Licenciatura em Tecnologias Digitais e Segurança de Informação

Turma do 2º Ano, 2º Semestre 2024/25

Trabalho realizado no âmbito das Unidades Curriculares:

- Sistemas Distribuídos e Segurança, com Professor/Investigador João Pedro Pavia
- Programação para a Internet, com Professor/Doutor Thiago Bessa Pontes
- Criptografia Aplicada, com Professor/Autor/Doutor Carlos Serrão

Trabalho realizado por:

- Eliaquim Alexandre, Nº 122051
- Gonçalo de Monção Meireles Liberato Ferreira, Nº 120037

GRUPO 8

Tema do trabalho:

Sistema de Armazenamento Seguro de Informação na Cloud

Índice

Introdução

O objetivo do presente documento, escrito no formato de um relatório, é fazer uma listagem dos requisitos que serão necessários para o desenvolvimento do projeto prático das cadeiras Criptografia Aplicada, Programação para a Internet e Sistemas Distribuídos e Segurança.

Para que sejam implementadas as funcionalidades que o grupo pretende, uma pesquisa foi realizada, sendo que serão descritos os requisitos funcionais e não-funcionais necessários para que isso aconteça de acordo com os temas abordados em cada cadeira e da forma como se relacionam com elas.

Este processo será também frutífero no sentido em que fazer o levantamento destes requisitos leva a um planeamento mais minucioso do projeto.

Sistemas Distribuídos e Segurança

Requisitos Funcionais

Este tipo de sistemas deve ser capaz de lidar com, potencialmente, milhares de pedidos, alguns feitos em simultâneo. Surgiram muitas soluções com a finalidade de preparar a arquitetura do mesmo para situações deste tipo, claro, adaptadas às necessidades de cada um.

Por exemplo, no caso dos serviços de streaming, fazer o envio dos vídeos após a sua compressão, ou enviá-lo por partes, para quando o utilizador terminar o primeiro segmento lhe ser enviado o restante.

Uma vez que o objetivo do grupo é desenvolver um sistema de armazenamento de informação em cloud, de forma segura, este terá de tentar prever que problemas o sistema pode vir a enfrentar, e adaptar a sua solução de acordo.

Antes de apresentar as soluções o grupo começará por listar as dificuldades que prevê encontrar, humildemente admitindo que é impossível prever todas, e apenas depois os requisitos para resolver as mesmas.

- Sobrecarga dos servidores web;
- Má distribuição da carga e/ou partilha de recursos;
- Diferentes capacidades/velocidades de processamento por parte dos servidores empregados para comunicações servidor/servidor, servidor/cliente;

- Falhas na rede;
- Incompatibilidade das camadas middleware;
- Ataques cibernéticos com intuíto de indisponibilizar ou roubar informações;
- Largura de banda insuficiente;

Cada solução para estes problemas é um requisito funcional do projeto.

Procura-se desenvolver um sistema de modelo Cliente/Servidor replicado para distribuir a carga e evitar pontos de falha únicos, bem como um servidor proxy como serviço. O proxy servirá apenas para transformação e reencaminhamento de pedidos, uma vez que não se encontrou um motivo válido para implementação de memória cache no mesmo.

Implementar-se-ão web sockets, uma vez a troca de certificados digitais entre o utilizador e o backend principal requer alguma capacidade de armazenamento e, posteriormente, as comunicações entre servidores serão feitas através do protocolo IPSec (comunicação full-duplex sobre TCP seguro).

A forma de interação pela qual dispõem é a mensagem, sempre unicast/ponto-a-ponto, porém também sempre bidirecional. Alguns servidores não estão autorizados a comunicar com outros, nomeadamente, apenas o backend principal pode comunicar com o backend Key-Management (KMS).

A comunicação será do tipo assíncrona.

Requisitos Não-Funcionais

Os requisitos não funcionais (RNFs) devem abordar aspetos como desempenho, segurança, escalabilidade e operacionalidade do sistema.

Com a finalidade de diminuir a latência e tempo de resposta médio para pedidos críticos (ex: autenticação, acesso a ficheiros), a utilização de CPU/memória em pods Kubernetes não deve exceder 70% sob carga máxima, para permitir escalabilidade automática. Será implementado um limite $\leq$ 200 ms para esse tipo de pedidos, e quanto à comunicação assíncrona entre servidores, esta deverá garantir a entrega de mensagens em $\leq$ 500 ms. Poderão ser feitos até 10.000 pedidos simultâneos, por instância do servidor backend principal (com escalabilidade horizontal via Kubernetes).

A comunicação será do tipo assíncrona uma vez que o proxy será capaz de gerir a distribuição de tarefas, ou, se um pedido de envio não for processado antes do time-out, os próprios containers estão configurados para se clonarem e processarem o pedido o quanto antes.

Quanto a questões de escalabilidade, será empregue a funcionalidade de auto-scaling de pods (Kubernetes HPA) baseado em métricas de CPU, memória e latência. O servidor proxy NGINX atuará como load balancer, distribuindo tráfego entre instâncias do backend principal. Para facilitar esta replicação, todos os serviços terão um design stateless, ou seja, os servidores tratam cada pedido de forma independente, em vez de o fazer no formato statefull, onde o estado de uma aplicação é conservado.

