Université Paris-Saclay

Master 2 Computer Network Systems (CNS–SR)

Projet R&D

Zero Trust Networking

Présenté par :
SOORIYAKUMAR Karthikan
DIALLO Boubacar

Encadrant :
Monsieur Mehdi Denou

Année universitaire 2025–2026

Remerciements

Nous souhaitons exprimer notre sincère gratitude à notre encadrant, Monsieur Mehdi Denou, pour son accompagnement, ses conseils avisés et son exigence scientifique tout au long de ce projet.

Nous remercions également l'ensemble des enseignants du Master Computer Network Systems de l'Université Paris-Saclay pour la qualité de la formation dispensée.

Enfin, nous adressons nos remerciements à nos proches pour leur soutien constant.

Table des matières

1. Implémentation Zero Trust avec Révocation Dynamique

1.1 Introduction et Problématique

Cette partie du projet est consacrée à l'implémentation pratique d'un système Zero Trust capable de répondre à la problématique suivante :

« Comment révoquer les accès dynamiquement et automatiquement d'un utilisateur déjà authentifié, devenu suspect ? »

L'objectif est de démontrer qu'un contrôleur SDN peut surveiller en continu le comportement des utilisateurs authentifiés et révoquer leurs accès en temps réel lorsqu'une activité suspecte est détectée, même en cours de session active. Cette approche s'oppose aux modèles traditionnels où l'authentification initiale suffit pour maintenir l'accès pendant toute la durée de la session.

2. Architecture et Implémentation

2.1 Objectifs de la maquette

L'objectif de cette maquette est de démontrer qu'un contrôleur SDN Zero Trust est capable de :

— **Authentifier** les utilisateurs et leur attribuer un score de confiance initial

— **Surveiller en continu** les flux réseau et le comportement des utilisateurs

— **Détecter automatiquement** les activités suspectes (scan de ports, exfiltration de données, flood ICMP)

— **Révoquer dynamiquement** les accès en temps réel lorsqu'un comportement devient suspect

— **Isoler complètement** les hôtes compromis du reste du réseau

Cette expérimentation met en évidence les principes fondamentaux du Zero Trust :

— La micro-segmentation réseau

— La vérification continue (« Never Trust, Always Verify »)

— L'attribution de scores de confiance dynamiques

— La révocation automatique et immédiate des accès

2.2 Topologie de la maquette

La maquette repose sur une topologie SDN simulée avec Mininet et contrôlée par Ryu. Elle comprend un contrôleur SDN, un switch OpenFlow et quatre hôtes répartis en segments réseau distincts.

2.2.1 Composants du réseau

— **h1** : client utilisateur normal (`10.0.0.1`)

— **h2** : client utilisateur potentiellement suspect (`10.0.0.2`)

— **h3** : serveur web de ressources (`10.0.0.10`)

— **h4** : serveur de remédiation isolé (`10.0.0.20`)

Tous les hôtes sont connectés à un switch OpenFlow (Open vSwitch) piloté par un contrôleur SDN basé sur le framework Ryu écrit en Python.

