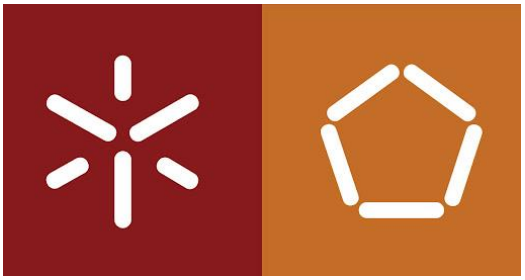

Segurança de Sistemas Informático

3ºAno, 2ºSemestre

Ano letivo 2023/2024

Concordia

Universidade do Minho

Licenciatura em Engenharia Informática

João Magalhães, A100740

Jorge Rodrigues, A101758

Rodrigo Gomes, A100555

Indíce

Introdução à Aplicação

O Concordia é um serviço de trocas de mensagens entre utilizadores locais de um sistema Linux. Além disso, esta aplicação permite aos seus utilizadores criar grupos de conversação para trocarem mensagens livremente entre os membros. Uma vez que o projeto é proposto pela Unidade Curricular de Sistemas de Segurança Informática, o foco do desenvolvimento será impedir que utilizadores externos, ou mal-intencionados consigam atacar o nosso sistema de mensagens. De seguida, vamos apresentar a nossa sugestão de arquitetura de sistema, que seja capaz de manter a estabilidade do projeto.

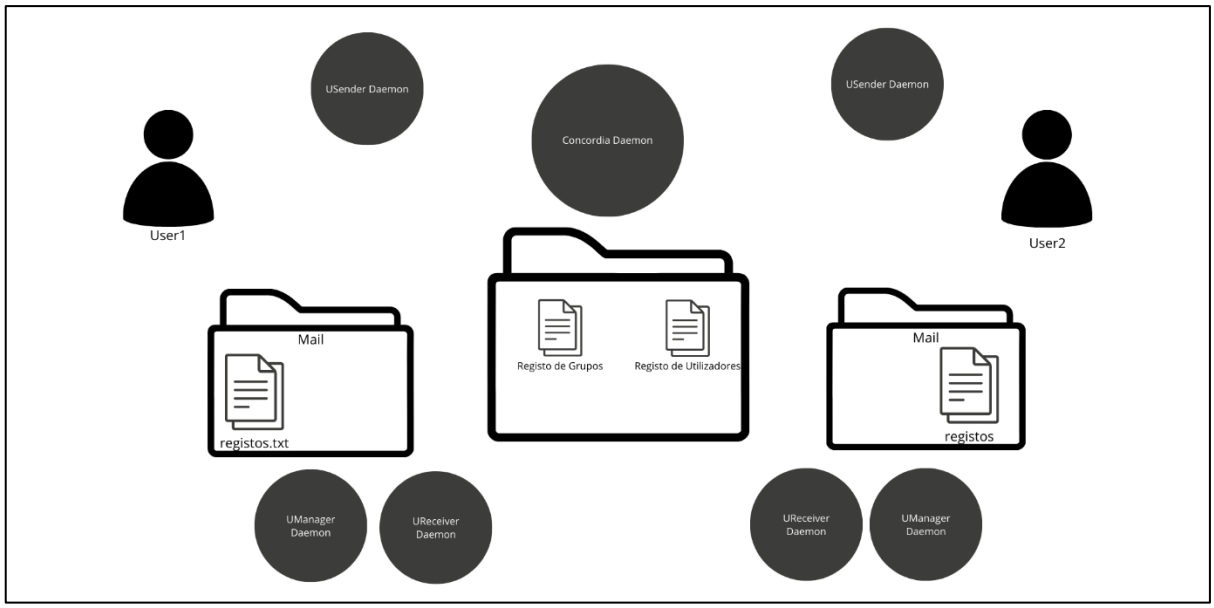

Figura 1 Arquitetura funcional do sistema concordia

Esta definição, preparada pelo grupo, pretende dificultar a tarefa de ataque e, simultaneamente, encapsular todas as estruturas necessárias de modo a que o acesso às informações não seja possível fora dos mecanismos da aplicação. As nossas preocupações de segurança assentam principalmente nos dois pontos referidos, onde não queremos de forma alguma que as peças do sistema sejam comprometidas, mas, caso seja esse o caso, minimizar os danos que um atacante pode provocar. Já que o sistema funciona todo sobre uma só máquina, não queremos também que utilizadores desse sistema sejam capazes de modificar os dados do programa. A segurança foi a propriedade determinante para o desenho de uma arquitetura baseada em microserviços, contrária a uma arquitetura monolítica.

