Mini-Internet using LXC (MI-LXC) : un générateur d'environnement réseau simulé

François Lesueur
Univ Lyon, INSA Lyon
Département Télécommunications, Services et Usages
F-69621 Villeurbanne, France
prenom.nom@insa-lyon.fr
https://github.com/flesueur/mi-lxc

***Résumé*—MI-LXC est un *framework* pour construire des infrastructures de TP. MI-LXC suit l'approche *infrastructure-as-code* pour pouvoir programmer la topologie et la construction des différents éléments du système. Cette construction est modulable, l'exemple actuel permettant de créer un SI offrant des services web, du mail, de l'authentification centralisée, des postes clients ainsi que des éléments extérieurs à ce SI. MI-LXC est diffusé sous licence libre AGPL.**

I. Introduction

L'enseignement pratique de la sécurité des réseaux et des systèmes se heurte à la complexité de proposer des systèmes d'étude adaptés. Sans un système suffisamment complexe, un simple TP *firewall*, par exemple, peut se résumer à appliquer la syntaxe iptables (ou manipuler l'interface d'une *appliance*) et analyser/constater les effets de ports bloqués. Hors, selon moi, l'intérêt d'un TP firewall réside d'abord, aujourd'hui, à réfléchir et proposer un cloisonnement adapté d'un système donné, plus fin que le périmétrique WAN/LAN/DMZ historique, puis à l'implémenter dans un langage quelconque (iptables ou autre). Cette réflexion ne peut se mener qu'au sein d'un système à la fois suffisamment complexe et suffisamment maîtrisé par les étudiants.

La réponse aujourd'hui classique à ce problème est l'utilisation de plateformes virtualisées, offrant aux étudiants un espace de jeu regroupant plusieurs VM interconnectées par des réseaux virtuels. Pour l'avoir pratiquée pendant plusieurs années, cette approche souffre d'un énorme défaut de lourdeur : chaque mise à jour est complexe, chaque changement de partie de sujet nécessite souvent une telle mise à jour, le redéploiement doit être refait. Cette lourdeur, de plus, limite l'adaptation de la plateforme à différents objectifs successifs et, par conséquent, limite la maîtrise que les étudiants pourront en avoir lors de son utilisation.

Dans cet article, je présente MI-LXC, un *framework* python afin de programmer la construction de telles plateformes. Cette programmation permet une mise à jour plus simple, une plus grande adaptabilité, une gestion de versions ainsi qu'une diffusion beaucoup plus simple. L'utilisation de conteneurs LXC au lieu de machines virtuelles (VirtualBox ou VmWare) permet de plus une consommation de ressources bien moindre (et donc la simulation de systèmes beaucoup plus grands), tout en offrant un usage similaire pour les étudiants.

II. Objectifs pédagogiques et technologiques

Ayant longtemps utilisé une approche avec des machines et réseaux virtuels VirtualBox, cette section se concentre sur les objectifs de MI-LXC comme incrément au-delà de l'utilisation de VM plus classiques.

A. Échelle des systèmes manipulés

Les machines physiques modernes permettent sans difficulté d'héberger 5-10 VM assez légères, le tout avec 2-3 Go de RAM. Cette étape est déjà une très grosse évolution par rapport à un seul système par poste mais, après plusieurs années à manipuler de tels systèmes, j'avais envie de voir plus grand. La sécurité des réseaux est un problème qui devient complexe quand le système l'est, et je souhaite donc progressivement complexifier les systèmes manipulés afin de mieux illustrer les concepts. Le choix de LXC comme support permet de gagner plusieurs ordres de grandeur sur la taille des systèmes simulés.

B. Adaptabilité à des scénarios divers

Pour pouvoir être exploitée au maximum, la plateforme visée doit idéalement pouvoir être utilisée sur un grand nombre de TP. En effet, chaque nouvelle plateforme induit un coût de prise en main pour les étudiants, à la fois côté outillage et côté système simulé. L'utilisation d'une même plateforme entre plusieurs matières permet de mutualiser ce coût de découverte des outils. De plus, si les systèmes simulés sont proches, alors il devient possible d'étudier plus finement des systèmes complexes, qui seront découverts au travers de nombreux TP et non d'un seul qui obligerait à simplifier énormément les problèmes.

