Secure Document Repository System

Introdução

Este relatório apresenta o desenvolvimento e a implementação de um Sistema de Repositório Seguro de Documentos. O principal objetivo deste projeto é desenvolver uma aplicação segura que permita o armazenamento, criação, gestão e controlo de acesso a documentos e permissões dentro de uma organização.

O sistema foi projetado para garantir a confidencialidade e integridade dos dados por meio de funcionalidades como criptografia de documentos, controlo de acesso baseado em funções e gestão segura de sessões.

Este relatório detalha a estrutura do projeto, as decisões de implementação e as medidas de segurança adotadas.

Arquitetura geral do sistema (modelo cliente-servidor)

O sistema do repositório implementa uma arquitetura cliente-servidor, onde o lado cliente é composto por um conjunto de comandos que realizam operações locais, como encriptação/desencriptação de documentos e gestão de credenciais. O servidor (*repository.py*), desenvolvido com Flask, gere as operações centrais do Repositório, incluindo a gestão de organizações, o armazenamento de documentos e o controlo de acessos.

Essa separação permite que operações sensíveis, como a encriptação, sejam realizadas no lado cliente antes da transmissão dos dados, enquanto o servidor se concentra no armazenamento seguro e na gestão de acessos. A comunicação entre cliente e servidor ocorre através de endpoints RESTful API, com todas as interações realizadas com as permissões adequadas, graças ao nosso sistema de gestão de sessões.

Análise das funcionalidades desenvolvidas

Os dados relativos a organizações, sessões, etc., são armazenados num ficheiro JSON com o nome db.json. Este ficheiro é criado ao rodar o ficheiro db.py pela primeira vez, e contém uma estrutura inicial predefinida que inclui categorias como organizações, utilizadores e sessões. Caso o ficheiro já exista, os dados são carregados para a memória; caso contrário, é gerada uma base de dados vazia.

Análise da encriptação

Para este projeto decidimos que a encriptação dos ficheiros e documentos deveria ter uma abordagem baseada em criptografia de chave simétrica AES-GCM, suportada por um sistema de derivação de chaves (KDF) seguro e por uma infraestrutura de chaves assimétricas para estabelecimento de segredos partilhados (ECDH). Para a encriptação das chamadas à API, quando já está estabelecida uma session key, é usada AES-GCM, até esse momento é usada uma encriptação híbrida com AES-CFB e RSA.

1. Estabelecimento da session e Chave de session

ECDH (Elliptic Curve Diffie-Hellman)

A chave de sessão, utilizada posteriormente para encriptar os documentos, é derivada através do protocolo **ECDH** (Elliptic Curve Diffie-Hellman). Nesse processo, o utilizador possui uma chave privada derivada da sua palavra-passe, utilizando um **KDF** (Key Derivation Function), enquanto a Organização disponibiliza uma chave pública. Quando o *user* pretende iniciar sessão, é calculado um segredo partilhado (shared secret) com base na troca de chaves ECDH entre a chave privada do utilizador, derivada da palavra-passe, e a chave pública da Organização.

HKDF (HMAC-based Key Derivation Function)

Com o shared secret (resultado do ECDH) em mãos, aplica-se uma função **HKDF** (HMAC-based Key Derivation Function) para gerar uma session key e um *salt* adicional. A HKDF permite extrair e expandir o segredo inicial, assegurando a aleatoriedade e robustez da criptografia. A session key gerada é armazenada temporariamente no servidor, associada a um session_id.

Armazenamento Seguro

A *session key* resultante é guardada em Base64 no lado do servidor, mas apenas durante a duração da session .

O utilizador também a recebe codificada em Base64, mas não a partilha explicitamente com terceiros.

2. Derivação de Chave de Documento (doc_key)

Para a derivação da chave do documento (doc_key), é utilizado um processo que assegura segurança e unicidade para cada documento encriptado. Sempre que um novo documento é criado, gera-se um nonce aleatório de 16 bytes, denominado doc_nonce, que funciona como um *salt* exclusivo para o documento. Esse *nonce* evita a reutilização de chaves, eliminando riscos associados a ataques que exploram padrões previsíveis na encriptação.

A doc_key é então derivada a partir da session key (já existente) e do doc_nonce, utilizando funções como HMAC ou HKDF. Esse método garante que cada documento possui uma chave simétrica única e independente, reforçando a confidencialidade.

