Segurança em sistemas SOA

Uma Análise dos Padrões de design e tecnologias de segurança para sistemas Java orientados a serviços

Alexandre Eleutério Santos Lourenço*

Resumo

Este artigo tem como objetivo apresentar e analisar práticas a serem seguidas no desenvolvimento de serviços, a fim de fortalecer a segurança do sistema contra ataques. Serão apresentados no artigo tecnologias de segurança no desenvolvimento de web services (com enfoque na tecnologia Java), a fim de fornecer uma visão inicial do desenvolvimento de segurança em web services. Também serão apresentados conceitos de padrões de design de segurança no desenvolvimento de serviços, enfocando nos padrões desenvolvidos por Thomas Erl, considerado uma das maiores autoridades em SOA. Ao final do trabalho, será realizada uma análise geral dos padrões e tecnologias apresentados, ressaltando suas principais características.

Palavras-chave: Segurança; SOA; Design Pattern, Java.

Introdução

No mundo do desenvolvimento de serviços, especialmente os projetados para serem disponibilizados na internet, é necessário enfatizar a proteção dos serviços. Isso ajuda a impedir problemas como consumidores mal intencionados utilizando os serviços para explorar falhas do banco de dados, roubo de dados sigilosos do sistema, entre outros tipos de ataques.

De acordo com matéria publicada pela Gartner, empresas que trabalharem a sua maturidade e eficiência em segurança de TI em 2010, terão uma redução de custos de 3 a 6% em 2011. A mesma matéria também cita que, a despeito da crise financeira que assolou o mundo recentemente, os investimentos de segurança em TI sofreram uma pequena queda de apenas 1% (de 5% em 2009 para 6% em 2010), isto demonstra a importância que o tema possui dentro da infra-estrutura de uma empresa.

*Alexandre Eleutério Santos Lourenço é Analista de Sistemas, Graduado em Ciência da Computação pela PUC-SP, Pós-Graduando em Engenharia de Software - SOA no IBTA. Email: alexandreesl@gmail.com

Através dos padrões e técnicas apresentadas a seguir, a aplicação a ser desenvolvida terá realizado medidas que ajudaram a se prevenir dos problemas relacionados com segurança ao exporem seus serviços para a rede, alcançando maior eficiência e maturidade, o que diminui os problemas com ataques, reduzindo os prejuízos e conseguindo reduções de custos, como a redução projetada pela Gartner.

Durante o artigo será apresentado material complementar, como gráficos e pequenos trechos de código (quando aplicável), a fim de enriquecer o material apresentado.

Revisão de Literatura

1. SOA

Com o passar dos anos, muitas empresas no segmento de tecnologia se apropriaram da palavra SOA (Service Oriented Architecture) para vender ferramentas, utilitários, e todo o tipo de software, acabando por associar a palavra SOA com tecnologias, o que causou frustração nas empresas clientes, que compraram ferramentas que não atingiam suas expectativas, e acabavam por culpar a metodologia como se essas ferramentas fossem sinônimas do uso de SOA.

A arquitetura SOA compreende mais do que uma ferramenta ou tecnologia, consistindo de uma gama de conceitos que fazem parte de sua definição. De acordo com Thomas Erl, considerado a principal autoridade em SOA, podemos dividir a computação orientada a serviços em:

- Arquitetura orientada a serviços.
- Serviços e Lógica de orientação a serviços.
- Inventário de Serviços.

Cada um desses conceitos serão explicados nas seções a seguir.

1. Arquitetura orientada a serviços

O alvo da arquitetura orientada a serviços consiste na eficiência, agilidade e produtividade, colocando o desenvolvimento de serviços como principal objetivo do desenvolvimento. Consiste de combinar diferentes tecnologias, produtos, linguagens de programação, sistemas legados, entre outros, de forma transparente e que permita uma fácil manutenção, além de refletindo os fluxos da lógica de negócio da empresa, de modo a aproximar as áreas de negócio das áreas de TI.

2. Serviços e Lógica de orientação a serviços

Um serviço é a unidade mais básica na implementação da computação orientada a serviços. Tecnicamente falando, entende-se por serviço como um programa fisicamente

independente, com padrões de design bem definidos, que realiza uma atividade de negócio da empresa. Todo serviço deve atender os padrões de design a seguir:

- Ter um contrato de serviço padronizado, onde estão definidas as operações que o serviço disponibiliza.
- Baixo acoplamento de serviço, ou seja, o serviço deve ser independente, com pouca dependência com outros serviços.
- Abstração, um serviço deve abstrair ao máximo a sua implementação, de modo que os seus consumidores desconheçam os detalhes de sua implementação.
- Reusabilidade, um programa só pode ser considerado serviço caso implemente atividades de alto grau de reusabilidade através da arquitetura tecnológica da empresa.
- Autonomia, um serviço deve ter o máximo possível de independência com relação aos recursos e o ambiente no qual está inserido.
- Independência de estado (stateless), um serviço por padrão deve se manter independente de "instâncias de processo", executando suas atividades de maneira atômica, independente de processos de execução específicos.
- Disponibilidade de serviço, um serviço necessita ter uma fácil visibilidade, de modo que possa ser facilmente utilizado.
- Composição de serviços, serviços devem prover uma interface que permita que sejam encadeados em grupos (composições) a fim de implementar processos de negócio.

3. Inventário de serviços

Um inventário de serviços consiste de um conjunto (normalmente implementado por ferramentas como um ESB (Enterprise Service Bus)) de serviços agrupados e disponibilizados na forma de um "catálogo" de serviços.

2. Design Pattern

De acordo com a definição feita pelo famoso "Gang of Four" (Erich Gamma, Richard Helm, Ralph Johnson e John Vlissides) um design pattern (também chamado de padrão de design) consiste de uma descrição de um problema, que ocorre várias vezes e em várias situações, acompanhado por uma solução, e uma descrição das conseqüências da implementação da solução no ambiente como um todo.

