

Sistemas Distribuídos

2016/2017




Komparator

Relatório de Segurança

A57

Repositório

<https://github.com/tecnico-distsys/A57-Komparator>

		
64776	69836	72887
João Cardoso	Rita Ferreira	Ângela Ferreira
joaocardoso816@gmail.com	rita.c.ferreira@live.com	angela.ferreira@tecnico. ulisboa.pt
Professor do laboratório: Luís Sá Couto		

Solução de Segurança

Para atingir estes requisitos de Segurança desejados para esta entrega, foram desenhados Handlers, cada um responsável por interceptar a mensagem e alterá-la (e verificá-la) de acordo com o seu requisito. O esquema que se segue representa a organização geral do projeto e como estes Handlers estão organizados:

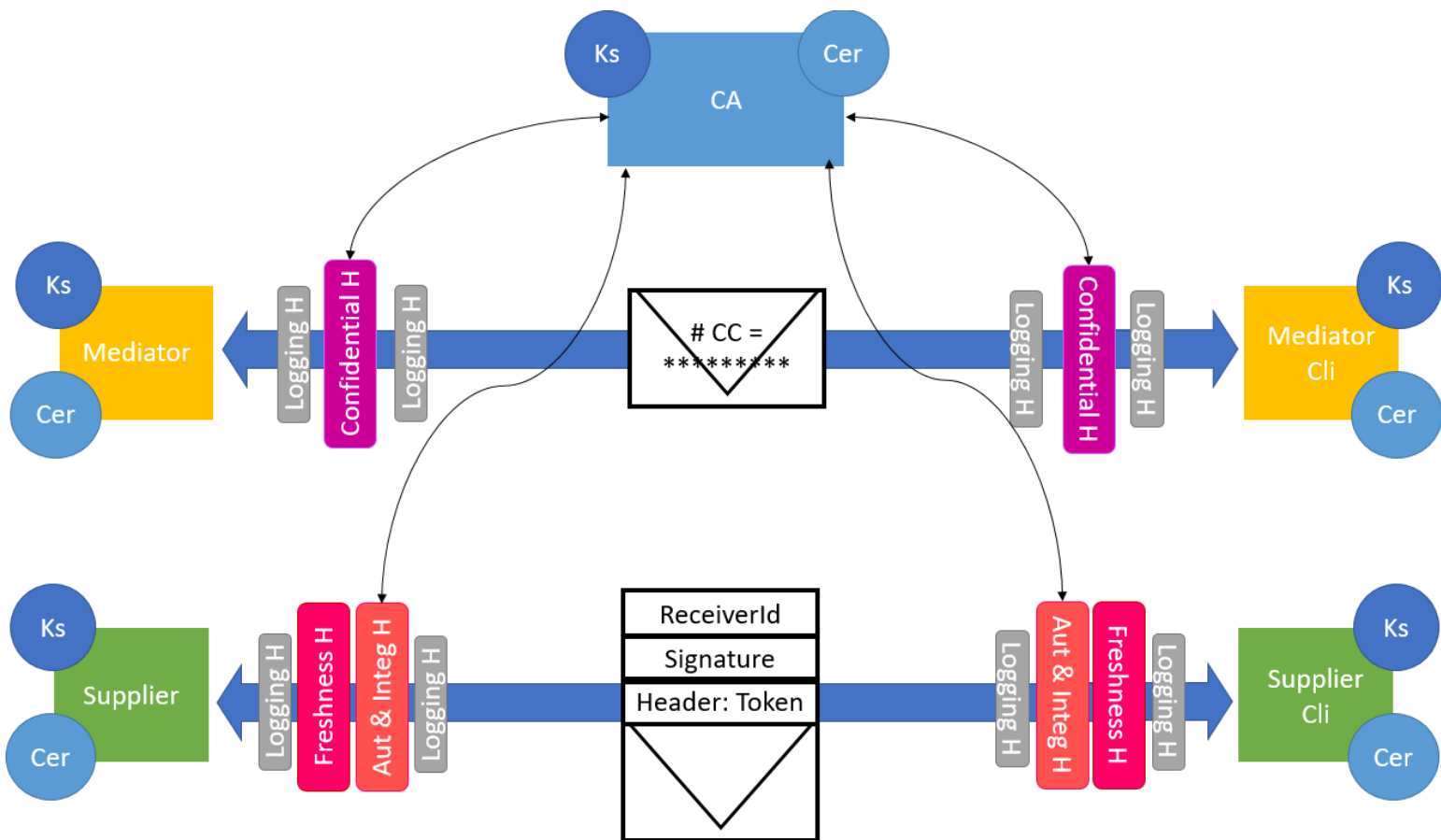


Figura 1 – Solução de Segurança

Cada caixa retangular representa a entidade indicada na figura. No caso do Supplier Cli, ele é sempre o Mediator, visto que só ele que comunica diretamente com o Supplier. Quanto ao Mediator Cli, este tanto pode representar o Supplier como um cliente que comunica com o Mediator.

Os Ks representam o Key Storage de cada entidade (nomeadamente os ficheiros “.jks”). Os Cer representam os certificados.

À saída das caixas aparecem os handlers presentes no projeto. O Logging Handler aparece neste esquema porque está a encapsular os Handlers de Segurança, como pedido, mas só serve para efeitos de teste.

Na interação do Supplier, o primeiro Handler a entrar em ação é o Freshness Handler, que trata da frescura. Para isso, ele intercepta a mensagem e coloca um header com um Token gerado aleatoriamente, como apresentado no envelope da figura. Este Token é posteriormente guardado no recetor, num ConcurrentHashMap. Isto permite verificar se alguma mensagem está a ser reenviada – basta comparar o Token com os que estão guardados no recetor. A mensagem de retorno também recebe um novo Token. Os Tokens são guardados com um Timestamp que serve para limpar a lista de Tokens e evitar desperdício de memória.

