Engenharia de Software Seguro

CipherShare

Relatório do Projeto Final

Grupo 2

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

Introdução

A CipherShare é uma plataforma inovadora e robusta desenvolvida para atender às necessidades contemporâneas de segurança na partilha de ficheiros. Concebida como uma solução abrangente, esta aplicação destaca-se pela sua ênfase na proteção e na facilidade de uso. Combinando recursos de autenticação de usuários, manipulação segura de arquivos e opções versáteis de compartilhamento, a CipherShare visa proporcionar uma experiência ágil e confiável para os usuários que buscam uma plataforma segura e eficiente para a troca de informações sensíveis.

Baseada em funcionalidades chave, como a autenticação robusta de usuários para garantir a legitimidade dos acessos, a capacidade de ler e manipular arquivos de texto com segurança, além das operações intuitivas de upload, download e compartilhamento via link, a CipherShare se destaca como uma ferramenta essencial para a segurança informática. A priorização da proteção dos dados e a facilidade de acesso são os pilares fundamentais que norteiam o desenvolvimento e a filosofia por trás desta aplicação.

Este relatório detalha a arquitetura, controlo de acesso, funcionalidades de segurança e os benefícios proporcionados pela CipherShare, evidenciando seu papel crucial no ambiente de compartilhamento de arquivos, ao oferecer uma solução confiável para os requisitos de segurança da informação.

Arquitectura

A aplicação CipherShare foi construída com uma base composta por HTML, CSS e JavaScript, enquanto a infraestrutura de backend é sustentada por quatro servidores desenvolvidos em Rust. Estes servidores são: o app-server, responsável pela comunicação entre os serviços e a aplicação cliente; o auth-server, encarregado das operações de autenticação; o server-fileshare, que gerencia as operações de compartilhamento de arquivos via links; e o server-filestore, dedicado às operações essenciais de manipulação de arquivos, incluindo listagem, leitura, download e upload.

Arquitetura CipherShare:

Aplicação Cliente (HTML, CSS, JavaScript):

Interface acessível e intuitiva para os utilizadores.
Comunicação com o app-server para operações de gestão de ficheiros e com o auth-server para autenticação de utilizadores.

App-Server:

Funciona como intermediário entre a aplicação cliente e os serviços de gestão de ficheiros.
Gerencia as requisições e respostas, direcionando-as aos servidores apropriados.

```
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>
```

Figura 1 - Rotas do app-server.

Auth-Server:

Responsável pelas operações de autenticação de usuários.
Validação de tokens JWT para garantir a legitimidade dos acessos.
Controla o acesso de utilizadores a recursos segundo o ***role-based access control***.

```
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<St
    .route(path: "/user/:user/is/:role", method_router: delete(handler: remove_role_from_user)) Router<Arc<R
```

Figura 2 - Rotas do servidor de autenticação.

Server-FileShare:

Encarregado das operações de compartilhamento de ficheiros via links.
Comunica com o server-filestore para manipulação de ficheiros.

```
Router::new() Router<Arc<RwLock<State>>>
    .route(path: "/config", method_router: get(handler: config)) Router<Arc<R
    .route(path: "/db", method_router: get(handler: db)) Router<Arc<RwLock<St
    .route(path: "/links", method_router: get(handler: user_links)) Router<Ar
    .route(path: "/link", method_router: put(handler: add_link)) Router<Arc<R
    .route(path: "/link/:code", method_router: get(handler: file_of_link)) Ro
    .route(path: "/link/:code", method_router: delete(handler: delete_link))
```

Figure 3 - Rotas do servidor de partilha de ficheiros.

Server-FileStore:

Gerencia o armazenamento e manipulação de ficheiros.
Operações de listagem, leitura, download e upload de ficheiros.

```
Router::new() Router<Arc<RwLock<State>>>
    .route(path: "/config", method_router: get(handler: config)) Router<Arc<RwLo
    .route(path: "/files", method_router: get(handler: list)) Router<Arc<RwLock<S
    .route(path: "/files/:file", method_router: get(handler: read)) Router<Arc<Rw
    .route(path: "/files/:file", method_router: put(handler: write)) Router<Arc<R
    .route(path: "/file-exists/:file", method_router: get(handler: exists)) Route
    .route(path: "/file-shared/:file", method_router: get(handler: read_shared))
```

Figure 4 - Rotas do serivdor de gestão de ficheiros.