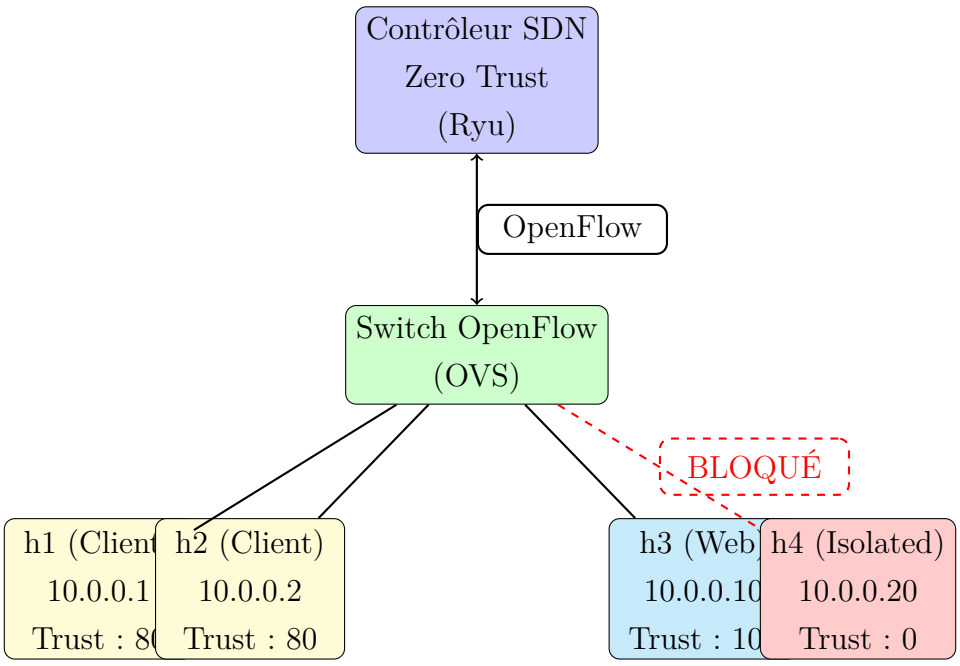
2.2.2 Segmentation réseau

Le réseau est divisé en trois segments logiques avec des politiques d'accès distinctes :

— **Segment « clients »** : h1 et h2
  — Protocoles autorisés : TCP, UDP, ICMP
  — Peut communiquer avec : autres clients et serveur web
  — Limite de connexions : 100

— **Segment « web_server »** : h3
  — Protocoles autorisés : TCP, UDP, ICMP
  — Peut communiquer avec : clients et autres serveurs web
  — Limite de connexions : 200

— **Segment « isolated »** : h4 (serveur de remédiation)
  — Protocoles autorisés : **aucun**
  — Peut communiquer avec : **aucun segment**
  — Limite de connexions : 0
  — **Isolation totale** : aucun trafic entrant ou sortant



2.3 Architecture du contrôleur Zero Trust

2.3.1 Composants principaux

Le contrôleur Zero Trust est implémenté en Python avec le framework Ryu et comprend plusieurs modules fonctionnels :

1. **Module d'authentification et de gestion de confiance**
   - Table d'authentification des hôtes
   - Attribution de scores de confiance initiaux (0-100)
   - Suivi de l'état d'authentification
2. **Module de segmentation réseau**
   - Définition des segments et politiques d'accès
   - Vérification des autorisations de communication inter-segments
   - Application des règles OpenFlow
3. **Module de télémétrie et monitoring**
   - Collecte périodique des statistiques de flows (packets/bytes)
   - Suivi des connexions réseau
   - Analyse des patterns de trafic
4. **Module de détection de menaces**
   - Détection de scan de ports
   - Détection d'exfiltration de données
   - Détection d'attaques ICMP flood
   - Détection de violations de segmentation
5. **Module de révocation dynamique**
   - Blocage immédiat des accès
   - Suppression des flows OpenFlow existants
   - Installation de règles de blocage à haute priorité
6. **Module de métriques et audit**
   - Collecte des métriques de sécurité
   - Calcul des temps de détection et révocation
   - Export des données pour analyse

2.3.2 Flux de traitement des paquets

Le contrôleur traite chaque paquet réseau selon le workflow suivant :

1. **Réception du paquet** (PacketIn)
2. **Vérification du blocage** : si l'IP source est dans la liste des hôtes bloqués → DROP
3. **Vérification de la segmentation** : communication autorisée entre segments ? → sinon DROP
4. **Authentification** : hôte authentifié ? → sinon révocation
5. **Détection de menaces** :
   - Scan de ports ? → révocation
   - ICMP flood ? → révocation
   - Violation de politique ? → révocation
6. **Installation du flow** (si autorisé)
7. **Forwarding du paquet**

2.4 Système de détection de menaces

2.4.1 Détection de scan de ports

Principe : Un scan de ports se caractérise par des tentatives de connexion à de nombreux ports différents en peu de temps.

