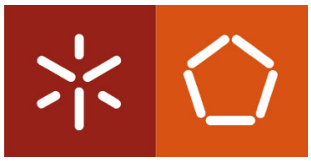
Universidade do Minho

Escola de Engenharia

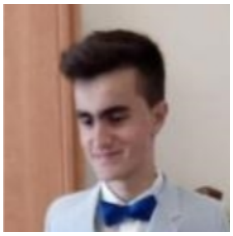


Tecnologias de Segurança

Mestrado em Engenharia Informática

DJUMBAI
Serviço local de troca de mensagens

Aluno:	*Número:*
Eduardo Francisco Longras Figueiredo	pg52679
Gonçalo Nuno Pereira Senra	pg52683
Henrique Miguel da Silva Costa	pg52684



Conteúdo

Introdução

O presente relatório visa especificar todas as decisões tomadas e estudadas para o conceção do serviço local de troca de mensagens: **DJUMBAI**, bem como detalhar as características arquiteturais, as estratégias de segurança e outros minuciosos aspetos tidos em conta.

O DJUMBAI, do ponto de vista funcional, é um serviço que suporta o envio de mensagens para um utilizador *unix* local e a sua leitura pelo respetivo destinatário, de forma segura, permitindo ainda gerir utilizadores do serviço. Também, tem a noção de grupos privados de conversação, oferecendo mecanismos para criação de grupos, remoção de grupos e de gestão dos seus membros.

Relativamente à definição da arquitetura do sistema, o grupo foi desafiado a criar uma proposta de arquitetura que potenciasse a segurança da troca e armazenamento de mensagens. Tendo este objetivo em mente, a adoção de um arquitetura de micro-serviços, (tal como especificado no enunciado), foi um aspeto fulcral para a componentização do sistema. Este tipo de abordagem, permitiu-nos ajustar as permissões de cada micro-serviço, para o estritamente necessário, ou seja, cada processo *djumbai* apenas tem acesso a recursos que realmente precisa.

Numa solução monolítica, caso fosse encontrada uma brecha, e a mesma fosse explorada com sucesso, todo o sistema estaria comprometido, e todos os dados críticos aos quais o serviço tem acesso (mensagens dos clientes incluídas). Por outro lado, com uma arquitetura distribuída em vários serviços, apesar de geralmente ser prejudicial ao desempenho do sistema, beneficia a redução da criticidade de possíveis ataques ao mesmo. Na eventualidade de um micro-serviço ser atacado, não implicaria que todos os outros fossem comprometidos, ou até que a confidencialidade e integridade dos dados do sistema fossem postos em causa.

A arquitetura do sistema, e os aspetos funcionais deste serviço foram altamente inspirados no agente de troca de emails local e remoto **qmail**, desenvolvido por Daniel J. Bernstein, que influenciou este projeto relativamente às questões do desenvolvimento de software seguro.

Arquitetura Funcional

Nesta secção, iremos apresentar as características funcionais do sistema desenvolvido, detalhes arquiteturais do serviço, informações sobre cada componente, e todas as tomadas de decisão ao longo do projeto, que nos permitiram garantir o bom funcionamento, confidencialidade e integridade do serviço. Na figura 2.1, apresentamos a arquitetura do sistema desenvolvido.

