

SEGURANÇA E CONFIABILIDADE

# Relatório Projeto 1

## Engenharia Informática – 2018/2019

**Grupo 17**

**Diogo Nogueira, Nº 49435**

**Filipe Capela, Nº 50296**

**Filipe Silveira, Nº 49506**

Índice

[Balanço dos objetivos cumpridos 2](#_Toc4413949)

[Mensagens e formato das mesmas 2](#_Toc4413950)

[Configuração da Sandbox 3](#_Toc4413951)

[Organização do Software 4](#_Toc4413952)

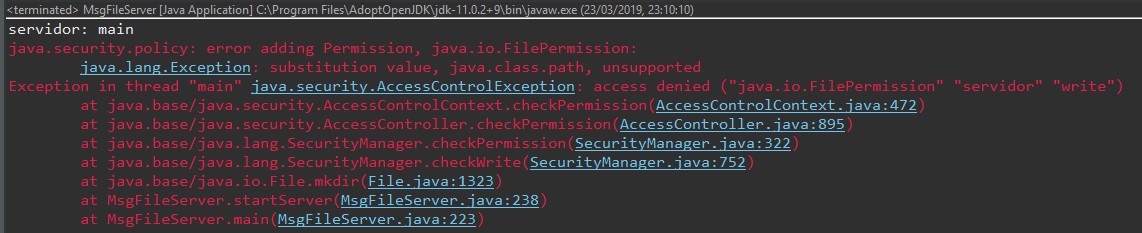
[Requisitos de Segurança 6](#_Toc4413953)

### Balanço dos objetivos cumpridos

Para a primeira fase do projeto da disciplina de Segurança e Confiabilidade, foi-nos pedido para criar e desenvolver uma aplicação do tipo cliente-servidor, em que fossem permitidas ligações de múltiplos clientes a um servidor, onde estes pudessem comunicar e fazer partilha de ficheiros e de mensagens entre si, e com o servidor.

Durante o tempo estipulado para elaboração da primeira fase, conseguimos desenvolver com sucesso todos os aspetos da aplicação, podendo então os clientes comunicar com o servidor, apresentando os resultados esperados e, em caso de erro, este mesmo é apresentado para que o cliente consiga aperceber-se do mesmo.

Por outro lado, foi-nos pedido para, utilizando ficheiros .policy, restringir o acesso a diversos recursos (escrita de ficheiros, leitura de sockets, entre outros). Para este ponto criámos dois ficheiros, um para o servidor e outro para o cliente. Tentámos correr a nossa aplicação utilizando estes ficheiros, mas esta apresentava diversos erros, como o apresentado em seguida:



Em suma, conseguimos criar a aplicação, mas não conseguimos restringir utilizando ficheiros .policy o acesso a recursos do sistema.

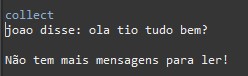
### Mensagens e formato das mesmas

As mensagens trocadas pelos clientes têm um formato bastante simples. Visto que, na interpretação do grupo estas seriam apenas mensagens de texto, estas passar-se-iam pelo terminal do seguinte modo: <msg Utilizador1 ola, eu chamo-me João>, sendo Utilizador1 o nome do utilizador ao qual nos queremos dirigir, e “ola, eu chamo-me João” um exemplo de mensagem enviada para o mesmo.

Tomámos liberdade de definir deste modo o formato das mensagens pois se fosse necessário enviar mensagens de outro tipo (p. ex. um ficheiro de imagem ou se­­­­­­­­­melhante), esta seria enviada utilizando a função download presente no programa.

O carácter destas mensagens pode ser informativo, tal como no caso em que o Utilizador1 deseja informar o Utilizador2 de que fez download de um ficheiro da área do servidor do Utilizador2. O primeiro pode assim enviar uma mensagem de texto a informar a ocorrência. Estas mensagens podem também ser apenas usadas para comunicação “tipo chat”, ou seja, podemos deixar uma mensagem e esperar por uma resposta passado um tempo.

