

PROJET PKI

Pour ce projet mettre en place une machine virtuelle Linux. Personnellement je travaille sur une machine virtuelle VirtualBox Ubuntu Serveur avec 2 cartes réseaux en mode NAT (et pas Réseau NAT). Si vous souhaitez savoir comment créer une machine sur VirtualBox cf. [ce document](#) (même si depuis la version de VirtualBox a changé).

Tous les fichiers sont disponibles dans [mon dossier Git dans le dossier 00-Code](#). Tous les scripts sont à exécuter en sudo.

Vidéo explicative du projet dans le fichier [02-Video.pdf](#), le sommaire est disponible en description de la vidéo si vous souhaitez regarder que certaines parties.

Ordre d'exécution des scripts :

- 1- *A la main : faire l'étape 1 de ce document.*
- 2- `creation_conteneur.sh nbr_clients`
- 3- `creation_data.sh`
- 4- `installation_mqtt.sh`
- 5- `paho_sur_lxc.sh`
- 6- `start_ecoute_mqtt`
- 7- `certificat_pki` (installe les paquets, exécute le script python qui permet de générer le certificat sur le serveur et copie ce certificat sur les clients)
- 8- `start_echange` qui appelle `echange_sym`

I. Création des conteneurs (Client + Serveur)

J'ai décidé de mettre en place un script Bash qui permet de créer de 2 à 8 clients plus un serveur LXC Ubuntu. Le script prend un seul paramètre, n, le nombre de clients.

1. Mise en place du réseau

Ces entités vont communiquer à l'aide d'un bridge. On attribue une carte réseau de l'hôte à ce bridge.

a. Création du bridge

(Attention à l'indentation, la copie depuis ce document peut générer des problèmes)

```
nano /etc/netplan/*.yaml
```

```
network:
  ethernets:
    enp0s3:
      dhcp4: true
    enp0s8:
      dhcp4: false

  bridges:
    br0:
      dhcp4: true
      interfaces:
        - enp0s8
  version: 2
```

netplan apply

Pour vérifier les configurations rentrer la commande « ip a ». La seconde carte réseau ne devrait plus apparaître puisqu'elle est « absorbée » par le bridge br0. Voici ce que j'ai obtenu :

```
adm_jma@ubuntu:~$ ip a
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN group default qlen 1000
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
        valid_lft forever preferred_lft forever
    inet6 ::1/128 scope host
        valid_lft forever preferred_lft forever
2: enp0s3: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc fq_codel state UP group default qlen 1000
    link/ether 08:00:27:6e:ec:95 brd ff:ff:ff:ff:ff:ff
    inet 10.0.2.15/24 metric 100 brd 10.0.2.255 scope global dynamic enp0s3
        valid_lft 86377sec preferred_lft 86377sec
    inet6 fe80::a00:27ff:fe6e:ec95/64 scope link
        valid_lft forever preferred_lft forever
3: enp0s8: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc fq_codel master br0 state UP group default qlen 1000
    link/ether 08:00:27:52:71:eb brd ff:ff:ff:ff:ff:ff
4: br0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue state UP group default qlen 1000
    link/ether 1e:4e:5d:d9:f7:1b brd ff:ff:ff:ff:ff:ff
    inet 10.0.3.15/24 metric 100 brd 10.0.3.255 scope global dynamic br0
        valid_lft 86377sec preferred_lft 86377sec
    inet6 fe80::1c4e:5dff:fed9:f71b/64 scope link
        valid_lft forever preferred_lft forever
```

b. Téléchargement de LXC

apt-get update

apt install lxc lxc-templates -y

Attention ! Les scripts suivants supposent que vous n'avez pas d'autres conteneurs de créés.

c. Configuration de LXC sur l'hôte

Changer le fichier de configuration « etc/default/lxc-net » pour attribuer notre bridge br0 aux conteneurs. Voici à quoi le fichier de configuration devrait ressembler après modifications :

```
# containers. Set to "false" if you'll use virbr0 or another existing
# bridge, or mactan to your host's NIC.
USE_LXC_BRIDGE="false"

# If you change the LXC_BRIDGE to something other than lxcbr0, then
# you will also need to update your /etc/lxc/default.conf as well as the
# configuration (/var/lib/lxc/<container>/config) for any containers
# already created using the default config to reflect the new bridge
# name.
# If you have the dnsmasq daemon installed, you'll also have to update
# /etc/dnsmasq.d/lxc and restart the system wide dnsmasq daemon.
USE_LXC_BRIDGE="true"
LXC_BRIDGE="br0"
LXC_ADDR="10.0.3.1"
LXC_NETMASK="255.255.255.0"
LXC_NETWORK="10.0.3.0/24"
LXC_DHCP_RANGE="10.0.3.2,10.0.3.254"
LXC_DHCP_MAX="253"
LXC_DHCP_CONF=""
LXC_DOMAIN=""
# Uncomment the next line if you'd like to use a conf-file for the lxcbr0
```