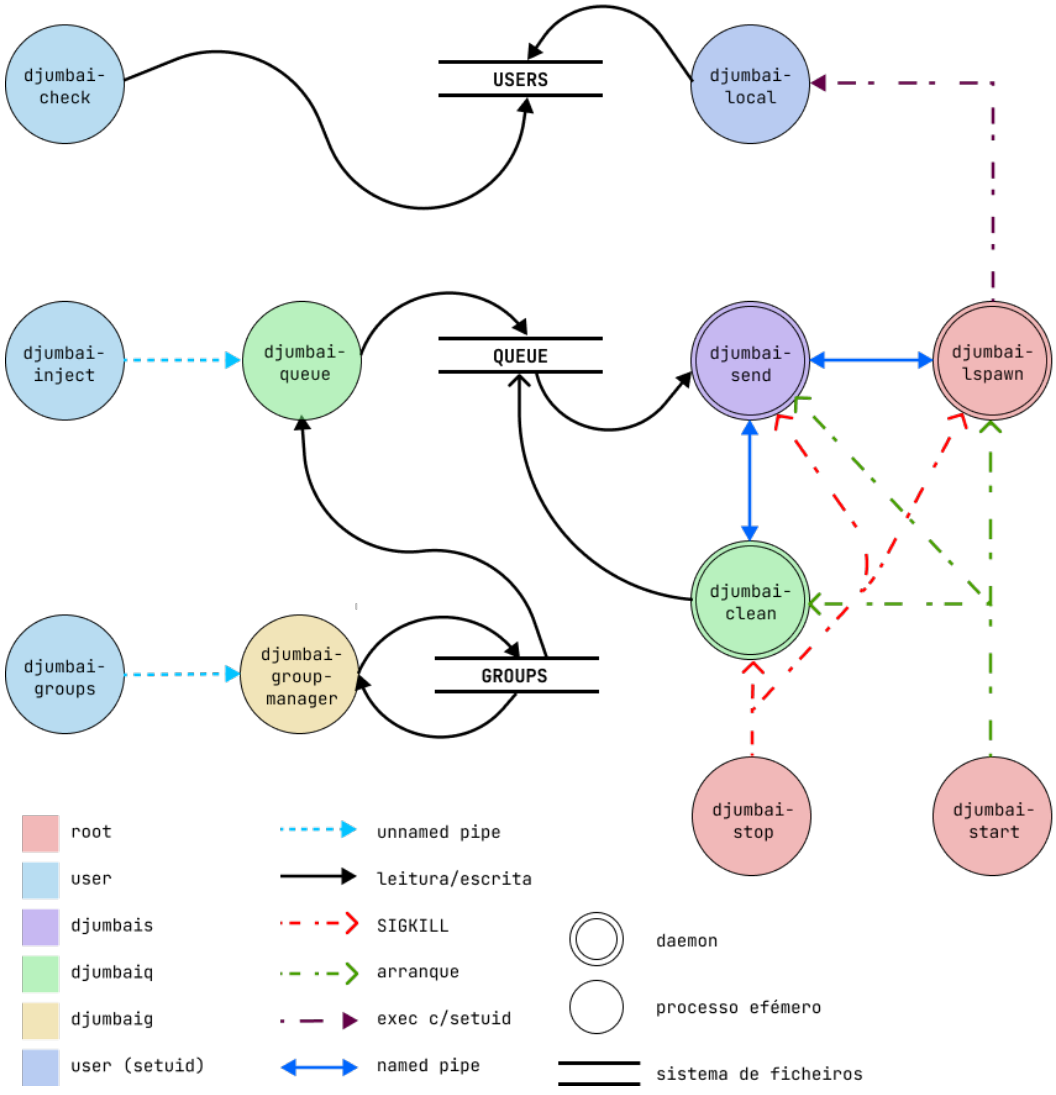


Figura 2.1: Diagrama arquitetural

2.1 DJUMBAI-INJECT

Este componente, é responsável por servir uma interface ao utilizador, de forma a permitir que o mesmo possa enviar uma mensagem para um outro utilizador, ou grupo. Este programa pode ser utilizado das seguintes formas:

- djumbai-inject
- djumbai-inject -g

O **djumbai-inject** permite a um utilizador construir uma mensagem(máx. 512 caracteres), associada a um uid de destinatário válido, e a um assunto (máx. 200 caracteres), enviando-a para o **djumbai-queue**, que é executado com o objetivo de colocar a mensagem, e o seu envelope na "queue". Caso o comando seja efetuado com a *flag -g*, a mensagem é construída com o intuito de ser distribuída por todos os elementos do grupo especificado pelo mesmo.

Diretamente dependente do **djumbai-queue**, este programa cria *unnamed pipes* (leitura e escrita), para comunicar com o anterior. Uma vez que, o ciclo de vida deste tipo de IPC (inter process comunication) é o mesmo dos processos, achamos que foi uma decisão acertada, optar por este tipo de comunicação. A mensagem enviada do **djumbai-inject** para o **djumbai-queue**, é "embrulhada" numa *struct*, e posteriormente serializada, de modo a facilitar a comunicação entre eles.

2.2 DJUMBAI-QUEUE

O **djumbai-queue** quando recebe os dados serializados, efetua a sua desserialização e posteriormente, os processa guardando-os num sistema de pastas que no seu aglomerado podemos denominar de "queue".

A forma como guardamos a mensagem e o respetivo envelope na "queue", foi inspirada no qmail, que utiliza uma metodologia que permite que as mensagens sejam armazenadas para futuro envio, com um **id** único, mesmo quando a inserção de mensagens é feita de forma concorrente.