Importante ressaltar que a doc_key não é armazenada no servidor em plaintext. Sempre que necessário, ela é recriada dinamicamente com base na session key e no doc_nonce, evitando a exposição direta da chave e dificultando o acesso não autorizado em caso de comprometimento do sistema.

3. Encriptação com AES-GCM

Geração de IV

Para cada processo de encriptação, gera-se um IV (vetor de inicialização) aleatório de 12 bytes. O **GCM** (*Galois/Counter Mode*) requer um IV único para cada encriptação, minimizando os riscos de reutilização de IV.

Encriptação

Utilizamos a *doc_key* em conjunto com o IV para inicializar o algoritmo **AES-GCM**. Em paralelo, o modo **GCM** gera um *authentication tag (tag)* que garante a integridade e autenticidade dos dados.

Armazenamento do Resultado

Guardamos no servidor apenas:

- O ciphertext (em Base64),
- O IV (Base64),
- O tag (Base64),
- O doc_nonce (Base64),
- Metadados adicionais (por exemplo, file_handle, datas, ACL, etc.).

O *plaintext* (conteúdo original) não é guardado, assegurando a confidencialidade.

A doc_key não é guardada: sempre que precisamos de desencriptar, voltamos a derivá-la através da chave de session e do doc_nonce, ajudando na proteção contra ataques de *eavesdropping.*

4. Desencriptação e Verificação de Integridade

Recuperação da doc_key

Para desencriptar, o sistema volta a derivar a doc_key recorrendo à mesma chave de session e ao doc_nonce. Isto garante que só quem possui a *session key* correta consegue reconstruir a doc_key.

AES-GCM

Com a doc_key, o IV e a tag, aplica-se o Cipher(algorithms.AES(doc_key), modes.GCM(iv, tag)) para desencriptar. Caso a tag não coincida, ou se o ciphertext tiver sido modificado, a desencriptação falha automaticamente, detetando manipulações.

Verificação HMAC (Requests)

Todos os pedidos HTTP são validados com HMAC no payload, usando a *session key.* Desta forma garante-se a autenticidade das chamadas e protege-se contra ataques de *replay* (através de um counter sequencial), ataques manipulação no request, ataques de *eavesdropping* e ataques de *impersonation.*

5. Sumário da Resiliência Contra Ataques

Eavesdropping

- Utilização de criptografia simétrica **AES-GCM** para todos os dados confidenciais, assegurando a confidencialidade do conteúdo.
- Comunicação protegida por **TLS**, impedindo a captura e leitura de dados transmitidos.
- As chaves de session são derivadas através de **ECDH** e nunca transmitidas diretamente, garantindo que apenas as partes autorizadas têm acesso às chaves de encriptação.
- Mesmo que um atacante intercete as comunicações, os dados permanecem ilegíveis sem acesso às chaves de session ou doc_keys.

Impersonation

- Autenticação robusta baseada em chaves derivadas de passwords e salt (ECDH).
- Assinatura de todas as requisições com HMAC-SHA256, usando a session key, garantindo a autenticidade das mensagens.
- Gestão de permissões e roles, limitando o acesso de utilizadores a recursos específicos.
- Verificação de identidade através da comparação de chaves públicas associadas aos utilizadores.
- Um atacante não pode se passar por outro utilizador ou pelo servidor sem acesso às chaves privadas ou às session keys legítimas.

Manipulation

- O uso de **AES-GCM** gera uma tag de autenticação (authentication tag) durante a encriptação, garantindo a integridade dos dados.
- Qualquer tentativa de modificar os dados cifrados resulta na invalidação da tag, impedindo a desencriptação.
- A aplicação de HMAC-SHA256 aos pedidos HTTP assegura a integridade do payload das mensagens, protegendo-as contra alterações não autorizadas.
- Alterações não autorizadas nos dados ou nas comunicações são identificadas de forma imediata, evitando o uso de informações comprometidas.

Replay

- Cada *request* inclui um counter sequencial gerido pelo cliente e validado pelo servidor.
- O servidor rejeita qualquer mensagem com um valor de counter duplicado ou fora de ordem.

- Um atacante não pode reutilizar mensagens legítimas capturadas anteriormente (Replay Attack), pois estas serão rejeitadas como inválidas.