Persistência de dados dos utilizadores:

A base de dados dos utilizadores da plataforma consiste num ficheiro .json onde se encontra a lista de utilizadores detalhada com username, roles e hashed password do utilizador.

Servidor Comum:

Existe ainda uma componente em comum na arquitetura da aplicação onde se inicializa a configuração do TLS, o *logging* global para efeitos de debug, inicializa-se a configuração da plataforma através de um ficheiro de configuração, defini-se a estrutura de utilizador e outras estruturas importantes para a plataforma tal como os campos do token de autenticação.

A intercomunicação entre os servidores ocorre de maneira coordenada, permitindo a execução eficiente das funcionalidades oferecidas pela CipherShare. O app-server atua como o ponto central, direcionando solicitações para os servidores apropriados, garantindo uma experiência fluida e segura para os usuários finais.

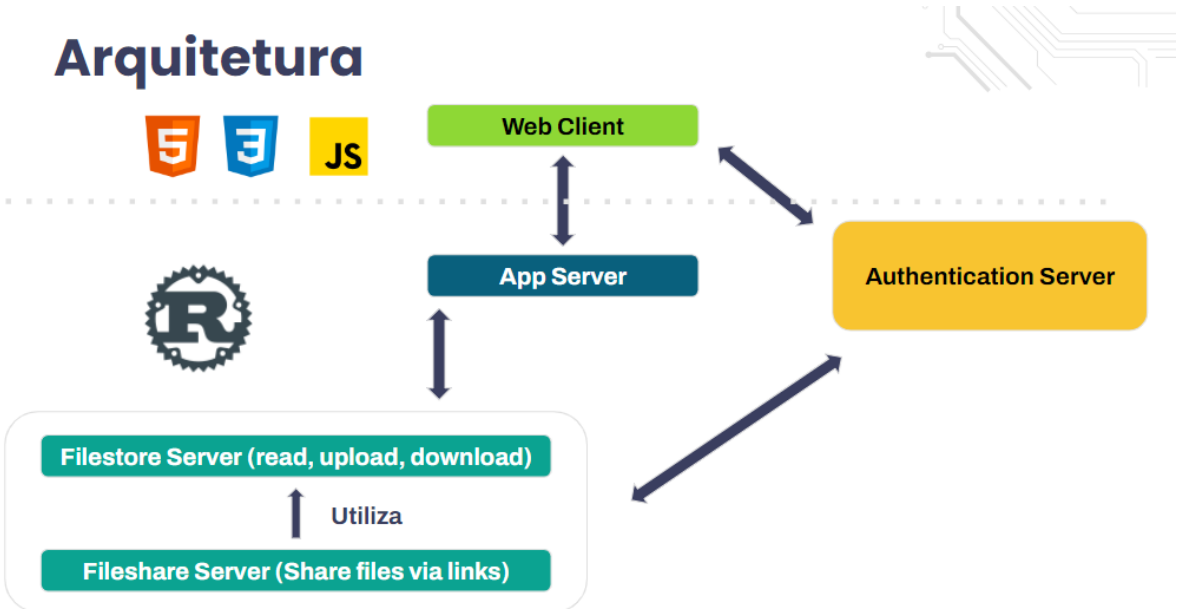

Figura 5 - Arquitetura CipherShare.

Controlo de Acesso

Controle de Acesso Baseado em Funções na CipherShare

O controle de acesso é uma parte crucial da segurança da informação na CipherShare. A implementação do Role-Based Access Control (RBAC) oferece uma estrutura flexível e granular para gerenciar as permissões dos utilizadores, permitindo diferentes níveis de acesso com base nas funções atribuídas a eles.

Roles de Utilizadores:

Admin: Os usuários com a função de administrador possuem privilégios de acesso elevados, permitindo a gestão global da plataforma. Eles têm permissão para realizar operações como adicionar ou remover usuários, modificar permissões e acessar todas as funcionalidades da CipherShare.

Viewer: Os usuários com função de visualização têm permissão para acessar e visualizar os ficheiros presentes na plataforma.