Cette approche oblige à mutualiser un squelette générique et offrir une adaptation à moindre coût. MI-LXC se configure à travers un fichier qui décrit la topologie réseau visée puis des scripts et des modèles pour peupler les systèmes créés. Typiquement, nous travaillons actuellement à son usage pour illustrer le routage BGP entre différents AS.

C. Maintenabilité

Pour l'avoir expérimenté, il est très difficile de versionner, sauvegarder, faire évoluer une infrastructure virtuelle type VirtualBox. J'ai au fur et à mesure des années accumulé

quelques dizaines de giga-octets d'exports datés, difficiles à réutiliser. Chaque mise à jour ou ajout de paquet demande des modifications sur plusieurs machines, puis leur export (en re-minimisant la taille des disques), leur déploiement, une nouvelle sauvegarde… Le Go ne coûte certes pas très cher mais cette méthode de gestion a 20 ans de retard et cet usage illustrait ma *dette technique* sur le sujet.

MI-LXC utilise le concept d'*infrastructure-as-code* et déploie une infrastructure similaire à partir d'un code (python + bash + fichiers spécifiques) d'environ 1Mo. Ce code, en format texte et non binaire, est facilement versionné avec git.

D. Facilité de diffusion

La difficulté de diffusion peut tenir à deux aspects : la mise à disposition et la capacité d'utilisation par des étudiants (ou autres). La dernière version de mon infrastructure avec Virtualbox nécessite de diffuser/récupérer une archive de 4 Go, contenant 7 VM qui une fois extraites nécessitent plus de 15 Go d'espace disque disponible. En effet, bien que ces images dérivent toutes d'un même master en ne conservant que les deltas, chaque évolution/mise à jour fait dériver de ce master qu'il n'est pas possible de ≪ rattraper ≫. L'expérience en TP m'a montré que c'est souvent cet espace disque nécessaire qui a empêché les étudiants d'installer facilement cette infrastructure sur leur poste (cette infra était, bien sûr, disponible sur les machines de TP).

MI-LXC est disponible publiquement sur github (sous licence AGPL) et est donc facile à récupérer par des tiers. La recréation systématique de l'infrastructure permet de bien mieux mutualiser le disque master : l'infrastructure d'exemple crée actuellement 11 conteneurs (équivalent VM) pour moins de 4 Go, chaque conteneur consommant entre 60 Mo et 500 Mo.

E. Un peu de swag !

Enfin, nous formons nos étudiants à l'agilité, au *devops*, au *serverless* (sic) et je pense qu'une infrastructure de TP obéit à des concepts partiellement similaires. Je précise partiellement car l'étude des outils existants m'a montré les limites de ce point. Moderniser cette approche me paraissait donc cohérent avec le reste de la formation et intéressant personnellement. L'approche ouverte et abordable en terme de volume de code a permis à un étudiant de proposer des améliorations de l'interface, ce qui est un élément très motivant.

III. Choix technologiques

La construction de ce *framework* implique des choix sur essentiellement trois aspects : le choix de l'outil de virtualisation, la création des machines virtuelles et leur configuration (*provisioning*).

A. Outils de virtualisation

La première possibilité est d'utiliser un outil de virtualisation avec hyperviseur, allant d'ESXi à un simple VirtualBox. Dans ce cadre, les VMs ont une très bonne indépendance, les outils sont mûrs et le ressenti utilisateur est très proche d'une vraie machine (qui est de toutes façons, aujourd'hui, souvent elle-même virtualisée !). L'inconvénient majeur est le coût en ressources, essentiellement côté RAM. Mon objectif est de pouvoir monter progressivement à plusieurs dizaines (centaines ?) de VM sur des postes de TP classiques. À l'opposé, je veux aussi pouvoir faire tourner un TP à 10 VMs sur un simple portable d'étudiants, qui n'a pas toujours aujourd'hui 4 Go de RAM.