A segurança é mantida através de diversos processos, incluindo a utilização do protocolo IPSec com AES-256 para comunicação servidor-servidor. Os restantes requisitos não-funcionais que envolvem criptografia encontram-se listados mais abaixo na secção dedicada à cadeira de Criptografia Aplicada.

Quanto a resistência a ataques, optámos por limitar a taxa de pedidos no NGINX para mitigar ataques de DDoS, equipando o mesmo ainda com uma firewall que deverá detetar a presença de payloads maliciosos antes de interagirem com o servidor frontend.

Existe uma firewall entre os servidores de backend principais e a frontend/backend PKI/log-management server, o que não deverá aumentar significativamente a latência de tráfego uma vez que o tipo de pacotes que vão trocar é previsível e relativamente fácil de configurar.

Por fim, estará ainda disposto um honeypot no sistema, comunicando por um porto específico com a frontend e backend principais, para simular um log-management server.

Para promover resistência a falhas, configurar-se-á a rede de Kubernetes com Circuit Breakers, que desativam o pod com falhas até que a funcionalidade Kubernetes self-healing permita que se retome o serviço. Até que isto aconteça um método retry com backoff exponencial.

Para resolver problemas de compatibilidade, utilizar-se-ão RESTful API´s para suporte a clientes web (Chrome, Firefox, Safari) e mobile (Android/iOS). Cada container será equipado com uma imagem Docker que dê uso a diversas bibliotecas de dependências, de forma a maximizar a compatibilidade do sistema com dispositivos e algoritmos de cifra mais antigos, desde que sejam capazes de se conformar com as políticas de segurança.

Existe ainda o problema da largura de banda. QoS será implementada no Kubernetes de forma que tráfego crítico seja sempre priorizado, já que esta pode revelar-se insuficiente para realização de todas as comunicações em simultâneo.

Criptografia Aplicada

Requisitos Funcionais

Este serviço de cloud mantém todos os dados sensíveis sempre seguros por meio de encriptação, desde o momento em que o utilizador se autêntica até que decide terminar a sessão. Mantém todos os ficheiros segmentados e encriptados em diversas bases de dados, tem um servidor backend para lidar com autorizações e outro para guardar metadados e lidar com certificados/entidades.

Os metadados a que me referi são, no caso do servidor de manutenção de logs, informações como "evento:" (login/logout, accessed, removed, etc), "status:" (fail, success, etc), "timestamp" (2025-01-01, etc), "IP" (x.x.x.x); e no caso do servidor backend PKI/KMS, que lida com validação de certificados, os metadados são "Hash do ficheiro original:" e "Número de série dos blocos" ({"bloco1": "UUID-123", "bloco2": "UUID-456"});

Poderá ser consultado um relatório acerca dos logs feitos numa determinada conta caso o utilizador esteja autenticado, bem como consultar todos os ficheiros que tem armazenados na database.

Requisitos Não-Funcionais

Idealmente estes processos devem ser rápidos, para além de, claro, eficientes.

Para que isto seja possível é necessário que existam implementações de bibliotecas OpenSSL nos servidores backend principal e PKI/KMS, uma vez que os restantes servidores estão encarregues somente de guardar informações já encriptadas e metadados em plain-text.

O primeiro processo em que criptografia é essencial é o de autenticação. O user autentica-se perante uma Certificate Autorithy e obtém um certificado digital que utilizará para se autenticar perante o sistema:

[User]à[Frontend(login/UI)]ßà[Backend Principal (recebe chave privada/certificado digital do user encriptada com a sua pública)]ßà[Backend PKI/KMS (valida certificado/identidade)]

Rotação automática de certificados de 70 em 70 dias.

O segundo processo criptográfico que ocorre é o de armazenamento das informações do login (e posteriormente logout) do utilizador no Log-Management server.

Depois do processo já descrito o servidor de backend principal comunica ao frontend server que o utilizador pode utilizar a página de UI e esta é então renderizada. Quanto a informações como "IP", "timestamp" e "status", estas são todas encriptadas com recurso ao algoritmo de encriptação AES e enviadas para o servidor de Log-Management, onde são guardadas.

Como foi referido, o utilizador pode consultar o seu registo de logs, sendo que nesse caso, depois de o utilizador fazer o pedido para tal, o log management server filtra os logs da conta em questão, envia para o backend para serem desencriptados e enviados para o servidor de frontend, onde podem ser visualizados em plain-text.

[BE Principal (envio de autenticação bem sucedida para FE/encriptação de logs mas não metadados)]à [Log-Management Server(guardar logs encriptados e metadados em plain-text)]

O processo mais complexo que terá de ser realizado pelo sistema é o upload e download de ficheiros.

