UNIVERSIDADE FEDERAL DE SANTA CATARINA

WS SECURITY - ESTUDO E VERIFICAÇÃO DA QUALIDADE DE PROTEÇÃO EM WEB SERVICES

Gabriel Hanauer

UNIVERSIDADE FEDERAL DE SANTA CATARINA
DEPARTAMENTO DE INFORMÁTICA E ESTATÍSTICA
CURSO DE CIÊNCIAS DA COMPUTAÇÃO

RELATÓRIO SUBMETIDO COMO REQUISITO PARA A OBTENÇÃO DO GRAU DE BACHAREL EM CIÊNCIAS DA COMPUTAÇÃO

TÍTULO: WS Security – estudo e verificação da qualidade de proteção em web services
AUTOR: Gabriel Hanauer
ORIENTADOR: Prof. Dr. José Eduardo De Lucca
BANCA EXAMINADORA: Prof. Dr. Ricardo Felipe Custódio
Prof. Dr. Vitório Bruno Mazzola

PALAVRAS-CHAVE: web services, segurança, ws security

Florianópolis, 18 de junho de 2004

“Three Rings for the Elven-kings under the sky,
Seven for the Dwarf-lords in their halls of stone,
Nine for Mortal Men doomed to die,
One for the Dark Lord on his dark throne
In the Land of Mordor where the Shadows lie.
One Ring to rule them all, one Ring to find them
One Ring to bring them all and in the darkness bind them
In the Land of Mordor where the Shadows lie.”

J R R Tolkien

Agradecimentos

Aos meus bem-aventurados mestres, a quem a Providência Divina me deu a suprema graça de poder chamá-los pais José Canísio Hanauer e Inez Maria Hanauer.

A todo o apoio do dia-a-dia do meu irmão Felipe José Hanauer.

Ao meu grande amigo Dalvani Hanauer, por todas as horas gastas em conversas.

À Gabriela, que demonstrou uma paciência única ao meu lado, apoiando-me sempre com atenção e carinho.

Ao orientador José Eduardo De Lucca, que muito agradeço por aceitar esta empreitada com determinação até o fim.

E a todo aquele que, de uma forma ou outra, contribuiu com esta obra que a todos deixo.

Sumário

Índice de Figuras

Siglas Utilizadas no Trabalho

Assinatura Digital

Método para determinar a integridade dos dados de uma mensagem.

Autenticação

O processo pelo qual o web service determina a legitimidade para uma chamada de serviço.

Autorização

O tipo de acesso que um cliente ou usuário tem.

Certificados

Uma entidade digital que verifica a identidade de uma pessoa ou aplicação.

Cifragem

Método que modifica a mensagem de maneira tal que somente pessoas autorizadas podem lê-la.

HTTPS

HyperText Transfer Protocol –Secure.

IPSec

IP Security Protocol. Conjunto de especificações para cifrar, autenticar, manter a integridade e confidencialidade na camada do datagrama IP.

OASIS

Organization for the Advancement of Structured Information Standards.

SAML

Security Assertion Markup Language. Protocolo desenvolvido pelo W3C para assegurar a autenticação e autorização de informações.

SOAP

Simple Object Access Protocol. Protocolo baseado em XML usado para definir como as mensagens entre dois sistemas serão trocadas.

PKI

Public Key Infrastructure. Uso de chaves para garantir a privacidade e integridade dos dados na comunicação entre serviços.

Web Services

Protocolos e tecnologias usados por aplicações para se comunicar entre si através de tecnologias web como HTTP, HTTPS, SMTP, etc. Os protocolos usados incluem o XML, SOAP e WSDL. Normalmente um web service consiste de operações que podem ser chamadas, como métodos.

WSDL (Web Services Definition Language)

Protocolo baseado em XML que define o tipo de mensagem, protocolos de transporte e pontos de acesso numa rede de serviços.

WS Security

WS Security é uma especificação criada com o intuito de prover segurança nos web services.

XKMS (XML Key Management Specification)

Protocolo desenvolvido pelo W3C que descreve a distribuição e registro de chaves públicas.

XML (Extensible Markup Language)

Protocolo desenvolvido pelo W3C com a finalidade de descrever dados.

XML Encryption

Protocolo desenvolvido pelo W3C que descreve como devemos cifrar conteúdo digital.

XML Signature

Protocolo desenvolvido pelo W3C que descreve como devemos assinar conteúdo digital.

Resumo

Os web services tem evoluído muito lentamente, pois não oferecem segurança necessária. Dessa maneira, como podemos saber que um certo web service está relamente fazendo o que ele se propõe a fazer ou que ele é confiável? Muitas empresas tem desenvolvido suas próprias soluções para resolver esses problemas de segurança.

Porém, uma especificação chamada WS Security foi proposta. O principal objetivo dessa especificação é criar um padrão que mais tarde poderá ser usado pelos desenvolvedores e pela comunidade em geral.

O WS Security se propõe a resolver alguns dos problemas de segurança existentes, como integridade das mensagens, autenticação usando assinatura XML, cifrar usando XML, etc.

Nossa análise do protótipo mostra que somente o WS Security não oferece toda a segurança necessária. E também não oferece segurança total nas áreas propostas. O WS Security precisa ser complementado com outras especificações que estão surgindo e também com outros sistemas para garantir uma segurança adequada.

Abstract

It is commonly stated that Web Services does not provide security and therefore its journey has been very slow. For instance, how will you know that a certain Web service really does what it claims to do or that it comes from a trusted party? Companies
has been developing their own security solutions to overbuild this lack of security.

Therefore a specification titled WS Security has been proposed. The main goal has been to make a standard that the market will use.

WS Security claims to solve some of the existing security issues, such as message integrity, authentication by using XML signature, XML encryption and more.

Our analysis of the prototype shows that WS Security alone does not provide the needed security. Neither does it provide full security in the areas it targets. WS Security has to be complemented with other upcoming specifications and other systems in order to give adequate security.

1 Introdução

A internet cresceu rapidamente nos últimos anos, criando meios alternativos de comunicação e aumentando a disponibilidade de informação e serviços. Mas por trás desta disponibilidade existe uma demanda. Uma demanda por mais e melhores serviços.

O resultado dessa demanda criou um fenômeno chamado Web Services. Muitas empresas estão começando a usar e desenvolver Web Services.

Numa definição mais concreta de Web Services podemos dizer que

"Um Web Service é uma aplicação identificada por uma URI, onde suas interfaces são capazes de serem definidas, descritas e descobertas por artefatos XML, e suporta interações diretas com outras aplicações que usam mensagens baseadas em XML através de protocolos Internet." W3C Web Services Architecture Group

A partir dessa definição podemos nos fazer uma pergunta: porque esse fenômeno ainda não está sendo largamente usado? Esta pergunta pode ser respondida de várias maneiras. Uma delas é que não há consideração nenhuma à segurança nesses Web Services. E as soluções dadas até o momento não estão maduras o suficiente para serem aceitas no mercado.

Entretanto, algumas empresas como Microsoft, IBM e Sun criaram um comitê chamado OASIS[1]. OASIS criou uma especificação chamada WS Security com o objetivo de resolver alguns dos problemas de segurança dos Web Services. O objetivo é fazer da especificação WS Security um padrão, e também criar referências para aumentar a integridade, confidencialidade e autenticação das mensagens, obtendo assim melhor segurança. É importante lembrar que o WS Security não trata de todos os problemas de segurança. A especificação é apenas um guia. Entretanto, é interessante analisar esta especificação e sua qualidade de segurança de uma maneira prática e teórica. Este trabalho tem o objetivo de fazer esta análise através de um caso de estudo de Web Services baseado no WS Security.

[1] http://www.oasis-open.org

2 Objetivo

2.1 Tema

Neste trabalho iremos estudar e verificar a qualidade de proteção de web services implementados usando as especificações WS Security. Essas especificações definem o mecanismo básico para que haja uma troca segura de mensagens entre os serviços usando SOAP e que garantam qualidade de proteção na integridade, confidencialidade e autenticação dessas mensagens. WS Security também prove um mecanismo que associa tokens seguros às mensagens.

2.2 Objetivo Geral

Estudar e verificar a qualidade de proteção de serviços web implementados usando as especificações WS Security.

2.3 Objetivos Específicos

- Estudar e compreender as especificações WS Security
- Verificar a qualidade de proteção em web services implementados com WS Security
- Implementar serviços web usando WS Security

2.4 Justificativa

Segurança em Web Services vem sendo um dos assuntos mais comentados nos últimos meses na área de Web Services. Um dos motivos que está atrasando a aceitação e implementação desses serviços é a falta de um padrão de segurança.

Muitas empresas e analistas mais céticos estão questionando a segurança nesses serviços. Algumas questões que estão sendo feitas é como podemos proteger adequadamente esses serviços? De que forma estarão seguros os servidores e a rede interna que oferecem esses serviços?

Em abril de 2002 a Microsoft, IBM e Verisign publicaram uma nova especificação de segurança para Web Services, o WS Security. Essa especificação propõe um conjunto padrão de extensões SOAP que podem ser usadas para implementar integridade e conficencialidade no desenvolvimento de Web Services seguros.

E o motivo que nos leva a fazer um estudo mais aprofundado sobre a segurança e qualidade de proteção em Web Services é disponibilizar mais uma fonte de referência para analistas e empresas que pretendem fazer uso dessa nova tecnologia.

2.5 Objeto

2.5.1 Problema

O maior problema de os Web Services ainda não terem conquistado o mercado é a sua falta de segurança. Para solucionar esse problema desenvolvedores tem usado a segurança de camadas inferiores no processo de desenvolvimento, ou criando suas próprias soluções.

Não é muito interessante deixar que as empresas desenvolvam soluções de segurança única e exclusivamente para os seus problemas. Isto fere um dos motivos pelo qual os Web Services foram criados: a interoperabilidade. Por isso, é necessário que haja um padrão de desenvolvimento, um guia, em que os desenvolvedores possam se basear. Foi por esse motivo que o WS Security foi proposto. Mas como o WS Security é uma especificação e não um padrão, é necessário que sejam feitos testes e análises para podermos validá-lo.

2.5.2 Questões

Quais as falhas de segurança que o WS Security se propõe a resolver?

O WS Security resolve os problemas de segurança nas áreas propostas?

O processo recomendado pelo WS Security é suficiente para garantir a segurança nos Web Services?

3 Metodologia

3.1 Estudo de Caso

Para medir o nível de segurança proposto é importante que se use um sistema real. Usaremos um sistema que irá simular o acesso de um usuário em um sistema qualquer como ponto de partida para a implementação e estudo do WS Security.

3.2 Fonte de Informação

Como o WS Security é um termo novo, não há muita material nessa área, e o que existe não são documentos científicos. Dessa forma, a maior fonte de informação serão artigos e papers retirados da internet.

3.3 Estudo e aprofundamento

Para aprender mais sobre segurança em web services é indispensável o conhecimento de tecnologias que formam a base para a pesquisa: segurança em computação e web services. A linguagem escolhida para a implementação do protótipo é Java. A princípio não é necessário ter um grande conhecimento sobre Java para se entender a segurança em web services.

Como o WS Security é baseado em SOAP e XML, é necessário ter um bom conhecimento desses termos para entendermos o WS Security. O WS Security é uma tecnologia nova, não há livros e existem poucos documentos disponíveis na internet. Por isso nossa fonte primária de pesquisa será a especificação.

3.4 Análise, Modelagem e Implementação do Protótipo

A funcionalidade do web service não é importante, levando-se em conta que deve haver segurança mesmo não havendo funcionalidade. Dessa forma, a funcionalidade será limitada e o foco será todo na parte do WS Security.

O objetivo dessa etapa é encontrar requisitos mínimos para que o protótipo seja desenvolvido.

3.5 Teste e análise

O protótipo será analisado e testado para ver se realmente os requisitos de segurança foram alcançados.

O objetivo desta etapa é testar uma implementação do WS Security para verificar se existem problemas de segurança.

4 Estudo Inicial

Para entender a tecnologia que está por trás do WS Security é necessário que sejam explicadas algumas definições importantes. Esta parte do trabalho tem o objetivo de explicar o conceito de segurança e mostrar algumas técnicas de segurança usadas atualmente.

4.1 O que é um Web Service

Muitas pessoas e empresas têm discutido a definição exata de Web Services.

O W3C define web services dessa maneira:

"A Web service is a software system designed to support interoperable machine-to-machine interaction over a network. It has an interface described in a machine-processable format (specifically WSDL). Other systems interact with the Web service in a manner prescribed by its description using SOAP messages, typically conveyed using HTTP with an XML serialization in conjunction with other Web-related standards."[2]

Em outras palavras, pode-se dizer que web services é um pedaço de software que se torna disponível através da internet e usa um sistema padrão de mensagens XML. Os Web Services usam a web para realizar integração entre aplicações.

XML é usado para codificar a comunicação entre Web Services. Por exemplo, um cliente chama um web service enviando uma mensagem XML e espera uma resposta XML. Como toda a comunicação é XML, os web services não estão presos a sistema operacional ou linguagem de programação.

Existem muitas outras definições para web services. Criar uma definição genérica do que é um web service vem sendo umas das tarefas mais difíceis que o grupo de trabalho do W3C vem tendo em relação a web services.

4.2 Segurança

A segurança pode estar relacionada a software, hardware e também a um processo humano. A segurança está dividida em quatro áreas principais que englobam todos os seus aspectos. Estas áreas são:

- Confidencialidade – somente usuários autorizados tem acesso à informação
- Integridade – impede a modificação da informação, seja por usuários não autorizadas ou por falhas do sistema
- Acessibilidade – serviços não são interrompidos por falha de hardware, software ou por rotinas de manutenção
- Autenticação – impede a falsificação de identidade[3]

[2] http://www.w3.org/TR/ws-arch/#whatis
[3] http://www.tbc365.co.uk/tbc365/information/firewall/index.asp

4.3 A Segurança em Web Services com SSL/IPsec

Como os Web Services não tem uma solução adequada para segurança, eles dependem da segurança de outros sistemas e camadas. Por isso, SSL e IPsec se tornaram populares para garantir a segurança de Web Services.

O SSL usa o protocolo "Handshake" e certificados para fazer a autenticação do cliente e servidor[4]. O SSL só pode autenticar e cifrar a comunicação entre dois pontos. Mas as requisições e respostas de Web Services normalmente usam mais de uma rota para se comunicar. O SSL é uma solução adequada se conhecermos previamente o caminho que a mensagem irá percorrer. Outro ponto importante é que o SSL protege os dados somente durante a transmissão. No momento que os dados chegam ao seu destino é necessário fazer uso de outras técnicas para manter a segurança.

Outro problema das técnicas de SSL para Web Services é que a autenticação e cifragem do SSL consomem muito tempo de CPU e conseqüentemente o processo da transação se torna lento.

Uma das conseqüências das falhas de segurança para web services mencionadas acima, é que as mensagens são fáceis de serem copiadas.

IPsec é um formato seguro do IP. Ele consiste em duas partes: AH para a autenticação e ESP para a cifragem. O IPsec se propõe a prover autenticação, integridade e cifragem.[5]

Essas técnicas conferem integridade, assinatura e cifragem para a informação transportada pelas mensagens, mas elas falham num ponto importante: autenticação ponto-a-ponto e a possibilidade de cifrar e assinar somente alguns pontos da mensagem.

4.4 HTTPR

"*HTTPR is a protocol for the reliable transport of messages from one application program to another over the Internet, even in the presence of failures either of the network or the agents on either end*"[6]

O HTTP é um protocolo bastante inseguro, pois ele não garante que a informação transmitida chegue ao seu destino. Para corrigir esse problema foi criada uma versão mais segura do HTTP, chamado HTTPR. O HTTPR garante que todas as mensagens cheguem ao seu destino pelo menos uma vez e no seu formato original. Se a mensagem não chegar ao seu destino o HTTPR notifica quem a enviou dizendo que a mensagem não foi recebida.

---

[4] http://www.rsasecurity.com/rsalabs/faq/5-1-2.html
[5] http://www.rsasecurity.com/rsalabs/faq/5-1-4.html
[6] http://www-106.ibm.com/developerworks/webservices/library/ws-httprspec/

Para obter os mesmos benefícios do HTTPS, existe uma outra versão do HTTPR chamada HTTPSR.[7]

4.5 Bloqueio por IP

Uma técnica chamada bloqueio por IP pode ser usada para limitar um web service. Quando um usuário tenta conectar a um web service, o web service verifica o IP do usuário para ver se ele tem permissão para acessar o web service. O web service pode manter uma lista de endereços IP para fazer essa checagem.

Um dos problemas do bloqueio por IP é que somente os endereços que estão na lista de acesso podem acessar o web service. Usuários que não estão na lista não tem acesso ao web service e nem ao WSDL, que são informações adicionais do web service.

Outro problema dessa técnica é a administração dessa lista de acesso de endereços. Se o web service é muito usado, a lista de acesso pode crescer bastante e torna-se difícil de manter. Ainda existe o problema de "IP spoofing".[8]

4.6 Segurança XML (cifragem e assinatura XML e XKMS)

Através de assinaturas XML, pode-se assinar somente algumas partes de um documento XML. Com a cifragem XML pode-se cifrar partes de um documento XML simétrica e assimetricamente, isto é, pode-se cifrar um elemento específico.[9]

Assinatura e cifragem XML cobrem três falhas de segurança de um web service: autenticação, integridade e a garantia de que uma mensagem falsa não possa ser enviada.

4.6.1 Assinaturas XML

Assinaturas XML podem ser feitas de 3 maneiras:

- a informação assinada fica dentro da assinatura;
- a assinatura fica dentro da informação que está sendo assinada ;
- a informação assinada e a assinatura ficam em arquivos XML diferentes.

A assinatura XML contém informações sobre as técnicas de assinatura que estão sendo usadas, como por exemplo, hashing e algoritmos de cifragem.[10]

---

[7] http://www-106.ibm.com/developerworks/webservices/library/ws-phtt/
[8] http://builder.com.com/article.jhtml?id=u00320020415gcn01.htm
[9] http://www.vordel.com/knowledgebase/tutorial_xml_security/

Através dessa técnica é possível garantir que as informações não serão mudadas.

4.6.2 Cifragem XML

A cifragem XML garante que a informação não será lida por usuários desautorizados. Diferentemente do SSL, essa técnica garante que as informações estarão protegidas durante o transporte e também no computador do usuário.

Na cifragem XML podemos usar chaves simétricas ou assimétricas. As mais usadas são as simétricas, pois elas consomem menos CPU e são mais adequadas para a troca de grandes quantidades de informação.

4.6.3 XKMS

Para poder usar PKI, precisamos de um sistema que gerencia chaves e certificados. A versão XML para o gerenciamento de PKI é chamada XKMS (XML Key Manager Specification). XKMS é um guia para fazer a integração de chaves e certificados com as aplicações e também um guia para o registro, cancelamento e atualização de chaves. XKMS usa SOAP sobre uma rede HTTP.[11]

O XKMS consiste em três partes: X-KISS (XML Key Information Service Specification), X-KRSS (XML Key Registration Service Specification) e a especificação do Protocolo.

Para obter uma boa segurança esses três padrões devem ser usados em conjunto. Uma cifragem XML precisa usar assinaturas para garantir a autenticidade de quem está enviando as informações. E a assinatura XML precisa usar XKMS para gerenciar as chaves que estão sendo trocadas.

Um ponto importante a ser considerado com respeito a assinaturas e cifragem XML é que essas técnicas são novas e ainda não existe um protótipo implementado com elas.

4.7 SAML e XACML

SAML (Security Assertion Markup Language) é um padrão que faz recomendações de como as informações de segurança devem ser trocadas, usando a internet. Similar ao WS Security, o SAML também é uma especificação da OASIS.[12]

---

[10] XML and Security, XML journal, December 8, 2001, Mark O'Neill
[11] http://www.xmltrustcenter.org/xkms/faq.htm
[12] http://www.vordel.com/knowledgebase/tutorial_xml_security/XML4.html

SAML é um guia para prover autenticação e autorização no pedido e resposta de mensagens. SAML mostra como a assinatura pode ser feita apenas uma vez quando vários web services estão interagindo. Isso significa que um web service não precisa ser autenticado toda vez que ele precisa de um web service adicional. Ele pode se autenticar através de outro web service, e esse por sua vez diz que o web service está autenticado quando ele requisitar outro web service.[13]

O SAML, como o WS Security usa técnicas como assinatura e cifragem XML.
Outro termo relacionado a SAML e XML é XACML (eXtensible Access Control Markup Language). XACML é um conjunto de regras de como a autorização deve ser feita através da internet. Uma definição de XACML é *"XACML definirá as regras que especificam de quem, o que, quando e como a informação será acessada"*.[14]

O XACML pode ser considerado um complemento do SAML. Quando um web service acha um elemento SAML em um documento XML, ele processa a requisição checando as regras do seu XACML através do PRP (Policy Retrieval PDP).[15]

4.8 Problemas de Segurança no SOAP e SOAP-Sec

"SOAP, the Simple Object Access Protocol, is XML syntax for exchanging messages.
Because it is XML, it is both language and platform independent"[16]

O SOAP usa a mesma porta que o HTTP, e é um protocolo comum usado para a troca de informação entre redes. O SOAP não foi desenvolvido considerando se as falhas de segurança de web services.

Como as mensagens SOAP consistem de código XML, qualquer pessoa que pegar uma mensagem poderá ver as informações que estão sendo trocadas. E algumas dessas informações podem ser importantes como, por exemplo, senhas. Desse modo, é muito importante que a informação seja cifrada.

Outro problema com as mensagens SOAP é que elas são transmitidas num único sentido. A conseqüência disso é que elas são fáceis de serem roubadas e reenviadas.

Para corrigir alguns dos problemas de segurança do SOAP, uma nova técnica foi proposta pela IBM e Microsoft, chamada SOAP-Sec. SOAP-Sec permite que mensagens sejam assinadas adicionando para isso um cabeçalho ao SOAP.

---

[13] http://www.nwfusion.com/news/tech/2002/0701tech.html
[14] Simon Y. Blackwell of Psoom, chair of the OASIS XACML Technical Committee
[15] http://www.vordel.com/news/articles/02-03-08.html
[16] http://www.vbxml.com/soapworkshop/articles/intro/default.asp

4.9 WS Security

A especificação WS Security começou a ser desenvolvida por três grandes empresas: IBM, Microsoft e Verisign. O WS Security foi criado para corrigir as falhas de segurança que existem nos Web Services e também para ser um padrão na troca e assinatura de mensagens seguras.

O WS Security não resolve todos os problemas de segurança e também não prove um modelo específico de como um Web Service deve ser desenvolvido. Ele oferece somente um caminho a ser seguido.É importante perceber que o WS Security é apenas uma especificação. Uma especificação focada nas extensões SOAP.

Essas extensões provêem autenticação, integridade, confidencialidade e assinatura para as mensagens. O WS Security está focado na autenticação e na cifragem.

O WS Security não está limitado a um modelo ou mecanismo específico. Ele suporte vários modelos e mecanismos de segurança. Por exemplo, um desenvolverdor pode usar tokens de software como também tokens de hardware.

Entretanto, o WS Security possui alguns requisitos de sistema, que são:

- a linguagem do web service deve ter suporte a múltiplos tokens de segurança;
- várias tecnologias de criptografia;
- segurança entre o transmissor e o receptor;
- segurança no transporte.

O WS Security preenche esses requisitos fazendo uso da assinatura e cifragem XML, e também de outras técnicas. Pode-se complementar a segurança com SSL e outras técnicas parecidas.[17]

O WS Security é considerado um substituto do SOAP-Sec.[18]

4.10 Futuros Padrões de Segurança

O WS Security foi a base para várias outras especificações. Essas especificações são: WS-Policy, WS-Trust, WS-Privacy, WS-Secure Conversation, WS-Federation e WS-Authorization.

Todas essas especificações juntas tem o objetivo de preencher todas os problemas de segurança conhecidos nos Web Services.

---

[17] http://www.webservicesarchitect.com/content/articles/apshankar04.asp
[18] http://www.vordel.com/news/articles/02-03-08.html

4.10.1 WS-Policy

No WS Security a escolha dos produtos de desenvolvimento é livre. Essa política, similarmente ao WS Security, não especifica os requerimentos do vendedor. Ela é um guia para os requerimentos mínimos que o sistema deve suportar. Esses requerimentos mínimos podem ser, por exemplo, algoritmos de criptografia. Essa especificação também será baseada no SOAP.

4.10.2 WS-Trust

O WS-Trust é um guia de como criar uma comunicação confiável. Essa comunicação pode ser entre dois usuários, e também com um terceiro usuário, que irá fazer um trabalho intermediário. Essa especificação será um guia de como deixar uma comunicação confiável usando tokens. Ela também define como uma distribuição segura desses tokens pode ser feita, usando WS-Security e outras técnicas já existentes.

4.10.3 WS-Privacy

O intuito dessa especificação é para se ter certeza de que o requisitante de um web service aceite e entenda as políticas e regras do web service. Com a ajuda e uso das três especificações acima podemos estabelecer uma política de segurança para um web service. Essa política de segurança pode conter informações de como o WS-Security e o WS-Policy deveriam ser usados para garantir uma comunicação confiável. O WS-Privacy também descreve como usar o WS-Trust para estimar as políticas de privacidade.

