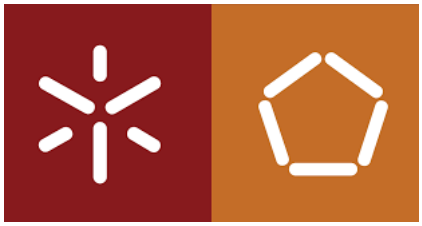

Universidade do Minho
Escola de Engenharia | Departamento de Informática
Licenciatura em Engenharia Informática

Segurança de Sistemas Informáticos

Ano Letivo de 2023/2024

TP2

Ricardo Miguel Queirós de Jesus (a100066)
Rui Pedro Fernandes Madeira Pinto (a100659)

SSI

Índice

1 Introdução **1**

2 Arquitetura Pensada **2**
2.1 Concordia-mensagens 2
2.2 Concordia-grupos 4
2.3 Decisões tomadas 4
2.4 Aspetos de Segurança 5

3 Arquitetura funcional **7**
3.1 Descrição dos programas 7
3.1.1 concordia-enviar [-g] dest 7
3.1.2 mail-queue 8
3.1.3 mail-send 8
3.1.4 mail-lspawn 8
3.1.5 grupo-criar 8
3.1.6 grupo-remover 8
3.1.7 grupo-user-adicionar 9
3.1.8 grupo-user-remover 9
3.1.9 grupo-listar 9
3.1.10 Outros programas 9
3.2 Estruturas de dados 10

4 Extras **11**
4.1 Sistema de registo de eventos 11

5 Conclusão **12**

1 Introdução

Este documento serve para descrever o trabalho realizado no âmbito da unidade curricular de Segurança de Sistemas Informáticos, onde foi proposto o desenvolvimento de um serviço de conversação entre utilizadores locais de um sistema Linux denominado de "Concordia". Este serviço foi projetado para suportar a comunicação assíncrona, gestão de utilizadores e grupos.

2 Arquitetura Pensada

A arquitetura deste programa pode se dividir em duas secções. A primeira secção, denominada concordia-mensagens, lida com todas as funcionalidades relacionadas às mensagens. Isso inclui enviar mensagens para outros users, listar as mensagens recebidas, responder diretamente a mensagens e ler as mensagens recebidas. A segunda secção, chamada concordia-grupos, é dedicada à gestão de grupos. Esta parte permite a criação e remoção de grupos, adição e remoção de membros de um grupo, e a listagem dos membros de um grupo.

2.1 Concordia-mensagens

A solução desenhada para este serviço foi inspirada na arquitetura do *qmail*, sugerida no enunciado deste trabalho prático, que é conhecida pela sua abordagem modular e focada na segurança. Esta arquitetura guiou-nos para a criação de uma arquitetura que separa responsabilidades em diferentes componentes, cada um deles com responsabilidades mínimas necessáias para realizar as suas funções, garantindo, deste modo, uma maior segurança e confiabilidade do serviço. Na seguinte imagem é ilustrada a arquitetura planejada pelo nosso grupo:

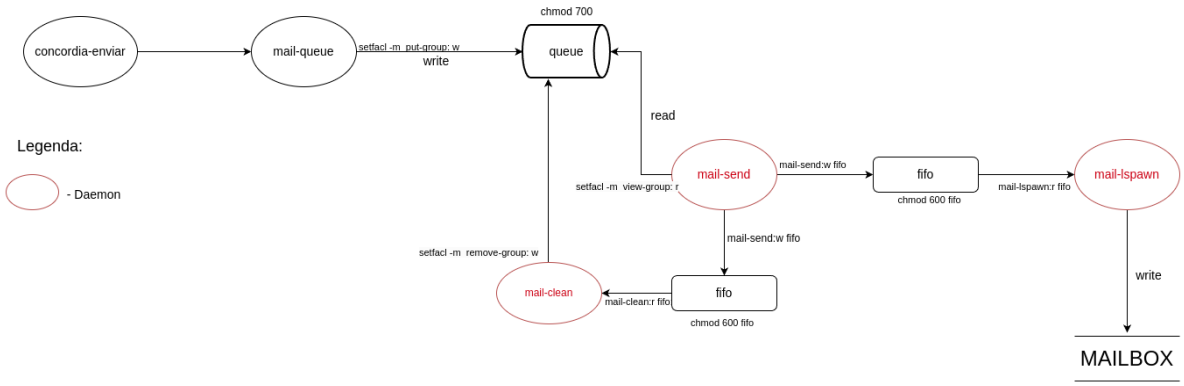

Figura 2.1: Arquitetura Planeada do concordia-mensagens

Como é possível observar pela figura, o objetivo da aquitetura é descentralizar ao má-

ximo as diferentes operação do sistema, permitindo assim que mesmo que a segurança de uma funcionabilidade seja quebrada, o invasor não possuí controlo sobre o resto das funções do sistema. Para além disto para haver uma maior segurança sobre o sistema, os executáveis dos programas são rodados por diferentes users do sistema, sendo que estes user pertencem a grupos com níveis de permissões diferentes perante o sistema, sendo que a lógica usada para atribuir estas permissões foi dar sempre o mínimo possível a cada um. Os grupos criados foram:

- **put-group :** este grupo possuí as permissões de escrita na diretoria da queue, e o intuito é apenas introduzir novos ficheiros de textos na queue;
- **view-group :** este grupo tem apenas permissão de ler os ficheiros da queue;
- **remove-group :** neste caso damos as permissões necessárias para remover ficheiros da diretoria;

2.2 Concordia-grupos

Para a secção da arquitetura que se concentra na gestão de grupos, procuramos seguir a mesma filosofia que o concordia-mensagens, com um foco constante numa arquitetura de micro-serviços. Esta abordagem visa proteger o sistema, garantindo que, mesmo em caso de uma violação de segurança num dos programas, as funcionalidades dos outros programas permanecem seguras.

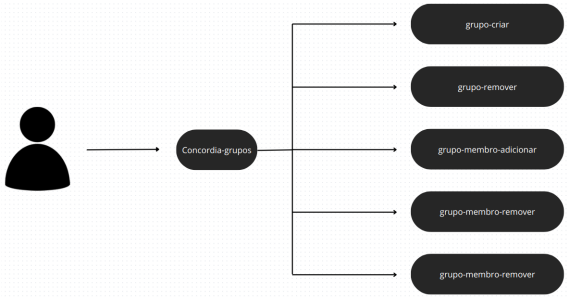

Figura 2.2: Arquitetura Planeada para o concordia-grupos

Quanto à gestão geral dos grupos nós aproveitamos a já existente gestão dos grupos no sistema operativo, portanto quando um qualquer user quer criar um novo grupo este vai ser criado no sistema operativo e a *ownership* do grupo é dada ao sistema que o criou. Esta *ownership* dada ao user que permite que o criador do grupo possa adicionar e remover os users do grupo sem necessidade de estar a dar esta funcionabilidade ao root, para além disso responde ao requisito que apenas o criador de um grupo pode gerir o mesmo.

Quando é criado um grupo é adicionado uma diretoria com o nome do grupo na diretoria */home/concordia/grupos*, sendo que esta nova directoria é criada sem qualquer permissões para users que não pertencem ao grupo, para impedir que pessoas fora do grupo consigam ver as mensagens.

2.3 Decisões tomadas

Neste capítulo mencionamos certas decisões de grupo que foram sendo tomadas e explicamos o porquê de termos optado por essas decisões.

Ownership do grupo

No que concerne à gestão de grupos, a nossa equipa ponderou entre duas soluções possíveis. A primeira seria atribuir ownership do grupo a um user do sistema. Sempre que fosse necessária alguma gestão, o sistema verificaria se o pedido vinha do administrador do grupo e, em seguida, executaria o pedido. No entanto, esta solução apresenta um problema: a centralização da gestão de grupos num único utilizador. Isto significa que, se este utilizador fosse comprometido, os invasores teriam acesso imediato à gestão de todos os grupos do sistema Concordia. A outra solução encontrada foi dar a *ownership* ao user que pede para criar o grupo, o problema desta é a atribuição de permissões aos users, que geralmente temos de tratar como de pouca confiança.

