

Faculdade de Engenharia da Universidade do Porto



**Arquitectura de Segurança num ambiente SOA
(*Service Oriented Architecture*)**

Sónia Cristina Santos de Pinho

Relatório de Dissertação realizada no Âmbito do
Mestrado Integrado em Engenharia Informática e Computação

Orientador: Miguel Pimenta Monteiro (Prof. Doutor)

Julho de 2008

Arquitectura de Segurança num ambiente SOA (*Service Oriented Architecture*)

Sónia Cristina Santos de Pinho

Relatório de Dissertação realizada no Âmbito do
Mestrado Integrado em Engenharia Informática e Computação

Aprovado em provas públicas pelo Júri:

Presidente: Ana Cristina Ramada Paiva Pimenta (Prof^a. Doutora)

Arguente: Manuel Bernardo Barbosa (Prof. Doutor)

Vogal: Miguel Pimenta Monteiro (Prof. Doutor)

31 de Julho de 2008

Resumo

Actualmente, muitas empresas estão a adoptar arquitecturas SOA (*Service Oriented Architecture*) para exporem/consumirem serviços, implementados na forma de WS (*Web Services*). Um dos objectivos do uso das arquitecturas SOA é permitir a integração de aplicações implementadas em ambientes heterogéneos e distribuídos que, muitas vezes, estão localizados para lá das fronteiras da própria organização.

Com a introdução deste novo estilo de arquitectura, surgiram também novas preocupações de segurança a ter em conta na implementação dos sistemas envolvidos. Como SOA permite a partilha de informação e processos entre organizações, foram introduzidas novas formas de interacção entre os componentes da arquitectura, surgindo também novos tipos de ataques aos sistemas. Estes ataques não são detectados/prevenidos pelas tecnologias de segurança tradicionais. Por consequência, uma empresa que necessite de implementar uma arquitectura SOA com exposição dos seus WS para o exterior, deverá também implementar as medidas de segurança mais adequadas para não colocar em perigo a restante infra-estrutura interna.

Este trabalho tem como objectivos a identificação dos novos riscos de segurança aos quais as arquitecturas SOA ficam expostas, identificação dos requisitos de segurança deste tipo de arquitecturas e das tecnologias já disponíveis para implementar algumas medidas de segurança específicas das arquitecturas SOA. No final deste trabalho, são sugeridas algumas medidas de segurança a implementar num caso prático, na arquitectura SOA do Banco BPI. Como já existem equipamentos de infra-estrutura disponíveis para introduzir alguma segurança nas arquitecturas SOA, no caso prático apresentado, está também descrito o processo de avaliação das XML *Security Gateways* existentes actualmente no mercado.

Abstract

Nowadays, many enterprises are adopting SOA (Service Oriented Architecture) to expose/consume services, implemented using WS (Web Services). One of the main goals when using SOA is to allow application integration through heterogeneous and distributed environments that, many times, are located beyond the companies' boundaries.

With the introduction of this new style of architecture, new security concerns emerged, that are to be considered during system's implementation. As SOA allows the share of information and processes through different organizations, new types of interaction among the architecture's components were introduced, thereby emerging new types of system's attacks. These attacks aren't detected/prevented by the traditional security technologies. Consequently, a company that needs to implement SOA with WS accessed from the outside will have to implement new security measures and expose its services without endangering the security of the internal infrastructure.

The main goal of this work is the identification of the new security risks at which SOA will be exposed, identification of this type of architecture's security requirements and the technologies available to implement some security measures specific to the SOA. At the end of this work, some security measures are suggested that can be applied in a real SOA environment, at Banco BPI. As there are many types of equipment available that we can introduce at the infrastructure to add some security to the architecture, for this real case, the XML Security Gateways' process of evaluation is described.

Agradecimentos

- Ao Banco BPI, S.A. e, em particular, ao Departamento de Sistemas de Informação, pelas facilidades concedidas e pelo apoio dado.
- Ao Prof. Miguel Pimenta Monteiro, da FEUP, pela sua orientação e acompanhamento.
- Ao Prof. Raul Vidal da FEUP, ao Eng. Pedro Silvestre do Banco BPI e à minha família, pelo incentivo na realização deste trabalho.

Conteúdo

1	Introdução	1
2	Web Services (WS)	4
2.1	Tipos de <i>Web Services</i> : REST versus SOAP <i>Web Services</i>	4
2.2	SOAP <i>Web Services</i> (ou simplesmente, <i>Web Services</i>)	5
2.3	Tecnologias usadas na implementação dos <i>Web Services</i>	6
2.3.1	XML (eXtensible Markup Language)	7
2.3.2	SOAP (Simple Object Access Protocol)	9
2.3.3	WSDL (Web Services Description Language)	10
2.3.4	UDDI (Universal Description, Discovery and Integration)	11
2.4	Funcionamento de um modelo baseado em <i>Web Services</i>	13
3	Problemas de Segurança da Arquitectura Orientada a Serviços (SOA)	16
3.1	O HTTP e HTTPS estão permitidos na maior parte das <i>firewalls</i>	16
3.2	O uso de HTTPS/VPNs não é suficiente para garantir a segurança das mensagens	17
3.3	Descoberta de informação sobre os sistemas	17
3.4	Falhas nos processos de autenticação	18
3.5	Falhas nos processos de autorização	19
3.6	Intercepção das mensagens	20
3.7	Falhas no registo dos acessos aos WS	20
3.8	Alteração das mensagens (integridade) e falhas na garantia da confidencialidade	20
3.9	Ataques ao cliente	22
3.10	Ataques usando referências externas (<i>External Reference Attack</i>)	23
3.11	Transmissão de mensagens com conteúdo ou anexos maliciosos	24
3.12	XML <i>Denial of Service</i> (XDoS), <i>Denial of Service</i> (DoS), <i>Buffer Overflows</i> , ...	24
3.13	Sistemas/Serviços distribuídos	27
4	Requisitos de Segurança de uma Arquitectura Orientada a Serviços (SOA)	28
4.1	Gestão de Identidades/Autenticação	28
4.2	Autorização	28
4.3	Integridade e confidencialidade dos dados	29
4.4	Integridade das transacções e comunicações	29
4.5	Disponibilidade dos serviços	29
4.6	Não repúdio	30
4.7	Integridade dos dados relativos a UDDI	30
4.8	Registo das actividades (Auditoria)	30
4.9	Cumprimento das políticas de segurança aplicadas aos sistemas distribuídos	30
5	<i>Standards WS</i> disponíveis para implementar segurança numa Arquitectura SOA	32
6	Algumas tecnologias disponíveis para a implementação de medidas de segurança numa Arquitectura SOA	34
6.1	XML <i>Security Gateway</i> (XSG)	34
6.2	Implementação de segurança a nível da configuração dos próprios WS	42
6.3	Ferramentas de gestão de identidades e acessos	46
6.4	SOA <i>Governance</i>	47

7	Boas práticas de segurança a seguir na implementação de uma Arquitectura Orientada a Serviços.....	49
8	CrITÉrios de avaliação das XML Security Gateways	57
8.1	Arquitectura de implementação.....	57
8.2	Identificação e Autenticação	59
8.3	Autorização	61
8.4	Registo das actividades	61
8.5	Validação do conteúdo das mensagens/Detecção de ataques.....	62
8.6	Filtragem com base em IP/porta	63
8.7	Método usado para bloqueio do tráfego.....	63
8.8	Standards de segurança suportados	64
8.9	Formatos de documentos suportados.....	64
8.10	Controlo de acessos ao WSDL	64
8.11	Armazenamento das chaves usadas nas tarefas criptográficas	65
8.12	Gestão da própria XSG.....	65
8.13	Gestão/Monitorização dos WS com base em SLAs.....	65
8.14	Integração da XSG com outros equipamentos/ferramentas.....	65
8.15	Suporte pós-venda.....	66
8.16	Investimento necessário.....	66
9	Processo de avaliação da XML Security Gateway a ser adoptada pelo Banco BPI	67
9.1	Levantamento das soluções existentes no mercado.....	67
9.2	Elaboração do RFI (<i>Request for Information</i>) e análise das respostas dos fornecedores	68
9.3	Elaboração do RFP (<i>Request for Proposal</i>) e análise das respostas dos fornecedores.....	69
9.4	Elaboração do Caderno de Testes das XSG e execução de Pilotos de Testes.....	69
9.5	Escolha da XML Security Gateway	77
10	Proposta de implementação da infra-estrutura de segurança no ambiente SOA do Banco BPI.....	79
11	Conclusões.....	85
	Referências e Bibliografia	86
ANEXO A:	Standards de Segurança.....	88
	SSL/TLS	88
	XML-Encryption	88
	XML-Signature.....	90
	XACML (<i>eXtensible Access Control Markup Language</i>)	93
	SAML (<i>Security Assertion Markup Language</i>).....	96
	WS-Security.....	100
	WS-SecureConversation	105
	WS-Trust	109
	WS-Federation	114
	XKMS (<i>XML Key Management Specification</i>)	115
	WS-Reliability	115
	WS-ReliableMessaging	116
	WS-Addressing.....	117
	WS-Policy	117
	WS-SecurityPolicy	119

WS-MetadataExchange.....	120
WS-PolicyAttachment.....	121
WS-Privacy.....	122
WS-Authorization.....	122
WS-I Basic Profile (BSP)	122
ANEXO B: RFI (<i>Request for Information</i>) elaborado pelo Banco BPI.....	124
ANEXO C: Lista de requisitos constantes no RFP (<i>Request for Proposal</i>) elaborado pelo Banco BPI	128
ANEXO D: Ferramentas disponíveis para testar a forma como as XSG detectam algumas situações de ataques	132

Lista de Figuras

Figura 2.1: <i>Protocol stack</i> dos <i>Web Services</i>	6
Figura 2.2: "Percurso" de uma mensagem SOAP	7
Figura 2.3: Anatomia de uma mensagem SOAP	9
Figura 2.4: Exemplo da estrutura de uma mensagem SOAP	10
Figura 2.5: Registo/publicação, descoberta, "contrato" e utilização dos Serviços.....	14
Figura 3.1: Níveis de segurança de uma arquitectura SOA.....	16
Figura 5.1: Alguns dos <i>standards</i> de segurança aplicados aos <i>Web Services</i>	33
Figura 6.1: Mensagens SOAP e equipamentos da infra-estrutura de segurança	34
Figura 6.2: <i>XML Security Gateway</i> – interacção com os WS.....	37
Figura 6.3: Transformação XSLT	38
Figura 10.1: Proposta da infra-estrutura de segurança no ambiente SOA do Banco BPI	79
Figura A. 1: Modelo da linguagem das políticas XACML	94
Figura A. 2: Configuração ponto-a-ponto (SSL, ...).....	101
Figura A. 3: Configuração <i>end-to-end</i> (WS-Security)	101
Figura A. 4: Componentes de segurança WS-Security de uma mensagem SOAP	103

Lista de Tabelas

Tabela 6-1 – Correspondência dos <i>bindings</i> predefinidos do WCF às opções de transporte...	42
Tabela 6-2 – Valores para cada um dos <i>bindings</i> WCF predefinidos	43
Tabela 6-3 – Suporte dos tipos de credenciais por cada um dos <i>bindings</i> WCF.....	45
Tabela 7-1 – Tecnologias/ <i>standards</i> e respectivos benefícios de segurança obtidos com a sua utilização.....	55
Tabela 8-1 – Funções de <i>proxy</i> relacionadas com a autenticação	61

Abreviaturas e Símbolos

ACL	<i>Access Control List</i>
CORBA	<i>Common Object Request Broker Architecture</i>
DCOM	<i>Distributed Common Object Model</i>
DNS	<i>Domain Name Server</i>
DoS	<i>Denial of Service</i>
DTD	<i>Document Type Definition</i>
FTP	<i>File Transfer Protocol</i>
HTML	<i>HyperText Markup Language</i>
HTTP	<i>HyperText Transfer Protocol</i>
HTTPS	<i>HyperText Transfer Protocol Secure</i>
IDS	<i>Intrusion Detection System</i>
IETF	<i>Internet Engineering Task Force</i>
LDAP	<i>Lightweight Directory Access Protocol</i>
NAT	<i>Network Address Translation</i>
OASIS	<i>Organization for the Advancement of Structured Information Standards</i>
PKI	<i>Public Key Infrastructure</i>
QoS	<i>Quality of Service</i>
RADIUS	<i>Remote Authentication Dial-in User Service</i>
REST	<i>REpresentational State Transfer</i>
RFI	<i>Request for Information</i>
RFP	<i>Request for Proposal</i>
RMI	<i>Remote Method Invocation</i>
RPC	<i>Remote Procedure Call</i>
SCT	<i>Security Context Token</i>
SLA	<i>Service Level Agreement</i>
SMTP	<i>Simple Mail Transport Protocol</i>
SOA	<i>Service Oriented Architecture</i>
SOAP	<i>Simple Object Access Protocol</i>
SSL	<i>Secure Sockets Layer</i>
SSO	<i>Single Sign-On</i>
STS	<i>Security Token Services</i>
TLS	<i>Transport Layer Security</i>
UDDI	<i>Universal Description, Discovery and Integration</i>
URI	<i>Uniform Resource Identifier</i>
VLAN	<i>Virtual Local Area Network</i>
W3C	<i>World Wide Web Consortium</i>
WAN	<i>Wide Area Network</i>
WCF	<i>Windows Communication Foundation</i>
XML	<i>eXtensible Markup Language</i>
WS	<i>Web Service</i>
WSD	<i>Web Service Description</i>
WSDL	<i>Web Services Description Language</i>
XSG	<i>XML Security Gateway</i>
XSLT	<i>eXtensible Stylesheet Language Transformation</i>

1 Introdução

Com o objectivo de desenvolver uma arquitectura cujos protocolos permitissem a interacção entre aplicações e sistemas, independentemente do tipo de ambiente e arquitectura onde estivessem inseridos, foi criado um grupo de trabalho no W3C onde estavam incluídos representantes das maiores empresas da área dos Sistemas de Informação (Microsoft, IBM, Oracle, Sun, etc.). Este grupo de trabalho criou o conceito de SOA (*Service Oriented Architecture*) como sendo um estilo de desenvolvimento e integração de *software*. Segundo a OASIS, a definição sumária de SOA é a seguinte:

“Service Oriented Architecture is a paradigm for organizing and utilizing distributed capabilities that may be under the control of different ownership domains. It provides a uniform means to offer, discover, interact with and use capabilities to produce desired effects consistent with measurable preconditions and expectations.” [14]

SOA sugere uma nova forma de pensar relativamente à forma como os sistemas informáticos são desenhados, usados, geridos e combinados. SOA está relacionada com os processos de negócio, isto é, nestas arquitecturas são os processos de negócio que conduzem a definição e criação dos serviços. Algumas das características associadas aos serviços SOA são as seguintes: independentes da plataforma, reutilizáveis, partilháveis, detectáveis e oferecem uma infra-estrutura de acesso baseada em *standards*.

Entretanto, surgiu também o conceito de *Web Services* (WS) como sendo um modelo que permite a interacção entre processos de negócio de uma forma mais dinâmica e, consequentemente, facilitando a comunicação entre processos de empresas diferentes.

SOA e WS são dois conceitos que permitem resolver os problemas de integração das aplicações, tanto dentro das empresas, como entre elas. No entanto, é importante ter em conta que a utilização de WS não implica estarmos perante uma arquitectura SOA e que uma arquitectura SOA pode ser implementada recorrendo a outras tecnologias diferentes dos WS. Ou seja, definindo de uma forma simplista SOA, podemos considerar que é um conceito ou estilo de arquitectura que usa um conjunto de “serviços” para alcançar os seus objectivos. O “serviço” é um sistema autónomo (negócio) que aceita um ou mais pedidos e retorna uma ou mais respostas através das suas *interfaces*, que são publicadas e bem definidas.

A implementação da Arquitectura Orientada a Serviços utilizando WS permite um grau de flexibilidade que não era alcançado através das tentativas anteriores usando, por exemplo, as tecnologias CORBA, RMI e DCOM. Como exemplo, podemos considerar que um dos requisitos dos ambientes onde são utilizadas algumas destas tecnologias, é o facto da gestão dos componentes existentes nos sistemas que interagem entre si ter de ser efectuada por uma única organização e a localização dos componentes não poder variar. Ou seja, estes cenários funcionam bem dentro da organização mas, quando é necessário ultrapassar as fronteiras da organização, as limitações destas tecnologias tornam-se mais visíveis. Com esta nova arquitectura podem ser criados novos modelos de negócio que permitem fornecer novos produtos e serviços para promover uma melhor integração entre ambientes diferentes, tanto externos como dentro da mesma empresa.

No passado, de forma a permitirem a integração das aplicações, foram adoptadas várias opções de implementação através do recurso a transferências de ficheiros, base de dados partilhadas, chamadas remotas de procedimentos (RPC) e mensagens. Embora algumas destas

aproximações façam sentido em determinados contextos, a utilização de mensagens, tal como normalmente é implementado nas SOA, traz bastantes vantagens: integração de plataformas diferentes, possibilidade de utilizar comunicações assíncronas¹, possibilidade de garantir confiança nas comunicações (princípio de entrega “*store-and-forward*” que permite maior confiança do que o RPC), segurança *end-to-end* (as mensagens podem transportar o contexto de segurança completo), etc..

Através da implementação de SOA, é possível aumentar a velocidade de resposta à implementação de novas tecnologias que permitem melhorar as funcionalidades disponíveis para o negócio. No entanto, com este tipo de arquitectura, também se verifica uma maior complexidade relativamente aos requisitos de segurança a aplicar aos sistemas e aplicações. Devido à necessidade de partilha de informação e processos entre as organizações, foram introduzidas novas formas de interacção entre os componentes da arquitectura, surgindo também novas formas de ataques aos sistemas. Como a maior parte das *firewalls* não estavam preparadas para protegerem a infra-estrutura destes novos riscos de segurança associados à exposição dos sistemas informáticos e dos respectivos dados, a garantia da segurança dos WS tornou-se uma das principais preocupações a ter em conta neste tipo de arquitecturas. Para tornar esta arquitectura mais segura, existem políticas de segurança que deverão ser seguidas durante a implementação dos respectivos serviços e que podem ser aplicadas recorrendo também a equipamentos de infra-estrutura dedicados a este tipo de soluções. Portanto, organizações tais como o IETF, OASIS e W3C têm contribuído para a criação de *standards* de segurança que podem ser aplicados a este tipo de arquitectura. Nestes *standards* são abordadas medidas relacionadas com autenticação, autorização, relações de confiança/federação/delegação, integridade, confidencialidade, privacidade, segurança dos canais de comunicação e auditoria de todos os processos envolvidos. Neste conjunto de *standards* estão incluídos: XML-Encryption, XML-Signature, WS-Security, WS-SecureConversation, WS-Trust, WS-Policy, etc..

Com o objectivo de permitir implementar algumas das medidas de segurança, surgiram as *XML Security Gateways*. Estes equipamentos de infra-estrutura permitem implementar políticas de segurança para garantirem a protecção dos *Web Services* perante potenciais ataques. Alguns destes ataques tentam explorar vulnerabilidades que podem existir nos WS devido à sua própria natureza: permitir a integração de aplicações para além das fronteiras das próprias empresas, expondo algumas funcionalidades e informação a aplicações externas e domínios de segurança diferentes.

Tal como tem acontecido na maior parte das empresas, no Banco BPI também surgiu a necessidade de expor, através da *Internet*, alguns dos seus WS a parceiros de negócio. Com esta necessidade de exposição externa, torna-se imprescindível a criação de uma infra-estrutura ainda mais segura que permita a publicação dos WS para o exterior. Embora algumas das medidas de segurança já estejam a ser implementadas na exposição dos WS para consumo interno, como a *Internet* é ainda mais “insegura” do que a rede interna e será necessário identificar/autenticar/autorizar acessos de entidades externas ao Banco, passa a ser necessário tomar medidas de segurança adicionais e efectuar algum investimento,

¹ O emissor e receptor das mensagens não necessitem de estar, em simultâneo, disponíveis e a processar as mensagens.

nomeadamente, na aquisição de *XML Security Gateways*. O estudo constante neste relatório tenta enumerar algumas das tecnologias envolvidas nas arquitecturas SOA, listar alguns dos riscos aos quais elas poderão ficar expostas, formas de tentar garantir a segurança deste tipo de arquitecturas, descreve o processo de avaliação/escolha de algumas *XML Security Gateways* disponíveis no mercado e uma proposta de implementação de uma infra-estrutura de segurança num ambiente SOA real.

2 Web Services (WS)

2.1 Tipos de Web Services: REST versus SOAP Web Services

Podemos identificar como sendo as duas principais classes de *Web Services* usados em arquitecturas SOA, os SOAP (*Simple Object Access Protocol*) e REST (*REpresentational State Tranfer*):

- SOAP *Web Services* – Esta é a forma mais comum de WS utilizada nas empresas e, portanto, muitas vezes o termo *Web Service* é associado só a SOAP e serviços descritos através de WSDL. Nos SOAP *Web Services*, com excepção dos dados em anexos binários, todas as mensagens são transmitidas através de SOAP e a descrição do serviço deve ser em WSDL. SOAP permite a criação de uma mensagem que pode ser trocada através de um grande número de protocolos básicos (HTTP, FTP, SMTP, etc.). Ou seja, SOAP é uma espécie de envelope que “encapsula” o conteúdo dessas mensagens. Uma das vantagens do SOAP é o facto de permitir a troca de mensagens “ricas”, que vão desde simples perguntas-e-respostas, a *broadcast* e correlação dessas mensagens. Existem dois tipos de SOAP *Web Services*:
 - *Document-centric* SOAP *Web Service* (são SOA).
 - SOAP RPC (não são SOA). – Neste tipo de WS, as mensagens RPC são codificadas em mensagens SOAP, transformando *interfaces* RPC específicas de determinadas aplicações numa *interface* genérica. Como o comportamento dos sistemas distribuídos é difícil de fixar, as operações criadas com SOAP RPC, pela sua natureza, não interagem automaticamente. Perante esta dificuldade, como RPC tem tendência a ser “instrutivo” em vez de “descritivo”, o que é contra o princípio de SOA, este tipo de WS não são considerados compatíveis com SOA.
- REST *Web Services* – Este tipo de WS é baseado em SOA tendo em conta o conceito de “recurso” (fonte de determinada informação), que é representado através de um URI. O principal objectivo destes serviços é a manipulação de representações XML dos recursos *Web*, usando um conjunto uniforme de operações *stateless*. Os REST WS seguem os seguintes princípios:
 - Os nomes dos recursos são referenciados através de URLs.
 - As *interfaces* são limitadas a HTTP (não usam SOAP). Os únicos quatro métodos disponíveis são a criação, obtenção, alteração e eliminação dos recursos através do uso de HTTP PUT, HTTP GET, HTTP POST e HTTP DELETE.

Os REST WS são considerados simples e eficientes porque HTTP é a *interface* mais comum e é considerada suficiente, em termos de funcionalidades, para a maior parte das aplicações. Muitas vezes, a simplicidade do HTTP traz mais vantagens do que a complexidade de introduzir uma camada de transporte de rede adicional. A criação dos REST WS foi uma reacção à complexidade do SOAP. No entanto, apesar de ser simples de implementar, está limitado ao HTTP.

Ambas as classes de WS usam URIs para identificar os recursos e utilizam protocolos *Web* (HTTP) e formatos XML para os dados da mensagem. WSDL v.2.0 já permite que os serviços sejam definidos tanto como SOAP ou REST. SOAP 1.2 suporta tanto serviços REST (HTTP GET) como SOAP. Num documento do W3C é referido o seguinte relativamente a SOAP v1.2:

*“It should be noted that SOAP 1.2 **can** be used in a manner **consistent** with REST. However, SOAP 1.2 **can** also be used in a manner that **is not consistent** with REST.” [26]*

Actualmente, existem muitas especificações WS-* criadas para adicionarem determinadas funcionalidades aos WS SOAP. No entanto, como essas especificações foram criadas para serem usadas sobre SOAP, não é possível aplicá-las nos WS REST. Embora os utilizadores dos WS REST argumentem que não precisam dessas especificações pois confiam na segurança e garantia das comunicações, é sempre necessário implementar algumas medidas de segurança para criar uma infra-estrutura minimamente segura.

A não ser que seja indicado especificamente, neste relatório, o uso do termo *Web Service* refere-se a SOAP *Web Service*.

2.2 SOAP Web Services (ou simplesmente, Web Services)

Segundo o grupo de trabalho W3C *Web Services Architecture*, a definição de *Web Service* (WS) é a seguinte:

“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.” [26]

A arquitectura de *Web Services* é baseada no conceito de distribuição e modularidade, seguindo padrões e protocolos abertos, de forma a permitir facilmente a integração de aplicações. As suas *interfaces* são baseadas em protocolos *Internet*, tais como HTTP (o mais comum), SMTP, FTP, etc.. Com excepção dos dados binários dos anexos, todas as mensagens devem ser em XML.

Normalmente, os WS são construídos com base nos seguintes *standards* (Figura 2.1):

- XML (*Extensible Markup Language*) – Um *standard* do formato das mensagens.
- SOAP (*Simple Object Access Protocol*) – Um *standard* do protocolo de comunicações usado na invocação dos serviços usando tecnologias *Web*, tais como, HTTP e XML.
- XSD (*XML Schema Definition*).
- WSDL (*Web Services Description Language*) – Um *standard*, em formato XML, usado para descrever as operações e parâmetros suportados por um determinado serviço. Através da análise do ficheiro WSDL de um determinado WS, este pode ser descoberto e os seus métodos invocados.

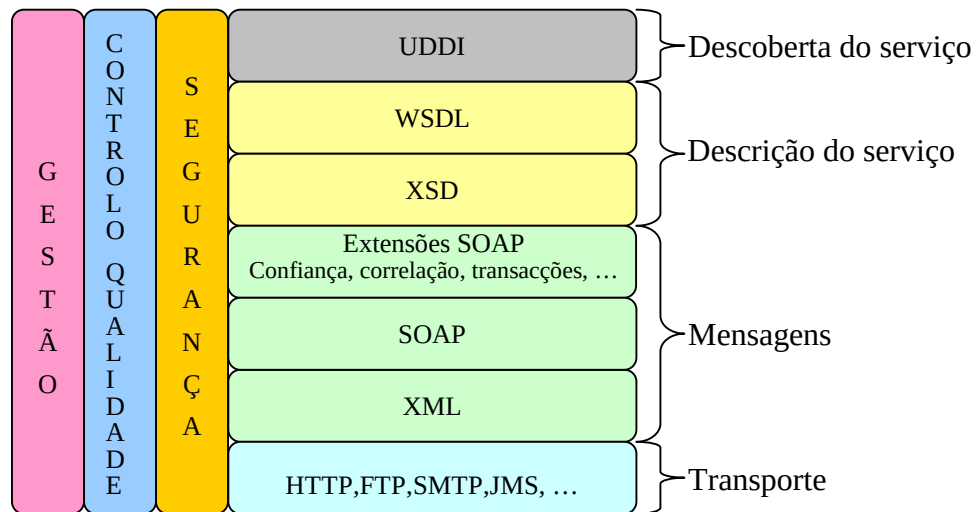


Figura 2.1: Protocol stack dos Web Services

- UDDI (*Universal Description, Discovery and Integration*) – Um *standard* utilizado para registar, descobrir e localizar os *Web Services*.