4.10.4 WS-SecureConversation

Para saber se um usuário de um web service possui as permissões corretas para usar esse web service o requisitante precisa se autenticar nesse web service. Por isso, o web service precisa provar que ele é o serviço requisitado. O WS-SecureConversation é um guia de como se guardar as informações de chaves de sessão, chaves derivadas, etc. Ele também é um guia de como essas chaves podem ser trocadas de uma maneira segura. Essa especificação também é baseada no WS-Security, no WS-Trust e SOAP.

4.10.5 WS-Federation

O objetivo dessa especificação é de ser um guia de como criar relações confiáveis baseadas nas especificações mencionadas anteriormente. Essa especificação também serve como um guia de como gerenciar uma comunicação segura.

4.10.6 WS-Authorization

Essa especificação descreve como uma autorização deve ser feita. Por exemplo, como um usuário deve fazer um pedido a um web service para ser validado, como um processo responde ao requisitante. Para manter a independência que os Web Services propõe, essa especificação deverá ser muito ampla no que diz respeito a formatos específicos de autorização e linguagens.

Padrão	**Descrição**	**Função de Segurança**	**Descrição Adicional**
Secure Sockets Layer (SSL) 3.0	Provê uma conexão privada e autenticada entre dois pontos	Cifragem	Sendo muito usada hoje. Provavelmente será substituída por TLS
Transport Layer Security (TLS) 1.0	SSL 3.0 com o suporte à tecnologia proprietária removida	Cifragem	Eventualmente poderá substituir o SSL 3.0.
Security Assertion Markup Language (SAML)	Framework XML usado para a troca, autenticação e autorização de informações	Autenticação Autorização	Não será muito usada com Web Services
XML Encryption	Especifica uma sintaxe de cifragem para o XML e o processo para se cifrar parte ou todo o documento XML	Cifragem	Pré-requisito para o WS-Security. XML Encryption previne p conhecimento do conteúdo de documentos XML por pessoas

			desautorizadas, mas ele não faz nada a respeito da autenticação da mensagem.
XML Signature	Especifica regras para assinar um documento XML digitalmente e para processar assinaturas	Autenticação	Pré-requisito para o WS-Security. Uma assinatura digital sozinha não pode provar que uma mensagem está autenticada.
XML Key Management (XKMS)	Especifica um método para que clientes baseados em XML obtenham chaves criptografadas com segurança	Autenticação (troca de chaves)	Pré-requisito para o WS-Security. XML Signature, XML Encryption e a sua combinação no WS-Security requerem chaves criptografadas. O XKMS se preocupa em como obter essas chaves.
WS-Security	Especifica o uso do XML Signature e XML Encryption nos cabeçalhos SOAP.	Autenticação e Cifragem	Teve um bom começo. IBM, Microsoft e Verisign aprenderam com problemas de segurança que tiveram no passado e usaram padrões de segurança XML existentes para tentar

			tornar mensagens SOAP seguras.
WS-Policy WS-Trust WS-Privacy WS-SecureConversation WS-Federation WS-Authorization	Em projeto	TBD	Essas especificações ainda estão somente no projeto. Tem a finalidade de melhorar a segurança dos Web Services no futuro.

- Padrão ainda não submetido
- Padrão já submetido
- Padrão final

4.11 JAX-RPC x SAAJ

O JAX-RPC é uma API para desenvolvimento e uso de web services. Um web service baseado em RPC é uma coleção de procedimentos que podem ser invocados remotamente por um cliente através da internet. No JAX-RPC, uma chamada a procedimento remoto é representada por um protocolo baseado em XML, como por exemplo SOAP. A especificação SOAP define a estrutura do envelope, regras de ciframento e convenções que servem para representar chamadas a procedimentos remotos e suas respostas. Essas chamadas e respostas são transmitidas através de mensagens SOAP (arquivos XML) através do HTTP.

O JAX-RPC esconde a complexidade das mensagens SOAP do desenvolvedor. Dessa maneira, o desenvolvedor especifica procedimentos remotos definindo métodos através de uma interface escrita em Java. O desenvolvedor também escreve uma ou mais classes que implementam esses métodos. Com o JAX-RPC, o desenvolvedor não gera ou interpreta mensagens SOAP. É o sistema JAX-RPC que em tempo de execução converte as chamadas e respostas à API em mensagens SOAP.

O SAAJ (SOAP with Attachments API for Java) prove uma maneira padrão para se enviar documentos XML através da internet usando a plataforma Java. Ele é baseado no SOAP 1.1 e SOAP com especificações anexas, que define um framework básico para a troca de mensagens XML.

O SAAJ é usado pelo mecanismo de mensagens SOAP que está por trás das implementações JAX-RPC e JAXR. Além do mais, é uma API que pode ser usada por desenvolvedores que desejam

escrever aplicações que usam mensagens SOAP diretamente, em vez de usar JAX-RPC. A API SAAJ permite que se crie mensagens XML usando a plataforma Java. Com a API SAAJ podemos criar, enviar e consumir mensagens XML através da internet simplesmente fazendo chamadas a métodos da API.

A especificação SAAJ 1.2 define o pacote javax.xml.soap, que contém a API para criar e popular mensagens SOAP.

4.12 Java Web Services Developer Pack

O Java Web Services Developer Pack (Java WSDP) é uma ferramenta que propicia desenvolvedores Java a criar e testar aplicações XML, Web Services e aplicações Web. Algumas das tecnologias incorporadas ao WSDP são: Java APIs para XML, Java Architecture para XML Binding (JAXB), JavaServer Faces, Web Services Interoperability Sample Application, XML e Web Services Security, JavaServer Pages Standard Tag Library (JSTL), Java WSDP Registry Server, Ant Build Tool, e Apache Tomcat container.

4.13 Apache Ant

O Apache Ant é uma ferramenta para criar aplicações Java. Na verdade, é uma espécie de make, sem todos os problemas que o make possui. Em vez de um modelo baseado em comandos shell, Ant é extendido usando classes Java. Em vez de escrever comandos shell, os arquivos de configuração são baseados em XML.

5 Análise, Modelagem e Implementação

O passo inicial foi a análise e modelagem dos dados e informações do protótipo.

A seguir entraremos na parte que nos interessa: a implementação de web services para verificar a autenticação e cifragem usando o WS Security.

Vamos dividir nosso objeto de estudo em 3 cenários diferentes:

1. O cliente envia uma mensagem com um token (esse token é o nome do usuário e a senha no formato de texto puro). O servidor verifica a existência do usuário e manda uma mensagem de volta ao cliente sem nenhum elemento de segurança.

2. O cliente envia uma mensagem com um token cifrado (o token é cifrado usando o serviço de identificação por chave pública). O servidor decifra o token, verifica a existência do usuário e manda uma mensagem ao cliente sem nenhum elemento de segurança.
3. O cliente assina (usando uma chave privada para a identificação do cliente) e cifra o corpo da mensagem (usando o serviço de identificação por chave pública). O cliente ainda adiciona um timestamp no cabeçalho. O servidor decifra o corpo da mensagem, verifica a assinatura e manda uma mensagem de volta ao cliente – o corpo da mensagem que é enviada de volta ao cliente é assinada com o serviço de identificação por chave privada e então é cifrada usando-se a chave pública do cliente que enviou a mensagem ao servidor.

Para que a assinatura e a cifragem sejam possíveis o cliente e o serviço devem possuir o par de chaves pública e privada e devem possuir o certificado digital um do outro nas suas bases de chaves.

Para entendermos melhor o funcionamento do protótipo detalhamos a seguir as principais classes usadas. As classes não estão descritas inteiramente. Apenas o conteúdo essencial para o entendimento do leitor. Para ver as classes na íntegra verificar anexo.

5.1 Classes

5.1.1 Servidor

5.1.1.1 WsSecurityService

Classe responsável pela criação do web service. É nessa classe que estão os serviços que vão ser disponibilizados para o cliente. A seguir mostraremos as passagens mais importantes dessa classe e uma breve explicação.

5.1.1.1.1 WsSecurityService.Java

Linhas	WsSecurityService.Java
18	public class WsSecurityService {
19	
20	/**

```
21      * Cenario 1: recebe o username token; nao ha elemento seguro na resposta.
22      * @param text text
23      * @return text
24      */
25     public String usernameToken(String text) {
26
27        // criando a configuração da mensagem
28        Configurable wsConf = Configurator.newRuntimeConfigurable();
29        MessageConf msgConf = (MessageConf) wsConf.narrow(MessageConf.class);
30        msgConf.setNoSecurityHeader(Boolean.TRUE);
31
32        // armazenando as configurações da mensagem no contexto de chamada
33        Current.getCallContext().getContextData().put(Constants.CD_MESSAGE_CONF,
34                                              msgConf);
35
36        return text;
37     }
38
39     /**
40      * Cenario 2: recebe o username token cifrado; nao ha elemento seguro na resposta.
41      * @param text text
42      * @return text
43      */
44     public String encryptedUsernameToken(String text) {
45
46        // criando a configuração da mensagem
47        Configurable wsConf = Configurator.newRuntimeConfigurable();
48        MessageConf msgConf = (MessageConf) wsConf.narrow(MessageConf.class);
49        msgConf.setNoSecurityHeader(Boolean.TRUE);
50
51        // armazenando as configurações da mensagem no contexto de chamada
52        Current.getCallContext().getContextData().put(Constants.CD_MESSAGE_CONF,
53                                              msgConf);
54
55        return text;
56     }
57
58     /**
       * Cenario 3: recebe o corpo assinado e cifrado; o corpo da mensagem de reposta e assinado e
59  cifrado.
```

```
60      * @param text text
61      * @return text
62      */
63     public String signedEncryptedBody(String text) {
64
65         // criando a configuração da mensagem
66         Configurable wsConf = Configurator.newRuntimeConfigurable();
67         MessageConf msgConf = (MessageConf) wsConf.narrow(MessageConf.class);
68
69         // criando um token binario para assinar a reposta (service identity private key)
70         SecurityTokenConf signingSecurityTokenConf = msgConf.newSecurityToken();
71         signingSecurityTokenConf.setType(Constants.ST_VALUE_TYPE_X509V3);
72         signingSecurityTokenConf.setWsuId("SigningSecurityToken-1");
73         signingSecurityTokenConf.setOrder(new Integer(10));
74
75         // criando a assinatura...
76         SignatureConf signatureConf = msgConf.newSignature();
77         signatureConf.setSignBody(Boolean.TRUE);
78         KeyInfoConf keyInfoConf = signatureConf.newKeyInfo();
79         keyInfoConf.setSecurityTokenMode(Constants.STM_REFERENCE);
80         keyInfoConf.setSecurityTokenId("SigningSecurityToken-1");
81         signatureConf.setKeyInfo(keyInfoConf);
82         signatureConf.setOrder(new Integer(0));
83         signatureConf.setWsuId("Signature-1");
84         signatureConf.setSignatureMethod(Constants.ALGO_ID_SIGNATURE_RSA);
85
86         // criando os dados cifrados
87         EncryptedDataConf encryptedDataConf = msgConf.newEncryptedData();

88  encryptedDataConf.setEncryptionMethodAlgorithm(Constants.ALGO_ID_BLOCKCIPHER_TRIPLEDES);
89         encryptedDataConf.setEncryptBody(new Boolean(true));
90         encryptedDataConf.setEncryptElementContent(new Boolean(true));
91         encryptedDataConf.setWsuId("EncryptedData-1");
92         encryptedDataConf.setOrder(new Integer(20));
93
94         // criando a chave cifrada
95         EncryptedKeyConf encryptedKeyConf = msgConf.newEncryptedKey();

96  encryptedKeyConf.setEncryptionMethodAlgorithm(Constants.ALGO_ID_KEYTRANSPORT_RSA15);
97         keyInfoConf = encryptedKeyConf.newKeyInfo();
```

```
 98        keyInfoConf.setSecurityTokenMode(Constants.STM_KEYIDENTIFIER);
 99        keyInfoConf.setSecurityTokenId(Scenario3IncomingValidator.EXT_TOKEN_CLIENT);
100        encryptedKeyConf.setKeyInfo(keyInfoConf);
101        encryptedKeyConf.setWsuId("EncryptedKey-1");
102        encryptedKeyConf.setOrder(new Integer(25));
103
104        // criando a referência para a cifragem
105        EncryptionReferenceConf encryptionReferenceConf = encryptedKeyConf.newReference();
106        encryptionReferenceConf.setRefUri("#" + encryptedDataConf.getWsuId());
107        encryptedKeyConf.setReferences(new EncryptionReferenceConf[]{encryptionReferenceConf});
108
109        // setando as configurações da mensagem
110        msgConf.setSignatures(new SignatureConf[]{signatureConf});
111        msgConf.setEncryptedData(new EncryptedDataConf[]{encryptedDataConf});
112        msgConf.setEncryptedKeys(new EncryptedKeyConf[]{encryptedKeyConf});
113        msgConf.setSecurityTokens(new SecurityTokenConf[]{signingSecurityTokenConf});
114
115        // armazenando as configurações da mensagem no contexto de chamada
116        Current.getCallContext().getContextData().put(Constants.CD_MESSAGE_CONF,
117                                                      msgConf);
118
119        return text;
120    }
```

A seguir vamos comentar essa parte do arquivo WsSecurityService.java:

Linhas	WsSecurityService.java
25 a 37	Método usado no cenário 1. São criadas as configurações da mensagem e essas são armazenadas no contexto da chamada.
44 a 56	Método usado no cenário 2. Da mesma maneira que no cenário 1 são criadas as configurações da mensagem e essas são armazenadas no contexto da chamada.
63 a 120	Método usado no cenário 2. Vamos detalhar este método um pouco melhor a seguir.
70 a 73	Criação do token que vai ser usado para assinar a resposta que o web service vai enviar ao cliente. Esse token é um certificado do tipo X509 versão 3. Esse certificado é gerado na hora da compilação da classe através do arquivo run.bat.
76 a 84	Criação da assinatura que vai ser usada para assinar o corpo da mensagem.
87 a 92	Especificando como será cifrada essa mensagem de resposta no cenário 3. O método para cifrar o corpo da mensagem foi o TripleDES.

95 a 102	Nesta parte do método do cenário 3 estamos criando a chave que será usada para cifrar os dados. Essa chave é criada usando o método RSA versão 1.5.
110 a 117	São criadas as configurações da mensagem e essas são armazenadas no contexto da chamada.

5.1.1.2 WsSecurityServer

Classe que cria o servidor WASP com um serviço publicado em 3 pontos com configurações diferentes.

5.1.1.2.1 WsSecurityServer.Java

Linhas	WsSecurityServer.java
22	public class WsSecurityServer {
23	private static String servicePath = "/teste/security/ws-security/WsSecurityService";
24	private static final String alias = "WsSecurityTesteService";
25	private static final String password = "changeit";
26	
27	private static void setupEndpointContextSecurity(ServiceEndpoint serviceEndpoint) {
28	
29	// pegando o contexto
30	ServiceEndpointContext ctx = serviceEndpoint.getContext();
31	
32	// criando a configuração de segurança
33	Configurable configurable = Configurator.newRuntimeConfigurable();
34	WSSEConf securityConf = (WSSEConf) configurable.narrow(WSSEConf.class);
35	
36	// aceitando username tokens
37	securityConf.setNoUsernameTokenValidation(Boolean.TRUE);
38	
39	// criando o token para a decifrar os dados
40	ExternalSecurityTokensConf extSecToken = securityConf.newExternalSecurityTokens();
41	SecurityTokenConf etokenConf = extSecToken.newSecurityToken();
42	etokenConf.setType(Constants.ST_VALUE_TYPE_X509V3);
43	extSecToken.setSecurityTokens(new SecurityTokenConf[]{etokenConf});
44	securityConf.setExternalSecurityTokens(extSecToken);
45	
46	// adicionando a classe para validar o token
47	securityConf.setValidatorClassName(Scenario3IncomingValidator.class.getName());
48	
49	// setando as configurações de segurança do endpoint

```
50          ctx.getContextData().put(Constants.CD_SECURITY_CONFIGURATION,
51                                  securityConf);
52
53          // setando as configurações do provider e da autenticação do serviço
54          try {
55              Credentials credentials = WaspSecurity.acquireServerCredentials(alias, password,
56                                                                  Constants.PROVIDER_NAME);
57              WaspSecurity.setAcceptingProviders(ctx, new String[]{Constants.PROVIDER_NAME});
58              WaspSecurity.setCredentials(ctx, new Credentials[]{credentials});
59          } catch (NoSuchProviderException e) {
60              throw new RuntimeWrappedException("Essa excecao nunca deveria acontecer!", e);
61          }
62      }
63
64      public static void main(String[] args) throws Exception {
65          String serverURL = System.getProperty("systinet.teste.server.url", "http://localhost:6060");
66
67          // inicializando o servidor WASP
68          Wasp.startServer(serverURL);
69
70          // criando uma instancia do serviço e publicando o endpoint
71          WsSecurityService svc = new WsSecurityService();
72          ServiceEndpoint endpoint = ServiceEndpoint.create(servicePath, svc);
73          setupEndpointContextSecurity(endpoint);
74          Registry.publish(endpoint);
75
76          // serviço publicado!
77          System.out.println("Servico publicado em: " + endpoint.getPath());
78      }
79  }
```

A seguir vamos comentar essa parte do arquivo WsSecurityServer.java:

Linhas	WsSecurityServer.java
23	Declaração do caminho do serviço web.
24	Declaração do apelido do serviço web.
25	Declaração da senha para acesso ao serviço.
27 a 62	Método usado para a configuração do endpoint do serviço web. Detalhes a seguir.
33 a 34	Criando a configuração de segurança do web service usando WS-Security.

40 a 44	Criação do token que será usando para decifrar os dados. Esse token é um certificado do tipo X509 versão 3. Esse certificado é gerado na hora da compilação da classe através do arquivo run.bat.
54 a 61	É nesta parte do código que é configurado a autenticação do serviço e suas credenciais.
64 a 78	Método main que é usado para iniciar o servidor, criar uma instância do serviço web e publicá-lo no endpoint determinado.

5.1.1.3 Scenario3IncomingValidator

Classe para validar os tokens do cenário 3.

5.1.1.3.1 Scenario3IncomingValidator.Java

Linhas Scenario3IncomingValidator.java

```
16 /**
17  * Classe para validar os tokens do cenário 3.
18  */
19 public class Scenario3IncomingValidator implements IncomingValidator {
20     /** token de referencia para o servico */
21     public static final String EXT_TOKEN_CLIENT = "EncryptingBinaryToken-1";
22
23     public void validate(MessageConf wsSecIncomingMessageConf) throws WSSecurityException {
24         // configuração do token x509
25         SecurityTokenConf peerTokenConf = null;
26
27         // temos que achar o token que foi usado na assinatura...
28         SignatureConf[] signatures = wsSecIncomingMessageConf.getSignatures();
29         if (signatures == null) {
30             return;
31         }
32         String signingTokenId = signatures[0].getKeyInfo().getSecurityTokenId();
33         SecurityTokenConf[] tokensConf = wsSecIncomingMessageConf.getSecurityTokens();
34         for (int i = 0; i < tokensConf.length; i++) {
35             if (tokensConf[i].getWsuId().equals(signingTokenId)) {
36                 peerTokenConf = tokensConf[i];
37                 break;
38             }
39         }
40
```

```
41          // adicionando um token externo para cifrar a resposta
42          Configurable configurable = Configurator.newRuntimeConfigurable();
43          WSSEConf securityConf = (WSSEConf)configurable.narrow(WSSEConf.class);
44          ExternalSecurityTokensConf extSecToken = securityConf.newExternalSecurityTokens();
45          SecurityTokenConf etokenConf = peerTokenConf;
46          etokenConf.setOrder(new Integer(2));
47          etokenConf.setType(Constants.ST_VALUE_TYPE_X509V3);
48          etokenConf.setWsuId(EXT_TOKEN_CLIENT);
49          extSecToken.setSecurityTokens(new SecurityTokenConf[]{etokenConf});
50          securityConf.setExternalSecurityTokens(extSecToken);
51          CallContext ctx = Current.getCallContext();
52          ctx.getContextData().put(Constants.CD_SECURITY_CONFIGURATION, securityConf);
53      }
54  }
```

A seguir vamos comentar essa parte do arquivo Scenario3IncomingValidator.java:

Linhas	Scenario3IncomingValidator.Java
21	Declaração do token externo usado pelo cliente
23 a 53	Método usado para validar os tokens. Detalhes abaixo.
28	Pegando a assinatura na mensagem SOAP.
32	Pegando o token de segurança da assinatura.
33	Pegando os tokens de segurança da mensagem recebida.
34 a 39	Verificando se algum dos tokens recebidos na mensagem é igual ao token usado na assinatura da mensagem.
41 a 52	Adicionando o token externo para cifrar a resposta que vai ser enviada ao cliente, usando para isso o identificador declarado anteriormente. O tipo do token novamente é X509v3.

5.1.2 Cliente

5.1.2.1 WsSecurityClient

Classe que faz a simulação do cliente que tem a função de chamar os serviços web publicados no servidor.