Tendo em conta os prós e contras de cada solução decidimos optar pela segunda opção, pois ainda que devemos tratar os users dando lhes pouca confiança, acreditamos que, num aspeto geral de segurança será perfível "correr o risco"de dar a permissão de grupo aos users do que estar a centralizar as permissões todas num só de mais confiança mas que em caso de *breach* implicaria terem acesso a todos os grupos do sistema.

Separação dos comandos de grupos

Também relacionado aos grupos, nós apercebemos-nos que as funcionabilidades relacionadas a estes se podiam agrupar em 3 grupos com permissões no sistema diferentes. Um que necessitaria de *root acess*(criação e remoção de grupos do sistema operativo), outro apenas precisaria de permições de o user ser *owner* do grupo (adicionar e remover user do grupo) e um último que não necessitaria de qualquer tipo de permição (listar os membros do grupo).

Tendo isto em mente, em uma fase inicial estes programas foram feitos tendo em conta esta divisão de permissões necessárias, agrupando os programas. Posteriormente com uma melhor análise acerca da segurança do mesmo o grupo achou por melhor alterar este agrupamento de funcionabilidades por programa, isto porque nos apercebemos que ainda que sim os programas partilham do mesmo grau de premissão, agrupa-los significa que, havendo uma *breach* no programa o invasor ficaria logo com acesso a duas funcionalidades. Tendo isto em conta, resolvemos separar cada funcionalidade pelo seu próprio programa, seguindo assim uma arquitetura de micro serviços mais segura.

2.4 Aspetos de Segurança

Para além dos diversos mecanismos já anteriormente mensionados para tornar o nosso sistema seguro, o grupo ainda pensou em mais políticas que pudessem ir sendo adotadas no sistema para prevenir contra ataques.

- **Token authentication :** Algo que o grupo gostaria de ter implementado seria um sistema de *Token authentication*, isto consistiria em na activação do user no sistema iria ser criado um *token* único para ele que seria guardado em um ficheiro seguro no sistema, sendo que sempre que houvesse uma comunicação de user sistema, o user mandaria o seu *token* de autenticação para o servidor fazer um reconfirmação da sua identidade. Este mecanismo iria adicionar uma nova camada de porteção contra ataques de *impersonation* onde um invasor tentaria se passar por um user do sistema.

- **Validar sempre o input :** seguir uma filosofia de *validate first, act later*, prevem muitos ataques ao sistema, impedindo que comportamentos inesperados e potencialmente maliciosos sejam permitidos.

- **Fechar os ficheiros:** os ficheiros do sistema serem o máximo fechados possível, temos como exemplo mais óbvio as diretorias de cada user serem fechadas para outros demais, ou mesmo a diretoria *queue* só dar acesso (sempre restristo ao máximo) a grupos que tenham de interagir com ela. Mas também há o caso dos *FIFOS* de comunicação entre os processos demónio que deveram ser restritos,tirando até permissão de ler do programa que só precisa de escrever e visse versa.

- **Limitar os buffers :** aplicar limites aos buffers que são usados no pragrama é uma prática crucial para prevenir vulnerabilidades como buffer overflows, que podem ser exploradas para executar código arbitrário ou corromper dados. Ao impor limites estritos ao tamanho dos buffers usados, garantimos a integridade e a segurança dos dados processados, mitigando o risco de ataques e melhorando a estabilidade do sistema.

3 Arquitetura funcional

Na imagem abaixo ilustra a arquitetura utilizada pelo nosso grupo. Cada um destes programas têm as suas próprias funcionalidades. Esta arquitetura tal como foi dito anteriormente no presente relatório é inspirado no qmail, contudo, tem algumas diferenças em relação ao mesmo.

