SLH 2024-2025

Lab 3 - Autorisation

Dans ce laboratoire, nous allons implémenter un mécanisble de contrôle d'accès via Casbin, pour l'application KARAK, un programme de gestion de dossier électronique du patient.

Lisez la description de l'architecture, puis implémentez les changements requis, et répondez aux questions à la fin, dans un fichier `report.md`.

1. Modèle

L'application travaille avec des comptes utilisateur. Il existe trois niveaux de permission: Admin, Médecin et Patient. Chaque utilisateur peut avoir exactement un dossier médical, ou ne pas en avoir.

Si le dossier médical existe, il contient des informations générales (Ici, N° AVS et groupe sanguin). Il contient également une liste de rapports médicaux.

Chaque rapport se réfère à un patient, et possède un auteur, un titre, et un contenu.

2. Architecture

La crate `karak` est organisée de la manière suivante:

2.1. `models`

Contient les structures de données métier de l'application.

La cohérence des données est garantie de manière *structurelle*: il n'existe pas d'invariants a maintenir au delà de ce que le système de types autorise, et c'est pourquoi tous les champs sont publics.

Lorsque de la validation est nécessaire, des types wrappers (p.ex. `UserID` ou `PWHash`) pour distinguer les valeurs validées de celles pas encore validées.

Toutes les structures font une utilisation intensive des macros `derive`, en particulier `Serialize` et `Deserialize` de la crate `serde`. Ces traits sont utilisés à la fois pour le stockage dans la base de données simulée, et pour la communication avec Casbin.

2.2. `db`

Contient la base de données simulées. On sacrifie totalement la performance à la convenance, en stockant simplement les données en mémoire (dans des `HashMaps`), en en sérialisant la totalité du contenu de la base à chaque écriture.

Le module gère deux collections, une pour les utilisateurs et une pour les rapports médicaux. Il implémente également l'équivalent de queqlques requêtes relationnelles spécifiques.

2.3. `services`

Offre une API qui pourrait, par exemple, être offerte via une interface web. Cette couche sert de goulet d'étranglement pour l'application de la politique d'autorisation, elle est donc responsable d'appeler les fonctions d'autorisation pour les opérations concernées.

2.4. `authorization`

Offre une façade type-safe au mécanisme d'autorisation. L'implémentation fournie utilise Casbin mais il serait possible de la remplacer par un autre.

L'autorisation se fait via un contexte, qui stocke le sujet actif (utilisateur connecté), et offre une méthode par action autorisable, recevant le (ou les) objets en argument. Leur valeur de retour est un `Result<bool>` qui distingue les erreurs dans le processus d'autorisation `Err`, des refus d'accès `Ok(false)`.

2.5. `utils::password_utils`

Permet le hachage des mots de passe, et leur vérification (en minimisant le risque de fuite par canal auxiliaire pendant l'authentification)

3. Authentification et Validation des entrées

Implémentez les `todo!()` dans `utils::password_utils` et `utils::input_validation`. Vérifiez que la méthode de vérification des hashes ne permet pas d'énumérer les utilisateurs valides avec un canal auxiliaire.

Dans le reste de l'application, l'énumération des utilisateurs n'est pas un problème, tant que nous sommes capables de logguer l'identité du coupable. Vous n'avez pas besoin d'implémenter de défenses supplémentaires à ce propos.

4. Casbin

Le modèle casbin est défini dans le fichier `access_control/model.conf`. Le modèle choisi est une variante d'ABAC; la politique est une allowlist définissant des conditions logiques pour chaque action. Ces conditions logiques sont écrites dans le langage de règles propre à Casbin, qui se rapproche d'un JavaScript simplifié, et ont accès à tout le contenu fourni par la couche d'autorisation.

La politique casbin peut être stockée via divers mécanismes, typiquement dans une base de données, mais ici nous la stockerons dans un simple fichiern `access_control/policy.csv`, par souci de simplicité.

Chaque ligne dans le fichier `policy.csv` correspond aux colonnes déclarées dans la section `[policy_definition]`, le premier champ `p` identifiant le type de politique (ici il n'y en a qu'une seule).

Merci de **ne pas modifier**, dans le modèle fourni, les définitions des requêtes, pour ne pas casser nos tests. Vous pouvez, si vous le souhaitez, modifier la définition des politiques, le matcher ou l'effet.

Vous pouvez utiliser l'éditeur en ligne sur https://casbin.org/editor pour développer et tester votre politique.

L'application crée dans le répertoire courant un log `karak.log` qui vous aidera à tracer les décisions d'autorisation prises par votre programme.

4.1. Intégration de l'autorisation

Le module `services` execute les opérations sans les soumettre à autorisation. Implémentez le code manquant; Dans chaque méthode, assurez-vous d'ajouter un appel à l'enforcer pour vérifier si l'opération est autorisée ou pas.

Utilisez la méthode `.enforce()` de `Service` pour obtenir l'enforceur, appelez les méthodes appropriées pour vérifier si une opération doit être autorisée ou pas, et interrompez l'opération si l'autorisation est refusée.

4.2. Politique

Implémentez la politique suivante, **uniquement en modifiant le contenu de `policy.csv`**:

- Les utilisateur `Admin` ont tous les droits
- Un utilisateur peut voir, créér, ou détruire son dossier personnel
- Un utilisateur peut mettre à jour les informations personnelles dans son dossier
- Un utilisateur peut choisir ses médecins traitants
- Un utilisateur peut voir les données personnelles des utilisateurs dont il est médecin traitant
- Un utilisateur médecin peut enregistrer des nouveaux rapports concernant n'importe quel utilisateur dont le dossier existe
- L'utilisateur auteur d'un rapport peut voir et modifier ce rapport
- Un médecin peut voir tous les rapports concernant un patient dont il est le médecin traitant

Pour vous aider à démarrer, un exemple de base de données est fourni. Cet exemple contient quatre comptes aux mots de passe très solides: `admin`:`admin1234`, `patient`:`patient1234`, `medecin1`:`medecin1234` et `medecin2`:`medecin1234`.

4.3. Questions

1. Voyez-vous des problèmes avec la politique spécifiée dans l'énoncé ?
2. Parmi les politiques définies ci-dessus, la ou lesquelles serai(en)t pénibles à implémenter s'il fallait utiliser à la place d'ABAC un modèle RBAC traditionnel ?
3. Que pensez-vous de l'utilisation d'un `Option<UserID>` mutable dans la structure `Service` pour garder trace de l'utilisateur loggué ? Comment pourrait-on changer le design pour savoir à la compilation si un utilisateur est censé être connecté ou pas ? Est-ce que cela premet d'éliminer une partie du traitement d'erreurs ?
4. Que pensez-vous de l'utilisation de la macro de dérivation automatique pour `Deserialize` pour les types de `model` ? Et pour les types de `input_validation` ?
5. Que pensez-vous de l'impact de l'utilisation de Casbin sur la performance de l'application ? sur l'efficacité du système de types ?
6. Avez-vous d'autres remarques ?