3. **Segurança da informação**

Segurança da informação compreende por um conjunto de medidas, políticas e processos para a manipulação das informações, medidas estas que costumam ser estipuladas por grupos formados nas empresas especificamente alocados para esse fim, tendo por objetivo garantir a segurança das informações trafegadas dentro de uma corporação, através de cinco pilares fundamentais: integridade, disponibilidade, não repúdio (pilar que visa garantir que o autor de determinada informação não possa negar a sua autoria), autenticidade e confidencialidade.

4. **Java**

Criada em 1995, Java é uma das mais linguagens de programação mais utilizadas no mundo, possuindo capacidades de se estender e adaptar que a tornaram presente nas mais diversas áreas do desenvolvimento de software.

Desenvolvimento

Padrões de design de segurança em SOA

Iniciando o estudo sobre segurança em SOA, serão abordados agora os padrões desenvolvidos por Thomas Erl, considerado uma das maiores autoridades em SOA.

Pattern Exception Shielding

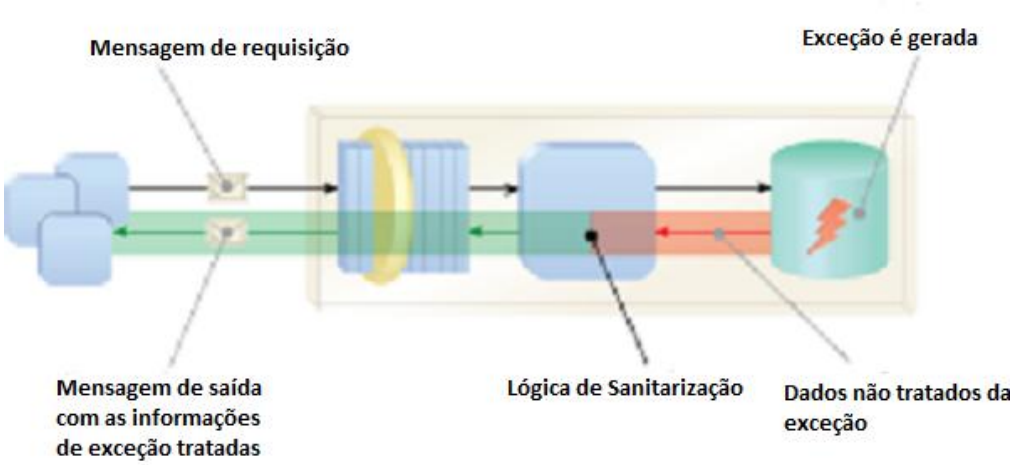

Figura 1 – Desenho da solução do pattern Exception Shielding

(Fonte: Adaptado de ERL, THOMAS. **SOA Design Patterns**, 1. ed. Prentice Hall.)

Muitas vezes, durante o desenvolvimento de uma aplicação, a equipe desenvolvedora insere código da implementação no disparo das exceções (erros), de modo a facilitar a identificação dos erros durante a etapa de testes. Essa prática, porém, não deve ser mantida

após a promoção do sistema para a produção, visto que abre uma brecha para que usuários maliciosos utilizem essa informação para ataques contra o sistema.

Um exemplo desse tipo de problema é quando no disparo de uma exceção, a mensagem da exceção contém detalhes da configuração da conexão do banco de dados da aplicação. Desse modo, um usuário malicioso pode deliberadamente forçar o disparo dessa exceção, através do envio de informações conflituosas que causem esse erro, e assim obter os dados de acesso da base de dados.

O pattern Exception Shielding tem por meta disponibilizar uma solução para esse problema, através do conceito de "sanitarização" das exceções. Toda exceção que for disparada pelo sistema, deve passar antes por um processo de checagem, onde são checadas as informações que estão sendo enviadas pelo sistema. O processo então realiza a substituição de informações consideradas sensíveis e que não podem ser enviadas pelo sistema, por informações mais seguras, e então envia a exceção para o cliente consumidor do serviço, sem as informações comprometedoras.

No processo de Sanitarização das informações, deve ser formalizado entre os membros da equipe que tipos de informações são consideradas sensíveis, e que tipos de informações devem ser retornadas nesses casos. Essa formalização pode ser elaborada em conjunto com a equipe de manutenção, de modo a criar mecanismos como códigos de mapeamento, a fim de facilitar e centralizar o controle das exceções e facilitar a identificação dos erros.

Uma forma recomendada de se aplicar a lógica responsável por fazer a checagem das exceções é centralizando essa lógica como um serviço utilitário, facilitando também dessa forma as eventuais manutenções que forem necessárias de se realizar nessa lógica. Outro ponto de atenção a ser seguido é a questão do log da aplicação, onde uma alternativa seria gravar em log as informações sensíveis da exceção dentro de um arquivo de log protegido no sistema, afim de que essas informações fiquem disponíveis para a equipe de manutenção, sem comprometer a segurança.

Impacto da aplicação do pattern de design no sistema

Por conta da sanitarização de mensagens remover os detalhes do erro ocorrido, esse pattern pode dificultar consideravelmente o rastreio e identificação dos erros, dependendo da forma que for aplicado. Desenvolve-lo de forma que possa ser ativo ou desativado dinâmicamente durante a execução pode auxiliar a aliviar esse tipo de problema. Devido à lógica de tratamento de exceções somente ser chamada no momento em que um erro ocorre, o impacto do pattern na performance dos serviços é mínimo.

Este pattern pode ser utilizado pelo Pattern Service Perimeter Guard (a ser explicado adiante neste artigo), de modo a isolar o seu funcionamento para reutilização através de vários serviços.