Para esse efeito, como retrata a imagem, por utilizador, existem três novos *daemons* que agem com *ids* de outros utilizadores criados só para essa função. As funcionalidades encontram-se dívidas em quatro programas, executados pelo utilizador do serviço, onde cada um interage com o respetivo *daemon* responsável por acudir o pedido. Com isto, ambicionamos que, caso um destes utilizadores seja quebrado, nem todo o sistema, nem o próprio utilizador, fique comprometido.

Resta ainda falar do *daemon* principal, o *concordia daemon*. Uma última preocupação nossa, determinante para o modelo que construímos, é as permissões de *root*. Visto que o Concordia é apenas uma aplicação, achamos que deveríamos minimizar o uso de permissões *root* sempre que possível. Desta forma, existe um utilizador Concordia que suporta o *daemon* principal escutando, gerindo e reencaminhando mensagens dos utilizadores. A *root*, na nossa aplicação, serve apenas para criar os utilizadores necessários.

Arquitetura Funcional

Com a arquitetura apresentada, peça a peça, vamos agora discutir as funcionalidades de cada parte, assim como as estruturas que podem, ou não, ser geridas pelas mesmas.

Concordia Daemon

Este processo, como referido anteriormente, corre sobre o *id* do utilizador *concordia. A* tarefa principal deste componente é gerir os utilizadores, grupos e garantir o encaminhamento correto de mensagens. Este elemento verifica sempre as mensagens que recebe, e regista no *syslog* as diversas ações que concretiza. Para realizar estas tarefas, existem estruturas montadas para esse efeito. Primeiramente, todas as mensagens são transmitidas por um *Named Pipe,* acessível para todos os utilizadores. As restantes estruturas servem para o controlo da aplicação. Esta informação encontra se na *home directory* do utilizador, guardada numa pasta a que apelidamos de *db.*

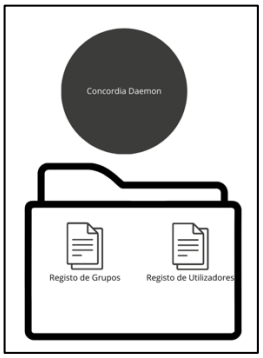

Figura 2 Representação do funcionamento do daemon concordia

Vamos ainda especificar como é que estão estruturados estes ficheiros. Começando no registo de utilizadores, é um ficheiro de texto onde constam os nomes dos *end users* da aplicação. Caso o utilizador “nuno” e “rodrigo” pretendessem utilizar a aplicação, seria necessário para o sistema que estes dois identificadores estivessem no ficheiro. O segundo ficheiro, inspirado no próprio *Unix*, é o ficheiro dos grupos. Como o nome indica, aqui são registados os grupos com os respetivos membros. Um exemplo, para o grupo “ssi”, onde o dono é o utilizador “nuno” com os membros “rodrigo” e “jony”, aparecia numa linha única com a seguinte sintaxe:

```
nuno;ssi:nuno,rodrigo,jony
```

Figura 3 Exemplo de entrada no ficheiro de registo dos grupos

Manager Daemon

Do lado do utilizador, é preciso um componente que detenha a caixa de correio. Consideramos este processo indispensável, assim como as estruturas que este manipula. O meio de comunicação com o utilizador é um *Named Pipe,* que é acedido pelo utilizador para fazer pedidos que envolvam as mensagens que este recebeu.