Seguindo estes *standards*, é possível que os WS residam em qualquer lugar e sejam acedidos por qualquer um. Ao contrário dos *Web Sites*², os WS são serviços de interacção “servidor-para-servidor”, através de formatos e protocolos já definidos, independentes da plataforma e que utilizam uma linguagem “neutra”. Para garantir a fiabilidade da arquitectura onde estão integrados os WS é necessário tomar medidas que garantam:

- Controlo de qualidade. – Contabilizar as falhas ocorridas durante a execução do WS e verificar se o tempo de resposta do serviço está dentro do preestabelecido.
- Segurança. – Controlar os acessos ao serviço e registá-los, garantir a integridade e confidencialidade dos dados, proteger contra possíveis ataques à infra-estrutura, etc..
- Gestão. – Permite que o fornecedor do WS monitorize e controle estatísticas, históricos e parâmetros do serviço.

2.3 Tecnologias usadas na implementação dos *Web Services*

Numa arquitectura baseada em *Web Services* são utilizadas diversas tecnologias que interagem entre si e permitem implementar funcionalidades em diversos níveis, nomeadamente: XML, SOAP, WSDL. Estes *standards* são utilizados na troca de mensagens entre serviços e podemos considerar a seguinte analogia com o serviço dos correios postais:

WSDL	→	Entrada na lista de endereços
Protocolo comunicações (ex.:http)	→	Carteiro (transporte)
SOAP	→	Envelope (encapsulamento)
XML	→	Carta (mensagem)

² Tecnologias baseadas na interacção através de um *browser* ou dependentes da plataforma.

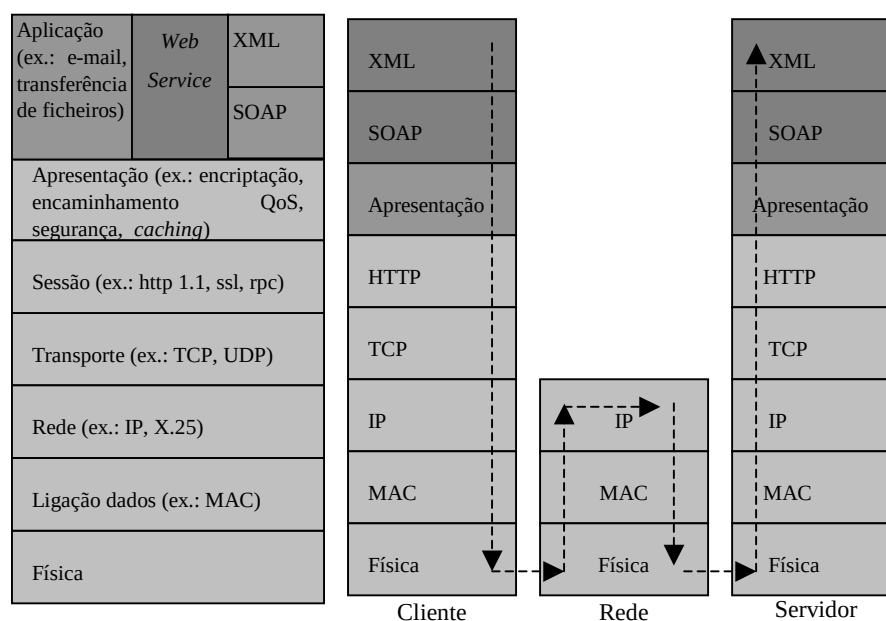


Figura 2.2: "Percurso" de uma mensagem SOAP

(baseado num documento da F5 [24])

Tendo como base o modelo OSI, a camada de “apresentação” permite fornecer alguma independência na representação dos dados através da tradução entre o formato “aplicacional” e de “rede”, e vice-versa (Figura 2.2). Por exemplo, a camada de “apresentação” transforma os dados num formato aceite pela camada aplicacional e formata-os e encripta-os de forma a poderem ser enviados através da rede. Como com este modelo é possível fornecer um grau de liberdade que permite resolver os problemas de incompatibilidade entre os sistemas, muitas vezes esta camada é denominada “camada da sintaxe”.

2.3.1 XML (eXtensible Markup Language)

XML (*eXtensible Markup Language*), uma recomendação formal do W3C, é a base de todos os modelos de *standards* utilizados na implementação de WS. É uma linguagem usada para definir a forma como os dados trocados entre aplicações devem ser estruturados pois, através de um formato único de apresentação, tanto permite descrever os próprios dados como também a forma de os apresentar. Enquanto que o HTML é usado para definir a apresentação/formato da informação nas páginas da *Internet* (sem contexto ou comportamento dinâmico), o XML permite implementar a noção de contexto e dar significado aos dados (foca-se no conteúdo).

O XML é baseado em declarações, que não estão predefinidas, no formato de “etiquetas” (*tags*), que têm de ser abertas e fechadas – as mensagens XML “transportam” a sua própria *metadata*. Podem existir etiquetas dentro de outras etiquetas e o XML deve ter um nó raiz. O XML é classificado como sendo uma linguagem extensível precisamente pelo facto de permitir que os próprios utilizadores definam as suas próprias *tags*.

Devido ao XML poder ser definido na forma de texto, ter marcações bem definidas e dada a sua natureza hierárquica, é facilmente entendido tanto por um ser humano como por um programa informático.

Exemplo:

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<carro xmlns="http://www.empresaxxx.com"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://www.empresaxxx.com carro.xsd" >
    <fabricante>Fiat</fabricante>
    <modelo>Punto</modelo>
    <ano>2008</ano>
    <cor>azul</cor>
    <descricao>Carro em bom estado</descricao>
</carro>
```

Os documentos XML são definidos através de DTD (*Document Type Definition; standard* antigo) ou XSD (*XML Schema Definition; standard* actual). Os XSD utilizados para definir os formatos dos ficheiros XML podem ser os do próprio *standard* ou podem ser adaptados. Os esquemas XML são usados para:

- Definir a relação, ordem e número de elementos (no exemplo, o modelo é um elemento do carro e só existe um modelo aplicável a cada carro).
- Definir os tipos de dados e quais os valores admissíveis (no exemplo, a cor deve ser uma cadeia de caracteres que pode conter [A-Z][a-z]).

Através da validação dos esquemas XML e confirmação da sua correcta formatação, podem ser evitados muitos dos ataques XML.

Exemplo de esquema XML:

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
targetNamespace="http://www.empresaxxx.com"
xmlns="http://www.empresaxxx.com"
elementFormDefault="qualified">
<xs:element name="carro">
    <xs:complexType>
        <xs:sequence>
            <xs:element name="fabricante" type="xs:string"/>
            <xs:element name="modelo" type="xs:string"/>
            <xs:element name="ano">
                <xs:simpleType>
                    <xs:restriction base="xs:integer">
                        <xs:minInclusive value="1900"/>
                        <xs:maxInclusive value="2050"/>
                    </xs:restriction>
                </xs:simpleType>
            </xs:element>
            <xs:element name="cor" type="xs:string"/>
            <xs:element name="descricao" type="xs:string"/>
        </xs:sequence>
    </xs:complexType>
</xs:element>
</xs:schema>
```

2.3.2 SOAP (*Simple Object Access Protocol*)

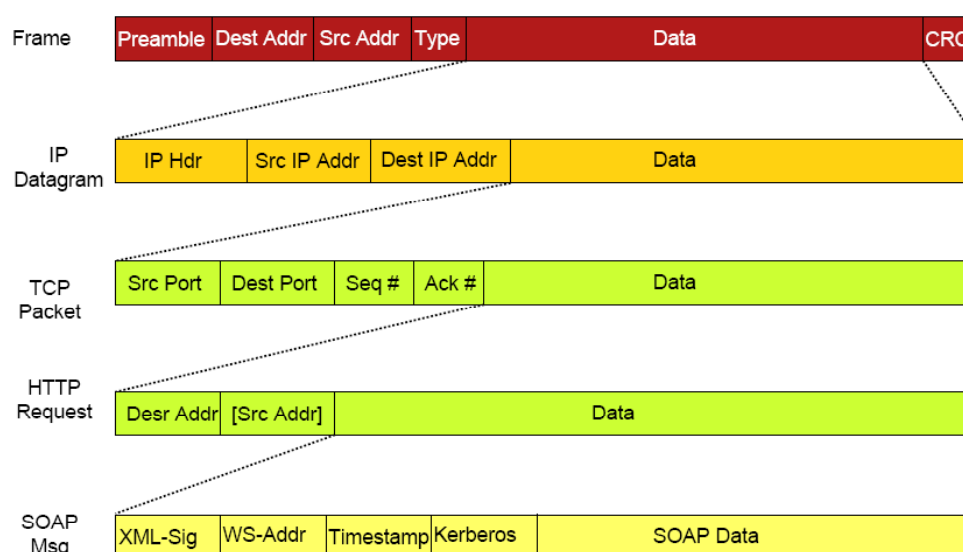


Figura 2.3: Anatomia de uma mensagem SOAP

(Fonte: Cisco [19])

SOAP (*“Simple Object Access Protocol”*) é um protocolo de comunicação *connectionless*, baseado em XML, usado na troca de mensagens entre as aplicações. SOAP foi criado para ser usado nas comunicações via *Internet* e é independente da linguagem e plataforma. É um protocolo simples e extensível. As mensagens SOAP são independentes do protocolo de transporte, ou seja, podem ser “transportadas” por uma considerável variedade de protocolos, tais como: HTTP, SMTP, FTP, RMI/IIOP ou outros protocolos proprietários. Actualmente, a forma mais comum de troca de mensagens SOAP é através de HTTP que, também é o único protocolo suportado pelo *WS-I Basic Profile*.

Como o SOAP é usado para invocar métodos em componentes de *software* remotos usando um “vocabulário” XML, este “vocabulário” é utilizado para descrever o método remoto que será chamado, os parâmetros necessários, os valores retornados e as excepções. Ou seja, SOAP é um protocolo de mensagens, baseado em XML, que encapsula os dados RPC em pedidos e respostas WS, normalmente transportadas por HTTP (Figura 2.3).

Uma mensagem SOAP é composta por quatro partes (três delas são opcionais; Figura 2.4):

- **Envelope SOAP.** – É um documento XML que encapsula a mensagem que está a ser transmitida. É a única parte da mensagem SOAP que é obrigatória; todas as outras são opcionais. É constituído por um ou mais cabeçalhos e pelo corpo SOAP.

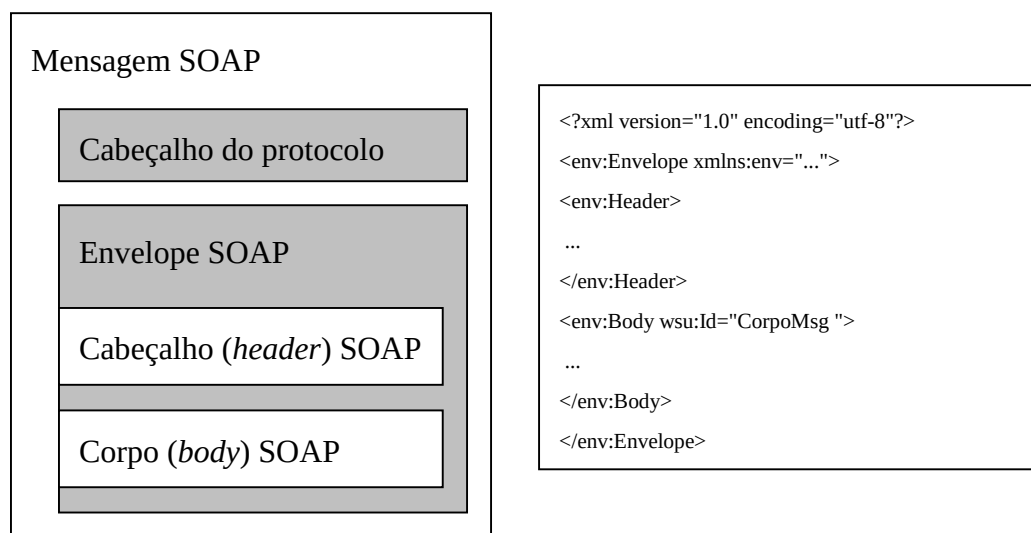


Figura 2.4: Exemplo da estrutura de uma mensagem SOAP

- Conjunto de regras para representar os tipos de dados definidos na aplicação. – Esta parte da mensagem SOAP é usada para definir os tipos de dados proprietários que a aplicação está a usar. Como XSD (*XML Schema Definition*) é uma linguagem descritiva independente da plataforma, SOAP usa-a para definir os seus esquemas de dados através da descrição da estrutura e tipos da informação, recorrendo a um sistema uniforme de tipos.
- Convenção para representar as chamadas remotas dos procedimentos (RPC) e respectivas respostas.
- Conjunto de regras onde é definida a forma como usar SOAP com HTTP 1.1.

O cabeçalho SOAP é uma estrutura predefinida, existente na mensagem XML, onde são indicadas as informações sensíveis ao contexto, incluindo tanto os *tokens* de segurança (ex.: SAML) como outra informação que se espera ser necessária para o processamento que será efectuado pelos intermediários ou *end-points* (informação sobre encaminhamento da mensagem, instruções para o processamento, *timestamp*, atributos relacionados com a qualidade do serviço, etc.).

No corpo da mensagem está incluída a “mensagem” que tem de ser trocada que, normalmente, é uma chamada remota de um método. É também no corpo SOAP que são incluídas as mensagens relativas às falhas SOAP.

2.3.3 WSDL (*Web Services Description Language*)

WSDL é uma linguagem que permite criar um documento, escrito em XML, que é usado para localizar e descrever os WS: o que é que o serviço pode fazer, como é que pode ser acedido e onde pode ser encontrado. Desta forma, o consumidor e o fornecedor do WS podem interagir correctamente durante a execução de uma determinada tarefa. O documento WSDL contém a informação de todos os requisitos necessários para aceder ao WS através da descrição dos seguintes items:

- Localização do serviço. – URLs que definem o protocolo usado (esquema do transporte), o nome/endereço IP da máquina onde o serviço está a correr e o caminho para o serviço. A porta tcp/udp é um campo opcional do URL (normalmente, quando é por HTTP a porta é a 80, HTTPS é 443, etc.).

Exemplo de URL: <http://ws.empresa.pt/nomeservico.asmx>

- Tipos de dados que serão utilizados pelo WS (ex.: de acordo com o esquema XML, uso de *string*, *integer*, etc.).

- Caracterização das mensagens trocadas. Exemplo:

```
<message name="GetModeloRequest">
  <part name="parameters" type="xsd:string"/>
</message>
<message name="GetModeloResponse">
  <part name="return" type="xsd:string"/>
</message>
```

- Definição das mensagens que definem uma conversação (portType). Exemplo:

```
<portType name="GetModeloPortType">
  <operation name="GetModelo">
    <input
      message="tns: GetModeloRequest"
      name="GetModelo"/>
    <output
      message="tns: GetModeloResponse"
      name="GetModeloResponse"/>
  </operation>
</portType>
```

- Detalhes de como o WS é implementado no SOAP (*bindings*). – Através dos *bindings* podem ser controlados:
 - o O transporte (HTTP, MSMQ, Named Pipes, Net TCP, etc.).
 - o Os canais (só num sentido, nos dois sentidos, pedidos-respostas).
 - o A codificação (XML, binária, MTOM (*Message Transmission Optimization Mechanism*), etc.).
 - o Os *standards* WS-* suportados (WS-Security, WS-Federation, WS-Reliability, WS-Transactions, etc.).

Exemplo: uso de tipos de documentos RPC que usam determinados *namespaces*.

Normalmente, os documentos WSDL são gerados automaticamente, sem intervenção humana, pela *framework* do próprio WS. No documento WSDL, normalmente, não estão especificados os controlos a nível de quem é que está autorizado a aceder a essa informação.

De uma forma geral, para aceder ao WSDL de um determinado WS basta adicionar o argumento “?wsdl” no final do URL (ex.: ws.comp.pt/nomeservico.asmx?wsdl). Em alternativa, o WSDL pode ser disponibilizado na forma de um ficheiro que é trocado entre as entidades (SMTP, FTP, etc.).

2.3.4 UDDI (*Universal Description, Discovery and Integration*)

UDDI é um directório, independente da plataforma, aprovado pela OASIS. Nesse directório é guardada a informação relativa às interfaces dos WS, descritas através de WSDL, permitindo que os WS sejam localizados facilmente e, em seguida, possam ser invocados. UDDI comunica através de SOAP e é, ele mesmo, um WS.

Os servidores UDDI suportam autenticação e controlo dos acessos mas, esta configuração nem sempre é a que está configurada por defeito. Como muitas vezes a informação constante neste directório pode ser utilizada para originar alguns ataques aos WS, ao implementar os UDDI, é necessário identificar bem quais as “fronteiras” que devem ser definidas nestes registos pois eles podem ser: públicos, internos à empresa ou só devem estar disponíveis para alguns parceiros de negócio.

UDDI é uma espécie de “páginas amarelas” onde podem ser efectuadas pesquisas por empresa, procurar os serviços oferecidos por cada uma delas, obter informação técnica relativa ao WS que é disponibilizada via WSDL, contactar as empresas para obter mais informação, etc.. Existem seis tipos de dados nos registos UDDI que fornecem toda a informação necessária:

- BusinessEntity – Nome, endereço e informação de contacto da organização.
- BusinessService – Dados técnicos relativos aos serviços e produtos disponibilizados pela organização.
- BindingTemplate – Disponibiliza detalhes relativos a um determinado WS que é disponibilizado.
- TModel – Define como interagir com o WS de destino, nomeadamente, protocolo a usar, formatos usados na troca de mensagens, regras relativas à sequência das trocas, etc. (ex.: WSDL).
- PublisherAssertion – Descreve as relações entre as entradas businessEntity.
- Subscription – Descreve o pedido para monitorização de alterações relativas a determinadas entidades.

Os directórios UDDI também incluem várias formas de pesquisa que permitem criar vários tipos de filtros: localização geográfica, tipo de negócio, determinado fornecedor, etc..

Originalmente, o principal foco do UDDI era o UBR (*UDDI Business Registry*), um directório público de WS, composto por servidores suportados por várias empresas (ex.: IBM, Microsoft e SAP), que disponibilizava *interfaces* para a pesquisa dos WS e onde era possível o registo dos serviços de qualquer empresa. Como a maior parte dos WS não são para uso público, foi necessário expandir a especificação UDDI para incluir implementações privadas dos registos UDDI através de *UDDI Registry* privados. Os *UDDI Registry* privados disponibilizam mecanismos que permitem que os utilizadores e aplicações internas descubram e acedam aos WS da organização com pouca, ou nenhuma, interacção humana. Existem quatro tipos de *UDDI Registry* privados:

- Registo de serviços baseados na *Internet*. – Registos alojados por um grupo de organizações que fornecem informação relativa a essas organizações para consumo público.
- Portal de registos. – Reside numa rede pública e fornece informação relativa aos WS de uma determinada organização.
- Catálogo de registos dos parceiros. – Reside na rede interna da empresa e fornece informação relativa aos serviços oferecidos por essa organização e pelos seus parceiros de negócio.

- Registo de serviços internos. – Reside na rede interna e fornece informação relativa aos WS oferecidos dentro dessa rede.

Adicionalmente, UDDI v.3 permite definir privilégios de acesso de forma a só permitir os acessos, a determinados registos, a utilizadores autorizados. Na v.3 também passou a ser possível aplicar assinaturas XML aos registos de forma a permitir a verificação da integridade e identificação da entidade que criou esses registos.

Entretanto, após cinco anos de existência, o UBR foi descontinuado em 2005. O UBR surgiu integrado no projecto UDDI como forma de provar a robustez e interoperabilidade das especificações UDDI através de uma implementação com visibilidade pública. Como esse objectivo inicial foi atingido com grande sucesso, foi um grande impulsionador para a criação das especificações UDDI actualmente aprovadas pela OASIS e para o facto de alguns fornecedores terem desenvolvido UDDI *Registries* para resolver alguns dos desafios resultantes da necessidade de integração das aplicações e serviços.

2.4 Funcionamento de um modelo baseado em *Web Services*

Web Service é um termo abstracto que, na realidade, é implementado através de um agente que envia e recebe mensagens. Esse agente pode ser uma peça de *software* ou de *hardware*. O serviço é o conjunto de funcionalidades disponibilizadas por esse agente.

O objectivo do WS é permitir que uma determinada entidade (fornecedor do WS) disponibilize algumas funcionalidades que são utilizadas por uma outra entidade (consumidor do WS). Para que esta troca de mensagens seja efectuada com sucesso, é necessário que a entidade que disponibiliza o serviço e a que vai consumi-lo acordem previamente a semântica e o mecanismo utilizado na troca dessas mensagens. O mecanismo de troca das mensagens está documentado na especificação da *interface* dos WS denominada *Web Service Description* (WSD) e que é escrita em *Web Services Description Language* (WSDL). A semântica do serviço é a expectativa, partilhada pelas várias entidades envolvidas, sobre o comportamento desse serviço, nomeadamente nas respostas às mensagens que lhe são enviadas. Ou seja, existe um acordo entre o fornecedor e o consumidor do WS relativo aos objectivos e consequências desta interacção. Enquanto que a descrição do serviço representa um acordo relativo ao mecanismo de interacção com determinado serviço, a semântica representa um acordo relativo ao significado e objectivos dessa interacção.

O consumidor de um WS pode invocá-lo e usá-lo de várias formas. De uma forma geral, os passos necessários para a utilização de um determinado WS são os seguintes (Figura 2.5):

1. O fornecedor e o consumidor do WS têm de se “conhecer” mutuamente ou, pelo menos, um deles tem de “conhecer” o outro. Este passo pode ser efectuado de duas formas:

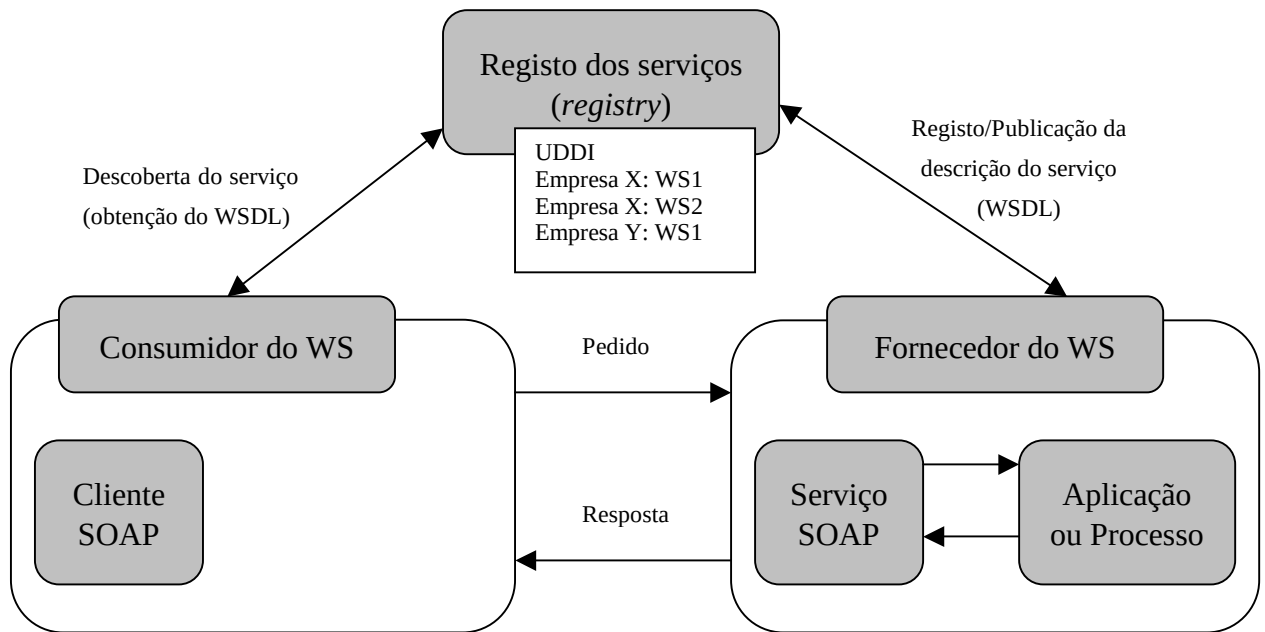


Figura 2.5: Registo/publicação, descoberta, “contrato” e utilização dos Serviços

- O agente do consumidor do WS é o que inicia este processo de procura do endereço do agente da entidade fornecedora do serviço. Este endereço pode ser obtido directamente a partir da entidade fornecedora do serviço ou então, o agente que irá consumir o WS, poderá usar um serviço de descoberta (manual ou automática) para localizar, a partir de uma descrição associada a esse serviço (funcional ou não), a descrição do serviço que estava a ser procurado – a descrição do serviço (WSD) contém o endereço do agente. Neste segundo caso, o serviço de descoberta pode usar um motor de pesquisa que percorrer a rede, recolhendo WSDs até encontrar o pretendido ou, caso o serviço de descoberta seja implementado como “registry” (ex.: UDDI), será necessário que o fornecedor do serviço publique num directório, que possa ser acedido pelos clientes, a descrição do respectivo serviço e da respectiva função.
 - O agente do fornecedor do serviço é que inicia o processo de troca de mensagens entre os dois agentes. Neste caso, o agente do fornecedor do serviço consegue obter o endereço do agente consumidor do WS. Este é um método menos utilizado e dependente das próprias aplicações. Normalmente, é usado em cenários de “push” ou de subscrições de serviços.
2. O fornecedor e o consumidor do WS têm de chegar a acordo relativamente à descrição e semântica que serão aplicadas às interações entre os dois agentes. Este acordo pode ser alcançado através de uma das seguintes formas:
- As duas entidades comunicam directamente uma com a outra de forma a acordarem qual a semântica e descrição do serviço.
 - A entidade fornecedora do serviço publica a descrição e semântica do serviço e, caso a entidade consumidora do serviço o queira utilizar, tem de aceitar as condições de uso sem poder negociar qualquer alteração.

- A descrição e semântica do serviço (com excepção do endereço) podem ser definidas como sendo um *standard*, definido por uma organização, que é usado por várias entidades. Neste caso, o acordo entre as duas entidades é assumido automaticamente pois as duas seguem o mesmo *standard* relativamente a este item.
 - A descrição e semântica do serviço (por vezes com excepção do endereço) é definida e publicada pela entidade consumidora do serviço que a apresenta à entidade fornecedora do serviço. Também neste caso não é possível efectuar qualquer alteração à proposta apresentada. Esta abordagem, por exemplo, pode ser usada nos casos em que uma empresa exige que os seus fornecedores disponibilizem WS que sigam uma determinada descrição e semântica.
3. A descrição e semântica do serviço têm de ser compreendidas por ambos os agentes (o cliente “contrata” ao fornecedor do WS esse serviço, ficando apto a poder usá-lo). A informação contida na descrição e semântica do serviço deve ser introduzida e implementada em cada um dos agentes. Estes procedimentos podem ser alcançado de várias formas, como por exemplo:
- O agente pode ter no seu próprio código a forma de implementar só uma determinada semântica e descrição do serviço (fixas e que não podem ser alteradas).
 - O agente é programado de forma a estar preparado para poder aceitar determinadas descrições e/ou semânticas de serviços de uma forma dinâmica. Ou seja, está preparado para poder receber algumas “variações” da descrição e semântica do serviço.
 - Só após a criação do agente é que a descrição e/ou semântica do serviço são geradas, a partir do código do próprio agente ou de determinadas variáveis (ex.: uma determinada aplicação pode analisar determinados ficheiros e gerar a descrição do serviço).
4. Os agentes trocam mensagens SOAP para executarem determinadas tarefas.