Primeiramente, após a receção e processamento da mensagem o **djumbai-queue**, cria um ficheiro na pasta **pid**, cujo nome será o próprio *pid* (será único enquanto o processo estiver ativo), e escreve o conteúdo da mensagem. De seguida, com a ajuda da função "lstat", obtemos o *inode* desse ficheiro, com isto renomeámos o ficheiro para esse *inode*, que será único enquanto a mensagem não for apagada. Após a alteração do nome do ficheiro, o mesmo é movido para a pasta **mess**, que será a pasta definitiva

para essa mensagem. Por fim, é criado um ficheiro com o envelope da mensagem na pasta **intd**, e um link para esse envelope na pasta **todo** (ambos com o mesmo nome do ficheiro da mensagem), para permitir que o **djumbai-send**, possa proceder à leitura e envio destas informações.

2.3 DJUMBAI-SEND

O **djumbai-send** é o programa que permite o tratamento de mensagens presentes na "queue", para posterior envio, através do programa **djumbai-lspawn**, efetuando este processamento periodicamente. De notar que este programa estará sempre ativo.

Sempre que um link para um envelope é encontrado por este processo, o mesmo cria um ficheiro para o envelope na diretoria **info**, e outro ficheiro com os respetivos destinatários na pasta **local** (diretoria que guarda ficheiros com conjuntos chave "destinatário" e chave DONE caso a mensagem já tenha sido enviada, e NOT DONE caso contrário). Após a criação desses ficheiros é enviada uma ordem de eliminação dos ficheiros relativos a essa mensagem, das pastas **intd** e **todo**, e de seguida as mensagens são enviadas para cada destinatário um a um, através de um *named pipe* para o programa **djumbai-lspawn**. Dependendo da reposta desse micro-serviço, o destinatário será marcado como DONE, ou não.

Quando um ficheiro tem todos os destinatários marcados como DONE, podemos garantir que as mensagens foram entregues a todos os destinatários, logo é enviada uma ordem de eliminação de uma mensagem da "queue", para o micro-serviço **djumbai-clean** através de *named pipes*.

Aquando do arranque deste serviço, é verificado se o sistema foi encerrado abruptamente, procurando mensagens por resolver através da análise dos ficheiros que ficaram pendentes na pasta **local**, desta forma conseguimos garantir que nenhuma mensagem é perdida, e que no pior dos cenários a mesma é enviada duas vezes.

2.4 DJUMBAI-CLEAN

O **djumbai-clean** é um micro-serviço, que estará sempre ativo, e é responsável por receber ordens de remoção de ficheiros do **djumbai-send** através *named pipe*.

Este micro-serviço foi criado de forma a evitar que o mesmo processo tivesse permissões de escrita e leitura na "queue", e assim evitar que um possível ataque ao **djumbai-send** pudesse ser catastrófico no que toca à confidencialidade e integridade da "queue". Mais detalhes sobre a gestão de permissões e medidas de segurança serão explicados mais à frente na próxima secção.

2.5 DJUMBAI-LSPAWN

O programa **djumbai-lspawn** é o terceiro e último programa que estará sempre ativo desde que o serviço também esteja. O mesmo é responsável por receber mensagens do **djumbai-send** através de um *named pipe* e executar o programa **djumbai-local**, para que este coloque a mensagem na caixa de correio do destinatário.

A principal responsabilidade deste programa é criar um processo filho para executar o programa **djumbai-local** e fazer setuid para o uid do destinatário, para isso este micro-serviço será o único *daemon* a correr com privilégios de **root**.

2.6 DJUMBAI-LOCAL

Este programa recebe a mensagem por parâmetro, e coloca-a na caixa de correio do destinatário. Para isso, este processo acede à diretoria **users**, e acede à pasta com o UID do destinatário (caso não exista cria-a). De seguida, coloca a mensagem num ficheiro na pasta **new**, que se destina às mensagens não lidas.

2.7 DJUMBAI-CHECK

O *djumbai-check* permite ao utilizador verificar se possui mensagens na sua caixa de correio. Caso o utilizador queira ver se tem mensagens ou ler uma mensagem recebida pode executar os seguintes comandos:

- djumbai-check
- djumbai-check -g <numero_mensagem>

Após a mensagem ser lida, é movida da pasta **new** onde se encontram as mensagem por ler, para a pasta **cur** onde estão disponíveis todas as que até ao momento já foram lidas.

