Relatório do Projeto da Fase 2 de Segurança e Confiabilidade 20/21

Grupo35

Martim Silva 51304

Francisco Freire 52177

David Rodrigues 53307

Instruções…

Para compilar servidor

javac -d bin src/server/SeiTchizServer.java src/communication/Com.java src/security/Security.java

Para compilar cliente

javac -d bin src/client/SeiTchiz.java src/client/ClientStub.java src/communication/Com.java src/security/Security.java

Para correr servidor com policies

java -cp bin -Djava.security.manager -Djava.security.policy==server.policy server.SeiTchizServer 45678 serverKeyStore passserver

Para correr clientes com policies

java -cp bin -Djava.security.manager -Djava.security.policy==client.policy client.SeiTchiz localhost ts_client 1KS passclient1 client1

java -cp bin -Djava.security.manager -Djava.security.policy==client.policy client.SeiTchiz localhost ts_client 2KS passclient2 client2

java -cp bin -Djava.security.manager -Djava.security.policy==client.policy client.SeiTchiz localhost ts_client 3KS passclient3 client3

java -cp bin -Djava.security.manager -Djava.security.policy==client.policy client.SeiTchiz localhost ts_client 4KS passclient4 client4

Para correr servidor com policies por jar

java -Djava.security.manager -Djava.security.policy==server.policy -jar SeiTchizServer.jar 45678 serverKeyStore passserver

Para correr cliente com policies por jar

java -Djava.security.manager -Djava.security.policy==client.policy -jar SeiTchiz.jar localhost ts_client 1KS passclient1 client1

java -Djava.security.manager -Djava.security.policy==client.policy -jar SeiTchiz.jar localhost ts_client 2KS passclient2 client2

java -Djava.security.manager -Djava.security.policy==client.policy -jar SeiTchiz.jar localhost ts_client 3KS passclient3 client3

java -Djava.security.manager -Djava.security.policy==client.policy -jar SeiTchiz.jar localhost ts_client 4KS passclient4 client4

Limitações do trabalho da fase 1 + 2

O único argumento que deve ser passado obrigatoriamente para o servidor correr e 45678.

O serverAddress passado pelo cliente como argumento pode ser apenas <endereço IP> (por ex:localhost) ou <endereço IP>:<porto 45678> (por ex: localhost:45678).

Username e passwords passados como argumento em SeiTchiz não devem conter espaços nem dois pontos(:) nem hífens(-) nem forward slashes(/).

Nomes de grupos não devem conter espaços nem dois pontos(:) nem hífens(-) nem forward slashes(/).

Mensagens não devem conter dois pontos(:) nem hífens(-).

UserIDs inseridos nos comandos que recebem userIDs não devem conter espaços nem dois pontos(:) nem hífens(-) nem forward slashes(/).

Um user que já esteja logged on nao deve fazer login enquanto a sessão inicial não tenha sido terminada.

groupIDs inseridos nos comandos que recebem userIDs não devem conter espaços nem dois pontos(:) nem hífens(-) nem forward slashes(/).

As fotos de stock que se podem partilhar são apenas as que se encontram no ficheiro Fotos na root do projeto, ou seja o argumento <photo> de post deve ser foto<1 ou 2 ou 3 ou 4>.jpg

A pasta bin não deve ser apagada apenas os seus conteúdos podem ser apagados.

A pasta files pode ser apagada para dar um "restart" do servidor e todos os seus conteúdos

Como não foi dito no enunciado não foi implementado o impedimento de um utilizador dar múltiplos likes a mesma fotografia nem um utilizador poder dar like a sua própria fotografia.

Para interromper o funcionamento de um cliente usar a opção s ou stop.

Existem bugs que não conseguimos tratar relacionados com os comandos collect e history mais especificamente, caso se crie um grupo, se envie uma mensagem e só depois se adicionar um utilizador e se efetue um desses comandos há casos em que isto provoca uma exceção NullPointerException.

Comandos de chaves assimétricas usados

keytool -genkeypair -alias serverKeyStore -keyalg RSA -keysize 2048 -storetype JKS -keystore keystores/serverKeyStore

Password usada: passserver

keytool -genkeypair -alias 1KS -keyalg RSA -keysize 2048 -storetype JKS -keystore keystores/1KS

Password usada: passclient1

keytool -genkeypair -alias 2KS -keyalg RSA -keysize 2048 -storetype JKS -keystore keystores/2KS

Password usada: passclient2

keytool -genkeypair -alias 3KS -keyalg RSA -keysize 2048 -storetype JKS -keystore keystores/3KS

Password usada: passclient3

keytool -genkeypair -alias 4KS -keyalg RSA -keysize 2048 -storetype JKS -keystore keystores/4KS

Password usada: passclient4

Comandos de chaves simétricas usados

keytool -genseckey -alias serverKey -storetype JCEKS -keystore ServerKeyStore

Comandos de verificar chaves

keytool -list -keystore keystores.serverKeyStore

keytool -list -keystore keystores.1KS

keytool -list -keystore keystores.2KS

keytool -list -keystore keystores.3KS

keytool -list -keystore keystores.4KS

Decisões de desenho das soluções implementadas, nomeadamente em termos de arquitetura de software, de segurança, de desempenho e de funcionalidade

Dividimos os ficheiros em 7 partes:

- "groups" - neste diretório encontramos toda a informação sobre todos os grupos criados como por exemplo todas as mensagens que se partilham nele e a informação sobre as mesmas (utilizadores que as viram / não viram), o conteúdo das mesmas, o emissor da mensagem, a informação sobre como aceder ao conteúdo da mensagem (chaves e respetivo identificador).
- "serverStuff" – neste diretório guardamos apenas 2 ficheiros, um dos quais contem os ids dos utilizadores que estão autenticados e o caminho para os seus certificados, esta maneira substitui a versão menos segura de proteger a informação dos clientes que era guardar o id dos clientes e a sua password que era apenas uma "string" definida pelos mesmos e o outro ficheiro guarda um contador global para atribuição de nomes únicos às fotografias postadas na aplicação.
- "userStuff" – neste diretório guardamos as pastas com a "metadata" do utilizador, ou seja, a que grupos pertence e a que grupos preside como dono também guarda informação sobre quem cada utilizador segue e os seguidores desse mesmo utilizador e por fim tem um sub-diretório que armazena as fotografias postadas pelo mesmo na aplicação.
- "Fotos" – contém as "stock fotos" que são postadas pelos clientes
- "PubKeys" – contém os certificados de clientes autenticados
- "truststore" – contém a chave pública do servidor que todos os clientes irão usar para se ligarem ao servidor
- "keystores" – contém as chaves simétricas dos clientes todos e do servidor sendo que cada uma é apenas acessível pela entidade que a "criou" (ex: se o cliente1 gerasse a keystore 1KS só esse cliente é que sabe a chave privada lá contida).

Sobre a ligação estabelecida são usadas sockets seguras TLS com autenticação unilateral que são preferenciais do ponto de vista de autenticidade do servidor e confidencialidade da comunicação entre cliente e servidor às sockets “normais”.

A autenticação do cliente é feita através de criptografia assimétrica em vez de passwords. Cada cliente possui uma keystore sua a qual apenas ele tem acesso e ao estabelecer a ligação com o servidor este manda ao cliente um token de uso único e se o servidor não reconhecer o utilizador então pede dele o mesmo token tal como a assinatura do mesmo através da sua chave pública contida na keystore dele e o certificado que contém essa chave pública. O servidor verifica a autenticidade do token e confere se a assinatura dada foi de facto obtida a partir do certificado recebido garantindo que o utilizador é quem diz ser. Se ambas estas condições se verificarem o utilizador passa a ser reconhecido pelo servidor normalmente ao colocar num diretório PubKeys o certificado que recebeu. No caso de um utilizador já reconhecido tentar fazer log-in o processo é parecido. O utilizador apenas tem de enviar o token de uso único e a assinatura do mesmo pois o servidor já terá acesso ao certificado que o cliente usou para gerar a assinatura e verificar a autenticidade deste cliente a partir daí.

Todos os ficheiros em si com a exceção dos ficheiros que têm o conteúdo de mensagens enviadas por utilizadores em grupos e os ficheiros de fotos, “likes” e contadores de fotos são cifrados pelo servidor com a sua chave simétrica e consequentemente esta chave é cifrada com a chave assimétrica do servidor garantindo que apenas o mesmo consegue decifrar o conteúdo de todos eles.

As fotos quando são postadas por clientes geram no folder de “photos” do cliente que as postou 3 ficheiros, a própria foto não alterada em nenhuma maneira, o ficheiro de “likes” que contém o número de “likes” dessa foto especifica e um ficheiro que contém uma síntese segura (MAC) da foto gerada pelo cliente que garante a integridade da mesma. Se forem feitas alterações á foto, na chamada de “wall” por outro cliente que siga esse cliente que postou a foto este não a vai ver pois foi “corrompida” e fotos com a sua integridade danificada nunca apareceram no “mural” de qualquer cliente.

Os processos de adicionar membros e remover membros de um grupo cabe apenas ao cliente que criou o grupo e para aumentar o grau de segurança na troca de mensagens o que foi implementado foi o uso de chaves de grupo simétricas para cifrar e decifrar mensagens trocadas no grupo, estas mensagens serão cifradas pelos próprios clientes membros dos grupos por isso nem o servidor pode ser a causa de uma “breach” da informação no grupo pois este desconhece as chaves necessárias para decifrar quaisquer mensagens enviadas no mesmo.

A maneira como isto é feito é quando o dono cria um grupo também gera uma chave simétrica que vai cifrar com a sua pública e acoplar a este conjunto de bytes um identificador, mais especificamente um número e colocar isto tudo dentro do grupo. Sempre que um cliente é adicionado ou removido, o dono gera uma nova chave que vai ser também cifrada, mas desta vez cada cliente pertencente no grupo no momento da adição/remoção de um membro também vai cifrar a chave gerada pelo dono e assim se tivermos um grupo de 5 membros teremos a mesma chave simétrica gerada no momento cifrada pela chave pública de cada um dos membros, estas chaves serão ser todas atribuídas um mesmo identificador único e toda esta informação será guardada dentro do diretório do grupo, este trabalho é feito pelo servidor. Finalmente no momento de um utilizador enviar mensagem para o grupo esta cifra ele mesmo a mensagem com a chave simétrica mais recente guardada e de seguida dar “wrap” a essa chave com a pública do cliente. Quando os outros membros do grupo a forem ler, ou por “history” ou por “collect” eles vão encontrar pelo identificador das chaves qual a chave cifrada deles que corresponde á mesma usada para fazer a mensagem, obtém essa chave e decifram-na com a sua privada e ficam com a chave simétrica que foi usada para cifrar a mensagem original e usando essa chave têm então acesso ao seu conteúdo. Esta abordagem também garante que utilizadores que façam parte do grupo depois de uma mensagem ser enviada não a consigam ver pois não faziam parte do grupo quando ela foi criada.