3 Problemas de Segurança da Arquitectura Orientada a Serviços (SOA)



Figura 3.1: Níveis de segurança de uma arquitectura SOA

Tal como foi referido na Introdução, apesar das vantagens da utilização dos WS³, com a adopção de Arquitecturas Orientadas a Serviços, os riscos de segurança associados à exposição dos sistemas informáticos e dos dados envolvidos aumentam. Estes riscos de segurança são devido ao facto de se verificar uma maior exposição dos dados, as ameaças de segurança associadas aos protocolos base (ex.: HTTP, SSL/TLS, etc.) também se aplicarem aos WS e à existência de novas ameaças relacionadas com os protocolos ao nível das mensagens (ex.: SOAP e XML).

Normalmente, quando se fala em segurança SOA, associa-se a segurança só ao nível das mensagens em trânsito. Mas, além desta componente, também se deverá ter em conta que existem problemas de segurança relacionados com a autenticação/autorização, cumprimento das políticas de segurança aplicadas aos sistemas distribuídos, integridade e confidencialidade dos dados e sistemas, etc. (Figura 3.1).

Em seguida estão listados alguns exemplos de problemas de segurança que se podem verificar numa arquitectura SOA (esta lista não é exaustiva).

3.1 O HTTP e HTTPS estão permitidos na maior parte das *firewalls*

Normalmente, os protocolos de transporte utilizados são o HTTP e o HTTPS que, estão permitidos na maior parte das *firewalls*. Tem-se provado que a filtragem efectuada pelas *firewalls*, que só se baseiam em análise dos IPs e portas (*IP packet filtering*), não é suficiente para controlar os WS a nível das questões de segurança. Para o provar, alguns *hackers* criaram pacotes com conteúdo malicioso, como por exemplo, *payloads* hostis e executáveis. Ou seja, como este tipo de *firewall* não efectua filtragem a nível aplicacional, este tráfego é considerado como sendo “normal” e, na verdade, são formas de ataques que

³ Ex.: fácil acesso aos dados, facilidade de estabelecimento de ligações entre aplicações, relativa autonomia e independência da plataforma.

podem provocar a indisponibilidade do serviço ou mesmo alteração do próprio sistema onde está instalado o WS.

As *firewalls* que além da filtragem anterior também verificam se o “transporte” (HTTP) está de acordo com o respectivo RFC do HTTP, também não protegem os WS porque não conseguem verificar se a mensagem SOAP tem conteúdo malicioso, se está corrompida ou se contém alguma informação que possa ser prejudicial à aplicação.

Além disso, a maior parte dos IDS (*Intrusion Detection System*) não suportam a detecção de cenários de intrusão baseados em vulnerabilidades das mensagens SOAP.

3.2 O uso de HTTPS/VPNs não é suficiente para garantir a segurança das mensagens

Os mecanismos de segurança tradicionais, tais como *Virtual Private Networks* (VPNs), *Internet Protocol Security* (IPSec) e *Secure Sockets Layer/Transport Layer Security* (SSL/TLS), são tecnologias do tipo “ponto-a-ponto”. Embora estas tecnologias sejam usadas para proteger os WS, este nível de segurança das mensagens não é suficiente nas SOA porque estas arquitecturas são baseadas em esquemas complexos de interacção entre os diversos intervenientes, verificando-se a necessidade de troca de mensagens entre eles. Entre o consumidor e o fornecedor do WS pode existir um número elevado de intervenientes e, muitas vezes, os intermediários têm de aceder a alguma da informação disponível nessas mensagens. Por exemplo, o TLS, que é usado para autenticar e encriptar mensagens *Web*, é inadequado para proteger mensagens SOAP pois foi desenhado para operações entre dois pontos – não é possível utilizar TLS em situações em que um WS, simultaneamente, envie mensagens para múltiplos WS.

Ou seja, como SOAP é um protocolo *connectionless*, as mensagens SOAP têm de ser tratadas por vários intermediários e as tecnologias orientadas à ligação (ex.: SSL) são específicas das comunicações ponto-a-ponto, o uso dessas tecnologias não é suficiente para garantir a confidencialidade e integridade das mensagens que atravessam vários intermediários, perdendo-se o contexto de segurança.

Além das questões anteriores, nas situações em que o “canal seguro” é terminado no servidor onde está a correr o WS, é necessário garantir a gestão das chaves privadas e certificados no próprio servidor.

Quando o conteúdo dos pacotes está encriptado, não é possível criar mecanismos de auditoria em que sejam registados alguns dos detalhes relativos às mensagens, sessões ou medidas de segurança aplicadas ao conteúdo trocado.

3.3 Descoberta de informação sobre os sistemas

Neste tipo de ataques, o objectivo é obter informação relativa aos sistemas onde os serviços residem. Nessa informação estão incluídos: o sistema operativo, a versão dos próprios serviços, níveis de actualizações, localização das cópias de segurança e dos ficheiros temporários, operações disponibilizadas pelo serviço, etc.. Através desta informação, podem ser executados ataques que têm como base as vulnerabilidades já publicadas relativamente a uma determinada versão do sistema operativo ou aplicação.

WSDL scanning

Neste tipo de ataque, através do recurso a ferramentas automáticas, o atacante tenta encontrar e executar todas as operações constantes no documento WSDL, de forma a recolher alguma informação sobre esse WS. Um WSDL não protegido pode revelar toda a *interface* de programação da aplicação (API) do WS, incluindo as partes cuja utilização está desactivada ou as usadas para questões de diagnóstico/testes. Se o WSDL contiver mais informação pública do que a estritamente necessária, através desta, pode ser possível ter acesso a informação relativa à estrutura e requisitos de segurança do WS. Esta informação pode permitir a execução, com sucesso, de diversos ataques contra esse WS, incluindo a violação da privacidade do próprio WS.

Como muitas vezes as mensagens de erro retornadas pelo WS são bastante explícitas, indicando claramente qual a origem do erro, os atacantes podem também basear-se nessa informação para encontrarem facilmente as falhas existentes no WS, permitindo que ele consiga executar o ataque com sucesso.

Ataques aos UDDI

Através da análise dos registos UDDI mal protegidos, pode ser também possível obter detalhes relativos às funções disponibilizadas por esse WS e como acedê-lo. Como, muitas vezes, os WSDL são gerados automaticamente por utilitários criados para exporem toda a informação disponível relativa a um determinado método, é necessário ter alguma atenção na escolha/utilização desses utilitários de forma a ser possível proteger os WS desse tipo de ataques. Através da utilização de UDDI v.3.0.2 já é possível implementar alguma protecção relativa a este tipo de ataques. Essa versão já permite solicitar a identificação, autenticação e autorização das entidades antes de lhes dar acesso aos registos UDDI.

3.4 Falhas nos processos de autenticação

Por vezes, são exploradas vulnerabilidades existentes nos processos de autenticação que permitem validar a identidade dos utilizadores, serviços ou aplicações. A autenticação é efectuada com base em um ou três mecanismos: “algo que tu tens”, algo que tu sabes” ou “algo que tu és”. Em seguida, estão listados alguns dos ataques que exploram essas falhas.

Brute Force (também denominado “ataque dicionário”)

O ataque “*brute force*” é baseado em processos automáticos de tentativa/erro usados para descobrir os nomes dos utilizadores, palavras passe, chaves de encriptação, números de cartões solicitados nos processos de identificação dos utilizadores, etc..

Como alguns sistemas permitem a utilização de palavras passe ou chaves de encriptação consideradas “fracas” e, muitas vezes, os utilizadores escolhem palavras passe que constam em alguns dicionários, estes cenários são os ideais para serem considerados como alvo destes tipos de ataque. Ou seja, são criados processos automáticos que, com base em dicionários, conseguem gerar bastantes combinações de palavras de forma a tentar descobrir palavras passe válidas. Em cenários em que são usadas chaves de encriptação “fracas” ou pequenas, também é possível gerar todas as chaves possíveis e descobrir qual é a que permite aceder aos recursos.

Autenticação insuficiente

Autenticação insuficiente ocorre quando um WS permite que um atacante aceda a informação sensível, ou a determinadas funções, sem ser devidamente autenticado. De forma a evitar este tipo de ataques, para todos os recursos disponibilizados que sejam considerados não públicos, os WS devem ser implementados de forma a não ser possível aceder directamente sem ser verificada a identidade do consumidor desse recurso. Muitas vezes, estes ataques também podem ser evitados se os recursos não estiverem directamente expostos. Ou seja, a sua localização é “escondida” e só disponibilizada a quem precisa mesmo de aceder, não existindo nenhuma referência a esses recursos a partir dos que são considerados “públicos” (“*Security Through Obscurity*”). No entanto, estes serviços podem ser alvo de ataques de “*brute force*” que permitem listar todos os URLs ou, através da análise das mensagens de erro, descobrir como chegar a esses serviços – estes ataques podem ser efectuados em conjunto com os ataques “*WSDL scanning*”.

Spoofing

O ataque “*Spoofing*” é bastante complexo e explora falhas nas relações de confiança estabelecidas entre os sistemas. Neste tipo de ataque, o atacante assume uma certa identidade pertencente a uma entidade considerada, pelo outro sistema, como sendo de confiança. Como o sistema que está a ser atacado não consegue detectar esta falsa identidade, mantém a conversação normalmente.

3.5 Falhas nos processos de autorização

Os ataques relacionados com os processos de autorização tentam explorar as falhas existentes nos processos que permitem determinar se um utilizador/serviço/aplicação tem as permissões necessárias para executar uma determinada acção ou aceder a determinada informação. Usando várias técnicas, o atacante consegue aceder a essa informação/acção através da elevação dos seus próprios privilégios relativos a essas áreas (ex.: conseguir ter os mesmos privilégios que o *root* ou administrador).

Previsão das credenciais/sessões

Previsão das credenciais/sessões é um método através do qual o atacante consegue deduzir ou descobrir os identificadores únicos de um utilizador ou sessão em particular. Tendo essa informação, o atacante consegue gerar pedidos com os privilégios do utilizador ou da sessão comprometidos.

Autorização insuficiente

Autorização insuficiente ocorre quando é possível aceder a informação sensível ou funções que deveriam requerer controlo das restrições de acesso. Um utilizador/serviço/aplicação, estando já autenticado, não significa que pode aceder, sem controlo, a todos os recursos. A seguir ao processo de autenticação deve existir um processo de autorização que controle o que é permitido fazer.

Fixação da sessão

A fixação da sessão é uma técnica de ataque que força que o número de identificação de uma determinada sessão seja fixado num valor explícito definido pelo atacante. Estas técnicas vão desde exploração de “*Cross-site Scripting*” até bombardeamento do serviço

com pedidos efectuados anteriormente. Nestes casos, o atacante espera que seja iniciada a respectiva sessão e depois assume a identidade do utilizador/serviço/aplicação.

Gestão de Identidades/Autorizações

Normalmente, nas aplicações distribuídas, são usadas as listas de controlo de acessos, grupos de utilizadores e a hierarquia das funções/papéis dos utilizadores. Por exemplo, um utilizador pode pertencer a vários grupos e pode ter determinadas permissões baseadas na combinação desses grupos. Muitas vezes, este tipo de esquema não funciona numa Arquitectura Orientada a Serviços em que os utilizadores e fornecedores de serviços estão distribuídos por vários domínios de segurança independentes.

3.6 Intercepção das mensagens

Replay Attacks

Nos “*Replay Attacks*”, o atacante intercepta a mensagem e é ele que responde ao agente de destino. Nestes ataques, o atacante pode reenviar mensagens que já tinham sido previamente enviadas, ou incluir partes de uma ou mais mensagens previamente enviadas numa nova mensagem (“*replay*” de partes da mensagem).

Estes ataques são difíceis de detectar como sendo uma intrusão pois, o IP de origem é válido, o comportamento dos pacotes na rede também é válido e os pedidos HTTP são bem formatados. No entanto, o comportamento não é legítimo e constitui uma espécie de “intrusão” XML, podendo ser gerados ataques DoS com *payloads* completamente válidos.

Man-in-the-middle

Os ataques do tipo “*Man-in-the-middle*” podem ser efectuados através do comprometimento de um dos intermediários SOAP. Desta forma, o atacante consegue interceptar as mensagens trocadas entre o consumidor e o fornecedor do WS sem que ambas as partes detectem que esse tráfego está a ser analisado ou mesmo alterado.

3.7 Falhas no registo dos acessos aos WS

Muitas vezes, por má configuração do sistema de auditoria ou devido a um ataque bem sucedido que conseguiu desactivar esse serviço, alguns dos acessos aos WS não são registados. Neste caso, torna-se bastante difícil, ou mesmo impossível, conseguir obter a informação relativa a quem acedeu ao WS e que operações foram efectuadas.

3.8 Alteração das mensagens (integridade) e falhas na garantia da confidencialidade

Através da intercepção das mensagens em trânsito, podem ser alterados alguns campos, ou mesmo toda a mensagem, afectando a integridade das mesmas. Estas alterações podem consistir na modificação de partes da mensagem (corpo da mensagem ou do próprio cabeçalho), eliminação/adição de alguns campos, alteração dos anexos, etc..

Além disso, através dessa intercepção das mensagens, algumas entidades que supostamente não deviam ter acesso ao conteúdo das mesmas, conseguem fazê-lo através de várias tentativas ou mesmo falha no mecanismo de segurança relacionado com a protecção da própria mensagem (acesso aos WS por utilizadores não autenticados ou não autorizados). Em seguida estão listados alguns tipos de ataques incluídos neste grupo.

Parameter tampering

Os parâmetros são usados para passar informação do cliente para o WS de forma a permitir a execução remota de uma operação específica. Como as instruções de como usar os parâmetros estão explicitamente descritos nos documentos WSDL, os atacantes baseiam-se nessa informação para combinarem diferentes opções dos parâmetros para conseguirem obter informação à qual não deveriam ter acesso. Este tipo de ataques podem ser concretizados, por exemplo, através da submissão de caracteres especiais ou conteúdos não esperados e que podem causar situações de DoS ou acesso ilegal a determinados registos. Normalmente, este tipo de ataques são bem sucedidos em situações em que o processo de validação desses parâmetros de entrada não é executado ou não é suficientemente eficaz, não sendo possível detectar essas sequências de parâmetros maliciosos.

SQL Injection

Nos ataques “*SQL Injection*” o atacante envia determinados parâmetros como argumentos das funções que, após processados pelo WS, originarão procedimentos SQL definidos pelo atacante. Esses procedimentos poderão retornar informação não prevista, alterar dados constantes nas bases de dados ou provocar indisponibilidade do próprio WS. Neste tipo de ataques podem ser executadas “*store procedures*” nativas ou comandos SQL inválidos. Exemplo:

```
<SOAP>
  <Header>
</Header>
  <Body>
    <fn:PerformFunction xmlns:fn="">
      <fn:uid>' or 1=1 or uid='</fn:uid>
      <fn:password>abcd</fn:password>
    </fn:PerformFunction>
  </Body>
</SOAP>
```

XQuery Injection

Xquery é uma linguagem baseada em SQL e, como é bastante flexível, permite efectuar *queries* a fontes de dados que contêm informação em XML, incluindo bases de dados e documentos. Os ataques do tipo “*XQuery Injection*” são uma variante do ataque “*SQL Injection*”. Ou seja, através do recurso a dados mal validados, que são passados nos comandos *XQuery*, são executadas rotinas que têm acesso a determinada informação. Estas rotinas podem permitir a enumeração de elementos do ambiente da própria vítima, “injecção” de comandos no servidor local, execução de *queries* a bases de dados ou ficheiros remotos, etc.. O atacante pode passar as expressões *XQuery* embebidas noutros documentos XML ou pode “injectar” os conteúdos *XQuery* como sendo parte da mensagem SOAP, permitindo que o serviço de destino manipule o documento XML incorrectamente.

Ex.: A cadeia de caracteres seguinte permite que o atacante aceda ao “user.xml” para solicitar que o fornecedor do serviço retorne todos os nomes dos utilizadores.

```
doc(users.xml)//user[name='*']
```

XPath Injection

XPath é a linguagem usada para referenciar partes de um documento XML. Esta linguagem pode ser usada directamente pelas aplicações de forma a efectuar *queries* aos documentos XML ou a partes de grandes operações (ex.: aplicar uma transformação XSLT a um documento XML). Nos ataques *XPath Injection*, são passados para o WS parâmetros maliciosos com o objectivo de conseguirem alterar a semântica de um sistema de filtragem XML, sendo possível aceder a informação cujo acesso não foi autorizado. Exemplos:

- Uso de XPath para localizar o utilizador/*password* no XML:

```
//user[nome='Sonia' and pass='minhapass']
```

- Ataque *XPath Injection* simples que retorna todos os utilizadores:

```
//user[nome='Sonia' or 1=1 or ''='' and pass='minhapass']
```

- Ataque *XPath Injection* que retorna todos os utilizadores com ID=1:

```
//user[nome='Sonia' or ID=1 or ''='' and pass='minhapass']/ID
```

XML Injection

Nos ataques “XML Injection”, aproveitando as situações em que o processo de validação do XML não é efectuado da forma correcta, são inseridas *tags* num documento XML. As referidas *tags* XML podem alterar a estrutura do documento XML de tal forma que, os processamentos seguintes são afectados e o comportamento da aplicação passa a ser diferente do esperado.

Exemplo: Através da aplicação, a *tag* XML será guardada na base de dados (“<ID>0</ID>”). Quando os dados forem retornada a partir da base de dados, a *tag* XML já fará parte da cadeia de caracteres XML.

```
<RegistoDoUtilizador>
  <ID>54321</ID>
  <Nome>Sonia</Nome>
  <Email>sonia@comp.pt</Email><ID>0</ID><Email>sonia@comp.pt</Email>
  <Localidade>Porto</Localidade>
</RegistoDoUtilizador>
```

LDAP Injection

LDAP (*Lightweight Directory Access Protocol*) é um protocolo usado para questionar e manipular serviços de directórios X.500. “LDAP Injection” é uma técnica de ataque usada para explorar sistemas que constroem procedimentos LDAP a partir de dados fornecidos pelos clientes. Se o atacante for capaz de alterar os procedimentos LDAP, os processos poderão correr com as permissões do componente que executa o comando (ex.: servidor de base de dados, servidor aplicacional, etc.). Esta alteração pode causar sérios problemas quando, com essas permissões, é possível executar pesquisas, modificar ou apagar qualquer elemento existente na árvore LDAP.

3.9 Ataques ao cliente

Nos ataques ao cliente estão incluídos os que se baseiam na exploração de falhas do lado do cliente ou abusos das relações de confiança estabelecidas entre o cliente e o fornecedor do serviço. Após o estabelecimento destas relações de confiança, o consumidor

espera que o serviço onde está a aceder seja de confiança. O atacante baseia-se nesta “suposta” confiança e consegue atacar o cliente.

3.10 Ataques usando referências externas (*External Reference Attack*)

Neste tipo de ataques, o atacante tenta “desviar-se” das protecções criadas (ex.: validadores do XML) através da inclusão de referências externas que só serão consultadas após a validação do XML mas, antes da aplicação iniciar o seu processamento.

External Entity attack

Uma das vantagens do XML é a capacidade de criar documentos dinamicamente através do uso de apontadores para o URI dos dados actualizados. Em alguns casos, esses URIs apontam para recursos de entidades externas que podem não ser de confiança. Se esses recursos tiverem uma origem maliciosa, podem conduzir o WS a abrir determinados ficheiros ou ligações TCP e provocar situações de DoS, substituir os dados recolhidos por dados maliciosos, etc..

Schema Poisoning

O esquema XML fornece instruções relativas aos formatos e processos dos documentos XML. Essas instruções deverão ser seguidas pelos *parsers* no processo de interpretação desses documentos. Ou seja, o ficheiro do esquema do WS é usado pelos *parsers* XML para perceber a gramática e estrutura do XML e contém instruções de pré-processamento essenciais. Exemplo:

```
<xs:schema xmlns:xs="...">
  (...)
  <xs:element name="nome" type="xs:string"/>
  <xs:element name="endereço" type="xs:string"/>
  <xs:element name="país" type="xs:string"/>
  (...)
</xs:schema>
```

Numa situação de ataque “*schema poisoning*”, o atacante tenta comprometer o esquema XML de tal forma que, no momento da validação, é utilizado o esquema XML alterado. Ou seja, mantendo a localização do esquema XML, este é substituído por um similar mas, modificado. Se esta alteração do esquema não for detectada, o *parser* processa as mensagens SOAP maliciosas que podem levar à execução de comandos no sistema operativo do servidor ou nas bases de dados.

Em seguida está exemplificado um ataque “*schema poisoning*” aplicado ao esquema XML listado anteriormente. Neste exemplo, retirando as restrições relativas aos tipos de dados, o atacante fica livre para enviar dados de tipos diferentes dos esperados, que não irão ser processados correctamente.

```
<xs:schema xmlns:xs="...">
  (...)
  <xs:element name="nome">
  <xs:element name="endereço">
  <xs:element name="país">
  (...)
</xs:schema>
```

Através da inserção de mais intermediários no fluxo de trocas de mensagens XML entre as aplicações, este tipo de ataque pode também ser usado para implementar ataques “*Man in the Middle*” e “*Routing detour*”.

Routing Detours

O *standard* WS-Routing permite definir a forma como o tráfego XML deve ser redireccionado através de um ambiente complexo. Nos ataques “*routing detours*”, o atacante, através da posse do controlo de um dos intermediários constante no WS-Addressing, tenta orientar a mensagem SOAP, enviando-lhe instruções de encaminhamento erradas. Desta forma, os documentos confidenciais são redireccionados para localizações não autorizadas ou encaminhados para localizações inexistentes, podendo causar, por exemplo, situações de DoS.

3.11 Transmissão de mensagens com conteúdo ou anexos maliciosos

O atacante envia anexos SOAP (ex.: imagens, executáveis ou documentos específicos da aplicação) que contêm código ou dados maliciosos, tais como, vírus ou “cavalos de Tróia”. Estes são transmitidos dentro de mensagens XML válidas, podendo infectar os próprios WS e, conseqüentemente, causar alterações nos próprios sistemas que provocarão comportamentos inesperados.

3.12 XML Denial of Service (XDoS), Denial of Service (DoS), Buffer Overflows, ...

Um ataque “*Denial of Service*” (DoS) tem como objectivo indisponibilizar um determinado serviço tornando a sua utilização impossível.

Normalmente, encontramos dois tipos de ataques DoS:

- O ataque DoS leva a que o sistema passe a consumir totalmente os seus recursos (ex.: memória, CPU, espaço em disco, etc.) de tal forma que, o sistema deixa de responder aos pedidos efectuados pelos clientes dos serviços que disponibiliza. Ex.: envio de um número de pedidos maior do que o limite máximo que esse sistema suporta.
- Outra forma de ataque tem como objectivo indisponibilizar as comunicações entre os clientes e o sistema em causa. Ex.: gerar bastante tráfego na rede onde se encontra o sistema, bloquear o acesso de um determinado utilizador ao sistema, etc..

A maior parte dos ataques DoS destinados aos WS não são detectáveis pelas *firewalls* e IDS pois estes equipamentos não têm a capacidade de controlar os pedidos ao nível da transacção/operação base. Como eles não têm “conhecimento” do número médio de pedidos destinados a cada uma das transacções/operações de cada um dos serviços e, o que normalmente conseguem controlar são os endereços IP de origem e destino, não conseguem detectar se determinado serviço está nesse preciso momento a sofrer um ataque DoS.

Ataques “Flooding”

Os ataques “*Flooding*” podem ser criados através da cópia de pedidos válidos efectuados aos WS e reenviando-os para o fornecedor do serviço. Com o reenvio sucessivo dessas mensagens XML/SOAP, o WS vai sendo sobrecarregado. Este tipo de actividade normalmente não é detectado como sendo uma tentativa de intrusão porque o endereço IP de origem é válido, o comportamento do pacote de rede é válido e a mensagem

SOAP/XML está bem formatada. Como este comportamento, ao nível do “negócio” não é legítimo, é considerado um ataque DoS.

Transform Injection

O objectivo final dos ataques “*transform injection*” é provocar uma situação de DoS, consumindo a memória e CPU das máquinas clientes. Estes tipos de ataques podem ocorrer durante a conversão de dados XML noutra forma de apresentação. Como este tipo de transformações é baseado em regras, num ambiente em que seja utilizado XSLT, estão incluídas funções de controlo de fluxos. Tal como estas funções de controlo de fluxos permitem criar tarefas iterativas e repetitivas (ex.: converter dados XML para HTML), também podem ser usadas maliciosamente. Por exemplo, podem permitir criar ciclos infinitos que venham a consumir todos os recursos do servidor.

Também podem ser criadas transformações redundantes que são incluídas (“injectadas”) nas instruções das transformações a serem efectuadas pelo *parser* XML. Por exemplo, neste caso, seguindo estas instruções, o *parser* XML poderá transformar repetidamente os dados originais sem os modificar, consumindo os recursos do sistema onde está implementado o *parser* XML.

“Bombas XML”/”Entity expansion attack”/ ”Entity recursion attack”

No XML é muito fácil definir entidades internas ou externas através da sua declaração nos DTD (*Document Type Definition*): `<!ENTITY constante1 “valor”>`. Após a sua definição, a entidade pode ser referenciada usando a sintaxe “`&NomeDaEntidade`”.

Nos ataques “Bombas XML”, são adicionadas entradas de entidades extra ao documento XML, normalmente através da inclusão da sua definição no DTD, aparecendo antes do elemento *root* desse documento. Frequentemente, essas entidades são numeradas sequencialmente e são definidas várias vezes como variáveis dos programas ou *scripts*.

No exemplo seguinte, o CPU é monopolizado conforme as entidades vão sendo expandidas e, cada uma delas, utiliza mais uma determinada percentagem de memória.

```
<!doctype ROOT [
    <!ENTITY ha “Ha !”>
    <!ENTITY ha2 “&ha; &ha;”>
    <!ENTITY ha3 “&ha2; &ha2;”>
    <!ENTITY ha4 “&ha3; &ha3;”>
    ...
    <!ENTITY ha128 “&ha127; &ha127;”>
]>
<root>&ha128;</root>
```

Ataque “Oversized Payload”

Neste tipo de ataque DoS são enviados documentos XML “grandes” criados para sobrecarregar o *parser* XML e esgotar completamente os recursos do WS disponíveis.

```
<SOAP>
  <Header>
    <wsse: Security>
      1 GB dados binários
      <Signature> (...) </Signature>
    </wsse: Security>
  </Header>
  <Body> (...)
</Body>
</SOAP>
```

Recursive Payload

Como os recursos necessários para o *parser* processar o XML são muito maiores que os necessários para gerar e transmitir XML malicioso, numa situação de ataque, para que o alvo se consiga defender através do tratamento e rejeição desse tráfego, precisa de utilizar bastantes recursos. Como o XML suporta estruturas de documentos complexas, elementos embebidos noutros elementos e recursividade, podem ser criadas situações em que são gerados ciclos infinitos. Um ataque DoS pode ser lançado a partir de uma simples mensagem XML de 2 KB mal formatada.