2.8 DJUMBAI-GROUPS

O componente *djumbai-groups* serve como interface para os utilizadores gerirem os grupos dentro do sistema DJUMBAI. Sendo que estes utilizadores têm as seguintes opções:

- **Criar um novo Grupo:** djumbai-groups -c <nome_grupo> <uid_utilizador> ... <uid_utilizador>
- **Adicionar um user a um Grupo criado pelo próprio:** djumbai-groups -a <nome_grupo> <uid_utilizador>
- **Remover um user de um Grupo criado pelo próprio:** djumbai-groups -ru <nome_grupo> <uid_utilizador>
- **Remover um Grupo criado pelo próprio:** djumbai-groups -rg <nome_grupo>
- **Listar os utilizadores de um grupo:** djumbai-groups -l <nome_grupo>
- **Listar os grupos ao qual pertence:** djumbai-groups -lg

Esta componente é diretamente dependente do "djumbai-group-manager", que é responsável por processar os comandos enviados pelo "djumbai-groups" e realizar as operações. A comunicação entre os dois componentes é realizada através de *unnamed pipes*(leitura e escrita), no qual a mensagem a enviar é "embrulhada" numa *struct* e posteriormente serializada.

2.9 DJUMBAI-GROUP-MANAGER

O micro-serviço *djumbai-group-manager* recebe a mensagem serializada do *djumbai-groups*, desserializando-a e processando os dados nela contida para realizar a ação especificada pelo utilizador. Este processamento de um modo geral, para as flags *-c*, *-a*, *-ru* e *-rg* verifica se o emissor é o dono do grupo, não sendo necessário para as outras duas operações.

Ao criar um grupo, o *djumbai-group-manager*, inicialmente, verifica se já existe um grupo com o nome escolhido e em caso negativo, cria a diretoria *groups*, se necessário, contendo o ficheiro do grupo(com dados do grupo e respetivos elementos). Todas as outras operações são efetuadas sobre este ficheiro.

2.10 DJUMBAI-START & DJUMBAI-STOP

Estes programas destinam-se a serem executados pelo administrador com permissões de **root**, de modo a iniciar e parar o serviço respetivamente. O **djumbai-start** executará todos os *daemons* com os respetivos UIDs, e o **djumbai-stop** matará os mesmos.

Decisões tomadas no domínio da segurança do serviço

Nesta secção, iremos detalhar todas as decisões de segurança efetuadas fruto da necessidade de mitigar e minimizar potencias vulnerabilidades. Na figura 3.1, apresentamos todas as permissões do sistema de ficheiros.

