universidade de aveiro

SECURE INSTANT MESSAGING SYSTEM – M2

Relatório

Segurança

Departamento de Eletrónica, Telecomunicações e Informática

18 de Dezembro de 2016

Professores:

João Paulo Barraca

André Zúquete

Autores:

Miguel Oliveira nº 72638

Tiago Henriques nº 73046

universidade de aveiro

Índice

universidade de aveiro

Introdução

O objetivo do projeto a ser desenvolvido no âmbito da unidade curricular de Segurança, é permitir a troca de mensagens entre utilizadores através de um sistema seguro composto por um Servidor e múltiplos Clientes que trocam mensagens entre si. Tal sistema tem diversas características comuns com um já existente, *Signal,* que é uma aplicação para *Android* ou *IOS* com um sistema de troca de mensagens instantâneas cifradas, logo seguras. Foi usada a linguagem de programação '*python'*, a biblioteca *'cryptography'* para a primeira parte do projeto e a biblioteca *'pyOpenSSL'* para a segunda parte do projeto.

Na primeira fase do projeto, e após terem sido considerados todos os aspetos relacionados entre comunicações entre entidades, incluindo cifras e controlo de integridade, garantindo "*Message Confidentiality and Integrity* (but not authentication), *Forward and Backward Secrecy",* fazemos agora a ligação para a segunda fase do projeto.

Nesta segunda fase deve ser adicionado o suporte para autorização e autenticação através do Cartão de Cidadão. Isto é, a implementação de mecanismos de preservação de identificação, validação de destino, consistência entre participantes, autenticação de mensagens e controlo de fluxo de informação.

Estrutura:

Existem vários processos que têm de ser suportados no projeto.

- **Connect to Server:** Cliente envia uma mensagem *CONNECT* para o servidor e mais mensagens deste tipo serão trocadas até que o tipo de cifras seja acordado e uma sessão válida segura seja estabelecida.

- **List clients connected:** Cliente envia uma mensagem *LIST* para o servidor. O servidor responde com uma mensagem do mesmo tipo contendo uma lista de clientes disponíveis.

- **Client-to-client connection:** Cliente envia uma mensagem *CLIENT-CONNECT* que é encaminhada pelo servidor para o cliente especificado. Novamente, o tipo de cifras têm de ser acordadas e uma sessão válida e segura tem de ser estabelecida.

- **Client-to-client disconnection:** Cliente envia uma mensagem *CLIENT-DISCONNECT* que é encaminhada pelo servidor para o cliente especificado.

- **Client-to-client communication:** Cliente envia uma mensagem *CLIENT-COM* que é encaminhada pelo servidor para o cliente especificado pelo primeiro cliente.

Geração do certificado e Assinatura

Nesta segunda parte do trabalho, o objetivo e ponto fulcral é a autenticação e autorização das entidades envolvidas no processo de comunicação, utilizando para isso, o Cartão de Cidadão.

Assim, quando um Cliente inicia uma sessão é-lhe pedido para escolher a slot manualmente e introduzir o PIN da autenticação para depois poder assinar todas as mensagens a enviar, de modo a garantir a sua autenticidade a terceiros.

Desta forma, acede-se ao certificado do CC para ter acesso à chave privada necessária para o Cliente assinar as respetivas mensagens.

Cliente

Como já foi referido, um cliente necessita de se autenticar e para isso utiliza o CC. Através do CC é possível extrair o certificado, onde a partir deste, se terá acesso à chave privada RSA para depois proceder à assinatura e obter a chave pública para verificação da assinatura.

Para proceder à assinatura foram usados os primitivos *SHA1* para gerar a *hash*, *PKCS1v15* para padding.

Servidor

O servidor não possui um CC mas precisa de uma identificação, uma maneira de se autenticar para que o Cliente possa saber que é de confiança, ou seja, que é mesmo o Servidor com quem queremos estabelecer uma conexão.