Pattern Message Screening

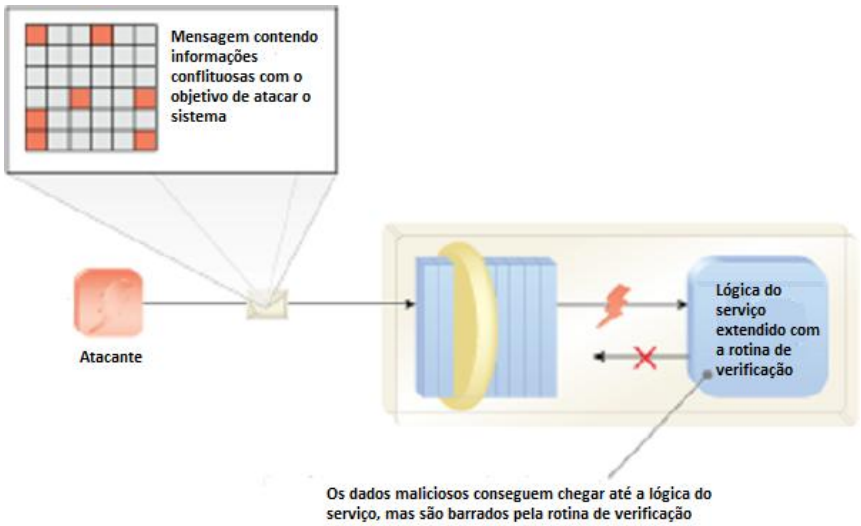

Figura 2 – Desenho da solução do pattern Message Screening

(Fonte: Adaptado de ERL, THOMAS. **SOA Design Patterns**, 1. ed. Prentice Hall.)

Durante o processamento de uma requisição de serviço, são enviados para o serviço dados de entrada (input), como dados para cadastro, verificação do crédito etc. Esses dados podem, em sua concepção, conter conteúdo prejudicial para a execução do serviço, como má-formatação, tipos de dados incorretos etc. Esse conteúdo prejudicial pode tanto ser enviado de maneira acidental, quanto na intenção de promover um ataque ao sistema, tipo de ataque conhecido como ataque de injeção (injection attack).

Uma solução para esse tipo de problema é o pattern Message Screening, onde todo o dado recebido pelo serviço é tratado como prejudicial, até que se prove o contrário. Dessa forma, são criadas rotinas dentro da implementação do serviço, responsáveis por realizar a verificação dos dados recebidos pelo serviço, rejeitando os dados inválidos que venham a ser enviados para o serviço. Como essa lógica adicional reside dentro do serviço, não há quaisquer mudanças que precisem ser feitas no lado consumidor. As rotinas responsáveis pela verificação desses dados devem ser invocadas sempre que o serviço receber novos dados para processar.

Durante o design da lógica de verificação, vários aspectos devem ser considerados, como:

- No caso de mensagens criptografadas, a lógica de verificação precisa ter acesso ao mecanismo de descriptografia das mensagens.
- Entre as verificações a serem feitas nas mensagens, podem ser necessárias verificações binárias, caso o serviço trabalhe com arquivos. Desse modo, devem-se fazer integrações da lógica com mecanismos de filtragem antivírus.
- Na construção da lógica de verificação dos dados, deve-se atentar fortemente na questão da performance, pois caso a lógica se torne muito lenta e demande muitos recursos computacionais, pode-se ela própria ser alvo de ataques externos, ou se tornar um gargalo na execução do serviço.
- Apesar da lógica de verificação puder ser isolada como um serviço único utilitário, tal estratégia pode gerar um ponto único de falha para todo o sistema, e, portanto deve ser analisado com cuidado, sendo muitas vezes preferível nesse caso manter lógicas independentes para cada serviço.
- No planejamento das mensagens do serviço, recomenda-se utilizar tipos de dados o mais específicos possível, evitando tipos como o String, que podem aceitar grandes variedades de dados, a fim de filtrar adiantadamente parte do trabalho da lógica de verificação.

Impacto da aplicação do pattern no sistema

Devido ao seu alto grau de complexidade, e da enorme variedade de verificações que podem ser realizadas sobre dados, a lógica de verificação pode ter a sua latência de performance seriamente prejudicada com o passar do tempo, gerando um gargalo que pode vir a se tornar um problema para o funcionamento dos serviços, causando problemas de requisições legítimas deixando de ser atendidas devido a quedas ocasionadas pela rotina de verificação dos dados. Portanto, é muito importante que o desenvolvimento e as manutenções da rotina sejam feitos de forma planejada e com um alto controle.

Pattern Trusted Subsystem

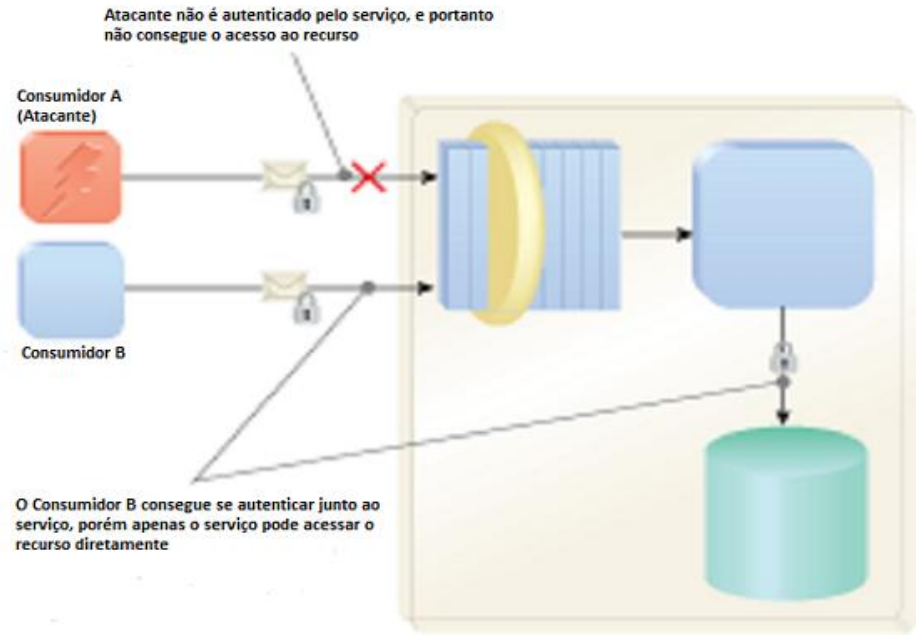

Figura 3 – Desenho da solução do pattern Trusted Subsystem

(Fonte: Adaptado de ERL, THOMAS. **SOA Design Patterns**, 1. ed. Prentice Hall.)

