Analyze and fix an application

LABO 3

Alec Berney
SEC – 13.06.2022

Améliorations apportées

Nous allons ici parcourir chaque amélioration apportée au programme et par thème.

TLS

Le client se connectait en TLS sans ajouter de certificat. Il autorisait les noms d'hôtes (hostname) et certificats invalides pour se connecter.

Pour palier à ceci, j'ai ajouté le certificat au « builder » du « TLSConnector » à l'aide de la fonction « add_root_certificate » et j'ai également refusé les certificats et nom d'hôtes invalides.

Les versions du protocole TLS 1.0 et 1.1 étant dépréciée, il est également nécessaire de changer le « min_protocol_version » à la valeur « Some(Protocol::Tlsv12) » et celle de « max_protocol_version » à la valeur « None » afin de supporter la dernière version du protocole. Ceci est indiqué dans les docs du « TLSConnectorBuilder » :

https://docs.rs/native-tls/latest/native_tls/struct.TlsConnectorBuilder.html#method.min_protocol_version

https://docs.rs/native-tls/latest/native_tls/struct.TlsConnectorBuilder.html#method.max_protocol_version

Cette dernière manipulation a été réalisée dans le fichier « main » du serveur et du client.

Base de données

La base de données stockait le mot de passe des utilisateurs en clair, ce qui est HORRIBLE !

J'ai donc modifié la BD et les structures de données utilisées pour accueillir le hash du mot de passe et un sel à la place.

Login

Pour le login, il a tout d'abord fallu changer le système pour comparer les hash des mots de passes et plus les mots de passes eux-mêmes.

Une autre erreur c'était glissée dans la fonction de login côté serveur, la vérification du mot de passes était uniquement faite si ce dernier existe. Comme nous comparons actuellement des hash, il a été nécessaire de changer le code pour que la génération du hash se réalise de toute façon, et si nécessaire avec une valeur par défaut. Ceci a pour but d'empêcher les « Side channels attacks ».

La dernière erreur que j'ai corrigée, était le fait que des messages différents étaient transmis à l'utilisateur en fonction de la cause de l'échouement du login. Ceci est dangereux, car cela peut communiquer des informations supplémentaires à un attaquant. J'ai donc fait en sorte que les messages d'erreurs pour le login soient les mêmes, que le mot de passe soit faux ou que si l'utilisateur n'existe pas pour empêcher les « Usually Bad Error Messages ».

Entrées et sorties utilisateurs

En ce qui concerne la validation d'entrées utilisateur, presque rien n'avait été réalisé. J'ai donc créé une libraire externe nommé « input_validation », afin de valider les entrées suivantes :

- Les mots de passes,
- Les noms d'utilisateurs,
- Les numéros de téléphones suisses.

La librairie utilise principalement des Regex et la libraire Regex de Rust.

Cette libraire est utilisée par le serveur pour valider / refuser les requêtes en fonction des paramètres reçu. Elle est également utilisée du côté client pour valider les entrées et les redemander en cas d'erreur. Le fichier « input_handlers » contient les fonctions demandant et validant les entrées utilisateurs.

En ce qui concerne les outputs, lors de la demande des utilisateurs « Show users », toutes les informations du compte était passée au client, ce qui est dangereux et non souhaité.

De ce fait, une structure des données dédiées à l'envoie des données a été créées et est utilisées du côté client et serveur. Seul le nom d'utilisateur et le numéro de téléphone sont maintenant transmis du serveur au client lors de cette action.

La structure de données utilisées est nommée « UserAccountPublic ».

Role-Based Access Control

Un contrôle d'accès a été mis en place dans le fichier « access_control.rs ». Il expose la fonction can_perform_action qui vérifie si l'utilisateur donné peut exécuter l'action donnée. Cette fonction à été mise en place en utilisant la libraire casbin pour rust.

Voici la politique de type Role-Based Access Control (RBAC) mise en place :

	Show users	**Change own phone**	**Change phone**	**Add user**	**Login**	**Logout**	**Exit**
Anonymous users	X				X		X
Authenticated users	X	X				X	X
HR users	X	X	X	X		X	X

Cette politique est implémentée dans les fichiers « access_control.conf » et « access_control.csv » afin de fonctionner avec casbin.

Tout ceci permet d'avoir une gestion des droits centralisée dans une fonction et un fichier. Cela évite de répartir la logique à travers tout le programme.

Logs

Aucun log n'était mis en place dans l'application, j'ai donc ajouté des logs grâce à la librairie de Rust log. La méthodologie suivante a essayé d'être suivie le mieux possible :

Action	**Niveau de log**
Erreur pouvant crasher le programme	Error ou panic ! ou expect ! qui crash
Tentative d'accès à une ressource sans les permissions	Warn
Tentative d'action échouée	
Entrée utilisateurs ne respectant pas les formats	
Problème avec les connexions au serveur	
Tout action ou événement se déroulant comme prévu	Info
Information sur le passage dans une fonction pour debug	Trace
Toute autre forme de debug	Debug

Comme vous pouvez le voir, les logs ont été uniquement mis en place du côté serveur comme le veut la logique. Ils ont également été concentré sur les actions que l'utilisateur peut réaliser et quelques informations que le serveur fourni lors du démarrage notamment.

En ce qui concerne la politique des logs affichés et conservés, cette dernière a été mise en place avec la libraire Rust simplelog. Le niveau minimum choisi comme conservé pour la production est « Warn ». Ce qui n'était évidemment pas le cas lors du développement de l'application. Ce niveau peut-être facilement changer dans la fonction « main » au niveau du « TermLogger ».

Autres refactor

Nous allons ici traiter d'autres petit refactor de code réaliser, en voici une liste.

- La structure de données « UserConnected » a été dans déplacées dans un fichier dédié ;
- Un fichier contenant toutes les erreurs possibles du côté serveur a été créés ;
- Un fichier contenant les erreurs de validation a été introduit à la librairie de validation d'entrées utilisateurs « input_validation » ;
- Le support d'un fichier «. env » pour stocker les secrets du serveur, notamment le chemin d'accès aux clés utilisées par TLS et les données des utilisateurs par défaut. La lecture du fichier «. env » est réalisée dans le fichier « env_reader » et retourne une structure de données contenant la configuration du fichier « .env ». Cela a été réalisé pour éviter les secrets dans le code du côté serveur. La librairie utilisée est envfile.

Améliorations possibles restantes

Je vais, ici, lister une liste des améliorations possibles qui auraient pu être réalisée ou seraient intéressantes. Ces dernières n'ont pas été réalisées, car certaines l'ont été faite dans un labo antérieur et / ou prendraient beaucoup de temps supplémentaire.

- Une authentification multi facteur avec par exemple une « Yubikey » ou un « Google Authenticator » ;
- Implémenter un « Challenge-Response » avec un HMAC pour le login ;
- La mise en place d'un système ralentissant le temps de login après chaque erreur consécutive. Ou la mise en place de blocage de compte après x tentative échouée ;
- Redemander l'authentification pour les actions demandant des changements majeurs (changement de son numéro de téléphone, d'un numéro de téléphone ou ajout d'un utilisateur) ;
- Les logs devraient s'enregistrer sur un volume externe et uniquement accessible par le serveur et les administrateurs. Mais pour le bien du laboratoire, ils sont actuellement affichés dans la console du serveur.