Para isso, foi utilizado o software *xca* para gerar quer o certificado quer a respetiva chave privada RSA . Os certificados do servidor estão armazenados no diretório *'ServerCerts'*, que contém o certificado em si, o Servidor e o certificado Root (CA). Quanto à chave privada, esta está situada no diretório *'ServerKeys'* e é uma chave privada RSA com 2048 bits. Em relação à chave pública, esta pode ser obtida através do certificado.

Verificação dos Certificados

Tanto o servidor como o cliente possuem uma *Store* para proceder à verificação dos certificados recebidos utilizando a biblioteca *'pyOpenSSL'* . A *Store* possui os certificados responsáveis pela emissão dos certificados no cartão de cidadão bem como o certificado do emissor do Servidor.

Sempre que se inicia o estabelecimento de uma ligação (Connect e Client - Connect), são trocados e validados os certificados pelas entidades, porém verificar que o certificado está válido na *Store* não é suficiente, pois a qualquer momento o certificado pode estar revogado, por isso é necessário aceder à entidade emissora para saber a data de revogação. Na nossa implementação sempre que tanto o Cliente como o Servidor iniciam uma sessão ou quando são trocados os certificados, a Store é carregada e os certificados são validados quanto à sua data de revogação através do OCSP, usando a biblioteca *'ocspbuilder'*.

Alterações dos Processos da 1º Fase

Handshake entre Cliente - Servidor (Connect):

Antes de haver qualquer troca de chaves ou informação, tem primeiro de se proceder à autenticação, isto é, quando o Cliente inicia a ligação começa por enviar o certificado extraído do CC bem como o *cipher spec*. O Servidor ao receber o certificado verifica se na sua *Store* este é válido. Esta primeira mensagem é vista como um *'Client-Hello'* usado no modelo TLS. Caso o certificado não seja válido, o Servidor fechará imediatamente a ligação com o Cliente em questão, caso contrário, se o resultado da verificação for positivo, o Servidor envia o seu certificado, o qual também será verificado pela outra entidade, neste caso, pelo Cliente.

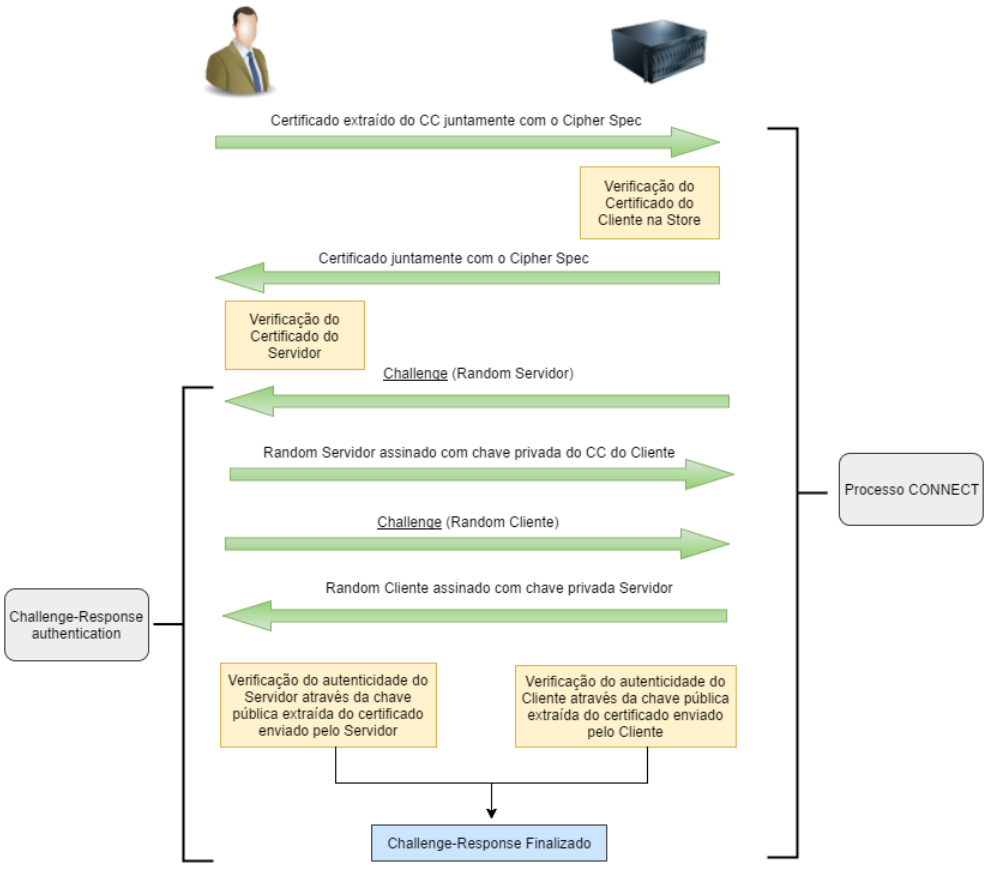
No entanto, a troca dos certificados não basta, é necessário proceder à autenticação das entidades, assim a seguir à troca dos certificados, inicia-se o *Challenge-Response authentication* para verificar a autenticidade das entidades.