Uploader: Usuários com função de uploader podem fazer upload de novos ficheiros para a plataforma, permitindo a contribuição de conteúdo à CipherShare.

Sharer: Esta função é designada para usuários capazes de compartilhar arquivos via links. Eles têm a capacidade de criar links únicos ao utilizador para ficheiros específicos e compartilhá-los com outros utilizadores, facilitando a distribuição controlada de informações.

Implementação do RBAC:

A rota **/user/:user/is/:role** do auth-server permite a verificação se um utilizador possui determinada função na plataforma, fornecendo uma resposta verdadeira ou falsa.
Operações como adicionar e remover funções de usuários (add_role_to_user e remove_role_from_user) estão restritas a administradores ou ao próprio usuário (apenas na remoção), garantindo uma gestão controlada das permissões.

Assim, um utilizador que pretende utilizar um serviço da plataforma tem que obter validação por parte do auth-server para garantir que tem acesso a tal recurso.

A implementação do controle de acesso baseado em funções na CipherShare é um componente vital na garantia da segurança e na gestão eficiente dos recursos da plataforma, assegurando que cada utilizador tenha acesso apenas ao necessário para suas responsabilidades e contribuições.

Autenticação

Autenticação CipherShare

O processo de autenticação desempenha um papel central na segurança da CipherShare, garantindo a validação e autorização adequada dos utilizadores. Através do auth-server, são implementadas diversas rotas e funcionalidades dedicadas à autenticação.

Geração e Verificação de Tokens JWT:

A rota /user/login é responsável por receber as credenciais dos utilizadores, autenticá-los e, em caso de sucesso, gerar tokens JWT (JSON Web Tokens) para autorização subsequente. A função create_jwt_response é crucial neste processo, onde é gerada a resposta contendo o token JWT e a chave privada associada ao usuário autenticado.

Autenticação de utilizadores com Token JWT:

A autenticação na plataforma pode ser feita através do token JWT guardado no armazenamento local do browser se este tiver disponível e válido. Se este não se encontrar válido, o utilizador é enviado para a janela de login para se autenticar de novo e receber novo token.

Registro Seguro de Novos Usuários:

O endpoint /user/register lida com o processo de registro de novos utilizadores na plataforma. Durante esse processo, são aplicadas verificações de segurança, como a avaliação da força da senha, com a biblioteca **Zxcvbn**, para garantir senhas robustas e prevenir vulnerabilidades. A biblioteca argon2 de rust foi utilizada para fazer hash das passwords recolhidas no pedido de registo.

A implementação eficaz da autenticação na CipherShare, por meio do auth_server, é um componente fundamental na garantia da segurança, controle de acesso e integridade dos dados na plataforma.

Funcionalidades de Segurança

A CipherShare prioriza a segurança em várias camadas, implementando uma série de funcionalidades e protocolos robustos para garantir a integridade, autenticidade e confidencialidade das informações compartilhadas.

TLS (Transport Layer Security)

Todas as comunicações entre os servidores e entre o servidor e o cliente são criptografadas com TLS. Esse protocolo garante a segurança dos dados transmitidos, protegendo contra interceptações e garantindo a privacidade e integridade das informações. Usando tls_rustls do axum_server, cria-se a camada de tls passando a configuração dada pelo certificado e a chave, guardados como ficheiros pem no servidor. A função "new_reqwest_client_from_certificates" cria os clientes que farão pedidos para o servidor, sendo necessário adicionar o "root_ca.cert" no browser para os pedidos funcionarem.

JSON Web Tokens (JWT)

A autenticação das requisições é realizada utilizando JSON Web Tokens. Esses tokens contêm informações como o username, expiração e chave pública do utilizador, permitindo a autenticação e autorização adequadas para operações na plataforma.

Message Authentication

Mensagens são autenticadas utilizando assinaturas digitais com a chave privada do utilizador. Isso permite a autenticação das requisições, garantindo que as mensagens não tenham sido alteradas e que sejam provenientes de uma fonte confiável. A mensagem é criada a partir do path do request e da timestamp de quando o pedido é feito, usando a private key que o cliente tem guardada e o algoritmo de hashing SHA-256. O timestamp e a assinatura são passadas como headers. Do lado do servidor, cria-se a mensagem outra vez, extraído o path, timestamp e a assinatura, e verifica-se se a assinatura corresponde a mensagem, usando a public key guardada no servidor. Usa-se JSEncrypt do lado do cliente e openssl do lado do servidor para tratar da criptografia.