Um sistema utiliza diversos recursos na sua execução, como bases de dados, FTPs etc, que são utilizados pelos serviços para realizar as suas operações. Quando no design de um serviço, o design do serviço é especificado de modo que os consumidores possam ter acesso direto aos recursos do serviço, como as bases de dados, a segurança do recurso se torna comprometida com o risco de acessos maliciosos, além de potencializar um alto acoplamento entre os consumidores e o sistema onde os serviços estão inseridos.

Para resolver esse problema, o pattern Trusted Subsystem propõe que os serviços gerenciem o acesso a esses recursos, como um subsistema, e que os consumidores podem acessar apenas através de credenciais. As credenciais que os consumidores utilizam são utilizadas apenas para que elas se autentiquem nos serviços, ao passo que o serviço utiliza suas próprias credenciais para acessar remotamente os recursos.

Uma verificação importante que os serviços também devem realizar é da autenticidade do serviço que enviou a solicitação de acesso do consumidor (nos casos de fluxos compostos), ao invés de permitir que qualquer processo do sistema tenha acesso ao recurso. Isso evita que possíveis ataques venham de usuários tentando simular serviços autênticos do sistema, e assim burlem a segurança. Dessa forma, quando temos vários serviços atuando como um fluxo único, podemos ter cada serviço como um subsistema confiável (Trusted Subsystem)

independente, onde os acessos aos recursos que cada serviço gerencia são validados independentemente.

Impacto da aplicação do pattern no sistema

Por conta de gerenciar todo o acesso a determinados recursos, caso um serviço implementando este pattern seja burlado, todo o acesso aos recursos que ele gerencia se tornam comprometidos. Por essa razão, esse tipo de serviço necessita de atenção especial, devido a ser um alvo primário de ataques na infra-estrutura de uma empresa.

Pattern Service Perimeter Guard

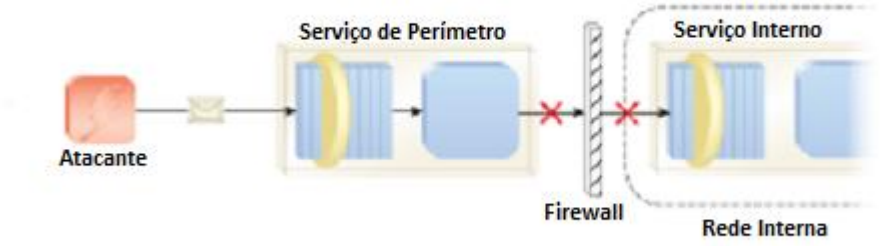

Figura 4 – Desenho da solução do pattern Service Perimeter Guard
(Fonte: Adaptado de ERL, THOMAS. **SOA Design Patterns**, 1. ed. Prentice Hall.)

Dentro da área de TI de uma empresa que se utiliza da arquitetura orientada a serviços, existem serviços que são disponibilizados para a rede externa da empresa, enquanto existem serviços de natureza mais delicada que permanecem na rede interna da empresa.

Durante o desenvolvimento de software, principalmente softwares que trabalham com a troca de informações entre diferentes empresas, são comuns situações onde consumidores de serviços externos necessitam acessar serviços da rede interna de uma empresa. A solução proposta para esse problema é o pattern Service Perimeter Guard.

O pattern Service Perimeter Guard propõe a criação de um serviço que atue como uma camada de interface, fazendo a comunicação entre o consumidor externo e o serviço interno, tornando-se o ponto central de comunicação do serviço com o exterior.

A forma mais comum de aplicação desse pattern é através de um serviço criado na DMZ (DeMilitarized Zone) da rede corporativa, acessando os recursos e serviços internos através de um firewall. Dessa forma, toda comunicação feita pelos consumidores é retransmitida para os serviços internos, e as suas respostas são retransmitidas para os consumidores também através desse serviço.

Um ponto muito importante desse pattern é que, por se tornar um ponto centralizador na execução dos serviços, alguns dos patterns citados anteriormente (Exception Shielding e Message Screening) podem ser aplicados dentro da lógica desse serviço, permitindo assim um ponto de integração entre esses patterns.

Impacto da aplicação do pattern no sistema

O uso desses serviços intermediários ou de perímetro, é que podem vir a se tornar gargalos no sistema, causando impacto na performance. Além disso, pelo fato de centralizarem os acessos externos dos serviços, tornam-se alvos primários de ataques maliciosos, o que torna indispensável que esses serviços tenham uma implementação extremamente robusta. Também não se pode deixar de ressaltar que a aplicação desse pattern não exclui a implementação de segurança nos serviços internos, principalmente na comunicação entre os serviços internos e os serviços de perímetro.

Implementações de segurança em SOA

Serão apresentadas agora diversas tecnologias de segurança para o desenvolvimento de web services, com enfoque na linguagem Java. Esta seção está dividida em: criptografia, autenticação e autorização.

Criptografia

Nesta seção, iremos apresentar alguns dos principais algoritmos de criptografia existentes. Primeiramente, no entanto, iremos analisar os dois tipos principais de algoritmos de criptografia existentes: criptografia por chave simétrica, e criptografia por chave pública, ou chave assimétrica.