Portanto, o servidor envia um *Challenge* para o Cliente, que não passa de um valor random. Este valor tem de ser assinado com a chave privada do CC do cliente e é enviado de volta para o Servidor, onde este último com a chave pública extraída do certificado enviado pelo Cliente,

verifica então se a assinatura corresponde de facto ao Cliente que possui aquele CC, de modo a comprovar a sua autenticidade. Seguidamente, os papéis são invertidos, e o cliente também envia um Challenge para poder verificar a autenticidade do Servidor. Depois de ambas as entidades verificarem com sucesso (com a chave pública dos certificados correspondentes) as assinaturas dos valores randoms, o Challenge Response está completo, e a partir desta fase, todas as mensagens trocadas serão sempre assinadas com a chave privada de cada entidade, e é a partir deste momento que o processo de acordo de chaves é efetuado.

Assim, foram acrescentadas mais 3 fases ao processo de *Handshake* anterior, que podiam ter sido reduzidas, mas como na primeira fase do projeto já nos estávamos a guiar pelo modelo *TLS*, considerámos por bem continuar o mesmo processo de pensamento.

No fim, se o cliente passar todas as fases de verificação com sucesso, a sessão é estabelecida e o Servidor atribui um nível ao Cliente que irá ser necessário posteriormente para realizar o processo de *Bell-Lapadula*, que será mais explicado mais à frente.



Mensagens Secure

Após o estabelecimento da sessão entre o Cliente e Servidor, todas as mensagens trocadas entre as duas entidades serão encapsuladas com o tipo *'Secure'*, onde o conteúdo destas mensagens será cifrado/encapsulado.

Contudo nesta fase, onde a autenticidade das entidades envolvidas tem de ser constantemente verificada, as mensagens do tipo *Secure* terão de ser assinadas pela entidade emissora. Portanto, processos como *Client-Connect*, *Client-Com*, *Client-Disconnect* e *ACK*, serão sempre assinados pela entidade emissora, como também serão assinadas no encapsulamento *Secure*. Por outras palavras, a entidade emissora terá de assinar a mensagem *Secure* e assinar a mensagem que vai dentro do encapsulamento, tendo por isso, de assinar duas vezes.

Desta forma, quando se processa uma mensagem do tipo *Secure*, na primeira fase terá de se verificar a assinatura da mensagem com o conteúdo cifrado, e só depois de feita a decifra, proceder à verificação da assinatura da mensagem vinda no payload, de forma a garantir a total integridade das mensagens.

