ;
- Proche du concept des jails de FreeBSD ou des Zones de Solaris ;
- Combiner :
  - gestion des ressources et de l'accès à certains périphériques avec les cgroups ;
  - isolation (partielle) avec les namespaces.
- Nécessite :
  - un programme capable de mettre en place les cgroups/namespaces ;
  - une arborescence propre au conteneur (« image » du /) ;
  - un programme pour gérer les images ;
  - un programme pour mettre en place le réseau entre les conteneurs.

Implémentations sous Linux

- Exemples :
  - Vserver : requiert un patch noyau qui ne sera probablement jamais intégré, support très limité, à éviter ;
  - OpenVZ : requiert un patch noyau qui ne sera probablement jamais intégré, à éviter ;
  - LXC/LXD : LinuX Containers initialement développé par IBM, maintenu par Ubuntu (Canonical) ;
  - Docker : initialement développé par dotCloud, maintenant Docker Inc. Orienté partage, copie, redistribution de conteneurs ;
  - rkt : initié par le projet CoreOS pour corriger certains défauts de Docker ;
  - systemd-nspawn : développé comme outils de test et remplaçant de chroot. Désormais la base de rkt.

LXC/LXD

- Objectif : faire tourner un système complet dans un conteneur ;
- Image : simple `tar.gz` d'un `/` ;
- Création de conteneurs à partir de templates ;
- Avantages :
  - Terrain connu, fonctionnement prévisible ;
  - Aussi flexible qu'un système classique.
- Désavantages :
  - Consommation en ressources ;
  - Duplication des services (journalisation, monitoring…) ;
  - Taille des images importantes ;
  - Gestion plus conséquente (plus proche de la virtualisation « classique ») : mises à jour, sauvegarde…
- LXD : gestionnaire de conteneur LXC.

Docker

- Objectif : faire tourner une seule application par conteneur ;
- Propose un modèle de partage d'images de conteneurs ;
- Dockerfile : sorte de script pour créer des conteneurs ;
- Avantages :
  - Simplificité (récupération des images, utilisation) ;
  - Consommation en ressources minimale ;
  - Images des conteneurs de taille réduite ;
  - Isolation claire entre chaque application.
- Désavantages :
  - Les applications classiques ne sont pas prévues pour fonctionner en tant que PID 1 dans un conteneur : processus zombie...
  - Aucun service support n'est disponible dans le conteneur ;
  - Démon principal : mise à jour Docker $\Rightarrow$ redémarrer tous les conteneurs.

rkt

- Concept de « pods » pour regrouper plusieurs applications ;
- Objectif : faire tourner un gestionnaire de processus par pod et une application par conteneur ;
- Avantages :
  - Petites images, une application par conteneur [19] ;
  - Architecture en « stages » [16], [17], [15] ;
  - Bonne intégration avec systemd [20] ;
  - Meilleure découpage des tâches $\Rightarrow$ meilleure sécurité [21].
- Désavantages :
  - Moins « populaire » que LXC ou Docker [22].

Vulnérabilités liées aux conteneurs

- Nouveau système de fichiers pour permettre de mutualiser le contenu des images de conteneurs, et donc nouvelles vulnérabilités : CVE-2015-1328 [10] ;
- Partage des systèmes de fichiers virtuels entre l'hôte et les conteneurs : `/proc` et `/sys` [11].

Vulnérabilités spécifiques à LXC

- Vulnérabilité à la création d'un conteneur [23] ;
- Vulnérabilité dans les templates de création de conteneurs [24].

Vulnérabilités spécifiques à Docker

Sécurité ?

« Docker is about running random code downloaded from the Internet and running it as root. »

Vulnérabilités spécifiques à Docker

- Démon `docker` tournant en `root` ;
- Récupération d'images depuis le réseau [28] ;
- Opération sur des images qui ne sont pas de confiance [25] ;
- Beaucoup d'images proposés sur le Registry ne sont pas à jour [29] ;
- Signatures des images pas avant la version 1.8 [26], [27] ;
- Plusieurs vulnérabilités : [30], [31], [32].

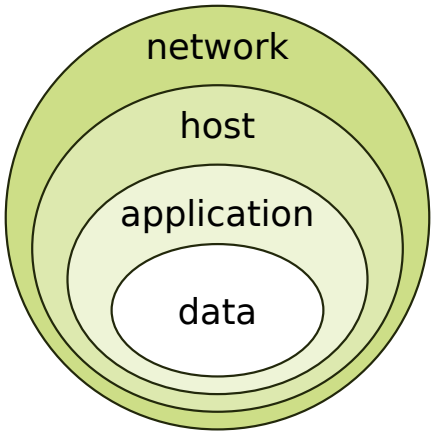
« Containers do not contain »

- L’isolation fournie par les namespaces et les cgroups a encore des limites ;
- Certaines parties du noyau n’ont pas encore été adaptées aux namespaces ;
- État de la situation : [33], [34] ;
- Tous les outils d’infrastructure pour gérer les conteneurs augmentent la surface d’attaque ;
- Une application vulnérable sera toujours vulnérable dans un conteneur.

