Projeto 3: Autenticação

Segurança Informática e nas Organizações

DETI – Universidade de Aveiro

31/01/2019

Gonçalo Almeida nº 79994

Hugo Oliveira nº 76322

Introdução

Este trabalho tem como objetivo avaliar competências adquiridas nas aulas sobre autenticação, considerando o seu planeamento, desenho, implementação e validação, tendo como base o Projeto anterior sobre comunicações seguras.

Planeamento

O workflow das mensagens:

1. Negociação dos algoritmos a utilizar (cifras simétricas, modos de cifra e funções de síntese)
2. Processo do algoritmo de Diffie-Helman
3. Processo de autenticação do servidor, requerido pelo cliente
4. Processo de autenticação do cliente, também requerido pelo cliente ao servidor, com 2 possibilidades distintas:
   4.1. Autenticação por mecanismo de desafio resposta
   4.2. Autenticação através do cartão de cidadão
5. Pedido de transferência do ficheiro por parte do cliente, com controlo de acessos.
6. Início da troca de informação segura através de uma mensagem OPEN cifrada.
7. Envio de pedaços (*chunks*) de um ficheiro através de várias mensagens DATA cifradas.
8. Conclusão da sessão após a transferência completa do ficheiro com uma mensagem CLOSE cifrada.

Todas as mensagens trocadas (até à conclusão do **ponto 2**) são cifradas no campo **content** de uma mensagem do tipo **SECURE_MESSAGE**, sucedidas de uma mensagem do tipo **MAC**, tendo por finalidade garantir a integridade das mesmas.

Implementação

- Implementação do protocolo para autenticação do servidor através de certificados X.509

A autenticação do *servidor* inicia-se com a geração de um NONCE por parte do *cliente*. Depois disso, o *cliente* envia ao *servidor* o NONCE através de uma mensagem do tipo **SERVER_AUTH_REQUEST**

O *servidor*, ao receber e processar esta mensagem, carrega o seu certificado, o certificado da sua raiz e a chave privada associada ao seu certificado, usando-a para assinar o NONCE enviado pelo *cliente*. Depois, o *servidor* envia ao cliente o seu certificado, o certificado da sua raiz e a assinatura através de uma mensagem do tipo **SERVER_AUTH_RESPONSE**:

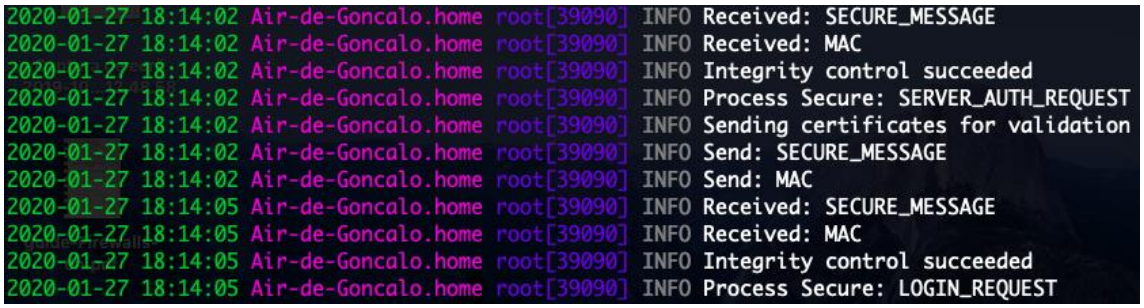
Após receber a mensagem com a assinatura e com os certificados, o *cliente* valida a **assinatura criada pelo *servidor*** com a chave pública do segundo. Depois, valida se o **common_name** do certificado do servidor é igual ao nome do servidor que ele supõe estar a aceder.

Por último, o *cliente* cria a **cadeia de certificados do *servidor*** e executa todas as operações necessárias para validar cada certificado da cadeia avaliando a data de expiração, o purpose (no primeiro certificado, o **certificado do servidor**, é necessário garantir que inclui a KeyUsage **SERVER_AUTH** enquanto nos seguintes certificados é preciso garantir que incluem a KeyUsage **key_cert_sign**), a assinatura do certificado, o common name e o estado de revogação (**OCSP, CRL** e **DeltaCRL**).

