CLASSCORD — MESSAGERIE SECURISEE MULTI-CLIENTS

Institution : Mediaschool Nice — IRIS

---

1. CONTEXTE ET OBJECTIF

Dans le cadre du BTS SIO option SISR, ce projet consiste à déployer, sécuriser et maintenir un serveur de messagerie instantanée multi-clients basé sur Python. L'objectif est de fournir aux étudiants une plateforme de tchat interne, isolée et sécurisée, adaptée à un environnement pédagogique.

Le projet se déroule sur 5 jours et couvre les aspects suivants :

- Mise en place d'un environnement Debian virtualisé
- Déploiement du serveur de messagerie Python
- Conteneurisation via Docker
- Sécurisation des accès (pare-feu UFW, Fail2ban, authentification SHA-256)
- Journalisation des événements et automatisation de la sauvegarde

2. ARCHITECTURE GLOBALE

L'architecture repose sur une machine virtuelle Debian hébergeant le serveur Classcord. Les clients Python se connectent au serveur via le réseau local ou pédagogique sur le port 12345.

Composant Rôle Adresse / Port

VM Classcord (Debian) Serveur de messagerie Python IP privée hôte

Serveur Python Moteur de messagerie multi-clients Port 12345 TCP

Conteneur Docker Isolation du service Port 12345:12345

Base SQLite Persistance des données utilisateurs Locale au conteneur

Pare-feu UFW Contrôle des accès réseau Règle 12345/tcp

Configuration réseau de la VM Debian :

- Adaptateur 1 : NAT (accès Internet)
- Adaptateur 2 : Réseau privé hôte (communication avec les clients)

3. TECHNOLOGIES UTILISEES

Technologie Version / Détail

Python 3 Langage de développement du serveur de messagerie

SQLite Base de données légère pour la persistance des données

Docker Conteneurisation du serveur

Docker Compose Orchestration du conteneur

UFW Pare-feu système (Uncomplicated Firewall)

Fail2ban Outil de détection et blocage des tentatives d'intrusion

systemd Gestion du service en tant que daemon système

Debian 11 Système d'exploitation hôte de la VM

VirtualBox Hyperviseur de type 2

SHA-256 Algorithme de hachage des mots de passe

4. DESCRIPTION DES COMPOSANTS

4.1 Serveur Python (server_classcord.py)

Le serveur Python gère les connexions multi-clients simultanées via des sockets TCP. Il implémente les fonctionnalités suivantes :

- Authentification des utilisateurs avec hachage SHA-256 des mots de passe
- Gestion de canaux de discussion : #général, #dev, #admin
- Diffusion des messages uniquement aux clients présents dans le même canal
- Stockage persistant des comptes utilisateurs (fichier users.pkl ou base SQLite)
- Interface d'administration pour la gestion des utilisateurs et des droits
- Export CSV des données

4.2 Conteneur Docker

Le serveur est conteneurisé à partir d'une image Python 3.11-slim. Le Dockerfile installe les dépendances et expose le port 12345. Le fichier docker-compose.yml orchestre le déploiement avec les volumes nécessaires pour la persistance des données et des logs.

Structure du Dockerfile :

- Image de base : python:3.11-slim
- Répertoire de travail : /app
- Port exposé : 12345
- Commande de démarrage : python server_classcord.py

4.3 Service systemd

Un fichier de service systemd (classcord.service) assure le démarrage automatique du serveur au boot du système et sa relance automatique en cas de défaillance. Les logs sont redirigés vers /var/log/classcord/classcord.log.

4.4 Pare-feu UFW

UFW est configuré pour autoriser uniquement les connexions sur le port 12345/tcp. Tous les autres ports entrants non nécessaires sont bloqués par défaut.