De realçar que, quando se executa o comando “collect”, este vai apresentar o utilizador que enviou a mensagem, seguido da mensagem que este mesmo enviou, como segue no exemplo da imagem:



De modo a armazenar as mensagens, e quem as enviou, cada utilizador tem, na sua área do servidor, um ficheiro que contém as mensagens a serem apresentadas no terminal, para que, assim que o utilizador execute o comando “collect”, o ficheiro seja limpo e as mensagens não sejam novamente guardadas após aparecerem no terminal).

### Configuração da Sandbox

De forma a configurar a nossa sandbox criámos então dois ficheiros, que foram entregues juntamente com o zip, denominados client.policy e server.policy. Nestes consta um conjunto de instruções que restringem o acesso a recursos como os sockets, os ficheiros e as threads, tornando a nossa aplicação mais segura. Infelizmente não conseguimos utilizar estes ficheiros pois não conseguimos perceber o path correto a utilizar para poder restringir o acesso aos ficheiros MsgFileServer.java e MsgFile.java.

Não obstante, segue uma apresentação das permissões que escrevemos nestes ficheiros, e uma explicação das mesmas:

Do lado do servidor:

* java.net.SocketPermission "\*", "listen, accept";
  + De modo a poder receber pedidos do socket e a aceitar a ligação feita pelo cliente.
* java.io.FilePermission "file:\SegC-grupo17-proj1\\*", "read, write, delete, execute";
  + Para podermos criar as pastas “Servidor” e “Clientes”, seria então necessário dar ao servidor permissões de escrita, leitura, remoção e execução para poder satisfazer os pedidos dos clientes.
* java.io.FilePermission "file:\SegC-grupo17-proj1\servidor\\*", "read, write, delete, execute";
  + Visando poder criar os ficheiros do lado do servidor (p. ex. dar store a um ficheiro) seria então necessário dar ao servidor permissões de escrita, leitura, remoção e execução para poder satisfazer os pedidos dos clientes.
* java.io.FilePermission "file:\SegC-grupo17-proj1\clientes\\*", "read, write, delete, execute";
  + De forma a poder criar os ficheiros do lado do servidor (neste caso apenas seria para armazenar os ficheiros resultantes do comando “Download”) seria então necessário dar ao servidor permissões de escrita, leitura, remoção e execução para poder satisfazer os pedidos dos clientes.
* java.lang.RuntimePermission "modifyThread";
  + É necessário dar estas permissões para o servidor poder manipular threads.

Do lado do cliente:

* java.io.FilePermission "file:\SegC-grupo17-proj1\clientes\\*", "read, write";
  + Necessário que o cliente consiga escrever ficheiros do lado do mesmo para poder armazenar os ficheiros que deseja transferir do servidor.
* java.net.SocketPermission "\*", "connect, resolve";
  + Necessário para que o cliente se consiga conectar ao socket do servidor.

### Organização do Software

A organização do programa rege-se do seguinte modo:

* No ficheiro MsgFileServer foram criadas três classes, nomeadamente:
  + A classe Utilizador, que contém um construtor de um utilizador, sendo que este tem três parâmetros, o seu ID (ou seja, o seu nome de utilizador), uma lista com os ficheiros que o mesmo tem armazenados no servidor, e ainda uma lista com os utilizadores que este utilizador tem como trusted. Ainda nesta classe Utilizador existem vários métodos para poder operar com estes mesmos parâmetros.
  + A classe MsgFileServer é então responsável pela componente main do nosso programa, sendo esta muito simples, pois apenas é necessário que esta inicialize o nosso servidor, inicializando a ligação feita por um socket e criando por fim threads para cada cliente com a ajuda da nossa classe ServerThread.
  + A classe ServerThread cria uma thread para cada cliente, podendo assim fazer conexão com um cliente que se liga. É nesta classe que lemos a informação passada pela inputStream do socket para o servidor, e desta maneira executar as funções pedidas (store, list, users, etc…). Para tal, o mais lógico seria então criar um switch-case de modo a, dependendo daquilo que o cliente passasse para o servidor, este faria operações diferentes.

Para desempenhar as funções pedidas, seguiu-se a seguinte metodologia.