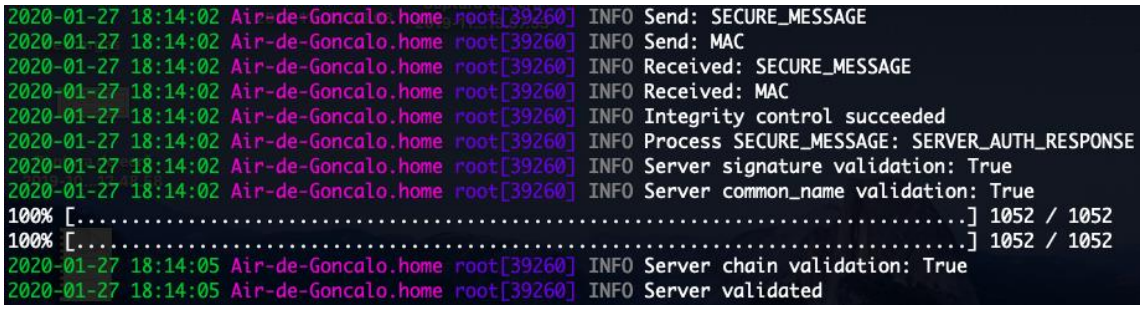
Após a validação (ou não) de todas estas condições, o *cliente* irá também validar (ou não) o *servidor*, respetivamente, e transitar para o ponto seguinte do processo, no qual se irá tentar autenticar.

Os certifcados usados para representar o servidor foram criados através do programa **XCA** e exportados no formato **PEM** para poderem ser carregados pelo *servidor* e enviados para o *cliente*.

Servidor:



Cliente



- Implementação do protocolo para autenticação de utentes através da apresentação de senhas

Após o processo de autenticação do *servidor*, o *cliente* precisa se autenticar antes de fazer qualquer pedido para a transferência de um ficheiro. Com isto, decidimos implementar duas formas distintas de autenticação do

cliente para com o servidor: através de um mecanismo de **desafio-resposta** ou através do **cartão de cidadão.**

Para alterar o modo de autenticação do cliente, será necessário mudar o valor de variável **self.validation_type**, no ficheiro client.py **(linha 66)** para **"CHALLENGE"** ou **"CITIZEN_CARD".**

Mecanismo de **desafio-resposta**:

Antes de qualquer pedido, o *cliente* gera um par de chaves assimétricas.

Em seguida, o *cliente* envia uma mensagem ao *servidor* do tipo **LOGIN_REQUEST**, juntamente com um **NONCE** e a **chave-pública** gerada no ponto anterior (ambos no formato de *base64*)

O *servidor*, depois de receber o pedido de autenticação por desafio-resposta, guarda o **NONCE** e a **chave-pública** do cliente e envia-lhe um desafio (um novo **NONCE**) por via de uma mensagem do tipo **CHALLENGE_REQUEST.**

O *cliente*, ao receber o desafio do *servidor*, primeiramente introduz o seu **username** e a sua **password** e depois concatena o seu **NONCE** gerado, com a sua **password** e com o **NONCE** recebido do servidor, por esta ordem. Com a obtenção desta **resposta**, o *cliente* assina-a com a sua **chave-privada** gerada no início deste processo.

Por fim, envia ao servidor o seu username e a mensagem assinada através de uma mensagem do tipo **CHALLENGE_RESPONSE.**

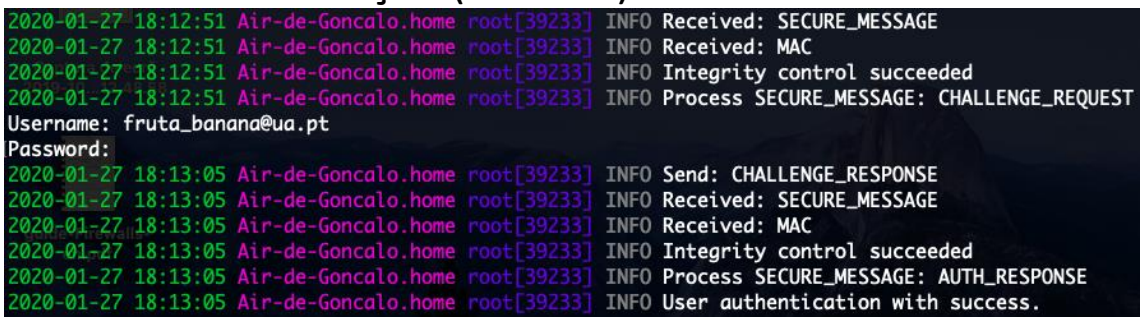
O *servidor*, ao receber a resposta do *cliente*, começa por verificar se o username fornecido está registado em