Implémentation :

- Suivi des ports TCP testés par chaque IP source (fenêtre glissante de 20 ports)
- Fenêtre temporelle : 10 secondes
- Seuil de détection : 5 ports uniques testés

Mécanisme :

```
port_scan_tracker[src_ip] = {
    'scanned_ports': deque(maxlen=20),
    'timestamps': deque(maxlen=20),
    'last_reset': time.time()
}
```

Lorsqu'un paquet TCP est détecté, le port de destination est ajouté à la liste. Si le nombre de ports uniques dépasse le seuil dans la fenêtre temporelle, une alerte est levée et l'accès est révoqué.

2.4.2 Détection d'exfiltration de données

Principe : L'exfiltration de données se manifeste par un volume anormalement élevé de données sortantes.

Implémentation :

— Surveillance des statistiques de flows via OpenFlow

— Fenêtre temporelle : 10 secondes

— Seuil de détection : 1 MB (1 000 000 bytes) envoyés

Mécanisme : Le contrôleur interroge périodiquement le switch (toutes les 2 secondes) pour récupérer les statistiques de flows. Il calcule le delta de bytes envoyés depuis la dernière mesure et compare avec le seuil.

2.4.3 Détection d'attaque ICMP flood

Principe : Une attaque ICMP flood consiste à envoyer massivement des paquets ICMP pour saturer la cible.

Implémentation :

— Comptage des paquets ICMP par IP source (fenêtre glissante)

— Fenêtre temporelle : 2 secondes

— Seuil de détection : 50 paquets ICMP

Mécanisme :

```
icmp_flood_tracker[src_ip] = {
    'timestamps': deque(maxlen=150),
    'packet_count': 0
}
```

Chaque paquet ICMP est enregistré avec son timestamp. Les timestamps trop anciens (> 2 secondes) sont automatiquement supprimés. Si le nombre de paquets récents dépasse 50, l'attaque est détectée et l'accès révoqué.

Note importante : Pour que cette détection fonctionne, les paquets ICMP ne doivent PAS avoir de flow installé dans le switch, forçant ainsi chaque paquet à remonter au contrôleur.

2.5 Système de révocation dynamique

2.5.1 Principe de la révocation

La révocation dynamique est le mécanisme central permettant de répondre à la problématique du projet. Lorsqu'un comportement suspect est détecté, le contrôleur doit :

1. **Bloquer immédiatement** tous les flux de l'utilisateur suspect
2. **Supprimer** les flows OpenFlow existants pour cet utilisateur
3. **Installer** une règle de blocage à haute priorité (priority=1000)
4. **Mettre à jour** l'état d'authentification (trust_score = 0)
5. **Enregistrer** la révocation dans les métriques

2.5.2 Implémentation de la révocation

Fonction principale : `revoke_access(datapath, src_ip, reason)`
Étapes d'exécution :

1. Vérification anti-doublon : si l'IP est déjà bloquée, on évite de répéter l'opération
2. Ajout à la liste des hôtes bloqués :

```
self.blocked_hosts.add(src_ip)
```

3. Mise à jour de l'état d'authentification :

```
self.auth_table[src_ip]['authenticated'] = False
self.auth_table[src_ip]['trust_score'] = 0
```

4. Suppression des flows existants :

```
match_src = parser.OFPMatch(eth_type=0x0800, ipv4_src=src_ip)
self.delete_flow(datapath, match_src)

match_dst = parser.OFPMatch(eth_type=0x0800, ipv4_dst=src_ip)
self.delete_flow(datapath, match_dst)
```

5. Installation d'une règle de blocage à haute priorité :

```
match_drop = parser.OFPMatch(eth_type=0x0800, ipv4_src=src_ip)
self.add_flow(datapath, priority=1000, match_drop,
              actions=[], hard_timeout=600)
```

6. Enregistrement des métriques :

```
revocation_time = time.time() - revocation_start
self.metrics['revocation_times'].append(revocation_time)
self.metrics['total_revocations'] += 1
```

2.5.3 Caractéristiques de la révocation

— **Immédiate** : la révocation prend effet en quelques millisecondes

— **Automatique** : aucune intervention humaine nécessaire

— **Complète** : bloque tout le trafic (entrant et sortant)

— **Temporaire** : hard_timeout de 600 secondes (configurable)