Ex.: O atacante define uma entidade única (ex.: 100KB) e, dentro de um elemento usado pela aplicação (ex.: um parâmetro de uma cadeia de caracteres), referencia-a muitas vezes (ex.: 50000 vezes).

```
<!DOCTYPE foobar [!Entity x "ABC... [100 KB de dados] ... CBA"]>
<root>
  <hi>&x;&x;... [50000 vezes] ... &x; &x; </hi>
</root>
```

Ataques DoS SYN Flood, LAND, ICMP Flood, UDP Flood,...

Os ataques listados em seguida não são exclusivos dos WS mas, como estes são implementados em sistemas que são também eles vulneráveis, estes comportamentos também deverão ser tidos em conta na implementação deste tipo de soluções:

- Um ataque “*SYN Flood*” consiste em introduzir na rede bastantes pacotes TCP/SYN, utilizando normalmente um endereço de origem inexistente. O sistema de destino, ao receber cada um destes pacotes, considera-os como sendo um pedido de início de ligação, criando essa ligação. O sistema envia de volta um TCP/SYN-ACK, ficando a aguardar o respectivo TCP/ACK que deveria ser enviado pelo sistema de origem inicial. Como o TCP/ACK nunca chega ao sistema de destino, este começa a consumir cada vez mais recursos porque vai mantendo os pedidos de ligação em aberto e, a partir de determinada altura, como já atingiu o número máximo de pedidos de ligação, deixa de conseguir responder aos pedidos de ligação dos “verdadeiros” clientes dos seus serviços.
- O ataque LAND é uma variante do “*SYN Flood*”. Neste caso, o endereço do suposto sistema de origem do pedido de serviço é substituído pelo próprio endereço do sistema que está a ser atacado. Desta forma, o sistema passa a responder a pedidos efectuados por ele próprio, até que fica totalmente indisponível.
- Os ataques “*ICMP Flood*” (também denominado “*Ping Flood*”, “*Ping of Death*” ou “*Smurf Attack*”) resultam do facto de existirem alguns equipamentos de rede mal configurados. Se esses equipamentos permitirem o envio de pacotes para todos os sistemas de uma determinada rede, usando o endereço de *broadcast*, podemos estar vulneráveis a este tipo de ataques (este tipo de redes, quando usadas nestes ataques, são denominadas “*smurf amplifier*”). Reunidas estas condições, o atacante envia um grande número de pacotes *ping* que têm o endereço de origem igual ao do sistema de destino. De forma a evitar o ataque, é usada uma “*Smurf Amplifier Registry*” para filtrar este tráfego existente na rede.
- Nos ataques “*UDP Flood*” (também denominados ataques “*Fraggle*”), o atacante envia para o sistema de destino muitos “*UDP echo*” através do uso do endereço de *broadcast*, usando endereços de origem incorrectos.

Ataque DoS Distribuídos

Uma das maiores preocupações da SOA é o facto dos ataques DoS poderem não ter origem num só sistema. A maior parte das *firewalls* e IDS conseguem bloquear este tipo de ataques quando são originados por um só sistema mas, quando entramos no domínio dos ataques DoS Distribuídos, a sua detecção e bloqueio torna-se mais difícil.

Um ataque DoS Distribuído usa um *botnet* para lançar um ataque DoS a partir de vários sistemas. Se estes sistemas estiverem eles próprios comprometidos, são denominados *zombies*. Num ataque deste tipo, o simples bloqueio de um endereço IP na *firewall* não é solução suficiente para proteger o sistema que está a ser atacado. Um ataque “*pulsing zombie*” é baseado na utilização de *zombies* que provocam a degradação da *performance* do sistema, tornando difícil a tarefa de identificação do grupo de endereços IP dos sistemas que estão a ser efectivamente controlados pelo atacante. Portanto, numa situação destas, é bastante difícil conseguir distinguir, em tempo real, quais são os endereços IP dos sistemas dos legítimos utilizadores desse serviço.

3.13 Sistemas/Serviços distribuídos

Um sistema distribuído consiste num grupo de agentes que colaboram entre si de forma a executarem uma determinada tarefa. Muitas vezes, estes agentes não estão implementados no mesmo “ambiente físico” e, portanto, têm de comunicar através de vários componentes de *hardware/software*, tornando esta troca de mensagens mais lenta e menos segura do que a troca de mensagens efectuada através da invocação directa do código no próprio sistema e utilização de memória partilhada. Portanto, neste tipo de arquitecturas, é necessário acautelar que as aplicações estão preparadas para esta latência adicional e ter em conta a possibilidade de falhas parciais devido a problemas de comunicação ou alteração das mensagens trocadas.

Como uma Arquitectura Orientada a Serviços é um tipo de arquitectura de sistemas distribuídos, também está sujeita aos problemas listados no parágrafo anterior:

- Problemas introduzidos pela latência e confiança/segurança disponibilizada pela camada de transporte das respectivas mensagens.
- Não partilha de memória entre os diversos agentes.
- Problemas introduzidos por falhas parciais de alguns dos sistemas envolvidos.
- O desafio de acessos concorrentes a recursos remotos.
- A fragilidade dos sistemas distribuídos no caso de algum dos participantes introduzir alterações que podem provocar algumas incompatibilidades entre os sistemas.

4 Requisitos de Segurança de uma Arquitectura Orientada a Serviços (SOA)

Tal como foi referido no ponto “3 – Problemas de Segurança da Arquitectura Orientada a Serviços (SOA)”, as ameaças à segurança dos WS podem estar relacionadas com o próprio servidor, com a aplicação ou com toda a infra-estrutura de rede. Portanto, de forma a minimizar este tipo de ameaças, é necessário criar mecanismos que garantam a segurança de toda a arquitectura nas seguintes áreas:

- Gestão de Identidades/Autenticação
- Autorização
- Integridade e confidencialidade dos dados
- Integridade das transacções e comunicações
- Disponibilidade dos serviços
- Não repúdio
- Integridade dos dados relativos a UDDI
- Registo das actividades (Auditoria)
- Cumprimento das políticas de segurança aplicadas aos sistemas distribuídos

4.1 Gestão de Identidades/Autenticação

Os mecanismos de autenticação são necessários para garantir as identidades dos agentes do fornecedor e do consumidor do WS. Em muitos casos, em que existem intermediários entre estes dois agentes, poderá ser necessário delegar, nesses intermediários, o uso da mútua autenticação.

Dada a importância da necessidade de garantir que a autenticação efectuada está correcta e não se trata de uma situação de tentativa do cliente se apresentar com a identidade que não lhe pertence, devem ser utilizados meios/tecnologias bastante robustos.

Outra questão relacionada com a autenticação é a localização das bases dados de utilizadores e respectivas credenciais. Se estiverem no próprio sistema, é necessário garantir a sua protecção contra ataques ao próprio sistema (ex.: encriptando os dados). Se essas bases de dados estiverem remotas, terão de ser assegurados mecanismos de comunicação seguros de forma a garantir a integridade e confidencialidade dos dados relativos a estes processos de validação da autenticação.

4.2 Autorização

De forma a ser possível controlar os acessos a determinados recursos (acesso aos serviços, às funções de negócio e aos próprios dados), é necessário utilizar mecanismos de controlo de autorizações. Só deverá ser possível aceder a determinado recurso após o sistema cliente estar devidamente autenticado e autorizado.

Normalmente, a implementação de processos de autenticação nos WS é efectuada recorrendo a soluções criadas especificamente para cada caso. No entanto, está disponível o *standard* XACML que permite definir as políticas relacionadas com as questões de

autorização de acesso aos recursos, reduzindo o tempo e custos associados à criação de soluções específicas.

4.3 Integridade e confidencialidade dos dados

Os mecanismos responsáveis pelo controlo da integridade das mensagens devem garantir que, durante a transmissão das mesmas, elas não foram alteradas e, no caso de terem sido alteradas, detectar essa modificação. Os mecanismos responsáveis por garantir a confidencialidade dos dados devem assegurar que os dados só são acedidos pelas entidades devidas. De forma a garantir a integridade e confidencialidade dos dados, poderão ser usadas técnicas de encriptação dos dados e assinaturas digitais.

É possível ter em conta que, esta necessidade de integridade e confidencialidade, também se aplica às mensagens que têm de passar por vários intermediários.

4.4 Integridade das transacções e comunicações

Em qualquer infra-estrutura, é aconselhável criar os meios necessários para garantir a integridade das transacções e comunicações. Desta forma, é possível assegurar que os processos de negócio são efectuados correctamente e as operações são executadas na ordem correcta. Para tal, além da integridade dos próprios dados, é necessário garantir que todas as mensagens chegaram ao seu destino e, que cada uma das mensagens, só foi recebida uma vez. Portanto, é indispensável implementar sistemas fiáveis que detectem que determinada mensagem não chegou ao seu destino e a reenviem só o número de vezes necessárias.

4.5 Disponibilidade dos serviços

Um dos requisitos de qualquer arquitectura é a garantia da disponibilidade dos serviços, tanto em situações de actividade normal, como perante tentativas de ataques DoS que explorem vulnerabilidades específicas dos WS. Ou seja, é necessário garantir QoS (*Quality of Service*) e QoP (*Quality of Protection*) perante situações de ataques dos WS.

QoS é importante na definição do nível de *performance* esperado para um determinado serviço. O aumento de *performance* do serviço pode ser obtido, por exemplo, através da atribuição de maior prioridade do tráfego relacionado com determinado serviço e garantindo a entrega de todas as mensagens na ordem correcta. Também é importante definir os valores máximos permitidos para as taxas de falhas, latência média, etc.. Para garantir certos valores de QoS, antes de determinado serviço entrar em produção, deverão ser efectuados testes de carga simulando o ambiente real.

Outro factor importante é a definição de formas de recuperação dos WS após sofrer, por exemplo, um ataque DoS. É necessário definir técnicas de replicação de dados e serviços de forma a assegurar a continuidade das operações no caso de ocorrer alguma falha.

Para evitar falhas de serviço é extremamente necessário implementar mecanismos de gestão e monitorização das soluções. Além disso, também é necessário recorrer a técnicas de desenvolvimento dos programas que garantam a segurança dos mesmos, assim como mecanismos de detecção das falhas relacionadas com a sua especificação/implementação que poderão ser exploradas e permitir que os ataques sejam efectuados com sucesso.

4.6 Não repúdio

“Não repúdio” é um serviço de segurança que protege cada uma das partes envolvidas nas transacções perante os casos em que uma das partes nega que tenha enviado/recebido as mensagens. Este serviço permite provar qual a origem da mensagem e a sua entrega. Ou seja, é possível verificar que tanto a entidade que envia a mensagem como a que a recebe são efectivamente as partes que supostamente as enviaram e receberam, respectivamente. Através da garantia de “não repúdio” da origem é possível provar que os dados foram enviados enquanto que através da garantia do “não repúdio” do destino é provado que os dados foram recebidos. Desta forma, nenhuma das entidades envolvidas pode, mais tarde, negar que tenha estado envolvida no processamento da informação.

Uma das formas de implementar este serviço é através do uso de mecanismos criptográficos que consigam comprovar a origem, submissão, transporte e entrega dos dados.

4.7 Integridade dos dados relativos a UDDI

Os registos dos UDDI fornecem detalhes relativos às funcionalidades disponibilizadas pelo serviço e como poderá ser acedido. Os WS utilizam os registos UDDI para descobrir e, dinamicamente, utilizar determinado serviço. Se, de alguma forma, esses registos sofrerem um ataque e forem comprometidos, é possível que determinado consumidor de um WS use um serviço de um fornecedor malicioso. Para evitar este problema, é importante que as entradas UDDI sejam assinadas digitalmente de forma a garantirem a identidade do fornecedor do serviço (esta funcionalidade de assinatura do UDDI foi introduzida na v3.0.2 – *standard* OASIS de 2005). Além destas medidas, também se devem controlar as permissões de escrita de entradas UDDI relativas a determinado serviço (ex.: não permitir que os consumidores do WS alterem esses registos).

4.8 Registo das actividades (Auditoria)

É necessário ter algum meio de registo das actividades pois, no caso de se verificar alguma acção anómala, esses registos podem ser utilizados para verificar a sua origem e quais as entidades envolvidas.

4.9 Cumprimento das políticas de segurança aplicadas aos sistemas distribuídos

No tipo de arquitectura SOA, existem três conceitos fundamentais relativos à segurança:

- Os recursos devem ser implementados de uma forma considerada segura.
- Devem existir mecanismos que controlam se os recursos são seguros.
- Existência de políticas aplicadas aos recursos.

Podemos dividir as políticas em dois grupos:

- Políticas de permissões – Políticas relacionadas com permissões em que são definidas as acções e acessos que são autorizados.
- Políticas de obrigações – Políticas em que, para uma determinada entidade, são definidas as acções e os estados esperados.

Os mecanismos que controlam se as políticas são ou não aplicadas aos recursos podem também ser de dois tipos:

- Controlador de permissões – verifica se uma determinada acção ou pedido são permitidos.
- Controlador de auditorias – após determinada acção, verifica se determinada obrigação/compromisso não foi cumprido.

Numa Arquitectura Orientada a Serviços é necessário garantir que as políticas de segurança estão definidas e correctamente aplicadas a todos os componentes da arquitectura. Como exemplo, podemos considerar as políticas de confiança nos processos distribuídos de autenticação/autorização. Como neste tipo de arquitecturas o consumidor do serviço precisa de confiar no ambiente do fornecedor desse serviço, e vice-versa, devem ser definidas políticas de confiança que deverão ser aplicadas tanto ao consumidor e fornecedor dos serviços como também aos respectivos intermediários. Ou seja, as políticas de confiança têm de ser aplicadas a todas as partes envolvidas e, até mesmo, a todos os domínios envolvidos.

É recomendado que a gestão das políticas a aplicar aos WS seja efectuada a partir de um ponto único de controlo. Desta forma, é possível reduzir os riscos da existência de erros e omissões que se poderão manifestar quando se configura a segurança ao nível de cada um dos servidores onde estão implementados os serviços.

5 Standards WS disponíveis para implementar segurança numa Arquitectura SOA

O modelo de segurança dos *Web Services* introduziu um conjunto de *standards* de forma a criar uma “camada” de segurança. Alguns desses *standards* dependem uns dos outros, exemplo: WS-Policy usa WS-PolicyAssertions, WS-Security usa XML-Signature e XML-Encryption, WS-Policy é usado em conjunto com WS-Security, etc.. Muitos deles começaram como sendo iniciativas proprietárias de algumas indústrias e, ao longo do tempo, foram sendo transferidos para *standards* da OASIS, W3C ou IETF (ex.: WS-Security, WS-Trust, WS-Policy). Os grupos de trabalho do IETF têm-se concentrado na criação de *standards* relacionados com a segurança a nível do transporte das mensagens enquanto que, os da OASIS e W3C têm-se focado mais na segurança a nível das aplicações. Sempre que seja necessário implementar alguma funcionalidade que já esteja assegurada por um *standard* reconhecido por estas entidades, é preferível usá-lo do que estar a desenvolver, na própria empresa, mecanismos que ofereçam essas funcionalidades. Desta forma, além do grau de compatibilidade com os WS das outras empresas externas ser maior, na maior parte dos casos, foram efectuados testes exaustivos para garantir o cumprimento dos objectivos para os quais foram criados. Ou seja, o seu uso é imprescindível pois são uma base essencial para garantir a interoperabilidade entre a grande variedade de produtos usados nos ambientes de cada uma das empresas que, geralmente, são bastante heterogéneos. As implementações SOA baseadas em *standards* permitem acelerar a fase de desenvolvimento, simplificam a integração e permitem reduzir os custos administrativos ao longo do tempo.

A Figura 5.1 tenta agrupar alguns dos *standards* de segurança aplicados aos WS de forma a ser mais fácil perceber a que nível é que são aplicados.

Os *standards* incluídos nas camadas de rede, transporte e segurança do XML são usados para tornar as mensagens seguras enquanto elas são transmitidas na rede. Cada um dos *standards* de segurança IPSec, SSL/TLS, XML-Encryption e XML-Signature são aplicados, a um nível diferente, às mensagens SOAP.

Por cima da camada de segurança do XML existem vários *standards* disponíveis que permitem implementar: segurança das mensagens, confiança na entrega das mesmas, controlo de acessos aos WS, definição de políticas, gestão de segurança e de identidades, etc.. Os *standards* WS-Security e WS-SecureConversation, incluídos no grupo de segurança das mensagens, definem como usar XML-Signature, XML-Encryption e as credenciais a serem utilizadas para tornarem SOAP seguro ao nível das mensagens. Os *standards* responsáveis por assegurar os aspectos relacionados com a “confiança” na entrega das mensagens SOAP (WS-Reliability, WS-ReliableMessaging, WS-Addressing, etc.) definem os protocolos e cuidados necessários para assegurar que as mensagens serão recebidas correctamente. Nos responsáveis pelo controlo de acessos, podemos incluir XACML (permite definir as políticas de acesso aos sistemas) e SAML (definição de declarações em qualquer ambiente). A nível das políticas, foi criado, por exemplo, o *standard* WS-Policy que define a gramática utilizada para comunicar os requisitos das políticas de um determinado WS.

No grupo de gestão de segurança estão incluídas as definições dos outros WS que gerem as credenciais, tais como certificados PKI aplicados a SOA. Os *standards* responsáveis pela gestão das identidades baseiam-se nos *standards* de controlo de acessos, nos relacionados com políticas e nos *standards* SOAP para fornecerem serviços para a distribuição e gestão de identidades e credenciais numa Arquitectura Orientada a Serviços.

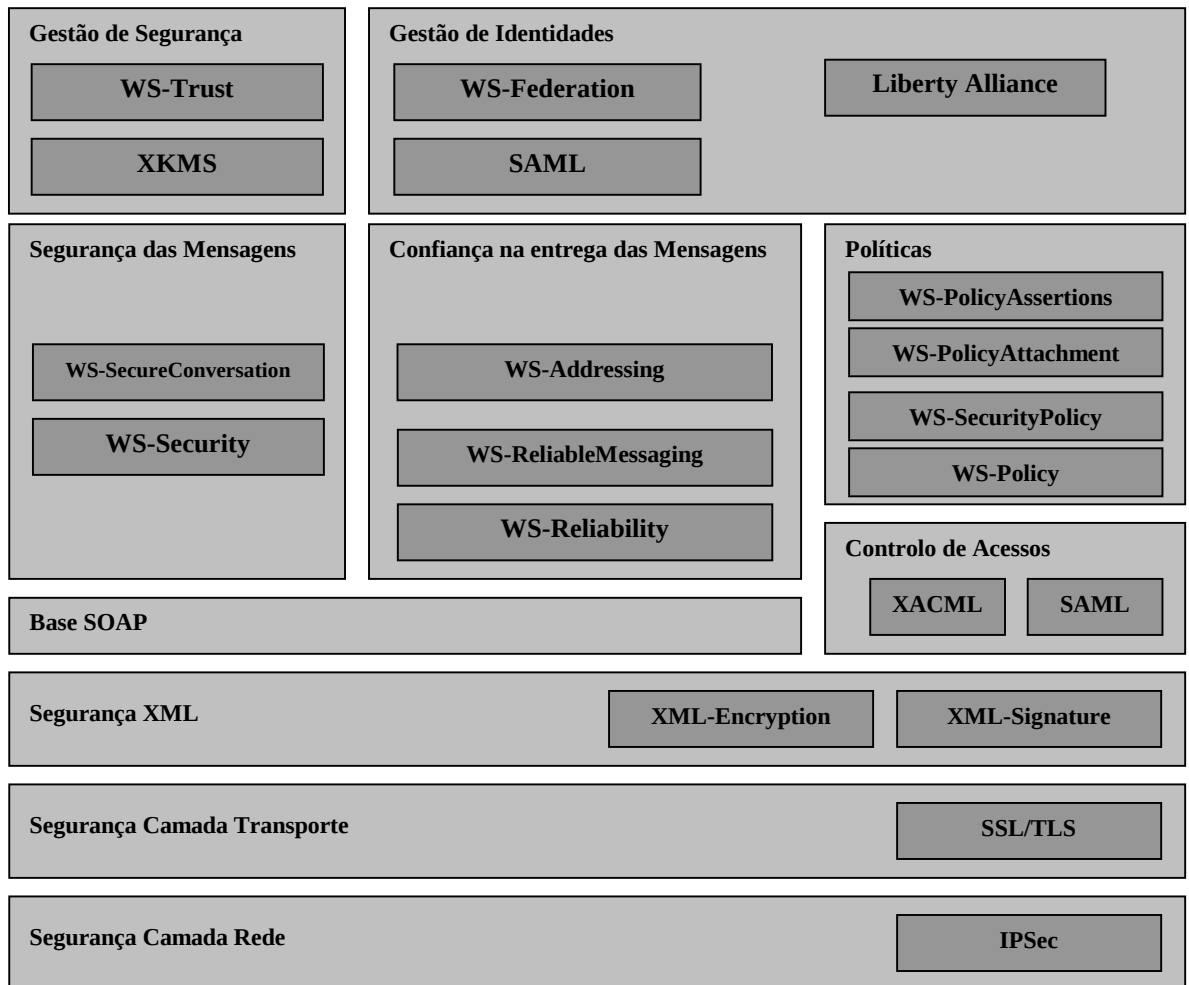


Figura 5.1: Alguns dos *standards* de segurança aplicados aos *Web Services*
(baseado no conceito desenvolvido pelo *National Institute of Standards and Technology* [23])

No “ANEXO A: *Standards* de Segurança” estão listados e descritos alguns dos *standards* de segurança que podem ser usados na implementação de arquitecturas SOA baseadas em *Web Services*.

6 Algumas tecnologias disponíveis para a implementação de medidas de segurança numa Arquitectura SOA

6.1 XML Security Gateway (XSG)

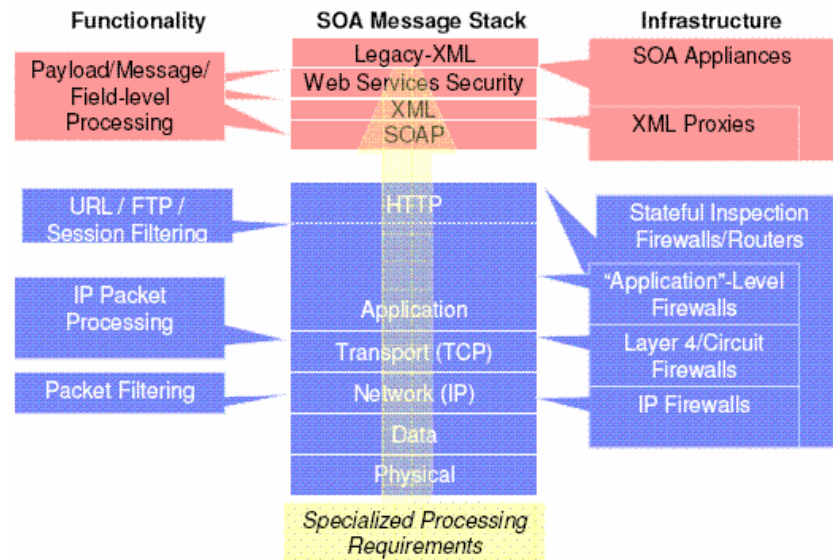


Figura 6.1: Mensagens SOAP e equipamentos da infra-estrutura de segurança

(Fonte: IBM [8])

As primeiras *appliances* XML foram disponibilizadas em 1999, pela DataPower, e, em 2001, pela Forum Systems. Nessa altura, existiam dois tipos principais de equipamentos: uns focados em otimizar as tarefas relacionadas com grandes volumes de transformações e os que ofereciam soluções com aceleração do processamento XML e segurança. Como, entretanto, passou a ser necessário trocar mensagens XML entre sistemas disponibilizados através das redes públicas, em 2003, estes dois tipos de equipamentos começaram a convergir, começando também a incluir novas funcionalidades que tentam garantir a implementação de algumas medidas de segurança dos sistemas: encriptação, assinaturas, detecção/protecção de ataques, etc..

Actualmente, tendo em conta as tecnologias disponíveis, uma forma de proteger os WS é introduzir, entre o consumidor e o fornecedor dos WS, uma *XML Security Gateway*. Esta pode actuar como um *proxy* de WS e, simultaneamente, acrescentar algumas funcionalidades de segurança à infra-estrutura (Figura 6.1). Assim, algumas das funcionalidades de segurança implementadas no próprio código do WS, que podem estar a degradar a *performance* da própria aplicação, podem passar para a XSG. No entanto, é necessário ter sempre em atenção que só a XSG não resolve todos os problemas de segurança pois, caso esta seja comprometida, todos os WS internos ficam vulneráveis a ataques. Portanto, tanto os WS para consumo externo como interno, devem ser desenhados, desenvolvidos e configurados tendo em conta as questões de segurança básicas. Ou seja, devem ser ponderadas quais as funcionalidades de segurança que se devem manter no código do próprio WS, quais as que devem ser implementadas no código e na *gateway* e quais as que devem ser só da responsabilidade da *gateway*.

As XSG podem ser baseadas em *hardware* ou *software*. Uma solução baseada em *hardware* normalmente é mais cara (o próprio *hardware* é otimizado para determinadas funções) mas, apresenta muitas vantagens:

- *Performance* – Quando a *performance* dos WS é um dos principais requisitos da Arquitectura Orientada a Serviços, normalmente opta-se por soluções baseadas em *hardware* pois permitem as funcionalidades de “aceleração” dos pacotes XML (*caching*, compressão, etc.) e dos processos relacionados com encriptação/descodificação e assinatura dos dados, verificação de assinaturas, análise/filtragem do XML, transformação das mensagens XML/protocolos e terminação SSL. Este tipo de *hardware* já está otimizado para não introduzir atrasos na entrega das mensagens e, por vezes, são utilizadas técnicas de processamento das mesmas que permitem que a sua entrega, validação, encriptação, etc., sejam efectuadas de uma forma ainda mais rápida. Devido às soluções baseadas em *software* estarem dependentes do sistema operativo, das aplicações que coexistem no mesmo servidor e do próprio *hardware*, não é possível garantir valores de *performance* muito elevados.

Como, o processamento das mensagens XML é bastante complexo, nas arquitecturas SOA o número de mensagens trocadas é maior do que noutras arquitecturas e as mensagens XML são três a dez vezes maiores do que as mensagens binárias equivalentes, a *performance* dos sistemas envolvidos neste processamento é bastante importante. Análises recentes indicam que aproximadamente 80% da capacidade de processamento dos servidores aplicativos é utilizada no processamento de tarefas de validação do XML, transformação dos dados e terminação de túneis SSL.

