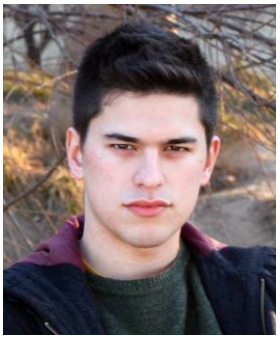
UPA TRANSPORTES

INSTITUTO SUPERIOR TÉCNICO
GRUPO A55
GITHUB URL: HTTPS://GITHUB.COM/TECNICO-DISTSYS/A_55-PROJECT
DOCENTE: MIGUEL NEVES

SISTEMAS DISTRIBUÍDOS
LICENCIATURA ENGENHARIA INFORMÁTICA E DE COMPUTADORES

AUTORES:

MIGUEL VERA, Nº 78980

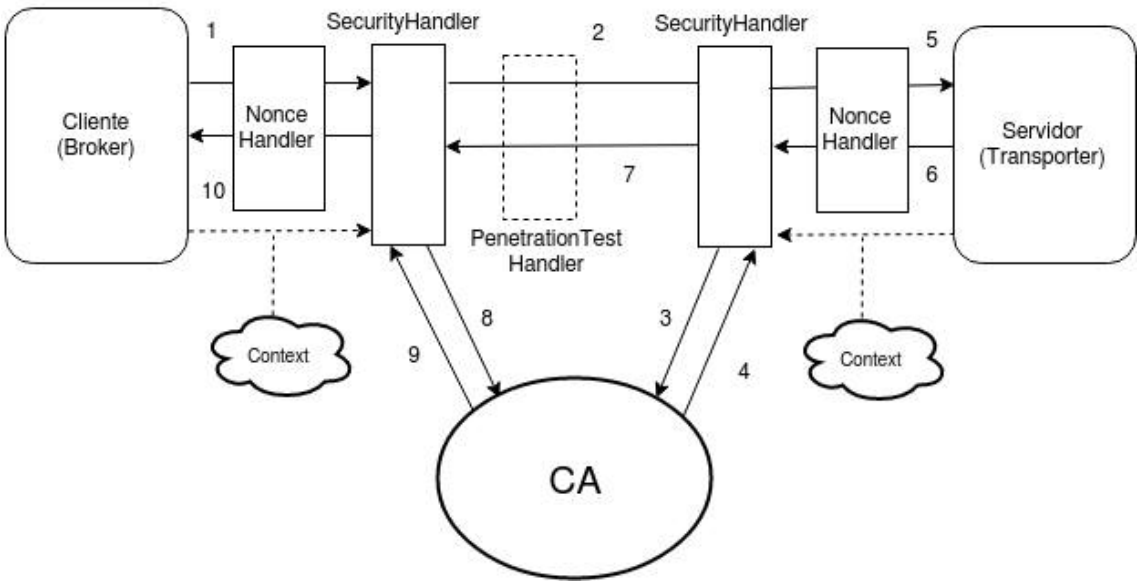


PEDRO ASCENSÃO Nº 78961

JORGE PESSOA Nº 78839

13 MAIO 2016

SEGURANÇA:



RACIONAL:

Para a concretização do projeto foram estabelecidos objetivos quanto a segurança da informação a ser trocada entre Broker e Transporters. As mensagens devem ser autenticadas e não repudiáveis à priori pelo emissor. Estas devem ser ainda protegidas contra repetições maliciosas.

O esquema acima apresentado traduz a arquitetura implementada para a realização destes requisitos de segurança, descrevendo uma troca simples de mensagens iniciada pelo cliente (as ações estão numeradas pela ordem de acontecimentos). Foi criada uma entidade CA que é responsável pela emissão de certificadas (assinando os certificados de cada entidade com a sua chave privada). No âmbito do projeto, a CA devolve certificados codificados em base64 como resposta aos pedidos que lhe são realizados.

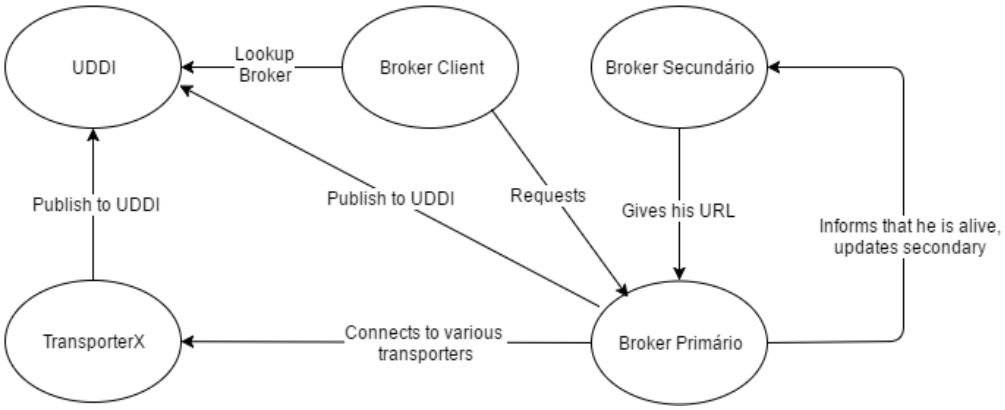
Para o processamento das mensagens foram criados dois handlers que operam tanto no lado do Transporter como do Broker: O NonceHandler e o SecurityHandler.