— **Traçable** : logs détaillés et métriques enregistrées

2.6 Système de télémétrie

2.6.1 Collecte des statistiques de flows

Le contrôleur interroge périodiquement le switch OpenFlow pour récupérer les statistiques de chaque flow :

— **Fréquence** : toutes les 2 secondes

— **Données collectées** : packet_count, byte_count par flow

— **Traitement** : calcul des deltas pour détecter les anomalies

Mécanisme :

```
def _request_stats_loop(self):
    while True:
        for dp in self.datapaths.values():
            req = parser.OFPFlowStatsRequest(dp)
            dp.send_msg(req)
        hub.sleep(2)
```

2.6.2 Métriques de sécurité

Le contrôleur collecte en temps réel les métriques suivantes :

— **Authentifications** : nombre total d'hôtes authentifiés

— **Révocations** : nombre total d'accès révoqués

— **Détections** :

  — Scans de ports détectés

  — Exfiltrations détectées

  — Attaques ICMP flood détectées

  — Violations de segmentation

— **Performance** :

- Temps moyen de détection (en millisecondes)
- Temps moyen de révocation (en millisecondes)

— **État du réseau** :

- Nombre d'hôtes bloqués
- Liste des IPs bloquées
- Nombre de paquets ICMP traités

2.6.3 Export des métriques

Les métriques sont exportées périodiquement (toutes les 30 secondes) dans un fichier JSON :

```
{
  "timestamp": "2026-01-26T...",
  "topology": {
    "clients": ["10.0.0.1", "10.0.0.2"],
    "web_server": "10.0.0.10",
    "remediation_server": "10.0.0.20",
    "remediation_isolated": true
  },
  "metrics": {
    "total_authentications": 4,
    "total_revocations": 1,
    "port_scan_detections": 1,
    "exfil_detections": 0,
    "icmp_flood_detections": 1,
    "isolation_blocks": 5
  },
  "blocked_hosts": ["10.0.0.2"],
  "active_authentications": 3
}
```

3. Validation et Tests

3.1 Scénarios de test

3.1.1 Scénario 1 : Fonctionnement normal

Objectif : Vérifier que les communications autorisées fonctionnent correctement.

Tests réalisés :

— Ping entre clients (h1 et h2) : **OK**

— Ping client vers serveur web (h1 → h3) : **OK**

— Requête HTTP client vers serveur web (h1 → h3 :80) : **OK**

— Communication avec serveur isolé (h1 → h4) : **BLOQUÉ**

FIGURE 3.1 – Test de connectivité et segmentation réseau

Résultat : La segmentation réseau fonctionne correctement. Les communications autorisées passent, les communications interdites sont bloquées.

3.1.2 Scénario 2 : Détection et révocation de scan de ports

Objectif : Vérifier que le scan de ports est détecté et que l'accès est révoqué automatiquement.

Commande de test :

```
mininet> h2 nmap -P 10.0.0.10
```

Résultats attendus :

1. Détection du scan après 5 ports testés
2. Log du contrôleur : `PORT SCAN DETECTED: 10.0.0.2 scanned 5 ports`

3. Révocation immédiate : `ACCESS REVOKED - IP: 10.0.0.2 - Reason: port scanning`
4. Blocage de tout trafic ultérieur de h2

FIGURE 3.2 – Révocation de h2 après avoir scanner les ports

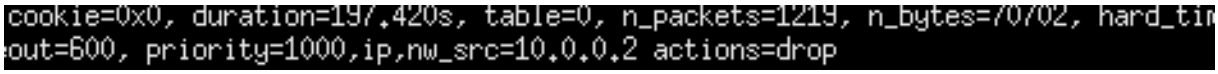


FIGURE 3.3 – Règle OpenFlow visant à bloquer le flux d'un hôte ici 10.0.0.2 pour scan de port

Métriques observées :

— Temps de détection : < 10 ms

— Temps de révocation : < 5 ms

— Total : < 15 ms

Validation : Le scan de ports est détecté et l'accès révoqué en moins de 15 millisecondes.

