

warning "Mise à jour du backend (octobre 2026)" L'inférence GitHub Models a été retirée le 30 juillet 2026. Les références ci-dessous sont conservées uniquement comme contexte historique pendant la migration de ce lab. Utilisez le Lab 013 ? Backends LLM gratuits pour configurer Foundry Local ou Ollama avant de continuer.

Lab 032 : Sécurité au niveau des lignes pour les agents

Niveau : L300 Parcours : Foundry + Sécurité Durée : ~45 min ? Coût : Gratuit (Docker local) ou Azure PostgreSQL (~15 \$/mois)

Ce que vous apprendrez

Pourquoi les agents ont besoin d'isolation des données (risque multi-locataire) Implémenter la sécurité au niveau des lignes PostgreSQL (RLS) ? la base de données applique l'accès, pas votre application Créer des rôles de base de données pour les identités d'agents Tester qu'un agent ne peut littéralement pas voir les données d'un autre utilisateur ? même avec des attaques par injection Appliquer le RLS au jeu de données exemple orders.csv

Introduction

Lorsque votre agent interroge la base de données, il peut servir le Client A ? mais la requête SQL qu'il génère pourrait accidentellement retourner les commandes du Client B. Les attaques par injection de prompt exploitent exactement cela.

La sécurité au niveau des lignes déplace le contrôle d'accès dans la base de données elle-même. Même si l'agent génère `SELECT * FROM orders`, PostgreSQL ajoute automatiquement `WHERE customer_id = current_user_id` en fonction du rôle de la session. L'agent ne peut pas y échapper.

Prérequis

Docker Desktop (gratuit) OU Azure PostgreSQL du Lab 031 Python 3.11+, pip install `psycopg2-binary` `openai` GITHUB_TOKEN configuré

? Fichiers de support

note "Téléchargez ces fichiers avant de commencer le lab" Enregistrez tous les fichiers dans un dossier lab-032/ dans votre répertoire de travail.

Fichier | Description | Téléchargement | 002_rls.sql | Migration de sécurité au niveau des lignes | ? Télécharger |

Exercice du lab

Étape 1 : Démarrer PostgreSQL avec Docker

[diagram or code sample]

Étape 2 : Charger les commandes exemple et configurer le RLS

[diagram or code sample]

[diagram or code sample]

Étape 3 : Activer la sécurité au niveau des lignes

[diagram or code sample]

[diagram or code sample]

Étape 4 : Tester l'isolation ? les agents ne peuvent pas franchir les limites

[diagram or code sample]

Sortie attendue : [diagram or code sample]

Étape 5 : Construire un agent compatible RLS

[diagram or code sample]sql").strip("[diagram or code sample]

Pourquoi c'est important

Sans RLS | Avec RLS | L'application doit filtrer : `WHERE customer_id = ?` | La base de données filtre automatiquement | L'injection de prompt peut contourner les filtres de l'application | Politique au niveau de la base de données ? impossible à contourner | Bug dans l'agent = fuite de données | Bug dans l'agent = ne voit toujours que ses propres

données | Il faut faire confiance à chaque requête générée par l'agent | La confiance repose sur le rôle de la base de données |

Nettoyage

[diagram or code sample]

Prochaines étapes

Observabilité des agents : ? Lab 033 ? Traçage App Insights RLS sur Azure PostgreSQL : ?

Lab 031 ? pgvector sur Azure

Kit d'accompagnement

Le build du site génère des téléchargements PDF et ZIP stables. Le ZIP contient le guide du lab et tous les fichiers d'accompagnement déclarés.

Télécharger le PDF généré Télécharger le ZIP généré