- Escalabilidade – Maior escalabilidade pois, com a implementação de uma XML *gateway* de alta capacidade, o número de servidores aplicativos necessários pode ser significativamente reduzido. A *gateway* XML pode suportar aumentos significativos de carga nas transacções dos WS sem ser necessário adicionar novos servidores aplicativos (algumas das tarefas de processamento dos WS são efectuadas pela *gateway*, que tem *hardware* otimizado para essas funções). Além disso, normalmente o processo de substituição ou adição de uma *gateway* XML é relativamente simples.
- Gestão – Através da canalização de todo o tráfego destinado aos WS através de um número pequeno de *gateways* XML de alta capacidade, o número de pontos onde são forçadas as políticas de segurança pode ser reduzido ao mínimo. Desta forma, através de uma gestão centralizada, é possível simplificar as configurações de segurança e o próprio processo de gestão das alterações.
- Simplicidade – As actuais *gateways* XML baseadas em *hardware* disponíveis no mercado, têm sistemas operativos proprietários ou, caso utilizem outros sistemas operativos, estes já estão quase totalmente configurados. Assim, a configuração da própria *gateway* XML é simplificada pois, a nível do sistema operativo, só é necessário configurar alguns parâmetros base (nome, endereços IP, identificação dos servidores DNS, NTP e SNMP, etc.).
- Segurança – Essencialmente, a *gateway* XML virtualiza os WS, actuando como intermediária entre o WS e o seu consumidor. Passando algumas das funções

relacionadas com a segurança do código das próprias aplicações para a *gateway*, é considerada a melhor prática pois, permite melhorar a segurança de toda a arquitectura. Tal como já acontece com as *firewall* de filtragem IP, uma *gateway* XML é uma plataforma já criada tendo em conta os princípios básicos de segurança. Ou seja, tanto o seu *hardware* como o *software* foram alvo de testes de segurança exaustivos. Além disso, ela está preparada para proteger a arquitectura das principais vulnerabilidades dos servidores aplicativos baseados em XML.

- Disponibilidade – As *gateways* XML baseadas em *hardware* estão optimizadas para, com uma alta *performance*, conseguirem detectar e proteger os WS dos ataques DoS.

Num estudo recente da Gartner é dito que, com uma combinação apropriada das tecnologias de rede, servidores, *software* e equipamentos de infra-estrutura de rede e segurança (neste grupo estão incluídas as *gateways* XML), é possível:

- Melhorar o tempo de resposta das aplicações desde 3 a 10 vezes.
- Reduzir a carga de CPU dos servidores *Web* em 80% e o número de servidores para metade.
- Melhorar a eficiência dos servidores aplicativos e diminuir o número de licenças das aplicações necessárias.
- Reduzir os requisitos de largura de banda da WAN em pelo menos 80%.
- Reduzir os requisitos de energia e refrigeração em cerca de 10% a 15%.

As soluções baseadas em *software* podem ser instaladas num servidor central ou como agentes distribuídos pelos servidores onde correm os WS. Normalmente, neste segundo cenário, os agentes são geridos centralmente. Embora, no início, esta solução pareça ser mais barata, com o aumento da complexidade da arquitectura devido ao aumento do número dos WS, os custos que inicialmente eram considerados reduzidos tornam-se maiores. Este aumento de custos está associado à necessidade de manter e gerir um maior número de agentes e, no caso do cenário do servidor central, além da configuração/manutenção do *software*, também é necessário manter o sistema operativo actualizado e seguro. Portanto, nessa fase de evolução da arquitectura, tendo em conta as vantagens das soluções por *hardware*, deve ser ponderada a possibilidade de se começar a implementar soluções desse tipo. Exemplos de fabricantes de XSG:

- Baseadas em *hardware*: IBM (DataPower), Cisco (ACE XML Gateway), Forum Systems, Vordel, F5, Layer 7 Technologies, etc..
- Baseadas em *software*: Intel, SOA Software, Microsoft (ISA – publicação de WS e filtros ISAPI), etc..

A maior parte das *gateways* XML existentes actualmente no mercado integram funções de gestão dos acessos aos *Web Services* XML (*XML Web Services Trust Enablement*) e protecção a ameaças XML (*Firewall* XML, IDS e IPS). Normalmente, são implementadas como *proxy* devido a ser necessário que todas as mensagens com destino ou origem nos WS sejam analisadas antes de serem passadas para a aplicação ou cliente (*proxy* XML).

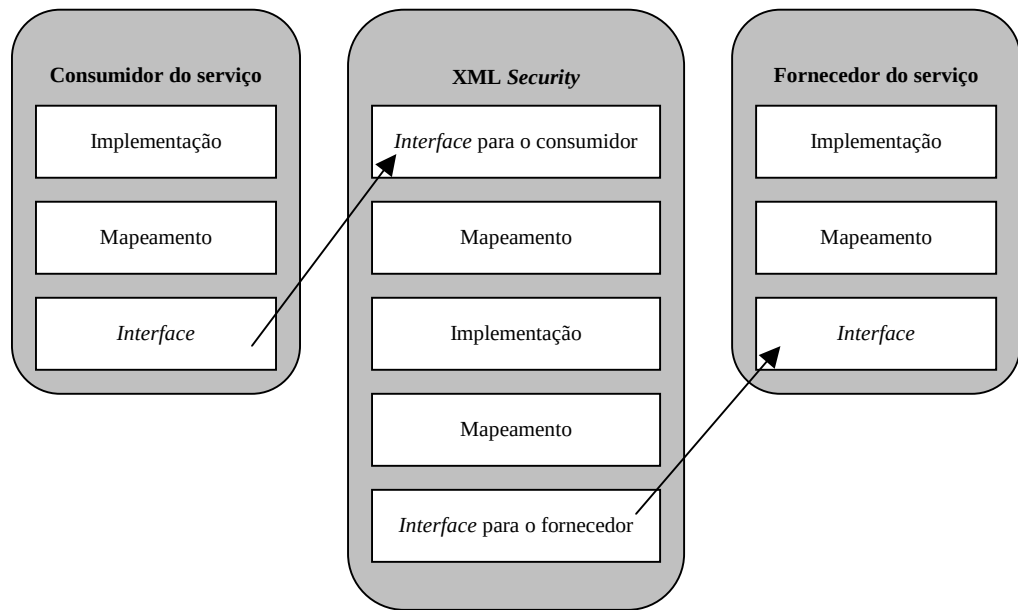


Figura 6.2: XML Security Gateway – interacção com os WS

Podemos considerar que as XSG são *proxies* aplicacionais pois efectuem funções de “intermediário” entre o cliente e o servidor (o termo *proxy* significa “to act on behalf of another”; Figura 6.2). Quando os clientes efectuem um pedido, é este “intermediário” que, baseando-se num conjunto de regras referentes à aplicação em causa, determina qual/quais o(s) servidor(es) que irá/irão responder a esse pedido e contacta-os “em nome” do cliente. O próprio *proxy* também pode alterar o conteúdo dos pedidos/respostas de acordo com as políticas preestabelecidas implementando, dessa forma, algumas medidas de segurança ou algumas funções que, normalmente, implicariam alterações nas próprias aplicações.

Em seguida, estão listadas e descritas as principais funcionalidades das XSG:

- “Mascarar” os recursos internos recorrendo à virtualização/transformação dos serviços. – Através das funcionalidades de *Reverse Proxy* Aplicacional, é possível não publicar directamente os servidores às entidades externas. Ou seja, desta forma, como os servidores não estão expostos directamente, é possível:
 - Não disponibilizar nos cabeçalhos das mensagens as informações relativas ao sistema operativo, versões de aplicações, assinaturas, etc..
 - Não indicar a localização real do sistema onde está implementado o WS. Assim, é possível mover os serviços para plataformas diferentes sem ser necessário alterar o seu endereço IP público e, conseqüentemente, sem que o próprio cliente se aperceba desta mudança.
- Simultaneamente, publicar várias versões do mesmo serviço através do recurso a funcionalidades relacionadas com a gestão das versões dos WS. Esta publicação poderá ser efectuada de uma forma transparente para o utilizador sem que ele se aperceba das actualizações efectuadas aos serviços disponibilizados.
- As mensagens de erro HTTP podem ser ocultadas/alteradas, não permitindo que o utilizador final se aperceba qual a causa do erro gerado pelo serviço.

- Efectuar uma validação prévia das mensagens retornadas de forma a verificar se estas incluem partes do código da aplicação, comentários privados, etc.. Se for necessário, estes dados podem ser removidos ou alterados.
- Recorrendo à XSLT (*eXtensible Stylesheet Language Transformation*), transformar os documentos XML de forma a fazer corresponder o conteúdo do seu esquema noutro ou efectuar as correspondências necessárias para outras manipulações do XML.

XSLT é uma linguagem usada na conversão das estruturas dos documentos XML em outros formatos de documentos de texto, incluindo XML. A XSLT usa XPath para aceder ao documento original e efectuar os “cálculos” necessários. Originalmente, XSLT foi criada para converter documentos XML em formato HTML mas, com o tempo, foi adaptada e usada também noutras situações.

Frequentemente, XSLT também é usada na implementação de algumas aplicações relacionadas com a lógica de negócio. Por exemplo, muitas vezes, a partir de dados formatados em XML, podem ser escritos programas XSLT que pesquisam, manipulam, ordenam, extraem ou executam cálculos baseados nos dados existentes no documento XML.

Exemplo (Figura 6.3):

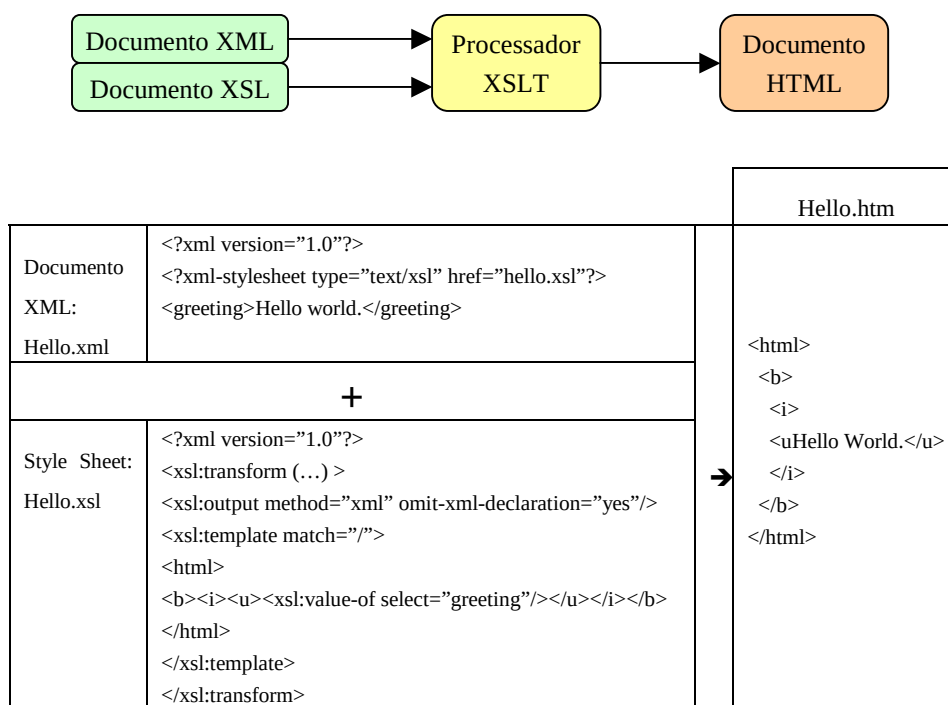


Figura 6.3: Transformação XSLT

- Transformação de protocolos. – A XSG pode efectuar tarefas de conversão de protocolos de acordo com as expectativas do consumidor e fornecedor do WS. Como HTTP e SSL não são os únicos protocolos usados para enviar tráfego XML, o pedido pode ser efectuado num protocolo e o serviço ser invocado em qualquer outro. Ex.: passar uma mensagem SOAP por HTTP para uma fila de mensagens JMS (*Java Messaging Service*), converter AJAX para SOAP, converter as credenciais de autenticação de HTTP Básica para SAML, etc..

- *Router XML*. – Tomadas de decisão relativamente ao encaminhamento, ou não, dos pedidos XML em função do seu conteúdo (cabeçalho ou corpo da mensagem) e contexto (credenciais do utilizador, cabeçalhos ao nível da camada de “transporte”, etc.). Esta funcionalidade pode ser bastante útil quando existem grupos de clientes com requisitos do nível de resposta diferentes. Desta forma, tendo em conta, por exemplo, a identidade e/ou “classe de serviço” do cliente, parâmetros da mensagem, disponibilidade e/ou *performance* do serviço, etc., os pedidos são encaminhados para determinado grupo de servidores. É possível que dois clientes tentem aceder ao mesmo endereço IP público e que sejam encaminhados para grupos de servidores diferentes que tenham níveis de *performance* diferentes, que tenham acesso a dados mais, ou menos, confidenciais, etc..
- Terminação SSL e outras funções criptográficas
 - Terminação de sessões SSL/TLS.
 - Gestão e armazenamento das chaves de encriptação. – As XSG podem ser as responsáveis pela gestão de todas as chaves e *tokens* usados para garantir a integridade e confidencialidade dos serviços expostos através da *gateway*. Ou seja, os clientes que invocam qualquer um dos serviços publicados por esta XSG só precisam de ser configurados para confiarem nas chaves ou *tokens* relativos a esta *gateway*, deixando de ser necessário confiarem nas chaves e *tokens* específicos de cada um dos serviços. Por outro lado, com esta funcionalidade, só as *gateways* é que precisam de ser configuradas para confiar nas chaves e *tokens* de cada um dos clientes.
 - Troca de chaves.
 - Encriptação/Descodificação XML. – XML-Encryption (inclui voltar a encriptar as mensagens com destino aos servidores WS).
- Assinatura de XML/Validação de assinaturas XML. – XML-Signature.
- Validação do formato e da semântica XML, detecção de ataques XML. – Neste conjunto de funcionalidades está incluída a capacidade da própria infra-estrutura de rede suportar tráfego XML e ter a possibilidade de se proteger contra ameaças XML (conhecidas ou desconhecidas) que possam ter um impacto adverso na disponibilidade e integridade das aplicações baseadas em WS – filtragem aplicacional.

Como o XML é utilizado na troca de dados, é extremamente importante a inspecção, validação e protecção das transacções XML (validação dos esquemas XML e do conteúdo, filtragem do XML, detecção de “*buffer overrun*”, protecção contra XDoS, etc.). Assegurar que os argumentos de entrada e os resultados gerados pelas aplicações baseadas em XML estão bem formatados e só contêm os elementos necessários é extremamente importante para o correcto funcionamento dos sistemas/aplicações. Esta validação pode ser efectuada através do recurso a *templates* específicos de cada uma das aplicações ou através da comparação com os DTDs (*Document Type Definitions*) ou WSDL. Centralizando o controlo e validação dos documentos DTD e WSDL, reduz o risco relativo à possibilidade de

comprometimento do sistema onde está instalado o WS e consequente modificação destes documentos de forma a passarem a apontar para sites “maliciosos”.

- Gestão centralizada das políticas de segurança e configurações. – Tendo em conta que todo o tráfego relacionado com os WS passa pela *gateway*, este é o local ideal para forçar a aplicação das políticas de segurança. Desta forma, é possível garantir, com maior rigor, a aplicação consistente das políticas (partilha da mesma política aplicada a vários WS, local único de gestão das versões e cópias de segurança dessas políticas, etc.).

A política é um conjunto de regras e comportamentos que determinam como é que a XSG deve processar o tráfego. Ou seja, onde estão definidas as regras de comportamento da *gateway* perante possíveis ataques, validações a efectuar relativamente ao conteúdo das mensagens, gestão de acesso aos serviços, transformações a efectuar ao conteúdo das mensagens, determinadas configurações específicas de cada serviço, etc..

- Gestão de acessos aos *Web Services*. – Este conjunto de funcionalidades inclui, principalmente, a capacidade de controlo de acessos: autenticação, autorização e responsabilização. Desta forma, é possível, a partir de um ponto centralizado, controlar (limitar, permitir e registar) o acesso a determinada informação com base em credenciais do utilizador, certificados SSL, SAML, etc.. Muitas vezes, quando não é possível implementar estas funções através do recurso aos *standards* de segurança existentes (ex.: a organização não ter acesso ao código fonte e ser necessário utilizar um *token* não suportado pelo WS-Trust), pode ser necessário que a XSG suporte a integração com ferramentas de IAM (*Identity and Access Management*).
- Registo de actividades. – Como todos os pedidos efectuados aos WS devem passar pela XSG, este é um bom ponto para efectuar o registo de todas as actividades.
- Monitorização. – Como todo o tráfego dos WS deve passar pela *gateway*, este é um óptimo ponto da infra-estrutura para serem obtidos os valores relativos à medição da *performance* dos WS, dados relativos a erros operacionais, estatísticas da percentagem de acessos efectuados com sucesso/insucesso, etc.. A *gateway* também é um elemento da arquitectura a partir do qual podem ser gerados os alertas relativos à detecção de ataques ou comportamentos anómalos dos WS.
- Publicação WSDL. – Normalmente, os serviços são importados na *gateway* e disponibilizados para os clientes. Desta forma, a *gateway* funciona como servidor *proxy*, podendo mesmo gerar, a partir de um ficheiro WSDL, um novo ficheiro WSDL que poderá ser disponibilizado para o exterior. Desta forma, os consumidores desse serviço usarão a *gateway* como o *end-point* do serviço. Sempre que for necessário, estes serviços poderão ser publicados em directórios UDDI.

Também poderão existir situações em que, na própria *gateway*, são importados serviços externos. Neste caso, a *gateway* poderá disponibilizá-los aos sistemas internos, servindo também de *proxy*.

- Compressão das mensagens ou anexos. – Por exemplo, os *standards* “SOAP with Attachments” e MTOM (*Message Transmission Optimization Mechanism*)

permitem implementar uma solução mais eficiente e com compressão das mensagens SOAP compostas por conteúdos binários.

- Interoperabilidade. – Como as próprias tecnologias e os *standards* de segurança relativos aos WS estão constantemente a ser alterados/criados, a XSG é a peça da arquitectura ideal onde devem ser efectuadas as actualizações necessárias, assim como a tradução entre os múltiplos transportes.

Além disso, a comunicação entre os fornecedores e consumidores do WS não é baseada em acordos de *runtime* (ex.: J2EE) mas sim, em acordos relativos aos contratos dos serviços, *standards* de troca de mensagens e esquemas. Os serviços de autenticação, autorização e auditoria ao nível do serviço e mensagens não estão restritos a uma única organização pois têm de suportar ambientes em que os serviços têm de atravessar as “fronteiras da organização”. Ou seja, são disponibilizados para outras empresas que, possivelmente, utilizam outras tecnologias.

Resumindo, as principais operações, a nível do processamento das mensagens, que deverão ser executadas nos equipamentos intermediários responsáveis por garantir a segurança da infra-estrutura, deverão ser as seguintes:

- Detectar alguns tipos de ataques, verificar formatação do XML;
- Se aplicável, terminar túneis SSL;
- Remover e validar a assinatura do cliente;
- Decodificar a mensagem XML;
- Validar o esquema;
- Efectuar outras validações/filtragens XML com base no conteúdo das mensagens;
- Extrair as credenciais dos utilizadores;
- Autenticar o utilizador;
- Verificar as autorizações de acesso ao WS para o utilizador em questão;
- Registar a informação anterior;
- Se os passos anteriores forem executados com sucesso,
 - Se necessário, adicionar o contexto de segurança, assinar, encriptar alguns campos, etc..
 - Efectuar as transformações necessárias.
 - Enviar a mensagem para o WS respectivo.
- Se não, retornar um erro e registar esta tentativa de acesso sem sucesso.

Conforme a organização em causa, a integração de ambas as funções de *proxy* e *firewall* aplicacional no mesmo equipamento pode ser considerada uma vantagem ou uma desvantagem. Em algumas organizações, o facto de estarem integradas todas estas funcionalidades no mesmo equipamento dificulta a implementação e operação das tarefas relacionadas com questões de rede, segurança e administração dos sistemas. Por outro lado, com a integração de todas estas funções poderão existir ganhos em termos do custo de aquisição/manutenção das *gateways*. Além do factor monetário, como actualmente

algumas das equipas dos Sistemas de Informação não têm pessoas com conhecimentos simultaneamente nas áreas de infra-estrutura, segurança e na área aplicacional/desenvolvimento, se estas *gateways* integrarem todas essas funções, é necessário delimitar bem quais as competências de cada equipa na configuração destes dispositivos. Portanto, é necessária uma boa coordenação entre todas as equipas envolvidas de forma a garantir que configurações efectuadas por uma das equipas “anulem” as configurações das outras.

6.2 Implementação de segurança a nível da configuração dos próprios WS

A segurança é uma das funcionalidades fundamentais de qualquer plataforma de *software*. Em seguida, serão descritas algumas formas de implementar segurança ao nível do próprio WS. Como exemplo, será considerada a sua implementação numa solução baseada em WCF (*Windows Communication Foundation*).

Bindings

No WCF, os *bindings* permitem definir quais os *standards* WS-* que são suportados, referindo-se a dois aspectos bastante importantes ao nível da mensagem: a codificação (forma como a mensagem é “serializada”) e o transporte (mecanismo usado para levar a mensagem do emissor para o receptor).

Na Tabela 6-1 estão representados os modos de transporte das mensagens que podem ser usados para cada um dos *bindings* WCF. As opções possíveis são as seguintes:

- HTTP(S) – Este é o único modo de transporte que suporta interoperabilidade, permitindo a integração com plataformas diferentes.
- NET.TCP – Baseado em codificação binária para otimizar as comunicações. Como só pode ser usado em ambientes Windows, não permite a integração com outros tipos de plataformas.
- MSMQ (*Microsoft Message Queuing*) – Usado em cenários “assíncronos” (*fire-and-forget*). Por exemplo, pode ser usado em situações em que existem processos longos e não é necessário esperar pela resposta obtida no final da sua execução. No entanto, WCF tem mecanismos para, nestes cenários, garantir a entrega da mensagem assíncrona (*one-way operation*).
- *Named Pipes* – O transporte *Named Pipes* do WCF só suporta comunicações locais, usando o esquema “net.pipes”.

<i>Binding</i>	HTTP	HTTPS	NET.TCP	MSMQ	<i>Named Pipes</i>
BasicHttpBinding	Sim	Sim	Não	Não	Não
WsHttpBinding	Sim	Sim	Sim	Não	Não
WsDualHttpBinding	Sim	Sim	Não	Não	Não
WsFederationHttpBinding	Sim	Sim	Não	Não	Não
NetTcpBinding	Não	Não	Sim	Não	Não
NetNamedPipeBinding	Não	Não	Não	Não	Sim
NetMsmqBinding	Não	Não	Não	Sim	Não
NetPeerTcpBinding	Não	Não	Sim	Não	Não
MsmqIntegrationBinding	Não	Não	Não	Sim	Não

Tabela 6-1 – Correspondência dos *bindings* predefinidos do WCF às opções de transporte

<i>Binding</i>	Configuração	Segurança (valor por defeito)	Sessão por defeito	Transacções	Comunicação em ambos os sentidos (<i>duplex</i>)
BasicHttpBinding	Basic Profile 1.1	Nenhuma	Não	-	-
WsHttpBinding	Standards WS-* (WS-Security, WS-Reliable Messaging, WS-Addressing)	Mensagem	Opcional	Sim	-
WsDualHttpBinding	Standards WS-* (WS-Security, WS-Reliable Messaging)	Mensagem	Sim	Sim	Sim
WsFederationHttpBinding	WS-Federation	Mensagem	Sim	Sim	Não
NetTcpBinding	.NET	Transporte	Opcional	Sim	Sim
NetNamedPipeBinding	.NET	Transporte	Sim	Sim	Sim
NetMsmqBinding	.NET	Transporte	Sim	Sim	Não
NetPeerTcpBinding	Peer	Transporte	-	-	Sim
MsmqIntegrationBinding	MSMQ	Transporte	Sim	Sim	-

Tabela 6-2 – Valores para cada um dos *bindings* WCF predefinidos

Na Tabela 6-2 [21] estão listados os valores para cada um dos *bindings* WCF predefinidos. Nessa tabela, são definidas as seguintes propriedades:

- Configuração. – Este parâmetro está relacionado com a interoperabilidade aplicacional. Ex.: quais os *standards* suportados.
- Segurança. – Segurança ao nível do transporte ou ao nível da mensagem.
- Suporte de sessões.
- Suporte de transacções.
- Suporte de comunicações em ambos os sentidos (*duplex*).

Em algumas situações é possível alterar os valores predefinidos, alterando as propriedades respectivas. Por exemplo, para o *binding* “basicHttpBinding”, o valor por defeito relativa à “segurança” é “none” mas, pode ser alterado através da atribuição de outro valor ao campo <security> desse *binding*.

O WCF também permite ao programador criar os seus próprios *bindings* através do recurso ao *namespace* “System.ServiceModel.Channels”. Podem ser definidas funcionalidades relativas à segurança, codificação e “serialização”. Por exemplo, pode ser utilizada a propriedade “ProtectionLevel” de forma a forçar que sejam implementadas medidas relativas à integridade e confidencialidade através da assinatura e/ou encriptação. Exemplo:

```

<bindings>
  <wsHttpBinding>
    <binding name="teste">
      <security mode="Message">
        <message defaultProtectionLevel="Sign"/>
      </security>
    </binding>
  </wsHttpBinding>
</bindings>

```

Os contratos dos serviços expressam os métodos que estão disponíveis – são a *interface* do serviço. Um dos atributos do [ServiceContract] e do [OperationContract] é o

“HasProtectionLevel” que define um valor, só de leitura, que indica o nível de protecção do serviço. A nível operacional, é possível definir que a mensagem dessa operação deve ser encriptada, assinada ou ambas (encriptada e assinada).

Credenciais e declarações

A segurança do WCF é baseada em credenciais que consistem numa ou mais declarações (nome do utilizador, certificados digitais, *tokens*, *tickets* *kerberos*, etc.). É com base nas credenciais que são tomadas as decisões de permitir ou negar o acesso às aplicações/dados.

As credenciais podem ser apresentadas de duas formas: ao nível do transporte ou ao nível da mensagem. Se for utilizada a opção ao nível do transporte, as credenciais são apresentadas como fazendo parte do nível de transporte da mensagem (idêntico às comunicações SSL). Neste caso, os protocolos ao nível de transporte verificarão as credenciais e estabelecerão uma sessão segura entre o cliente e o serviço. No entanto, não existe uma segurança explícita para as mensagens que são enviadas utilizando essa camada de protecção ao nível do transporte. Infelizmente, a segurança ao nível do transporte termina nos pontos terminais do SSL. Se não forem tomadas outras medidas de protecção, logo que a mensagem sai do equipamento onde termina o túnel SSL, fica logo exposta aos intrusos. Na segunda opção, em que as credenciais estão incluídas na mensagem, esta só fica exposta a utilizadores maliciosos a partir do momento em que o receptor da mensagem a descodifica utilizando uma chave especial que ele próprio conhece. No entanto, este segundo método introduz alguma lentidão no processo devido à encriptação extra das mensagens e, além disso, o tamanho das mensagens também se torna maior.

No WCF existem as seguintes opções de tipos de apresentação das credenciais:

- Nenhuma credencial
- Segurança ao nível do transporte (SSL) – SSL só suporta um subconjunto de tipos de declarações: autenticação Windows (*Windows Basic Authentication*, NTLM e *Kerberos*), autenticação *digest* e certificados (X.509). Ou seja, no WCF, o SSL não suporta, por exemplo, WS-Security “rico” e declarações de *tokens* SAML.