O primeiro Handler é responsável pela gestão dos nonces das mensagens. Quando este processa uma mensagem outbound, gera um long aleatorio e não utilizado antes (SecureRandom), sendo este valor (nonce) adicionado ao header da mensagem. Quando recebe uma mensagem inbound o seu nonce é verificado. O nonce não pode ser alterado por um agente malicioso devido ao cálculo da assinatura digital da mensagem, visto que a verificação da assinatura digital de uma mensagem inbound ocorre antes do processamento do nonce (explicado em maior detalhe em baixo).

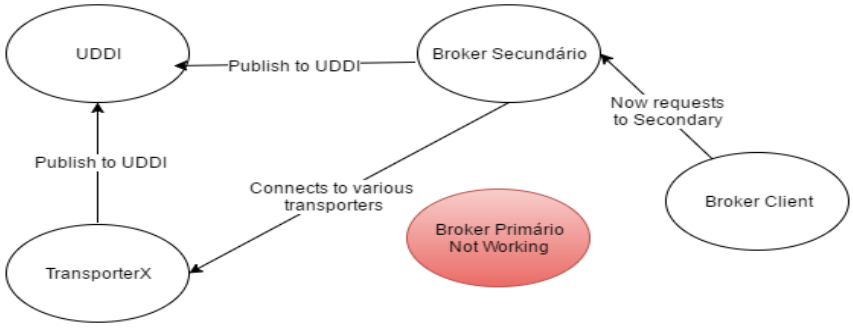
O SecurityHandler é responsável pela autenticação das mensagens, cálculo das assinaturas digitais e gestão dos certificados já obtidos. Para uma mensagem outbound o handler trata de construir a sua assinatura digital, com a chave privada da Entidade (presente na sua keystore). A construção da assinatura é feito a partir da concatençação do conteudo da mensagem (body) com o nonce da mensagem e com o identificador da entidade (também enviado num header da mensagem). Desta forma qualquer alteração a um destes parâmetros vai causar um erro na verificação da assinatura digital. A verificação da assinatura é feita com base na informação da concatenação body+nonce+identificador. Para garantir a autenticação de cada mensagem o Handler requesita o certificado da entidade que enviou a mensagem ao CA (caso já nao o tenha em cache), verifica que este certificado foi devidamente assinado pelo CA com a sua chave publica e só depois verifica a assinatura digital da mensagem com o certificado enviado pela CA. A classe estática Context faz uso do isolamento de cada processo na sua própria JVM para passar informação entre a entidade e o SecurityHandler.

Por fim, para efeitos de teste foi criado um PenetrationTestHandler que pode ser adicionado no fim de uma cadeia de handlers, de qualquer um dos lados da comunicação. Este permite, para efeitos de teste, realizar alterações ao conteudo de uma mensagem outbound, de forma a simular um ataque man-in-the-middle

REPLICAÇÃO:



Para aplicar replicação no projecto foi necessária a criação dum Broker secundário. Um Broker secundário é uma outra instância do BrokerPort, por isso, foi necessário alterar o comportamento da classe para conter o comportamento. O BrokerPort agora possui um BrokerPortClient
No funcionamento normal, o Broker primário regista-se no UDDI, assim como, os vários Transporters. O Broker primário avisa o secundário que está vivo, e quando o seu estado é alterado, este avisa o secundário de que deve mudar o seu estado também.



Quando, devido a algum erro, o Broker primário deixa de funcionar, o secundário começa a funcionar da mesma forma que o primário (no seu funcionamento normal), publicando-se, recebe pedidos e conecta-se às transportadoras.

RACIONAL:

De forma que haja ara poder falar com o outro Broker, o primário necessita de comunicar com o secundário e vice-versa.

Além disso, foi necessário criar 3 novas funções no WSDL. A função update que recebe um transportView e que adiciona ao secundário caso esse não tenha esse transport ou apenas faz update ao estado desse transport. Esta função é usada quando o primário executa as funções requestTransport, viewTransport. O clearTransport também muda o estado do primário, logo quando esta função é executada, o primário invoca este método também no secundário.

A função sendInfo é usada para o secundário enviar o seu URL para o primário poder ligar-se ao secundário que, sendo assim, não precisa de se publicar no UDDI.

A função primaryLives tem a função de avisar o secundário que o primário ainda está vivo. Se o secundário não receber esta chamada num determinado número de segundos, assume que o primário já não está ligado, e assume-se como Broker primário.