Rapport pour le projet de sécurité (RESG4)

Auteurs

- Michaluk David 49762@etu.he2b.be (mailto:49762@etu.he2b.be)
- El Amouri Meihdi 49262@etu.he2b.be (mailto:49262@etu.he2b.be)
- Azoud Ismael 42394@etu.he2b.be (mailto:42394@etu.he2b.be)
- Rossitto Nicolas 49282@etu.he2b.be (mailto:49282@etu.he2b.be)

Prérequis

- `debian`
- `LibreSSL ^2.6.4`

Installation des dépendances

- installer `nodejs` (et `npm`)

```
# Using Debian, as root
curl -sL https://deb.nodesource.com/setup_8.x | bash -
apt-get install -y nodejs
apt-get install -y build-essential
```

Référence:
https://github.com/nodesource/distributions/blob/master/README.md#de

- installer `docker`

```
# mettre à jour la liste des paquets et de leur dépendances
sudo apt-get update
```

```
# installer les dépendances de docker
sudo apt-get install \
    apt-transport-https \
    ca-certificates \
    curl \
    gnupg2 \
    software-properties-common
```

```
# ajouter la clé de docker au gestionnaire de paquets debian
curl -fsSL https://download.docker.com/linux/debian/gpg |
sudo apt-key add -
```

```
# ajouter la repository stable de debian au gestionnaire de
paquets de debian
sudo add-apt-repository \
   "deb [arch=amd64] https://download.docker.com/linux/debian
\
   $(lsb_release -cs) \
   stable"
```

```
# mettre à jour la liste des paquets et de leur dépendances
sudo apt-get update
```

```
# installer docker community edition, l'utilitaire de ligne
de commande et le logiciel containerd
sudo apt-get install docker-ce docker-ce-cli containerd.io
```

- tester si docker a bien été installé

```
# lancer une image de test
sudo docker run hello-world
```

Référence: https://docs.docker.com/install/linux/docker-ce/debian/

- installer docker-compose

```
# télécharger l'exécutable docker-compose
sudo curl -L
"https://github.com/docker/compose/releases/download/1.24.0/d
ocker-compose-$(uname -s)-$(uname -m)" -o
/usr/local/bin/docker-compose
```

Référence: https://docs.docker.com/compose/install/

Initialisation du projet

Le projet est divisé en deux parties: le client et le serveur

Il nous faut avant-tout générer les clés et certificats de l'autorité de certification et du serveur.

Les opérations suivantes doivent s'effectuer dans le sous-dossier `server/pki`

```
mkdir -p server/pki && cd server/pki
```

Les étapes qui suivent couvrent la génération des différentes clés privées et certificats.
Vous trouverez déjà tout le nécessaire dans les dossiers `server/pki` et `client/pki` et pouvez passer à l'étape `Installation et lancement du serveur`, néanmoins dans l'hypothèse où cette suite logicielle devrait être mise en production, il faudrait impéartivement regénérer les clés et certificats en suivant les étapes décrites ci-dessous afin de garantir une sécurité optimale.

Générer les clés et certificats de l'autorité de certification

- générer la clé privée de l'autorité de certification

```
openssl genrsa -out ca.key 2048
```

- générer le certificat de l'autorité de certification

```
openssl req -new -x509 -days 365 -key ca.key -subj
"/C=BE/ST=Brussels/L=Brussels/O=ESI/CN=RESG4 Root CA" -out
ca.crt
```

Générer les clés et certificats du serveur

- générer la clé privée et le certificat du serveur

```
openssl req -newkey rsa:2048 -nodes -keyout server.key -subj
"/C=BE/ST=Brussels/L=Brussels/O=ESI/CN=localhost" -out
server.csr
```

- signer la demande de signature de certificat du serveur avec le certificat de l'autorité de certification

Attention : l'adresse IP présente dans le champ subjectAltName est à adapter éventuellement, elle doit correspondre à l'adresse sur laquelle le client se connectera

```
openssl x509 -req -extfile <(printf "subjectAltName=IP:127.0.0.1") -days 365 -in server.csr -CA ca.crt -CAkey ca.key -CAcreateserial -out server.crt
```

Mettre à disposition le certificat du serveur au client

A exécuter à la racine du dossier contenant les deux programmes

```
mkdir -p client/pki && cp server/pki/ca.crt client/pki
```

