

1 Objectivos

Esta fase do trabalho estende a anterior, possibilitando aos alunos experimentarem diversos mecanismos de segurança, tais como: cifras, assinaturas, comunicação com um protocolo seguro (TLS – Transport Layer Security) e gestão básica de certificados. A envolvente do trabalho continua a ser a mesma, ou seja, a concretização de um sistema **simplicado** e seguro de *envio de mensagens com partilha de ficheiros entre utilizadores*, **MsgFile**. O trabalho será realizado utilizando a linguagem de programação Java e a API de segurança do Java, e ferramentas complementares. Neste trabalho, tal como o anterior, o cliente e o servidor devem ser executados dentro da *sandbox*.

2 Modelo de adversário

Iremos assumir no trabalho que existe um adversário que pretende comprometer o correto funcionamento do sistema. O adversário terá um conjunto de capacidades que poderão ser empregues na realização das suas ações maliciosas. Torna-se assim necessário dotar o sistema dos mecanismos de proteção que lhe possibilitem manter um funcionamento correto ainda que se encontre sob ataque.

Vamos assumir que o adversário tem as seguintes capacidades:

- Acesso à rede: tendo o adversário acesso à rede, poderá escutar os pacotes trocados entre o cliente e o servidor. Potencialmente, também poderá tentar corromper, alterar, introduzir, e reproduzir mensagens de forma a enganar quer o cliente quer o servidor.
- Controlar um ou mais clientes: o adversário controla uma (ou mais) conta(s) de um cliente do sistema. Através desta conta, ele poderia tentar aceder a ficheiros/mensagens para os quais não tem permissões ou corromper ficheiros com informação de outros utilizadores.
- Acesso à máquina onde corre o servidor em modo de leitura: o adversário tem acesso em modo de leitura aos ficheiros armazenados no servidor. Com esse acesso, ele pode potencialmente observar informação que eventualmente seria secreta.

Em seguida indicam-se e discutem-se as proteções que devem ser adicionadas ao sistema.

3 Proteções a adicionar ao sistema

Nesta fase, os alunos devem usar a mesma arquitectura da 1ª fase, e as funcionalidades a oferecer mantêm-se aproximadamente iguais. No entanto, o sistema será estendido de modo a ser garantida a sua segurança.

As alterações a introduzir são as seguintes:

1. Para gerir os utilizadores do sistema, os alunos têm de concretizar um programa Java, designado por *userManager*, que permite adicionar e remover utilizadores, bem como alterar as passwords. Este programa deve **ser o único** que pode alterar a configuração dos utilizadores (ou seja, o *MsgFile/MsgFileServer* deixam de ter essa capacidade). Para facilitar a implementação, o programa *userManager* só deve ser executado quando o servidor estiver desligado.

O *userManager* deve manter informação sobre os utilizadores registados num ficheiro. Para cada utilizador, uma linha é criada no ficheiro contendo o *username* e a *password* protegida separados pelo carácter ":". O *userManager* deve proteger a **privacidade das passwords** dos utilizadores. Com este objetivo, em vez de armazenar as *passwords* dos utilizadores em claro no ficheiro, deve guardar o *salted password hash* delas, que é criado pelo hash da concatenação de um número aleatório (salt) com a password do utilizador (i.e., $H(\text{salt} || \text{password})$). Desta forma, a informação do utilizador guardada no ficheiro das passwords será *username:salt:salted_password_hash*

O *userManager* deve proteger a **integridade do ficheiro das passwords**. Para tal, o ficheiro deve ser protegido com um MAC. O cálculo deste MAC utiliza uma chave simétrica calculada a partir de uma *password* que é pedida ao administrador quando este **inicia a execução** do programa. No início da sua execução, o *userManager* deve usar o MAC para verificar a integridade do ficheiro. Se o MAC estiver errado, então deve imprimir um aviso e terminar imediatamente a execução. Se não há MAC a proteger o ficheiro, o *userManager* deve imprimir um aviso, calcular o MAC e adicioná-lo ao sistema. O MAC deve ser verificado em todos os restantes acessos ao ficheiro e atualizado caso o ficheiro seja alterado. O MAC pode ser guardado num ficheiro separado, utilizado apenas para este efeito.

2. O servidor deve **autenticar os utilizadores**. O utilizador durante a autenticação fornece o *username* e a *password*. Esta informação deve ser transmitida para o servidor protegida de ataques na rede (ver abaixo). O servidor deve validá-la recorrendo ao ficheiro do ponto 1, e apenas se esta estiver correta deve ser dado acesso ao utilizador. O servidor também precisa de validar o MAC, necessitando para isso de receber durante a sua inicialização a mesma password do administrador que o *userManager*.
3. Dado que na 1ª fase do trabalho, o cliente está a comunicar com o servidor em claro, temos de garantir a **autenticidade do servidor** (um atacante não deve ser capaz de fingir ser o servidor e assim obter a password de um utilizador) e a **confidencialidade** da comunicação entre cliente e servidor (um atacante não deve ser capaz de escutar a comunicação). Para este efeito, devem-se usar **canais seguros** (protocolo TLS/SSL) e a verificação da identidade do servidor à

base de criptografia assimétrica (fornecida pelo protocolo TLS).