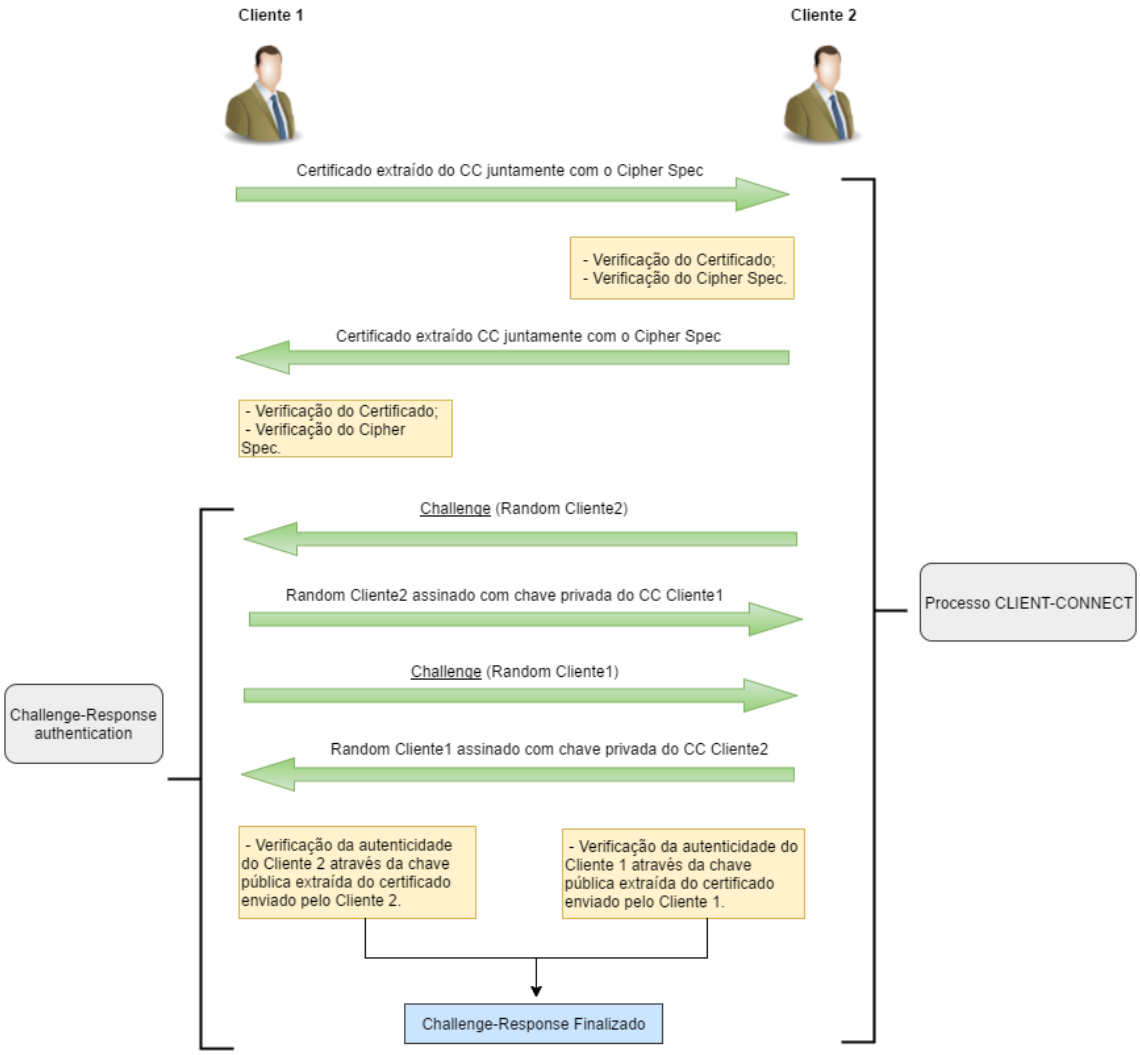
Handshake entre Cliente - Cliente (Client - Connect):

Tal como acontece no processo de *Handshake* entre Cliente - Servidor, entre dois ou mais Clientes também será necessário a garantir a autenticidade dos mesmos, efetuando o processo anteriormente realizado, *Challenge-Response authentication*, para realmente verificar e comprovar a autenticidade dos clientes ligados. Voltando a seguir o modelo TLS, iremos ter mais 3 fases para além das implementadas na primeira fase do trabalho.