Installation et lancement du serveur

Démarrer la base de donnée

```
docker-compose up -d
```

Créer la base de donnée vide

```
docker-compose exec db sh -c 'echo "CREATE DATABASE IF NOT EXISTS \`package-manager\`" | mysql -u root -presg4'
```

Installation des dépendances du serveur

```
npm install
```

Démarrage du serveur

```
npm start
```

Activer un compte client

```
docker-compose exec db sh -c 'echo "UPDATE \`package-manager\`.user SET enabled = 1 WHERE login = \"david123\"" | mysql -u root -presg4'
```

Autoriser un compte client à soumettre des paquets

```
docker-compose exec db sh -c 'echo "UPDATE \`package-manager\`.user SET canUploadPackages = 1 WHERE login = \"david123\"" | mysql -u root -presg4'
```

Installation et lancement du client

Installation des dépendances du client

```
npm install
```

Démarrage du client

Les opérations suivantes sont à exécuter dans le dossier `client`

```
cd client
```

Créer un compte client

```
./node_modules/.bin/ts-node ./src/index.ts register <user> <password>
# exemple
./node_modules/.bin/ts-node ./src/index.ts register david123 David19121995333
```

Installer un paquet (et ses dépendances)

```
./node_modules/.bin/ts-node ./src/index.ts -u <user> -p <password> install [packages...]
# exemple
./node_modules/.bin/ts-node ./src/index.ts -u david123 -p David19121995333 install debian@1.0.0 some@3.0.0
```

Désinstaller un paquet (spécifier la version est obligatoire)

```
./node_modules/.bin/ts-node ./src/index.ts -u <user> -p
<password> uninstall [packages...]
# exemple
./node_modules/.bin/ts-node ./src/index.ts -u david123 -p
David19121995333 uninstall debian@1.0.0 some@3.0.0
```

Mettre à jour un paquet

Une fois la mise à jour terminée, le système propose de supprimer les paquets jugés "inutiles"

```
./node_modules/.bin/ts-node ./src/index.ts -u <user> -p
<password> update [packages...]
# exemple
./node_modules/.bin/ts-node ./src/index.ts -u david123 -p
David19121995333 update curl 7zip wget
```

Envoyer un paquet sur le serveur (si autorisation accordée, voir ci-dessus)

Le paquet à envoyer doit se trouver dans le dossier packages à la racine du dossier du programme du client
Les dépendances du paquet doivent se trouver au même niveau que ce dernier.
Exemple pour un paquet nommé debian dont nous souhaitons envoyer la version 1.0.0 sur le serveur, le paquet debian@1.0.0 dépend de curl@3.0.0 et wget@2.0.0:

```
client
+-- packages/
    +-- wget/
        +-- 2.0.0/
            +-- package-descriptor.json
    +-- curl/
        +-- 3.0.0/
            +-- package-descriptor.json
    +-- debian/
        +-- 1.0.0/
            +-- package-descriptor.json
```

Le contenu du fichier package-descriptor.json du dossier debian/1.0.0 doit au moins contenir les propriétés suivantes (format JSON):

```
{
      "version":"1.0.0",
      "__package__":{
         "name":"debian"
      },
      "__dependencies__":[
         {
            "version":"3.0.0",
            "__package__":{
               "name":"curl"
            }
         },
         {
            "version":"2.0.0",
            "__package__":{
               "name":"wget"
            }
         }
      ]
   }
```

La même logique doit être suivie pour les fichiers package-descriptor.json des autres paquets. Un exemple complet est fourni dans le dossier packages du client.

La commande pour envoyer ce paquet sur le serveur sera la suivante:

```
./node_modules/.bin/ts-node ./src/index.ts -u <user> -p
<password> upload <packageName> <version>
# exemple
./node_modules/.bin/ts-node ./src/index.ts -u david123 -p
David19121995333 upload debian 1.0.0
```

Supprimer un paquet sur le serveur

```
./node_modules/.bin/ts-node ./src/index.ts -u <user> -p
<password> remove <packageName> <version>
# exemple
./node_modules/.bin/ts-node ./src/index.ts -u david123 -p
David19121995333 remove debian 1.0.0
```

Points de sécurité traités

Intégrité, confidentialité et authenticité

Les communications entre les clients et le serveur se font toujours de manière chiffrée à l'aide du protocole de sécurisation TLS.