5.1.2.1.1 WsSecurityClient.java

Linhas WsSecurityClient.java

```
20 /**
21  * Cliente WS-Security.
22  */
23 public class WsSecurityClient {
24
25     /** caminho do endpoint */
26     private static String serviceWSDLPath = "/teste/security/ws-security/WsSecurityService/wsdl";
27
28     /** URL */
29     private static String serverURL = null;
30
31     /** serviço do cliente */
32     private static ServiceClient serviceClient = null;
33
34     /** proxy */
35     private static WsSecurityService serviceSoap = null;
36
37     private static final String ENCRYPTING_TOKEN_ID = "EncryptingToken-1";
38
39     /**
40      * Cenário 1: o cliente envia o username token com uma senha em plaintext.
41      * @throws Exception if anything goes wrong
42      */
43     public static void usernameToken() throws Exception {
44         // criando as configurações da mensagem
45         Configurable wsConfigurable = Configurator.newRuntimeConfigurable();
46         MessageConf messageConf = (MessageConf) wsConfigurable.narrow(MessageConf.class);
47
48         // criando o username token
49         SecurityTokenConf tokenConf = messageConf.newSecurityToken();
50         tokenConf.setType(Constants.ST_VALUE_TYPE_USERNAME);
51         PropertyConf propertyConf = tokenConf.newProperty();
52         propertyConf.setPropertyName(Constants.ST_PROPERTY_NAME_PASSWORD_TYPE);
53         propertyConf.setPropertyValue(Constants.PT_TEXT_VALUE);
54         PropertyConf propertyConfNoNonceCreated = tokenConf.newProperty();
55         propertyConfNoNonceCreated.setPropertyName
   (Constants.ST_PROPERTY_NAME_NO_NONCE_CREATED);
56         propertyConfNoNonceCreated.setPropertyValue("true");
57         tokenConf.setProperties(new PropertyConf[]{propertyConf, propertyConfNoNonceCreated});
```

```
58          tokenConf.setWsuId("UsernameToken-1");
59          tokenConf.setOrder(new Integer(0));
60
61          // setando as opções de segurança da mensagem
62          messageConf.setSecurityTokens(new SecurityTokenConf[]{tokenConf});
63
64          // armazenando as configurações de segurança da mensagem no contexto
65          serviceClient.getCallContext().getContextData().put(Constants.CD_MESSAGE_CONF,
    messageConf);
66
67          // chamada do serviço
68          System.out.println("Cenario 1: username token");
69          String response = serviceSoap.usernameToken("Resposta");
70          System.out.println("  Resposta do servico: " + response + "\n");
71      }
72
73      /**
74       * Cenário 2: o cliente envia um username token cifrado.
75       * @throws Exception if anything goes wrong
76       */
77      public static void encryptedUsernameToken() throws Exception {
78
79          // criando as configurações da mensagem
80          Configurable wsConfigurable = Configurator.newRuntimeConfigurable();
81          MessageConf messageConf = (MessageConf) wsConfigurable.narrow(MessageConf.class);
82
83          // criando o username token
84          SecurityTokenConf tokenConf = messageConf.newSecurityToken();
85          tokenConf.setType(Constants.ST_VALUE_TYPE_USERNAME);
86          PropertyConf propertyConf = tokenConf.newProperty();
87          propertyConf.setPropertyName(Constants.ST_PROPERTY_NAME_PASSWORD_TYPE);
88          propertyConf.setPropertyValue(Constants.PT_TEXT_VALUE);
89          tokenConf.setProperties(new PropertyConf[]{propertyConf});
90          tokenConf.setWsuId("UsernameToken-1");
91          tokenConf.setOrder(new Integer(0));
92          messageConf.setSecurityTokens(new SecurityTokenConf[]{tokenConf});
93
94          // criando a chave para a cifragem
95          EncryptedKeyConf encKeyConf = messageConf.newEncryptedKey();
96          encKeyConf.setEncryptionMethodAlgorithm(Constants.ALGO_ID_KEYTRANSPORT_RSA15);
```

```
97          KeyInfoConf keyInfoConf = encKeyConf.newKeyInfo();
98          keyInfoConf.setSecurityTokenMode(Constants.STM_KEYIDENTIFIER);
99          keyInfoConf.setSecurityTokenId(ENCRYPTING_TOKEN_ID);
100         encKeyConf.setKeyInfo(keyInfoConf);
101         encKeyConf.setOrder(new Integer(20));
102
103         // criando os dados cifrados
104         EncryptedDataConf encDataConf = messageConf.newEncryptedData();
105         encDataConf.setEncryptionMethodAlgorithm(Constants.ALGO_ID_BLOCKCIPHER_TRIPLEDES);
106         encDataConf.setEncryptElementContent(new Boolean(false));
107         encDataConf.setEncryptionTargetId("UsernameToken-1");
108         encDataConf.setWsuId("EncryptedData-1");
109         encDataConf.setOrder(new Integer(10));
110
111         // criando uma referência para os dados cifrados
112         EncryptionReferenceConf encryptionReferenceConf = encKeyConf.newReference();
113         encryptionReferenceConf.setRefUri("#" + encDataConf.getWsuId());
114         encKeyConf.setReferences(new EncryptionReferenceConf[]{encryptionReferenceConf});
115
116         // setando as configurações de segurança da mensagem
117         messageConf.setEncryptedData(new EncryptedDataConf[]{encDataConf});
118         messageConf.setEncryptedKeys(new EncryptedKeyConf[]{encKeyConf});
119
120         // armazenando as configurações de segurança da mensagem no contexto
121         serviceClient.getCallContext().getContextData().put(Constants.CD_MESSAGE_CONF,
    messageConf);
122
123         // chamada do serviço
124         System.out.println("Cenario 2: username token cifrado");
125         String response = serviceSoap.encryptedUsernameToken("Resposta");
126         System.out.println("  Resposta do servico: " + response + "\n");
127     }
128
129     /**
130      * Cenário 3: o cliente assina e cifra o corpo da mensagem.
131      * @throws Exception if anything goes wrong
132      */
133     public static void signedEncryptedBody() throws Exception {
134         // criando as configurações de sugurança da mensagem
135         Configurable wsConfigurable = Configurator.newRuntimeConfigurable();
```

```
136         MessageConf messageConf = (MessageConf) wsConfigurable.narrow(MessageConf.class);
137
138         // criando um token binário para a assinatura (usa a chave privada do cliente)
139         SecurityTokenConf signingSecurityTokenConf = messageConf.newSecurityToken();
140         signingSecurityTokenConf.setType(Constants.ST_VALUE_TYPE_X509V3);
141         signingSecurityTokenConf.setWsuId("SigningSecurityToken-1");
142         signingSecurityTokenConf.setOrder(new Integer(10));
143
144         // criando a assinatura
145         SignatureConf signatureConf = messageConf.newSignature();
146         signatureConf.setSignBody(Boolean.TRUE);
147         KeyInfoConf keyInfoConf = signatureConf.newKeyInfo();
148         keyInfoConf.setSecurityTokenMode(Constants.STM_REFERENCE);
149         keyInfoConf.setSecurityTokenId("SigningSecurityToken-1");
150         signatureConf.setKeyInfo(keyInfoConf);
151         signatureConf.setOrder(new Integer(0));
152         signatureConf.setWsuId("Signature-1");
153         signatureConf.setSignatureMethod(Constants.ALGO_ID_SIGNATURE_RSA);
154
155         // criando os dados cifrados
156         EncryptedDataConf encryptedDataConf = messageConf.newEncryptedData();
157 encryptedDataConf.setEncryptionMethodAlgorithm(Constants.ALGO_ID_BLOCKCIPHER_TRIPLEDES);
158         encryptedDataConf.setEncryptBody(new Boolean(true));
159         encryptedDataConf.setEncryptElementContent(new Boolean(true));
160         encryptedDataConf.setWsuId("EncryptedData-1");
161         encryptedDataConf.setOrder(new Integer(20));
162
163         // criando a chave para a cifragem
164         EncryptedKeyConf encryptedKeyConf = messageConf.newEncryptedKey();
165 encryptedKeyConf.setEncryptionMethodAlgorithm(Constants.ALGO_ID_KEYTRANSPORT_RSA15);
166         keyInfoConf = encryptedKeyConf.newKeyInfo();
167         keyInfoConf.setSecurityTokenMode(Constants.STM_KEYIDENTIFIER);
168         keyInfoConf.setSecurityTokenId(ENCRYPTING_TOKEN_ID);
169         encryptedKeyConf.setKeyInfo(keyInfoConf);
170         encryptedKeyConf.setWsuId("EncryptedKey-1");
171         encryptedKeyConf.setOrder(new Integer(25));
172
173         // criando uma referência para os dados cifrados
174         EncryptionReferenceConf encryptionReferenceConf = encryptedKeyConf.newReference();
175         encryptionReferenceConf.setRefUri("#" + encryptedDataConf.getWsuId());
```

```
176         encryptedKeyConf.setReferences(new EncryptionReferenceConf[]{encryptionReferenceConf});
177
178         // setando as configurações de segurança da mensagem
179         messageConf.setSignatures(new SignatureConf[]{signatureConf});
180         messageConf.setEncryptedData(new EncryptedDataConf[]{encryptedDataConf});
181         messageConf.setEncryptedKeys(new EncryptedKeyConf[]{encryptedKeyConf});
182         messageConf.setSecurityTokens(new SecurityTokenConf[]{signingSecurityTokenConf});
183         messageConf.setCreateTimestampHeader(Boolean.TRUE);
184
185         // armazenando as configurações de segurança da mensagem no contexto
186         serviceClient.getCallContext().getContextData().put(Constants.CD_MESSAGE_CONF,
    messageConf);
187
188         // chamada do serviço
189         System.out.println("Cenario 3: corpo da mensagem SOAP assinado e cifrado");
190         String response = serviceSoap.signedEncryptedBody("Resposta");
191         System.out.println("  Resposta do servico: " + response + "\n");
192     }
193
194     private static void setupClientContextSecurity() {
195         // criando um token binário externo para a cifragem (chave pública do serviço)
196         Configurable configurable = Configurator.newRuntimeConfigurable();
197         WSSEConf securityConf = (WSSEConf)configurable.narrow(WSSEConf.class);
198         ExternalSecurityTokensConf extSecToken = securityConf.newExternalSecurityTokens();
199         SecurityTokenConf etokenConf = extSecToken.newSecurityToken();
200         etokenConf.setType(Constants.ST_VALUE_TYPE_X509V3);
201         etokenConf.setWsuId(ENCRYPTING_TOKEN_ID);
202         PropertyConf propertyConf = etokenConf.newProperty();
203         propertyConf.setPropertyName(Constants.ST_PROPERTY_NAME_ALIAS);
204         propertyConf.setPropertyValue("CN=WsSecurityTesteService"); // alias para o servico
205         etokenConf.setProperties(new PropertyConf[]{propertyConf});
206
207         // token binário externo usado para decifrar
208         SecurityTokenConf etokenConf2 = extSecToken.newSecurityToken();
209         etokenConf2.setType(Constants.ST_VALUE_TYPE_X509V3);
210
211         // setando os tokens externos
212         extSecToken.setSecurityTokens(new SecurityTokenConf[]{etokenConf,etokenConf2});
213         securityConf.setExternalSecurityTokens(extSecToken);
214
```

```
215         // setando o contexto de segurança do cliente
216         serviceClient.getContext().getContextData().put(Constants.CD_SECURITY_CONFIGURATION,
217                                                 securityConf);
218     }
219
220     /**
221      * Main method.
222      * @param args not used
223      * @throws Exception if anything goes wrong
224      */
225     public static void main(String args[]) throws Exception {
226         serverURL = System.getProperty("systinet.teste.server.url", "http://localhost:6060");
227
228         // criando uma instancia do cliente para o webservice dado
229         serviceClient = ServiceClient.create(serverURL + serviceWSDLPath);
230
231         // criando o proxy
232         serviceSoap = (WsSecurityService)serviceClient.createProxy(WsSecurityService.class);
233
234         // autenticando o cliente e setando as credenciais do serviço
235         Credentials creds = WaspSecurity.acquireClientCredentials("Gabriel", "asdf", "WS-Security");
236         WaspSecurity.setCredentials(serviceClient, new Credentials[]{creds});
237         WaspSecurity.setInitiatingProvider(serviceClient, "WS-Security");
238
239         // setando o contexto de segurança para o cliente
240         setupClientContextSecurity();
241
242         /** Cenário 1: o cliente envia o username token com uma senha em plaintext */
243         usernameToken();
244
245         /** Cenário 2: o cliente envia um username token cifrado */
246         encryptedUsernameToken();
247
248         /** Cenário 3: o cliente assina e cifra o corpo da mensagem */
249         signedEncryptedBody();
250     }
251 }
```

Comentários relevantes sobre o arquivo WsSecurityClient.java:

Linhas	WsSecurityClient.java
26	Declaração do caminho do endpoint do serviço web.
37	Declaração do identificador do token que será usado para cifrar dados.
43 a 71	Método usado para simular o cliente no cenário 1. Detalhes abaixo.
49 a 59	Criando o username token que será usado para a autenticação no cenário 1. Notar que são criadas duas propriedades, uma para o nome e outra para a senha. Na propriedade senha é configurado o tipo de valor enviado, no caso texto puro.
68 a 70	É feita a chamada do serviço, simulando a autenticação do cliente.
77 a 127	Método usado para simular o cliente no cenário 2. Detalhes abaixo.
84 a 92	Criando o username token que será usado para a autenticação no cenário 2. Notar que são criadas duas propriedades, uma para o nome e outra para a senha. Na propriedade senha é configurado o tipo de valor enviado, no caso texto puro.
95 a 101	Criação da chave que será usada para cifrar os dados. O algoritmo usado para a criação da chave é o RSA versão 1.5.
104 a 109	Criação dos dados cifrados. Notar que o algoritmo usado para cifrar os dados é o TripleDES
124 a 126	É feita a chamada do serviço, simulando a autenticação do cliente.
133 a 192	Método usado para simular o cliente no cenário 3. Detalhes abaixo.
139 a 142	Criação do token que será usado para assinar a mensagem. É criado um certificado X509v3.
145 a 153	Criação da assinatura que será usado para assinar a mensagem SOAP que será enviada para o serviço web. É usado o algoritmo RSA para a criação dessa assinatura.
156 a 161	Criação dos dados cifrados. Notar que o algoritmo usado para cifrar os dados é o TripleDES
164 a 171	Criação da chave que será usada para cifrar os dados. O algoritmo usado é o RSA versão 1.5.
189 a 191	É feita a chamada do serviço, simulando a autenticação do cliente.
194 a 218	Método usado para fazer as configurações de segurança do contexto. Segue detalhes.
196 a 205	Feita a criação do token externo (certificado digital) usado para cifrar os dados. É chamado de chave pública do serviço web. O certificado digital é criado usando X509v3.
208 a 209	Criação do token que será usado para decifrar a mensagem
225 a 250	Método main usado para iniciar o cliente. Detalhes abaixo.
229	Criação de uma instância do cliente para o serviço web.
235 a 237	Autenticando o cliente e configurando as credenciais do serviço web.
243	Iniciando o cenário 1.
246	Iniciando o cenário 2.
249	Iniciando o cenário 3.

5.2 O Arquivo run.bat

Este arquivo tem o objetivo de facilitar a compilação das classes e criar o serviço web no servidor WASP. A seguir vamos comentar algumas linhas deste arquivo para melhor entendermos seu funcionamento. O arquivo completo segue anexo.

Linhas	Run.bat
10	SET EXPORTED_SERVICE_CERT=.\build\service.crt
11	SET EXPORTED_CLIENT_CERT=.\build\client.crt
21	SET WSDL_URL=%SERVER_URL%/teste/security/ws-security/WsSecurityService/wsdl
22	SET SERVICE_IDENTITY=WsSecurityTesteService
24	SET CLIENT_IDENTITY=Gabriel
25	SET CLIENT_IDENTITY_PASSWORD=asdf
106	%JAVA_HOME%\bin\java "-Djava.security.auth.login.config=%WASP_HOME%/conf/jaas.config" -cp "%RUN_TOOLS_CLASSPATH%" com.idoox.wasp.security.tools.IdentityTool newServer -alias %SERVICE_IDENTITY% -keyPassword %SERVICE_IDENTITY_PASSWORD% -url %SERVER_URL% -username %USER% -password %PASSWORD%
108	%JAVA_HOME%\bin\java "-Djava.security.auth.login.config=%WASP_HOME%/conf/jaas.config" -cp "%RUN_TOOLS_CLASSPATH%" com.idoox.wasp.security.tools.IdentityTool exportServer -alias %SERVICE_IDENTITY% -url %SERVER_URL% -username %USER% -password %PASSWORD% -certFile %EXPORTED_SERVICE_CERT%
110	%JAVA_HOME%\bin\java "-Djava.security.auth.login.config=%WASP_HOME%/conf/jaas.config" -cp "%RUN_TOOLS_CLASSPATH%" com.idoox.wasp.security.tools.IdentityTool add -alias %SERVICE_IDENTITY% -url %SERVER_URL% -username %USER% -password %PASSWORD%
112	%JAVA_HOME%\bin\java "-Djava.security.auth.login.config=%WASP_HOME%/conf/jaas.config" -cp "%RUN_TOOLS_CLASSPATH%" com.idoox.wasp.security.tools.IdentityTool new -alias %CLIENT_IDENTITY% -keyPassword %CLIENT_IDENTITY_PASSWORD% -username %USER% -password %PASSWORD%
114	%JAVA_HOME%\bin\java "-Djava.security.auth.login.config=%WASP_HOME%/conf/jaas.config" -cp "%RUN_TOOLS_CLASSPATH%" com.idoox.wasp.security.tools.IdentityTool export -alias %CLIENT_IDENTITY% -certFile %EXPORTED_CLIENT_CERT%
116	%JAVA_HOME%\bin\java "-Djava.security.auth.login.config=%WASP_HOME%/conf/jaas.config" -cp "%RUN_TOOLS_CLASSPATH%" com.idoox.wasp.security.tools.IdentityTool addServer -alias %CLIENT_IDENTITY% -url %SERVER_URL% -username %USER% -password %PASSWORD%
118	%JAVA_HOME%\bin\java "-Djava.security.auth.login.config=%WASP_HOME%/conf/jaas.config" "-Dwasp.policy.xml.file=%WASP_HOME%" -cp "%RUN_TOOLS_CLASSPATH%" com.idoox.wasp.tools.security.IdentitiesManager -t %SERVER_URL% -a %CLIENT_IDENTITY% -p password -v %CLIENT_IDENTITY_PASSWORD% -p X509Certificate -f %EXPORTED_CLIENT_CERT% --username %USER% --password %PASSWORD%
120	%JAVA_HOME%\bin\java "-Djava.security.auth.login.config=%WASP_HOME%/conf/jaas.config" -

```
cp "%RUN_TOOLS_CLASSPATH%" com.idoox.wasp.security.tools.IdentityTool add -certFile
%EXPORTED_CLIENT_CERT%
135 %JAVA_HOME%\bin\javac -classpath "%COMPILE_SERVER_CLASSPATH%" -sourcepath
%SRC_DIR% -d %SERVER_CLASSDIR%
%SRC_DIR%\teste\security\wssecurity\persistent\server\*.java
142 CALL Java2WSDL --classpath "%SERVER_CLASSDIR%" -d %BUILD_DIR% --soap-binding-style
document --soap-encoding-style literal --force --package-mapping
teste.security.wssecurity.persistent.server=http://hanauer.com/wsdl/teste/security/wssecurity/server/
--output-file-mapping
http://hanauer.com/wsdl/teste/security/wssecurity/server/=WsSecurityService.wsdl
teste.security.wssecurity.persistent.server.WsSecurityService
147 CALL WaspPackager -p WsSecurityTesteService -o %BUILD_DIR%\service.jar --force -s
%SERVER_CLASSDIR% --dd dd/package.xml -w build\WsSecurityService.wsdl
162 %JAVA_HOME%\bin\javac -classpath "%COMPILE_SERVER_CLASSPATH%" -sourcepath
%SRC_DIR% -d %SERVER_CLASSDIR%
%SRC_DIR%\teste\security\wssecurity\runtime\lowlevel\server\*.java
189 CALL Deploy -j %BUILD_DIR%/service.jar -t %SERVER_URL% --redeploy --username %USER% -
-password %PASSWORD%
191 %JAVA_HOME%\bin\java "-Djava.security.auth.login.config=%WASP_HOME%/conf/jaas.config" -
cp "%RUN_TOOLS_CLASSPATH%" com.idoox.wasp.tools.security.ProvidersManager -t
%SERVER_URL% -b /teste/security/ws-security/WsSecurityService --iadd -p WS-Security --iname
%SERVICE_IDENTITY% --ipass %SERVICE_IDENTITY_PASSWORD% --username %USER% --
password %PASSWORD%
204 CALL WSDL2Java --output-directory %BUILD_DIR%\src --url %WSDL_URL% -p
teste.security.wssecurity.persistent.client --force
208 %JAVA_HOME%\bin\javac -classpath "%COMPILE_CLIENT_CLASSPATH%" -sourcepath
%BUILD_DIR%\src -d %CLIENT_CLASSDIR%
%BUILD_DIR%\src\teste\security\wssecurity\persistent\client\WsSecurityService.java
212 %JAVA_HOME%\bin\javac -classpath "%COMPILE_CLIENT_CLASSPATH%" -sourcepath .\src -d
%CLIENT_CLASSDIR% .\src\teste\security\wssecurity\persistent\client\*.java
218 CALL WaspPackager -C -o %BUILD_DIR%\client.jar --force --dd dd/client.xml
255 %JAVA_HOME%\bin\java "-Djava.security.auth.login.config=%WASP_HOME%/conf/jaas.config" "-
Dhanauer.teste.server.url=%SERVER_URL%" -cp "%RUN_CLIENT_DD_CLASSPATH%"
teste.security.wssecurity.persistent.client.WsSecurityClient
281 %JAVA_HOME%\bin\java "-Djava.security.auth.login.config=%WASP_HOME%/conf/jaas.config" "-
Dhttp.proxyHost=%PROXY_HOST%" "-Dhttp.proxyPort=%PROXY_PORT%" "-
Dhttp.nonProxyHosts=" "-Dhanauer.teste.server.url=%SERVER_URL%" -cp
"%RUN_CLIENT_DD_CLASSPATH%" teste.security.wssecurity.persistent.client.WsSecurityClient
307 CALL Undeploy -j %BUILD_DIR%/service.jar --target %SERVER_URL% --username %USER% --
password %PASSWORD%
```

Comentários sobre as linhas mais importantes do arquivo run.bat:

Linhas	Run.bat
10	Seta o nome do arquivo onde será exportado o certificado digital do serviço
11	Seta o nome do arquivo onde será exportado o certificado digital do cliente
21	Seta a URL do WSDL
22	Seta a identidade do serviço
24	Seta a identidade do cliente
25	Seta a senha do cliente
106	Cria a identidade do serviço no servidor
108	Exporta o certificado da identidade do serviço para um arquivo
110	Adiciona a identidade certificada do serviço na base de chaves do cliente para garantir que ele seja confiável
112	Cria a identidade do cliente
114	Exporta o certificado da identidade do cliente para um arquivo
116	Adiciona a identidade certificada do cliente na base de chaves do servidor para garantir que ele seja confiável
118	Designa o certificado e senha ao cliente pelo servidor
120	Adiciona a identidade certificada do cliente na base de chaves do cliente para garantir que ele seja confiável
135	Compila as classes do serviço
142	Gera o arquivo WSDL a partir das classes Java
147	Cria o pacote Java
162	Compila as classes do servidor
189	Adiciona o serviço no servidor
191	Associa a identidade ao endpoint
204	Compilando o WSDL para código fonte Java
208	Compila os stubs
212	Compila o cliente
218	Cria o pacote do cliente
255	Roda o cliente
281	Roda o cliente em modo spy para pegarmos as mensagens SOAP com o SoaSpy
307	Removendo serviço do servidor

5.3 Os Arquivos de descrição client.xml e package.xml

Esses arquivos possuem as definições XML dos pacotes que serão criados para o cliente e para o serviço web. Esses pacotes são usados pelo servidor WASP. Segue uma descrição de algumas linhas desses arquivos. Anexo segue uma cópia dos arquivos.

5.3.1 Client.xml

Linhas	Client.xml
15	<!-- tokens externos -->
16	<externalSecurityTokens>
17	<!-- token para a cifragem da mensagem (cenario 2, 3) -->
18	<securityToken type="X509v3" wsuId="EncryptingSecurityToken-1">
19	<property propertyName="alias" propertyValue="CN=WsSecurityTesteService"/>
20	</securityToken>
21	<!-- token usado para decifrar a resposta (cenario 3) -->
22	<securityToken type="X509v3"/>
23	</externalSecurityTokens>
26	<securedMessage>
27	<methodName>usernameToken</methodName>
28	<messageConf>
29	<securityToken type="Username">
30	<wsuId>SecurityToken-1</wsuId>
31	<order>0</order>
32	<property propertyName="PasswordType" propertyValue="PasswordText"/>
33	<property propertyName="noNonceAndCreated" propertyValue="true"/>
34	</securityToken>
35	</messageConf>
36	</securedMessage>
38	<!-- cenario 2 mensagem segura (username cifrado) -->
39	<securedMessage>
40	<!-- metodo -->
41	<methodName>encryptedUsernameToken</methodName>
42	<messageConf>
43	<!-- username token -->
44	<securityToken type="Username">
45	<wsuId>SecurityToken-1</wsuId>
46	<order>10</order>

```
47                  <property propertyName="PasswordType" propertyValue="PasswordText"/>
48              </securityToken>
49
50              <!-- chave para cifrar -->
51              <encryptedKey>
52                  <wsuId>EncryptedKey-1</wsuId>
                    <encryptionMethodAlgorithm>http://www.w3.org/2001/04/xmlenc#rsa-
53  1_5</encryptionMethodAlgorithm>
54                  <keyInfo securityTokenId="EncryptingSecurityToken-1"
55                      securityTokenMode="keyidentifier"/>
56                  <reference URI="#EncryptedData-1"/>
57                  <order>50</order>
58              </encryptedKey>
59
60              <!-- dados cifrados -->
61              <encryptedData>
62                  <wsuId>EncryptedData-1</wsuId>
                    <encryptionMethodAlgorithm>http://www.w3.org/2001/04/xmlenc#tripledes-
63  cbc</encryptionMethodAlgorithm>
64                  <encryptionTargetId>SecurityToken-1</encryptionTargetId>
65                  <order>15</order>
66              </encryptedData>
67          </messageConf>
68      </securedMessage>
69
70      <!-- cenario 3 mensagem segura (corpo da mensagem assinado e cifrado) -->
71      <securedMessage>
72          <!-- metodo -->
73          <methodName>signedEncryptedBody</methodName>
74          <messageConf createTimestampHeader="true">
75
76              <!-- token para assinatura -->
77              <securityToken type="X509v3">
78                  <wsuId>SigningSecurityToken-1</wsuId>
79                  <order>10</order>
80              </securityToken>
81
82              <!-- assinatura do corpo -->
83              <signature>
84                  <wsuId>Signature-1</wsuId>
```

```
85                  <signatureMethod>http://www.w3.org/2000/09/xmldsig#rsa-sha1</signatureMethod>
86                  <keyInfo securityTokenId="SigningSecurityToken-1"
87                     securityTokenMode="reference"/>
88                  <signBody>true</signBody>
89                  <order>0</order>
90               </signature>
91
92               <!-- chave para cifrar -->
93               <encryptedKey>
94                  <wsuId>EncryptedKey-1</wsuId>
                    <encryptionMethodAlgorithm>http://www.w3.org/2001/04/xmlenc#rsa-
95  1_5</encryptionMethodAlgorithm>
96                  <keyInfo securityTokenId="EncryptingSecurityToken-1"
97                     securityTokenMode="keyidentifier"/>
98                  <reference URI="#EncryptedData-1"/>
99                  <order>25</order>
100              </encryptedKey>
101
102              <!-- dados cifrados -->
103              <encryptedData>
104                 <wsuId>EncryptedData-1</wsuId>
                    <encryptionMethodAlgorithm>http://www.w3.org/2001/04/xmlenc#tripledes-
105 cbc</encryptionMethodAlgorithm>
106                 <encryptBody>true</encryptBody>
107                 <encryptElementContent>true</encryptElementContent>
108                 <order>20</order>
```

Comentários sobre as linhas mais importantes do arquivo cliente.xml:

Linhas	Client.xml
15 a 23	Especificação dos tokens externos usados nas trocas das mensagens
26 a 36	Especificação do cenário 1, isto é, do método do cenário 1 como também do username token e da senha
38 a 68	Especificação do cenário 2. É especificada também a chave usada para cifragem. A chave é gerada usando-se o algoritmo RSA versão 1.5. Neste cenário ainda é especificada a maneira como os dados serão cifrados e o algoritmo usado neste caso foi o Triple DES no modo CBC.
70 a 108	Especificação do cenário 3. Assim como no cenário 2, é especificada a chave e a maneira como os dados serão cifrados (Triple DES no modo CBC). Além disso é especificado o token usado para assinar a mensagem e de que maneira a mensagem será assinada (RSA-SHA1).