6. Chamadas à API

Para a encriptação das chamadas à API foram usados 2 métodos, um para as chamadas após ter uma sessão, e um para o antes.
Antes de existir uma sessão com uma session_key associada, é usado um método de encriptação híbrida, usando AES-CFB e RSA , este segundo sendo usado para a encriptação da chave proveniente do AES. Para o servidor é enviado essa chave, o iv e o resto do payload encriptado. Foi usado um método híbrido de modo a não só aumentar a quantidade de bytes encriptados que se conseguem enviar sem sacrificar segurança na transmissão.
Já com uma sessão criada é usada a encriptação AES-GCM. Para o servidor é agora enviado o iv, o tag e o resto do payload encriptado.

Access Control System

A função user_has_permission apresenta diversos pontos positivos no que diz respeito à segurança e à organização do controlo de acessos em sistemas baseados em papéis. A implementação de uma verificação de permissões é fundamental para assegurar que os utilizadores tenham apenas os acessos necessários às suas funções.

Outro aspeto positivo é o feedback informativo gerado pela função, que fornece mensagens claras e detalhadas quando uma permissão é negada. A estrutura modular da função é também uma mais-valia, permitindo a sua reutilização em várias partes do sistema, o que reduz a duplicação de código e potencia a manutenção eficiente. Por fim, a simplicidade da implementação torna a função intuitiva e fácil de compreender, o que é vantajoso para equipas de desenvolvimento e para adaptações futuras.

Comandos da Anonymous API

- Create organization

O comando **create_org** faz parte da API anónima, permitindo a criação de uma organização sem a necessidade de uma session ativa. Em termos de segurança, a implementação abrange várias boas práticas que visam garantir a integridade dos dados e a proteção contra abusos. A validação de dados de entrada é um ponto positivo, pois a função create_organization verifica se todos os campos obrigatórios estão presentes antes de prosseguir com a criação da organização, evitando dados incompletos ou maliciosos. Além disso, a API também verifica se o nome da organização já existe na base de dados, prevenindo a criação de organizações duplicadas, o que seria um risco para a integridade do sistema.

Outro ponto importante é a validação e o carregamento da chave pública enviada com os dados da organização. Se houver algum problema ao carregar a chave, um erro é retornado, evitando o armazenamento de chaves inválidas ou manipuladas. Além disso, a chave pública é convertida para o formato PEM antes de ser guardada, o que é uma prática

comum para garantir a segurança das chaves criptográficas. A configuração de uma lista de controle de acesso (ACL) para a organização também é uma boa prática, pois inicialmente restringe o acesso e a modificação de roles, documentos e outras informações sensíveis aos administradores, o que assegura que somente utilizadores com permissões adequadas possam realizar tais ações.

A gestão de utilizadores e roles dentro da organização também é bem implementada, já que o administrador da organização é automaticamente atribuído ao role de Manager, com o controlo completo sobre as permissões e a gestão dos membros da organização.

- **List organizations**

O comando **rep_list_orgs** é responsável por listar todas as organizações definidas em um repositório. Quando executado, ele faz uma requisição GET para o endpoint /organization/list, que, por sua vez, retorna todas as organizações armazenadas na base de dados. Os dados retornados são então exibidos em formato JSON de maneira legível, com indentação e chaves ordenadas para facilitar a leitura.

- **Create session**

O comando **rep_create_session** cria uma session para um utilizador dentro de uma organização. Ele começa por carregar as credenciais do utilizador, verificando a senha com base na chave pública e salt. Em seguida, envia uma requisição POST ao servidor para criar a session , que retorna um session_id, a chave de session codificada em base64 e a data de expiração.

No servidor, o endpoint /session/create valida a organização e o utilizador, gera uma chave de session a partir de ECDH e HKDF, cria um session_id único e armazena a session no banco de dados. O servidor responde com os dados da session . Esse processo permite a autenticação e autoriza o utilizador a interagir com a organização usando a chave de session gerada.

O servidor também define um tempo de expiração para a sessão criada. Durante a iteração do utilizador com a organização, é verificado se a sessão está expirada através da função is_session_expired. Esta função verifica a validade da sessão com base no tempo de expiração. Caso a sessão esteja expirada, ela é automaticamente eliminada da base de dados, avisando o utilizador que a sessão expirou. Caso não esteja expirada, a sessão permanece ativa, permitindo que o utilizador continue a interagir com a organização e o tempo de expiração volta ao inicial, neste caso definido como 1 hora.