persistência e se tem permissão de autenticação (explicado com maior detalhe mais à frente).

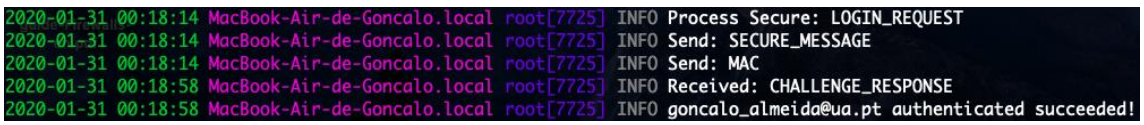
Caso o nome de utilizador introduzido esteja registado no sistema, o *servidor* carrega a sua **password** e concatena o **NONCE** recebido do cliente na fase inicial com a password carregada e com o seu **NONCE**, novamente por esta ordem. Depois, o *servidor* valida a mensagem assinada recebida através da chava-pública do cliente. Caso a mensagem assinada seja corretamente validada, o *servidor* garante a autenticidade do username em questão, enviando-lhe uma mensagem do tipo **AUTH_RESPONSE** com um estado de **sucesso**.

Caso a mensagem assinada não seja validada, o *servidor* não consegue autenticar o *cliente*, pelo que se sucede uma mensagem do tipo **AUTH_RESPONSE** mas com um estado de **falha**.

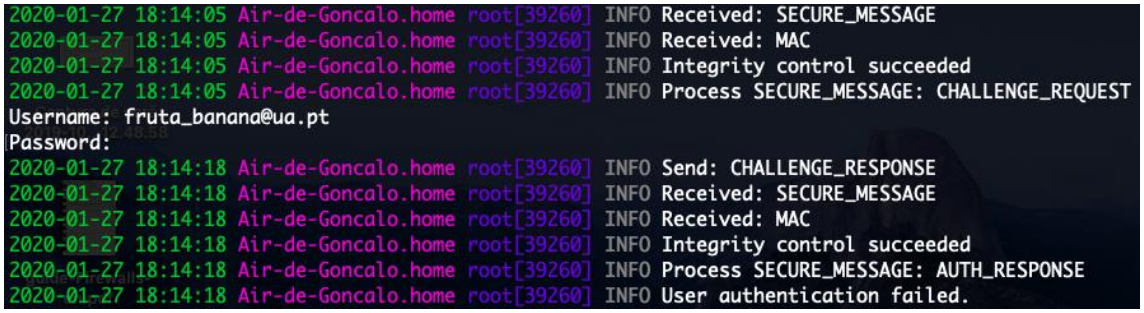
Cliente – autenticação (sucesso)



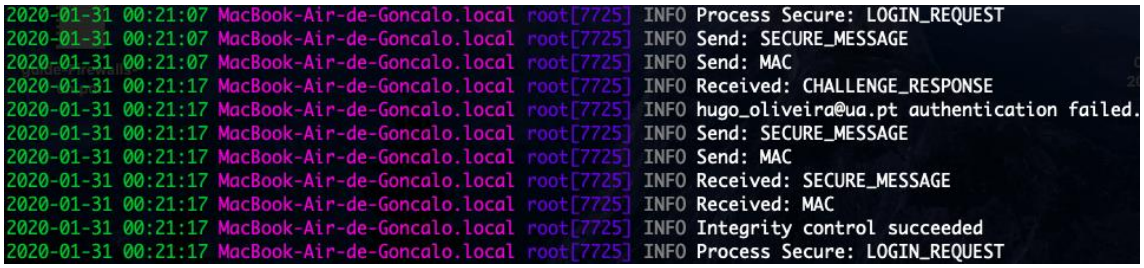
Servidor – autenticação (sucesso)