Figura 4 Representação do funcionamento do daemon manager

A diretoria *mail,* presente na *home directory* do respetivo *manager*, segura as diversas mensagens em ficheiro binário, assim como um ficheiro de texto dos registos. Este último ficheiro é fundamental para o funcionamento do programa, já que é aqui onde se guarda informações como a data de chegada e o estatuto (lida ou não lida) da mensagem. Um exemplo de uma entrada deste ficheiro, para a mensagem com id “0001”, seria algo como “0001:2024-05-05:84:1”, onde primeiro encontra-se o *id* da mensagem, a data, o tamanho e o seu estatuto.

```
0001:2024-05-05:84:0
0002:2024-06-05:32:1
....
```

Figura 5 Exemplo de entrada no ficheiro de registos do manager

Receiver *Daemon*

Este *daemon* funciona em sintonia com o *manager* descrito anteriormente. A sua função é receber e depositar corretamente as mensagens do servidor na diretoria correta, mais concretamente, na diretoria *mail* do seu utilizador, gerida pelo *manager daemon*. Além disso, este processo deve gerir a atribuição de ids (uma simples variável inteira, incrementada com o tempo), e colocar as entradas no ficheiro de registos.

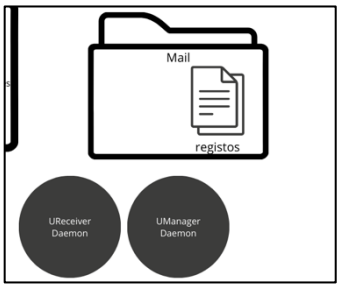

Figura 6 Representação do funcionamento do daemon receiver

Sender *Daemon*

O *sender* é a ponte entre o utilizador e o serviço no que toca a envio de mensagens. Este *daemon* dispõe de um *Named pipe* por onde o seu utilizador pode enviar as mensagens que deseja cheguem a um utilizador ou aos utilizadores de um grupo a que pertença. A nível de estruturas, contemplamos a ideia de usar uma espécie de fila de espera sob a forma de diretoria, mas tendo em conta que as mensagens são pequenas e exclusivamente texto, achamos desnecessário.

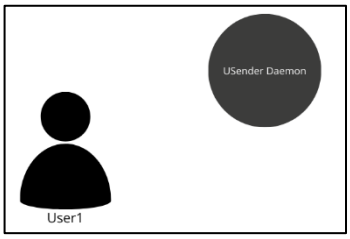

Figura 7 Representação do funcionamento do sender daemon

Mensagem

Para o sistema de comunicação ser possível, tem que haver um formato de comunicação reconhecido por todas as partes. Assim, achamos necessário criar uma estrutura para as nossas mensagens. Esta estrutura guarda informações sobre o remetente, o destinatário, o tipo de mensagem e, finalmente, o texto. Para além da estrutura, dispomos também de funções capazes de verificar, escrever e serializar mensagens.

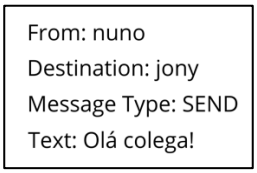

Figura 8 Exemplo de uma mensagem

Programas do Utilizador

Resta mencionar de que forma o *end-user* interage com o nosso sistema. Os programas escolhidos foram os seguintes:

Concordia-boot

Estre programa é responsável pelas ações “concordia-ativar” e “concordia-desativar”. Assim como descrito no enunciado, a ativação exige o começo dos processos demónios e montar as suas estruturas. Além disso, este comando faz ainda um pedido ao *concordia daemon* para ser incluído na lista de utilizadores. O comando “concordia-desativar” é precisamente o oposto. Este comando termina os demónios e remove todas as estruturas auxiliares criadas.

Concordia-roles

Já este programa, encapsula todo o tipo de comandos que envolvam gestão de grupos, mais concretamente, criar e apagar um grupo, listar os seus membros e adicionar ou remover participantes. Este programa juntamente com o *concordia-boot*, são os únicos que comunicam diretamente com o *daemon-concordia*, usando o próprio *username* do *end-user*.