5.3.2 Package.xml

Linhas	Package.xml
15	<service-endpoint name="WsSecurityService"
16	path="/teste/security/ws-security/WsSecurityService"
17	service-instance="tns:WsSecurityService_inst"
18	accepting-security-providers="WS-Security">
19	<wsdl uri="WsSecurityService.wsdl"
20	service="wsdlns:WsSecurityService"/>
28	<!-- adicionando o token X509 para decifrar os dados (cenario 2, 3) -->
29	<externalSecurityTokens>
30	<securityToken type="X509v3"/>
31	</externalSecurityTokens>
36	<!-- cenario 1 mansagem de resposta (sem segurança) -->
37	<securedMessage>
38	<methodName>usernameToken</methodName>
39	<messageConf noSecurityHeader="true"/>
40	</securedMessage>
42	<!-- cenario 2 mensagem de resposta (sem segurança) -->
43	<securedMessage>
44	<methodName>encryptedUsernameToken</methodName>
45	<messageConf noSecurityHeader="true"/>
46	</securedMessage>
48	<!-- cenario 3 mensagem de reposta (corpo assinado e cifrado) -->
49	<securedMessage>
50	<!-- metodo -->
51	<methodName>signedEncryptedBody</methodName>
52	<messageConf>
53	<!-- token para assinatura -->
54	<securityToken type="X509v3">
55	<wsuId>SigningSecurityToken-1</wsuId>
56	<order>10</order>

```
57                </securityToken>
58
59                <!-- assinando o corpo da mensagem -->
60                <signature>
61                   <wsuId>Signature-1</wsuId>
62                   <signatureMethod>http://www.w3.org/2000/09/xmldsig#rsa-sha1</signatureMethod>
63                   <keyInfo securityTokenId="SigningSecurityToken-1"
64                      securityTokenMode="reference"/>
65                   <signBody>true</signBody>
66                   <order>0</order>
67                </signature>
68
69                <!-- chave para cifrar -->
70                <encryptedKey>
71                   <wsuId>EncryptedKey-1</wsuId>
                     <encryptionMethodAlgorithm>http://www.w3.org/2001/04/xmlenc#rsa-
72  1_5</encryptionMethodAlgorithm>
73                   <keyInfo securityTokenId="EncryptingBinaryToken-1"
74                      securityTokenMode="keyidentifier"/>
75                   <reference URI="#EncryptedData-1"/>
76                   <order>25</order>
77                </encryptedKey>
78
79                <!-- dados cifrados -->
80                <encryptedData>
81                   <wsuId>EncryptedData-1</wsuId>
                     <encryptionMethodAlgorithm>http://www.w3.org/2001/04/xmlenc#tripledes-
82  cbc</encryptionMethodAlgorithm>
83                   <encryptBody>true</encryptBody>
84                   <encryptElementContent>true</encryptElementContent>
85                   <order>20</order>
86                </encryptedData>
87             </messageConf>
88          </securedMessage>
```

Comentários sobre as linhas mais importantes do arquivo package.xml:

Linhas	Package.xml
15 a 20	Especificação do endpoint do web service, isto é, o caminho pelo qual poderemos acessar o web

	service
28 a 31	Epecificação do token X509 que será usado para decifrar os dados dos cenários 2 e 3.
36 a 40	Dizendo que a mensagem do cenário 1 não possuirá cabeçalho seguro.
42 a 46	Dizendo que a mensagem do cenário 2 não possuirá cabeçalho seguro.
53 a 57	Especificação do token que será usado para assinar a mensagem do cenário 3. O tipo do token é X509 versão 3.
60 a 67	Especificação de como será assinado o corpo da mensagem do cenário 3. O corpo da mensagem será assinado usando o método de assinatura RSA-SHA1.
70 a 77	Especificação da chave que será usada para cifrar os dados.
80 a 86	Como os dados serão cifrados. È usado o algoritmo triple DES em modo cbc.

Após uma explicação do funcionamento das classes e seus métodos vamos partir para os resultados obtidos na execução do protótipo.

Primeiramente vamos mostrar passo a passo como o protótipo e o serviço web foram criados a partir do arquivo run.bat que já foi explicado anteriormente.

5.4 Passo a passo – criação e execução

5.4.1 Criando as identidades e gerando os certificados

A criação das identidades e dos certificados é feita através de uma classe chamada IdentityTool que vem junto com o servidor WASP. A figura a seguir mostra a criação das identidades e certificados.

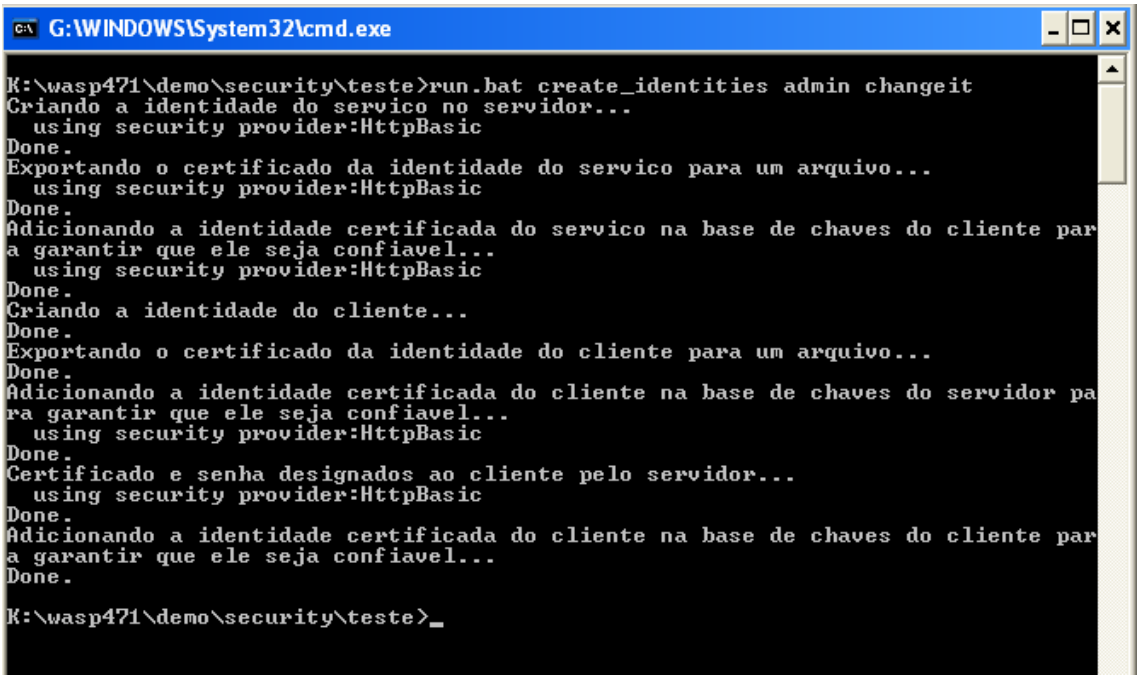

Figura 5.1 - Criando Identidades

Após a criação das identidades no servidor e da exportação dos certificados podemos perceber a criação do diretório build que contém os respectivos certificados. Do cliente e do serviço web. A figura a seguir mostra os certificados.

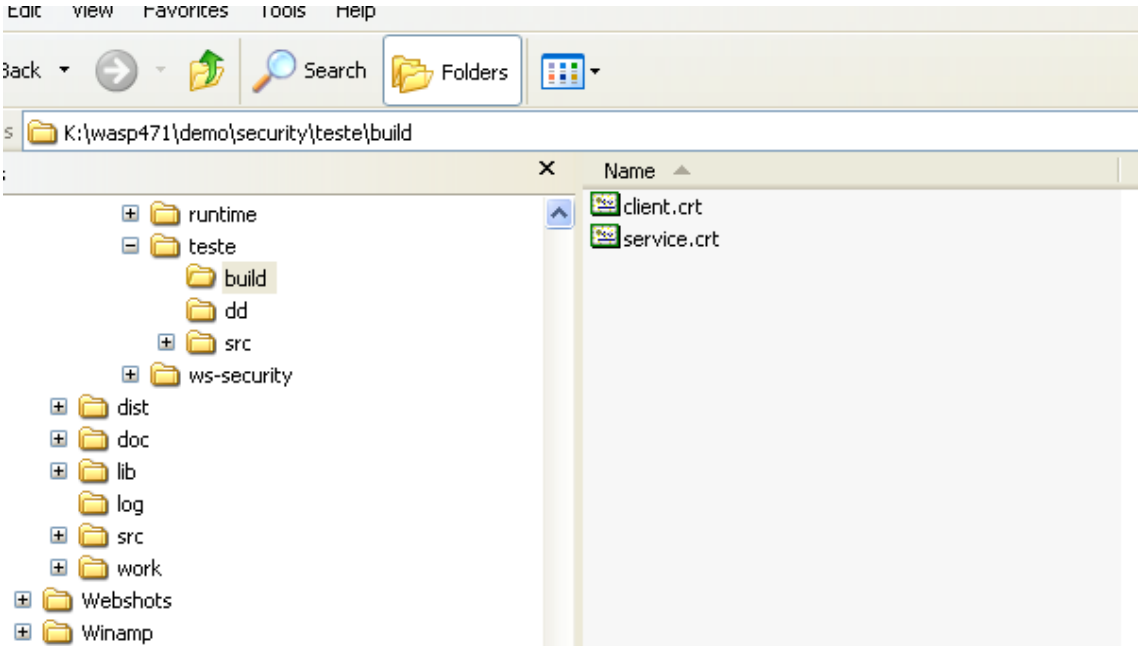

Figura 5.2 - Certificados

5.4.2 Compilando o serviço web, gerando a definição wsdl e criando o pacote

Nesta etapa vamos compilar as classes dos serviço web, gerar a definição WSDL e criar o pacote que será usado pelo servidor. A definição WSDL é gerada usando-se a ferramenta Java2WSDL.

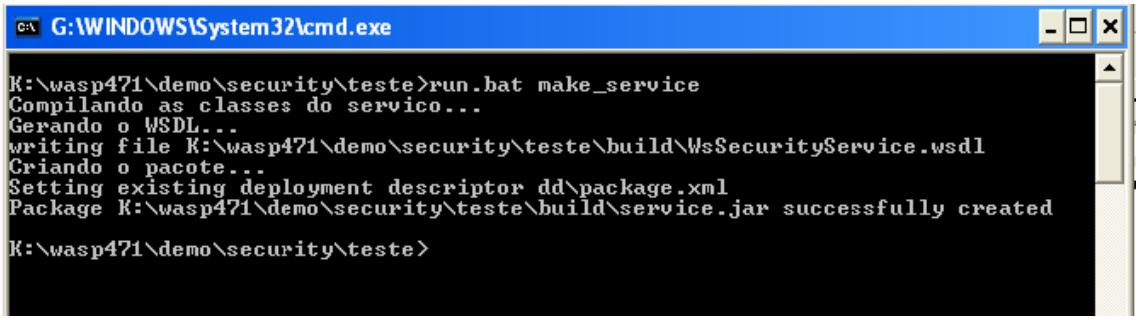

Figura 5.3 - Compilando o web service

Após a execução desse passo, podemos notar a criação da definição WSDL e do pacote Java, bem como o diretório onde foi colocado o arquivo .class do serviço web.

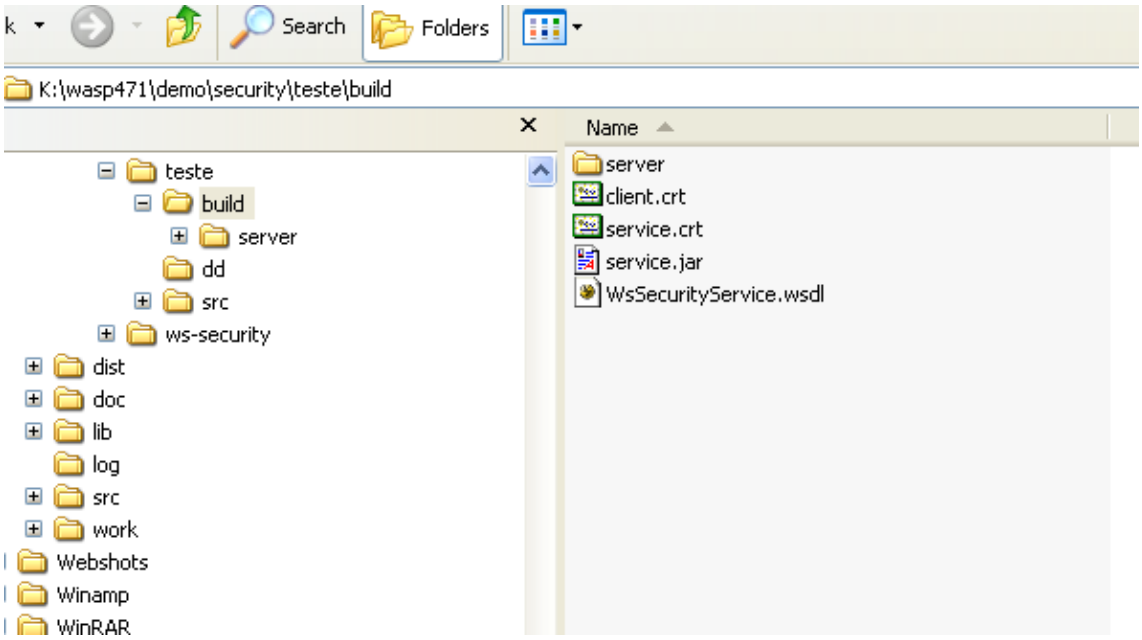

Figura 5.4 - Arquivos WSDL e pacote java do web service

5.4.3 Adicionando o serviço web no servidor WASP

Nesta etapa vamos adicionar o serviço web (deploy) no servidor WASP. É por este motivo que na etapa anterior criamos o pacote (service.jar), para que ele seja usado pelo servidor WASP.

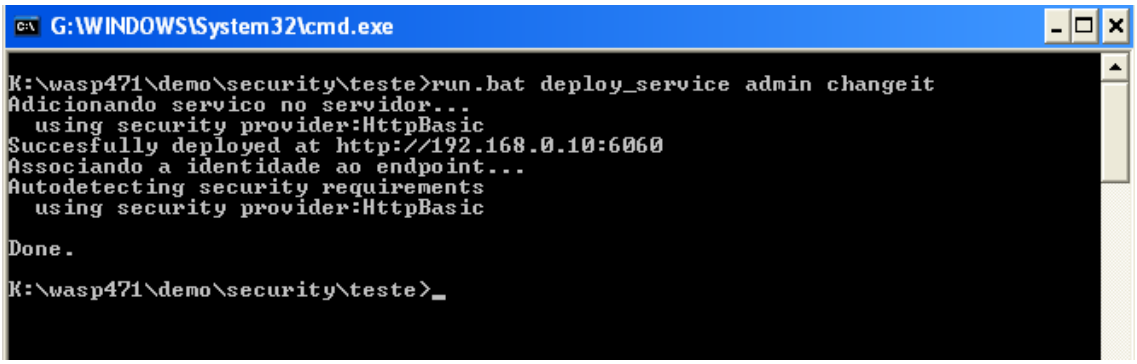

Figura 5.5 - Deploy do web service no servidor WASP

Podemos verificar nos logs do servidor WASP que o serviço web foi publicado com sucesso.

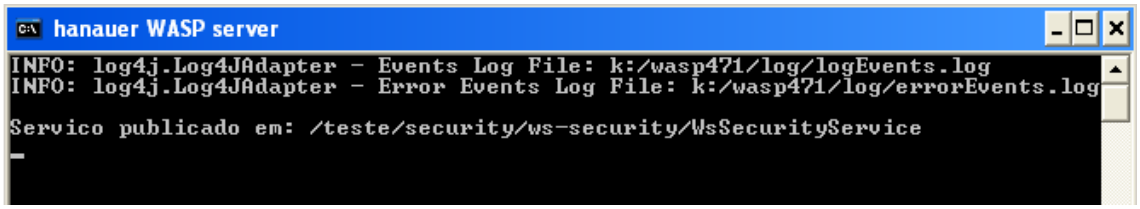

Figura 5.6 - Web service publicado

5.4.4 Compilando o cliente

Nesta etapa vamos gerar o código fonte Java a partir da definição WSDL criada anteriormente e compilar esses fontes. Esses fontes são o cliente que vai simular a autenticação do sistema. É também nesta etapa que criamos o pacote Java do cliente (client.jar).

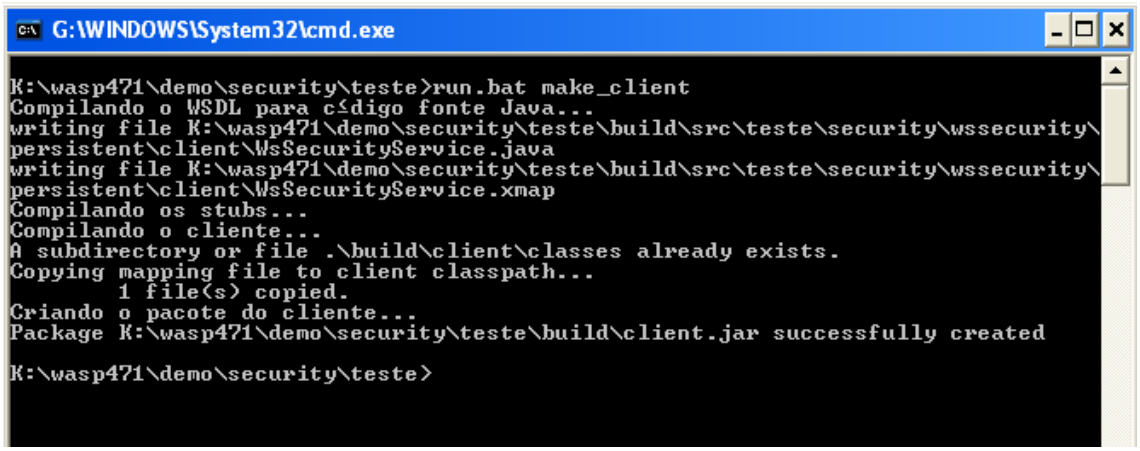

Figura 5.7 - Compilando o cliente

Agora já podemos notar a criação do pacote Java do cliente (client.jar) e do diretório onde estão as classes compiladas do cliente.

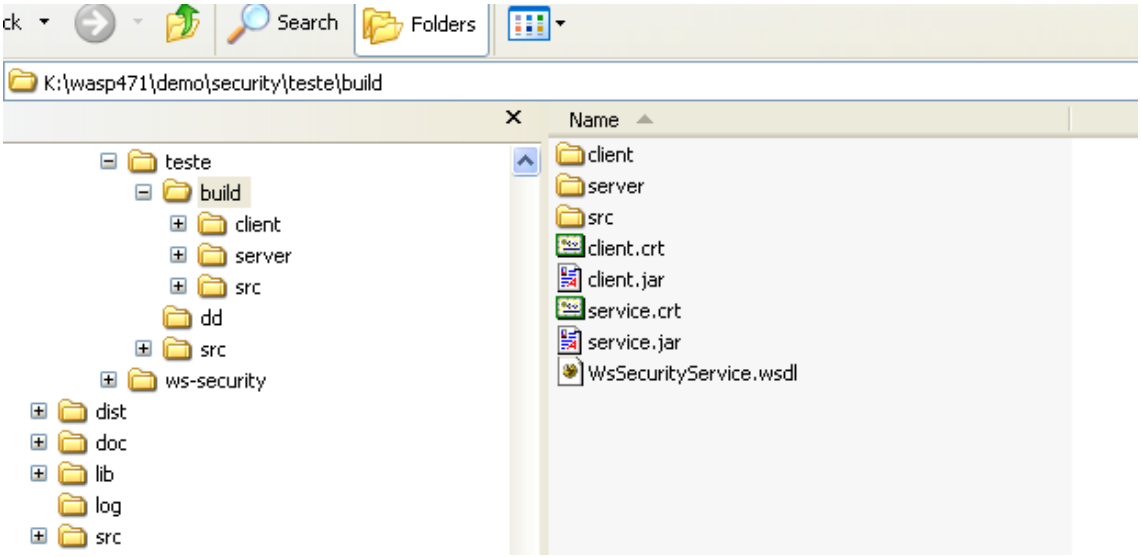

Figura 5.8 - Pacote do cliente gerado

5.4.5 Rodando o cliente

E finalmente vamos rodar o cliente para verificar se tudo está funcionando corretamente.

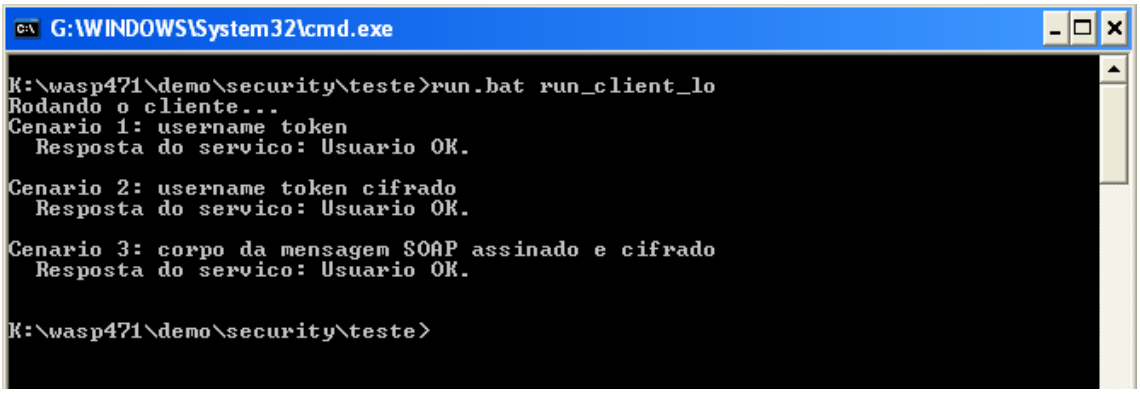

Figura 5.9 - Rodando o cliente

5.5 Resultados

A análise dos resultados foi feita usando-se a ferramenta Soap Spy. Esta ferramenta permite monitorarmos um determinado servidor e analisar as mensagens SOAP que estão sendo enviadas e recebidas. Além disso, podemos também analisar as mensagens HTTP de pedido e resposta.

Para podermos verificar e entender melhor os resultados, vamos dividir nossa análise em quatro pontos, a saber:

1. SOAPAction: faz parte do cabeçalho HTTP. Pode ser usado para indicar a intenção da requisição SOAP.
2. SOAPRequest: define a mensagem SOAP dentro de uma ação POST do HTTP.
3. Target URL: a página que vai ser usada para efetuar as ações HTTP.
4. SOAPResponse: a resposta SOAP segue as semânticas do HTTP em relação a códigos de erros.

Para facilitar o entendimento das mensagens vamos analisar apenas as mensagens SOAP. As mensagens HTTP seguem anexo.

A primeira chamada que o cliente faz é a requisição das definições WSDL do serviço web para saber quais serviços estão disponíveis.

HTTP Request

```
GET /teste/security/ws-security/WsSecurityService/wsdl HTTP/1.0
User-Agent: Systinet WASP Server for Java/4.7.1 (Java/1.4.0; Windows XP/5.1)
Host: localhost:6060
Connection: close
Proxy-Connection: close
```

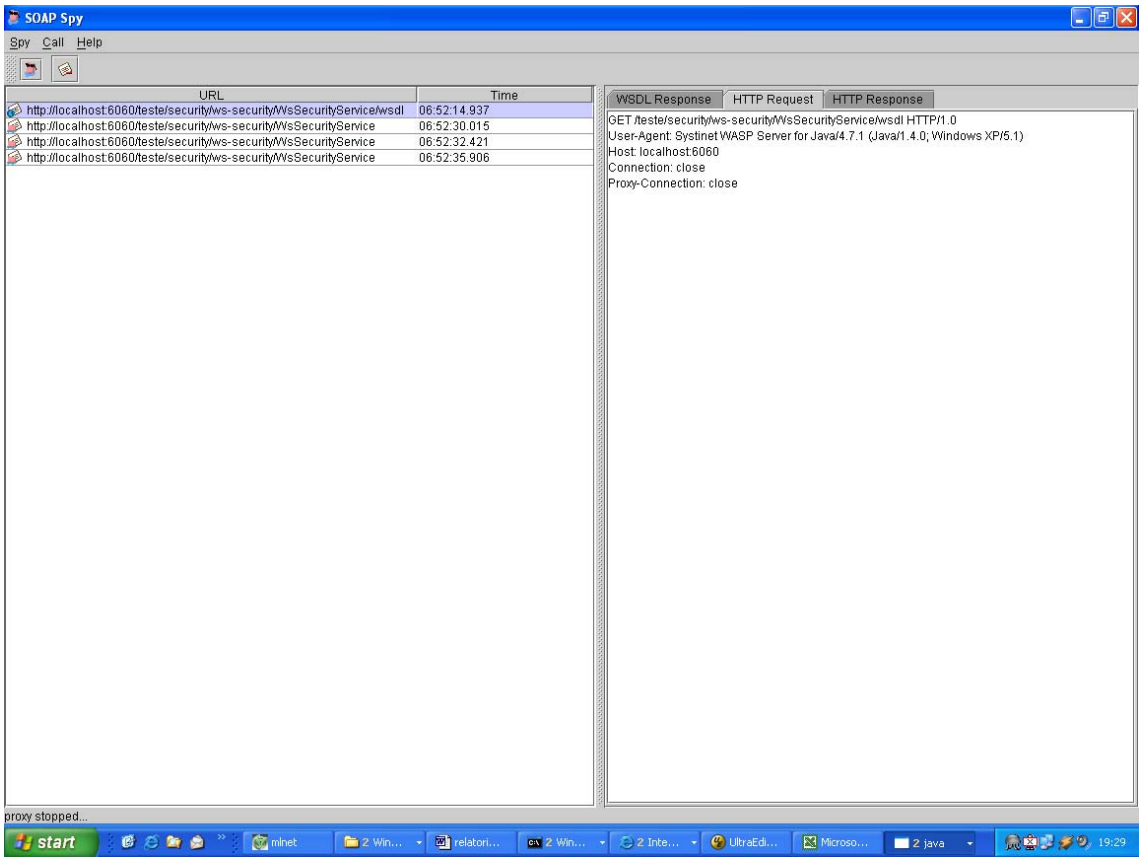

Figura 5.10 - SoapSpy

Após a requisição das definições WSDL, o cliente inicia a requisição dos serviços web disponíveis.