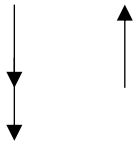
PC#1- Upload:

- O envio do ficheiro para o servidor de backend principal por parte do user está já encriptado pelo protocolo IPsec v2;
- Ao chegar ao backend principal é desencriptado e segmentado, sendo-lhes atribuído a cada segmento um número de série. A estes chamaremos "blocos". Os conjuntos de blocos que formam um ficheiro têm um nome de sére;
- Apenas o bloco é encriptado, com AES, e concatenado ao nome e número de série em plain-text que lhe corresponde;
- É feito um envio de cada segmento para uma base de dados de acordo com o algoritmo "Round-Robin", ou seja, de forma cíclica (SERVER-EXPLODER);
- As bases de dados não possuem a chave de cifra, logo não têm acesso à informação;
- A chave de cifra é enviada para o Key-Management Server (KMS), bem como o nome de série associado ao conjunto de blocos que essa chave cifra. A comunicação entre estes está protegida pelo protocolo Ipsec v2;

PC#2- Download:

- Quando o utilizador quer reaver um ficherio em específico, escreve ou escolhe de uma lista disponibilizada pelo frontend o nome do ficheiro (este nome é o mesmo que o nome de série);
- O servidor de backend principal envia um pedido às bases de dados (SERVER-EXPLODER) que contém blocos concatenados com o nome de série especificado;
- Recebe cada bloco e organiza-os por número de série;
- Pede a chave de cifra ao servidor KMS associada ao nome;
- Utiliza a mesma para fazer a desencriptação e envio para a página de frontend, onde pode ser visualizado pelo user;
- A chave de cifra é descartada, uma vez que não deverá ser novamente utilizada;

[User]à[Frontend]à[BE Principal(segmenta/encripta/desencripta]à[Bases de Dados(c/ nomes e nºs de série]



[KMS (chave de cifra e metadados em plain-text)]

Toda a comunicação entre servidores está encriptada com o protocolo IPSec, e o único servidor que consegue estabelecer conexão com o KMS é o backend principal. Apenas um porto estará aberto para o fazer.

Programação para a Internet

Requisitos Funcionais

As páginas web disponibilizadas pelo servidor frontend são simples e o utilizador consegue organizar-se facilmente, não sendo necessário habituação antes que este consiga utilizar aptamente as funcionalidades da página.

O primeiro contacto que o utilizador tem é entre ele e o servidor NGINX, mesmo que não perceba, uma vez que este serve apenas de proxy para os restantes. O objetivo é proteger a conta do utilizador e os restantes servidores do sistema, bem como dividir tarefas pelos mesmos de forma a acelerar processos e oferecer uma experiência mais satisfatória ao user.

Inicialmente existirá apenas um frontend server. O servidor de frontend renderiza a página de login primeiro e apenas depois de feita a autenticação a UI. A user interface terá um formato

muito simples, organizado por pastas. Nestas estão contidos os ficheiros que o utilizador decide guardar, e o nome que é dado a cada ficheiro pelo mesmo deve ser utilizado no desenvolvimento do código para facilitar o desenvolvimento dos mecanismos que suportam todas as operações associadas com o ficheiro.

Vai existir inicialmente apenas um servidor backend principal, o que pode ser alterado pelo servidor NGINX se este entender que o mesmo está sobrecarregado. O mesmo acontece com o frontend.

Vão existir diversas bases de dados, desenvolvidas em diversas plataformas (PostgreSQL, MongoDB, AirTable, etc.). O honeypot que serve de deteção de intrusos será programada numa destas plataformas.

Requisitos Não-Funcionais

Uma vez que o objetivo é um sistema que funcione rápida e eficazmente, pensou-se na utilização de tecnologias associadas a containers, mais especificamente uma rede de Kubernetes.

Para equipar a frontend com uma interface amigável, serão utilizados recursos disponibilizados pela framework "Bootstrap 5", que foi apresentada ao grupo pelo professor Thiago no decorrer de uma aula.

A rede de Kubernetes, gerida pelo servidor NGINX, uma vez que é o único componente que interage com o utilizador, é útil para fornecer escalabilidade ao sistema.

Faz o papel de proxy, fazendo distribuição lógica de tarefas pelos restantes e processando inbound traffic para testar a presença de malware. Tem a capacidade de clonar e eliminar containers, onde correm as aplicações e dependências das mesmas, necessárias para um bom funcionamento do sistema. Está configurado para reduzir ao máximo o número de recursos que a rede utiliza bem como manter a velocidade e suavidade na experiência do utilizador.

Resumindo, será utilizado o Docker para criação de containers capazes de desempenhar as tarefas necessárias para o funcionamento do sistema, Docker-Compose para que os containers comuniquem entre si, e um servidor NGINX capaz de utilizar as imagens destes containers para os clonar e dividir tarefas. Aliado às capacidades de self-healing e auto-scaling, que podemos configurar no servidor proxy, fica assim configurada uma rede K8.

Por fim, para melhorar um pouco mais a experiência do utilizador, decidiu-se realizar uma implementação do "dark mode", através da sua especificação e styling em CSS, seguida de um script de java que automatize a sua utilização. Estes serão concatenados no documento HTML que disser respeito à página de UI/login (frontend).