- **Download file**

O comando **rep_get_file** permite fazer o download de um arquivo a partir do seu identificador (file handle). Através de uma requisição GET ao servidor, o comando procura o arquivo correspondente e, se encontrado, exibe no terminal o nome do documento, o identificador do arquivo e o conteúdo criptografado. Caso seja fornecido um caminho de saída, o conteúdo pode ser salvo num ficheiro local. Se não for especificado um caminho, o conteúdo criptografado é impresso no terminal. Em caso de erro, como resposta inválida ou falha na ligação ao servidor, o comando exibe a mensagem de erro correspondente.

No lado do servidor, o endpoint /file/<file_handle> recebe o identificador do arquivo, procura-o nos documentos armazenados e, se encontrado, devolve os metadados do documento, incluindo o nome, o identificador e o conteúdo criptografado.

Comandos da Authenticated API

- **Assume session role**

O comando **rep_assume_role** permite que um utilizador assuma um role dentro de uma organização durante uma session ativa. O comando começa lendo o arquivo de session para obter o session_id, em seguida, envia uma requisição POST ao servidor com o session_id e o role que o utilizador deseja assumir. Se a requisição for bem-sucedida, o comando informa que o role foi assumido com sucesso, caso contrário, exibe um erro com o código de status e a mensagem retornada. No lado do servidor, o endpoint verifica se o session_id e o role foram fornecidos corretamente. Se o session_id for válido e o role existir na organização, e estiver ativo, o role é adicionado à lista de "active_roles" da session e registado na base de dados. Este comando permite que o utilizador altere dinamicamente os papéis dentro de uma session , o que pode ser útil para controlar o acesso e as permissões dentro da organização.

- **Release session role**

O comando **rep_drop_role** permite que um utilizador liberte um role previamente assumido. O comando começa por ler o arquivo de session para obter o session_id, e em seguida envia uma requisição POST ao servidor, fornecendo o session_id e o role que o utilizador deseja libertar. Se a requisição for bem-sucedida, o comando confirma que o role foi libertado com sucesso, caso contrário, exibe um erro com o código de status e a mensagem de erro. No servidor, o endpoint verifica se o session_id e a role são válidos. Caso o role tenha sido assumido na session , esta é removida da lista de "active_roles". Se o role for o role atualmente assumida na session , este é também removida como role assumido. O role é removido da organização, e a base de dados é atualizada com essas mudanças. Este comando é útil para permitir que um utilizador liberte papéis quando estes não são mais necessários, assegurando a flexibilidade no controlo de acesso e permissões dentro da organização.

- **List session roles**

O comando **rep_list_roles** lista os papéis ativos numa session . Este lê o session_id do arquivo de session , envia uma requisição GET ao servidor e, se a session for válida, exibe os papéis ativos. Caso não haja papéis ativos, informa o utilizador. No servidor, o endpoint verifica o session_id e retorna os papéis associados à session . Este comando facilita a visualização dos papéis existentes numa session .

- **List subjects (of my organization)**

O comando **list_subjects** lista os users de uma organização, ou os dados de um user em específico caso seja dado o username do mesmo como atributo. O endpoint verifica se o session_id é válido e está associado a uma organização. Caso seja válido, o servidor retorna todos os sujeitos associados à organização, caso nenhum username seja especificado ou apenas o sujeito filtrado, caso o username corresponda a um sujeito da organização.

- **List roles (from my organization)**

rep_list_roles, este comando é responsável por listar todas as roles de uma session.
Quando o session_id é válido, busca a lista de papéis ativos (active_roles) da session . Se não houver papéis ativos registrados, retorna uma lista vazia.

- **List the subjects in one of my organization's roles**

list_subject_roles trata de listar as roles associadas a um user dentro de uma organização. Recebe como atributos o username e o session_id. Após verificar ambos, retorna uma lista com as roles associadas a esse user.

- **List the roles of one of my organization's subjects**

list_role_subjects é basicamente o inverso do endpoint anterior, dado que recebe o session_id e uma role como atributos e após que ambos existem e que a session está ativa, lista os users que têm associada a role dada como atributo.