La possibilité retenue est d'utiliser un modèle de conteneurs, beaucoup plus légers. Les conteneurs partagent le même noyau mais vivent dans un espace de nommage indépendant, incluant une arborescence complète et leurs propres interfaces réseau. Sous Linux, les conteneurs sont aujourd'hui essentiellement LXC et Docker, qui partagent les mêmes primitives fournies par le noyau. Docker est beaucoup plus populaire mais est pensé pour lancer un *process* dans un espace indépendant et non pas un système entier ; la composition des process se fait ensuite avec docker-compose, dans une approche différente de la combinaison fournie par une machine classique. Il est a priori possible de lancer un processus d'init qui lance à son tour un système (bien qu'il y ait des incompatibilités avec systemd, il existe des alternatives dédiées), mais c'est une utilisation très spécifique allant contre la philosophie fondamentale de l'outil, ce qui ne manquerait pas de poser des difficultés. LXC, lui, initialise un processus d'*init* et lance donc un système *userspace* complet plus classique.

Le choix s'est donc arrêté sur LXC car il permet un passage à l'échelle intéressant tout en offrant aux étudiants un usage similaire à un système classique.

B. Création des machines virtuelles

À ma connaissance, l'outil principal de génération automatisée de VM est *vagrant*. Vagrant permet d'instancier un grand nombre d'images de VM. Cependant, son interaction avec LXC passe par un plugin dont la maintenance ne semble plus assurée, ce qui pose un problème de pérennité.

LXC, contrairement à VirtualBox par exemple, intègre cependant directement l'outillage basique nécessaire à la création d'un conteneur pour un OS donné. Il est ainsi possible en une ligne de commande de demander la création d'un conteneur Debian, version stable, architecture x64. MI-LXC utilise donc directement cette primitive fournie par la bibliothèque LXC.

C. Configuration des machines virtuelles

Une fois les conteneurs génériques instanciés, il faut les configurer, installer des paquets, copier des fichiers spécifiques… Un certain nombre d'outils existent, dont par exemple *ansible* ou *puppet*. Leur usage n'a pas été retenue car ils couvrent, selon ma compréhension, des difficultés qui ne se posent pas ici.

Le *provisioning de masse* qu'ils adressent inclut deux problèmes : spécifier les configurations souhaitées d'une part, les appliquer de manière cohérente au fur et à mesure de leur évolution sur une masse de systèmes. Pour MI-LXC, il n'y a pas d'évolution une fois l'infrastructure créée, elle sera plutôt détruite puis regénérée pour un autre TP (elle n'entre

pas en « production »). Le seul problème restant est donc de spécifier la configuration souhaitée, pour lequel l'essentiel du travail est similaire que l'on utilise un outil dédié ou pas. MI-LXC appelle finalement directement des scripts shell dédiés, notamment au travers d'un mécanisme de *templates* permettant de mutualiser des éléments de configuration répétitifs.

D. Projets similaires

Un premier groupe de projets similaires est composé de Marionnet [1] (utilisant User-Mode Linux) et Internet simulator [2] (utilisant LXC). Tous deux permettent de simuler de très grands systèmes et démontrent la viabilité de l'approche par conteneurs. Par contre, ils sont centrés sur l'interconnexion réseau d'hôtes issus d'une bibliothèque prédéfinie sans considérer la construction de ces images, construction qui est au contraire l'aspect central de MI-LXC. Ces outils sont, en fait, conçus pour l'enseignement du réseau (services, routage) plutôt que de la sécurité.

Un second groupe de projets similaires est composé de Dockernet [3] et Kathara [4] (anciennement NetKit, qui était basé sur UML mais n'est plus maintenu). Tous deux utilisent Docker. Ils sont conçus et utilisés pour des TP de sécurité mais, comme décrit dans la sous-section III-A, l'outil Docker est prévu pour lancer un unique process et non un système entier. Ainsi, Dockernet et Kathara permettent d'expérimenter certains points précis de sécurité mais ne permettent pas de simuler un système complet de machines physiques interconnectées, chaque machine exécutant un OS complet avec ses pivots potentiels entre services, ce qui est un objectif de MI-LXC. Ils n'intègrent pas non plus nativement la création d'images docker personnalisées, nécessitant ainsi d'autres pièces pour obtenir le *workflow* complet.