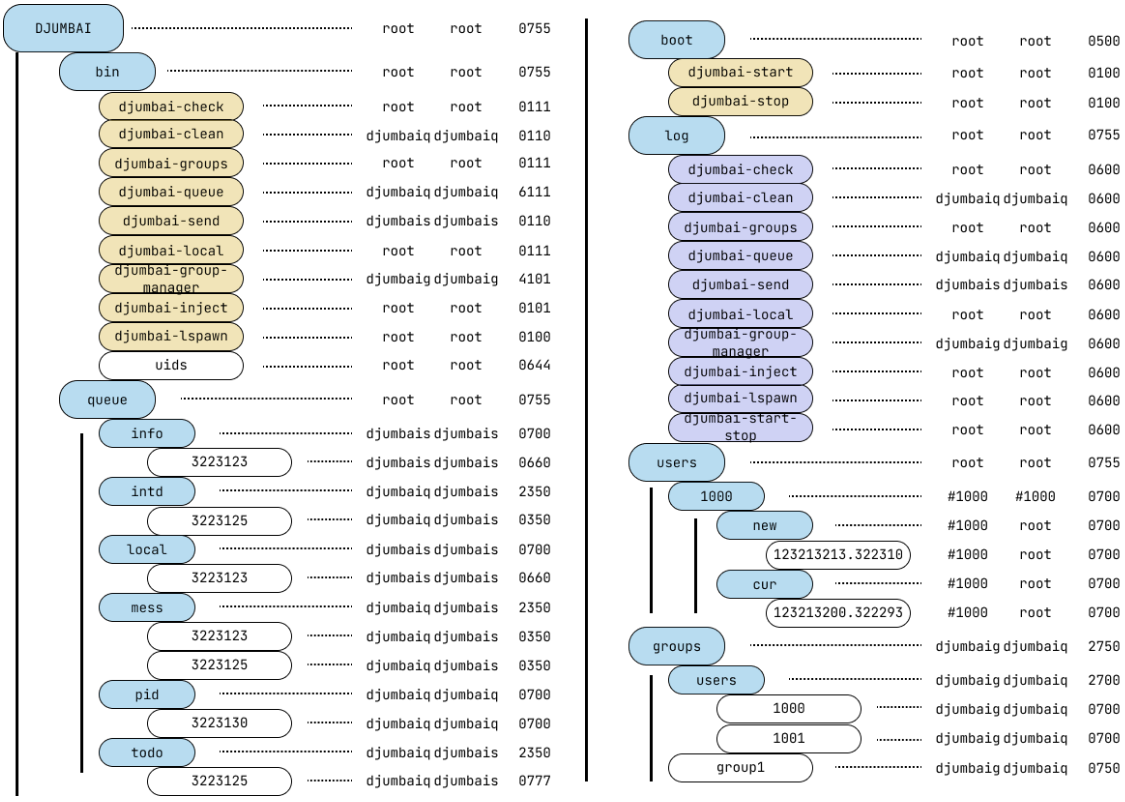


Figura 3.1: Permissões no sistema de ficheiros

Para a conceção do **DJUMBAI** e fruto da necessidade de seguir os princípios de fazer o mínimo possível com permissões de *root* e de minimizar ao máximo o uso de *setuid* em programas, decidimos criar três utilizadores *unix*: *djumbais*, *djumbaiq* e *djumbaig*.

O nosso principal objetivo com esta gestão de permissões era que nenhum utilizador pudesse ler e escrever numa pasta ao mesmo tempo, desta forma conseguimos evitar que um atacante consiga violar a confidencialidade e integridade da informação na eventualidade de um micro-serviço ser comprometido.

Para viabilizar a todo este serviço, optamos por utilizar os *uids* como identificadores dos utilizadores, permitindo assim que estes fossem sempre únicos evitando problemas de coexistência de nomes.

Relativamente à queue (diretoria onde são guardadas as mensagens e envelopes antes de serem enviados), a nossa estratégia assentou em dois utilizadores principais, com permissões distintas, **djumabiq** e **djumbais**. O utilizador **djumbaiq** foi criado apenas e só para escrever neste conjunto de diretorias, ou seja, tanto o micro-serviço **djumbai-queue**, que terá de escrever as mensagens e envelopes nas respetivas pastas, como o **djumbai-clean** que é responsável por apagar ficheiros neste conjunto de diretorias.

Quanto ao utilizador **djumbais**, que é utilizado pelo *daemon* **djumbai-send**, apenas permite a leitura de ficheiros na queue, que nada mais nada menos do que aquilo que o programa precisa, já que o mesmo se limite a retirar as mensagens e envelopes da queue e enviá-los para o **djumbai-lspawn**. De forma a que os ficheiros criados nas sub-pastas da queue obtivessem permissões de escrita para o *owner*, e permissões de leitura para o grupo utilizamos o *set* de permissões **2**350, em que o **2** permite que todos os ficheiros e pastas criadas criadas nessa diretoria, obtenham todos as mesmas permissões para o owner e grupo.

Com o serviço de mensagens já implementado e com vista a suportar a noção de grupos, queríamos que este novo micro-serviço na mesma, não pudesse em causa a segurança de todo o serviço. Para tal, criamos um utilizador *djumbaig* para permitir apenas não fosse possível escrever, ler e executar por outros *users*, ou seja, mesmo que haja unidades comprometidas, não conseguiriam ter acesso a dados críticos dos grupos. Os *users* tem todas as permissões uma vez que os dados são do próprio e existe a necessidade de leitura para fazer envios para todos elementos do grupo e de escrita para necessidades de remoção ou adição de elementos aquando do uso do **djumbai-groups** ou do **djumbai-inject**.

Neste projeto, tivemos uma abordagem cuidadosa para minimizar o uso de privilégios de *root*. Os privilégios de "superadministrador" foram essenciais apenas no micro-serviço **djumbai-lspawn**, pois tem como principal objetivo executar a componente **djumbai-local** em nome do destinatário da mensagem. Por outro lado, no executável **djumbai-queue**, definimos as permissões dele como **6**111, ou seja, como quem vai chamar este micro-serviço, são utilizadores normais, e como este não tem permissões de *root*, utilizamos permissões especiais, pois o **6**("setuid","setgid") permite que o utilizador execute o ficheiro com privilégios do proprietário do arquivo e também com privilégios do grupo proprietário do arquivo, o que era necessário, pois o proprietário é o **djumbaiq**, e era necessário para escrever nas diretorias da "queue". Isto se aplica também ao **djumbai-group-manager** que é também chamado por um utilizador normal e definimos as permissões **4**101 , **4**("setuid") para permitir a quem o execute ter os privilégios do proprietário do ficheiro para este conseguir aceder escrever e ler ficheiros da diretoria "groups".