Cliente – autenticação (falha)



Servidor – autenticação (falha)



- Implementação do protocolo para autentição de utentes através do cartão de cidadão

Em alternativa à autenticação do *cliente* através da apresentação de senhas, implementámos a autenticação através do cartão de cidadão, onde, tal como no protocolo de senhas, o cliente começa por enviar uma mensagem ao servidor (após a sua validação) com um **NONCE** através de uma mensagem do tipo **CARD_LOGIN_REQUEST**.

O *servidor*, ao receber e processar esta mensagem, guarda o **NONCE** enviado pelo cliente e gera outro **NONCE**, que irá ser enviado para o cliente através de uma mensagem do tipo **CARD_LOGIN_RESPONSE**.

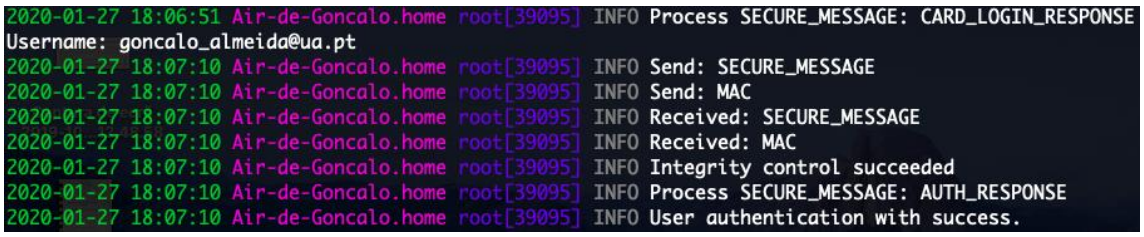
Após receber a mensagem com o **NONCE** do servidor, o *cliente* irá inserir o seu username para garantir a validação do acesso e irá assinar com o seu CC a concatenação do **NONCE** gerado por ele com o **NONCE** enviado pelo servidor. Depois disso, envia o certificado de autenticação do CC utilizado para gerar a assinatura, a assinatura e o username através de uma mensagem do tipo **AUTH_CERTIFICATE**.

O *servidor*, ao receber esta mensagem, verifica se a existência do username lista de utilizadores (para posteriormente validar no controlo de acesso), e de seguida valida a assinatura enviada pelo cliente, com a chave pública presente no certificado do mesmo.

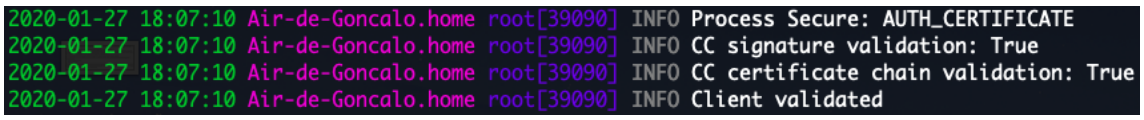
Por último, à semelhança da validação da cadeia de certificação do servidor, este irá construir a **cadeia de certificados** associada ao **CC do cliente**, validando cada certificado em relação à data de expiração, purpose, assinatura do certificado, common name e estado de revogação.

Se todas estas condições forem efetivamente validadas, o *servidor* irá autenticar o cliente e transitar para o próximo passo, no qual o cliente poderá iniciar o envio do ficheiro.

Cliente – CC auth



Servidor – CC auth



- Implementação do mecanismo para controlo de acesso

Foram gerados 3 **usernames**, cada um com uma **password** de 16 caracteres aleatórios. O alfabeto de geração destas passwords foi composto pelos seguintes 90 carateres:

[a-zA-Z] e **1234567890,;._-}{"\'|+*()[]&%$#!@€£<>/**

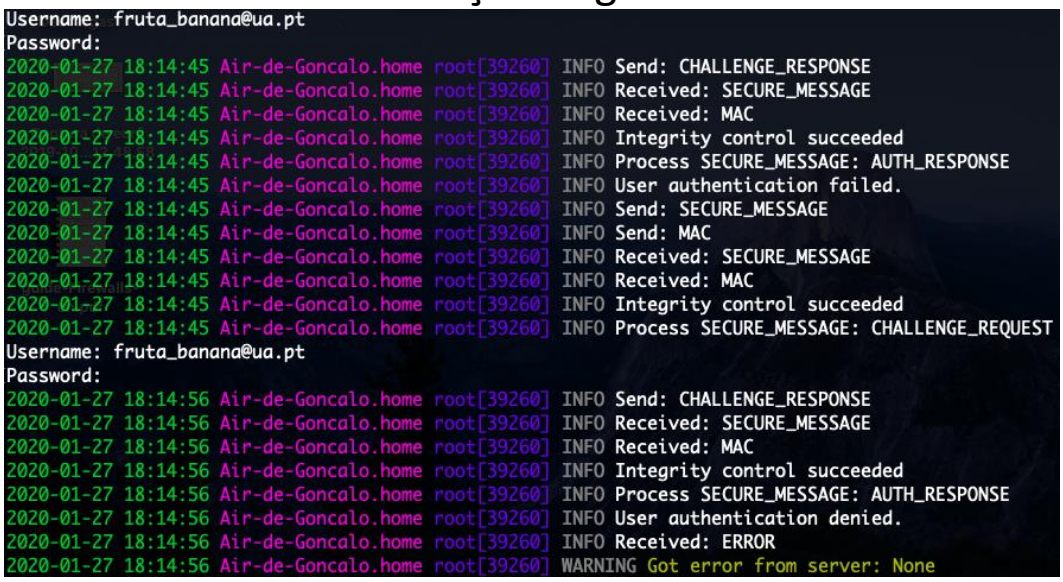
permitindo gerar **90^16 passwords distintas**, com vista a impossibilitar o sucesso de ataques por dicionários. Relativamente às permissões de acesso de cada utilizador, foram definidos 2 tipos de permissões: **AUTENTICAÇÃO** (flag 'A') e **TRANSFERÊNCIA** (flag 'T'), em que a flag 'A', verifica se um determinado username registado em persistência no servidor tem permissão de se autenticar ('A'=1) ou não ('A'=0) e a flag 'T' verifica se um determinado username tem permissão de tranferência de ficheiros ('T'=1), ou não ('T'=0).

O ficheiro users.csv contêm os usernames, permissões e passwords. As **passwords** encontram-se **cifradas com a chave pública do servidor** para garantir uma mínima confidencialidade dos dados. No entanto trata-se apenas de uma aproximação a um caso mais real, uma vez que na realidade a melhor forma de tratar este procedimento seria aquando do registo de um utilizador, encriptar a palavra passe com uma KDF tipo **Scrypt** (com um **sal** de 16 bytes, **n**>=2^14, **r**=8, **p**=1), com o sal a ser guardado na persistência e a veracidade da password introduzida pelo utilizador a ser atestada através do método verify(password, key) da KDF, dado que esta KDF apenas permite que cada chave seja derivada e verificada uma única vez.

- controlo de autenticação

Quando um cliente se tenta autenticar, será estritamente necessário que tenha a **flag 'A'** com o valor 1. Por defeito, todos os novos utilizadores registados têm **A = 1**. No entanto, para prevenir tentativas de autenticação indevidas, caso um username tente autenticar-se no servidor **três vezes seguidas sem sucesso**, esta tentativa de autenticação é catalogada como **suspeita**, levando à perda de permissão de autenticação no servidor por parte desse username, atualizando o valor da **flag 'A'** para 0 (com o devido registo em persistência).