Criptografia por chave simétrica

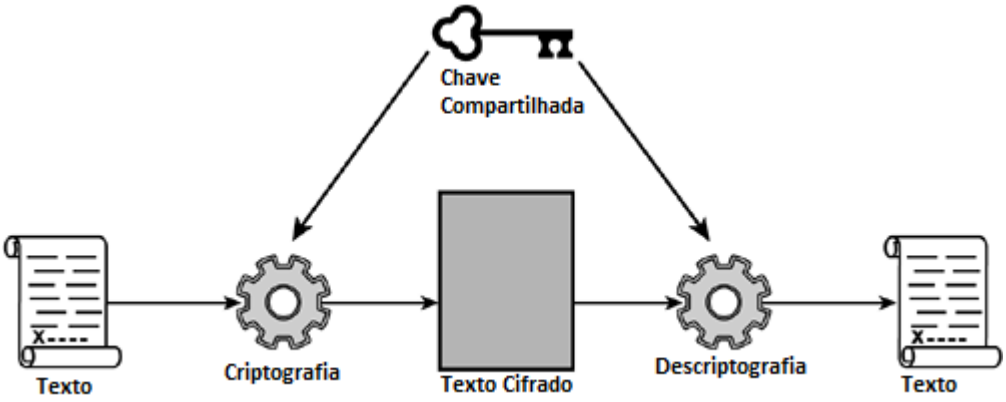

Figura 5 – Estrutura de execução da criptografia por chave simétrica

(Fonte: Adaptado de ROSENBERG, JOTHY; REMY, DAVID. **Securing Web Services with WS-Security Demystifying WS-Security, WS-Policy, SAML, XML Signature, and XML Encryption**, 1. ed. Sams Publishing.)

Nos algoritmos de criptografia por chave simétrica, tanto a função de criptografia quanto a de descriptografia são executadas com base na mesma chave. Nessa categoria de algoritmo, o algoritmo de criptografia é simetricamente oposto ao de descriptografia, ou seja, são executados exatamente os mesmos cálculos, porém a fim de obter o resultado oposto. Isso significa que uma mensagem cifrada por esse tipo de criptografia levará o mesmo tempo e esforço para ser criptografado ou descriptografado. A figura acima representa graficamente esse conceito.

Algoritmos de chave simétrica costumam ser encontrados como orientados a bloco (realizam a criptografia e descriptografia em blocos de texto, ao invés de com mensagens completas), ou como orientados a corrente (stream), onde a criptografia e descriptografia são realizadas byte a byte, conforme a mensagem é enviada ou recebida. Exemplos de algoritmos desse tipo são o DES e AES.

A principal limitação desse tipo de algoritmo consiste na falta de um mecanismo seguro para que o remetente e o destinatário das mensagens possam definir uma chave. Isto é, já que tanto o remetente quanto o destinatário necessitam negociar uma chave para operar, ela não pode ser enviada em texto puro através das comunicações de negociação para estabelecimento da comunicação segura, pois se algum programa malicioso estiver monitorando a comunicação naquele momento, ele poderá obter a chave.

Normalmente, são utilizadas comunicações “não-computacionais” (reuniões etc) entre as equipes de desenvolvimento a fim de formalizar uma chave para as comunicações, mas tais medidas não eliminam a limitação que esse tipo de criptografia apresenta.

Criptografia por chave pública (assimétrica)

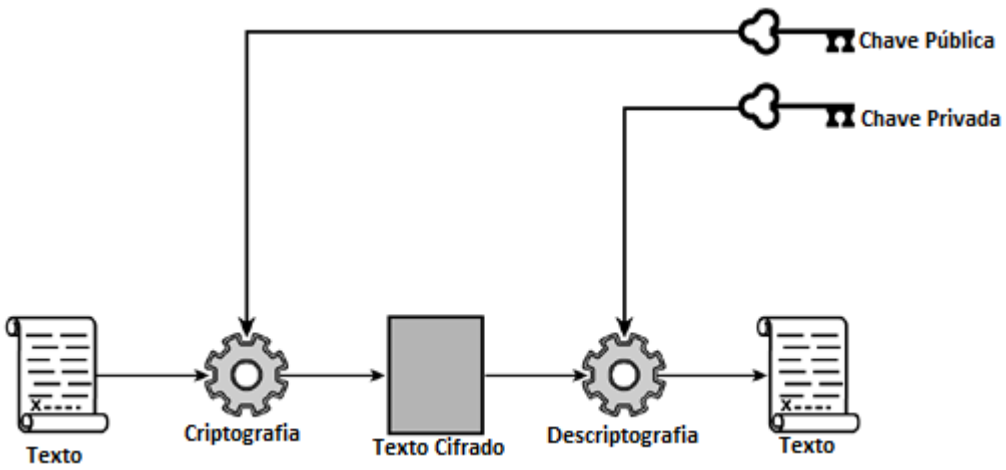

Figura 6 – Estrutura de execução da criptografia por chave pública
(Fonte: Adaptado de ROSENBERG, JOTHY; REMY, DAVID. **Securing Web Services with WS-Security Demystifying WS-Security, WS-Policy, SAML, XML Signature, and XML Encryption**, 1. ed. Sams Publishing.)

O principal objetivo na criação dos algoritmos de chave pública, foi precisamente resolver a principal limitação da criptografia por chave simétrica. Nos algoritmos de chave pública, é possível utilizar duas chaves diferentes, isto é, o remetente utilizar uma chave para criptografar, e o destinatário se utilizar de outra chave para realizar a descriptografação. Nesse modelo, cada aplicação possui a sua própria chave oculta (chamada chave privada), além de possuírem outra chave, chamada de chave pública, que utilizam para informar o destinatário que chave irão utilizar.

Quando um lado deseja enviar uma mensagem para o outro, o lado emissor a criptografa utilizando a sua chave pública, porém uma mensagem criptografada com a chave pública do lado emissor, somente poderá ser descriptografada com a chave privada do lado receptor. A figura acima ilustra esse conceito. Exemplos de algoritmos nesse formato são o SHA1 e RSA.

A principal limitação desse tipo de algoritmo é que, devido a sua maior complexidade, sua performance é bastante inferior em relação aos algoritmos de chave simétrica, portanto é recomendado que seja utilizado apenas para o tráfego de blocos menores de dados, nunca para enormes conjuntos de dados maciços sendo enviados de uma só vez.

Com as definições de tipos de criptografia tendo sido devidamente elucidadas, seguimos em frente agora com as descrições dos algoritmos citados anteriormente. Comecemos com os algoritmos do tipo chave pública.