Les clients disposent du certificat de clé publique du serveur. Nous partons du principe que celui-ci a été fourni par un canal jugé fiable avant toute communication par le réseau entre les deux parties.

Le serveur écoute sur deux ports:

- le port 3003: les clients ne disposant pas de certificat client se connectent à ce port afin de s'enregistrer. Les communications sont d'emblée chiffrée à l'aide de TLS et limitées seulement à l'enregistrement de l'utilisateur. Pour s'enregistrer, le client fourni un login unique sur le serveur, un mot de passe respectant des critères précis (voir ci-dessous) ainsi qu'une demande de signature de certificat.
  Le client soumet donc son certificat à signature au serveur. Le serveur injecte le login dans le certificat du client avant de le signer numériquement grâce à son pouvoir d'autorité de certification. Les futures communications entre le client et le serveur se feront dorénavant sur le second port.

- le port 3000: seul les clients disposant d'un certificat signé par le serveur ont l'autorisation de se connecter via ce port. Le serveur propose le reste des fonctionnalités du gestionnaire du paquet via ce port (soumission et suppression de paquets si l'utilisateur dispose des permissions nécessaires, installation si l'utilisateur est activé). Pour se connecter sur ce port, l'utilisateur doit fournir un couple login et mot de passe valide ainsi qu'un certificat signé par le serveur contenant exactement le même login en guise de métadonnée que celui soumis avec le mot de passe.

Il est donc impossible pour un utilisateur de se connecter avec un login et un mot de passe s'il ne dispose pas du certificat analogue qui a été signé par le serveur pour le login en question car ce login est directement stocké dans le certificat client.

De plus, la clé privée du client ainsi que la demande de signature du certificat client est générée par le client et non par le serveur, évitant ainsi la transmission de la clé privée sur le réseau et assurant l'identité du client.

Attaque par injection de code

Pour toute communication avec la base de donnée, nous utilisons les `PreparedStatement` (requêtes préparées) dont le format est prédéfini avant toute insertion de donnée utilisateurs, prévenant ainsi toute forme d'injection de code de type SQL.

Il en est de même pour la signature du certificat client par le serveur: la signature se faisant à l'aide d'un appel système via `shell`, le certificat du client est non seulement vérifié à l'aide d'une expression régulière afin de s'assurer qu'il ne contient pas de caractère interdit mais est aussi passé à l'utilitaire `openssl` au travers de l'entrée standard `stdin` afin d'éviter toute injection de code au niveau de *bash* (en lieu et place d'être passée sous forme de chaîne de caractère en temps que paramètre `shell` directement en ligne de commande).

Attaque par déni de service

Le serveur dispose d'un système de limitation de taux de requêtes par utilisateur.
Celui-ci limite le nombre de requêtes à 100 par 15 minutes par utilisateur, au-delà, le système considère les requêtes à l'API comme étant une tentative d'attaque par déni de service et renverra une erreur HTTP 429 au client.

Attaque par dictionnaire

Afin de rendre plus difficile l'attaque par dictionnaire, le système impose certaines règles pour la définition du mot de passe lors de l'inscription, à savoir:

- être de longueur 8 minimum

- disposer d'au moins de deux des caractéristiques suivantes: être composé d'une lettre minuscule, être composé d'une lettre majuscule, être composé d'un chiffre

Ces caractéristiques à respecter ont été définies en tenant compte du meilleur compris entre sécurité et confort de mémorisation pour l'utilisateur.

Une attaque par dictionnaire est d'autant plus compliquée étant donné que l'attaquant devra également trouver la clé privée de l'utilisateur et le certificat client, ce qui est très peu probable avec la puissance de calcul actuelle, sans lesquels il lui est impossible de se connecter.

Stockage des mots de passes

Afin d'être préparé à toute éventualité, les mots de passes sont hachés à l'aide de `bcrypt` (avec salage) avant d'être stockés dans la base de donnée. Dans le cas d'une fuite de donnée, les mots de passes sont ainsi surtout protégés dans l'intérêt des utilisateurs qui auraient utilisés un login et mot de passe identiques que pour un autre programme. Si un pirate arrive à trouver la valeur de départ d'un hash, cela ne lui permettra pas pour autant de se connecter car il lui manquerait le certificat client.