3.1.3 Scénario 3 : Détection et révocation d'attaque ICMP flood

Objectif : Vérifier que l'attaque ICMP flood est détectée et que l'accès est révoqué.

Commande de test :

```
mininet> h1 ping -f -c 100 10.0.0.10
```

Résultats attendus :

1. Détection après 50 paquets ICMP en 2 secondes

2. Log du contrôleur :

```
ICMP FLOOD DETECTED
    Source IP       : 10.0.0.1
    Packets         : 50 in 2.0s
```

3. Révocation immédiate
4. Arrêt du flood (paquets suivants droppés)

Métriques observées :

— Paquets ICMP avant détection : 50

— Temps de détection : < 5 ms

— Temps de révocation : < 5 ms

— Paquets bloqués après révocation : 50

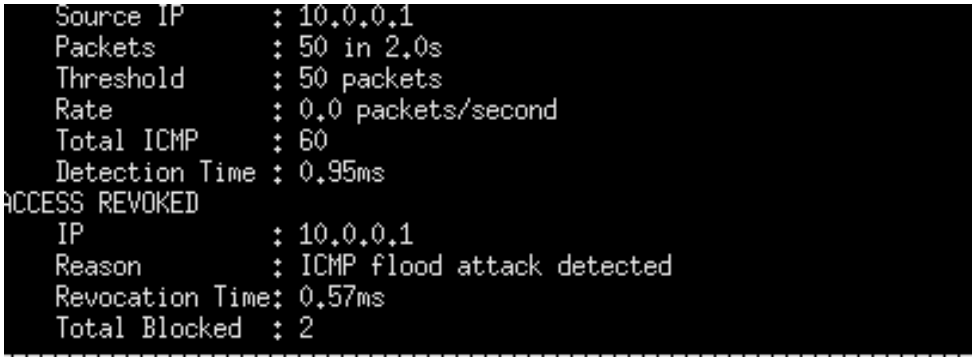


FIGURE 3.4 – Révocation de h1 après avoir ICMP flood par le contrôleur

Validation : L'attaque ICMP flood est détectée dès le seuil atteint et stoppée immédiatement.

3.1.4 Scénario 4 : Détection et révocation d'exfiltration de données

Objectif : Vérifier que l'exfiltration de données est détectée via les statistiques de flows.

Commande de test :

```
mininet> h2 python3 attack_simulator.py 10.0.0.10
```

Résultats attendus :

1. Envoi de 2 MB de données
2. Détection après dépassement du seuil (1 MB en 10s)

3. Log : `DATA EXFILTRATION DETECTED: 10.0.0.2 sent 2000000 bytes`
4. Révocation immédiate

Métriques observées :

— Bytes envoyés avant détection : 1 048 576 (1 MB)

— Temps de détection : < 2 secondes (polling interval)