- **List the permissions in one of my organization's roles**

O endpoint **list_role_permissions** lista as permissões associadas a uma role dentro de uma organização associada a uma session ativa. Após verificar se o session_id é válido vê as roles definidos para essa organização em db["organizations"] e retorna-as numa lista.

- **List the roles that have a given permission**

O endpoint **list_permission_roles** lista os roles associadas a uma permission específica dentro de uma organização associada à session ativa. a permission é recebida na URL, enquanto o session_id é dado como atributo. Primeiramente, verifica se o session_id está presente e é válido no banco de dados db["sessions"]. Os papéis que possuem a permissão são adicionados à lista matching_roles que é depois retornada.

- **List documents (with filtering options, such as dates or creators).**

O endpoint **list_documents** permite listar documentos filtrados com base em critérios específicos, como o nome de user e a data. O filtro pode ser aplicado para exibir apenas documentos criados por um user específico ou criados antes, depois ou na data fornecida. O código valida se o documento possui um arquivo associado, verificando o valor de file_handle. Se o valor for falso ou ausente, o documento é ignorado. Em seguida, se o parâmetro date_filter_value for fornecido, ele aplica o filtro de data de acordo com o tipo de filtro (nt para antes, ot para depois, ou et para igual à data fornecida). O create_date de cada documento é comparado com a data fornecida, e os documentos que não atendem ao critério de data são descartados. Caso algum filtro seja aplicado, a lista final de documentos será ajustada de acordo.

Comandos da Authorized API

- **Add subject**

O endpoint **add_subject** permite adicionar um novo user a uma organização ou associar um sujeito existente a uma organização, caso ele ainda não esteja associado. Começa por validar o session_id fornecido, verifica se o user tem a permissão SUBJECT_NEW para adicionar um sujeito, utilizando a função user_has_permission. Após isso, o sistema verifica se todos os campos obrigatórios (session_id, username, name, email, public_key_pem) foram fornecidos. Em seguida, o sistema verifica se o sujeito com o username fornecido já existe na base de dados. Se o sujeito já existir, ele é adicionado à organização caso não tenha sido adicionado previamente, retornando uma mensagem de sucesso. Se o sujeito for novo, um novo registro é criado com os dados fornecidos, incluindo o nome, e-mail, chave pública e status.

- **Change subject status (suspend/reactivate)**

Os endpoints **reactivate_role** e **suspend_subject** são usados para reativar uma role dentro de uma organização e suspender um user, respectivamente. O primeiro endpoint, **reactivate_role**, inicia verificando o session_id e o role fornecido. Em seguida, verifica se o user possui a permissão ROLE_UP para reativar o papel.O endpoint também checa se o papel já está ativo. Se todas as condições forem atendidas, o status do papel é alterado para "active" e a base de dados é atualizada.

Já o endpoint **suspend_subject** começa por verificar o session_id e o username fornecidos. O código também valida se o user possui a permissão SUBJECT_DOWN para suspender o sujeito. Em seguida, o sistema verifica se o sujeito existe na base de dados e se já está suspenso. Se não, o status do sujeito é alterado para "suspended", e a base de dados é atualizada.

- **Add role**

O comando **rep_add_role** adiciona uma role a uma organização associada à session atual. Este analisa o session_id do arquivo de session e envia uma requisição POST ao servidor para adicionar a nova role. Para que o comando seja executado, o utilizador precisa ter permissão ROLE_NEW. Se a permissão for verificada e válida, a role é adicionada à organização. Caso contrário, um erro é retornado. No servidor, o endpoint verifica se o utilizador tem a permissão necessária e, em seguida, adiciona a role à organização, se ainda não existir.

- **Change role status (suspend/activate)**

Os comandos **rep_suspend_role** e **rep_reactivate_role** permitem suspender ou reativar uma role numa organização associada à session atual. O comando rep_suspend_role suspende uma role, enquanto o comando rep_reactivate_role reativa uma role. Ambos requerem permissões específicas: ROLE_DOWN para suspender e ROLE_UP para reativar. Quando o comando é executado, a session é verificada e, se o utilizador tiver a permissão adequada, o status da role na organização é alterado para "suspended" ou "active". Caso contrário, um erro é retornado.