Risques principaux

- Vulnérabilités noyau ;
- Dénis de service ;
- Sortie d’un conteneur ;
- Image de conteneur malveillante ;
- Compromission des secrets.

Défense en profondeur



- Protections réseau : pare-feu, WAF...
- Protection de l'hôte, isolation des conteneurs : DAC, MAC...
- Protection à l'intérieur du conteneur : binaires durcis...
- Protection des données : chiffrement, intégrité...

`root` & capabilities

- Ne pas faire tourner de conteneur en root ;
- Retirer le maximum de capabilities aux processus s'exécutant dans un conteneur ;
- Cf. section sur les namespaces : `CAP_SYS_ADMIN` $\Rightarrow$ `root` [36] ;
- Attention : de nombreuses capabilities sont en fait équivalentes à `root` [35].

Mises à jour

- Conteneur $\Leftrightarrow$ énorme binaire statique ;
- Absolument prévoir un cycle de mise à jour ;
- Outils pour vérifier qu'un image ne contient pas de vulnérabilité déjà connue [37], [38].

SELinux

- Objectif : contenir les conteneurs ;
- Supporté par Docker et rkt [40], [39], [41] ;
- Limite la portée d'une grande partie des vulnérabilités.

Noyau Linux : surface d'attaque

- Vecteurs d'attaque :
  - Faille exploitable à distance : souvent réseau, très rare ;
  - Faille locale : appels systèmes, surface d'attaque large :
    - Gestion de la mémoire ;
    - Systèmes de fichiers ;
    - Drivers : ioctl...
    - Réseau : netfilter...
- Retirer des appels systèmes (dur) ;
- Limiter les appels systèmes disponibles : seccomp.

Noyau Linux : réduire la surface d'attaque avec seccomp

- Ce n'est pas un mécanisme de « sandbox » ;
- Permet uniquement de limiter les appels systèmes disponibles pour une application ;
- Si un appel système bloqué est appelé, l'application peut être tuée avec un signal (configurable) ;
- Système de blacklist / whitelist ;
- A combiner impérativement avec un LSM pour le contrôle d'accès.

Noyau Linux : durcissement

- grsecurity & PaX.

Spécificités Docker

- Accès au démon docker $\Leftrightarrow$ `root` ;
- Restreindre impérativement l'accès au démon docker : [44] ;
- Plusieurs guides pour utiliser Docker correctement : [42], [43].

Plan

CoreOS et Project Atomic

- Systèmes d'exploitation centrés sur l'utilisation de conteneurs ;
- Adaptés pour la disponibilité et l'intégration dans un cluster sur un Cloud ;
- Deux principaux exemples :
  - CoreOS : [45], [46], [47] ;
  - Project Atomic (Fedora, RHEL et CentOS) : [48].

Kubernetes, fleet, Swarm...

- Plus de conteneurs par hôte donc plus de gestion ;
- Similitude forte entre les problèmes liés aux Clouds et les problèmes conventionnels liés aux clusters ;
- Nouveaux outils de gestion de clusters de conteneurs :
  - Kubernetes [49], [50], [51] ;
  - fleet [52], etcd [53], flannel [54] ;
  - Swarm [55], Compose [56] ;
  - Flocker [57] ;
  - OpenShift [58] ;
  - ...

Références I

1. xfs_quota(8) : http://man7.org/linux/man-pages/man8/xfs_quota.8.html
2. Control Groups Resource Management : http://libvirt.org/cgroups.html
3. Control Group v2 : https://git.kernel.org/cgit/linux/kernel/git/torvalds/linux.git/tree/Documentation/cgroup-v2.txt
4. CVE request Linux kernel : ns : user namespaces panic : http://www.openwall.com/lists/oss-security/2015/05/29/5
5. Linux kernel source : commit 51dfcb076d1e1ce7006aa272cb7c4514740c7e47

Références II

6. Linux kernel source : commit 87c31b39abcb6fb6bd7d111200c9627a594bf6a9
7. Linux namespaces : It is possible to escape from bind mounts : http://www.openwall.com/lists/oss-security/2015/04/03/7
8. User namespaces + overlayfs = root privileges : https://lwn.net/Articles/671641/
9. Anatomy of a user namespaces vulnerability : https://lwn.net/Articles/543273/
10. CVE-2015-1328 : incorrect permission checks in overlayfs, ubuntu local root : http://seclists.org/oss-sec/2015/q2/717

Références III

11. /proc & /sys : Enforcing mount options for sysfs and proc : https://lwn.net/Articles/647757/
12. Linux Kernel Exploitation - Earning Its Pwnie a Vuln at a Time : https://jon.oberheide.org/files/source10-linuxkernel-jonoberheide.pdf
13. Exploiting the Linux kernel : Measures and countermeasures : https://jon.oberheide.org/files/syscan12-exploitinglinux.pdf
14. rkt : https://coreos.com/blog/rocket/