Separamos os resultados de acordo com os cenários propostos.

5.5.1 Cenário 1 – envio do nome do usuário e da senha em texto puro

Neste cenário devemos verificar se os dados não foram cifrados nem assinados.

5.5.1.1 SOAP Action

http://systinet.com/wsdl/teste/security/wssecurity/runtime/lowlevel/server/WsSecurityService#usernameToken?KExqYXZhL2xhbmcvU3RyaW5nOylMamF2YS9sYW5nL1N0cmluZzs=

Está chamando o método usernameToken do serviço web passando como parâmetro um token.

5.5.1.2 SOAP Request

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<e:Envelope xmlns:d="http://www.w3.org/2001/XMLSchema"
    xmlns:e="http://schemas.xmlsoap.org/soap/envelope/"
    xmlns:i="http://www.w3.org/2001/XMLSchema-instance"
xmlns:wn0="http://systinet.com/xsd/SchemaTypes/">
    <e:Header>
        <wsse:Security e:mustUnderstand="1"
            xmlns:wsse="http://schemas.xmlsoap.org/ws/2003/06/secext"
xmlns:wsu="http://schemas.xmlsoap.org/ws/2003/06/utility">
            <wsse:UsernameToken wsu:Id="UsernameToken-1">
                <wsse:Username>Gabriel</wsse:Username>
                <wsse:Password Type="wsse:PasswordText">asdf</wsse:Password>
            </wsse:UsernameToken>
        </wsse:Security>
    </e:Header>
    <e:Body>
        <wn0:p0 i:type="d:string">Usuario OK.</wn0:p0>
    </e:Body>
</e:Envelope>
```

5.5.1.3 SOAP Response

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<e:Envelope xmlns:d="http://www.w3.org/2001/XMLSchema"
    xmlns:e="http://schemas.xmlsoap.org/soap/envelope/"
    xmlns:i="http://www.w3.org/2001/XMLSchema-instance"
    xmlns:wn0="http://idoox.com/interface" xmlns:wn1="http://systinet.com/xsd/SchemaTypes/">
    <e:Body>
        <wn1:string_Response i:type="d:string">Usuario OK.</wn1:string_Response>
    </e:Body>
</e:Envelope>
```

Através do SOAP Request e SOAP Response podemos notar claramente que nenhum elemento XML foi cifrado.

5.5.2 Cenário 2 - o cliente envia um username token cifrado

Neste cenário o cliente envia o nome do usuário e a senha crifrados e recebe uma resposta do serviço web.

5.5.2.1 SOAP Action

```
http://systinet.com/wsdl/teste/security/wssecurity/runtime/lowlevel/server/
WsSecurityService#encryptedUsernameToken?KExqYXZhL2xhbmcvU3RyaW5nOylMam
F2YS9sYW5nL1N0cmluZzs=
```

Está chamando o método encryptedUsernameToken do serviço web passando como parâmetro um token.

5.5.2.2 SOAP Request

```
<?xml version="1.0" encoding="UTF-8"?>
<e:Envelope
  xmlns:d="http://www.w3.org/2001/XMLSchema"
  xmlns:e="http://schemas.xmlsoap.org/soap/envelope/"
  xmlns:i="http://www.w3.org/2001/XMLSchema-instance"
  xmlns:wn0="http://systinet.com/xsd/SchemaTypes/">

  <e:Header>
    <wsse:Security
      xmlns:wsse="http://schemas.xmlsoap.org/ws/2003/06/secext"
      xmlns:wsu="http://schemas.xmlsoap.org/ws/2003/06/utility"
      e:mustUnderstand="1">
```

```
<xenc:EncryptedKey
 xmlns:xenc="http://www.w3.org/2001/04/xmlenc#">

 <xenc:EncryptionMethod
Algorithm="http://www.w3.org/2001/04/xmlenc#rsa-1_5">
 </xenc:EncryptionMethod>

 <ds:KeyInfo xmlns:ds="http://www.w3.org/2000/09/xmldsig#">
  <wsse:SecurityTokenReference>
   <wsse:KeyIdentifier
    EncodingType="wsse:Base64Binary"
    ValueType="wsse:X509v3">
    T+DYSrzeRop+C64ebI8TuR4TyCQ=
   </wsse:KeyIdentifier>
  </wsse:SecurityTokenReference>
 </ds:KeyInfo>

 <xenc:CipherData>
  <xenc:CipherValue>
   HFOkPIRnqQr9czdpytKOLas6LiQPPQ/glY0/
   aNL1FBWl50O9NwJhCfQLnQww9aksICXoxtaU/YFg
   c9g8kTNEvM4rhpTvKDdf0DumDiE5YkFtn9Muc
   G9gbWWVoQZXS9ndRsteEPK9uRGKCeSQrrSrrJ60
   wboGWK/Kb+4e6rTIpYQ=
  </xenc:CipherValue>
 </xenc:CipherData>

 <xenc:ReferenceList>
  <xenc:DataReference
   URI="#EncryptedData-1">
  </xenc:DataReference>
 </xenc:ReferenceList>
</xenc:EncryptedKey>
```

```
            <xenc:EncryptedData
              xmlns:xenc="http://www.w3.org/2001/04/xmlenc#"
              Id="EncryptedData-1"
              Type="http://www.w3.org/2001/04/xmlenc#Element">

              <xenc:EncryptionMethod
                Algorithm="http://www.w3.org/2001/04/xmlenc#tripledes-cbc">
              </xenc:EncryptionMethod>

              <xenc:CipherData>
                <xenc:CipherValue>
                  PRMentWAtN7ix0H5lcL5RMcoJuDZIrXhim3T2
                  dEAxU7C86qUr1CS0yLl/gLN6dq/2Kx/zshYM6ew
                  wTzVIYqm2MUr9Xf6a9Ee+PfPjZomP4m/gxQo3Wq
                  Jvyuu+0pLarMOcMYUrF4fNVxI/A32kGA6I5PX
                  0xnJeBSzca6fryMkaIEOK2RnAh4C/7U4CmgaC2oV
                  XR9/mVwsD3vv5IbyurGGIhsyLeSZKgsLsLvR
                  dCF9NNgDcid9H0tcDR1Wxb99U3iDQGoi3dMrdORB9
                  7s2e9BuQzfVXOBCnSTVYtyenBYw+Zqg+pV8
                  bbYFZPF4TgdfAvHj9REzXEX9CHq6EuT9522D+Z7
                  gJaxQJ2ccmPce0xn8wUx2R91Y73QRFvVm1jpZ
                  Ky7vKzge9S7IdY6xciWXJZhxdF9qmejH7u8DcZ
                  ma+5T2k8rU+qL8YiCxUw==
                </xenc:CipherValue>
              </xenc:CipherData>
            </xenc:EncryptedData>
          </wsse:Security>
        </e:Header>
        <e:Body>
          <wn0:p0 i:type="d:string">Usuario OK.</wn0:p0>
        </e:Body>
</e:Envelope>
```

5.5.2.3 SOAP Response

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<e:Envelope xmlns:d="http://www.w3.org/2001/XMLSchema"
   xmlns:e="http://schemas.xmlsoap.org/soap/envelope/"
   xmlns:i="http://www.w3.org/2001/XMLSchema-instance"
   xmlns:wn0="http://idoox.com/interface"
xmlns:wn1="http://systinet.com/xsd/SchemaTypes/">
   <e:Body>
      <wn1:string_Response i:type="d:string">Usuario OK.</wn1:string_Response>
   </e:Body>
</e:Envelope>
```

Vamos analisar o cabeçalho da requisição SOAP que é a parte que nos interessa. Podemos notar a criação do elemento EncryptedKey que vai definir a chave que será usada para cifrar os dados. O método usado para cifrar a chave é o RSA 1.5. Outro elemento XML criado foi o KeyInfo que possui vários informações sobre a chave, como por exemplo, o tipo, que no caso é X509v3. Após a especifição das informações da chave foi criado outro elemento que é o CipherData que contém a chave cifrada. Após a definição da chave aparece a definição dos dados crifrados que é definida dentro do elemento EncryptedData. Dentro desse elemento é definido o método que será usado para cifrar os dados (EncryptionMethod), que é o triple DES em modo cbc. O próximo elemento são os dados cifrados.

A resposta SOAP é idêntica ao cenário 1.

5.5.3 Cenário 3 - o cliente assina e cifra o corpo da mensagem

Neste cenário o cliente envia o nome do usuário e a senha crifrados e assinados e recebe uma resposta cifrada do serviço web.

5.5.3.1 SOAP Action

```
http://systinet.com/wsdl/teste/security/wssecurity/runtime/lowlevel/server/
WsSecurityService# signedEncryptedBody?KExqYXZhL2xhbmcvU3RyaW5nOylMam
F2YS9sYW5nL1N0cmluZzs=
```

Está chamando o método signedEncryptedBody do serviço web passando como parâmetro um token.

5.5.3.2 SOAP Request

```
<?xml version="1.0" encoding="UTF-8"?>
<e:Envelope
 xmlns:d="http://www.w3.org/2001/XMLSchema"
 xmlns:e="http://schemas.xmlsoap.org/soap/envelope/"
 xmlns:i="http://www.w3.org/2001/XMLSchema-instance"
 xmlns:wn0="http://systinet.com/xsd/SchemaTypes/">

 <e:Header>
  <wsse:Security
   xmlns:wsse="http://schemas.xmlsoap.org/ws/2003/06/secext"
   xmlns:wsu="http://schemas.xmlsoap.org/ws/2003/06/utility"
   e:mustUnderstand="1">

   <xenc:EncryptedKey
    xmlns:xenc="http://www.w3.org/2001/04/xmlenc#"
    Id="EncryptedKey-1">

    <xenc:EncryptionMethod
     Algorithm="http://www.w3.org/2001/04/xmlenc#rsa-1_5">
    </xenc:EncryptionMethod>

    <ds:KeyInfo
```

```
    xmlns:ds="http://www.w3.org/2000/09/xmldsig#">

    <wsse:SecurityTokenReference>
      <wsse:KeyIdentifier
        EncodingType="wsse:Base64Binary"
        ValueType="wsse:X509v3">
        T+DYSrzeRop+C64ebI8TuR4TyCQ=
      </wsse:KeyIdentifier>
    </wsse:SecurityTokenReference>
  </ds:KeyInfo>

  <xenc:CipherData>
    <xenc:CipherValue>
      TX+gl6mTmEmTwp6lFWSHKxtO4lnw0WfZIXwfiEgcp
      yAVJH+6NcTaZcIFvIZ5v9kGRYOJ949QO5n0
      nAqFq8R0dP7D558O0vVS0DNU8H13L2/XDB2BCx1
      jpqhyV+KJWWoF1SKEKTQ5NxlUAyIsTot/NSv7
      Q7Y78Yj3ulj4iO4QL6I=
    </xenc:CipherValue>
  </xenc:CipherData>

  <xenc:ReferenceList>
    <xenc:DataReference
      URI="#EncryptedData-1">
    </xenc:DataReference>
  </xenc:ReferenceList>

</xenc:EncryptedKey>

<wsse:BinarySecurityToken
  EncodingType="wsse:Base64Binary"
  ValueType="wsse:X509v3"
  wsu:Id="SigningSecurityToken-1">
```

```
MIIBkzCB/QIGAPxmR9StMA0GCSqGSIb3DQEBB
AUAMBIxEDAOBgNVBAMTB0dhYnJpZWwwGhcLMDQ
NVBAMTB0dhYnJpZWwwgZ8wDQYJKoZIhvcNAQEB
BQADgY0AMIGJAoGBANU2/MCVaggHUunOtiGgwR
CVjyyY6ngRVk0CDq2+bJR6+PAgHYQgLSQgLXix
msJ8EILLyo/aR6ssHpWazdEnOP6SUAWH3yxoCb
8x2U/oHL9Pktl+Fb6eRJDxtn0V5N2DmE3C1moS
tfJIoSjW0m9wcmwAwVQdUhENgfRebOvoI5bXAg
MBAAEwDQYJKoZIhvcNAQEEBQADgYEAxk0BahTR
tu76thJruBanenqitegHWOFPfdV5kg6ZVFX4JH
hXR5WntYgknt076j+IVeSOl0GZgljBcY9JiJoO
JKoVf5lzNG3A8Km5z8oyIxhe2+KmSUtVRIDF4O
3ZRfq7x2NBcifNibGzkmCCsu6xh3FwzVWFUHXt
dRQK1g7ohww=
</wsse:BinarySecurityToken>

<ds:Signature
 xmlns:ds="http://www.w3.org/2000/09/xmldsig#"
 wsu:Id="Signature-1">

 <ds:SignedInfo>
  <ds:CanonicalizationMethod
   Algorithm="http://www.w3.org/2001/10/xml-exc-c14n#">
  </ds:CanonicalizationMethod>

  <ds:SignatureMethod
   Algorithm="http://www.w3.org/2000/09/xmldsig#rsa-sha1">
  </ds:SignatureMethod>

  <ds:Reference
   URI="#Body-Id-eb9af5a0-a397-11d8-aa21-e6c7e9c1aa21">
   <ds:Transforms>
    <ds:Transform
```

```
        Algorithm="http://www.w3.org/2001/10/xml-exc-c14n#">
       </ds:Transform>
      </ds:Transforms>

      <ds:DigestMethod
       Algorithm="http://www.w3.org/2000/09/xmldsig#sha1">
      </ds:DigestMethod>

      <ds:DigestValue>
       pX48QWQ17fdvrXwhjTfXSVydlfw=
      </ds:DigestValue>
     </ds:Reference>
    </ds:SignedInfo>

    <ds:SignatureValue>
     Ik6s4IScU/1eliE9o0qjUvK8KhtK/q0wFGN3D/
     Owk3x0uCoBCIlrEwMqk9oN7rCfXbnnwqUoflll
     eYyymB7s5hgejud1A9savt76udVcw5UZdp3lAS
     WPnIqMacJJWcI7SV75YR7Emh6uy/Im0QQnJhiR
     JsqCW9gODxSMRbq4YEM=
    </ds:SignatureValue>

    <ds:KeyInfo>
     <wsse:SecurityTokenReference>
      <wsse:Reference
       URI="#SigningSecurityToken-1">
      </wsse:Reference>
     </wsse:SecurityTokenReference>
    </ds:KeyInfo>
   </ds:Signature>

   <wsu:Timestamp
    wsu:Id="Id-TimestampHeader">
```

```
        <wsu:Created>
          2004-05-11T22:09:49Z
        </wsu:Created>
        <wsu:Expires>
          2004-05-11T22:14:49Z
        </wsu:Expires>
      </wsu:Timestamp>

    </wsse:Security>
  </e:Header>
<e:Body
  xmlns:wsu="http://schemas.xmlsoap.org/ws/2003/06/utility"
  wsu:Id="Body-Id-eb9af5a0-a397-11d8-aa21-e6c7e9c1aa21">

  <xenc:EncryptedData
    xmlns:xenc="http://www.w3.org/2001/04/xmlenc#"
    Id="EncryptedData-1"
    Type="http://www.w3.org/2001/04/xmlenc#Content">

    <xenc:EncryptionMethod
      Algorithm="http://www.w3.org/2001/04/xmlenc#tripledes-cbc">
    </xenc:EncryptionMethod>

    <xenc:CipherData>
      <xenc:CipherValue>
        MmljBzOaITbv4pQ+tHyb7bnALIUyt8LkyZgX
        La9H9exMu1LvOJuFZuLzPmKFsogaVlw/iLtnVHl/
        7EDesYzA+CngZE2NeYXzURXbO02stTSycXeag
        nNJsvE8cayC+mpc8yecRTRvmp5rc0/yc6PUiYyD
        YtSJaz5ZE9MKqfWD1+LO9pGx8CEyvn/qMvSIl
        ih8+0qnhfkvxSxFDSA0AgKeKeYy0Em6j2A9vcja
        Yy+5QOd/4tpzVH7ZzqJY7ojzP6VZZXnx5xF/w
        1PXpXN87anaPr9tJeSZ/shP5Kz0pmLJHCFOlmPE
```

```
            u33+uAeyVpX0hyLB2FkdvqORR8jVi9RUylWg
            LNCUYiW/gGWkIG03uouH/yK7Dnfk3ZMHpfPuD0XS
            j+2qvXTosJ6n3fIIGCXffVVQZUdzrlU6nam
            glrscg6JO9ws=
          </xenc:CipherValue>
        </xenc:CipherData>
      </xenc:EncryptedData>
    </e:Body>
</e:Envelope>
```

A definição da chave que vai ser usada para cifrar os dados é a mesma que no cenário 2, dispensando comentários. Após este elemento começam a aparecer as diferenças entre o cenário 2 e 3. É criado um token binário que vai ser usado na assinatura da mensagem, através do elemento BinarySecurityToken. Após a criação do token é iniciada a criação da assinatura em si com o elemento Signature. Dentro do elemento Signature temos várias configurações referentes à assinatura, que são:

- SignatureMethod: especifica o método usado na assinatura, no caso, RSA-SHA1.
- DigestMethod: especifica o método usado para criar o resumo criptográfico, que no caso foi o SHA-1.
- DigestValue: valor do resumo criptográfico.

Depois de especificadas as configurações da assinatura podemos ver o valor da assinatura que está no elemento SignatureValue.

Ainda dentro da assinatura temos informações sobre a chave usada, que como podemos ver foi usado o token binário criado anteriormente.

Também temos um outro elemento, o Timestamp, igualmente importante para a assinatura, pois ele possui a data de criação e de vencimento da assinatura.

Desse modo termina o cabeçalho da requisição SOAP onde foram criados a chave e a assinatura necessários para cifrar e assinar o documento XML.

Podemos verificar que o corpo da mensagem foi cifrado usando o triple DES em modo cbc. Desse modo temos também o corpo da mensagem cifrado.

5.5.3.3 SOAP Response

```
<?xml version="1.0" encoding="UTF-8"?>
<e:Envelope
 xmlns:d="http://www.w3.org/2001/XMLSchema"
 xmlns:e="http://schemas.xmlsoap.org/soap/envelope/"
 xmlns:i="http://www.w3.org/2001/XMLSchema-instance"
 xmlns:wn0="http://idoox.com/interface"
 xmlns:wn1="http://systinet.com/xsd/SchemaTypes/">

 <e:Header>
  <wsse:Security
   xmlns:wsse="http://schemas.xmlsoap.org/ws/2003/06/secext"
   xmlns:wsu="http://schemas.xmlsoap.org/ws/2003/06/utility"
   e:mustUnderstand="1">

   <xenc:EncryptedKey
    xmlns:xenc="http://www.w3.org/2001/04/xmlenc#"
    Id="EncryptedKey-1">

    <xenc:EncryptionMethod
     Algorithm="http://www.w3.org/2001/04/xmlenc#rsa-1_5">
    </xenc:EncryptionMethod>

    <ds:KeyInfo
     xmlns:ds="http://www.w3.org/2000/09/xmldsig#">

     <wsse:SecurityTokenReference>
      <wsse:KeyIdentifier
       EncodingType="wsse:Base64Binary"
       ValueType="wsse:X509v3">
       iFdf9f3Uj1O11NxzwG1jfayzXc8=
      </wsse:KeyIdentifier>
```

```
    </wsse:SecurityTokenReference>
  </ds:KeyInfo>

  <xenc:CipherData>
    <xenc:CipherValue>
      d7AJttRKc+uWWm3tON23DdY3dxtFBcBTxN+e/
      Tw+BTwybgfgcV77RJjrcYHE87iNRygh4oTzadpJ
      b4oq6An5/i7hCsHlHluMBX6SxIOao9O0FsL2Ht
      M/Fn6YsGB5Pm0lzknvOBeH8/0rBl+wRH3p8Ddg
      5GNnsi01m/0yy7c+3pY=
    </xenc:CipherValue>
  </xenc:CipherData>

  <xenc:ReferenceList>
    <xenc:DataReference
      URI="#EncryptedData-1">
    </xenc:DataReference>
  </xenc:ReferenceList>

</xenc:EncryptedKey>

<wsse:BinarySecurityToken
  EncodingType="wsse:Base64Binary"
  ValueType="wsse:X509v3"
  wsu:Id="SigningSecurityToken-1">
  MIIBsjCCARsCBgD8ZkeN4DANBgkqhkiG9w0BAQ
  QFADAhMR8wHQYDVQQDExZXc1NlY3VyaXR5VGVz
  dGVTZXJ2aWNlMBoXCzA0MDUwODIwMjJaFwswNj
  A1MDgyMDIyWjAhMR8wHQYDVQQDExZXc1NlY3Vy
  aXR5VGVzdGVTZXJ2aWNlMIGfMA0GCSqGSIb3DQ
  EBAQUAA4GNADCBiQKBgQCnjdPYPSguZahnUlj0
  lIUpP+tz/O363qKGF7QX8OmE+3y6PnrZJrLzMK
  bxVZFHK2kPF1BAsvAGhwe+xNdEdy2IwYpskVd3
```

```
  4IZIhpzoECHPTBkqBf2GQaWTDhZ6neiALklO2Z
  EW4znImY4QN/2NyfG2o/9mPK8+z0eYBEIOmM00
  hwIDAQABMA0GCSqGSIb3DQEBBAUAA4GBAEmT3i
  sCVE6mJrX8fFRBLP2ovM2UCyfe8Mazph0/Z5Uy
  4pGeAp1nfzyNIbyjKQ4punRQu7X8j4G6aVSfeI
  gOYLyQO1fcLos1nORp/8ivC/GVpoizYy1EiK1m
  r0t6mQLQ91bj7PNsXHveBPMkqNC7++EWp1oKaO
  WtV7hdLdEUbWFo
</wsse:BinarySecurityToken>

<ds:Signature
  xmlns:ds="http://www.w3.org/2000/09/xmldsig#"
  wsu:Id="Signature-1">

  <ds:SignedInfo>
    <ds:CanonicalizationMethod
      Algorithm="http://www.w3.org/2001/10/xml-exc-c14n#">
    </ds:CanonicalizationMethod>

    <ds:SignatureMethod
      Algorithm="http://www.w3.org/2000/09/xmldsig#rsa-sha1">
    </ds:SignatureMethod>

    <ds:Reference
      URI="#Body-Id-ec9c8720-a397-11d8-8b79-d1d117818b79">

      <ds:Transforms>
        <ds:Transform
          Algorithm="http://www.w3.org/2001/10/xml-exc-c14n#">
        </ds:Transform>
      </ds:Transforms>

      <ds:DigestMethod
```

```
          Algorithm="http://www.w3.org/2000/09/xmldsig#sha1">
        </ds:DigestMethod>

        <ds:DigestValue>
          r/Bu9JJZgZ8zlnlfgdxqZ5+rXdQ=
        </ds:DigestValue>

      </ds:Reference>
    </ds:SignedInfo>

    <ds:SignatureValue>
      fKSZRp8bIwxYEddKkLoeK+SteAggkWs//gHT
      Mxw3cU+Lk6E1xIhYUtY0bAF2Hotimw3BznfBpwvu
      l43guE7MSnxkmIPMh2bHfSJIdCiXduOAxEGN
      ZmBIUVXRKmoMXfWFJV0JRaCxxZq5CJxS3/QHqhuk
      nDoY3JNtNjt/taogMkQ=
    </ds:SignatureValue>

    <ds:KeyInfo>
      <wsse:SecurityTokenReference>
        <wsse:Reference
          URI="#SigningSecurityToken-1">
        </wsse:Reference>
      </wsse:SecurityTokenReference>
    </ds:KeyInfo>
  </ds:Signature>
 </wsse:Security>
</e:Header>

<e:Body
 xmlns:wsu="http://schemas.xmlsoap.org/ws/2003/06/utility"
 wsu:Id="Body-Id-ec9c8720-a397-11d8-8b79-d1d117818b79">
```