- **Add/remove subject to role**

Os endpoints **add_subject_to_role** e **remove_subject_from_role** são usados para adicionar ou remover um user de uma role dentro de uma organização.

O **add_subject_to_role** começa verificando se todos os dados necessários (como session_id, role e username) foram fornecidos. Em seguida, valida o session_id fornecido. Após essa verificação, o endpoint checa se o user possui a permissão ROLE_MOD para modificar papéis. O código então verifica se o papel fornecido existe dentro da organização. Além disso, o endpoint verifica se o user já pertence à organização e se o sujeito já possui o papel. Caso contrário, o sujeito é adicionado ao papel, a base de dados é salva, e uma mensagem de sucesso é retornada com o status 200.

O **remove_subject_from_role** segue uma lógica similar. Ele começa verificando se os campos necessários estão presentes e se o session_id é válido. O endpoint também verifica se o user tem a permissão ROLE_MOD. Após isso, verifica se o papel existe dentro da organização e se o sujeito está associado a esse papel. Caso o sujeito esteja associado ao papel, ele é removido da lista de "subjects" do papel.

- **Add/remove permission to role**

Os comandos **rep_add_permission** e **rep_remove_permission** são utilizados para alterar as propriedades de um papel numa organização associada à session atual, permitindo adicionar ou remover permissões e subjects desse role. Quando um argumento (username ou permission) é passado, o comando verifica se é uma permissão conhecida ou um nome de utilizador. Dependendo do caso, ele adiciona ou remove o subject ou permissão do role correspondente. Ambos os comandos requerem a permissão ROLE_MOD. Caso o utilizador tenha a permissão adequada, a alteração é realizada e a base de dados é atualizada. Se ocorrer algum erro, é retornada uma mensagem com o erro específico.

- **Upload a document**

O endpoint **/documents/<document_name>** permite o upload de documentos criptografados. Ele verifica se o user tem permissão DOC_NEW para adicionar documentos, e se todos os campos obrigatórios estão presentes. O documento é armazenado com metadados públicos e restritos, e o id do documento é gerado usando HMAC com a chave da session. O nome e proprietário do documento são guardados na DB da organização.

- **Download a document metadata**

O endpoint **/documents/<document_name>/file** permite o download de um documento criptografado. Ele verifica se o user tem permissão DOC_READ, e valida o session_id fornecido. Caso o documento exista e a organização tenha os metadados necessários, a resposta inclui os parâmetros necessários para descriptografá-lo, como o vetor de inicialização (iv), conteúdo criptografado, tag de autenticação e o nonce do documento. Se algum critério falhar, a resposta retorna um erro correspondente com status adequado.

- **Delete a document**

O endpoint **/documents/<document_name>** permite a eliminação de um documento, basicamente removendo o identificador do arquivo associado ao documento. Ele valida a permissão do user para deletar o documento (DOC_DELETE) e verifica se o session_id é válido. Se o documento e os dados da organização forem encontrados, o campo file_handle é limpo, marcando o documento como excluído. Caso algum critério falhe, um erro apropriado é retornado.

- **Change document ACL**

O endpoint **/documents/<document_name>/acl** permite alterar as permissões (ACL) de um documento específico. Ele verifica se o user tem permissão DOC_ACL, se os campos obrigatórios estão presentes e se o session_id é válido. A permissão fornecida deve começar com DOC_, e o parâmetro sign deve ser "+" para adicionar uma permissão ou "-" para removê-la. Se o papel (role) e a permissão especificada existirem, a ACL do documento é atualizada. Caso contrário, são retornados erros específicos. A resposta final confirma a atualização com uma mensagem de sucesso.

Análise utilizando os critérios OWASP (V4)

V4.1 - Controlo de Acesso Baseado em Funções (RBAC)

O Controlo de Acesso Baseado em Funções (RBAC) é uma abordagem amplamente utilizada em sistemas de segurança, onde os utilizadores são atribuídos a roles, e estas roles determinam as permissões associadas. No projeto, esta abordagem é implementada através de comandos como:

rep_add_role: Adicionar roles.
rep_assume_role: Assumir roles.

rep_drop_role: Remover roles.