Reflexão

Com vista a identificar problemas aquando da produção e desenvolvimento do **DJUMBAI**, utilizamos várias *flags* de compilação, sendo que destacamos as seguintes:

- -Wall: ativa avisos extra durante a compilação;
- -Wextra: ativa outros avisos extras além dos habilitados por -Wall, permitindo uma cobertura mais ampla;
- -Werror: trata todos os avisos como erros durante a compilação, fazendo com que o compilador pare a compilação se algum aviso for emitido;
- -fstack-protector: ativa a proteção contra estouro de pilha (*stack smashing protection*), adicionando código extra durante a compilação para detetar tentativas de corromper a pilha;
- -D_FORTIFY_SOURCE=2: ativa a verificação adicional de segurança para algumas funções de manipulação de *buffer*, como *strcpy*, *memcpy* e *sprintf*, a fim de evitar *buffer overflow*.
- -pie: gera um executável independente de posição, tornando-o mais seguro contra ataques de exploração de vulnerabilidades;
- -fPIE: Compila o código como um Executável Independente de Posição.

Para além da *flags* de compilação acima descritas, utilizamos também a ferramenta de depuração e deteção de problemas relacionados com gestão de memória dinâmica, que em alguns casos foi bastante útil para detetar *memory leaks* durante o desenvolvimento da solução.

É de destacar que a escolha do C++, como linguagem de desenvolvimento foi na nossa opinião uma escolha acertada para este projeto, na medida em que conseguimos obter o melhor de dois mundos; uma linguagem de baixo nível que nos permite operar de forma mais próxima ao *kernel* do sistema, mas também uma linguagem que comparativamente ao C é bem mais "amigável", já que nos permite agilizar determinadas operações e evitar alguns erros e falhas no desenvolvimento, que de outra forma poderiam por em causa a velocidade de desenvolvimento da solução, ou pior, a segurança do sistema.

Ao longo do desenvolvimento, também foram criados testes unitários com vista a verificar entradas e saídas esperadas ou até integrações entre módulos, aquando da execução de algumas funcionalidades. Estes testes, são uma parte fundamental para qualquer tipo de desenvolvimento seguro, e revelaram ser bastante úteis durante a conceção do projeto, já que, nos permitiram detetar rapidamente quando uma alteração danificou o bom funcionamento do serviço.

No que toca aos aspetos funcionais de segurança, os mesmo já foram detalhados ao longo da secção anterior.

Conclusão

Ao longo deste trabalho prático, desde a descrição geral do serviço até aos aspetos cruciais de segurança de sistema, achamos fundamental a boa gestão de permissões de utilizadores para restringir acessos indesejados, com vista a garantir a integridade e confidencialidade dos dados. Para além disso a componentização do sistema, foi também um aspeto fulcral para o controlo do risco associado a um potencial ataque.

As limitações da nossa implementação desde logo, prende-se com o uso de bibliotecas que não conseguimos ter a certeza que um dia mais tarde possam ser encontradas vulnerabilidades nas mesmas, pondo a segurança do serviço em causa, sendo que a solução para o descrito passaria por implementar todas as funções necessárias por nós próprios, assim assegurando a fiabilidade das mesmas e evitando a necessidade de confiar noutros. No entanto, devido à nossa falta de experiência, neste tipo de desenvolvimento, esta solução poderia transformar-se numa ameaça.

A comunicação síncrona não foi implementada, não pela dificuldade em a integrar no serviço, mas sim porque devido à carga de trabalhos que temos dos diferentes perfis, não tivemos tempo suficiente para tal, sendo que uma possível solução poderia ser o uso da função *select* que nos permitiria obter informação do *standard input* e de um ficheiro que poderia ser partilhado entre dois utilizadores, e mediado por um micro-serviço do sistema.

Por fim, enquanto grupo achamos que este trabalho prático ajudou a consolidar os conteúdos lecionados ao longo da cadeira, e a obter um conhecimento mais profundo sobre as preocupações do desenvolvimento de software seguro, e controlo de acesso. No entanto, ficamos com um sentimento de que caso houvesse mais tempo para a realização do trabalho, o mesmo poderia estar bem mais completo.

Bibliografia

1. https://man7.org/linux/man-pages/
2. Slides e documentos fornecidos pelos Docentes