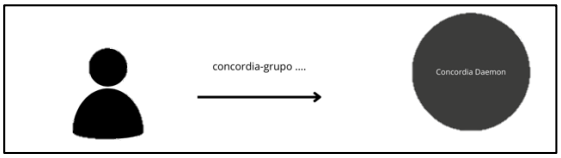

Figura 9 Exemplo do funcionamento do concordia-roles

Concordia-enviar

Este programa é responsável por enviar mensagens em nome do utilizador para outro utilizador ou grupo. Desta forma, responde aos comandos "concordia-enviar dest msg" e "concordia-responder mid msg". Começando com o comando "concordia-enviar", é enviada uma mensagem para o *sender daemon* que fará o seu reencaminhamento. Naturalmente, o programa escreve a mensagem para o *pipe* disponibilizado pelo *sender.* Para o comando "concordia-responder", primeiramente, é pedida a mensagem ao *manager* e só depois é feito o envio pelo *sender*. O concordia *daemon* fica responsável por reencaminhar a mensagem para o utilizador ou grupo destino.

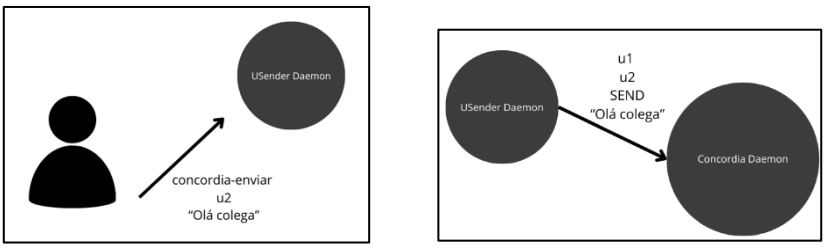

Figura 10 Exemplos de envio de uma mensagem

Concordia-mail

Para terminar os programas dos utilizadores, surge o *concordia-mail* que, de forma sucinta, encapsula os acessos ás mensagens por parte do utilizador. Isto inclui os comandos "concordia-listar [-a]" e "concordia-ler" sugeridos no enunciado. Este programa interage exculisvamente com o *manager daemon* que controla a caixa de mensagens do respetivo utilizador.

Figura 11 Exemplo de leitura de uma mensagem

Segurança do Serviço

A componente de segurança é o foco central deste projeto. Não só interferiu no desenho das várias componentes do sistema, como também precisou da aplicação de técnicas de controlo lecionadas na unidade curricular.

Concordia Daemon e os Registos do sistema

A parte central do sistema é o concordia *daemon*. Por este motivo, a segurança relativa às suas estruturas tem de ser rigorosa e apertada. Como referido, existe uma noção de registos e um meio de comunicação. Começando com os registos, facilmente entendemos que apenas o utilizador concordia deve ser capaz de modificar estes ficheiros. Desta forma, ao utilizar o octeto "0700" para a diretoria *db* e o octeto "0600" para os ficheiros de registos, asseguramos que apenas o utilizador concordia pode aceder aos mesmos. No que toca ao *pipe* preparado para a receção de mensagens, é preciso pensar de forma distinta. Para este caso concreto, precisamos que todos as partes que participam no sistema sejam capazes de escrever para este mecanismo. A solução para atender estes requisitos foi o uso do grupo concordia. Cada utilizador num sistema Linux é, pelo menos, membro de um grupo com o mesmo nome de utilizador. Tendo em conta que o utilizador concordia não pertence a mais nenhum grupo, ao atribuir permissões de grupo estará apenas, e apenas só a atribuir permissões aos membros desse mesmo grupo. Desta forma, o *pipe* possui as permissões "0730", onde o dono pode abrir descritores tanto para leitura como escrita, e os membros do grupo *concordia* podem abrir apenas para escrita. O *daemon* concordia precisa de permissões de escrita, pois desta forma consegue evitar esperas ativas.