Em alguns casos, opta-se por não pedir autenticação do cliente pois, como está a ser usado SSL, considera-se que não é necessário acrescentar mais este nível de segurança.

- Segurança ao nível das mensagens (segue as normas do WS-Security) – Este método permite o estabelecimento de contextos *end-to-end*, funcionando bem em ambientes com múltiplos intermediários que, por sua vez, podem utilizar mecanismos de autenticação diferentes. Ao contrário da segurança ao nível do transporte, este método já suporta declarações “ricas”: SAML, *tokens* personalizados, WS-Trust, etc.. No WCF, estão disponíveis os seguintes tipos de credenciais ao nível da mensagem: credenciais *Windows*, nome do utilizador (neste caso, WCF não fornece forma de encriptar o nome do utilizador), certificados (X.509) e *Infocard* (utilização do *Windows CardSpace* – exemplo de um STS (*Security Token Service*)). Exemplo:

```
WSHttpBinding binding = new WSHttpBinding();
Binding.Security.Mode = SecurityMode.Message;
Binding.Security.Message.ClientCredentialType = MessageCredentialType.Certificate;
```

<i>Binding</i>	Modo transporte?	Modo mensagem?	Modo “misto”?	Ambos?
BasicHttpBinding	Sim	Sim	Sim	Não
WsHttpBinding	Sim	Sim	Sim	Não
WsDualHttpBinding	Não	Sim	Não	Não
NetTcpBinding	Sim	Sim	Sim	Não
NetNamedPipeBinding	Sim	Não	Não	Não
NetMsmqBinding	Sim	Sim	Não	Sim
MsmqIntegrationBinding	Sim	Não	Não	Não

Tabela 6-3 – Suporte dos tipos de credenciais por cada um dos *bindings* WCF

- Modo “misto” (confidencialidade e integridade são garantidas pelo nível de transporte e a autenticação e autorização são através do nível da mensagem) – Este modo de apresentação das credenciais permite criar uma solução com uma gama de tipos de credenciais mais “rica” do que a segurança ao nível do transporte e mais rápida do que as soluções de segurança ao nível da mensagem.

Exemplo em que o URL é HTTPS (integridade e confidencialidade baseada no SSL) mas em que o cliente precisa de usar um certificado para poder aceder ao serviço:

```
Uri address = new Uri(https://server:8088/testservice);
WSHttpBinding binding = new WSHttpBinding();
Binding.Security.Mode = SecurityMode.TransportWithMessageCredential;
Binding.Security.Message.ClientCredentialType = MessageCredentialType.Certificate;
```

- Ambos (autenticação tanto ao nível da mensagem como ao nível do transporte).

Na Tabela 6-3 [21] estão resumidas as várias opções de apresentação de credenciais disponibilizadas pelos *bindings* WCF.

Autorizações

A *framework* do *.Net* oferece uma série de APIs, baseadas na *interface* “*IPrincipal*”, para gerir as autenticações e autorizações. Desta forma, é possível criar uma só vez, após a aprovação da autenticação, um objecto “*Principal*” que ficará na posse do cliente e que servirá todos os pedidos relacionados com a mesma sessão. O objectivo é transferir o contexto do cliente entre os vários sistemas que precisam de validar a autenticação/autorização, sem ser necessário alterar o código.

No *.Net*, as autorizações são baseadas em papéis/funções: um utilizador só tem acesso a determinada informação ou permissão para executar determinadas operações se pertencer a um grupo que tenha atribuído o “papel” respectivo. Ou seja, é necessário verificar os “papéis” que estão atribuídos a determinado grupo/utilizador através do uso da função “*IsInRole*”.

Auditoria

O WCF disponibiliza alguns utilitários que permitem registar as actividades, no *Event Viewer* dos servidores.

Confiança das comunicações na troca das mensagens

WCF permite implementar algumas medidas de forma a garantir alguma confiança a nível das comunicações baseadas em infra-estruturas que podem não garantir a correcta entrega das mensagens:

- Garantia de que cada mensagem só é entregue uma vez (não existem duplicados) e na ordem correcta.
- Medidas para recuperar de falhas da rede, das situações em que o servidor de destino está indisponível, erros SOAP, falhas dos intermediários, etc.. Estas medidas são implementadas através da conjugação dos seguintes parâmetros disponíveis no WCF: “*AcknowledgementInterval*”, “*FlowControl*” e “*InactivityTimeout*”.

Esta implementação é baseada no MSMQ (*Microsoft Message Queuing*) que disponibiliza um mecanismo de filas entre o cliente e o WS. No entanto, existem situações em que não é possível garantir a confiança das comunicações. Por exemplo, se o servidor estiver indisponível durante mais tempo do que o tempo máximo de espera definido no cliente, as mensagens são perdidas. Exemplo:

```
<!--configuracao do WsHttp binding para sessões confiáveis -->
<binding>
  <wsHttpBinding>
    <binding name="BindingDeConfianca">
      <reliableSession enable="true" />
    </binding>
  </wsHttpBinding>
</binding>
```

Actualmente, WCF suporta as opções de implementação “*ExactlyOnce*” (entregue só uma vez; nos outros casos gera um erro) com a opção de “*InOrder*” (entregue na mesma sequência que foi enviada; não trata mensagens duplicadas ou mensagens perdidas). Esta implementação é possível através da aplicação do elemento “*reliableSession*” num *binding* que suporte este tipo de sessões (“WSHttp”, “WSDual” ou “NetTcp”).

Um conceito que também é necessário ter em conta quando forem utilizadas estas funcionalidades do WCF relativas à garantia da confiança das mensagens, é o facto das sessões de confiança serem *statefull* e o seu estado se manter ao nível do domínio da aplicação. Portanto, como todas as mensagens que pertencem à mesma sessão têm de ser processadas dentro do mesmo domínio da aplicação, é preciso ter este factor em conta quando são utilizadas infra-estruturas em que o número de servidores é maior do que um.

6.3 Ferramentas de gestão de identidades e acessos

As ferramentas de gestão de identidades e acessos normalmente disponibilizam as seguintes funcionalidades:

- Gestão de vários “papéis” (identidades). – Na mesma instituição, o mesmo utilizador pode ter diferentes “papéis”, que variam conforme o contexto em que está inserido, os processos envolvidos, etc..
- Propagação das identidades. – As identidades têm de ser passadas entre os serviços envolvidos no processo de negócio. À identidade usada por um determinado processo, devem ser garantidos os acessos de forma a ser possível utilizar todos os recursos necessários para a conclusão desse processo, sem solicitar mais do que uma vez as credenciais. Ou seja, através da federação da identidade do utilizador entre as aplicações envolvidas, de uma forma transparente, é possível implementar um esquema de “*single sign-on*”.

- Integração das várias entidades envolvidas. – Como os processos de negócio podem envolver vários departamentos, empresas, países, etc., é necessário garantir a integração com sistemas “externos” ao departamento, à empresa, ao país, etc..
- Aplicação e gestão das regras de negócio relativas às identidades. – Após a criação e gestão de identidades é necessário definir regras relativas ao seu uso (ex.: limites do tempo de vida das identidades, regras relativas à *cache* de identidades, etc.).
- Delegação das funções de gestão às equipas de negócio. – Como a gestão das identidades é muito mais complexa do que gerir somente utilizadores/*passwords*, é necessário delegar as funções da sua gestão às equipas de negócio pois, na maior parte dos casos, estão envolvidos acessos a operações/dados relativos ao próprio negócio.

Os pacotes de *software* para gestão de identidades que actualmente se encontram disponíveis suportam os *standards* de segurança dos WS, tais como SAML e, quando as especificações ainda estão incompletas ou não estão ainda desenvolvidas, também suportam alguns *standards* suplementares com mecanismos de segurança proprietários. Por exemplo, alguns dos pacotes IAM (*Identity and Access Management*) usam uma combinação de autenticação local LDAP com autenticação distribuída SAML para os casos em que é necessário passar as credenciais para fora do domínio de segurança da própria empresa.

6.4 SOA Governance

O princípio básico de *governance* é a confiança, ou seja, todas as partes envolvidas (gestores de negócio, gestores das tecnologias de informação, programadores, parceiros de negócio e fornecedores) têm de poder confiar que as outras partes irão executar as suas funções correctamente, de acordo com as regras estabelecidas. No contexto de uma arquitectura SOA, através da implementação das funções de *governance*, é possível controlar os serviços, evitando que esta se torne numa arquitectura caótica em que os serviços são desenvolvidos/implementados sem ser-lhes aplicado qualquer conjunto de regras. Os principais objectivos a endereçar com a adopção de uma solução de SOA *Governance* são: simplificar a gestão, maximizar a segurança, maximizar a *performance* e definir SLAs.

Em termos de implementação, SOA *Governance* é uma combinação de políticas, processos e *metadata*⁴ em que podem ser definidos, por exemplo, mecanismos de comunicação entre todos os intervenientes, cadeias de responsabilidade, monitorização dos serviços, estabelecimento de políticas de acordo com os objectivos preestabelecidos, uso de mecanismos de controlo para assegurar a conformidade com as políticas/SLAs, etc.. Algumas das tecnologias disponíveis para controlar a forma como a infra-estrutura SOA é usada, gerida e testada, e como são aplicadas as medidas de segurança, são as seguintes:

- Gestor das políticas SOA. – Permite a criação, descoberta, referência e aplicação das políticas relacionadas com a infra-estrutura. Como as políticas são o ponto central do SOA *Governance*, preferencialmente, esse controlo deve ser efectuado a

⁴ dados que definem a origem do componente, o detentor desse componente e quem é que o pode alterar

partir de um ponto central de gestão (em vez de estar implementado a nível da configuração de cada um dos serviços). Essas políticas podem estar relacionadas com:

- Controlo de acessos (autenticação, autorização e auditoria).
- *Performance*, monitorização, disponibilidade e definição dos níveis de serviço.
- Políticas de desenvolvimento (ex.: requisitos relacionados com a linguagem de programação usada).
- Políticas de encaminhamento dos pedidos (encaminhamento com base no conteúdo, níveis de serviço, etc.).
- Políticas de transformação das mensagens.
- Correcta utilização das próprias políticas (ex.: aplicar as diversas políticas na ordem correcta pois, se não for seguida essa sequência, algumas políticas podem “anular” a segurança implementada por outras).
- Políticas de conformidade de forma a garantir o seguimento de determinadas normas impostas ao negócio (ex.: PCI DSS – *Payment Card Industry Data Security Standard*).
- *Registries* (registo de referências aos serviços e descoberta UDDI) e repositórios SOA (versões, *metadata*, contratos e políticas de acesso). – Permitem gerir *metadata* relacionada com SOA (serviços, políticas, processos, perfis, etc.) e, em alguns casos, criar as relações entre essa *metadata* e gerar a respectiva documentação (configuração e dependências).
- Tecnologia que permite garantir a qualidade da infra-estrutura SOA (boas práticas, princípios de programação, etc.) e efectuar algumas validações e testes relativos à consistência da própria arquitectura.
- Sistemas de monitorização dos WS e registo das actividades de forma a ser possível criar relatórios sobre o comportamento normal dos componentes da arquitectura, poder identificar situações anómalas, tomar decisões que permitam melhorar o desempenho dos WS, etc..
- Adaptadores, interfaces, aplicações e especificações WS que permitem a comunicação e partilha de informação entre todos os componentes da arquitectura. Numa arquitectura SOA é fundamental que as ferramentas de *governance* sejam integráveis com as aplicações existentes na infra-estrutura.

SOA *Governance* não implementa os serviços mas, ajuda a definir a forma como são desenhados e a responder, por exemplo, às seguintes questões:

- Que serviços estão disponíveis?
- Qual a sua fiabilidade?
- Quanto tempo decorre entre as várias versões dos mesmos serviços?
- Como proceder no caso de ser necessário alterar um serviço (ser necessário corrigir erros, novas funcionalidades, etc.)?
- Como proceder nos casos em que diferentes consumidores do mesmo serviço requerem comportamentos diferentes?

7 Boas práticas de segurança a seguir na implementação de uma Arquitectura Orientada a Serviços

Uma arquitectura SOA bem implementada deve ter mecanismos capazes de prevenir, proteger, filtrar e auditar:

- Prevenir – Bloquear as mensagens que potencialmente podem originar ataques ao WS.
- Proteger – Perante ataques cujo objectivo é tornar alguns elementos da infra-estrutura inoperacionais, as aplicações e a própria infra-estrutura devem ser capazes de se protegerem a elas próprias e aos sistemas aos quais precisam de aceder. Devem existir mecanismos de protecção que, na presença de determinados comportamentos, automaticamente tomem medidas para que o problema que está a afectar o sistema que está comprometido não se propague para o resto da infra-estrutura (ex.: o balanceador de tráfego deixar de redireccionar os pedidos para o sistema que está comprometido, etc.).
- Filtrar – A filtragem ao nível das mensagens deve incluir todas as funções das *firewalls* tradicionais, assim como também permitir que os administradores dos sistemas possam definir regras e acções relacionadas com mensagens específicas. Estas funções incluem validação dos esquemas XML, garantia da integridade, encriptação/descodificação, verificação da identidade, detecção de vírus, redireccionamento de algum tráfego para outros equipamentos que efectuem alguma filtragem específica, etc..
- Auditar – Registrar todas as operações efectuadas, independentemente de serem bem ou mal sucedidas.

Em seguida estão listados alguns dos princípios a considerar na implementação de uma arquitectura SOA⁵:

- Adicionar segurança à camada do nível de transporte (SSL/TLS). – Para acrescentar alguma segurança aos WS deverá ser introduzida ao HTTP, através do uso de SSL/TLS, uma nova camada de encriptação/autenticação.

Também é recomendada a utilização de certificados de servidor e, se possível, também certificados de cliente, para serem usados nos processos de autenticação. Desta forma, por exemplo, pode ser negado o acesso a sistemas críticos às entidades não consideradas de confiança. Estas medidas permitem proteger contra ataques conhecidos que se baseiam nas falhas de segurança relacionadas com as infra-estruturas que usam o endereço IP de origem e informação do DNS nas tarefas de controlo de acessos e autenticação.

Dado que as tarefas de terminação SSL/TLS requerem bastante processamento, numa situação ideal, estas tarefas podem ser implementadas em equipamentos baseados em *hardware* com funcionalidades de aceleração deste tipo de operações.

⁵ Alguns são baseados nas recomendações das empresas pertencentes ao grupo “*Fortune 500 Companies*”.

- “Mascarar” os recursos internos – uso de NAT e transformação XML.
 - “Mascarar” os recursos internos através do uso de NAT (*Network Address Translation*). – O endereço IP que é apresentado para exterior não deverá ser o endereço real do recurso interno correspondente.
 - “Mascarar” os detalhes dos WS internos através do uso de virtualização do serviço. – Através da virtualização dos serviços é possível apresentar ao utilizador uma “vista” do WS diferente da real. Esta funcionalidade é bastante importante pois permite “mascarar” alguns detalhes do WS, tais como, a sua plataforma (por ex.: se o nome terminar em “*.asmx” dá a entender que a plataforma é .NET). Se no futuro for necessário migrar os WS para outra plataforma, também se pode recorrer à virtualização dos serviços para facilitar esse processo de migração.
 - Inspeccionar o tráfego de saída de forma a detectar e remover os dados que não são necessários, nomeadamente, mensagens de erro (através das mensagens de erro retornadas é possível obter informação que pode ser bastante útil para alguém que queira atacar os sistemas onde estão implementados os WS).
 - Devido à existência de vários dialectos XML e vocabulários, muitas vezes, é necessário transformar o XML para “traduzir” as mensagens. Assim, é possível uma maior integração entre sistemas heterogéneos que podem suportar formatos e *standards* de mensagens diferentes.

O NAT pode ser implementado através de uma *firewall* convencional. A virtualização do serviço pode ser implementada por um *reverse proxy* com capacidade de reescrita dos URLs. As mensagens podem ser “transformadas” usando XSLT, sendo assim possível não divulgar às entidades externas os esquemas internos e estrutura dos objectos.

- Controlar as permissões de acesso ao WSDL. – Controlar o acesso aos WSDL através da implementação de políticas que autorizam ou não as mensagens de pedido dessa informação, as respectivas respostas e a recepção de mensagens em que são retornadas as explicações dos erros/falhas. Também devem ser definidas as normas a seguir na publicação dos WSDL pois é necessário enumerar os campos dos WSDL que deverão estar disponíveis nessa publicação, restringindo o acesso só a utilizadores autorizados.
- Proteger contra ataques XDoS. – Para proteger os WS contra ataques XDoS devem ser impostas restrições a aplicar às mensagens com destino ao próprio WS. Ex.: limite máximo do tamanho da mensagem, frequência das mensagens, duração das ligações, limites à profundidade máxima permitida dos objectos em cascata dos documentos baseados em XML, número máximo permitido de atributos por elemento, etc..

Esta validação deverá ser efectuada logo no perímetro, através do recurso a uma XML *Security Gateway* que actue como *proxy* ou *firewall*. Esta protecção a ataques deverá ser complementada com a implementação de filtragem XML.

- Implementar filtragem XML. – De forma a garantir que as mensagens não apresentam grande risco e podem ser encaminhadas para a rede interna, deverá ser efectuada alguma filtragem a nível do conteúdo XML. Os filtros podem basear-se, por exemplo, nos valores das seguintes variáveis: tamanho das mensagens, assinaturas digitais, conteúdo, endereço IP, cabeçalhos das mensagens, etc.. Como os filtros são baseados

em XML, quando aparecem novas ameaças, o processo da sua actualização geralmente não é uma tarefa muito difícil de ser executada.

Este tipo de filtragem deve ser efectuado o mais cedo possível, ou seja, logo no perímetro, antes de chegarem à zona onde estão os servidores com funções/informações mais críticas. No entanto, podem existir algumas validações que requerem o acesso a base de dados internas que, por serem críticas, não devem ser acedidas directamente pelos sistemas que estão na primeira linha de acessos (perímetro). Nesse caso, essas validações poderão ser efectuadas mais tarde, por um sistema que esteja já perto do WS ou, em alternativa, pelo próprio WS.

Através deste tipo de validação podem ser detectados, por exemplo, ataques “SQL Injection”.

- Validar todas as mensagens XML relativamente à sua estrutura. – Todas as mensagens, destinadas ou originadas pelos WS, devem ser analisadas por um *parser* XML de forma a verificar se os seus esquemas estão de acordo com os esquemas locais ou remotos. Também deve ser verificado se o XML está bem formatado, se faltam referências a alguns recursos, validação a nível do protocolo (ex.: SOAP), etc..

Para prevenir ataques baseados em falhas da definição do conteúdo, semântica e estrutura dos WS, nos esquemas XML deverão ser impostas restrições detalhadas e definidos alguns limites (tamanho dos campos, tipos de dados, etc.). Ex.:

```
<xs:simpleType name="CodigoPostal">
  <xs:restriction base="xs:string">
    <xs:pattern value="([0-9]{4})-([0-9]{3})">
  </xs:restriction>
</xs:simpleType>
```

Desta forma, é possível impedir que mensagens mal formatadas sejam processadas pelos servidores aplicacionais. Esta é uma medida que permite evitar que alguns ataques à infra-estrutura sejam bem sucedidos: “*coersive parsing*”, “*recursive payload*”, “*oversized payload*”, etc.. Validando se as mensagens estão de acordo com os esquemas XML permite evitar, por exemplo, ataques “*parameter tampering*”.

Esta validação deverá ser efectuada logo no perímetro, através do recurso a uma XML *Security Gateway* que actue como *proxy*.

- Validar que as mensagens não contêm vírus, tanto no próprio XML como nos anexos.
- Bloquear, ou restringir ao mínimo necessário, as referências externas existentes nas mensagens SOAP. Desta forma, podem ser evitados ataques do tipo “*external entity attack*”.
- Canonizar os dados para garantir que os documentos XML, além de estarem bem formatados (funções de validação do esquema), também seguem determinadas normas de formatação que garantem que o documento XML, independentemente de onde é criado (no fornecedor ou consumidor do WS), seja sempre idêntico.
- Segurança ao nível da mensagem: proteger o conteúdo das mensagens através de assinaturas, *timestamps* e encriptação. – De forma a proteger os sistemas contra “repúdio”, todas as mensagens devem ser assinadas digitalmente e devem ser adicionados *timestamps*. Através da assinatura das mensagens, além da garantia da integridade dos dados e autenticação, através do registo de todas as mensagens e

respectiva assinatura, que pode ser verificada após as transacções, o emissor da mensagem consegue proteger-se contra ameaças de “repúdio”. Se todos os intervenientes da arquitectura SOA tiverem os seus relógios sincronizados por uma fonte segura, os *timestamps* presentes nas mensagens assinadas podem ser considerados de confiança e, desta forma, também proteger os WS contra as ameaças de “repúdio”. Além das medidas anteriores, para garantir a confidencialidade dos dados, também se deve encriptar toda a mensagem ou, alguns campos de dados dos documentos, com chaves de encriptação diferentes. Devem ser utilizados algoritmos de encriptação fortes (RSA/DSA/3DES/AES) de 128-bit ou mais.

Como as tarefas de encriptação/descodificação e a assinatura digital das mensagens requerem bastante processamento, quando estão envolvidas muitas transacções, podem introduzir alguma degradação na *performance* dos sistemas onde estão os WS. Portanto, é aconselhável que estas operações sejam consolidadas num único equipamento que possa efectuar ambas as funções e optimizado para tal.

De forma a implementar estes níveis de segurança é aconselhável, pelo menos, aplicar os standards WS-Security, XML-Encryption e XML-Signature. No entanto, é necessário ter cuidado na combinação da encriptação e assinatura pois, o objectivo de um pode comprometer o objectivo do outro. Por exemplo, se forem encriptados dados assinados sem encriptar a assinatura, deixa-se a “porta aberta” a ataques.

- Garantir a integridade das comunicações. – Garantir que, no contexto da troca de mensagens, nenhuma é “perdida” e elas só são recebidas uma única vez e na mesma ordem que são enviadas. Esta garantia pode ser implementada através do uso de WS-ReliableMessaging, assinatura das mensagens, WS-Addressing, etc..

Implementando medidas que permitam detectar mensagens repetidas, em conjunto com o uso de autenticação forte e introdução de *timestamps* nas mensagens, é possível detectar os “*replay attacks*”.

- Autenticação, autorização e responsabilização. – Controlar os acessos aos WS de forma a garantir que só consumidores devidamente autorizados conseguem aceder. De forma a implementar este controlo de acessos, estão disponíveis vários métodos, nomeadamente: autenticação ao nível do transporte, autenticação por *tokens* recorrendo à especificação WS-Security (uso de declarações SAML ou outros *tokens*), autenticação através dos cabeçalhos SOAP, etc..

Em função do nível da “*stack IP*”, podem ser utilizados métodos de autenticação diferentes. Em seguida estão alguns exemplos que podem ser utilizados:

- TCP/IP: endereço IP.
- SSL: certificado X.509 do cliente.
- HTTP: Autenticação Básica, NTLM/SPEGNO.
- XML: nome de utilizador/*password* embebidas.
- SOAP: uso de *tokens* do WS-Security.

Em infra-estruturas em que é necessário o maior grau de segurança possível, é recomendado que os *tokens* de segurança estejam incluídos na própria mensagem. Ou seja, cada uma das transacções WS devem incluir a informação relativa à sua identificação (o que é que a transacção faz) e quem é que a criou e autorizou. Com a

utilização de *tokens* de confiança (ex.: declarações SAML), além de ser possível incluir essas informações na própria transacção, também é possível garantir que essa informação foi incluída recorrendo a mecanismos seguros e capazes de a confirmarem.

De forma a evitar os ataques baseados em falhas de autenticação (“*brute force*”, “*spoofing*”, autenticação fraca, etc.), devem ser usadas técnicas de autenticação forte e, em alguns casos, usar o XML-Signature. Os ataques do tipo “*man-in-the-middle*” podem ser evitados através do uso de autenticação mútua.

O uso de autenticação baseada só em *passwords* simples tem algumas desvantagens:

- As *passwords* são fáceis de roubar.
- Muitas vezes, quando são implementadas políticas para obrigar o uso de *passwords* complexas, os resultados esperados não são os obtidos. Como as *passwords* são complexas, o utilizador após as “inventar”, como muitas vezes não as consegue memorizar, escreve-as, correndo o risco de as perder ou serem vistas por outros utilizadores.
- Os utilizadores muitas vezes reutilizam as suas *passwords* em vários sistemas pois não conseguem memorizar muitas *passwords*.

A implementação de “*password digest*” resolve o problema de se ter de usar SSL/TLS para adicionar alguma segurança aos casos em que é usado o método de *username/password* pois, caso contrário, a *password* vai em claro. No entanto, o uso de SSL/TLS tem o problema de não permitir a existência de intermediários “activos” no processamento destas mensagens.

Relativamente ao uso de autenticação baseada em *Kerberos*, apesar do uso de PKI ser melhor em termos de segurança, existem algumas razões para ser recomendado o seu uso:

- *Kerberos* foi a primeira tecnologia capaz de implementar algumas das boas práticas de segurança actualmente em vigor e, muitas das ideias dos *standards* WS foram baseadas nos princípios do *kerberos*.
- A Microsoft usa *kerberos* (com as suas extensões) para a segurança dos seus servidores. Hoje em dia, muitas das implementações de WS são baseadas nos produtos desse fornecedor e, se não o forem, é comum ter de interagir com produtos ou WS baseados nessas tecnologias.
- *Kerberos* também pode ser usado para proteger a confidencialidade e verificar a integridade das mensagens.

Mas, apesar destas vantagens, *kerberos* tem uma grande desvantagem: não é uma solução facilmente escalável para fora da empresa pois depende de um KDC central ou de uma cadeia de KDCs com relações de confiança estabelecidas. Nestes casos, é preferível optar por uma solução baseada em PKI. Como, na maior parte das vezes, o uso de PKI apresenta uma melhor opção, este é mais utilizado do que o *kerberos*.

Muitas vezes, as tarefas relacionadas com as autorizações de acesso aos WS são efectuadas através do recurso a aplicações proprietárias. Nestes casos, é recomendável que se avalie a utilização da especificação XACML, desenvolvida pela OASIS. Ela foi criada precisamente para ajudar na definição das decisões de “autorização”, eliminando a necessidade de desenvolver aplicações próprias de cada empresa para

estas operações e reduzindo, desta forma, os custos associados com o desenvolvimento e testes das mesmas.

Outra questão a ter em conta na arquitectura de controlo de acessos é a necessidade de implementar mecanismos de monitorização, registo e envio de alertas (avisar quando existe um grande número de autenticações falhadas, tentativas sucessivas de acesso a recursos para os quais não está autorizado, etc.).

Este controlo de acessos deverá ser efectuado antes de chegar ao próprio WS. No entanto, dependendo da arquitectura, pode ser efectuado logo no perímetro ou mais perto do próprio WS. Se for efectuado muito perto do perímetro, como o equipamento que efectua esse controlo tem de aceder às bases de dados de utilizadores/acessos, esses equipamentos poderão ser alvo de ataques “*brute force*” em que se podem descobrir quais as credenciais necessárias para aceder aos WS.