Références IV

15. Container mechanics in rkt and Linux : http://events.linuxfoundation.org/sites/events/files/slides/Container%20mechanics%20in%20rkt%20and%20Linux.pdf

16. rkt architecture : https://coreos.com/rkt/docs/latest/devel/architecture.html

17. App Container - Rocket : https://www.socallinuxexpo.org/sites/default/files/presentations/appc%20%2B%20rocket%20(SCALE%2013x).pdf

18. Using Virtual Machines to Improve Container Security with rkt v0.8.0 : https://coreos.com/blog/rkt-0.8-with-new-vm-support/

Références V

19. App Container basics : https://coreos.com/rkt/docs/latest/app-container.html
20. Using rkt with systemd : https://coreos.com/rkt/docs/latest/using-rkt-with-systemd.html
21. Image Fetching Behavior : https://coreos.com/rkt/docs/latest/image-fetching-behavior.html
22. rkt vs other projects : https://coreos.com/rkt/docs/latest/rkt-vs-other-projects.html
23. LXC CVE-2013-6441 : http://www.cvedetails.com/cve/CVE-2013-6441/
24. Security issue in LXC (CVE-2015-1335) : http://www.openwall.com/lists/oss-security/2015/09/29/4

Références VI

25. Before you initiate a “docker pull” : https://securityblog.redhat.com/2014/12/18/before-you-initiate-a-docker-pull/
26. Docker 1.8 : https://blog.docker.com/2015/08/content-trust-docker-1-8/
27. The Update Framework (TUF) : https://theupdateframework.github.io/
28. Docker Registry : https://github.com/docker/distribution
29. Over 30% of Official Images in Docker Hub Contain High Priority Security Vulnerabilities : http://www.banyanops.com/blog/analyzing-docker-hub/

Références VII

30. Docker 1.3.2 - Security Advisory [24 Nov 2014] : http://www.openwall.com/lists/oss-security/2014/11/24/5
31. Docker 1.3.3 - Security Advisory [11 Dec 2014] : https://groups.google.com/forum/#!msg/docker-user/nFAz-B-n4Bw/0wr3wvLsnUwJ
32. Docker 1.6.1 - Security Advisory [150507] : http://seclists.org/fulldisclosure/2015/May/28
33. Are Docker containers really secure ? https://opensource.com/business/14/7/docker-security-selinux

Références VIII

34. Bringing new security features to Docker : https://opensource.com/business/14/9/security-for-docker
35. False Boundaries and Arbitrary Code Execution : https://forums.grsecurity.net/viewtopic.php?f=7&t=2522
36. CAP_SYS_ADMIN : the new root : https://lwn.net/Articles/486306/
37. CoreOS Introduces Clair : Open Source Vulnerability Analysis for your Containers : https://coreos.com/blog/vulnerability-analysis-for-containers/

Références IX

38 Security compliance of RHEL7 Docker containers : https://www.open-scap.org/resources/documentation/security-compliance-of-rhel7-docker-containers/

39 Container Security with SELinux and CoreOS : https://coreos.com/blog/container-security-selinux-coreos.html

40 Announcing rkt v0.7.0, featuring a new build system, SELinux and more : https://coreos.com/blog/rkt-0.7.0-with-selinux-and-new-build-system.html

Références X

41. Docker SELinux Security Policy : https://access.redhat.com/documentation/en/red-hat-enterprise-linux-atomic-host/7/container-security-guide/chapter-6-docker-selinux-security-policy

42. Docker Security : Using Containers Safely in Production : https://www.openshift.com/promotions/docker-security.html

43. Docker security : https://docs.docker.com/engine/articles/security/

Références XI

44. Why we don’t let non-root users run Docker in CentOS, Fedora, or RHEL : http://www.projectatomic.io/blog/2015/08/why-we-dont-let-non-root-users-run-docker-in-centos-fed
45. CoreOS : https://coreos.com/
46. What Trusted Computing Means to Users of CoreOS and Beyond : https://coreos.com/blog/coreos-trusted-computing.html
47. Making Sense of Container Standards and Foundations : OCI, CNCF, appc and rkt : https://coreos.com/blog/making-sense-of-standards.html
48. Project Atomic : http://www.projectatomic.io/

Références XII

49. Kubernetes : http://kubernetes.io/
50. Kubernetes user guide - Pods : http://kubernetes.io/v1.1/docs/user-guide/pods.html
51. App Container Spec - Pods : https://github.com/appc/spec/blob/master/spec/pods.md#app-container-pods-pods
52. fleet : https://coreos.com/using-coreos/clustering/
53. etcd : https://coreos.com/etcd/
54. flannel : https://coreos.com/flannel/docs/latest/
55. Docker Swarm : https://docs.docker.com/swarm/

Références XIII

56 Docker Compose : https://docs.docker.com/compose/

57 ClusterHQ Flocker : https://clusterhq.com/flocker/introduction/

58 OpenShift Origin : https://www.openshift.org/

59 Containers & Docker : How Secure Are They ? : http://blog.docker.io/2013/08/containers-docker-how-secure-are-they/