Na 1º fase, o Cliente enviará o seu *cipher spec* suportado e o seu certificado. O outro Cliente com o qual o primeiro se quer conectar, verifica o *cipher spec* e o certificado, e se o certificado estiver válido, este Cliente enviará também o *cipher spec* suportado, o seu certificado e o valor random gerado por si que representa *o Challenge*. O Cliente que estabeleceu a conexão recebe o *Challenge* e assina-o, gerando também ele um valor random, onde são enviados o valor random criado e o *Challenge* assinado. O outro cliente procederá do mesmo modo, e se todas as assinaturas forem válidas respetivamente ao correspondente certificado, o *Challenge Response* está completo e todas as mensagens enviadas de aí em diante serão devidamente assinadas.

Logo, cada Cliente terá de assinar o Secure para o Servidor com a mensagem de *Connect* para o outro Cliente, para que tanto o Servidor com o outro Cliente saibam com quem estão a lidar, garantindo a autenticidade das entidades envolvidas. Quando receber a mensagem *Secure* do

Servidor terá primeiro de verificar a assinatura do Servidor, e só depois verificar a assinatura do Cliente com quem quer estabelecer ligação.



Client - Com

Este tipo de mensagens, após estabelecida a ligação entre dois clientes, não sofreu nenhuma alteração quanto às etapas e procedimentos, continuando a ser enviada como *Secure* para o Servidor e depois o Servidor reencaminha a mensagem com encapsulamento *Secure* para o respetivo Cliente. A única diferença é que agora o Cliente que envia o Client-Com tem de assinar a mensagem, de forma a provar que é efetivamente a entidade emissora. Quando o

Servidor recebe a mensagem Secure assinada pelo cliente, necessita também de verificar a autenticidade do Cliente, para só depois proceder ao envio da mensagem para o Cliente destino.

Por fim, o cliente quando recebe o Client-Com tem de verificar a assinatura do cliente emissor, para poder ver a mensagem e responder.

Client - Disconnect

Mensagens que são enviadas quando um Cliente se quer desconectar de outro Cliente. Também não sofreram nenhuma alteração significativa, porém agora quando um Cliente envia uma mensagem deste tipo necessita de a assinar, bem como assinar a mensagem *Secure* que vai encapsular a mensagem de *Client-Disconnect*. Logo, o Servidor, antes de reencaminhar a mensagem tem de verificar a autenticidade do Cliente que enviou a mensagem. Antes de um cliente perder a conexão, tem de verificar a assinatura do Cliente que originou a mensagem bem como a assinatura do Servidor que enviou a mensagem Secure.

Mensagens Ack

Este tipo de mensagens têm um papel muito importante nesta segunda fase do projeto, pois permitem garantir a autenticidade das mensagens de forma a quem recebe uma mensagem, saiba mesmo de quem é a mensagem, ou seja, fique a conhecer a entidade que enviou a mensagem. Para isso, todos as mensagens *Ack* tem de estar assinadas com a chave privada do CC, no caso do Cliente.

Por outro lado, também têm o intuito de providenciar feedback de que a mensagem enviada foi de facto, recebida pelo destinatário.

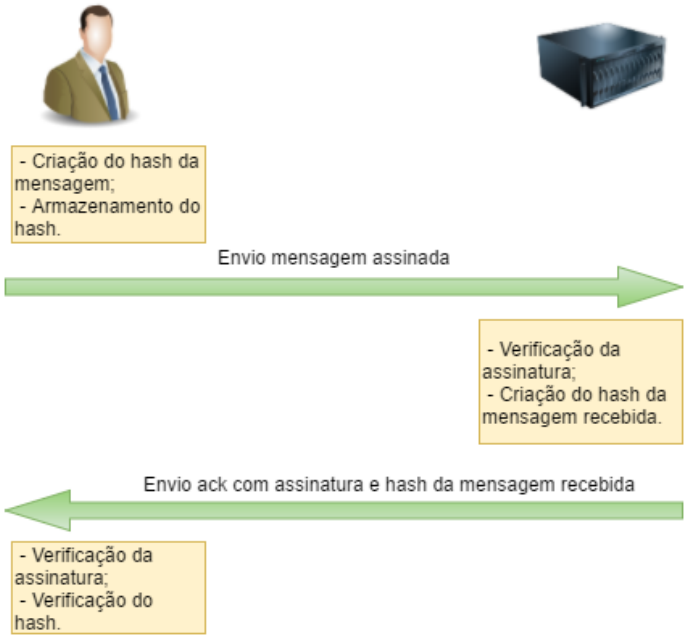
Portanto, as mensagens do tipo *Ack* são essenciais no processo de Destination Validation que vai ser visto mais à frente.