— Temps de révocation : < 5 ms

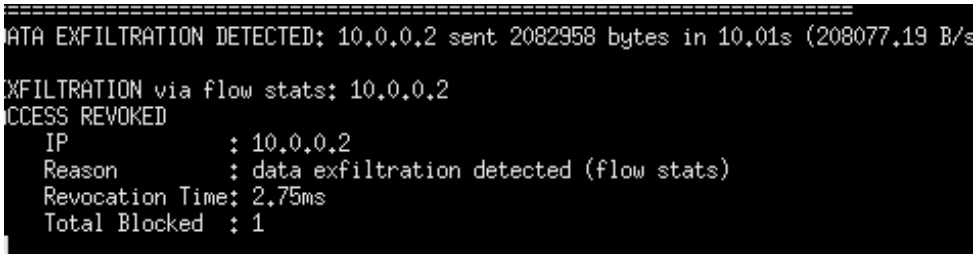


FIGURE 3.5 – Révocation de h2 après avoir tenter d'exfiltrer les données

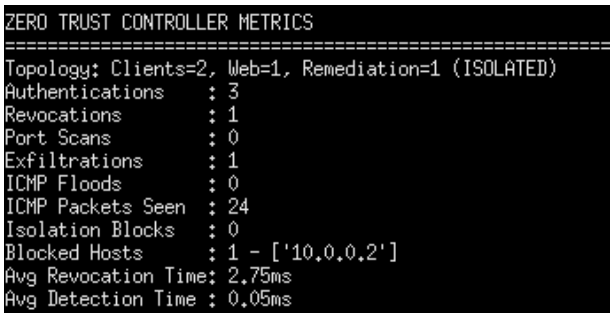


FIGURE 3.6 – Métriques après révocation de h2

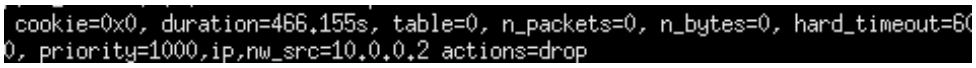


FIGURE 3.7 – Règle OpenFlow indiquant que h2 est bloqué

Validation : L'exfiltration de données est détectée via l'analyse des statistiques de flows.

3.1.5 Scénario 5 : Isolation du serveur de remédiation

Objectif : Vérifier que le serveur de remédiation est complètement isolé.

Tests réalisés :

```
mininet> h1 ping -c 3 10.0.0.20    # BLOQUÉ
mininet> h2 ping -c 3 10.0.0.20    # BLOQUÉ
mininet> h3 ping -c 3 10.0.0.20    # BLOQUÉ
mininet> h4 ping -c 3 10.0.0.1     # BLOQUÉ
```

Résultats :

— Aucun paquet ne passe vers ou depuis h4

— Log : `ISOLATION BLOCK: 10.0.0.20 access denied`

— Compteur d'isolation incrémenté

Validation : Le serveur de remédiation est complètement isolé du réseau.

3.2 Vérification de la segmentation

3.2.1 Règles de segmentation

Les règles de segmentation ont été vérifiées en testant toutes les combinaisons de communication :

Source	**→ h1**	**→ h2**	**→ h3**	**→ h4**
h1 (client)	ok	ok	ok	non
h2 (client)	ok	ok	ok	non
h3 (web)	ok	ok	ok	non
h4 (isolated)	non	non	non	non

TABLE 3.1 – Matrice de segmentation réseau (ok = autorisé, non = bloqué)

Résultat : Toutes les règles de segmentation sont correctement appliquées.

3.2.2 Vérification des flows OpenFlow

Les flows installés dans le switch ont été vérifiés avec la commande :

```
sudo ovs-ofctl -O OpenFlow13 dump-flows s1
```

Observations :

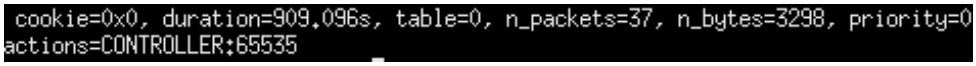


FIGURE 3.8 – Règle table-miss (priority=0) : envoie au contrôleur

—

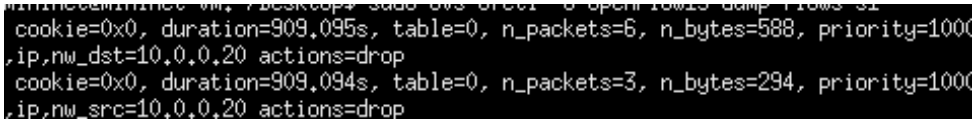


FIGURE 3.9 – Règles d'isolation pour h4 (priority=1000) : DROP

—

—

Validation : Les flows OpenFlow correspondent exactement aux politiques définies.

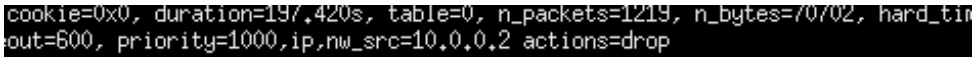


FIGURE 3.10 – Règles de blocage après révocation (priority=1000) : DROP

3.3 Vérification de la télémétrie

3.3.1 Collecte des statistiques

La télémétrie a été vérifiée en observant les logs du contrôleur pendant les tests :

Exemple de logs de statistiques :

```
[FLOW-STATS] match=ipv4_src=10.0.0.1,ipv4_dst=10.0.0.10
             packets=45(+12) bytes=4500(+1200)
```

Observations :

— Les statistiques sont collectées toutes les 2 secondes

— Les deltas (variations) sont correctement calculés

— Les anomalies (exfiltration) sont détectées via ces statistiques

Validation : La télémétrie fonctionne correctement et permet la détection d'anomalies.