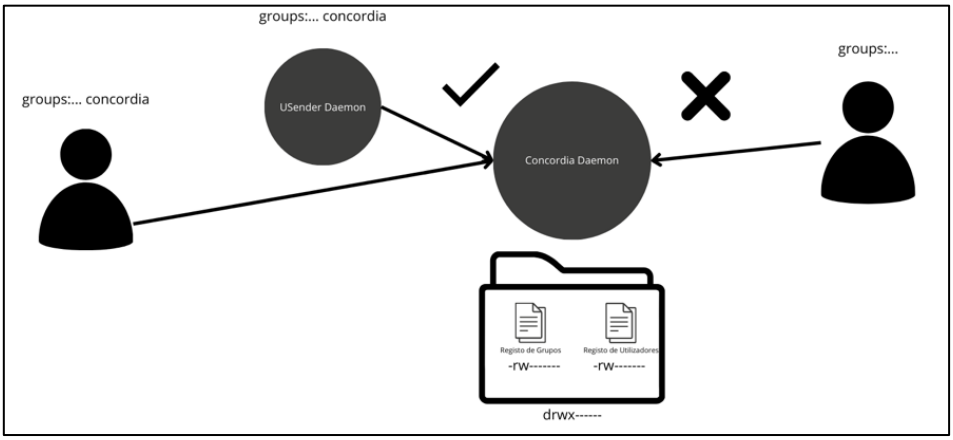

Figura 12 Exemplo de acesso ao daemon concordia

Utilizadores, *Daemons* e Comunicação

Os *daemons* e os respetivos utilizadores que correm os *daemons*, tornam possível a arquitetura de microserviços, com a benesse de poder aplicar permissões mais refinadas. Por

cada *daemon*, existe um utilizador dedicado a correr o mesmo. Estes utilizadores pertencem ao grupo principal do *end-user* e ao grupo concordia. Começando pelo *sender*, este processo gere apenas uma estrutura onde escuta à espera de mensagens para encaminhar por parte do utilizador principal. Como referido, utilizamos um *pipe* com nome, com as permissões "0700". Só com estas permissões, seria impossível alcançar o efeito pretendido, por isso utilizamos *access control lists* para definir prioridades para um só utilizador em específico, neste caso em concreto, o *end-user.*

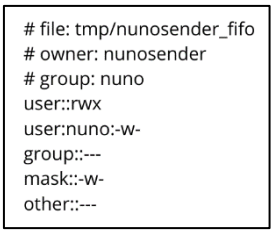

Figura 13 Permissões para o pipe do processo sender

É importante referir que a diretoria escolhida para acolher esta estrutura foi a "tmp", pois é aberta a todos os utilizadores e serve precisamente para colocar ficheiros temporários usados por *daemons* ou outro tipo de processos. A segurança deixa de ser um problema, pois as permissões já estão bem estabelecidas.

Avançando para o *receiver daemon*, que corre sobre o *uid receiver* destinado a esse efeito, no que toca a *pipes*, a situação é bastante idêntica, a diferença é que o único *user* que pode escrever para essa mesma estrutura é o *user* concordia.

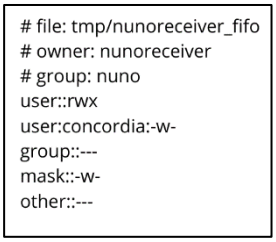

Figura 14 Permissões para o pipe do processo receiver

O último *pipe* é detido pelo *manager*. Como nos exemplos anteriores, o que varia é apenas a localização deste elemento que, para este caso concreto, optamos por deixar na *home directory* do próprio utilizador. Achamos que faria mais sentido já que é um processo com mais responsabilidades, e que a sua função principal é gerir e fornecer diretamente ao utilizador principal. A nível de permissões, é uma situação idêntica ao caso da figura x. Como o contacto com esta peça exige uma resposta, o *end-user* abre um *fifo* temporário na pasta "/tmp" onde só o manager pode enviar a resposta ao pedido.