Mensagens Acks entre Cliente - Servidor (Connect)

Visto que esta fase se destina ao handshake entre o Cliente e Servidor, ainda não se chegou a um segredo partilhado, logo, os acks serão enviados normalmente sem serem encapsulados

com o tipo *Secure*, ou seja, não têm o campo signature. Por outro lado, como neste processo as fases iniciais se destinam a efetuar o Challenge-Response authentication, apenas quando este processo estiver completo é que os *acks* passarão a serem assinados para garantir a autenticidade de quem envia as mensagens, até lá os acks são enviados sem estarem assinados.

Nesta fase, as mensagens ack terão como campos a hash da mensagem e a signature.



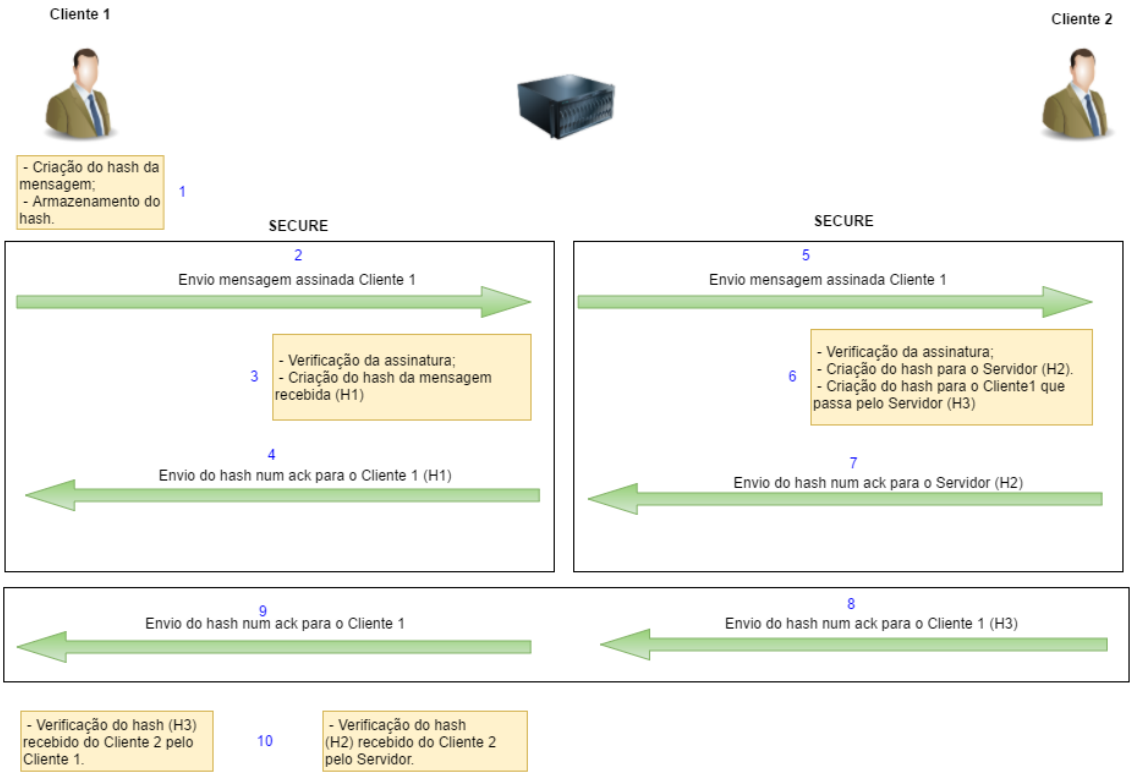
Mensagens Acks do tipo Secure

Cliente - Servidor

Após a sessão entre o cliente e servidor estar estabelecida tendo sido gerado um segredo comum, todas as mensagens em diante serão enviadas com o encapsulamento *Secure*. Desta forma, entre o cliente e servidor, apenas serão trocadas mensagens deste tipo, por isso, temos de garantir que o emissor cliente/servidor tem a confirmação de que a sua mensagem *Secure* foi entregue no destino certo. Portanto, a cada mensagem *Secure* recebida é enviada uma mensagem ack como Secure. Nesta fase, todas as mensagens *acks* enviadas como *Secure* terão de ser assinadas pela entidade emissora, de modo a fornecer autenticidade da mensagem tendo como campos a hash da mensagem e a signature.

Cliente - Cliente

Quando são processadas as mensagens do tipo Client - Connect, Client - Com e Client - Disconnect, temos de ter a certeza que a mensagem do tipo *Secure* chega ao Cliente destino correto. Por exemplo: o Cliente envia uma mensagem *Client-Com* que vai encapsulada como *Secure*. O Servidor então responderá com o *ack* da mensagem *Secure* recebida (como descrito em cima). O Cliente quando receber a mensagem *Secure* terá de enviar uma mensagem de *ack* para notificar o Servidor que recebeu a sua mensagem. Porém, o Cliente recetor também necessita de notificar o Cliente Emissor, assim enviará uma mensagem *ack Secure*, mas este *ack* terá os campos src e dst, que irão cifrados. Quando o servidor recebe a mensagem *ack*, decifra a mensagem e verifica os campos src e dst, onde procederá ao envio da mensagem de *ack* para o destino correto. Nesta fase, as mensagens *ack* terão como campos a hash da mensagem e a signature.