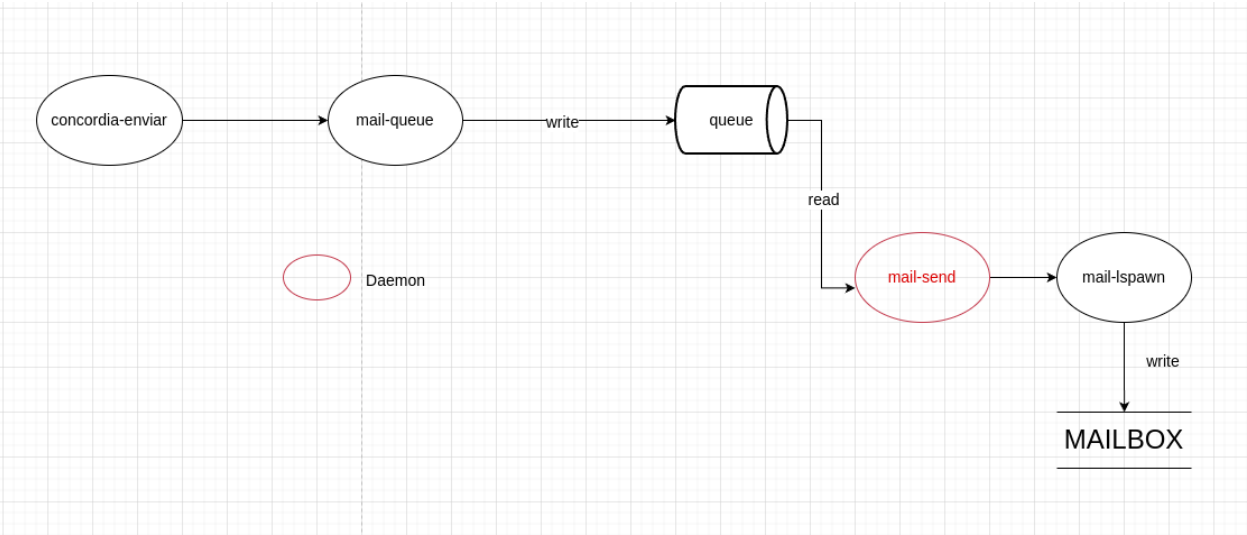

Figura 3.1: Arquitetura do Serviço - envio de mensagens

3.1 Descrição dos programas

3.1.1 concordia-enviar [-g] dest

O programa **concordia-enviar** é o programa utilizado pelo utilizador do sistema, para o envio de mensagens. Na execução deste programa o utilizador passa como argumento na linha de comandos o nome do utilizador de destino, no caso de o destinatário ser um grupo, a flag *-g* é passada como argumento na linha de comandos. Após isto, o programa lê do *standard input* a mensagem a ser enviada pelo utilizador. Neste programa um processo filho é criado, para a execução do programa **mail-queue**, passando como argumento o nome de utilizador de destino da mensagem.

3.1.2 mail-queue

O programa **mail-queue** é responsável por criar um ficheiro de texto na diretoria que é utilizada como *queue*. O ficheiro de texto criado por este programa tem o seguinte formato: **tag;remetente;destinatario;mensagem**. Este programa tem permissões de escrita na diretoria da queue localizada na diretoria `\home\condordia\queue`.

3.1.3 mail-send

O programa **mail-send** é responsável por estar de forma continua a monitorar, em modo daemon, a diretoria da queue. Este programa tem permissões de leitura dos ficheiros contidos na queue. Quando um novo ficheiro de texto é inserido na queue, o **mail-send** lê o conteúdo e envia, via fifo, o conteúdo presente dentro do ficheiro de texto para o programa **mail-lspawn**, que foi executado por um processo filho no programa **mail-send**. Após isto o programa remove da queue o ficheiro. O ideal, e que foi planeado inicialmente pelo grupo, foi apenas ser possível a leitura do ficheiro através do **mail-send** e este, enviaria via fifo, o nome do ficheiro a ser eliminado para um programa, em modo daemon, chamado **mail-clean** que posteriormente seria responsável por eliminar o ficheiro da queue.