Para cada utilizador cria-se uma lista com os ficheiros que este tem no servidor e uma outra lista que contém os trusted users deste utilizador.

Store: Enviar o ficheiro a partir da área do cliente, para o servidor, escrevendo bytes que serão lidos pelo mesmo, escrevendo um novo ficheiro na área do cliente no servidor.

List: Passar pela rede o nome de ficheiros guardados no servidor, contidos na lista de ficheiros do utilizador corrente.

Users: Transmitir pela rede o contéudo de uma lista de users, que contém a informação do ficheiro UtilizadoresRegistados.txt.

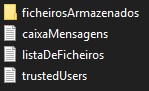
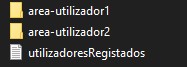
Trusted: Ler a lista dos trusted users do utilizador corrente, imprimindo o seu contéudo no terminal (Verificando se o utilizador existe, que ainda não é amigo e que não é dado o nome do próprio).

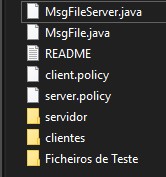
Untrusted: Apenas é necessário remover o nome do utilizador passado da lista de trusted users do cliente (Tem de ser amigo previamente).

Download: Fazer a verificação dos parâmetros (Utilizador tem de existir e ser trusted do utilizador cujo ID foi passado) e ler o ficheiro da área do utilizador no servidor e transmitir pela rede para o cliente que requisitou o download.

Msg: Fazer a verificação dos parâmetros (Utilizador tem de existir e ser trusted user) e escrever o contéudo da mensagem para o ficheiro caixaMensagens.txt do destinatário da mensagem.

Collect: Passar pela rede o contéudo do ficheiro caixaMensagens.txt para o cliente, limpando o mesmo.

* No ficheiro MsgFile foi criada apenas uma classe, de nome próprio:
  + Nesta temos a função main, onde tudo acontece. Após uma primeira fase de autenticação o cliente vai enviar comandos para o servidor, e recebe posteriormente a sua resposta, tratando de notificar o utilizador das mudanças que possam ter ocorrido ou dos erros que eventualmente possam ter sido despoletados pelo programa.
* Os diretórios gerados pelo nosso programa são os seguintes:
  + Pasta servidor:
  + Dentro desta pasta serão criadas tantas pastas quantos utilizadores estiverem registados no sistema, denominadas “area-XXXX” onde XXXX corresponde ao nome do utilizador, sendo esta gerada aquando do registo do utilizador.
  + Dentro de cada uma das pastas da área de utilizador serão gerados três ficheiros, caixaMensagens.txt, listaDeFicheiros.txt e trustedUsers.txt, onde são guardadas as mensagens recebidas, onde estão contidos os nomes dos ficheiros armazenados no servidor (por este utilizador) e onde estão contidos os nomes dos trusted users deste utilizador, respetivamente. É ainda criada uma pasta ficheirosArmazenados, onde ficarão guardados os ficheiros stored pelo cliente através do comando “store <files>”.
  + Ficheiro utilizadoresRegistados.txt que contém uma listagem dos utilizadores registados bem como as suas palavras-passe.
* Pasta clientes:
  + Nesta pasta serão colocados os ficheiros que se querem armazenar no servidor manualmente, de modo a simular a área local do cliente.
  + Será apenas criada a pasta ficheirosTransferidos onde serão guardados os ficheiros transferidos da área do servidor de outros utilizadores.



* Pasta Root do programa:
  + Criação das pastas “servidor” e “Clientes” aquando do primeiro arranque do servidor

De modo a informar o cliente dos erros que possam ocorrer (p. ex. enviar uma mensagem a um utilizador que não existe) o servidor trata de enviar essa mensagem pelo socket, visando abranger e tratar todos os erros que possam ocorrer ao inserir determinado comando. Esta mensagem é lida pelo cliente através do socket e impressa no terminal.

Para garantir o bom funcionamento da nossa aplicação, tomámos a liberdade de, primeiramente, tornar o acesso aos ficheiros algo sincronizado, prevenindo que a informação fosse corrompida ao ler dos ficheiros (p. ex. lista de utilizadores registados no sistema). Para tal criámos e utilizámos semáforos, para dar permissão de escrita ou leitura de um ficheiro apenas e só apenas a um utilizador de cada vez.

