Rapport Technique : Système de Détection d'Attaques Man-in-the-Middle (MITM) sur Flux TCP

Généré par l'Assistant IA

31 décembre 2025

Table des matières

1 Introduction

La sécurité des communications réseau constitue un enjeu majeur dans les infrastructures informatiques modernes. Les attaques de type *Man-in-the-Middle* (MITM) représentent une menace critique, permettant à un attaquant de s'interposer discrètement entre deux entités communicantes afin d'intercepter, modifier ou retarder les échanges.

Ce projet vise à concevoir un système expérimental de détection d'attaques MITM sur des flux TCP, basé sur l'analyse comportementale du trafic. L'approche repose sur l'observation d'anomalies telles que la latence excessive, la perte de trames et le désordonnancement des messages. L'objectif principal n'est pas la prévention, mais la détection active d'un comportement suspect au niveau applicatif.

2 Objectifs et Portée du Projet

Le projet se concentre sur la mise en œuvre d'une architecture client–serveur TCP instrumentée, intégrant un proxy simulant différentes attaques MITM.

Les objectifs principaux sont :

— Mettre en place une communication TCP fiable et persistante.
— Simuler un attaquant intermédiaire capable d'altérer le trafic.
— Développer des mécanismes de détection côté serveur.
— Garantir la reproductibilité via la conteneurisation Docker.

La portée du projet est volontairement limitée à un environnement contrôlé et ne vise pas à couvrir les communications chiffrées de type TLS.

3 Architecture Globale

L'architecture du système repose sur trois composants distincts, chacun exécuté dans un conteneur Docker indépendant :

1. **Client TCP** : génère un flux de messages structurés.
2. **Proxy MITM** : intercepte et manipule le trafic selon le mode d'attaque sélectionné.
3. **Serveur de Détection** : analyse les messages reçus afin d'identifier des anomalies.

Les conteneurs communiquent via un réseau Docker privé (`lab_net`), garantissant un environnement isolé et reproductible pour les expérimentations.

4 Implémentation TCP en Python

L'implémentation repose sur la bibliothèque standard `socket` de Python. Un protocole applicatif simple a été défini afin de faciliter l'analyse des flux.

4.1 Initialisation des Sockets

```
def connect(self):
    self.socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    self.socket.connect((self.host, self.port))
    self.logger.info(f"Connected to proxy at {self.host}:{self.port}")
```

Listing 1 – Connexion du client au proxy

```
self.server_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
self.server_socket.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, 1)
self.server_socket.bind((self.host, self.port))
self.server_socket.listen(1)
```

Listing 2 – Initialisation du serveur TCP

4.2 Structure des Messages

Chaque message transmis respecte le format suivant :

`SEQ=<numéro>|TS=<timestamp>|DATA=<payload>`

- `SEQ` : numéro de séquence incrémental.
- `TS` : horodatage Unix de l'envoi.
- `DATA` : charge utile.

5 Simulation MITM via Proxy

Le proxy MITM agit comme un relais intermédiaire entre le client et le serveur. Il supporte plusieurs modes d'attaque configurables et utilise le multi-threading pour gérer les flux bidirectionnels.

```
def _forward(self, source, destination, direction):
    while True:
        data = source.recv(self.buffer_size)
        if not data:
            break
        processed = self._process_data(data)
        if processed:
            destination.sendall(processed)
```

Listing 3 – Boucle de transfert du proxy

5.1 Modes d'Attaque

Les modes implémentés sont :
- **Transparent** : relais sans modification.
- **Random Delay** : ajout d'une latence aléatoire.
- **Drop** : suppression aléatoire de trames.
- **Reorder** : réordonnancement des messages.

6 Conteneurisation Docker

Docker est utilisé afin d'assurer l'isolation et la portabilité du système. Le fichier `docker-compose.yml` orchestre les trois services.
- `tcp_client` : génère le trafic.
- `tcp_proxy` : intercepte les communications.
- `tcp_server` : analyse et détecte les anomalies.

7 Logique de Détection

Le serveur implémente quatre mécanismes de détection principaux :

7.1 Détection de Latence

Une alerte est déclenchée si le délai entre l'envoi et la réception dépasse un seuil prédéfini.

7.2 Détection de Perte de trames

Une discontinuité dans les numéros de séquence indique une perte de trames.

7.3 Détection de Désordre

La réception d'une trame avec un numéro de séquence inférieur au numéro attendu révèle un réordonnancement.

7.4 Détection de violation d'intégrité

La réception d'une trame qui ne respecte pas la structure convenu par le client et le serveur.

8 Limitations et Perspectives

Les principales limitations du système sont :
— Absence de chiffrement des communications.
— Détection basée sur des seuils fixes.
Les perspectives d'évolution incluent l'intégration de TLS, l'utilisation de modèles statistiques ou d'apprentissage automatique, ainsi qu'un tableau de bord de visualisation en temps réel.

9 Conclusion

Ce projet démontre la faisabilité de la détection d'attaques MITM par analyse comportementale des flux TCP. L'architecture modulaire basée sur Docker et Python permet de simuler efficacement différents scénarios d'attaque et constitue une base solide pour des travaux futurs en sécurité des réseaux.