Il faut également modifier le fichier « » ainsi :

```
GNU nano 6.2 /etc/lxc/default.conf
lxc.net.0.type = veth
lxc.net.0.link = br0
lxc.net.0.flags = up
lxc.net.0.hwaddr = 00:16:3e:xx:xx:xx
```

2. Création du script de création des conteneurs

./creation_conteneur.sh nombre_clients : permet de créer les conteneurs LXC Ubuntu.

./creation_data.sh : permet de créer un fichier par client/serveur qui comprend l'adresse IP du conteneur et son nom.

Vous trouverez dans le dossier `src/debug` des fichiers bash permettant d'aider les tests. Ils sont appelés par les scripts du dossier `src`. Ce sont un peu des « fonctions » permettant d'alléger les fichiers principaux.

3. Mise en place des échanges par MQTT

`./installation_mqtt.sh` : met à jour les conteneurs et télécharge les modules nécessaires à la mise en place de MQTT. La librairie choisie est Mosquitto en Python.

Puis création des script Python « abonné » et « éditeur » dans le dossier `src_py`. Utilisation de la librairie Paho qui permet de mettre en place les échanges MQTT en Python. On copie ces deux scripts sur les conteneurs avec le script `paho_sur_lxc.sh`. Puisqu'il n'est pas possible de récupérer l'expéditeur d'un message en MQTT les messages envoyés sont des JSON. Une partie du message est le nom de l'expéditeur, l'autre le corps du message. Pour la suite du projet nous allons donc seulement chiffrer la deuxième partie du message.

II. PKI

1. Le serveur produit un certificat autosigné X509

Nous utilisons la classe [pyca/cryptography](#) de python pour générer le certificat autosigné X509.

Le script `certificat_pki.sh` installe les librairies nécessaires, génère un certificat auto-signé X509 sur le serveur (la PKI) qu'on copie en dur sur les clients avec sa clef publique.

Les clients créent leur propre clé privée et leur propre demande de signature de certificat (CSR). Ils ne révèlent pas leur clé privée à l'autorité de certification (PKI).

Les clients remettent leur CSR à la PKI et celle-ci leur renvoie un certificat signé. L'envoi de ce fichier se fait toujours dans le script : `certificat_pki.sh`. La PKI réceptionne les CSR et génère donc les certificats. Les certificats ont généralement une durée de validité d'un an, bien qu'une autorité de certification accorde généralement quelques jours supplémentaires pour des raisons de commodité.

Le serveur renvoie donc le certificat signé ainsi que la signature aux clients. Les clients stockent leur certificat signé ainsi que leur signature dans leur dossier « `secure` ».

2. Echanges

Les échanges sont générés aléatoirement grâce au script `choix_msg.py`.

a. Si les clients ne se connaissent pas encore

Echange asymétrique

Celui qui initie l'échange envoie sa signature et son certificat à son destinataire.

Le destinataire vérifie la validité du certificat qu'il reçoit et extrait la clef publique.

Il génère ensuite un secret (mot de 32 bits, encrypté avec AES 128 et la clef publique de l'expéditeur) qu'il envoie à son expéditeur.

Ce dernier déchiffre ce secret et envoie un message d'acquiescement à son destinataire pour le prévenir qu'il a bien récupéré le secret.

Les échanges symétriques sont désormais disponibles.

b. Les clients se connaissent

Echange symétrique

Le secret est partagé entre les deux clients, le fichier `SYM_OK.txt` est présent dans leur dossier `secure`. L'expéditeur a seulement besoin de crypter son message avec le secret qu'il partage avec son destinataire. Il l'envoie et son destinataire le décrypte de son côté.

L'historique des échanges des messages est stocké dans le fichier `historique.txt` sur chaque client.