### Requisitos de Segurança

Para proteger a informação dos utilizadores, a palavra-passe dos mesmos não está guardada dentro da classe Utilizador ou em qualquer uma das outras do ficheiro MsgFileServer, não permitindo assim que, um cliente ao estar ligado ao servidor, tenha acesso à palavra passe deste utilizador se este assim o pretender. Sempre que um utilizador se autentique no servidor, este não tem maneira de conseguir, através do mesmo, informação sobre a palavra-passe. Assim, o cliente é protegido de um possível ataque à sua conta, pois assim um indivíduo “mal-intencionado” não poderia obter a palavra-passe do cliente através do servidor. Para garantir isto apenas foi necessário não incluir a informação da palavra-passe no construtor ou em qualquer outro método da classe.

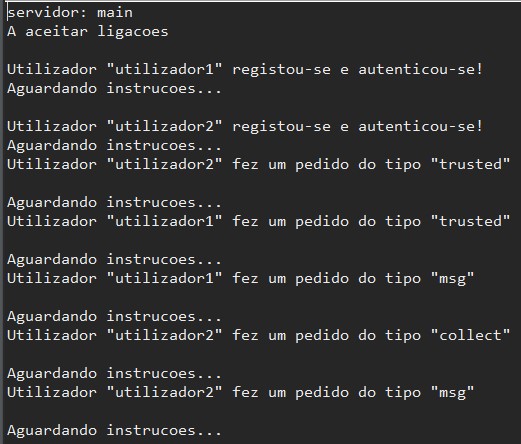
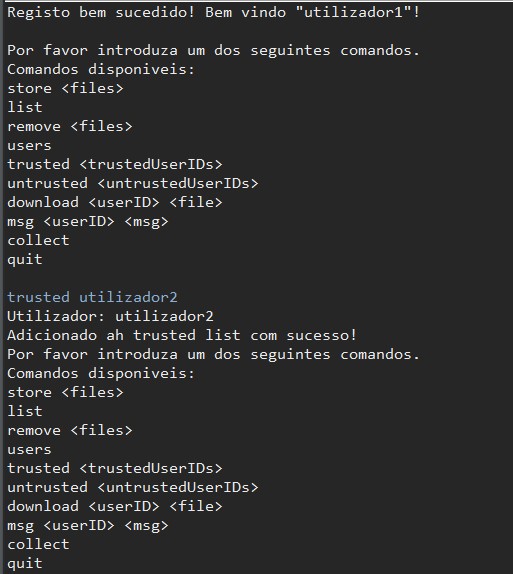
De maneira a tornar a troca de mensagens mais seguras, estas poderiam ser encriptadas, para que, se alguém estivesse “à escuta” do canal de troca de mensagens, não pudesse saber o conteúdo das mesmas facilmente.

Para aumentar a confidencialidade do nosso sistema e dos seus utilizadores, poder-se-ia encriptar os ficheiros no servidor, e as respetivas áreas remotas, para que, se alguma entidade acedesse à base de dados do servidor, não pudesse obter informação dos ficheiros armazenados.

A troca de ficheiros pela rede (caso do store ou download), deveria ser segmentada, e não enviada como um todo, para que, se alguém intercetasse pacotes trocados pelo cliente e o servidor, este alguém não poderia ter acesso aos ficheiros trocados, ficando apenas com alguns bytes deste ficheiro. Nesta fase ainda não foi implementado esta maneira de segmentação dos ficheiros, pois pensamos que no próximo seria mais pertinente colocar isto em ação.

Considera-se um ponto de falha do nosso sistema o ficheiro UtilizadoresRegistados, presente no diretório de raiz do nosso servidor. Este contém a informação de todos os utilizadores e das suas palavras passes, tornando a nossa aplicação menos segura. Isto apenas acontece pois seguimos a implementação pedida pelos professores.

#### Exemplo de funcionamento da aplicação



*Figura 3 - Terminal do Cliente Figura 4 - Terminal do Servidor*