Enfin, Labtainers [5] est le projet qui semble le plus proche. Il est par ailleurs maintenu, beaucoup plus mûr et propose déjà de nombreux scénarios. Labtainers utilise depuis juillet 2018 une image docker de base démarrant avec systemd (`solita/ubuntu-systemd`), ce qui illustre bien l'intérêt d'avoir un init pour un tel projet. Cependant, cette image (non officielle) n'est plus maintenue depuis août 2018, ce qui illustre également les difficultés et incompatibilités entre systemd et docker. Par exemple, les images officielles Debian et Ubuntu n'incluent pas d'init, alors que Centos le propose, en supprimant toutefois un certain nombre de cibles de systemd. Labtainers aurait pu être un bon candidat, les inconvénients résident comme expliqué dans le choix de docker ainsi que dans la complexité générale de l'outil, très complet. La raison de développer MI-LXC est également chronologique, l'intégration de l'image docker incluant systemd à Labtainers étant arrivée après que MI-LXC commence à être fonctionnel.

MI-LXC a pour objectif d'offrir un environnement simple permettant de programmer la création des systèmes qui seront exécutés. Ces créations sont automatisées depuis des images originales fournies par LXC et un mécanisme de *templates* permet de mutualiser les parties communes.

1. https://marionnet.org
2. https://github.com/nsec/the-internet
3. https://github.com/gmiotto/dockernet
4. http://www.kathara.org/
5. https://my.nps.edu/web/c3o/labtainers

IV. Configuration et usage de MI-LXC

MI-LXC est écrit en python dans `mi-lxc.py` et repose sur deux éléments principaux pour générer l'infrastructure cible : `setup.json` pour décrire la topologie et `files/` pour gérer les scripts de provisioning. Les fichiers décrits ici ainsi qu'un manuel d'utilisation plus détaillé peuvent être consultés sur la page du projet https://github.com/flesueur/mi-lxc.

A. Script mi-lxc.py

Le développement est réalisé en mode PM [6] dans un fichier python unique. Ce script traduit la configuration attendue en appelant le binding python de la bibliothèque LXC. Ce binding est officiel et maintenu.

B. Fichier setup.json

Le fichier setup décrit la topologie réseau en JSON. Chaque conteneur y est décrit avec son nom, ses interfaces réseaux et leurs IP, sa passerelle de sortie et les *templates* éventuels à lui appliquer. Le fichier permettant de décrire la topologie de mon TP actuel composé de 11 conteneurs, internes et externes au SI, fait 83 lignes de description.

C. Dossier files/

Le dossier `files/` contient, pour chaque conteneur ou *template*, la recette permettant de le configurer. L'essentiel du travail de conception d'une infrastructure se situe à ce niveau. Pour chaque conteneur, lors de sa création, le script `<container>/provision.sh` sera exécuté et a pour charge de spécifier la configuration, copier des fichiers, initialiser des home…

D. Usage

Une fois tout ceci réalisé, l'infrastructure est générée en appelant `./mi-lxc.py create` puis démarrée avec `./mi-lxc.py start`. Les utilisateurs peuvent obtenir un accès shell à chaque conteneur avec `./mi-lxc.py attach [user]@<container> [command]` ou graphique avec
`./mi-lxc.py display [user]@<container>`

V. Exemple de maquette pédagogique associée

J'ai pour l'instant utilisé MI-LXC pour supporter 3 des 4 TP d'une matière de 32h « Sécurité des Réseaux et des Systèmes ».

L'infrastructure générée est composée de machines Debian complètes :

- un pare-feu
- une DMZ
- un serveur applicatif interne
- un serveur de fichiers
- un serveur d'authentification centralisée