RSA

O Algoritmo RSA (que teve seu nome criado a partir dos nomes de seus criadores, Ronald **R**ivest, Adi **S**hamir e Leonard **A**dleman) se utiliza de um número como input (o que obriga a mensagem a ser transformada de texto para binário), onde o número deve ser múltiplo do número escolhido pela implementação do algoritmo para ser o módulo do algoritmo. Por essa razão, esse algoritmo possui uma limitação de 128 bytes como tamanho máximo para os blocos de mensagem, para um algoritmo RSA de 1024 bits, por exemplo. O algoritmo RSA é o principal algoritmo de criptografia por chave pública sendo utilizado atualmente.

SHA1

O algoritmo SHA1 (também conhecido por Algoritmo de Hash Seguro) é o primeiro algoritmo de chave pública a ser criado que utiliza o conceito de geração de hash. O algoritmo SHA1 gera códigos hash no formato fixo de blocos de 20 bytes, e atende os três requisitos principais de um algoritmo de hash, que são:

- Resistência de colisão, ou seja, a sua geração de hash possui uma complexidade tal que, caso algum código malicioso tente decifrar uma mensagem codificada por ele pelo método brute force, por exemplo, (método do tipo "tentativa e erro", no qual o código tenta todas as possibilidades uma por uma até encontrar a correta) levará um tempo inviável para decifrar a mensagem. Para uma mensagem cujos blocos de 20 bytes totalizem uma mensagem de 160 bits, por exemplo, levaria 10 elevado a 22 bilhões de anos para que o atacante conseguisse decifrar a mensagem através do método brute force.
- Simplicidade, o algoritmo em si do SHA1 se utiliza de simples cálculos de fatoração, e por esse motivo pode ser utilizado em softwares que dependam de alta rapidez no atendimento das mensagens.
- Eficiência, por conta da resistência de colisão, pois as chances de que duas mensagens diferentes gerem o mesmo valor de hash, ou de que se possa descobrir o conteúdo de uma mensagem a partir de sua hash sem conhecer a implementação do algoritmo que a gerou, são de uma magnitude muito pequena.

Esses são os principais algoritmos de chave pública existentes. Agora, serão apresentados os principais de chave simétrica.

DES

O algoritmo DES (Data Encryption Standard), é o algoritmo padrão de criptografia XML. Esse algoritmo possui melhor performance quando executado via hardware ao invés de software, pois na verdade ele foi projetado especificamente com esse objetivo.

O algoritmo começa quebrando as mensagens que recebe em dois blocos de 32 bits cada, e em seguida realiza uma série de permutações, deslocamentos, substituições e operações lógicas de OU exclusivo, a fim de obter o seu valor criptografado final. A versão para descriptografia deste algoritmo executa exatamente os mesmos passos da criptografia, alterando apenas a ordem de execução.

AES

O algoritmo AES (Advanced Encryption Standard), surgiu com o objetivo de substituir o DES como algoritmo padrão de criptografia XML. O AES se utiliza de operações de transformação (substituição de bits) para realizar as suas operações, realizando a operação em 10, 12 e 14 rodadas, para chaves de tamanhos 128, 192 e de 256 bits, respectivamente. O algoritmo trabalha de modo a quebrar os blocos de dados em arrays, formando uma matriz, seguindo para operações de substituição e OU exclusivo sobre os dados da matriz. Assim como o DES, esse algoritmo também possui maior performance quando executado via hardware.

Assim termina a seção sobre criptografia. Na prática, o desenvolvedor não irá implementar estes algoritmos, e sim configurar a aplicação para utilizar o protocolo de certificação (dentro do qual está implementado o algoritmo) que deseja utilizar. Dando como exemplo dessa configuração no framework de segurança WS-Security (cuja extensão responsável por criptografia se chama WSSE), utilizando o servidor JBoss, temos dois arquivos descritivos para essa configuração (*jboss-wsse-server.xml* para o lado servidor, e *jboss-wsse-client.xml* para o lado cliente), esses arquivos por padrão configuram a aplicação para utilizar um protocolo chamado X.509, que se utiliza do framework PKI, que utiliza algoritmos de chave pública para realizar a criptografia. Continuando o estudo, segue-se agora a seção de autenticação.

Autenticação

Nesta seção, serão abordadas as principais tecnologias existentes para autenticação de usuários. O conceito de autenticação engloba o uso de credenciais, onde apenas os usuários com credenciais válidas podem acessar os serviços e/ou recursos. De acordo com o enfoque dado para a linguagem Java, iremos analisar como modelo para implementação o framework WS-Security, que foi lançado em conjunto pela Microsoft e IBM em 2002, e é sancionado pela organização OASIS, grupo responsável pela disseminação de padrões para o desenvolvimento de web services.

Antes de adentrar na forma como o WS-Security realiza a autenticação de usuários, é preciso definir o conceito do contrato de serviço (WSDL, Web Services Description Language, ou Linguagem de Descrição de Web Services), pois a autenticação de usuários atua diretamente na forma como esse contrato é montado.

O conceito do WSDL consiste de uma linguagem padrão XML, a partir da qual é gerado um documento onde estão definidas todas as operações, parâmetros de entrada e de saída que o serviço vinculado ao documento pode suportar. Pode-se dizer que o WSDL seja o

documento a partir do qual um serviço pode informar para os seus consumidores que tipos de ações ele pode realizar.

Todas as mensagens trafegadas entre um web service e seus consumidores se utilizam de um protocolo chamado SOAP (Simple Object Acess Protocol, ou Protocolo Simples de Acesso a Objetos), dentro do qual se encontram descritas as operações do web service.A figura abaixo mostra a forma com que os pacotes (também chamados de envelopes) SOAP são montados.