Controle de Acesso Baseado em Funções (RBAC)

As operações dos usuários na CipherShare são limitadas pelas suas funções, como "viewer", "uploader", "sharer" e "admin". Esse controle granular de acesso permite a atribuição de permissões específicas com base nas funções dos utilizadores, garantindo um ambiente seguro e controlado.

A combinação dessas funcionalidades de segurança na CipherShare estabelece uma infraestrutura sólida, abordando aspectos cruciais como criptografia de comunicação, autenticação de requisições, integridade das mensagens e controle preciso de acesso, assegurando a proteção e confidencialidade dos dados compartilhados na plataforma.

Criptografia

A CipherShare utiliza criptografia RSA (Rivest-Shamir-Adleman) para geração e gestão de chaves, juntamente com Tokens Web JSON (JWTs) para comunicação segura e autenticação.

Criptografia RSA:

Geração de Chave: O projeto gera um par de chaves RSA com um tamanho de 2048 bits (KEY_SIZE). A chave privada é gerada ou carregada a partir do PRIVATE_KEY_PATH, enquanto a chave pública correspondente é derivada da chave privada.
Assinatura e Verificação: A chave privada RSA do servidor de autenticação é utilizada para assinar JWTs para garantir autenticidade e integridade. A chave pública é utilizada para verificar a assinatura desses tokens no lado dos serviços.
Também é gerado um par de chaves RSA para o utilizador após autenticação; a chave privada é enviada numa resposta juntamente com o JWT de volta para o utilizador e é utilizada para assinar uma mensagem enviada juntamente com pedidos para os serviços, para autenticação de pedidos do lado do serviço usando a chave pública do utilizador.

JSON Web Tokens (JWTs):

Geração de Tokens: Os JWTs são gerados para autenticação de utilizador e contêm reivindicações específicas (como nome de utilizador, expiração e possivelmente a chave pública do utilizador). Estes tokens são assinados utilizando a chave privada RSA do servidor de autenticação, garantindo a sua autenticidade e prevenindo manipulação.

Armazenamento Seguro:

A CipherShare cuida do armazenamento seguro de materiais criptográficos, como a chave privada do servidor de autenticação, em ficheiros especificados por PRIVATE_KEY_PATH e PUBLIC_KEY_PATH, utilizando operações de I/O de ficheiros para estes fins.

O JWT e a chave privada do utilizador recebidos na resposta após autenticação do utilizador na plataforma são guardados no armazenamento local do navegador.

Persistência:

A chave privada RSA é persistida e carregada a partir de um ficheiro (PRIVATE_KEY_PATH) para uso subsequente, garantindo consistência e continuidade em operações criptográficas mesmo após reinícios do servidor. O JWT e a chave privada do utilizador são persistidos no armazenamento local do navegador.

Em resumo, é utilizado criptografia RSA para geração, assinatura e verificação de chaves. A utilização de JWTs assinados com chaves RSA é uma abordagem robusta para autenticação de utilizadores e comunicação segura dentro da aplicação. O tratamento de materiais criptográficos e cenários de erro também contribui para uma implementação mais segura.

Mitigações

O uso do **TLS** dificulta a interceptação de dados por terceiros não autorizados. Isso ajuda a prevenir ataques do tipo "Man-in-the-Middle", nos quais um atacante intercepta e potencialmente altera as informações transmitidas entre o cliente e os servidores.

A **autenticação baseada em JWT** ajuda a mitigar ataques de **spoofing** de identidade e previne obtenção de acessos não autorizados. 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.

A **autenticação das mensagens** enviadas junto dos pedidos aos serviços por meio de assinatura digital com chave privada do utilizador protege contra a modificação não autorizada de dados. Isso ajuda a evitar ataques que procuram corromper dados durante a transmissão, como os ataques de "man-in-the-middle" que tentam alterar informações.

O controlo granular de acesso aos servicos reduz o risco de ataques de acesso não autorizado, ao limitar a operação dos usuários.

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.