Caixas de Correio

O nosso modelo precisa da existência de estruturas que segurem as informações relativas às mensagens recebidas. Como já referimos, é uma pasta na diretoria *home* do respetivo

manager, como os respetivos registos. Desta forma, esta secção vai focar-se nas permissões necessárias destas estruturas para que o sistema consiga operar.

```
# file:home/nunomanager/mail/
# owner: nunomanager
# group: nuno
user::rwx
user:nunoreceiver:rwx
group::---
mask::rwx
other::---
```

Figura 15 Permissões para a diretoria mail

Da figura conseguimos perceber que apenas os *daemons manager* e *receiver* podem manipular a diretoria. Isto faz sentido, uma vez que tanto um como o outro precisam de acesso e de alterar o seu conteúdo.

Avançando para os registos, encontramos uma situação bastante semelhante:

```
# file:home/nunomanager/mail/registos.txt
# owner: nunomanager
# group: nuno
user::rw-
user:nunoreceiver:rw-
group::---
mask::rw-
other::---
```

Figura 16 Permissões para o ficheiro de registos

Na arquitetura funcional, especificamos que ambas as partes precisam de modificar o este ficheiro. No caso do *receiver,* tem que adicionar entradas, enquanto o *manager* deve marcar as que já foram lidas.

Com este conjunto de permissões, garantimos um total encapsulamento da aplicação. Será impossível para outros utilizadores, membros ou não do grupo concordia, consultar ou alterar os ficheiros da caixa de correio. As consultas ficam exclusivamente ao dono das mensagens, que também só pode consultar através dos programas indicados.

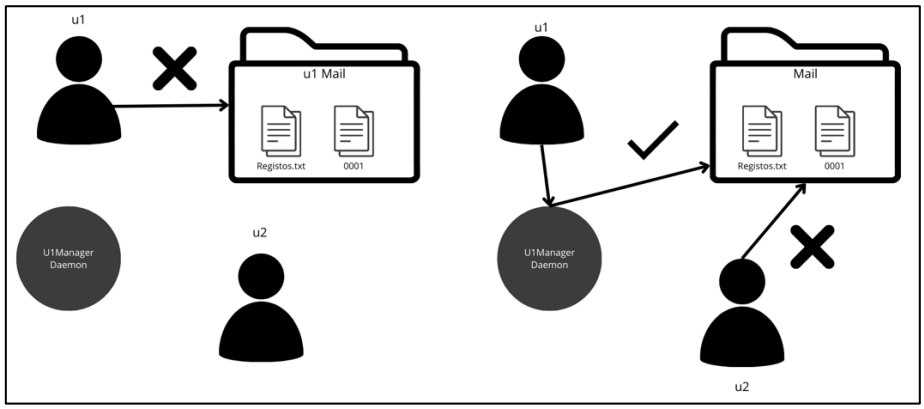

Figura 17 Exemplo de funcionamento das permissões

Mensagens

A comunicação entre processos é feita sobre um formato definido e conhecido a todos os programas. Achamos que é mais funcional do que outras soluções, pois podemos encapsular um conjunto de regras e funções de verificação neste conjunto. Esta limitação existe para questões de segurança, onde um atacante que desconhece o código fonte das mensagens, dificilmente consegue replicar e abalar o sistema.

A verificação é algo que nós consideramos importante. Por isso, definimos uma função de verificação para que todos os utilizadores possam utilizar e assegurar que as mensagens recebidas são corretas. Deste modo, a nossa função verifica se os campos seguem o tamanho e tipos definidos, e também se o remetente é um utilizador registado no sistema operativo.

Para a nossa arquitetura, é necessário depositar mensagens nas caixas de correio dos utilizadores. A opção que tomamos foi utilizar ficheiros binários. Deste modo, o processo de deseriliazição é dependente de uma função ou do conhecimento da estrutura, em oposição a utilizar ficheiros de texto conhecidos a humanos. Apesar de não ter sido aplicada, as técnicas de criptografia são melhores e normalmente desenvolvidas a pensar em ficheiros binários.

Reflexão sobre o Sistema

Nesta secção vamos justificar as nossas decisões no que toca à arquitetura e segurança, referir ferramentas e *flags* úteis para o nosso desenvolvimento e, finalmente, aspetos que poderíamos acrescentar ao sistema para torna-lo mais eficiente na sua tarefa.