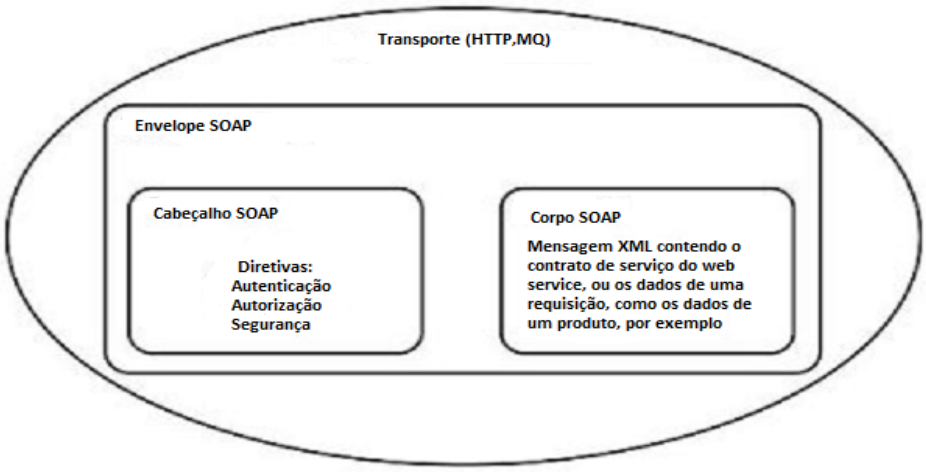

Figura 7 – Estrutura de um envelope SOAP

(Fonte: Adaptado de ROSENBERG, JOTHY; REMY, DAVID. **Securing Web Services with WS-Security Demystifying WS-Security, WS-Policy, SAML, XML Signature, and XML Encryption**, 1. ed. Sams Publishing.)

Como se pode ver na figura, as informações relacionadas com segurança se encontram no cabeçalho do envelope SOAP, cabeçalho que o servidor de aplicação onde o web service estiver instalado (no caso do Java, temos servidores como o JBoss e o Glassfish, que podem ser utilizados na implantação de web services) irá utilizar para obter o usuário e a senha que o consumidor fornecer, e irá conceder ou negar o acesso de acordo com o resultado da validação.

A seguir, será demonstrado de que forma essa implementação é feita nos dois lados, o lado do consumidor (lado cliente) e no lado do web service propriamente dito.Lembrando que o objetivo do artigo é apresentar uma visão inicial da tecnologia.

Implementação do lado cliente

Para realizar a passagem do usuário e da senha para o web service no lado cliente, existe um objeto responsável pelo binding (ligação) de diretivas para o cabeçalho do envelope SOAP, conforme foi explicado na figura 7. O nome desse objeto em Java se chama

BindingProvider, que é instanciado a partir do contexto do Proxy utilizado para acesso do consumidor com o web service. Segue agora a implementação para o lado servidor.

Implementação do lado servidor

No lado servidor da aplicação, a alteração a ser realizada é ainda mais simples. Basta inserir a anotação @WebContext, onde é definido o método de autenticação "BASIC". Essa anotação, com essa opção, informa para o servidor de aplicação onde o web service está instalado, que deve exigir dos consumidores do serviço que forneçam um usuário e uma senha para validação.

A forma como o servidor autentica os usuários foge ao escopo deste artigo, que enfoca mais em dar uma visão inicial para o leitor não familiarizado com as tecnologias de segurança, mas em nível de conhecimento pode ser dito que existe uma vasta gama de opções que variam de acordo com o servidor utilizado, como integração com o AD (Active Directory) ou mecanismos mais simples, como por exemplo, no caso do JBoss, onde os usuários são guardados dentro de um arquivo XML chamado *jbossws-users.xml*.

Outra forma de autenticação de usuários a ser utilizada é através do uso de certificação digital. Na certificação digital, o servidor exige que os clientes forneçam uma chave criptografada, também conhecida como certificado, para ser validada junto ao servidor. Seguindo o exemplo anterior, para alterar esse exemplo para o uso de certificação digital, seria mudado o atributo *authMethod* para "CONFIDENTIAL" no web service, e se utilizaria a ferramenta *Keytool* para a geração dos arquivos de certificação, que consistem de keystores (arquivos que constituem repositórios de chaves que podem ser enviadas) e truststores (arquivos que constituem repositórios de chaves consideradas como confiáveis caso sejam recebidas numa comunicação).

A fim de dar apenas uma visão mais conceitual das tecnologias, foram omitidos maiores detalhes dessa implementação, mas exemplos mais completos podem ser encontrados nos livros de referência deste artigo.Assim conclui-se a seção sobre Autenticação.

Autorização

O conceito de autorização de usuários envolve na criação e manutenção de grupos de acessos, ou perfis, de modo a restringir o acesso para as funcionalidades de acordo com um perfil de usuário. No WS-Security, existe uma extensão especificamente para essa finalidade, chamada WS-Authorization, onde a implementação dos perfis fica a cargo do servidor de aplicação, que deve definir os security realms (algo como "dimensões de segurança") da aplicação e inserir os grupos e usuários nesses security realms, onde os grupos passam a atuar

como perfis de usuário. Dessa forma, a implementação do mecanismo de autorização para o desenvolvedor final se torna uma tarefa trivial.

Tomando por exemplo o JBoss como servidor de aplicação de um web service, uma forma de realizar a implementação citada acima consiste em alterar os arquivos *jbossws-users.xml* e *jbossws-roles.xml*, onde estão guarda dos os usuários e perfis, respectivamente.A seguir, é preciso definir no servidor o security realm a ser utilizado, e por fim, deve se declarar o realm a ser utilizado na declaração do web service, utilizando a anotação *@SecurityDomain*.

Outra forma de implementar esse processo é através do padrão SAML (Security Assertion Markup Language, ou Linguagem de Marcação de Segurança Assertiva), que pode ser combinado com o framework WS-Security. Esse padrão, definido pelo grupo OASIS, ainda possui poucas ferramentas que o suportem no mercado, mas possui um formato de implementação que pode vir a se tornar um padrão no futuro, trabalhando a partir da definição de tags assertivas, que são inseridas dentro do cabeçalho do contrato de serviço do web service, e nos cabeçalhos dos pacotes de requisição SOAP, definindo os usuários e seus perfis.