- Implementar esquemas de auditoria seguros. – É necessário garantir que a informação relativa ao registo das actividades (acesso aos serviços e alterações de configurações) está a ser recolhida, guardada em locais seguros e não é alterada. Como o *syslog*, sozinho, não é considerado seguro, através da combinação do *syslog* e do uso de assinaturas digitais XML e *timestamp* aplicadas aos próprios registos, é possível implementar um esquema de registo de actividades relativamente seguro.
- Gestão das relações de confiança. – Uso de WS-Trust, WS-SecureConversation e WS-Federation.
- Gestão centralizada de PKI.
- Utilização de *standards* de segurança (em vez de mecanismos proprietários) de forma a permitir a interoperabilidade entre as diferentes organizações. Embora os *standards* de segurança não consigam só por si resolver todos os problemas de segurança, é recomendada a sua utilização pois eles são implementados e actualizados periodicamente por especialistas qualificados (representantes de várias empresas mundiais) que efectuem testes exaustivos. Sempre que se justifique, as organizações responsáveis pela manutenção desses padrões, publicam novas versões com correcções e novas funcionalidades.

Um dos *standards* a ter em conta deverá ser o WS-I *Basic Security Profile* (BSP) pois este restringe o uso dos *standards* W3C e OASIS numa forma que não crie problemas de interoperabilidade entre as plataformas e garantindo um conjunto de boas práticas de segurança. De forma a verificar se a implementação SOA está de acordo com as recomendações WS-I podem ser usadas algumas ferramentas de teste disponíveis no site da *Web Services Interoperability* (WS-I) *Organization*.

Na Tabela 7-1 estão listadas algumas das medidas de segurança que podem ser implementadas através do uso de alguns dos *standards* disponíveis:

Tecnologia/Standard	Alguns dos benefícios de segurança obtidos
SSL/HTTPS	Confidencialidade, integridade, autenticação, não repúdio.
WS-Security	Confidencialidade, integridade, autenticação, não repúdio – (segurança ao nível da mensagem).
XML-Encryption	Confidencialidade, integridade.
XML-Signature	Integridade, autenticação, não repúdio.
XKMS	Gestão de chaves.
SAML	Autenticação, autorização.
XACML	Autorização.
WS-Trust	Gestão e estabelecimento de relações de confiança.
WS-Federation	Federação de identidades.
WS-Policy, WS-SecurityPolicy, WS-PolicyAssertions, WS-PolicyAttachment	Definição de políticas de segurança.
WS-Reliability, WS-ReliableMessaging, WS-Addressing	Garantia de confiança nas comunicações (segurança na entrega das mensagens).
WS-SecureConversation	Estabelecimento e partilha de contextos de segurança e derivação das chaves de sessão.

Tabela 7-1 – Tecnologias/standards e respectivos benefícios de segurança obtidos com a sua utilização

- Garantir uma alta *performance* e confiança da infra-estrutura. – Implementação de esquemas de balanceamento eficazes, suporte a falhas, possibilidade de repor configurações anteriores, etc.. De forma a garantir uma boa *performance*, deverão ser implementadas formas de compressão de dados, optimização de algumas operações, definir regras de prioridade do tráfego em função do WS em causa (ex.: um WS associado a funções que têm de ser efectuadas em tempo real, deverá ter alguma percentagem dos recursos de rede/processamento reservados para ele de forma a poder oferecer SLAs satisfatórios), etc..
- Gerir os recursos de uma forma centralizada e segura. – Gestão centralizada das configurações aplicadas aos vários elementos da arquitectura SOA (ex.: XML Security Gateways). Essa gestão deverá ser efectuada de uma forma segura, tentando proteger as *interfaces* de administração (acesso remoto via SSH, HTTPS ou outro protocolo encriptado, configuração de ACLs às VLANs onde se encontram os IPs dessas *interfaces*) e garantindo que não existe a possibilidade de “elevação de privilégios” de administração.

Para a definição das políticas a aplicar aos diversos equipamentos deverão ser seguidas, por exemplo, as recomendações constantes nos *standards*: WS-Policy, WS-PolicyAssertions, WS-PolicyAttachment, WS-SecurityPolicy.

- Utilização de aplicações de gestão que sejam fáceis de utilizar e intuitivas. – Como a arquitectura SOA tem de ser gerida/monitorizada por várias pessoas, de equipas diferentes e, provavelmente, com níveis de conhecimento heterogéneos, as aplicações usadas para a definição/aplicação das configurações, monitorização do estado dos WS e tarefas de manutenção (cópias de segurança, actualizações, paragem e arranque de serviços, etc.) devem ser fáceis de utilizar.
- Possibilidade de integração dos vários elementos que constituem a arquitectura SOA. Por exemplo, deve ser possível integrar a solução de segurança com a aplicação que

contém a base de dados de utilizadores, deve ser possível partilhar algumas configurações/elementos de segurança por vários WS, monitorizar os elementos da arquitectura SOA a partir de soluções que já estejam a monitorizar outros equipamentos/aplicações da empresa, etc..

- Monitorização, em tempo real, de toda a infra-estrutura e envio de alertas em caso de:
 - Falha de alguns dos elementos da arquitectura SOA.
 - Serem atingidos determinados limites relacionados com SLAs, número de ligações simultâneas ao mesmo WS, etc..
- Periodicamente, efectuar testes de verificação da não existência de falhas de segurança, verificação da conformidade da arquitectura com as regras definidas pela própria empresa, verificação/actualização da utilização dos *standards* de segurança existentes.
- Optar pela existência de algum grau de heterogeneidade nas soluções. – Uma forma de aumentar a segurança e a “resistência total” da infra-estrutura a ataques baseados num determinado conjunto de vulnerabilidades é através da implementação de uma solução “heterogénea”. Ou seja, deve avaliar-se a possibilidade de implementar uma infra-estrutura em que nem todos os componentes sejam do mesmo fabricante e/ou baseados na mesma tecnologia. Desta forma, na presença de um ataque que explora uma determinada vulnerabilidade, este pode não propagar-se para todas as soluções. Como uma das vantagens de uma arquitectura SOA é o facto de ser suportada em várias plataformas, com diferentes tecnologias, podemos basear-nos nesta grande vantagem para tomar medidas que protejam internamente a própria arquitectura. Como desvantagens do aumento do grau de heterogeneidade, podemos listar o facto de ser necessário formar os recursos humanos para conseguir implementar e manter soluções diferentes, necessidade da existência de ferramentas de administração para cada uma das tecnologias, impedimento de concentrar nos mesmos sistemas todos os serviços ou equipamentos de infra-estrutura, etc.. Ou seja, cada uma das organizações deverá conseguir pesar os prós e contras e optar pelo grau de heterogeneidade que considerar satisfatório.

8 Critérios de avaliação das XML Security Gateways

Tal como já foi referido na Introdução, um dos objectivos deste trabalho é a avaliação de algumas XML Security Gateways que permitem adicionar alguma segurança à implementação de uma arquitectura SOA baseada em Web Services. Neste capítulo serão listados alguns dos critérios⁶ que poderão ser tomados em conta durante a avaliação das XML Security Gateways.

8.1 Arquitectura de implementação

a) Implementação física

As XSG podem ser baseadas em:

- *Hardware* dedicado. – Neste tipo de gateways devem ser comparados os valores de *performance* que podem ser obtidos.
- *Software*. – Neste tipo de implementações devem ser comparados os requisitos de *hardware*, sistema operativo, possibilidade de integração com componentes de *hardware* que podem aumentar a *performance* (ex.: placas de encriptação SSL), etc..
- *Hardware* dedicado e *software*.
- *Hardware* dedicado ou *software*.

b) Suporte de virtualização do próprio equipamento onde está implementada

No caso de ser baseada em *hardware* dedicado, possibilidade de criar múltiplos contextos de gateways XML no mesmo equipamento. Se se tratar de uma implementação por *software*, a possibilidade de instalar a aplicação num servidor virtual (ex.: VMWare, Microsoft Virtual Server, etc.).

c) Modo de operação

Este critério serve para classificar a localização e modo de operação da XSG. Os valores possíveis são os seguintes:

- *Bridge* – Pode ser instalada como uma *bridge* transparente.
- *Reverse proxy* – Na arquitectura devem existir regras de encaminhamento para direccionarem todo o tráfego relacionado com os WS para a XSG.
- *Plug-in* – As funções de XSG são implementadas a nível de um *plug-in* instalado no próprio servidor onde está implementado o WS.

d) SSL

De forma a acrescentar um nível de segurança às mensagens, normalmente é utilizado o SSL. No entanto, como o tráfego é encriptado ponto a ponto, os

⁶ Alguns dos critérios de avaliação apresentados foram baseados nas recomendações listadas no “XML Security Gateway Evaluation Criteria Project” [18] desenvolvido pelo OWASP (*Open Web Application Security Project*).

equipamentos intermediários não conseguem efectuar algumas validações. Relativamente a este critério, podemos classificar as XSG da seguinte forma:

- Terminam SSL – As operações de terminação SSL são implementadas a nível da XSG. Desta forma, a *gateway* encripta e descodifica o tráfego para conseguir aceder ao conteúdo das mensagens. As comunicações entre a *gateway* e o WS podem ser efectuadas em texto claro ou serem encriptadas novamente.
- De uma forma passiva, descodifica o SSL – Configurada com uma cópia da chave privada SSL do WS, a *gateway* descodifica o tráfego SSL e analisa-o. Entretanto, a mensagem original é encaminhada para o WS, sem sofrer qualquer alteração. O WS em seguida descodifica a mensagem e processa-a.
- Não aplicável – Se a XSG for implementada como uma espécie de *plug-in* instalado no mesmo sistema onde está o WS, em alguns casos, ela pode só efectuar as suas funções após a mensagem ser descodificada pelo próprio WS.

e) Capacidade de estabelecer VPNs com parceiros

f) Garantia da disponibilidade

De forma a garantir a alta disponibilidade das *gateways*, estas deverão ser implementadas, pelo menos, aos pares. O balanceamento do tráfego entre as *gateways* pode ser efectuado por equipamentos externos ou elas próprias têm mecanismos que permitem gerir o balanceamento do seu próprio tráfego e detecção de falhas de alguns dos membros do *cluster*. Se a gestão do balanceamento e detecção de falhas for efectuada por mecanismos “próprios”, para cada tipo de *gateway*, deverão ser recolhidas as seguintes informações:

- Qual o número máximo de *gateways* que podem compor um *cluster*?
- Os nós podem estar distribuídos geograficamente?
- Quais são os tipos de problemas detectados na *gateway* que originam uma situação de falha? Ao final de que tempo é que um determinado problema é considerado razão suficiente para passar a um estado de falha do sistema?
- O tráfego é balanceado por todos os nós do *cluster* de *gateways* ou, em cada momento, só está uma activa?
- A tabela de estados é partilhada por todas as *gateways* do mesmo *cluster*? Esta partilha aplica-se a sessões SSL?

g) Virtualização/Transformação dos WS

- “Mascarar” os serviços. – Por razões de segurança e regras de encaminhamento/balanceamento dos pedidos efectuados aos servidores aplicativos, se a *gateway* permitir virtualização dos WS, estes são expostos aos clientes com endereços/atributos “virtualizados”. Ou seja, os seus endereços/atributos reais são “mascarados”. A virtualização da representação externa dos recursos internos pode incluir “mascarar” as seguintes características do WS: endereço IP do servidor, porta de rede, URL real, domínios de autenticação, alteração das mensagens de erro retornadas, etc..

- Transformação XML. – Métodos de transformação do XML suportados: XSLT, XPath, Xquery, etc..
- Transformação de protocolos. – Exemplos de transformação de protocolos: converter as credenciais de autenticação de HTTP Básica para SAML, converter AJAX para SOAP, passar uma mensagem SOAP por HTTP para uma fila de mensagens JMS (*Java Messaging Service*), etc..
- Encaminhamento dos pedidos com base no seu conteúdo. – Encaminhamento dos pedidos em função do conteúdo do cabeçalho ou corpo da mensagem, credenciais do utilizador, identidade e/ou “classe de serviço” do cliente, disponibilidade e/ou *performance* do serviço, etc..
- Métodos de balanceamento dos WS. – Para cada serviço, a *gateway* deverá ter indicações da lista de servidores que o fornecem e que tipo de distribuição dos pedidos deverá implementar. A distribuição dos pedidos pelos servidores poderá ser efectuada recorrendo a algoritmos mais simples (ex.: *round-robin*) ou a algoritmos que têm em conta o estado dos recursos dos servidores em cada momento (a *gateway* terá de conseguir recolher as estatísticas relativas ao estado das aplicações em cada servidor).

Algoritmos de distribuição dos pedidos: *round-robin*, servidor com menor número de ligações, servidor com maior número de recursos disponíveis, etc..

8.2 Identificação e Autenticação

a) Métodos de autenticação suportados para as ligações com o WS

Neste critério serão listados os métodos de autenticação suportados a nível do serviço e das mensagens com destino/origem nos WS. Ex.: autenticação SSL mútua, HTTP *Basic*, HTTP *Digest*, WS-Security *Username Token*, WS-Security certificados X.509, WS-Security *Kerberos Token*, SAML, validação contra a *Active Directory*, *Active Directory Federation Services*, etc..

b) Protecção das Bases de Dados dos utilizadores ou, no caso de estarem em sistemas remotos, segurança das comunicações com essas Bases de Dados

Se as Bases de Dados dos utilizadores estiverem localizadas na própria *gateway*, deverão ser garantidas medidas de segurança que permitam a sua gestão segundo as boas práticas (só dar acesso a quem efectivamente deve-o ter, criar grupos de administração com as permissões necessárias, garantir a confidencialidade e integridade dos dados dos utilizadores, etc.). Se a Base de Dados estiver remota, deverá ser garantida a comunicação segura entre a *gateway* e esses sistemas (ex.: uso de sLDAP – *secure LDAP*).

c) Funções de proxy relacionadas com autenticação suportadas

Lista das funções de *proxy* relacionadas com a autenticação que são suportadas pela XSG. Para exemplificar, considerando um cenário em que existem dois sistemas com tarefas de validação (sist1 e sist2), podem ser consideradas as seguintes funções de autenticação suportadas pelas XSG:

- “*Trusted proxy*” – Neste caso, como o sist2 confia no sist1, o utilizador só se tem de autenticar no sist1. Após o sist1 autorizar o acesso do utilizador aos

serviços protegidos pelo sist2, o sist1 passa o pedido para o sist2 em seu nome (em nome do sist1).

- “*Authentication impersonator*” – O utilizador autentica-se no sist1 e envia-lhe também a *password* necessária para se autenticar no sist2 (ou outros dados de autenticação). O sist1 usa a *password* necessária para aceder aos recursos do sist2 em nome do utilizador. Em seguida, é o próprio sist2 que autoriza ou não o acesso aos recursos que está a proteger.
- “*Identity-asserting impersonator*” – Idêntico ao “*Authentication impersonator*” mas, como o sist2 confia no sist1 para verificar a correcta identidade do utilizador, não pede a *password* ao sist1. Ou seja, o sist1 não precisa da *password* do utilizador.
- Delegação – Nesta variante, o protocolo de segurança partilhado pelo sistema do utilizador, sist1 e sist2, suporta “delegação”. Ou seja, a capacidade do utilizador autorizar, para determinadas transacções, que o sist1 assuma a sua identidade perante o sist2.
- “*Authorizing proxy*” – O sist1 autentica e autoriza o utilizador e o sist2 simplesmente deixa que todos os pedidos com origem no sist1 sejam processados (sist2 não pede qualquer utilizador/*password*).
- “Túnel de *login*” – Neste caso, o sist1 autentica o utilizador e permite (ou não) que o tráfego seja enviado para o sist2. Caso a primeira autenticação seja efectuada com sucesso, em seguida, o sist2 autentica novamente o utilizador e autoriza, ou não, o acesso aos recursos.

Na Tabela 8-1 está apresentado um resumo das funções de *proxy* relacionadas com a autenticação. Na primeira linha dessa tabela está caracterizada a situação “ideal” para este tipo de funções e, a negrito, estão as situações que envolvem algum risco e cuja utilização deverá ser ponderada. O significado de cada uma das colunas é o seguinte:

- Password para sist1 – O utilizador tem de fornecer ao sist1 uma *password* (ou outros dados de autenticação) a ser usada nos acessos ao sist2. Existe um risco associado a este tipo de autenticação pois, como o sist1 conhece essa *password*, pode usar esses dados para autenticar-se, em nome do utilizador, perante transacções supostamente não autorizadas.
- ID do utilizador para sist2 – O identificador do utilizador é apresentado ao sist2 para que este efectue decisões baseadas nas suas políticas.
- Sist2 autenticação – Dados que é preciso passar para o sist2 para que ele execute as tarefas relacionadas com a autenticação (utilizador, sist1, nenhum dado, etc.).
- Sist2 autorização – A política de autorização do sist2 é baseada em determinada informação: utilizador, sist1, nenhum dado, ...
- SSO – O utilizador só precisa de fornecer uma vez as suas credenciais.
- Protocolo de delegação – O sistema do utilizador, o sist1 e o sist2 têm todos acesso ao mesmo protocolo de segurança e esse protocolo implementa funções de delegação.

Tipo autenticação	Password para sist1	ID do utilizador para sist2	Sist2 autenticação	Sist2 autorização	SSO	Protocolo de delegação
A ideal	Não	Sim	Utilizador	Utilizador	Sim	Não
“Trusted proxy”	Não	Não	Sist1	Sist1	Sim	Não
“Authentication”	Sim	Sim	Utilizador	Utilizador	Sim	Não
“Identity-asserting”	Não	Sim	Nada	Utilizador	Sim	Não
Delegação	Não	Sim	Utilizador	Utilizador	Sim	Sim
“Authorizing proxy”	Não	Não	Nada	Nada	Sim	Não
“Túnel de login”	Não	Sim	Utilizador	Utilizador	Não	Não

Tabela 8-1 – Funções de proxy relacionadas com a autenticação
(Fonte: “Security Design Patterns“[2])

8.3 Autorização

a) Standards de segurança suportados

Quais os *standards* de segurança relacionados com funções de autorização que são suportados: XACML, SAML, etc..

b) Marcação das mensagens já autorizadas

Descrição do tipo de marcação que a XSG efectua nas mensagens já definidas como autorizadas. Através desta marcação, os consumidores e fornecedores dos serviços poderão verificar que já foi aplicada às mensagens uma política de autorização.

8.4 Registo das actividades

a) Lista dos dados registados

Opções de registos que podem ser obtidos: IP de origem, tipo de autenticação, identificador do serviço, utilizador, etc..

b) Ferramentas disponíveis para visualização dos dados registados

As ferramentas disponíveis para análise dos dados registados podem ter uma *interface* gráfica de fácil utilização, gerar automaticamente relatórios/gráficos (quais os formatos: Word, RTF, HTML, PDF, XML, etc.), permitir efectuar filtragem dos dados registados, etc..

c) Possibilidade de envio de alertas perante determinados eventos

d) Armazenamento dos registos

Os registos podem ser guardados na própria XSG, enviados para um servidor de ficheiros/base de dados, etc..

Neste parâmetro de avaliação também estão incluídas as funções relacionadas com a gestão das cópias de segurança da informação registada e importação/exportação desses dados.

e) Segurança da informação registada

Sistemas de controlo de acessos aos dados registados, possibilidade de assinar os registos através de XML-Signature, forma de envio dos registos da *gateway* para sistemas de gestão dos registos (*syslog*, protocolo proprietário, encriptação dos dados, etc.), possibilidade de “mascarar” alguns campos dos registos (conforme os privilégios atribuídos ao utilizador, este terá ou não acesso ao conteúdo de determinados campos), etc..

f) Correlação dos dados

Existência de formas/ferramentas de correlacionar os dados registados na *gateway* com os registados no próprio WS de forma a ter um registo de dados global de todos os sistemas envolvidos.

8.5 Validação do conteúdo das mensagens/Detecção de ataques

a) Métodos de normalização suportados

Muitas vezes, o atacante tenta transformar o *payload* do ataque numa forma que pareça não apresentar qualquer perigo. De forma a garantir a utilidade das regras e assinaturas de ataques, as XSG devem tentar detectar os ataques e transformar os dados numa forma normalizada. Portanto, um dos critérios de avaliação das *gateways* consiste na listagem dos métodos de normalização suportados: *URL-decoding*, “*null byte string termination*”, “*self-referencing paths*” (ex.: uso de */./*) , “*path back-reference*” (ex.: uso de */../*), “*mixed case*”, uso excessivo de espaços em branco, remoção de comentários, conversão de caracteres “*backslash*” em caracteres “*forward slash*”.

b) Modelo de segurança utilizado

Suporte dos seguintes modelos de segurança:

- “Negativo” (*blacklist*) – Neste modelo, por defeito, as transacções são permitidas. Só as identificadas como sendo ataques é que são rejeitadas. Essa detecção pode ser efectuada com base em assinaturas ou em regras.
- “Positivo” (*whitelist*) – Neste modelo, por defeito, todas as ligações são negadas e só são permitidas as transacções que são conhecidas como sendo válidas e seguras. Esta aproximação pode ser mais segura e mais eficiente porque são processadas menos regras mas, requer um conhecimento exaustivo do funcionamento das aplicações que estão a ser protegidas. Se o comportamento das aplicações for alterado frequentemente, este modelo é mais difícil de manter. No caso da XSG suportar este modelo, é importante também verificar se é possível, após um período de aprendizagem do tráfego normal, que esta proponha automaticamente quais as regras que deverão passar a ser aplicadas.

c) Base de dados de assinaturas e regras

Relativamente às bases de dados de assinaturas, existem algumas características que podem ser analisadas:

- A XSG já tem algumas assinaturas e regras predefinidas?

- Qual é a frequência de actualizações de assinaturas?
- A base de dados está dividida em sistema operativo, aplicações, versões, etc., de forma a ser possível aplicar só alguns grupos de assinaturas?

d) Validação da formatação do XML

As XSG podem efectuar validações que permitem determinar se o XML está bem formatado, ou seja, se está de acordo com as regras da sintaxe do XML (ex.: verificar se todas as *tags* são fechadas, se o documento XML chegou completo, etc.).

Existem vários tipos *standards* de *parsers* XML, assim como *parsers* XML adaptados a cada empresa. No entanto, é aconselhável utilizar os *standards* pois foram criados para serem universais e, em princípio, como são sujeitos a um grande número de testes, são mais eficientes em termos de detecção de ataques.

e) Suporte da validação dos esquemas XML

Os pedidos efectuados a determinado WS devem ser validados de forma a verificar se estão de acordo com o que está definido nos esquemas XML desse WS. Ex.: verificar se os tipos de dados são consistentes, verificar o tamanho mínimo/máximo das cadeias de caracteres, etc..

f) Validação/filtragem do conteúdo das mensagens

Validação do conteúdo de forma a detectar “*injection attacks*”, “*external entity attack*”, prevenção de ataques “*buffer overflow*”, etc..

Através da análise dos dados constantes na mensagem pode ser possível detectar se o contexto da informação representa algum risco. Para tal, é necessária conhecer bem o formato e conteúdo possível dos dados XML.

A XSG também poderá efectuar alguma filtragem dos pedidos de acordo com algumas políticas estabelecidas relativas ao conteúdo das mensagens XML. Por exemplo, se a mensagem XML contiver um determinado valor, esse pedido poderá ser recusado pela própria XSG, retornando uma mensagem para a origem.

g) Antivírus específico para mensagens XML e respectivos anexos

8.6 Filtragem com base em IP/porta

Possibilidade de filtrar o acesso aos WS com base no endereço IP de origem e porta de destino (HTTP, HTTPS, etc.).

8.7 Método usado para bloqueio do tráfego

Após a análise das mensagens e aplicação das políticas de acesso/validação, em alguns casos, é a própria XSG que bloqueia as mensagens. Este bloqueio pode ser efectuado da seguinte forma:

- Intermediário da ligação – O tráfego é inspeccionado e as ligações são terminadas na XSG. Os ataques são bloqueados através do não encaminhamento dos pedidos para o seu destino.

- Interrupção da ligação – O tráfego é inspeccionado mas as ligações não são terminadas na XSG. O bloqueio dos ataques é efectuado através da interrupção das ligações. Esta interrupção pode ser efectuada antes da chegada de algum pacote ao destino (ataque baseado num único pacote) ou após parte da ligação já ter chegado ao destino mas, ainda não completamente (essa parte ainda está no *buffer* do destino) (ex.: ataques baseados em pacotes segmentados).
- Terminação da ligação – O tráfego é inspeccionado pela XSG através de mecanismos de inspecção activos, passivos ou embebidos. Os ataques são bloqueados através da terminação das ligações TCP. Este tipo de bloqueio normalmente é utilizado em conjunto com outros mecanismos de bloqueio.
- Bloqueio através de um equipamento externo – O tráfego é inspeccionado pela XSG. O bloqueio é efectuado através do envio de notificações para outros equipamentos (*routers, firewalls, ...*) que, por sua vez, bloqueiam a ligação.

Também é importante analisar se a XSG permite configurar as mensagens de erro que serão retornadas para o cliente em casos de bloqueio das ligações.

Outro parâmetro importante é verificar se é possível desactivar o bloqueio parcial de determinadas mensagens. Ou seja, numa situação em que queremos desactivar o bloqueio parcial, não necessitamos de desactivar completamente esta funcionalidade.

8.8 Standards de segurança suportados

Um dos critérios mais importantes na avaliação de uma XSG é a lista de *standards* de segurança suportado e respectivas versões: WS-Security, SAML, WS-Trust, WS-ReliableMessaging, etc.. Também é importante verificar se a *gateway* suporta implementações de acordo com as normas definidas pelo WS-I.

É necessário ter em conta que é preciso verificar se o *standard* de segurança é suportado na sua totalidade pois, podem existir casos em que nas especificações é dito que suporta WS-Security mas, não suporta todos os tipos de *tokens* de segurança listados nesse *standard*.

8.9 Formatos de documentos suportados

A *gateway* pode suportar o processamento de dados XML em vários formatos:

- “*Data Streams*” – Dados XML sequenciais.
- “*Document Object Module*” (DOM) – Dados estruturados em “árvores” e “nós”.
- “*Simple API for XML*” (SAX) – Formato orientado aos eventos.
- (Etc.).

8.10 Controlo de acessos ao WSDL

Possibilidade de publicar partes específicas do WSDL, restringindo desta forma o leque de potenciais tentativas de ataque através da exploração da informação disponibilizada.

Além disso, pode ser possível aplicar regras de controlo de acesso a qualquer tipo de mensagens relacionadas com o WSDL. As permissões são aplicadas com base em

privilégios (ex.: papéis, responsabilidades e grupos) definidos pelos administradores dos WS. Ou seja, só os utilizadores autenticados é que podem aceder a esses dados.

8.11 Armazenamento das chaves usadas nas tarefas criptográficas

As tarefas criptográficas envolvem o uso de chaves privadas que deverão ser devidamente armazenadas de forma a estarem protegidas de possíveis “roubos”. Quando as chaves são guardadas nos próprios servidores aplicativos, a sua cópia ou eliminação tornam-se relativamente fáceis. Quando é utilizada uma XSG que efectua algumas dessas tarefas criptográficas, esta fica responsável pela gestão e armazenamento dessas chaves (este armazenamento pode ser externo). No cenário em que se opta por utilizar a XSG, com alguma facilidade, é também possível registar todos os acessos e utilizações dessas chaves.