Exemplo de Implementação:
O comando **rep_drop_role** é utilizado para remover uma role atribuída a um utilizador numa session específica. Este comando chama uma função em client.py, que executa a lógica necessária para garantir que o utilizador tenha as permissões adequadas antes de remover uma role. A verificação das permissões do utilizador na session é feita antes de qualquer alteração ser realizada, garantindo que apenas utilizadores autorizados possam executar este comando.

V4.2 - Controlo de Acesso Baseado em Atributos (ABAC)

O Controlo de Acesso Baseado em Atributos (ABAC) define as permissões com base nos atributos do utilizador, recurso ou ambiente. Embora o RBAC seja a abordagem principal no projeto, a integração do ABAC poderia ser uma melhoria significativa. A introdução de atributos adicionais, como hora do acesso, localização ou contexto da operação, poderia aumentar a flexibilidade e precisão nas decisões de acesso, fornecendo uma camada extra de segurança e protegendo contra acessos não autorizados em cenários dinâmicos.

V4.3 - Separação de Privilégios

A separação de privilégios é um princípio essencial para a segurança, onde a atribuição de permissões é limitada a um conjunto específico de ações, evitando que entidades ou utilizadores tenham permissões excessivas.

No contexto do projeto, a separação de privilégios pode ser aplicada das seguintes maneiras:

Diferenciação de Permissões entre Perfis: O sistema utiliza roles e permissões para controlar o acesso dos utilizadores. Contudo, não está claro se operações críticas, como a remoção de roles (**rep_drop_role**), são restritas a administradores. É necessário garantir que apenas utilizadores com permissões elevadas possam realizar ações que possam afetar significativamente o sistema.

Definição de Permissões Claras e Restritas para Perfis: Apesar de o sistema permitir a adição e remoção de permissões (**rep_add_permission**, **rep_remove_permission**), não está totalmente claro se cada perfil tem um conjunto bem definido de permissões. Para melhorar, seria importante garantir que cada perfil tenha um escopo restrito de ações, minimizando o risco de abusos.

Para garantir a conformidade com o capítulo V4.3 da ASVS, as seguintes melhorias são sugeridas:

Verificar e implementar restrições explícitas para garantir que somente administradores possam realizar operações críticas.
Garantir que cada perfil tenha um conjunto de permissões claramente definido e restrito, evitando abusos de privilégios.
Essas melhorias ajudarão a proteger o sistema contra acessos não autorizados e a minimizar o risco de abusos.

V4.4 - Revisão e Auditoria de Permissões

A revisão e auditoria regular das permissões é fundamental para identificar possíveis atribuições incorretas ou desnecessárias de perfis. A implementação de registos de auditoria (logs) para monitorizar ações críticas, como a adição ou remoção de roles, reforça a segurança e ajuda a identificar potenciais abusos ou erros na gestão de permissões.

Melhorias Sugeridas
Para garantir que a revisão e auditoria de permissões sejam corretamente aplicadas, as seguintes melhorias podem ser implementadas:

Implementação de Registos de Auditoria (Logs): Criar logs detalhados para monitorizar as ações críticas, como a adição ou remoção de roles.

Revisão Regular dos Logs de Auditoria: Estabelecer um processo contínuo para revisar os logs, que pode ser feito manualmente ou através de scripts automatizados que analisam os logs em busca de atividades suspeitas ou anómalas.

Conclusão

O desenvolvimento do Sistema de Repositório Seguro de Documentos demonstrou ser uma solução robusta e eficaz para o armazenamento e gestão segura de informações sensíveis. Este projeto alcançou os objetivos propostos ao implementar um conjunto de práticas e mecanismos de segurança avançados, incluindo criptografia forte com AES-GCM, gestão segura de chaves com ECDH e HKDF, e um sistema de controlo de acesso baseado em roles e permissões rigorosamente verificadas.
A arquitetura cliente-servidor adotada mostrou-se eficiente, permitindo que operações sensíveis, como a encriptação de documentos, ocorram no lado do cliente, enquanto o servidor se foca na gestão de acessos e armazenamento seguro.
Adicionalmente, o projeto abordou de forma consistente ameaças comuns, como eavesdropping, impersonation, manipulação de dados e replay attacks, mitigando riscos com o uso de mecanismos como HMAC-SHA256, counters sequenciais, e validações rigorosas em cada etapa do processo.

z

João Varela 113780
Henrique Teixeira 114588
Carolina Prata 114246