Conclusão

Dessa forma conclui-se este artigo sobre segurança em SOA. No decorrer do artigo, foram mostrados padrões de design e tecnologias, que auxiliam na construção de aplicações orientadas a serviços com um maior reforço na sua segurança contra ataques. A principal característica que se pode notar, tanto no caso dos padrões, quanto das tecnologias, é a sua simplicidade para ser implementada pelo desenvolvedor final, como será elucidado a seguir.

Sobre os patterns (padrões) apresentados, pode ser visto que, a despeito da complexidade que envolve os detalhes das soluções de segurança em TI, soluções bastante simples podem ser adotadas a fim de melhorar a segurança, como o pattern Service Perimeter Guard, que consiste simplesmente da criação de uma nova camada intermediária para a chamada dos serviços, ou o pattern Message Screening, que nada mais é que uma simples rotina da validação de dados, tarefa esta que é cotidiana da maioria dos desenvolvedores que lidam com dados sensíveis. Isso demonstra que os padrões de design de segurança para SOA apresentados neste artigo possuem uma boa facilidade de implementação de um modo geral, cabendo apenas para a equipe desenvolvedora ter a maturidade de implementá-los de forma clara e organizada, a fim de facilitar as futuras manutenções.

Sobre as tecnologias apresentadas no artigo, pode-se afirmar que, assim como os patterns, também possuem uma implementação simples. Após o advento das anotações de

configuração na versão 5 do Java, em detrimento dos enormes arquivos XML descritivos de configuração (que ainda existem, porém em quantidade bastante inferior), tornou-se uma tarefa bastante simples para o desenvolvedor realizar a implementação de tarefas como Autenticação e Autorização de usuários.

É importante frisar, porém, que se deve dar atenção para a implementação nos dois lados da execução, tanto o consumidor quanto o servidor, pois tecnologias como a extensão WSSE do framework WS-Security, por exemplo, descrita na seção de criptografia, envolvem a configuração de ambos os lados, portanto é necessário que as equipes desenvolvedoras estejam com sua comunicação bem apurada. Também é importante frisar que a implementação dessas práticas não garante 100% de proteção contra ataques, apenas auxilia na robustez da segurança da aplicação, e que, portanto, não dispensa o uso de técnicas tradicionais de segurança, como regras de firewall, definições de DMZs, intranets corporativas etc.

Desse modo, foram apresentados os principais conceitos práticos e teóricos sobre segurança para SOA. Para se encontrar um exemplo prático das funcionalidades do WS-Security, recomenda-se o livro JBoss in Action, da editora Manning, que possui um exemplo completo de autenticação e autorização de web services através do uso do framework WS-Security.

Abstract

This Article has the objective of presenting and analyzing practices to be followed on the development of services, to strengthen the security of the system against attacks. It will be presented on the article security technologies on the development of web services (focusing the Java technology), to make an initial vision for the development of web services. It will be also presented concepts of design patterns of security on the development of services, focusing the patterns developed by Thomas Erl, considered one of the greatest authorities in SOA. At the end of this work, it will be made a general analysis of the patterns and technologies presented, emphasizing his main features.

keywords: Security; SOA; Design Pattern, Java.

REFERÊNCIAS

INSTITUTO BRASILEIRO DE TECNOLOGIA AVANÇADA. **Guia de apresentação de trabalho acadêmico**: relatório, trabalho de conclusão de curso (TCC), monografias, dissertações e teses. São Paulo: IBTA, 2005. 101f.

ASSOCIAÇÃO BRASILEIRA DE NORMAS TÉCNICAS – ABNT. NBR 6022. **Documentação. Artigo. Publicação periódica científica impressa – Apresentação** elaboração. Rio de janeiro, 2002 a.

ERL, THOMAS. **SOA Design Patterns**, 1. ed. Prentice Hall.

ERL, THOMAS. **SOA Principles of Service Design**, 1. ed. Prentice Hall.

GAMMA, ERICH; HELM, RICHARD; JOHNSON, RALPH; VLISSIDES, JOHN. **Design Patterns Elements of Reusable Object-Oriented Software**, 1. ed. Addison Wesley.

ROSENBERG, JOTHY; REMY, DAVID. **Securing Web Services with WS-Security Demystifying WS-Security, WS-Policy, SAML, XML Signature, and XML Encryption**, 1. ed. Sams Publishing.

KANNEGANTI, RAMARAO; CHODAVAPARU, PRASAD. **SOA Security**, 1. ed. Manning.

JAMAE, JAVID; JOHNSON, PETER. **JBOSS in Action,** 1. ed. Manning.

O'NEILL, MARK. **Web Services Security**, 1. ed. McGraw-Hill/Osborne.

E. AMOROSO, **Fundamentals of Computer Security Technology**, 1. ed. Prentice Hall.

ERL, THOMAS. SOA Patterns. Disponível em: <http://www.soapatterns.org/>. Acesso em: 01 agos. 2010.

ERL, THOMAS. What is SOA?. Disponível em: <http://www.whatissoa.com/>. Acesso em: 27 agos. 2010.

ERL, THOMAS. SOA Principles. Disponível em: <http://www.soaprinciples.com/>. Acesso em: 27 agos. 2010.

GARTNER. Gartner Says Efficient and Secure Enterprises Will Safely Reduce the Share of Security in their IT Budgets by 3 to 6 Percent of Overall IT Budgets Through 2011. Disponível em: <http://www.gartner.com/it/page.jsp?id=1384214>. Acesso em: 30 agos. 2010.

OASIS. Web Services Security: SOAP Message Security 1.0. Disponível em: < http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-soap-message-security-1.0.pdf>. Acesso em: 05 agos. 2010.

IBM. Web Services Security. Disponível em: < http://www.ibm.com/developerworks/library/specification/ws-secure/>. Acesso em: 05 agos. 2010.