Começando pele *daemon* concordia, achamos que a segurança está bem aplicada. Começando por limitar quem pode escrever para o *pipe*, é criada logo uma barreira que separa utilizadores membros do serviço, de utilizadores que não estão propriamente registados. Além disso, achamos que é obvio que a informação interna do programa não pode ser acedida por mais ninguém além do concordia, porque poderia levar a erros na troca de mensagens. Conseguimos também não ter de criar novos grupos, pois emulamos as estruturas usadas pelo sistema operativo para a aplicação. Assim, conseguimos que o nosso sistema não fosse dependente de *root*, servindo-se na mesma dos mecanismos de segurança do *os*.

A decisão de utilizar uma arquitetura de microserviços foi complacente com os objetivos da aplicação. Contudo, exige alguns desafios para conseguir alcançar a refinação de permissões capaz de sustentar o funcionamento da aplicação. Como referido, o uso de *acl´s* foi essencial para garantir que apenas os devidos processos conseguissem aceder aos meios de comunicação, ou, no caso do *manager* e *receiver*, ao próprio correio. No caso da comunicação para grupos, optamos por decidir que o reencaminhamento da mensagem é feito pela peça central, e chega como uma mensagem normal à caixa de correio. Alternativamente, pensamos que poderíamos também ter uma espécie de caixa partilhada, onde os membros do grupo conseguiam de algum modo chegara essas mensagens. Para isso, quebraríamos a independência da *root*, pois seria preciso criar grupos para o *os.* Desta

forma, achamos que a nossa solução garante mais segurança e cumpre os requisitos necessários para a aplicação.

Durante o desenvolvimento, utilizamos ferramentas como o *gdb* para encontrar e corrigir erros que pudessem levar a crashes da aplicação, e também optamos por utilizar a *flag* "-Wall" que ativa todos os *warnings* na fase de compilação. Tudo isto para garantir o menor número de erros possíveis para o produto final.

O encapsulamento dos diferentes módulos foi tido em conta, com sucesso na maioria dos casos, como por exemplo, o caso da mensagem ou de um dos programas dos *daemons* do lado do utilizador. No entanto, achamos que fragmentar o código do *daemon* concordia uma tarefa bastante difícil, dada a especificações que nós queríamos que o programa garantisse.

Para terminar a secção, queríamos mencionar que a aplicação tem espaço para melhoria em diversas partes da sua estrutura. Começando pelo *sender*, sendo que o único requisito era exclusivamente mensagens de texto, poderíamos abdicar deste processo. Contudo, se fossem acrescentadas funcionalidades que envolvessem envios mais lentos, esta estrutura seria indispensável. Além disso, por questões de tempo, não conseguimos experimentar com a biblioteca *oauth*, para tornar a experiência ainda mais segura, como também não conseguimos aplicar alguns dos conceitos criptográficos que aprendemos também na unidade curricular. Da maneira que desenhamos a arquitetura, não seria muito complicado acrescentar essas medidas de segurança.

Conclusão

O Concordia é uma aplicação com o propósito de ser um serviço de troca de mensagens seguro e estável para utilizadores locais em sistemas Linux. A sua arquitetura, baseada em microserviços, combinada uma abordagem meticulosa de técnicas de segurança, o que proporciona uma plataforma robusta para comunicação entre utilizadores e grupos de utilizadores.

Ao priorizar a segurança em cada componente do sistema, desde a autenticação de utilizadores até o armazenamento de mensagens, o concordia demonstra um compromisso com a proteção dos dados e a integridade do serviço. A abordagem de minimizar o uso de permissões de *root* e aplicar permissões refinadas para cada parte do sistema contribui para uma postura defensiva eficaz contra possíveis ataques.

Embora o sistema atual represente uma base sólida, há espaço para melhorias adicionais, como a implementação de medidas criptográficas e a exploração de bibliotecas de segurança adicionais. No entanto, no estado atual, o Concordia oferece uma solução confiável e segura para a troca de mensagens em ambientes Linux, atendendo às exigências de estabilidade e segurança exigidas pela Unidade Curricular de Segurança de Sistemas Informáticos.