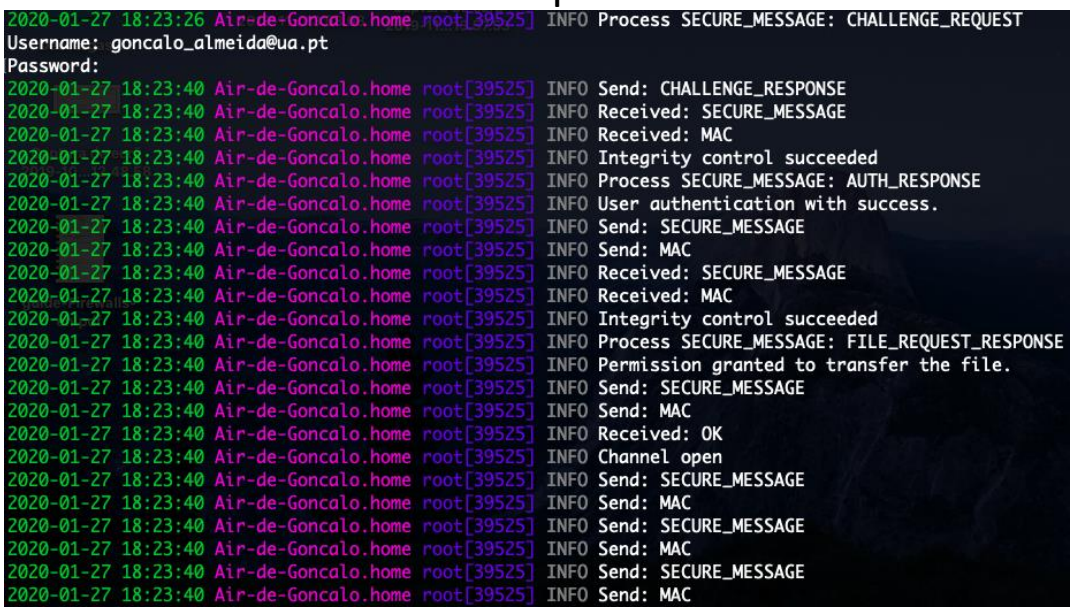
Cliente – autenticação negada



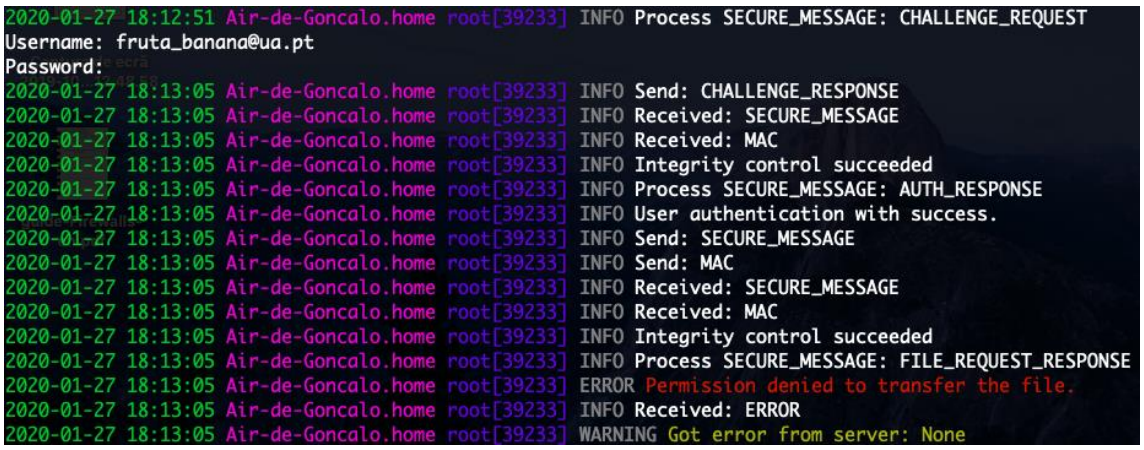
- controlo de transferência

Após ser autenticado, um utilizador pede permissão para transferir um ficheiro. Este requerente só poderá transferir o ficheiro caso o valor da **flag 'T'** seja 1. Caso contrário, o servidor encerra a conexão com este utilizador. Em caso de sucesso, o ficheiro começa a ser transferido logo de seguida.

Cliente – transferência permitida



Cliente – transferência negada



Conclusão

Com a realização deste trabalho, sentimos que conseguimos alcançar as metas estipuladas para o desenvolvimento do mesmo. Assim, conseguimos também colocar em prática e ainda descobrir um pouco mais sobre o que aprendemos relativamente a comunicações seguras, desta vez ao nível da autenticação e mecanismos para garantia da mesma.

Fontes de pesquisa

Cryptography.io

Joao.barraca.pt