Objetivos do Projeto

Message confidentiality, integrity and authentication

Para ter a troca de mensagens eficiente e segura foi necessário implementar estes 3 importantes pontos:

Confidencialidade

A confidencialidade das mensagens está relacionada com o encapsulamento do tipo Secure, quando estabelecida uma ligação segura entre o cliente e servidor. Assim, com o encapsulamento *Secure*, o conteúdo da mensagem passa a ser cifrado, o que o torna confidencial para entidades que não tenham chegado ao segredo comum final.

Integridade

A integridade da mensagem é alcançada fazendo sempre o hash da mensagem a enviar, onde o destino receberá o hash de origem e gera também ele próprio, o hash da mensagem recebida, enviando o respetivo hash no ack. Este processo permite à origem saber se a mensagem enviada chegou sem alterações de integridade ao destino.

Por outro lado, quando na primeira fase do projeto se efetuava o HMAC das mensagens enviadas, tinha-se apenas o objetivo de garantir a integridade das mensagens.

Autenticação

Sempre que uma entidade emissora pretende enviar uma mensagem, necessita de assinar toda a mensagem com a chave privada respetiva, no caso do Cliente, a chave presente no CC e no caso do Servidor, a chave RSA gerada no software *xca*.

Assim, sempre que a entidade recebe a mensagem assinada procederá à verificação desta com a chave pública presente nos certificados, e desta forma, consegue-se garantir que uma entidade não se faz passar por outra, pois está autenticada.

Destination Validation

Para que uma entidade emissora possa saber que mensagens foram realmente entregues ao destino certo, implementámos um dicionário em que a key é a hash da mensagem a enviar, e o value é a mensagem em si.

Assim, sempre que o Cliente ou Servidor pretender enviar uma mensagem é adicionado ao dicionário a hash da mensagem e a mensagem a enviar. A hash é enviada no campo na mensagem ack, que pode ou não ir encapsulada como *Secure*, dependente da ligação entre Cliente e Servidor estar ou não estabelecida.

