CipherShare

Uma plataforma segura de partilha de ficheiros

Projeto Final de Engenharia de Software Seguro - *Grupo 2*

- Joaquim Monteiro (up201905257)
- Fernando Rocha (up202200589)
- João Silva (up201906478)
- Alberto Cunha (up201906325)

Índice

01

Introdução

CipherShare – Plataforma de gestão de ficheiros

Autenticação

Leitura de ficheiros texto

Download e Upload de ficheiros

Partilha de ficheiros via link

02

Arquitetura

Arquitetura

App Server Endpoints

```rust
pub fn get_router() -> Router<AppState> {
    Router::new() Router<State>
        .route(path: "/config", method_router: get(handler: config)) Router<State>
        .route(path: "/files", method_router: get(handler: filestore_get)) Router<State>
        .route(path: "/files/:file", method_router: get(handler: filestore_get)) Router<State>
        .route(path: "/files/:file", method_router: put(handler: filestore_put)) Router<State>
        .route(path: "/links", method_router: get(handler: fileshare_get)) Router<State>
        .route(path: "/link", method_router: put(handler: fileshare_put)) Router<State>
        .route(path: "/link/:code", method_router: get(handler: fileshare_get)) Router<State>
        .route(path: "/link/:code", method_router: delete(handler: fileshare_delete)) Router<State>
        .fallback_service(ServeDir::new(path: "www")) Router<State>
        .layer(CorsLayer::permissive())
}
```

Authentication Server Endpoint

```rust
pub fn get_router() -> Router<AppState> {
    Router::new() Router<Arc<RwLock<State>>>
        .route(path: "/config", method_router: get(handler: config)) Router<Arc<RwLock<State>>>
        .route(path: "/db", method_router: get(handler: db)) Router<Arc<RwLock<State>>>
        .route(path: "/user/login", method_router: post(handler: login)) Router<Arc<RwLock<State>>>
        .route(path: "/user/register", method_router: post(handler: register)) Router<Arc<RwLock<State>>>
        .route(path: "/user/:user/is/:role", method_router: get(handler: user_in_role)) Router<Arc<RwLock<State>>>
        .route(path: "/user/:user/is/:role", method_router: put(handler: add_role_to_user)) Router<Arc<RwLock<State>>>
        .route(path: "/user/:user/is/:role", method_router: delete(handler: remove_role_from_user)) Router<Arc<RwLock<S
        .layer(CorsLayer::permissive())
}
```

03

Funcionalidades de Segurança

Funcionalidades de Segurança

JSON Web Tokens

Used for authenticating requests (username, exp, public_key_user)

TLS

Communications between servers and between the server and the client are encrypted with TLS

Message Authentication

Hashed messages signed with private key from the user to authenticate requests.

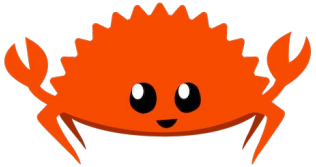
RBAC

A user’s operations are limited by their roles ("viewer", "uploader", "sharer", “admin”).

04

Mitigações

Mitigações de Vulnerabilidades



- A **autenticação baseada em JWT** ajuda a mitigar ataques de interceção de dados e acesso não autorizado. Os tokens JWT são assinados pela chave privada do servidor de autenticação, garantindo que os dados não sejam alterados e que apenas quem possui o token possa acessar aos serviços.
- **Autenticar os requests** evita ataques de manipulação de dados ou de integridade. Isso ajuda a mitigar vulnerabilidades como ataques de replay,
- **RBAC** reduz o risco de acesso não autorizado, limitando o acesso dos usuários apenas às informações e recursos necessários para suas funções. Isso ajuda a mitigar vulnerabilidades relacionadas à escalada de privilégios e acesso indevido a informações sensíveis.
- Temos políticas fortes em relação a definição de passwords através de **Password Validation**, o que ajuda a mitigar vulnerabilidades relacionadas a captura/roubo de palavras-passe (Brute force, etc ...).
- A **Linguagem Rust** oferece memory-safety, fazendo com que os serviços estejam protegidos contra problemas de acessos indevidos a memória.
- **SQL-Injection** impossível pois os nosso dados são guardados usando outra tecnologia.

Live Demo

Thanks!

Do you have any questions?