5. FONCTIONNEMENT TECHNIQUE

5.1 Connexion d'un client

1. Le client Python se connecte au serveur sur l'IP de la VM et le port 12345

2. Le serveur demande les identifiants (nom d'utilisateur et mot de passe)

3. Le mot de passe est haché en SHA-256 et comparé à la valeur stockée

4. En cas de succès, le client est placé dans un canal de discussion par défaut

5. Les messages du client sont broadcastés aux autres membres du même canal

5.2 Gestion des canaux

Les canaux disponibles sont définis dans le code serveur :

- #général : canal principal accessible à tous les utilisateurs
- #dev : canal dédié aux discussions techniques
- #admin : canal réservé aux utilisateurs administrateurs

5.3 Journalisation

Les événements sont journalisés via le module logging de Python. Chaque connexion, déconnexion, tentative d'authentification et message envoyé est horodaté et enregistré dans le fichier de log. La rotation des logs est gérée par logrotate (rotation quotidienne, conservation 7 jours).

6. SECURITE

6.1 Authentification

Les mots de passe des utilisateurs sont hachés en SHA-256 avant stockage. Aucun mot de passe n'est conservé en clair dans la base de données.

6.2 Pare-feu UFW

Les règles UFW limitent les connexions entrantes au strict nécessaire. Le port 12345/tcp est autorisé pour le service de messagerie ; tous les autres ports entrants non définis sont rejetés.

6.3 Fail2ban

Fail2ban surveille les tentatives d'authentification échouées dans les logs du serveur. Après 3 tentatives échouées en 5 minutes, l'adresse IP source est bannie pendant 1 heure.
Configuration Fail2ban :

- Filtre : détection du pattern [LOGIN] Failed login attempt from <HOST>
- maxretry : 3 tentatives
- findtime : 300 secondes
- bantime : 3600 secondes

6.4 Sauvegarde automatisée

Une tâche cron effectue une copie horaire du fichier users.pkl (données utilisateurs) vers un répertoire de sauvegarde avec horodatage.

7. DEPLOIEMENT ET CONFIGURATION

Pré-requis :

- Machine virtuelle Debian 11 sous VirtualBox
- Docker et Docker Compose installés
- Python 3 installé (en cas d'utilisation sans Docker)

Procédure de déploiement avec Docker :

1. Créer la VM Debian avec les deux adaptateurs réseau (NAT + Réseau privé hôte)

2. Mettre à jour le système : sudo apt update && sudo apt upgrade -y

3. Installer les outils : sudo apt install -y git curl python3 python3-pip ufw docker.io docker-compose

4. Cloner le dépôt du projet

5. Dans le répertoire classcord-docker, exécuter : docker compose up --build -d

6. Configurer UFW : sudo ufw allow 12345/tcp && sudo ufw enable

7. Installer et configurer Fail2ban selon la configuration décrite en section 6.3

8. Créer le service systemd si déploiement hors Docker

9. Mettre en place la tâche cron de sauvegarde

Vérification du déploiement :

- Contrôle du statut du conteneur : docker compose ps
- Test de connexion depuis un client Python sur le port 12345

8. TESTS ET VALIDATION

Tests réalisés sur 5 jours :

Jour 1 : Création de la VM Debian, clonage du projet, exécution locale du serveur, test multi-clients, configuration UFW

Jour 2 : Déploiement réseau, création du service systemd, conteneurisation Docker

Jour 3 : Mise en place de la journalisation Python, configuration logrotate, déploiement Fail2ban, automatisation de la sauvegarde

Jour 4 : Ajout de la gestion des canaux, persistance SQLite, interface d'administration, export CSV

Jour 5 : Tests finaux de sécurité et validation de l'ensemble des fonctionnalités

Résultats obtenus :

- Serveur de messagerie opérationnel en mode multi-clients
- Authentification SHA-256 fonctionnelle
- Canaux de discussion opérationnels
- Conteneurisation Docker fonctionnelle
- Fail2ban actif et bannissement testé
- Sauvegarde automatisée opérationnelle