3.1.4 mail-lspawn

O programa **mail-lspawn** é responsável por receber do fifo, o conteúdo a ser enviado para a *mailbox* do utilizador destinatário. Com o conteúdo presente no ficheiro de texto, o **mail-lspawn** extrai o utilizador de destino e se o destinatário é um utilizador ou um grupo, utilizando essa informação, para aceder à *mailbox* do utilizador ou do grupo e escreve na diretoria do utilizador com o caminho `\home\nomeUser\nomeUserMail` ou, `\home\concordia\Grupos\nomeGrupo` no caso dos grupos.

3.1.5 grupo-criar

O programa **grupo-criar** é responsável pela criação do grupo no sistema e criar a sua respetica diretoria na diretoria de Grupos do sistema

3.1.6 grupo-remover

O programa **grupo-remover** é responsável pela remoção do grupo no sistema e remover a sua respetica diretoria, sendo que também implica apagar potencias mensagens que

estajam dentro da diretoria.

3.1.7 grupo-user-adicionar

Este programa é reponsável pela funcionabilidade de adicionar o user ao grupo desejado.

3.1.8 grupo-user-remover

Este programa é responsável por remover o utilizador do grupo.

3.1.9 grupo-listar

Este programa lista todos os programas do grupo desejado.

3.1.10 Outros programas

Foram feitos outros programas de modo a cumprir com todas as funcionalidades pedidas no enunciado deste trabalho prático.

- **concordia-responder**
  - programa responsável por responder a uma mensagem. Acede ao conteúdo da mensagem, lê o remetente da mesma, e utilizando o programa **concordia-enviar**, envia uma resposta ao outro utilizador.
- **concordia-ativar**
  - programa responsável pela criação da *mailbox* do utilizador que executa este programa.
- **concordia-desativar**
  - responsável por remover a *mailbox* do utilizador que executa este programa.
- **concordia-ler**
  - programa que recebe como argumento o id da mensagem e, apresenta ao utilizador o conteúdo da mensagem.

- **concordia-listar**
  - programa responsável por aceder à *mailbox* do utilizador que executa o mesmo
- **concordia-grupos**
  - programa responsável por gerir as interações do utilizador e funcionabilidades do sistema relacionados a grupos

3.2 Estruturas de dados

- **Queue de mensagens**
  - As mensagens enviadas para a fila de espera encontram-se numa diretoria `/home/concordia/queue`
- **Fifo utilizado pelo mail-send e pelo mail-lspawn**
  - O fifo utilizado na pelo programa **mail-send** para enviar o conteúdo de um ficheiro para o programa **mail-lspawn** encontra-se na diretoria `/var/tmp`.

4 Extras

4.1 Sistema de registo de eventos

O daemon criado, executa o programa **mail-send** que, por sua vez, executa o programa **mail-lspawn**. Na arquitetura funcional implementada pelo nosso grupo, apenas o programa *mail-send* corre como processo daemon, contudo, o planeado pelo nosso grupo seria ter dois daemon's a executar cada um deste programas, sendo que os mesmos comunicariam entre si através de um fifo. Falando mais concretamente do registo de eventos, com auxílio à função *syslog*, as mensagens que achamos necessárias, utilizando dois níveis de severidade, o LOG_INFO e o LOG_ERR, sendo utilizados para informações que achamos necessárias estarem presentes, sendo utilizadas muitas vezes para fins de *debugging* e de deteção de erros, respetivamente.

5 Conclusão

Apesar de não termos conseguido implementar a arquitetura inicialmente planeada pelo grupo, que seria a mais ideal para este projeto, o trabalho realizado foi, na visão do grupo, bastante positivo. Conseguimos compreender profundamente os conceitos de segurança e aplicá-los em um contexto prático, embora nem todas as medidas de segurança planejadas tenham sido implementadas conforme desejado, devido a diversos problemas com as tecnologias utilizadas, acreditamos que estavamos no bom caminho. Este projeto proporcionou nos uma excelente oportunidade de aprendizagem e noções do que seria um sistema mais seguro.

Para trabalho futuro o objectivo seria conseguir implementar todas as nossas ideias de permições para os diversos executáveis e ficheiros utilizados.