Em seguida é ativado o Autenticity Integrity Handler, que é responsável por verificar se a mensagem enviada não foi alterada e se o autor é legítimo. Para isso, a mensagem à saída adiciona dois elementos ao Header: a assinatura da mensagem e o nome do recetor. A última serve para facilitar a obtenção do nome entidade – provavelmente há uma solução mais elegante para o fazer, mas não foi possível procurá-la por questões de tempo.

Quanto à assinatura, o cliente obtém a sua mensagem e cifra-os com a sua própria chave privada, obtida através do Key Store. A versão entregue do projeto falha neste ponto, devolvendo a chave privada da KeyStore a null. No entanto, se funcionasse, ele colocaria a assinatura no Header. À entrada de uma mensagem com assinatura, este pede o certificado de quem envia a mensagem ao CA e verifica por comparação se o remetente da mensagem era uma autoridade certificada. A primeira verificação para descartar a mensagem é aqui. Caso o certificado seja autentico, verifica-se se a assinatura recebida está correta, recorrendo à chave pública. Se não estiver, a mensagem é descartada.

Por fim, apresenta-se o handler que cifra o número de cartão de crédito sempre que é chamado o método buyCart(). O primeiro passo é verificar o nome do método e em seguida procurar o elemento de input relativo ao número do cartão. Este é encriptado com uma public key obtida a partir do certificado do recetor, que foi pedido ao CA. Aqui usa-se o mesmo método descrito acima, e a mensagem será descartada caso o certificado não tenha sido enviado por uma autoridade. Caso contrário, a mensagem é enviada e, quando recebida pelo Mediator, este decifra-a com a sua private key.

Problemas da Solução Atual

A solução atual apresenta algumas falhas, tal como já foi referido acima. Embora a funcionalidade esteja totalmente implementada, não foi possível testar os Handlers exceto os relativos à Frescura e ao seu ataque. O principal impedimento foi não ser possível testar o projeto na maior parte do dia da entrega visto que estávamos a obter apenas erros de HTTP Connection Refused do lado do Mediator.

Mensagens SOAP Capturadas

Devido aos problemas referidos anteriormente, só poderemos mostrar aqui mensagens entre Supplier e Supplier-CLI (correndo a SupplierApp.java presente no Supplier-ws-CLI).

Quadro I – Duas mensagens SOAP capturadas

Inbound SOAP Message - Exemplo 1

```
<S:Envelope xmlns:S="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/">
  <SOAP-ENV:Header>
    <ns:tokenHeader xmlns:ns="http://komparator">
      aJhYWCiLMTq7MgcM25sJJANG13/8cLyVbLGTFgo4e9Q=
    </ns:tokenHeader>
  </SOAP-ENV:Header>
  <S:Body>
    <ns2:ping xmlns:ns2="http://ws.supplier.komparator.org/">
      <arg0>client</arg0>
    </ns2:ping>
  </S:Body>
</S:Envelope>
```

Explicação

A mensagem acima apresenta tags XML que representam cada parte do corpo da mensagem SOAP – O Envelope, contendo os Headers e o Body.

O Header apresenta um Token aleatório, que foi colocado pelo Handler da Frescura.

O Body mostra a recepção de um ping(), vindo do Supplier. Apenas possui um argumento, arg0, cujo conteúdo é cliente.

Outbound SOAP Message - Exemplo 2

```
<S:Envelope xmlns:S="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/">
  <SOAP-ENV:Header/>
  <S:Body>
    <ns2:clear xmlns:ns2="http://ws.supplier.komparator.org/"/>
  </S:Body>
</S:Envelope>
```

Explicação

Este exemplo é muito semelhante ao anterior. As principais diferenças estão no conteúdo vazio do Header e na mensagem a enviar, que é um clear e também não tem argumentos.

A ideia inicial era apresentar uma das mensagens SOAP capturadas ao executar a operação buyCart entre mediator cli (como se estivessemos a simular um cliente a utilizar o Komparator) e o mediator. A escolha desta operação deve-se ao facto de esta não só demonstrar

explicitamente a concretização dos requisitos de segurança entre Mediator e Mediator-Cli, mas também entre Suppliers e Supplier-Cli (que neste caso é o Mediator).

Com o funcionamento dos restantes Handlers, seria possível observar mais elementos no header e mensagens com corpo mais preenchido.

Testes à Autenticidade, Integridade e Frescura

Para comprovar o sucesso no desenho das soluções implementadas, recorreremos a um conjunto de Handlers a que chamámos Attack Handlers. Não foi necessário desenhar um Attack Handler que garantisse a confidencialidade, visto que esta é comprovada usando o Logging Handler e verificando que o número de Cartão de Crédito está encriptado e que mesmo assim a solução funciona o método buyCart executa como esperado.

Assim, implementaram-se cinco Attack Handlers. Para os testar, basta descomentar a linha de código correspondente nas handler-chains do Supplier e o Supplier-cli e correr o projeto normalmente:

Attack Handler	# linha	Descrição
AntiFreshness	16 - 18	Este ataque consiste em alterar os tokens das mensagens de saída para um token permanente para provar que tokens iguais não são aceites.
CertificateAttackHandler	19 - 21	Este ataque consiste na simulação de um certificado falso introduzido.
DigitalSignatureAttackHandler	22 - 24	Este ataque consiste na alteração de um dos bytes da mensagem SOAP.
DigitalSignatureAttackHandler2	25 - 27	Este ataque consiste na alteração de todos os bytes da mensagem SOAP.