6. http://programming-motherfucker.com/

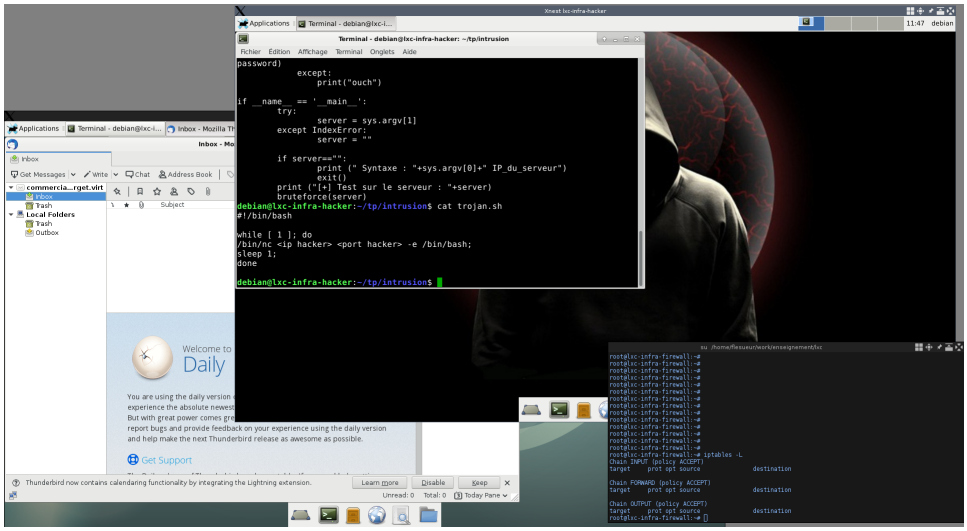

FIGURE 1. Exemple d'utilisation en TP

— 3 postes internes (admin, commercial, développeur)
— 2 postes externes (un attaquant, un poste au domicile d'un employé)
— un backbone de routage

Les sujets d'étude portent sur :

— Intrusion, dans lequel les étudiants découvrent l'infrastructure en jouant le rôle de l'attaquant (brute-force, reverse-shell, mapping, rebond)
— Firewall, dans lequel les étudiants, à partir de la découverte des services lors du TP précédent, proposent et implémentent un cloisonnement adapté
— IDS, dans lequel les étudiants analysent quels éléments peuvent être obtenus à quels points du réseau (HIDS, NIDS, corrélation)

L'usage de cette infrastructure, proposant notamment un accès X11 aux différentes machines, est illustré figure 1.

VI. DIFFICULTÉS D'EXPLOITATION

Le test auprès des étudiants a été concluant mais pas toujours immédiatement réussi. LXC a un comportement *presque* homogène entre les versions et distributions, ce qui a posé quelques difficultés, a priori actuellement traitées (de nouvelles apparaîtront probablement avec l'évolution des versions) : MI-LXC a été testé sous Debian, Kali, Arch et Ubuntu.

LXC ne fonctionne par ailleurs que sous Linux. Par inclusivité pour les utilisateurs de moins bons OS, un fichier vagrant est inclus dans le dépôt github permettant, sur tous les OS après l'installation de vagrant, de générer une machine VirtualBox incluant MI-LXC. Le surcoût est faible puisqu'une seule VM VirtualBox permet ensuite de lancer l'infrastructure LXC complète. J'utilise par ailleurs cette approche pour créer l'image VirtualBox que je déploie sur les postes de TP fournis dans les salles.

Enfin, nous avons rencontré quelques difficultés avec des ports bloqués par eduroam. Il s'agit d'un problème pour obtenir des clés PGP de vérification des images lors de la création initiale des conteneurs par LXC. Ce problème, résolu en ajoutant une variable d'environnement, illustre cependant la difficulté a débugger les problèmes éventuels côté LXC, souvent assez peu verbeux.

VII. CONCLUSION ET PERSPECTIVES

MI-LXC est un *framework* pour concevoir des infrastructures de TP virtualisées. Il permet une meilleure maintenabilité, évolution et passage à l'échelle qu'une approche basée sur la création ad hoc de VM VirtualBox. Je l'ai utilisé avec succès cette année.

L'objectif est maintenant d'essayer d'y intégrer des TP de réseau, par exemple de routage BGP, ainsi qu'un TP de déploiement cryptographique où l'on pourrait mettre en œuvre les attaques réseau de type MitM. Cela créerait des passerelles entre différents cours, augmentant la cohérence pédagogique globale et rentabilisant le temps de compréhension de la plateforme par les étudiants en début de TP.

Un autre objectif serait de pouvoir fournir des infrastructures de départ pour des projets étudiants. Par exemple, cette année, des étudiants ont pu y déployer un serveur ELK et intégrer les alertes des sondes réparties à travers l'infrastructure.

Enfin, la reconfiguration réseau d'une infrastructure déjà déployée et l'intégration de tests unitaires et d'intégration sont également des point qui seront amenés à être développés.

MI-LXC est disponible sous licence AGPL (réutilisation libre, licence contaminante, les modification doivent être publiées, y compris si utilisé dans le cadre d'un service auquel on peut accéder à distance) à l'URL suivante : https://github.com/flesueur/mi-lxc. En plus du libre usage, les contributions sont donc également les bienvenues, par exemple portant sur la construction de topologies/systèmes adaptés à d'autres apprentissages.