Quando uma entidade emissora recebe o ack, esta irá verificar se o hash que recebeu é igual à hash que tem no dicionário, caso coincida será removido o tuplo correspondente, caso não coincida, não é possível saber se a mensagem foi realmente entregue.

Este processo é realizado em todas as mensagens enviadas, excluindo o envio das mensagens *ack*, ou seja, tanto o Cliente como o Servidor, não guardam as mensagens *ack* enviadas.

Identity Preservation

Para que um cliente só esteja associado a um CC, após este se ter conectado com sucesso, o servidor atribui-lhe um número, que irá servir de key no dicionário onde estão guardados os certificados dos clientes ligados e autenticados.

Logo, logo na primeira fase do *Connect*, quando o servidor recebe o certificado de um cliente que se queira ligar, verifica no dicionário, se o certificado recebido já existe. Se já existir, o servidor terminará a ligação com o cliente que pretende estabelecer a sessão.

Information Flow Control

Este processo usa o modelo *Bell - LaPadula* que é usado em aplicações do Governo e Militares onde estabelece métodos de segurança de múltiplos níveis, centrando-se na confidencialidade dos dados e controlo de acesso a informação classificada/restrita.

Portanto, na nossa implementação, quando uma cliente estabelece ligação com sucesso com o Servidor, é-lhe atribuído um nível gerado aleatoriamente de 0 a 3, atualizando o dicionário onde a key é o id do cliente e o value é o valor do nível atribuído. Deste modo, quando o Servidor verifica que é um processo de Client - Com, terá de filtrar os níveis dos clientes conectados, impedindo o envio de mensagens entre clientes onde o nível do cliente src é maior do que o nível do cliente dst. Desta forma, um cliente com um maior nível é como se tivesse informações confidenciais (*Top-Secret*) que não podem ser transmitidas para outros níveis mais abaixo na hierarquia.

Participant Consistency

O objetivo da implementação deste tópico é o recetor de uma mensagem poder detetar se o emissor da mensagem está numa máquina diferente da sua.

Na nossa implementação quando um cliente se conecta com sucesso com o Servidor, é guardado num dicionário o seu certificado, associando o número do cliente ao seu certificado, assim clientes com o mesmo certificados são eliminados da conexão. Desta forma, para ver em que máquina o cliente está no momento, basta verificar na fase de Client - Connect.

Assim, na *fase 5,* quando o segredo comum já está gerado, é criado a hash da informação do hardware, e é cifrado tanto a hash da informação do hardware como a informação do hardware, que é enviado numa mensagem *Secure*. A criação da hash garante a integridade da mensagem e o facto das especificações do hardware irem cifradas, tem como o objetivo, se por qualquer motivo houver um ataque, essa informação não possa ser acedida.

Depois, quando o outro Cliente recebe a mensagem, decifra o seu conteúdo, e antes verifica se já existe algum cliente com aquele certificado que esteja ligado, e caso exista, verifica a hash que recebeu com a que tinha para saber se ocorreu ou não alguma mudança, fazendo sempre o update da hash e da informação do hardware.

Nota: Na nossa implementação, este processo apenas funciona quando desativamos o *Identity Preservation*.

Conclusão

A segunda parte do projeto a desenvolver tinha de suportar algumas funcionalidades de segurança e foi concluído que estes objetivos mais relevantes foram assegurados.

- Implementation of the Identity preservation mechanisms;

- Destination validation;

- Participant consistency;

- Message Authentication and Information Flow Control.

Referências

http://www.pyopenssl.org/en/stable/api/crypto.html

http://pkcs11wrap.sourceforge.net/api/

http://xca.sourceforge.net/

http://stackoverflow.com/questions/159137/getting-mac-address

http://stackoverflow.com/questions/3103178/how-to-get-the-system-info-with-python

https://pypi.python.org/pypi/py-cpuinfo