```
        <xenc:EncryptedData
          xmlns:xenc="http://www.w3.org/2001/04/xmlenc#"
          Id="EncryptedData-1"
          Type="http://www.w3.org/2001/04/xmlenc#Content">

          <xenc:EncryptionMethod
            Algorithm="http://www.w3.org/2001/04/xmlenc#tripledes-cbc">
          </xenc:EncryptionMethod>

          <xenc:CipherData>
            <xenc:CipherValue>
              xZ5UIXUOx7Zx/gzaobrI3qA3DhV2+NYg5oUSjZ
              Pe2W1rbame7n8PkMNPys9R8HiI84iMVpsausPo
              BKMNzO7PwMoYpCusiw0hmKitf5DALt81i1IHH6
              Vz956ntqJmKH9QJICV6Sw+w3rhM1orEkvqNIt+
              DgWEDPxT+jTBkb5lXlPnchXX6TYfEK02iei0h1
              LqWTBZv9lapZndBrWaRP3XPa6iterDZwztnKJV
              emijSsPPpCvKmryLP1oNYLtGpQH1kkFvVHjX9h
              0oyzabYRI2oC7q9eenlkB3esj+YXaWQBgFvJ06
              OA1CXn1ci4K6Bv7P4SmSZ+hE2yyDv3SxJCrjrM
              hlVwv6mH+UUt6hqROsigGCkU3iwI7of1+niXHK
              vD3NtRCGXyS1Z85iKEdo3SxvScA305/spMRw11
              fZgQwOa1yX/OTGoPjUd0OyjB3TPzUoyE/PmlbK
              umt/+F9zbeGzTix/VbvT52i8JDt75MTYrn7Hlz
              urlXXbikTlLM+zkAjr
            </xenc:CipherValue>
          </xenc:CipherData>
        </xenc:EncryptedData>
      </e:Body>
</e:Envelope>
```

Com relação a resposta SOAP no cenário 3 podemos verificar que é completamente diferente do cenário 1 e 2. Em contra partida, ela é idêntica a requisição que a originou, com a chave e assinatura definidos no cabeçalho e o corpo da mensagem cifrado.

6 Conclusão

O WS Security oferece uma estrutura extensa para melhorar a segurança dos Web Services. Em contrapartida, ele não oferece toda a segurança necessária que um sistema deveria ter. Um exemplo disso é que o WS Security não possui métodos para estabelecer uma relação confiável entre servidor e cliente. No nosso protótipo essa relação foi criada usando-se características do servidor WASP para a criação de certificados digitais e pares de chave pública e privada, e não através do WS Security. Normalmente a especificação resolve problemas que acontecem após estabelecida uma relação confiável entre as partes envolvidas.

Uma vantagem do WS Security é o suporte a vários tipos de protocolos e tokens. Entretanto, ele não oferece informações de como deixar esses protocolos seguros.

O WS Security é um guia de como assegurar que mensagens SOAP sejam seguras, mas também ele possui furos de segurança em outras áreas como disponibilidade.

Basicamente, o WS Security suporta autenticação, confidencialidade e integridade de uma maneira limitada. Por exemplo, ele não possui suporte para se revogar certificados de autenticação.

O timestamp, no WS Security, não possui métodos para a sincronização do tempo, o que aumenta o risco de ocorrer ataques do tipo replay. Essa falha é uma desvantagem no que diz respeito ao suporte a Kerberos, pois ele requer sincronização de tempo.

Outro ponto em que o WS Security falha é no que diz respeito a listas de acesso, pois ele não pode checar quais tipos de acesso um usuário possui ou se o certificado de um usuário foi revogado ou expirado.

Usar a cifragem XML e deixar partes do documento decifradas pode fazer com que usuários maliciosos tenham acesso a informações sigilosas.

Outro problema do WS Security é a falta de suporte a controle de versão e de que maneira vírus podem ser prevenidos.

Como mencionado anteriormente, o WS Security deve ser complementado com outras especificações. Por exemplo, para dar suporte ao estabelecimento de relações de confiança ele deve ser complementado com o WS-Trust e o WS-SecureConversation para dar suporte a mecanismos de handshake. Mas o problema é que o WS-Trust e o WS-SecureConversation ainda estão em desenvolvimento.

7 Sugestões de trabalhos futuros

Como podemos notar, o WS Security é bastante dependente das especificações que ainda virão para complementá-lo. Por isso, torna-se interessante analisar essas especificações uma por uma, mas também todas juntas para verificar se elas oferecem a segurança desejada. Como o WS Security ainda é uma especificação, seria interessante analisá-la após tornar-se um padrão.

8 Referências Bibliográficas

JENNINGS, Roger. Dig Into WS-Security With the WSDK. [S.l]. Fawcette Technical Publications, 20 set. 2002. Disponível em: <http://www.fawcette.com/xmlmag/2002_09/online/webservices_rjennings_09_20_02/default_pf.asp>.

GALLI, Peter. Spec Secures Web Services Apps. New Orleans. eWeek, 11 abr. 2002. Disponível em: <http://www.eweek.com/article2/0,3959,50620,00.asp>.

APSHANKAR, Kapil. WS-Security: Security for Web Services.[S.l]. Web Services Architect. 24 jul. 2002. Disponível em: <http://www.webservicesarchitect.com/content/articles/apshankar04.asp>.

OASIS. Web Services Architecture Working Group. 2001. Disponível em: < http://www.w3.org/2002/ws/arch/>.

A. Shunn. Managed Security: Build It Right the First Time. 2002. Disponível em: <http://archive.devx.com/security/articles/WebServices/partI/AS0102-1.asp>.

B.Atkinson, G.Della-Libera, S.Hada, M. Hondo, P.Hallam-Baker, C.Kaler, J.Klein, B.LaMacchia, P.Leach, J.Manferdelli, H. Maruyama, A. Nadalin, N. Nagaratnam, H. Prafullchandra, J. Shewchuk, D. Simon. Web Services Security (WS-Security). 05 abr. 2002. Disponível em: <http://msdn.microsoft.com/library/default.asp?url=/library/enus/ dnglobspec/html/ws-security.asp>.

RSA Security. What is SSL?. 2002. Disponível em: <http://www.rsasecurity.com/rsalabs/faq/5-1-2.html>.

RSA Security. What is IPSec?. 2002. Disponível em: < http://www.rsasecurity.com/rsalabs/node.asp?id=2295>.

Vordel. Web services security. 2002. Disponível em: <http://www.vordel.com>.

D.K. Taft. WS-Security Spec Sent to OASIS. 27 jun. 2002. Disponível em: <http://www.eweek.com/article2/0,3959,290627,00.asp>.

D. Eastlake, J. Reagle. XML-Signature Syntax and Processing. 2002. Disponível em: < http://rfc3075.x42.com/ >.

Dillaway, Fox, Imamura, LaMacchia, Maruyama, Schaad, and Simon. XML Encryption Syntax and Processing. 2001. Disponível em: < http://www.w3.org/Encryption/2001/05/11-proposal.html >.

J.Boyer, PureEdge Solutions Inc. Canonical XML Version 1.0. 2001. Disponível em: < http://www.w3.org/TR/xml-c14n >.

E.X. DeJesus. Secure Your Web Services Applications. Jun. 2002. Disponível em: <http://www.fawcette.com/dotnetmag/2002_06/magazine/features/edejesus/>.

A. Yang. XML Web Services Security Issues. 10 abr. 2002. Disponível em: <http://www.xwss.org/articlesThread.jsp?forum=34&thread=648>.

R.Shirey. Internet Security Glossary. Mai. 2000. Disponível em: <http://www.faqs.org/rfcs/rfc2828.html>.

A.Jøsang, D.Povey, A.Ho. What You See Is Not Always What You Sign. 2002. Disponível em: <http://citeseer.nj.nec.com/535070.html>.

Zaw-Sing Su. A SPECIFICATION OF THE INTERNET PROTOCOL (IP) TIMESTAMP OPTION. 1981. Disponível em: <http://www.faqs.org/rfcs/rfc781.html>.

Babylon, 2002, Disponível em: <http://www.babylon.com>.

Westbridge. XML Web Services Security Glossary. 2002. Disponível em: <http://www.westbridgetech.com/downloads/XWSS.orgWebServicesSecurityGlossary.pdf >.

Apêncide I: Arquivo run.bat

```
@ECHO OFF


SETLOCAL
CALL .\..\..\env.bat
SET BUILD_DIR=build
SET SRC_DIR=src
SET CLIENT_CLASSDIR=.\build\client\classes
SET SERVER_CLASSDIR=.\build\server\classes
SET EXPORTED_SERVICE_CERT=.\build\service.crt
SET EXPORTED_CLIENT_CERT=.\build\client.crt
SET SECURITY_TOOLS=%WASP_HOME%\lib\security_tools.jar;
SET SECURITY_PROVIDERS_CLIENT=%WASP_HOME%\lib\security_providers_client.jar;
SET SECURITY2_NG=%WASP_HOME%\lib\security2-ng.jar;
SET COMPILE_SERVER_CLASSPATH=%COMPILE_WITH_WASP_CLASSPATH%;%SECURITY2_NG%;%SECURITY_
SET COMPILE_CLIENT_CLASSPATH=%COMPILE_WITH_WASP_CLASSPATH%;%SECURITY_PROVIDERS_CLIENT
SET RUN_TOOLS_CLASSPATH=%SECURITY_PROVIDERS_CLIENT%;%SECURITY_TOOLS%
SET RUN_SERVER_CLASSPATH=%WASP_HOME%\lib\wasp.jar;%SECURITY_PROVIDERS_CLIENT%;%SERVER_C
SET RUN_CLIENT_CLASSPATH=%WASP_HOME%\lib\wasp.jar;%SECURITY_PROVIDERS_CLIENT%;%CLIENT_CLA
SET RUN_CLIENT_DD_CLASSPATH=%RUN_CLIENT_CLASSPATH%;%BUILD_DIR%\client.jar
SET WSDL_URL=%SERVER_URL%/teste/security/ws-security/WsSecurityService/wsdl
SET SERVICE_IDENTITY=WsSecurityTesteService
SET SERVICE_IDENTITY_PASSWORD=changeit
SET CLIENT_IDENTITY=Gabriel
SET CLIENT_IDENTITY_PASSWORD=asdf
SET USER="%2"
SET PASSWORD="%3"
SET PROVIDER=%4
SET TESTE_DIRNAME=security/ws-security




GOTO :checkComponentsok
:checkComponents

if not exist "%WASP_HOME%\lib\security_providers_client.jar" ( set errorlevel=5
  ECHO ************************ Error ************************
ECHO The component security_providers_client.jar" is missing.
ECHO If you want to run this application, please first install it.
ECHO ************************************************************
  GOTO :EOF
)
if not exist "%WASP_HOME%\lib\security_tools.jar" ( set errorlevel=5
  ECHO ************************ Error ************************
ECHO The component security_tools.jar" is missing.
ECHO If you want to run this application, please first install it.
ECHO ************************************************************
```

```
  GOTO :EOF
)
if not exist "%WASP_HOME%\lib\security2-ng.jar" ( set errorlevel=5
  ECHO ************************ Error ************************
ECHO The component security2-ng.jar" is missing.
ECHO If you want to run this application, please first install it.
ECHO ********************************************************
  GOTO :EOF
)

GOTO :EOF
:checkComponentsok



GOTO :helpok
:help

ECHO.
ECHO     Description: This is WS-Security application
ECHO.
ECHO     Requirements: Run the create_identities target to create identities used in this application first.
ECHO                   Please note that WASP server must be running.
ECHO.
ECHO     Deployment usage:
ECHO       create_identities creates identities
ECHO       make_service      compiles service sources and creates service package
ECHO       deploy_service    deploys service package on to WASP server
ECHO       make_client       creates client classes from service's WSDL
ECHO       run_client        runs client with persistent configuration
ECHO       undeploy_service  undeploys service package from WASP server
ECHO.
ECHO     Runtime usage:
ECHO       create_identities creates identities
ECHO                         (if you have not run it yet - see Requirements)
ECHO     a) low-level API server and client
ECHO       make_server_lo    compiles server and service sources
ECHO       run_server_lo     runs server and publishes the service
ECHO       make_client_lo    creates client classes from service's WSDL
ECHO       run_client_lo     runs client
ECHO.
ECHO     Specific client usage:
ECHO       spy_client       runs the client with persistent configuration with SOAPSpy related properties
ECHO       spy_client_lo    runs the low-level API client with SOAPSpy related properties

GOTO :EOF
:helpok


GOTO :create_identitiesok
:create_identities
```

```
CALL :checkComponents
if %errorlevel%==5 goto :EOF
REM Criando as identidades
ECHO Criando a identidade do servico no servidor...
MKDIR %BUILD_DIR%
%JAVA_HOME%\bin\java "-Djava.security.auth.login.config=%WASP_HOME%/conf/jaas.config" -cp "%RUN_TOOLS_CL
ECHO Exportando o certificado da identidade do servico para um arquivo...
%JAVA_HOME%\bin\java "-Djava.security.auth.login.config=%WASP_HOME%/conf/jaas.config" -cp "%RUN_TOOLS_CL
ECHO Adicionando a identidade certificada do servico na base de chaves do cliente para garantir que ele seja confiável...
%JAVA_HOME%\bin\java "-Djava.security.auth.login.config=%WASP_HOME%/conf/jaas.config" -cp "%RUN_TOOLS_CL
ECHO Criando a identidade do cliente...
%JAVA_HOME%\bin\java "-Djava.security.auth.login.config=%WASP_HOME%/conf/jaas.config" -cp "%RUN_TOOLS_CL
ECHO Exportando o certificado da identidade do cliente para um arquivo...
%JAVA_HOME%\bin\java "-Djava.security.auth.login.config=%WASP_HOME%/conf/jaas.config" -cp "%RUN_TOOLS_CL
ECHO Adicionando a identidade certificada do cliente na base de chaves do servidor para garantir que ele seja confiável..
%JAVA_HOME%\bin\java "-Djava.security.auth.login.config=%WASP_HOME%/conf/jaas.config" -cp "%RUN_TOOLS_CL
ECHO Certificado e senha designados ao cliente pelo servidor...
%JAVA_HOME%\bin\java "-Djava.security.auth.login.config=%WASP_HOME%/conf/jaas.config" "-Dwasp.policy.xml.file=%
%EXPORTED_CLIENT_CERT% --username %USER% --password %PASSWORD%
ECHO Adicionando a identidade certificada do cliente na base de chaves do cliente para garantir que ele seja confiável...
%JAVA_HOME%\bin\java "-Djava.security.auth.login.config=%WASP_HOME%/conf/jaas.config" -cp "%RUN_TOOLS_CL


GOTO :EOF
:create_identitiesok


GOTO :make_serviceok
:make_service

CALL :checkComponents
if %errorlevel%==5 goto :EOF
REM Let's compile service sources at first.
ECHO Compilando as classes do servico...
MKDIR %SERVER_CLASSDIR%
%JAVA_HOME%\bin\javac -classpath "%COMPILE_SERVER_CLASSPATH%" -sourcepath %SRC_DIR% -d %SERVER_

ECHO Gerando o WSDL...
REM OLDCLASSPATH statements are present only in beta release
SET OLDCLASSPATH=%CLASSPATH%

SET CLASSPATH=%SERVER_CLASSDIR%
CALL Java2WSDL --classpath "%SERVER_CLASSDIR%" -d %BUILD_DIR% --soap-binding-style document --soap-encod
http://hanauer.com/wsdl/teste/security/wssecurity/server/=WsSecurityService.wsdl teste.security.wssecurity.persistent.serv
SET CLASSPATH=%OLDCLASSPATH%


ECHO Criando o pacote...
CALL WaspPackager -p WsSecurityTesteService -o %BUILD_DIR%\service.jar --force -s %SERVER_CLASSDIR% --dd d
```

```
GOTO :EOF
:make_serviceok


GOTO :make_server_look
:make_server_lo

CALL :checkComponents
if %errorlevel%==5 goto :EOF
REM Let's make server sources.
ECHO Compilando as classes do servidor...
MKDIR %SERVER_CLASSDIR%
%JAVA_HOME%\bin\javac -classpath "%COMPILE_SERVER_CLASSPATH%" -sourcepath %SRC_DIR% -d %SERVER_


GOTO :EOF
:make_server_look


GOTO :run_server_look
:run_server_lo

CALL :checkComponents
if %errorlevel%==5 goto :EOF
ECHO Rodando o servidor...
start "hanauer WASP server" "%JAVA_HOME%\bin\java" "-Djava.security.auth.login.config=%WASP_HOME%/conf/jaas.c


GOTO :EOF
:run_server_look


GOTO :deploy_serviceok
:deploy_service

CALL :checkComponents
if %errorlevel%==5 goto :EOF
REM We're trying to deploy compiled service on to running WASP server.
ECHO Adicionando servico no servidor...
CALL Deploy -j %BUILD_DIR%/service.jar -t %SERVER_URL% --redeploy --username %USER% --password %PASSWO
ECHO Associando a identidade ao endpoint...
%JAVA_HOME%\bin\java "-Djava.security.auth.login.config=%WASP_HOME%/conf/jaas.config" -cp "%RUN_TOOLS_CL
%SERVICE_IDENTITY_PASSWORD% --username %USER% --password %PASSWORD%


GOTO :EOF
:deploy_serviceok


GOTO :make_clientok
:make_client
```

```
CALL :checkComponents
if %errorlevel%==5 goto :EOF
ECHO Compilando o WSDL para código fonte Java...
CALL WSDL2Java --output-directory %BUILD_DIR%\src --url %WSDL_URL% -p teste.security.wssecurity.persistent.client

ECHO Compilando os stubs...
MKDIR %CLIENT_CLASSDIR%
%JAVA_HOME%\bin\javac -classpath "%COMPILE_CLIENT_CLASSPATH%" -sourcepath %BUILD_DIR%\src -d %CLIEN

ECHO Compilando o cliente...
MKDIR %CLIENT_CLASSDIR%
%JAVA_HOME%\bin\javac -classpath "%COMPILE_CLIENT_CLASSPATH%" -sourcepath .\src -d %CLIENT_CLASSDIR

ECHO Copying mapping file to client classpath...
COPY %BUILD_DIR%\src\teste\security\wssecurity\persistent\client\WsSecurityService.xmap %CLIENT_CLASSDIR%\tes

ECHO Criando o pacote do cliente...
CALL WaspPackager -C -o %BUILD_DIR%\client.jar --force --dd dd/client.xml


GOTO :EOF
:make_clientok


GOTO :make_client_look
:make_client_lo

CALL :checkComponents
if %errorlevel%==5 goto :EOF
ECHO Compilando o WSDL para código fonte Java...
CALL WSDL2Java --output-directory %BUILD_DIR%\src --url %WSDL_URL% -p teste.security.wssecurity.runtime.lowleve

ECHO Compilando os stubs...
MKDIR %CLIENT_CLASSDIR%
%JAVA_HOME%\bin\javac -classpath "%COMPILE_CLIENT_CLASSPATH%" -sourcepath %BUILD_DIR%\src -d %CLIEN

ECHO Compilando o cliente...
MKDIR %CLIENT_CLASSDIR%
%JAVA_HOME%\bin\javac -classpath "%COMPILE_CLIENT_CLASSPATH%" -sourcepath .\src -d %CLIENT_CLASSDIR

ECHO Copying mapping file to client classpath...
COPY %BUILD_DIR%\src\teste\security\wssecurity\runtime\lowlevel\client\WsSecurityService.xmap %CLIENT_CLASSDI


GOTO :EOF
:make_client_look


GOTO :run_clientok
:run_client
```

```
CALL :checkComponents
if %errorlevel%==5 goto :EOF
ECHO Rodando o cliente...
%JAVA_HOME%\bin\java "-Djava.security.auth.login.config=%WASP_HOME%/conf/jaas.config" "-Dhanauer.teste.server.

GOTO :EOF
:run_clientok

GOTO :run_client_look
:run_client_lo

CALL :checkComponents
if %errorlevel%==5 goto :EOF
ECHO Rodando o cliente...
%JAVA_HOME%\bin\java "-Djava.security.auth.login.config=%WASP_HOME%/conf/jaas.config" "-Dhanauer.teste.server.

GOTO :EOF
:run_client_look

GOTO :spy_clientok
:spy_client

CALL :checkComponents
if %errorlevel%==5 goto :EOF
ECHO Rodando o cliente em modo spy para pegarmos as mensagens SOAP...
%JAVA_HOME%\bin\java "-Djava.security.auth.login.config=%WASP_HOME%/conf/jaas.config" "-Dhttp.proxyHost=%PR

GOTO :EOF
:spy_clientok

GOTO :spy_client_look
:spy_client_lo

CALL :checkComponents
if %errorlevel%==5 goto :EOF
ECHO Rodando o cliente em modo spy para pegarmos as mensagens SOAP...
%JAVA_HOME%\bin\java "-Djava.security.auth.login.config=%WASP_HOME%/conf/jaas.config" "-Dhttp.proxyHost=%PR

GOTO :EOF
:spy_client_look

GOTO :undeploy_serviceok
```

```
:undeploy_service

CALL :checkComponents
if %errorlevel%==5 goto :EOF
ECHO Removendo servico do servidor...
CALL Undeploy -j %BUILD_DIR%/service.jar --target %SERVER_URL% --username %USER% --password %PASSWOR


GOTO :EOF
:undeploy_serviceok


GOTO :cleanok
:clean

rmdir /S /Q .\build

GOTO :EOF
:cleanok



IF NOT "%1" == "" CALL :%*
IF "%1" == "" CALL :help
```

Apêncide II: Arquivo client.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<package client-package="true" name="WsSecurityTesteClient"
    targetNamespace="http://hanauer.com/package/teste/security/wssecurity/client"
    xmlns:tns="http://hanauer.com/package/teste/security/wssecurity/client"
    xmlns:ns0="http://hanauer.com/wsdl/teste/security/wssecurity/server/"
    xmlns="http://systinet.com/wasp/package/1.2"
    version="4.6">

    <service-client port-name="ns0:WsSecurityService"
        initiating-security-provider="WS-Security">

        <securityProviderPreferences xmlns="http://systinet.com/wasp/package/wssecurity/1.0"
            localName="WS-Security">

            <!-- tokens externos -->
            <externalSecurityTokens>
                <!-- token para a encriptação da mensagem (cenario 2, 3) -->
                <securityToken type="X509v3" wsuId="EncryptingSecurityToken-1">
                    <property propertyName="alias" propertyValue="CN=WsSecurityTesteService"/>
                </securityToken>
                <!-- token usado para decifrar a resposta (cenario 3) -->
                <securityToken type="X509v3"/>
            </externalSecurityTokens>

            <!-- cenario 1 -->
            <securedMessage>
                <methodName>usernameToken</methodName>
                <messageConf>
                    <securityToken type="Username">
                        <wsuId>SecurityToken-1</wsuId>
                        <order>0</order>
                        <property propertyName="PasswordType" propertyValue="PasswordText"/>
                        <property propertyName="noNonceAndCreated" propertyValue="true"/>
                    </securityToken>
                </messageConf>
            </securedMessage>

            <!-- cenario 2 mensagem segura (username cifrado) -->
            <securedMessage>
                <!-- metodo -->
                <methodName>encryptedUsernameToken</methodName>
                <messageConf>
                    <!-- username token -->
                    <securityToken type="Username">
                        <wsuId>SecurityToken-1</wsuId>
                        <order>10</order>
                        <property propertyName="PasswordType" propertyValue="PasswordText"/>
                    </securityToken>
```

```
        <!-- chave para cifrar -->
        <encryptedKey>
           <wsuId>EncryptedKey-1</wsuId>
           <encryptionMethodAlgorithm>http://www.w3.org/2001/04/xmlenc#rsa-1_5</encryptionMethodAlgorithm>
           <keyInfo securityTokenId="EncryptingSecurityToken-1"
              securityTokenMode="keyidentifier"/>
           <reference URI="#EncryptedData-1"/>
           <order>50</order>
        </encryptedKey>

        <!-- dados cifrados -->
        <encryptedData>
           <wsuId>EncryptedData-1</wsuId>
           <encryptionMethodAlgorithm>http://www.w3.org/2001/04/xmlenc#tripledes-cbc</encryptionMethodAlgorithm
           <encryptionTargetId>SecurityToken-1</encryptionTargetId>
           <order>15</order>
        </encryptedData>
     </messageConf>
  </securedMessage>

  <!-- cenario 3 mensagem segura (corpo da mensagem assinado e cifrado) -->
  <securedMessage>
     <!-- metodo -->
     <methodName>signedEncryptedBody</methodName>
     <messageConf createTimestampHeader="true">

        <!-- token para assinatura -->
        <securityToken type="X509v3">
           <wsuId>SigningSecurityToken-1</wsuId>
           <order>10</order>
        </securityToken>

        <!-- assinatura do corpo -->
        <signature>
           <wsuId>Signature-1</wsuId>
           <signatureMethod>http://www.w3.org/2000/09/xmldsig#rsa-sha1</signatureMethod>
           <keyInfo securityTokenId="SigningSecurityToken-1"
              securityTokenMode="reference"/>
           <signBody>true</signBody>
           <order>0</order>
        </signature>

        <!-- chave para cifrar -->
        <encryptedKey>
           <wsuId>EncryptedKey-1</wsuId>
           <encryptionMethodAlgorithm>http://www.w3.org/2001/04/xmlenc#rsa-1_5</encryptionMethodAlgorithm>
           <keyInfo securityTokenId="EncryptingSecurityToken-1"
              securityTokenMode="keyidentifier"/>
           <reference URI="#EncryptedData-1"/>
           <order>25</order>
```

```
            </encryptedKey>

            <!-- dados cifrados -->
            <encryptedData>
               <wsuId>EncryptedData-1</wsuId>
               <encryptionMethodAlgorithm>http://www.w3.org/2001/04/xmlenc#tripledes-cbc</encryptionMethodAlgorithm
               <encryptBody>true</encryptBody>
               <encryptElementContent>true</encryptElementContent>
               <order>20</order>
            </encryptedData>
         </messageConf>
      </securedMessage>

   </securityProviderPreferences>

</service-client>

</package>
```