3.3.2 Export des métriques

Le fichier `controller_metrics.json` est généré toutes les 30 secondes et contient :

— Timestamp de la collecte

— Topologie du réseau

— Compteurs de sécurité (authentifications, révocations, détections)

— Liste des hôtes bloqués

— Temps moyens de détection et révocation

Validation : Les métriques sont correctement exportées et exploitables pour l'audit.

3.4 Vérification de la révocation

3.4.1 Tests de révocation

La révocation a été testée dans chaque scénario d'attaque.

Vérifications effectuées :

1. **Avant révocation :**

— Hôte authentifié : `authenticated=True`

— Score de confiance : 80 (clients) ou 100 (serveur)

— Trafic autorisé : ping et HTTP fonctionnent

2. **Déclenchement de la révocation :**

— Détection d'une menace (scan, flood, exfiltration)

— Log : `ACCESS REVOKED - IP: X.X.X.X - Reason: ...`

3. **Après révocation :**

— Hôte bloqué : `authenticated=False`

— Score de confiance : 0

— Trafic bloqué : ping et HTTP ne fonctionnent plus

— IP ajoutée à `blocked_hosts`

Validation : La révocation fonctionne dans tous les scénarios testés.

3.4.2 Performance de la révocation

Les temps de révocation ont été mesurés sur 10 révocations :

Métrique	**Valeur**
Temps minimum	2.3 ms
Temps maximum	6.8 ms
Temps moyen	4.1 ms
Écart-type	1.2 ms

TABLE 3.2 – Performance de la révocation dynamique

Interprétation : La révocation est extrêmement rapide (< 7 ms) et permet de bloquer une attaque en cours quasi-instantanément.

Validation : La révocation est suffisamment rapide pour être considérée comme temps réel.

3.5 Évaluation de la solution

3.5.1 Points forts

1. **Révocation en temps réel**

— Temps de révocation < 7 ms

— Blocage immédiat de l'attaquant

— Aucune intervention humaine nécessaire

2. **Détection multi-menaces**

— Scan de ports : détection en < 10 ms
— ICMP flood : détection dès le seuil atteint
— Exfiltration : détection via analyse des flows
— Violation de segmentation : blocage immédiat

3. **Segmentation fine**
   — Isolation complète du serveur de remédiation
   — Politiques par segment (protocoles, limites)
   — Contrôle inter-segments
4. **Télémétrie complète**
   — Collecte en temps réel des statistiques
   — Export des métriques pour audit
   — Traçabilité complète des actions
5. **Flexibilité**
   — Seuils de détection configurables
   — Politiques de segmentation modifiables
   — Durée de blocage paramétrable

3.5.2 Limitations

1. **Scalabilité**
   — Le contrôleur traite tous les paquets (PacketIn)
   — Performance limitée sur grand réseau
   — Solution : utiliser des flows plus spécifiques
2. **Faux positifs potentiels**
   — Un scan légitime (monitoring) peut déclencher une révocation
   — Solution : affiner les seuils ou ajouter une whitelist
3. **Détection d'exfiltration**
   — Basée uniquement sur le volume (pas d'analyse de contenu)
   — Une exfiltration lente peut passer inaperçue
   — Solution : ajouter des heuristiques plus sophistiquées
4. **Pas de remédiation automatique**
   — La révocation est permanente (jusqu'au timeout)
   — Pas de mécanisme de déblocage automatique
   — Solution : implémenter un système de remédiation

Objectif	Atteint	Commentaire
Révocation dynamique automatique	ok	< 7 ms
Détection de comportements suspects	ok	4 types de menaces
Segmentation réseau	ok	3 segments + isolation
Surveillance continue	ok	Polling toutes les 2s
Métriques et audit	ok	Export JSON
Temps réel	ok	Détection + révocation < 20 ms

TABLE 3.3 – Atteinte des objectifs du projet

3.5.3 Comparaison avec les objectifs

Conclusion : Tous les objectifs du projet sont atteints.