- Ligações TLS: Deve-se substituir a ligação TCP por uma ligação TLS/SSL. O protocolo TLS vai verificar a autenticidade do servidor e garantir a integridade e confidencialidade de toda a comunicação.
 - A utilização do protocolo TLS exige configurar as chaves tanto no cliente (*truststore* com o certificado do servidor) como no servidor (*keystore* com a sua chave privada).
4. Os ficheiros e as mensagens devem ser protegidos de eventuais ataques que possam ocorrer na máquina servidora, nomeadamente que tenham em vista **observar e alterar o seu conteúdo**. Para isso, sugerimos a utilização de criptografia híbrida. A ideia seria aplicar genericamente o seguinte algoritmo aos ficheiros (incluindo ficheiros de mensagens, caso aplicável, dependendo da maneira como concretizaram o projeto):
- I. O servidor gera uma chave simétrica K AES aleatoriamente
 - II. Cifra o conteúdo do ficheiro F recebido do cliente com K usando AES
 - III. O servidor cifra K usando a sua chave pública, e armazena-a num ficheiro separado com extensão *.key* (p. ex., o ficheiro *fich.txt* teria a chave armazenada em *fich.txt.key*)
 - IV. Os ficheiros com o conteúdo cifrado e com a chave são armazenados no servidor

Quando um utilizador lê o ficheiro do servidor, deve ser executado o processo inverso ao definido anteriormente:

- I. O servidor deve obter uma cópia do ficheiro cifrado e do ficheiro *.key* correspondente
- II. O servidor deve ler a chave cifrada, e decifrá-la com a sua chave privada, obtendo K
- III. K deve ser usada para decifrar o conteúdo do ficheiro
- IV. O ficheiro decifrado deve ser enviado para o cliente.

Claro está que o servidor só disponibiliza os ficheiros aos utilizadores que têm acesso ao mesmo (p. ex., o dono dos ficheiros mais os seus amigos).

5. Os ficheiros de controlo que eventualmente sejam mantidos no servidor (p. ex., onde é guardada a informação sobre os amigos) também devem ser protegidos para assegurar que a informação **não é observada**, e para garantir que **eventuais alterações maliciosas são detectadas**. Estes ficheiros para além de serem cifrados de uma maneira semelhante ao descrito acima, devem ainda ser assinados.
- I. O conteúdo do ficheiro é assinado usando a chave privada do servidor. A assinatura é armazenada num ficheiro separado com extensão *.sig*
 - II. O conteúdo do ficheiro é em seguida cifrado como descrito acima --- com uma chave aleatória K , e a chave é armazenada num ficheiro com extensão *.key* (após ter sido cifrada com a chave pública do servidor)

Quando o servidor acede ao ficheiro, deve realizar os seguintes passos:

- I. A chave K é obtida e depois usada para decifrar o conteúdo do ficheiro
 - II. A integridade e autenticidade do conteúdo do ficheiro são verificadas com a validação da assinatura digital
6. Alguém que se torne amigo de um utilizador deve passar a ter acesso a todos os ficheiros desse utilizador, exceto às suas mensagens.
 7. Quando se retira alguém da lista de amigos de um utilizador, esta deixa de poder aceder à informação do utilizador, i.e., aos ficheiros.

NOTA: Toda criptografia assimétrica no projeto deve usar RSA com chaves de 2048 bits. A criptografia simétrica deve ser efetuada com AES e chaves de 128 bits. O algoritmo de síntese deve ser o SHA-256.

4 Relatório e discussão

No relatório devem ser apresentados e discutidos os seguintes aspetos:

- Indicação dos objetivos **concretizados com êxito e os que não foram**.
- Os problemas encontrados.
- A segurança da aplicação criada, identificando possíveis **fraquezas e melhorias** a incluir em versões futuras.

O relatório deve ter no máximo 5 páginas (sem contar com o código) e **é necessário incluir o código fonte**.

5 Entrega

- **Código.**

Dia **26 de abril**, até às 23:55 horas. O código do trabalho deve ser entregue da seguinte forma:

- Criar um **ficheiro zip** com o nome **SegC-grupoXX-proj2.zip**, onde XX é o número do grupo, contendo o código do trabalho e um readme (txt) sobre como executar o projeto.
- Submeter o ficheiro zip através da atividade disponibilizada para esse efeito na página da disciplina no Moodle. Apenas um dos elementos do grupo deve submeter. Se existirem várias submissões do mesmo grupo, será considerada a mais recente.

- **Relatório.**

Dia **29 de abril**, até às 18:00 horas. O relatório deve ser entregue da seguinte forma:

- Criar o relatório **em formato pdf**, com o nome **SegC-grupoXX-proj2-relat.pdf**.
- Submeter o ficheiro pdf através da atividade disponibilizada para esse efeito na página da disciplina no Moodle. Apenas um dos elementos

do grupo deve submeter. Se existirem várias submissões do mesmo grupo, será considerada a mais recente.

Não serão aceites trabalhos por email nem por qualquer outro meio não definido nesta secção. Se não se verificar algum destes requisitos o trabalho é considerado não entregue.