Apêncide III: Arquivo package.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<package name="WsSecurityTeste"
   targetNamespace="http://hanauer.com/package/teste/security/wssecurity/server"
   xmlns:tns="http://hanauer.com/package/teste/security/wssecurity/server"
   xmlns:wsdlns="http://hanauer.com/wsdl/teste/security/wssecurity/server/"
   xmlns:security_providers="http://systinet.com/wasp/app/security_providers"
   xmlns="http://systinet.com/wasp/package/1.2"
   version="4.6">

   <dependency ref="security_providers:security_providers" version="1.0"/>

   <service-instance name="WsSecurityService_inst"
      implementation-class="teste.security.wssecurity.persistent.server.WsSecurityService"/>

   <service-endpoint name="WsSecurityService"
      path="/teste/security/ws-security/WsSecurityService"
      service-instance="tns:WsSecurityService_inst"
      accepting-security-providers="WS-Security">
      <wsdl uri="WsSecurityService.wsdl"
         service="wsdlns:WsSecurityService"/>

      <securityProviderPreferences localName="WS-Security"
         xmlns="http://systinet.com/wasp/package/wssecurity/1.0">

         <!-- aceitando Username tokens (cenario 1) -->
         <noUsernameTokenValidation>true</noUsernameTokenValidation>

         <!-- adicionando o token X509 para decifrar os dados (cenario 2, 3) -->
         <externalSecurityTokens>
             <securityToken type="X509v3"/>
          </externalSecurityTokens>

         <!-- especificacao da classe que irá validar a assinatura e a resposta cifrada no cenario 3 -->
         <validatorClassName>teste.security.wssecurity.persistent.server.Scenario3IncomingValidator</validatorClassNam

         <!-- cenario 1 mansagem de resposta (sem segurança) -->
         <securedMessage>
            <methodName>usernameToken</methodName>
            <messageConf noSecurityHeader="true"/>
         </securedMessage>

         <!-- cenario 2 mensagem de resposta (sem segurança) -->
         <securedMessage>
            <methodName>encryptedUsernameToken</methodName>
            <messageConf noSecurityHeader="true"/>
         </securedMessage>

         <!-- cenario 3 mensagem de reposta (corpo assinado e cifrado) -->
```

```
        <securedMessage>
          <!-- metodo -->
          <methodName>signedEncryptedBody</methodName>
          <messageConf>
            <!-- token para assinatura -->
            <securityToken type="X509v3">
              <wsuId>SigningSecurityToken-1</wsuId>
              <order>10</order>
            </securityToken>

            <!-- assinando o corpo da mensagem -->
            <signature>
              <wsuId>Signature-1</wsuId>
              <signatureMethod>http://www.w3.org/2000/09/xmldsig#rsa-sha1</signatureMethod>
              <keyInfo securityTokenId="SigningSecurityToken-1"
                 securityTokenMode="reference"/>
              <signBody>true</signBody>
              <order>0</order>
            </signature>

            <!-- chave para cifrar -->
            <encryptedKey>
              <wsuId>EncryptedKey-1</wsuId>
              <encryptionMethodAlgorithm>http://www.w3.org/2001/04/xmlenc#rsa-1_5</encryptionMethodAlgorithm>
              <keyInfo securityTokenId="EncryptingBinaryToken-1"
                 securityTokenMode="keyidentifier"/>
              <reference URI="#EncryptedData-1"/>
              <order>25</order>
            </encryptedKey>

            <!-- dados cifrados -->
            <encryptedData>
              <wsuId>EncryptedData-1</wsuId>
              <encryptionMethodAlgorithm>http://www.w3.org/2001/04/xmlenc#tripledes-cbc</encryptionMethodAlgorithm
              <encryptBody>true</encryptBody>
              <encryptElementContent>true</encryptElementContent>
              <order>20</order>
            </encryptedData>
          </messageConf>
        </securedMessage>

      </securityProviderPreferences>

    </service-endpoint>

</package>
```

Apêncide IV: Arquivo WsSecurityService.java

```
package teste.security.wssecurity.runtime.lowlevel.server;

import org.systinet.wasp.webservice.Current;
import org.systinet.wasp.security.ws.conf.SecurityTokenConf;
import org.systinet.wasp.security.ws.conf.MessageConf;
import org.systinet.wasp.security.ws.conf.EncryptedDataConf;
import org.systinet.wasp.security.ws.conf.EncryptedKeyConf;
import org.systinet.wasp.security.ws.conf.KeyInfoConf;
import org.systinet.wasp.security.ws.conf.EncryptionReferenceConf;
import org.systinet.wasp.security.ws.conf.SignatureConf;
import org.systinet.wasp.security.ws.Constants;
import org.idoox.config.Configurable;
import org.idoox.config.Configurator;

/**
 * WS-Security service.
 */
public class WsSecurityService {

    /**
     * Cenario 1: recebe o username token; nao ha elemento seguro na resposta.
     * @param text text
     * @return text
     */
    public String usernameToken(String text) {

        // criando a configuração da mensagem
        Configurable wsConf = Configurator.newRuntimeConfigurable();
        MessageConf msgConf = (MessageConf) wsConf.narrow(MessageConf.class);
        msgConf.setNoSecurityHeader(Boolean.TRUE);

        // armazenando as configurações da mensagem no contexto de chamada
        Current.getCallContext().getContextData().put(Constants.CD_MESSAGE_CONF,
                                        msgConf);

        return text;
    }

    /**
     * Cenario 2: recebe o username token cifrado; nao ha elemento seguro na resposta.
     * @param text text
     * @return text
     */
    public String encryptedUsernameToken(String text) {

        // criando a configuração da mensagem
        Configurable wsConf = Configurator.newRuntimeConfigurable();
        MessageConf msgConf = (MessageConf) wsConf.narrow(MessageConf.class);
```

```
    msgConf.setNoSecurityHeader(Boolean.TRUE);

    // armazenando as configurações da mensagem no contexto de chamada
    Current.getCallContext().getContextData().put(Constants.CD_MESSAGE_CONF,
                                        msgConf);

    return text;
}

/**
 * Cenario 3: recebe o corpo assinado e cifrado; o corpo da mensagem de reposta e assinado e cifrado.
 * @param text text
 * @return text
 */
public String signedEncryptedBody(String text) {

    // criando a configuração da mensagem
    Configurable wsConf = Configurator.newRuntimeConfigurable();
    MessageConf msgConf = (MessageConf) wsConf.narrow(MessageConf.class);

    // criando um token binario para assinar a reposta (service identity private key)
    SecurityTokenConf signingSecurityTokenConf = msgConf.newSecurityToken();
    signingSecurityTokenConf.setType(Constants.ST_VALUE_TYPE_X509V3);
    signingSecurityTokenConf.setWsuId("SigningSecurityToken-1");
    signingSecurityTokenConf.setOrder(new Integer(10));

    // criando a assinatura...
    SignatureConf signatureConf = msgConf.newSignature();
    signatureConf.setSignBody(Boolean.TRUE);
    KeyInfoConf keyInfoConf = signatureConf.newKeyInfo();
    keyInfoConf.setSecurityTokenMode(Constants.STM_REFERENCE);
    keyInfoConf.setSecurityTokenId("SigningSecurityToken-1");
    signatureConf.setKeyInfo(keyInfoConf);
    signatureConf.setOrder(new Integer(0));
    signatureConf.setWsuId("Signature-1");
    signatureConf.setSignatureMethod(Constants.ALGO_ID_SIGNATURE_RSA);

    // criando os dados cifrados
    EncryptedDataConf encryptedDataConf = msgConf.newEncryptedData();
    encryptedDataConf.setEncryptionMethodAlgorithm(Constants.ALGO_ID_BLOCKCIPHER_TRIPLEDES);
    encryptedDataConf.setEncryptBody(new Boolean(true));
    encryptedDataConf.setEncryptElementContent(new Boolean(true));
    encryptedDataConf.setWsuId("EncryptedData-1");
    encryptedDataConf.setOrder(new Integer(20));

    // criando a chave cifrada
    EncryptedKeyConf encryptedKeyConf = msgConf.newEncryptedKey();
    encryptedKeyConf.setEncryptionMethodAlgorithm(Constants.ALGO_ID_KEYTRANSPORT_RSA15);
    keyInfoConf = encryptedKeyConf.newKeyInfo();
    keyInfoConf.setSecurityTokenMode(Constants.STM_KEYIDENTIFIER);
    keyInfoConf.setSecurityTokenId(Scenario3IncomingValidator.EXT_TOKEN_CLIENT);
```

```
        encryptedKeyConf.setKeyInfo(keyInfoConf);
        encryptedKeyConf.setWsuId("EncryptedKey-1");
        encryptedKeyConf.setOrder(new Integer(25));

        // criando a referência para a cifragem
        EncryptionReferenceConf encryptionReferenceConf = encryptedKeyConf.newReference();
        encryptionReferenceConf.setRefUri("#" + encryptedDataConf.getWsuId());
        encryptedKeyConf.setReferences(new EncryptionReferenceConf[]{encryptionReferenceConf});

        // setando as configurações da mensagem
        msgConf.setSignatures(new SignatureConf[]{signatureConf});
        msgConf.setEncryptedData(new EncryptedDataConf[]{encryptedDataConf});
        msgConf.setEncryptedKeys(new EncryptedKeyConf[]{encryptedKeyConf});
        msgConf.setSecurityTokens(new SecurityTokenConf[]{signingSecurityTokenConf});

        // armazenando as configurações da mensagem no contexto de chamada
        Current.getCallContext().getContextData().put(Constants.CD_MESSAGE_CONF,
                                        msgConf);

        return text;
    }

}
```

Apêncide V: Arquivo WsSecurityServer.java

```
package teste.security.wssecurity.runtime.lowlevel.server;

import org.systinet.wasp.Wasp;
import org.systinet.wasp.security.ws.conf.WSSEConf;
import org.systinet.wasp.security.ws.conf.ExternalSecurityTokensConf;
import org.systinet.wasp.security.ws.conf.SecurityTokenConf;
import org.systinet.wasp.security.ws.Constants;
import org.systinet.wasp.webservice.ServiceEndpoint;
import org.systinet.wasp.webservice.Registry;
import org.systinet.wasp.webservice.ServiceEndpointContext;
import org.idoox.config.Configurable;
import org.idoox.config.Configurator;
import org.idoox.security.Credentials;
import org.idoox.wasp.WaspSecurity;
import org.idoox.util.RuntimeWrappedException;

import java.security.NoSuchProviderException;

/**
 * Servidor WASP com um serviço publicado em 3 pontos com configurações diferentes.
 */
public class WsSecurityServer {
    private static String servicePath = "/teste/security/ws-security/WsSecurityService";
    private static final String alias = "WsSecurityTesteService";
    private static final String password = "changeit";

    private static void setupEndpointContextSecurity(ServiceEndpoint serviceEndpoint) {

        // pegando o contexto
        ServiceEndpointContext ctx = serviceEndpoint.getContext();

        // criando a configuração de segurança
        Configurable configurable = Configurator.newRuntimeConfigurable();
        WSSEConf securityConf = (WSSEConf) configurable.narrow(WSSEConf.class);

        // aceitando username tokens
        securityConf.setNoUsernameTokenValidation(Boolean.TRUE);

        // criando o token para a decifrar os dados
        ExternalSecurityTokensConf extSecToken = securityConf.newExternalSecurityTokens();
        SecurityTokenConf etokenConf = extSecToken.newSecurityToken();
        etokenConf.setType(Constants.ST_VALUE_TYPE_X509V3);
        extSecToken.setSecurityTokens(new SecurityTokenConf[]{etokenConf});
        securityConf.setExternalSecurityTokens(extSecToken);

        // adicionando a classe para validar o token
        securityConf.setValidatorClassName(Scenario3IncomingValidator.class.getName());
```

```
        // setando as configurações de segurança do endpoint
        ctx.getContextData().put(Constants.CD_SECURITY_CONFIGURATION,
                                 securityConf);

        // setando as configurações do provider e da autenticação do serviço
        try {
            Credentials credentials = WaspSecurity.acquireServerCredentials(alias, password,
                                                        Constants.PROVIDER_NAME);
            WaspSecurity.setAcceptingProviders(ctx, new String[]{Constants.PROVIDER_NAME});
            WaspSecurity.setCredentials(ctx, new Credentials[]{credentials});
        } catch (NoSuchProviderException e) {
            throw new RuntimeWrappedException("Essa excecao nunca deveria acontecer!", e);
        }
    }

    public static void main(String[] args) throws Exception {
        String serverURL = System.getProperty("systinet.teste.server.url", "http://localhost:6060");

        // inicializando o servidor WASP
        Wasp.startServer(serverURL);

        // criando uma instancia do serviço e publicando o endpoint
        WsSecurityService svc = new WsSecurityService();
        ServiceEndpoint endpoint = ServiceEndpoint.create(servicePath, svc);
        setupEndpointContextSecurity(endpoint);
        Registry.publish(endpoint);

        // serviço publicado!
        System.out.println("Servico publicado em: " + endpoint.getPath());
    }
}
```

Apêncide VI: Arquivo Scenario3IncomingValidator.java

```
package teste.security.wssecurity.runtime.lowlevel.server;

import org.idoox.config.Configurable;
import org.idoox.config.Configurator;
import org.systinet.wasp.security.ws.IncomingValidator;
import org.systinet.wasp.security.ws.WSSecurityException;
import org.systinet.wasp.security.ws.Constants;
import org.systinet.wasp.security.ws.conf.ExternalSecurityTokensConf;
import org.systinet.wasp.security.ws.conf.MessageConf;
import org.systinet.wasp.security.ws.conf.SecurityTokenConf;
import org.systinet.wasp.security.ws.conf.WSSEConf;
import org.systinet.wasp.security.ws.conf.SignatureConf;
import org.systinet.wasp.webservice.CallContext;
import org.systinet.wasp.webservice.Current;

/**
 * Classe para validar os tokens do cenário 3.
 */
public class Scenario3IncomingValidator implements IncomingValidator {
    /** token de referencia para o servico */
    public static final String EXT_TOKEN_CLIENT = "EncryptingBinaryToken-1";

    public void validate(MessageConf wsSecIncomingMessageConf) throws WSSecurityException {
        // configuração do token x509
        SecurityTokenConf peerTokenConf = null;

        // temos que achar o token que foi usado na assinatura...
        SignatureConf[] signatures = wsSecIncomingMessageConf.getSignatures();
        if (signatures == null) {
            return;
        }
        String signingTokenId = signatures[0].getKeyInfo().getSecurityTokenId();
        SecurityTokenConf[] tokensConf = wsSecIncomingMessageConf.getSecurityTokens();
        for (int i = 0; i < tokensConf.length; i++) {
            if (tokensConf[i].getWsuId().equals(signingTokenId)) {
                peerTokenConf = tokensConf[i];
                break;
            }
        }

        // adicionando um token externo para encriptar a resposta
        Configurable configurable = Configurator.newRuntimeConfigurable();
        WSSEConf securityConf = (WSSEConf)configurable.narrow(WSSEConf.class);
        ExternalSecurityTokensConf extSecToken = securityConf.newExternalSecurityTokens();
        SecurityTokenConf etokenConf = peerTokenConf;
        etokenConf.setOrder(new Integer(2));
        etokenConf.setType(Constants.ST_VALUE_TYPE_X509V3);
        etokenConf.setWsuId(EXT_TOKEN_CLIENT);
```

```
        extSecToken.setSecurityTokens(new SecurityTokenConf[]{etokenConf});
        securityConf.setExternalSecurityTokens(extSecToken);
        CallContext ctx = Current.getCallContext();
        ctx.getContextData().put(Constants.CD_SECURITY_CONFIGURATION, securityConf);
    }
}
```

Apêncide VII: Arquivo WsSecurityClient.java

```
package teste.security.wssecurity.runtime.lowlevel.client;

import org.idoox.config.Configurable;
import org.idoox.config.Configurator;
import org.idoox.wasp.WaspSecurity;
import org.idoox.security.Credentials;
import org.systinet.wasp.security.ws.Constants;
import org.systinet.wasp.security.ws.conf.EncryptedDataConf;
import org.systinet.wasp.security.ws.conf.EncryptedKeyConf;
import org.systinet.wasp.security.ws.conf.EncryptionReferenceConf;
import org.systinet.wasp.security.ws.conf.ExternalSecurityTokensConf;
import org.systinet.wasp.security.ws.conf.KeyInfoConf;
import org.systinet.wasp.security.ws.conf.MessageConf;
import org.systinet.wasp.security.ws.conf.PropertyConf;
import org.systinet.wasp.security.ws.conf.SecurityTokenConf;
import org.systinet.wasp.security.ws.conf.SignatureConf;
import org.systinet.wasp.security.ws.conf.WSSEConf;
import org.systinet.wasp.webservice.ServiceClient;

/**
 * Cliente WS-Security.
 */
public class WsSecurityClient {

    /** caminho do endpoint */
    private static String serviceWSDLPath = "/teste/security/ws-security/WsSecurityService/wsdl";

    /** URL */
    private static String serverURL = null;

    /** serviço do cliente */
    private static ServiceClient serviceClient = null;

    /** proxy */
    private static WsSecurityService serviceSoap = null;

    private static final String ENCRYPTING_TOKEN_ID = "EncryptingToken-1";

    /**
     * Cenário 1: o cliente envia o username token com uma senha em plaintext.
     * @throws Exception if anything goes wrong
     */
    public static void usernameToken() throws Exception {
        // criando as configurações da mensagem
        Configurable wsConfigurable = Configurator.newRuntimeConfigurable();
        MessageConf messageConf = (MessageConf) wsConfigurable.narrow(MessageConf.class);

        // criando o username token
```

```
    SecurityTokenConf tokenConf = messageConf.newSecurityToken();
    tokenConf.setType(Constants.ST_VALUE_TYPE_USERNAME);
    PropertyConf propertyConf = tokenConf.newProperty();
    propertyConf.setPropertyName(Constants.ST_PROPERTY_NAME_PASSWORD_TYPE);
    propertyConf.setPropertyValue(Constants.PT_TEXT_VALUE);
    PropertyConf propertyConfNoNonceCreated = tokenConf.newProperty();
    propertyConfNoNonceCreated.setPropertyName(Constants.ST_PROPERTY_NAME_NO_NONCE_CREATED);
    propertyConfNoNonceCreated.setPropertyValue("true");
    tokenConf.setProperties(new PropertyConf[]{propertyConf, propertyConfNoNonceCreated});
    tokenConf.setWsuId("UsernameToken-1");
    tokenConf.setOrder(new Integer(0));

    // setando as opções de segurança da mensagem
    messageConf.setSecurityTokens(new SecurityTokenConf[]{tokenConf});

    // armazenando as configurações de segurança da mensagem no contexto
    serviceClient.getCallContext().getContextData().put(Constants.CD_MESSAGE_CONF, messageConf);

    // chamada do serviço
    System.out.println("Cenario 1: username token");
    String response = serviceSoap.usernameToken("Resposta");
    System.out.println("  Resposta do servico: " + response + "\n");
}

/**
 * Cenário 2: o cliente envia um username token cifrado.
 * @throws Exception if anything goes wrong
 */
public static void encryptedUsernameToken() throws Exception {

    // criando as configurações da mensagem
    Configurable wsConfigurable = Configurator.newRuntimeConfigurable();
    MessageConf messageConf = (MessageConf) wsConfigurable.narrow(MessageConf.class);

    // criando o username token
    SecurityTokenConf tokenConf = messageConf.newSecurityToken();
    tokenConf.setType(Constants.ST_VALUE_TYPE_USERNAME);
    PropertyConf propertyConf = tokenConf.newProperty();
    propertyConf.setPropertyName(Constants.ST_PROPERTY_NAME_PASSWORD_TYPE);
    propertyConf.setPropertyValue(Constants.PT_TEXT_VALUE);
    tokenConf.setProperties(new PropertyConf[]{propertyConf});
    tokenConf.setWsuId("UsernameToken-1");
    tokenConf.setOrder(new Integer(0));
    messageConf.setSecurityTokens(new SecurityTokenConf[]{tokenConf});

    // criando a chave para a encripação
    EncryptedKeyConf encKeyConf = messageConf.newEncryptedKey();
    encKeyConf.setEncryptionMethodAlgorithm(Constants.ALGO_ID_KEYTRANSPORT_RSA15);
    KeyInfoConf keyInfoConf = encKeyConf.newKeyInfo();
    keyInfoConf.setSecurityTokenMode(Constants.STM_KEYIDENTIFIER);
    keyInfoConf.setSecurityTokenId(ENCRYPTING_TOKEN_ID);
```

```
    encKeyConf.setKeyInfo(keyInfoConf);
    encKeyConf.setOrder(new Integer(20));

    // criando os dados encriptados
    EncryptedDataConf encDataConf = messageConf.newEncryptedData();
    encDataConf.setEncryptionMethodAlgorithm(Constants.ALGO_ID_BLOCKCIPHER_TRIPLEDES);
    encDataConf.setEncryptElementContent(new Boolean(false));
    encDataConf.setEncryptionTargetId("UsernameToken-1");
    encDataConf.setWsuId("EncryptedData-1");
    encDataConf.setOrder(new Integer(10));

    // criando uma referência para os dados encriptados
    EncryptionReferenceConf encryptionReferenceConf = encKeyConf.newReference();
    encryptionReferenceConf.setRefUri("#" + encDataConf.getWsuId());
    encKeyConf.setReferences(new EncryptionReferenceConf[]{encryptionReferenceConf});

    // setando as configurações de segurança da mensagem
    messageConf.setEncryptedData(new EncryptedDataConf[]{encDataConf});
    messageConf.setEncryptedKeys(new EncryptedKeyConf[]{encKeyConf});

    // armazenando as configurações de segurança da mensagem no contexto
    serviceClient.getCallContext().getContextData().put(Constants.CD_MESSAGE_CONF, messageConf);

    // chamada do serviço
    System.out.println("Cenario 2: username token cifrado");
    String response = serviceSoap.encryptedUsernameToken("Resposta");
    System.out.println("  Resposta do servico: " + response + "\n");
}

/**
 * Cenário 3: o cliente assina e encripta o corpo da mensagem.
 * @throws Exception if anything goes wrong
 */
public static void signedEncryptedBody() throws Exception {
    // criando as configurações de sugurança da mensagem
    Configurable wsConfigurable = Configurator.newRuntimeConfigurable();
    MessageConf messageConf = (MessageConf) wsConfigurable.narrow(MessageConf.class);

    // criando um token binário para a assinatura (usa a chave privada do cliente)
    SecurityTokenConf signingSecurityTokenConf = messageConf.newSecurityToken();
    signingSecurityTokenConf.setType(Constants.ST_VALUE_TYPE_X509V3);
    signingSecurityTokenConf.setWsuId("SigningSecurityToken-1");
    signingSecurityTokenConf.setOrder(new Integer(10));

    // criando a assinatura
    SignatureConf signatureConf = messageConf.newSignature();
    signatureConf.setSignBody(Boolean.TRUE);
    KeyInfoConf keyInfoConf = signatureConf.newKeyInfo();
    keyInfoConf.setSecurityTokenMode(Constants.STM_REFERENCE);
    keyInfoConf.setSecurityTokenId("SigningSecurityToken-1");
    signatureConf.setKeyInfo(keyInfoConf);
```

```
    signatureConf.setOrder(new Integer(0));
    signatureConf.setWsuId("Signature-1");
    signatureConf.setSignatureMethod(Constants.ALGO_ID_SIGNATURE_RSA);

    // criando os dados encriptados
    EncryptedDataConf encryptedDataConf = messageConf.newEncryptedData();
    encryptedDataConf.setEncryptionMethodAlgorithm(Constants.ALGO_ID_BLOCKCIPHER_TRIPLEDES);
    encryptedDataConf.setEncryptBody(new Boolean(true));
    encryptedDataConf.setEncryptElementContent(new Boolean(true));
    encryptedDataConf.setWsuId("EncryptedData-1");
    encryptedDataConf.setOrder(new Integer(20));