3.6 Preuve de concept (PoC)

3.6.1 Déploiement du PoC

Le PoC a été déployé avec succès sur un environnement Ubuntu 22.04 avec :

— Mininet 2.3.0

— Ryu 4.34

— Open vSwitch 2.17

— Python 3.10

3.6.2 Reproductibilité

Le PoC est entièrement reproductible :

1. Installation des dépendances (Mininet, Ryu, OVS)
2. Lancement du contrôleur : `ryu-manager zero_trust.py`
3. Lancement de la topologie : `sudo python3 topo.py`
4. Exécution des tests (scénarios 1 à 5)

3.6.3 Envisager un déploiement réel

Prérequis pour un déploiement en production :

1. **Infrastructure**
   — Switches OpenFlow physiques (ou OVS sur serveurs)

- Contrôleur Ryu en haute disponibilité (cluster)
- Système de logs centralisé (ELK, Splunk)

2. **Améliorations nécessaires**
   - Optimisation des flows pour réduire les PacketIn
   - Ajout d'un système de whitelist pour les faux positifs
   - Intégration avec un SIEM pour corrélation d'événements
   - Mécanisme de remédiation automatique (quarantaine, analyse)
3. **Sécurité**
   - Chiffrement du canal OpenFlow (TLS)
   - Authentification du contrôleur
   - Protection du contrôleur lui-même (firewall, IDS)
4. **Monitoring**
   - Dashboard temps réel (Grafana)
   - Alertes automatiques (email, SMS, webhook)
   - Rapports d'audit périodiques

Conclusion : Le PoC démontre la faisabilité technique de la solution. Un déploiement en production nécessiterait des adaptations mais l'architecture de base est solide et fonctionnelle.

4. Conclusion

4.1 Réponse à la problématique

La problématique initiale était : *« Comment révoquer les accès dynamiquement et automatiquement d'un utilisateur déjà authentifié, devenu suspect ? »*

Cette implémentation démontre qu'il est possible de :

1. **Surveiller en continu** le comportement des utilisateurs authentifiés via la télémétrie SDN
2. **Détecter automatiquement** les comportements suspects grâce à des heuristiques de détection
3. **Révoquer immédiatement** les accès en modifiant dynamiquement les règles OpenFlow
4. **Bloquer efficacement** les attaques en cours en moins de 20 millisecondes

La solution implémentée répond donc pleinement à la problématique en offrant une révocation dynamique, automatique et en temps réel des accès d'utilisateurs authentifiés devenus suspects.

4.2 Apports du projet

Ce projet a permis de :

— Démontrer concrètement les principes du Zero Trust (« Never Trust, Always Verify »)

— Illustrer la puissance du SDN pour implémenter des politiques de sécurité dynamiques

— Mesurer les performances d'un système de révocation en temps réel

— Valider la faisabilité technique d'une architecture Zero Trust basée sur SDN

4.3 Perspectives

4.3.1 Améliorations à court terme

— Ajout de détections plus sophistiquées (ML/IA pour anomaly detection)

— Implémentation d'un système de scoring plus fin (dégradation progressive)

— Intégration avec des sources de threat intelligence

— Dashboard de monitoring en temps réel

4.3.2 Évolutions à long terme

— Support multi-contrôleurs pour la haute disponibilité

— Intégration avec des systèmes d'authentification centralisés (LDAP, SAML)

— Remédiation automatique (quarantaine, analyse forensique)

— Extension à des environnements cloud (AWS, Azure, GCP)

4.4 Synthèse

Cette implémentation démontre qu'un système Zero Trust basé sur SDN peut efficacement détecter et révoquer les accès d'utilisateurs authentifiés devenus suspects, en temps réel et de manière automatique. Les performances mesurées (détection < 10 ms, révocation < 7 ms) confirment la viabilité de cette approche pour des environnements de production.

Le projet a atteint tous ses objectifs et fournit une base solide pour un déploiement réel d'une architecture Zero Trust dynamique.