    // criando a chave para a encriptação
    EncryptedKeyConf encryptedKeyConf = messageConf.newEncryptedKey();
    encryptedKeyConf.setEncryptionMethodAlgorithm(Constants.ALGO_ID_KEYTRANSPORT_RSA15);
    keyInfoConf = encryptedKeyConf.newKeyInfo();
    keyInfoConf.setSecurityTokenMode(Constants.STM_KEYIDENTIFIER);
    keyInfoConf.setSecurityTokenId(ENCRYPTING_TOKEN_ID);
    encryptedKeyConf.setKeyInfo(keyInfoConf);
    encryptedKeyConf.setWsuId("EncryptedKey-1");
    encryptedKeyConf.setOrder(new Integer(25));

    // criando uma referência para os dados encriptados
    EncryptionReferenceConf encryptionReferenceConf = encryptedKeyConf.newReference();
    encryptionReferenceConf.setRefUri("#" + encryptedDataConf.getWsuId());
    encryptedKeyConf.setReferences(new EncryptionReferenceConf[]{encryptionReferenceConf});

    // setando as configurações de segurança da mensagem
    messageConf.setSignatures(new SignatureConf[]{signatureConf});
    messageConf.setEncryptedData(new EncryptedDataConf[]{encryptedDataConf});
    messageConf.setEncryptedKeys(new EncryptedKeyConf[]{encryptedKeyConf});
    messageConf.setSecurityTokens(new SecurityTokenConf[]{signingSecurityTokenConf});
    messageConf.setCreateTimestampHeader(Boolean.TRUE);

    // armazenando as configurações de segurança da mensagem no contexto
    serviceClient.getCallContext().getContextData().put(Constants.CD_MESSAGE_CONF, messageConf);

    // chamada do serviço
    System.out.println("Cenario 3: corpo da mensagem SOAP assinado e cifrado");
    String response = serviceSoap.signedEncryptedBody("Resposta");
    System.out.println("  Resposta do servico: " + response + "\n");
}

private static void setupClientContextSecurity() {
    // criando um token binário externo para a encriptação (chave pública do serviço)
    Configurable configurable = Configurator.newRuntimeConfigurable();
    WSSEConf securityConf = (WSSEConf)configurable.narrow(WSSEConf.class);
    ExternalSecurityTokensConf extSecToken = securityConf.newExternalSecurityTokens();
    SecurityTokenConf etokenConf = extSecToken.newSecurityToken();
    etokenConf.setType(Constants.ST_VALUE_TYPE_X509V3);
    etokenConf.setWsuId(ENCRYPTING_TOKEN_ID);
```

```
        PropertyConf propertyConf = etokenConf.newProperty();
        propertyConf.setPropertyName(Constants.ST_PROPERTY_NAME_ALIAS);
        propertyConf.setPropertyValue("CN=WsSecurityTesteService"); // alias para o servico
        etokenConf.setProperties(new PropertyConf[]{propertyConf});

        // token binário externo usado para decifrar
        SecurityTokenConf etokenConf2 = extSecToken.newSecurityToken();
        etokenConf2.setType(Constants.ST_VALUE_TYPE_X509V3);

        // setando os tokens externos
        extSecToken.setSecurityTokens(new SecurityTokenConf[]{etokenConf,etokenConf2});
        securityConf.setExternalSecurityTokens(extSecToken);

        // setando o contexto de segurança do cliente
        serviceClient.getContext().getContextData().put(Constants.CD_SECURITY_CONFIGURATION,
                                        securityConf);
    }

    /**
     * Main method.
     * @param args not used
     * @throws Exception if anything goes wrong
     */
    public static void main(String args[]) throws Exception {
        serverURL = System.getProperty("systinet.teste.server.url", "http://localhost:6060");

        // criando uma instancia do cliente para o webservice dado
        serviceClient = ServiceClient.create(serverURL + serviceWSDLPath);

        // criando o proxy
        serviceSoap = (WsSecurityService)serviceClient.createProxy(WsSecurityService.class);

        // autenticando o cliente e setando as credenciais do serviço
        Credentials creds = WaspSecurity.acquireClientCredentials("Gabriel", "asdf", "WS-Security");
        WaspSecurity.setCredentials(serviceClient, new Credentials[]{creds});
        WaspSecurity.setInitiatingProvider(serviceClient, "WS-Security");

        // setando o contexto de segurança para o cliente
        setupClientContextSecurity();

        /** Cenário 1: o cliente envia o username token com uma senha em plaintext */
        usernameToken();

        /** Cenário 2: o cliente envia um username token cifrado */
        encryptedUsernameToken();

        /** Cenário 3: o cliente assina e encripta o corpo da mensagem */
        signedEncryptedBody();
    }
}
```

Apêncide VIII: Arquivo WsSecurityService.wsdl

```
<?xml version='1.0' encoding='utf-8' ?>
<definitions name='teste.security.wssecurity.persistent.server.WsSecurityService' targetNamespace='http://hanauer.com/w
   xmlns:map='http://systinet.com/mapping/'
   xmlns:soap='http://schemas.xmlsoap.org/wsdl/soap/'
   xmlns:ns0='http://systinet.com/xsd/SchemaTypes/'
   xmlns:tns='http://hanauer.com/wsdl/teste/security/wssecurity/server/'
   xmlns='http://schemas.xmlsoap.org/wsdl/'>
   <types>
      <xsd:schema elementFormDefault="qualified"
         targetNamespace="http://systinet.com/xsd/SchemaTypes/"
         xmlns:tns="http://systinet.com/xsd/SchemaTypes/"
         xmlns:xsd="http://www.w3.org/2001/XMLSchema">
         <xsd:element name="p0" nillable="true" type="xsd:string"/>
         <xsd:element name="string_Response" nillable="true"
            type="xsd:string"/>
      </xsd:schema>

   </types>
   <message name='WsSecurityService_encryptedUsernameToken_Response_Soap'>
      <part name='response' element='ns0:string_Response'/>
   </message>
   <message name='WsSecurityService_encryptedUsernameToken__Request_Soap'>
      <part name='p0' element='ns0:p0'/>
   </message>
   <message name='WsSecurityService_signedEncryptedBody_Response_Soap'>
      <part name='response' element='ns0:string_Response'/>
   </message>
   <message name='WsSecurityService_signedEncryptedBody__Request_Soap'>
      <part name='p0' element='ns0:p0'/>
   </message>
   <message name='WsSecurityService_usernameToken_Response_Soap'>
      <part name='response' element='ns0:string_Response'/>
   </message>
   <message name='WsSecurityService_usernameToken__Request_Soap'>
      <part name='p0' element='ns0:p0'/>
   </message>
   <portType name='WsSecurityService'>
      <operation name='encryptedUsernameToken' parameterOrder='p0'>
         <input message='tns:WsSecurityService_encryptedUsernameToken__Request_Soap'/>
         <output message='tns:WsSecurityService_encryptedUsernameToken_Response_Soap'/>
      </operation>
      <operation name='signedEncryptedBody' parameterOrder='p0'>
         <input message='tns:WsSecurityService_signedEncryptedBody__Request_Soap'/>
         <output message='tns:WsSecurityService_signedEncryptedBody_Response_Soap'/>
      </operation>
      <operation name='usernameToken' parameterOrder='p0'>
         <input message='tns:WsSecurityService_usernameToken__Request_Soap'/>
         <output message='tns:WsSecurityService_usernameToken_Response_Soap'/>
```

```
      </operation>
    </portType>
    <binding name='WsSecurityService' type='tns:WsSecurityService'>
      <soap:binding transport='http://schemas.xmlsoap.org/soap/http' style='document'/>
      <operation name='encryptedUsernameToken'>
        <map:java-operation name='encryptedUsernameToken' signature='KExqYXZhL2xhbmcvU3RyaW5nOylMamF2YS
        <soap:operation soapAction='http://hanauer.com/wsdl/teste/security/wssecurity/server/WsSecurityService#encrypte
        <input>
          <soap:body parts='p0' use='literal'/>
        </input>
        <output>
          <soap:body parts='response' use='literal'/>
        </output>
      </operation>
      <operation name='signedEncryptedBody'>
        <map:java-operation name='signedEncryptedBody' signature='KExqYXZhL2xhbmcvU3RyaW5nOylMamF2YS9sYW
        <soap:operation soapAction='http://hanauer.com/wsdl/teste/security/wssecurity/server/WsSecurityService#signedE
        <input>
          <soap:body parts='p0' use='literal'/>
        </input>
        <output>
          <soap:body parts='response' use='literal'/>
        </output>
      </operation>
      <operation name='usernameToken'>
        <map:java-operation name='usernameToken' signature='KExqYXZhL2xhbmcvU3RyaW5nOylMamF2YS9sYW5nL1
        <soap:operation soapAction='http://hanauer.com/wsdl/teste/security/wssecurity/server/WsSecurityService#usernam
        <input>
          <soap:body parts='p0' use='literal'/>
        </input>
        <output>
          <soap:body parts='response' use='literal'/>
        </output>
      </operation>
    </binding>
    <service name='WsSecurityService'>
      <port name='WsSecurityService' binding='tns:WsSecurityService'>
        <soap:address location='urn:unknown-location-uri'/>
      </port>
    </service>
  </definitions>
```

Apêncide IX: Artigo

WS SECURITY - ESTUDO E VERIFICAÇÃO DA QUALIDADE DE PROTEÇÃO EM WEB SERVICES

Gabriel Hanauer
Universidade Federal de Santa Catarina
Departamento de Informática e Estatística

hanauer@inf.ufsc.br

José Eduardo De Lucca
Universidade Federal de Santa Catarina
Departamento de Informática e Estatística

delucca@inf.ufsc.br

RESUMO

Os web services tem evoluído muito lentamente, pois não oferecem segurança necessária. Dessa maneira, como podemos saber que um certo web service está relamente fazendo o que ele se propõe a fazer ou que ele é confiável? Muitas empresas tem desenvolvido suas próprias soluções para resolver esses problemas de segurança.

Porém, uma especificação chamada WS Security foi proposta. O principal objetivo dessa especificação é criar um padrão que mais tarde poderá ser usado pelos desenvolvedores e pela comunidade em geral.

O WS Security se propõe a resolver alguns dos problemas de segurança existentes, como integridade das mensagens, autenticação usando assinatura XML, cifrar usando XML, etc.

Nossa análise do protótipo mostra que somente o WS Security não oferece toda a segurança necessária. E também não oferece segurança total nas áreas propostas. O WS Security precisa ser complementado com outras especificações que estão surgindo e também com outros sistemas para garantir uma segurança adequada.

ABSTRACT

It is commonly stated that Web Services does not provide security and therefore its journey has been very slow. For instance, how will you know that a certain Web service really does what it claims to do or that it comes from a trusted party? Companies has been developing their own security solutions to overbuild this lack of security.

Therefore a specification titled WS Security has been proposed. The main goal has been to make a standard that the market will use.

WS Security claims to solve some of the existing security issues, such as message integrity, authentication by using XML signature, XML encryption and more.

Our analysis of the prototype shows that WS Security alone does not provide the needed security. Neither does it provide full security in the areas it targets. WS Security has to be complemented with other upcoming specifications and other systems in order to give adequate security.

Palavras-chave

Web Services, segurança, ws-security.

INTRODUÇÃO

A internet cresceu rapidamente nos últimos anos, criando meios alternativos de comunicação e aumentando a disponibilidade de informação e serviços. Mas por trás desta disponibilidade existe uma demanda. Uma demanda por mais e melhores serviços.

O resultado dessa demanda criou um fenômeno chamado Web Services. Muitas empresas estão começando a usar e desenvolver Web Services.

Numa definição mais concreta de Web Services podemos dizer que

> *"Um Web Service é uma aplicação identificada por uma URI, onde suas interfaces são capazes de serem definidas, descritas e descobertas por artefatos XML, e suporta interações diretas com outras aplicações que usam mensagens baseadas em XML através de protocolos Internet."* W3C Web Services Architecture Group

A partir dessa definição podemos nos fazer uma pergunta: porque esse fenômeno ainda não está sendo largamente usado? Esta pergunta pode ser respondida de várias maneiras. Uma delas é que não há consideração nenhuma à segurança nesses Web Services. E as soluções dadas até o momento não estão maduras o suficiente para serem aceitas no mercado.

Entretanto, algumas empresas como Microsoft, IBM e Sun criaram um comitê chamado OASIS[19]. OASIS criou uma especificação chamada WS Security com o objetivo de resolver alguns dos problemas de segurança dos Web Services. O objetivo é fazer da especificação WS Security um padrão, e também criar referências para aumentar a integridade, confidencialidade e autenticação das mensagens, obtendo assim melhor segurança. É importante lembrar que o WS Security não trata de todos os problemas de segurança. A especificação é apenas um guia. Entretanto, é interessante analisar esta especificação e sua qualidade de segurança de uma maneira prática e teórica. Este trabalho tem o objetivo de fazer esta análise através de um caso de estudo de Web Services baseado no WS Security.

ESTUDO INICIAL

Para entender a tecnologia que está por trás do WS Security é necessário que sejam explicadas algumas definições importantes. Esta parte do trabalho tem o objetivo de explicar o conceito de segurança e mostrar algumas técnicas de segurança usadas atualmente.

Segurança

A segurança pode estar relacionada a software, hardware e também a um processo humano. A segurança está dividida em quatro áreas principais que englobam todos os seus aspectos.

[19] http://www.oasis-open.org

A Segurança em Web Services com SSL/IPsec

Como os Web Services não tem uma solução adequada para segurança, eles dependem da segurança de outros sistemas e camadas. Por isso, SSL e IPsec se tornaram populares para garantir a segurança de Web Services.

O SSL usa o protocolo "Handshake" e certificados para fazer a autenticação do cliente e servidor[20]. O SSL só pode autenticar e cifrar a comunicação entre dois pontos. Mas as requisições e respostas de Web Services normalmente usam mais de uma rota para se comunicar. O SSL é uma solução adequada se conhecermos previamente o caminho que a mensagem irá percorrer. Outro ponto importante é que o SSL protege os dados somente durante a transmissão. No momento que os dados chegam ao seu destino é necessário fazer uso de outras técnicas para manter a segurança.

Outro problema das técnicas de SSL para Web Services é que a autenticação e cifragem do SSL consomem muito tempo de CPU e conseqüentemente o processo da transação se torna lento.

Uma das conseqüências das falhas de segurança para web services mencionadas acima, é que as mensagens são fáceis de serem copiadas.

IPsec é um formato seguro do IP. Ele consiste em duas partes: AH para a autenticação e ESP para a cifragem. O IPsec se propõe a prover autenticação, integridade e cifragem.[21]

Segurança XML (cifragem e assinatura XML)

Através de assinaturas XML, pode-se assinar somente algumas partes de um documento XML. Com a cifragem XML pode-se cifrar partes de um documento XML simétrica e assimetricamente, isto é, pode-se cifrar um elemento específico.[22]

Assinatura e cifragem XML cobrem três falhas de segurança de um web service: autenticação, integridade e a garantia de que uma mensagem falsa não possa ser enviada.

Assinaturas XML

Assinaturas XML podem ser feitas de 3 maneiras:

- a informação assinada fica dentro da assinatura;
- a assinatura fica dentro da informação que está sendo assinada ;
- a informação assinada e a assinatura ficam em arquivos XML diferentes.

A assinatura XML contém informações sobre as técnicas de assinatura que estão sendo usadas, como por exemplo, hashing e algoritmos de cifragem.[23]

Através dessa técnica é possível garantir que as informações não serão mudadas.

Cifragem XML

A cifragem XML garante que a informação não será lida por usuários desautorizados. Diferentemente do SSL, essa técnica garante que as informações estarão protegidas durante o transporte e também no computador do usuário.

Na cifragem XML podemos usar chaves simétricas ou assimétricas. As mais usadas são as simétricas, pois elas consomem menos CPU e são mais adequadas para a troca de grandes quantidades de informação.

Problemas de Segurança no SOAP e SOAP-Sec

"SOAP, the Simple Object Access Protocol, is XML syntax for exchanging messages. Because it is XML, it is both language and platform independent"[24]

O SOAP usa a mesma porta que o HTTP, e é um protocolo comum usado para a troca de informação entre redes. O SOAP não foi desenvolvido considerando se as falhas de segurança de web services.

Como as mensagens SOAP consistem de código XML, qualquer pessoa que pegar uma mensagem poderá ver as informações que estão sendo trocadas. E algumas dessas informações podem ser importantes como, por exemplo, senhas. Desse modo, é muito importante que a informação seja cifrada.

Outro problema com as mensagens SOAP é que elas são transmitidas num único sentido. A conseqüência disso é que elas são fáceis de serem roubadas e reenviadas.

Para corrigir alguns dos problemas de segurança do SOAP, uma nova técnica foi proposta pela IBM e Microsoft, chamada SOAP-Sec. SOAP-Sec permite que mensagens sejam assinadas adicionando para isso um cabeçalho ao SOAP.

WS Security

A especificação WS Security começou a ser desenvolvida por três grandes empresas: IBM, Microsoft e Verisign. O WS Security foi criado para corrigir as falhas de segurança que existem nos Web Services e também para ser um padrão na troca e assinatura de mensagens seguras.

O WS Security não resolve todos os problemas de segurança e também não prove um modelo específico de como um Web Service deve ser desenvolvido. Ele oferece somente um caminho a ser seguido.É importante perceber que o WS Security é apenas uma especificação. Uma especificação focada nas extensões SOAP.

Essas extensões provêem autenticação, integridade, confidencialidade e assinatura para as mensagens. O WS Security está focado na autenticação e na cifragem.

O WS Security não está limitado a um modelo ou mecanismo específico. Ele suporte vários modelos e mecanismos de segurança. Por exemplo, um desenvolverdor pode usar tokens de software como também tokens de hardware.

Entretanto, o WS Security possui alguns requisitos de sistema, que são:

- a linguagem do web service deve ter suporte a múltiplos tokens de segurança;
- várias tecnologias de criptografia;
- segurança entre o transmissor e o receptor;
- segurança no transporte.

O WS Security preenche esses requisitos fazendo uso da assinatura e cifragem XML, e também de outras técnicas. Pode-

[20] http://www.rsasecurity.com/rsalabs/faq/5-1-2.html
[21] http://www.rsasecurity.com/rsalabs/faq/5-1-4.html
[22] http://www.vordel.com/knowledgebase/tutorial_xml_security/
[23] XML and Security, XML journal, December 8, 2001, Mark O'Neill
[24] http://www.vbxml.com/soapworkshop/articles/intro/default.asp

se complementar a segurança com SSL e outras técnicas parecidas.[25]
O WS Security é considerado um substituto do SOAP-Sec.[26]

IMPLEMENTAÇÃO

O passo inicial foi a análise e modelagem dos dados e informações do protótipo.
A seguir entraremos na parte que nos interessa: a implementação de web services para verificar a autenticação e cifragem usando o WS Security.
Vamos dividir nosso objeto de estudo em 3 cenários diferentes:

4. O cliente envia uma mensagem com um token (esse token é o nome do usuário e a senha no formato de texto puro). O servidor verifica a existência do usuário e manda uma mensagem de volta ao cliente sem nenhum elemento de segurança.
5. O cliente envia uma mensagem com um token cifrado (o token é cifrado usando o serviço de identificação por chave pública). O servidor decifra o token, verifica a existência do usuário e manda uma mensagem ao cliente sem nenhum elemento de segurança.
6. O cliente assina (usando uma chave privada para a identificação do cliente) e cifra o corpo da mensagem (usando o serviço de identificação por chave pública). O cliente ainda adiciona um timestamp no cabeçalho. O servidor decifra o corpo da mensagem, verifica a assinatura e manda uma mensagem de volta ao cliente – o corpo da mensagem que é enviada de volta ao cliente é assinada com o serviço de identificação por chave privada e então é cifrada usando-se a chave pública do cliente que enviou a mensagem ao servidor.

Para que a assinatura e a cifragem sejam possíveis o cliente e o serviço devem possuir o par de chaves pública e privada e devem possuir o certificado digital um do outro nas suas bases de chaves.

CONCLUSÃO

O WS Security oferece uma estrutura extensa para melhorar a segurança dos Web Services. Em contrapartida, ele não oferece toda a segurança necessária que um sistema deveria ter. Um exemplo disso é que o WS Security não possui métodos para estabelecer uma relação confiável entre servidor e cliente. No nosso protótipo essa relação foi criada usando-se características do servidor WASP para a criação de certificados digitais e pares de chave pública e privada, e não através do WS Security. Normalmente a especificação resolve problemas que acontecem após estabelecida uma relação confiável entre as partes envolvidas.
Uma vantagem do WS Security é o suporte a vários tipos de protocolos e tokens. Entretanto, ele não oferece informações de como deixar esses protocolos seguros.
O WS Security é um guia de como assegurar que mensagens SOAP sejam seguras, mas também ele possui furos de segurança em outras áreas como disponibilidade.
Basicamente, o WS Security suporta autenticação, confidencialidade e integridade de uma maneira limitada. Por exemplo, ele não possui suporte para se revogar certificados de autenticação.

O timestamp, no WS Security, não possui métodos para a sincronização do tempo, o que aumenta o risco de ocorrer ataques do tipo replay. Essa falha é uma desvantagem no que diz respeito ao suporte a Kerberos, pois ele requer sincronização de tempo.
Outro ponto em que o WS Security falha é no que diz respeito a listas de acesso, pois ele não pode checar quais tipos de acesso um usuário possui ou se o certificado de um usuário foi revogado ou expirado.
Usar a cifragem XML e deixar partes do documento decifradas pode fazer com que usuários maliciosos tenham acesso a informações sigilosas.
Outro problema do WS Security é a falta de suporte a controle de versão e de que maneira vírus podem ser prevenidos.
Como mencionado anteriormente, o WS Security deve ser complementado com outras especificações. Por exemplo, para dar suporte ao estabelecimento de relações de confiança ele deve ser complementado com o WS-Trust e o WS-SecureConversation para dar suporte a mecanismos de handshake. Mas o problema é que o WS-Trust e o WS-SecureConversation ainda estão em desenvolvimento.

REFERÊNCIAS BIBLIOGRÁFICAS

[1] JENNINGS, Roger. Dig Into WS-Security With the WSDK. [S.l]. Fawcette Technical Publications, 20 set. 2002. Disponível em: <http://www.fawcette.com/xmlmag/2002_09/online/webservices_rjennings_09_20_02/default_pf.asp>.

[2] GALLI, Peter. Spec Secures Web Services Apps. New Orleans. eWeek, 11 abr. 2002. Disponível em: <http://www.eweek.com/article2/0,3959,50620,00.asp>.

[3] APSHANKAR, Kapil. WS-Security: Security for Web Services.[S.l]. Web Services Architect. 24 jul. 2002. Disponível em: <http://www.webservicesarchitect.com/content/articles/apshankar04.asp>.

[4] OASIS. Web Services Architecture Working Group. 2001. Disponível em: < http://www.w3.org/2002/ws/arch/>.

[5] A. Shunn. Managed Security: Build It Right the First Time. 2002. Disponível em: <http://archive.devx.com/security/articles/WebServices/partI/AS0102-1.asp>.

[6] RSA Security. What is SSL?. 2002. Disponível em: <http://www.rsasecurity.com/rsalabs/faq/5-1-2.html>.

[7] RSA Security. What is IPSec?. 2002. Disponível em: < http://www.rsasecurity.com/rsalabs/node.asp?id=2295>.

[8] Vordel. Web services security. 2002. Disponível em: <http://www.vordel.com>.

[9] D.K. Taft. WS-Security Spec Sent to OASIS. 27 jun. 2002. Disponível em: <http://www.eweek.com/article2/0,3959,290627,00.asp>.

[10] D. Eastlake, J. Reagle. XML-Signature Syntax and Processing. 2002. Disponível em: < http://rfc3075.x42.com/ >.

[11] Dillaway, Fox, Imamura, LaMacchia, Maruyama, Schaad, and Simon. XML Encryption Syntax and Processing. 2001. Disponível em: < http://www.w3.org/Encryption/2001/05/11-proposal.html >.

[25] http://www.webservicesarchitect.com/content/articles/apshankar04.asp

[26] http://www.vordel.com/news/articles/02-03-08.html

[12] J.Boyer, PureEdge Solutions Inc. Canonical XML Version 1.0. 2001. Disponível em: < http://www.w3.org/TR/xml-c14n >.

[13] E.X. DeJesus. Secure Your Web Services Applications. Jun. 2002. Disponível em: <http://www.fawcette.com/dotnetmag/2002_06/magazine/features/edejesus/>.

[14] A. Yang. XML Web Services Security Issues. 10 abr. 2002. Disponível em: <http://www.xwss.org/articlesThread.jsp?forum=34&thread=648>.

[15] R.Shirey. Internet Security Glossary. Mai. 2000. Disponível em: <http://www.faqs.org/rfcs/rfc2828.html>.

[16] A.Jøsang, D.Povey, A.Ho. What You See Is Not Always What You Sign. 2002. Disponível em: <http://citeseer.nj.nec.com/535070.html>.

[17] Zaw-Sing Su. A SPECIFICATION OF THE INTERNET PROTOCOL (IP) TIMESTAMP OPTION. 1981. Disponível em: <http://www.faqs.org/rfcs/rfc781.html>.

[18] Babylon, 2002, Disponível em: <http://www.babylon.com>.

[19] Westbridge. XML Web Services Security Glossary. 2002. Disponível em: <http://www.westbridgetech.com/downloads/XWSS.orgWebServicesSecurityGlossary