8.12 Gestão da própria XSG

a) Tipo de aplicação usada na sua gestão

Tipos de ferramentas (gráficas ou não) para a gestão da XSG. Facilidade de utilização e simplicidade da interface gráfica disponível.

b) Cópias de segurança da sua configuração

c) Controlo de acessos às suas configurações

d) Gestão das versões das configurações

e) Registo das alterações de configurações

Registos relativos a quem é que alterou a configuração, a que hora e que alterações foram efectuadas.

f) Ferramentas de controlo operacional da própria XSG

Possibilidade de serem efectuadas medições relativas à performance da XSG, envio de alertas (por correio electrónico, SMS, *syslog*, *SNMP*, etc.), ferramentas de diagnóstico de problemas, etc..

g) Métodos utilizados para actualizações de versões e correcções

8.13 Gestão/Monitorização dos WS com base em SLAs

Possibilidade da própria XSG incluir funcionalidades para gerir os WS com base nos SLAs – níveis de disponibilidade e *performance* acordados previamente.

8.14 Integração da XSG com outros equipamentos/ferramentas

Possibilidade de integração da XSG com ferramentas de testes de segurança, de desenvolvimento de código para os WS, ferramentas de IAM (*Identity and Access Management*), SOA Governance, ESB – *Enterprise Service Bus*, UDDI registry/repository, etc..

8.15 Suporte pós-venda

Um dos critérios mais importantes relativamente à escolha da XSG está relacionado com o grau de suporte pós-venda que é oferecido. Ou seja, se têm representantes locais, se é possível assinar contratos de suporte de *hardware* que incluem troca dos equipamentos com deslocação de técnicos especializados ao local, quais os tempos de resposta/reparação oferecidos, se o suporte a nível de *software* pode ser efectuado directamente com os técnicos da empresa ou tem de ser através dos fornecedores locais, etc..

8.16 Investimento necessário

Um dos factores a ter em conta na escolha da XSG é o investimento necessário para conseguir implementar uma solução com as redundâncias necessárias e em que estejam incluídos os sistemas de gestão/monitorização das próprias *gateways* (sistemas de gestão de políticas/configurações, servidores onde serão armazenados todos os registos efectuados, etc.).

9 Processo de avaliação da XML Security Gateway a ser adoptada pelo Banco BPI

Tendo como base os critérios de avaliação das XML Security Gateways listados no ponto anterior e, devido ao facto do número de soluções que neste momento estão disponíveis no mercado ser elevado, a análise das várias opções de implementação de XSG efectuada pelo Banco BPI foi feita em várias fases:

- Levantamento das soluções existentes no mercado.
- Elaboração do RFI (*Request for Information*) e análise das respostas dos fornecedores.
- Elaboração do RFP (*Request for Proposal*) e análise das respostas dos fornecedores.
- Elaboração do Caderno de Testes das XSG e execução de Pilotos de Testes.
- Escolha da XML Security Gateway.

9.1 Levantamento das soluções existentes no mercado

Nesta fase, foram consultados vários documentos relativos a SOA disponibilizados por empresas de consultoria, assim como alguma documentação de produtos actualmente disponíveis no mercado que implementam as funcionalidades das XSG. A partir destas consultas, foi criada uma lista inicial de fornecedores que têm produtos relacionados com as tarefas de segurança/optimização do tráfego relacionado com arquitecturas SOA: IBM, Forum Systems, Vordel, Cisco, Layer7 Technologies, F5, Oracle, CA, Amberpoint, SOA Software, Intel, Bloombase Technologies, Dajeil, Radware, Progress, Cast Iron, Solace Systems, Citrix, Xlipstream, Tarari, Alcatel-Lucent, etc..

Após estas consultas, foram seleccionados seis fornecedores, tendo em conta os seguintes requisitos básicos definidos pelo Banco BPI:

- Serem *appliances* baseadas em *hardware* optimizado para este tipo de tarefas e cujo Sistema Operativo já esteja preparado para estas funções (ex.: por questões de segurança do próprio sistema, algumas das funcionalidades do seu Sistema Operativo estão desactivadas). Ou seja, tendo como base os critérios de avaliação definidos no ponto “8 – Critérios de avaliação das XML Security Gateways”:
- Modo de operação diferente de “*plug-in*”. – Um dos requisitos do Banco BPI relativamente à escolha do *hardware/software* que venha a implementar segurança nos WS é esse componente ser externo aos próprios servidores onde estão implementados os WS. Este requisito deve-se ao facto de se querer passar a carga de processamento das tarefas de validação do XML para fora dos próprios servidores e à necessidade de implementar uma arquitectura escalável em que seja possível adicionar novos servidores aplicativos sem ser necessário um grande investimento em licenças deste tipo de produtos. Além disso, numa solução em que o controlo dos acessos é efectuada num ponto central, as tarefas operacionais relacionadas com alterações das configurações e actualizações serão menores (o número de pontos onde essa configuração/actualização terá de ser aplicada são menores). Portanto, de forma a cumprir estes requisitos, não deverão ser adoptados produtos baseados na instalação de agentes nos servidores aplicativos. Ou seja, para que as

tarefas de validação sejam efectuadas, deverão ser implementadas em equipamento dedicado para tal. Com a introdução das XSG na infra-estrutura de rede serão cumpridos estes requisitos básicos da solução.

- Implementação física diferente de “só *software*”. – *Hardware* que permite aceleração das tarefas de processamento do XML e operações relacionadas com encriptação e SSL/TLS.
- Suportar alguns dos padrões relacionados com os *Web Services*.
- O fornecedor assegurar a sua representação e suporte em Portugal.

9.2 Elaboração do RFI (*Request for Information*) e análise das respostas dos fornecedores

Após a triagem de produtos efectuada na fase anterior, o Banco BPI elaborou o documento RFI (*Request for Information*) que foi enviado para os fornecedores seleccionados na fase anterior. A elaboração do RFI teve como base alguns dos pontos listados no capítulo “8 – Critérios de avaliação das XML *Security Gateways*” deste relatório e alguns dos requisitos considerados pelo Banco BPI como relevantes para a escolha do produto. No “ANEXO B: RFI (*Request for Information*) elaborado pelo Banco BPI” encontra-se o conteúdo do respectivo RFI.

Após a análise das respostas dos fornecedores relativas ao RFI e de algumas apresentações efectuadas ao Banco BPI, foram seleccionados apenas cinco fornecedores.

Como um dos objectivos do RFI era conseguir comparar a lista de requisitos do Banco BPI com a lista de funcionalidades disponibilizadas pelos fornecedores deste tipo de equipamentos, foi necessário reajustar a lista de requisitos do Banco BPI. Este reajustamento foi devido a existirem alguns que não estavam implementados por nenhum dos fornecedores ou a sua implementação ser independente da XSG escolhida. Nessa lista de funcionalidades estão incluídas:

- Integração com *reverse proxy* HTTP/HTTPS – a existência de um *reverse proxy* HTTP/HTTPS é “transparente” para a XSG.
- Suporte de WS-MetadataExchange, WS-AtomicTransaction, WS-Coordination, BPML, WS-BPML, WS-CDL.
- Actualização automática das assinaturas relativas a IDS/IPS.
- O facto de *clusters* de XSG conseguirem distribuir, entre elas, o seu próprio tráfego sem precisarem de recorrer a balanceadores externos.

Por outro lado, foi também necessário eleger alguns dos requisitos como fundamentais:

- Suporte de WS-Security (inclui assinatura e encriptação das mensagens ao nível dos campos XML), WS-SecureConversation, WS-Trust, WS-Federation, XACML, WS-Policy, WS-PolicyAttachment, WS-ReliableMessaging, WS-Addressing, WS-SecureExchange.
- Suporte e aceleração das tarefas relacionadas com: terminação SSL, encriptação, validação de esquemas XML, validação da formatação do XML, operações de transformação XML.
- Detecção/Protecção ataques direccionados a tráfego XML (XML IDS/IPS).

- Validação do conteúdo das mensagens.
- Encaminhamento dos pedidos com base no seu conteúdo, SLAs, carga dos próprios WS, etc.
- Integração com serviços de directório, nomeadamente por LDAP.
- Registo dos acessos aos WS e das alterações efectuadas às configurações das XSG.
- Integração com algumas ferramentas específicas de SOA *Governance*, UDDI *registry/repository* e gestão de identidades/autorizações e os custos de integração associados (podem existir situações em que esta integração demora bastante tempo a ser concluída). Como no Banco BPI estão a ser implementadas outras soluções que beneficiarão da existência dessas ferramentas, de forma a otimizar os recursos e o investimento, essas ferramentas têm de ser partilhadas pelas diversas soluções.
- Gestão centralizada das configurações, análise dos logs e monitorização.
- Garantia da disponibilidade da XSG (métodos de balanceamento e tolerância a falhas das próprias XSG).

9.3 Elaboração do RFP (*Request for Proposal*) e análise das respostas dos fornecedores

Após a consulta efectuada na fase anterior, foi elaborado um outro documento: RFP – *Request for Proposal*. O objectivo deste documento foi solicitar, através do Departamento de Compras do Banco BPI, a proposta comercial para a aquisição e implementação das XML *Security Gateways* apresentadas por cada um dos fornecedores, com os respectivos custos associados e as condições contratuais impostas pela instituição (suporte pós-venda, formação, política de actualizações, etc.). Neste documento foram descritos:

- Os objectivos esperados pelo Banco BPI com a aquisição das XSG.
- As especificações técnicas de forma a ser possível quantificar as necessidades relativas ao número/tipo de equipamentos necessários.
- A infra-estrutura base onde deverão ser implementados estes novos elementos da arquitectura.
- Os requisitos em termos de redundância, recuperação em caso de desastre e escalabilidade.
- A lista dos requisitos técnicos, divididos nas mesmas categorias apresentadas no RFI: preferenciais, desejáveis e acessórios (ANEXO C: Lista de requisitos constantes no RFP (*Request for Proposal*) elaborado pelo Banco BPI).

Após a recepção das respostas dos fornecedores aos RFP, serão seleccionados os três fornecedores que apresentarem as melhores propostas técnicas/financeiras e que estejam de acordo com as expectativas do Banco BPI. Esses fornecedores serão convidados a participar em Pilotos de Teste de forma a ser possível comprovar que os respectivos produtos respondem aos requisitos apresentados pelo Banco BPI.

9.4 Elaboração do Caderno de Testes das XSG e execução de Pilotos de Testes

Para avaliar as XSG em ambientes reais, foi criado um caderno de testes a serem executados durante os pilotos que terão a duração de duas semanas. Nesses testes serão

criados alguns cenários para simular as várias possibilidades de implementação de autenticação/autorização, encriptação e assinatura das mensagens, geração de ataques de segurança e verificação do comportamento das XSG, forçar a indisponibilidade de algumas XSG e verificar o comportamento da restante arquitectura, etc..

Em seguida, estão listadas as funcionalidades e cenários a avaliar. Além da avaliação das funcionalidades, é importante também avaliar as soluções em termos de *performance*, escalabilidade e facilidade de configuração. Relativamente aos dois primeiros pontos, poderão ser efectuados alguns testes de carga utilizando um subconjunto dos cenários criados.

Autenticação e Autorização

Para testar as funcionalidades relacionadas com a autenticação e autorização deverão ser avaliados os seguintes cenários:

- Suporte genérico dos *standards* WS-Security, WS-SecureConversation, WS-Trust e WS-Federation;
- Utilização de WS-Security com o *token kerberos* na invocação de um serviço através da consulta de uma *Active Directory* ou outro directório LDAP.
- Utilização de WS-Security, WS-Trust e WS-Federation com o *token SAML* na invocação de um serviço (STS – *Security Token Service*).
- Utilização de WS-Security com o *token username* na invocação de um serviço (consulta de uma base de dados proprietária cujo acesso está ele próprio disponibilizado na forma de um *Web Service*).
- Utilização de WS-Security com o *token X.509* na invocação de um serviço (consulta de uma *Certificate Authority*).
- Integração da *XML Security Gateway* com um sistema de *Directory Services* (*Active Directory* ou outro directório LDAP).
- Suporte do *standard* XACML (WS-Security + SAML + XACML) para especificação de atributos de acesso.
- Estabelecimento de configurações de federação com parceiros.
- Integração com outros sistemas de identidade: ADFS (*Active Directory Federation Services*), sistemas SSO – *SSO Session Cookies*.

Gestão de políticas de segurança

Para testar a gestão de políticas de segurança será utilizado um repositório UDDI para registo das políticas de segurança configuradas através da XSG. Os objectivos dos cenários a avaliar serão os seguintes:

- Facilidade de gestão de políticas de segurança e respectivo registo num repositório UDDI. Para tal, devem ser alteradas políticas de segurança de um serviço *kerberos*, passando o processo de autenticação para *username*, X.509 e SAML.
- Suporte dos protocolos WS-Policy, WS-PolicyAttachment e WS-MetadataExchange para registo e obtenção das políticas de segurança.

Assinatura e Encriptação

De forma a avaliar alguns cenários que permitam garantir a integridade e confidencialidade dos dados, deverá ser criada uma infra-estrutura que permita que os consumidores dos WS acedam por HTTPS (segurança ao nível do transporte – HTTPS, PKI), seja usado o WS-Security (segurança ao nível da mensagem) e sejam assinados e encriptados dados específicos nas mensagens XML. Portanto, os objectivos deste cenário são avaliar a XSG relativamente a:

- Capacidade de terminar túneis SSL e prosseguir as comunicações em HTTP até à camada dos *Web Services*.
- Possibilidade de assinar e/ou encriptar campos específicos na mensagem XML (avaliar também as configurações necessárias para assegurar o processamento dos campos assinados e/ou encriptados no lado dos servidores aplicacionais).
- Capacidade de verificação de assinaturas.
- Capacidade de assegurar a integração com uma implementação PKI interna (descodificação dos pacotes e posterior encriptação).
- Capacidade de estabelecer VPNs com parceiros.

Validações das mensagens XML

Um dos principais objectivos das XSG implementadas em *hardware* optimizado para este tipo de tarefas é permitir transferir algumas das funções, que consomem bastantes recursos, dos servidores aplicacionais para estes equipamentos. Portanto, alguns dos cenários a criar nos pilotos deverão permitir verificar se as XSG executam algumas dessas funções:

- Validação de esquemas XML de acordo com a definição do serviço obtida de um repositório de serviços interno ou externo. – Para efectuar este teste pode ser efectuada uma chamada HTTP POST com um esquema XML inválido e confirmar se é retornada uma mensagem de erro.
- Validação da formatação do XML. – Criação de mensagens XML com *tags* não fechadas.
- Validação dos esquemas XML e formatação tanto para mensagens *inbound* como *outbound* (possibilidade de usar a XSG para controlar também acessos dos nossos servidores a WS externos).

Filtragem das mensagens

Possibilidade de filtrar os pedidos em função:

- Do IP de origem.
- Do conteúdo da mensagem (ex.: um pedido em que o valor do campo país é diferente de “Portugal” não deverá ser autorizado).

Protecção contra ataques XML

Recorrendo a ferramentas específicas e criação de mensagens SOAP que representem tentativas de ataques XML (ver “ANEXO D: Ferramentas disponíveis para testar a forma como as XSG detectam algumas situações de ataques”), simular algumas situações de ataque e verificar o comportamento da XSG perante:

- Ataques XML DoS (elementos recursivos, “*oversized payload*”, etc.). Exemplos:
 - Simular ataques “Bombas XML” e verificar se o processamento do XML é recusado e que tipo de retorno de erro é gerado para o utilizador.
 - Nível de protecção contra ataques DoS. – Estes ataques podem ser simulados através da geração de muitos pedidos em simultâneo. No entanto, a tendência é que a geração de carga seja efectuada com base sempre na mesma mensagem e originada pelo mesmo sistema. Como a maior parte das *gateways* de segurança já estão preparadas para proteger os sistemas destes ataques DoS básicos, as mensagens dos pedidos dirigidos aos WS deverão ser geradas com valores dinâmicos, semanticamente válidos e que consigam abranger o maior número de pedidos possíveis. Também deverão ser simulados pedidos com origem em endereços IP diferentes.
- Tentativas de acessos não autorizados (“*brute force*”, “*replay attack*”, etc.). Exemplo:

Protecção contra “*replay attacks*”. – Estes ataques podem ser simulados enviando múltiplos pedidos com o mesmo identificador da mensagem. Por exemplo, se estiverem a ser usados *tokens* do tipo *username*, o teste pode consistir em enviar várias mensagens com os mesmos valores e verificar qual a mensagem retornada. Ou seja, verificar se este ataque é ou não detectado. De forma a protegerem-se contra este tipo de ataques, o próprio serviço, ou a XSG, devem ter em *cache* os valores mais recentes. Algumas das implementações WS-Security não têm em consideração estes factores, ficando vulneráveis a este tipo de ataques.
- Ataques à integridade/confidencialidade das mensagens (“*parameter tampering*”, “*SQL Injection*”, “*Xpath Injection*”, “*XML Injection*”, etc.).
- Ataques à integridade dos sistemas (ex.: vírus XML, etc.).
- Se a XSG suportar a publicação de WSDL, simular ataques “*WSDL scanning*”. Exemplo:
 - Testar as interfaces WSDL tentando todas as combinações de parâmetros de entrada e verificar se ocorre alguma violação a nível de segurança do WS.
 - Verificação dos mecanismos de segurança dos WSDL. – As vulnerabilidades de acesso aos WSDL podem ser detectadas através de tentativas de obtenção do WSDL sem utilizar o canal de segurança definido (ex.: se o WSDL estiver protegido por SSL na porta TCP 443, não deve ser possível aceder na porta normal do HTTP (TCP 80)). Também é necessário verificar quais são as mensagens de erro geradas quando a *gateway* e o WS estão perante testes dirigidos aos WSDL que tentam explorar vulnerabilidades relacionadas com a definição de esquemas e mensagens não usadas.
- “*XML port scanning*” pode ser usado para verificar se a XSG está configurada correctamente. Esta técnica utiliza um *parser* XML para executar o “*port scanning*” dos sistemas que estão por trás da *gateway*. Através dos resultados obtidos pode ser divulgada informação detalhada sobre os serviços disponíveis. Se a *gateway* estiver

configurada correctamente, a informação disponibilizada deve ser só a estritamente necessária para o bom funcionamento dos WS.

- Testar se, em situações de troca da ordem das mensagens enviadas, existe algum mecanismo de reordenação para que o resultado a retornar pelo WS seja igual ao que seria obtido numa situação normal.

Após a detecção destes ataques, ou em situações em que é atingido o limite da capacidade de processamento, é importante observar nas *gateways*:

- Quais as acções, alertas e mensagens de erro retornadas. – Por exemplo, pode ser definido um alerta para as situações em que é atingido um determinado valor limite e, em seguida, ao atingir outro valor limite, é bloqueado o acesso a determinado IP de origem. Nesse exemplo, pode ser verificado o comportamento da *gateway* nos casos em que o IP de origem não é sempre o mesmo.
- Como é que se pode garantir que só a *gateway* é afectada, não se verificando a indisponibilidade do resto da arquitectura.

Como a maior parte dos fornecedores disponibilizam listas de ataques que podem ser detectados pelas suas *gateways*, se tivermos constrangimentos de tempo durante as duas semanas de piloto, poderemos dispensar as demonstrações reais destes tipos de ataques XML e confiar nas listas previamente disponibilizadas.

Além dos testes anteriores, também deverá ser avaliada:

- A forma como são efectuadas as actualizações das assinaturas dos ataques.
- Periodicidade de disponibilização das novas assinaturas.
- Possibilidade de definir limite máximo do tamanho da mensagem.
- Possibilidade de definir o número máximo de ligações por segundo, ligações simultâneas, etc..

Registo das actividades

Como é fundamental o registo de todas actividades relacionadas com a invocação dos serviços e com as próprias alterações efectuadas aos sistemas/políticas, em cada um dos cenários deverão ser analisados os registos efectuados durante o piloto. Concretamente, esta análise tenta avaliar:

- A possibilidade de auditoria de todas as componentes da solução e quais os dados registados.
- A nível da definição de grupos com papéis diferentes na administração/operação da própria XSG, avaliar a possibilidade de integração com os sistemas de directório existentes já na empresa (*Active Directory*, TACACS, RADIUS, etc.).
- Possibilidade de disponibilizar os registos num servidor central.
- Possibilidade de exportar automaticamente, para base de dados, os registos relativos ao tráfego destinado aos WS. Dar exemplos de tipos de bases de dados suportadas e algumas ferramentas de análise desses dados que já estão disponíveis.
- Integração com soluções de recolha de registos/eventos já existentes no Banco, nomeadamente CS-MARS, MOM, etc..

Virtualização dos serviços: transformações e reencaminhamento das mensagens XML

De forma a verificar se a XSG permite virtualizar os serviços, deverão ser criados cenários que permitam validar as seguintes funcionalidades:

- Transformação da mensagem XML enviada pela aplicação cliente de forma a coincidir com a expectável pelo *Web Service*.
- Manipulação de campos específicos da mensagem SOAP (ex.: alterar formato da data, alterar formato do número de conta, validar campos do tipo *string* para existência de formatos XML inadequados, etc.).
- Quais as transformações de protocolos possíveis.
- Reencaminhamento da mensagem SOAP, para servidores diferentes, mediante um parâmetro da mensagem.
- Suporte dos *standards* WS-SecureConversation, WS-ReliableMessaging, WS-Addressing, WS-SecureExchange.

Transacções

Em determinadas situações, alguns dos serviços poderão apresentar comportamentos transaccionais, requerendo estar no contexto de uma transacção distribuída quando são invocados. Para suportar estes cenários é necessário assegurar o suporte dos *standards* WS-AtomicTransaction e WS-Coordination para que o contexto transaccional flua até ao serviço.

Composição de serviços (“orquestração e coreografia”)

Alguns processos de negócio baseiam-se na conjugação de serviços que pode ser predefinida (“orquestração”) ou articulada em *runtime* (“coreografia”). Alguns produtos permitem gerir a composição de serviços constringindo a sua execução a processos predefinidos, ou simplesmente alertando para padrões de execução suspeitos. Para tal, deverá ser avaliada a capacidade da XSG em suportar os *standards* BPEL, WS-BPEL e WS-CDL (se a XSG não tiver um papel “activo” nestes processos, pelo menos deverá processar as mensagens sem alterar o conteúdo relativo à aplicação destes *standards*).

SLAs e monitorização

- Avaliar a capacidade de estabelecer SLAs, monitorizar a invocação de serviços e definição de alertas relacionados com o estado dos serviços. Para tal, definir um SLA de tempo de execução e configurar um alerta a ser originado no momento em que se verificar o não cumprimento do SLA.
- Envio de alertas relativos à operacionalidade da própria *gateway* (SNMP, SMTP, *Syslog*)
- Demonstração dos relatórios e gráficos disponibilizados pelo sistema de gestão da XML *Security Gateway* relativos à monitorização do estado dos serviços e da própria *gateway*.

Integração com soluções externas

Ao longo dos cenários testados, deverá ser avaliada a capacidade da XSG ser integrada com soluções:

- SOA Governance
- UDDI *Registry/Repository*
- ESB
- Sistemas de gestão de identidades e de controlo de acessos

Para tal, deverão ser solicitadas aos fornecedores pequenas demonstrações desta integração.

Redundância e tolerância a falhas

De forma a avaliar a possibilidade de garantir a redundância e tolerância a falhas da infra-estrutura SOA, deverão ser avaliados:

- Tipos de operação das XSG (activo/activo, activo/passivo).
- Suporte de balanceamento das XSG:
 - Suporte de *clusters* de XSG.
 - Sendo definido um *cluster* de XSG, elas são capazes de distribuir o seu próprio tráfego ou precisam de um balanceador externo? No primeiro caso, que algoritmos de distribuição do tráfego estão disponíveis?
 - As XSG partilham a tabela de estados (*stateful* e *stateless*)?
- Tolerância a falhas:
 - Em que situações é que uma XSG muda o seu estado para inoperacional.
 - Se a XSG estiver em *cluster*, em que situações é que consideram as suas “parceiras” como estando numa situação de inoperacionalidade?
 - Se uma XSG falhar, as outras podem assumir as ligações dessa XSG (*stateful* e *stateless*)?
- Capacidade de virtualização da própria XSG (criação de vários contextos de *gateways* com diferentes configurações/políticas aplicadas).
- Método de balanceamento dos WS. – A XSG tem a capacidade de distribuir o tráfego pelos diversos servidores (*round-robin*, em função da carga de cada servidor, etc.) ou é necessário recorrer a um sistema de balanceamento externo?
- Sistema de gestão das XSG (inclui sistema a partir do qual são configuradas as *gateways* e sistema de recolha de registos/eventos):
 - Possibilidade de redundância (instalação em *cluster*, possibilidade de existirem dois sistemas de gestão redundantes que consigam gerir a mesma *gateway*, etc.).
 - Possibilidade de instalação em máquinas virtuais.
 - Gestão automática dos registos efectuados (rotação, criação automática de cópias de segurança, etc.).
 - Gestão das configurações: comparação de versões, criação automática de cópias de segurança, possibilidade de repor configurações anteriores, etc.).

- Possibilidade de gerir mais do que um grupo de *gateway*, em cidades diferentes.
- Como o Banco BPI tem dois centros de dados principais, em diferentes cidades, e ambos estão simultaneamente em produção, é necessário gerir os *Web Services* em ambos os centros. Qual a infra-estrutura proposta de forma a garantir o melhor compromisso entre custo e garantia dos níveis de serviço?

Aceleração das operações

Como já foi dito anteriormente, um dos parâmetros principais a ter em conta na avaliação da XSG é a sua capacidade de efectuar algumas operações de validação das mensagens XML, terminação de túneis SSL, etc.. Portanto, como normalmente os fornecedores destes equipamentos já têm os valores de performance obtidos em determinados cenários, deverão ser solicitados os valores máximos de performance que poderão ser obtidos nas seguintes situações:

- Terminação SSL.
- Tarefas de encriptação.
- Validação de esquemas XML.
- Validação da formatação do XML.
- Operações de transformação XML.

Caso sejam efectuadas alguns testes de carga durante a fase de piloto, é necessário ter em atenção que não deverão ser utilizadas sempre as mesmas mensagens, mas sim mensagens dinâmicas:

- Algumas *gateways* estão configuradas para considerar que múltiplas mensagens idênticas sejam identificadas como situações de ataques DoS.
- Se forem utilizadas mensagens em que a autenticação também é baseada no *timestamp*, é importante que este valor seja sempre diferente.
- Se forem efectuados testes de encriptação/assinatura, os valores de *hash* e assinaturas utilizadas não deverão ser sempre os mesmos.
- Algumas *gateways* disponibilizam funcionalidades de *cache* e, em alguns testes, os valores retornados poderão ter origem nessa *cache*.

Além dos cuidados anteriores a ter em conta na geração de testes de carga, é importante também tentar criar situações tão próximas da realidade quanto possível, através da geração de mensagens que representem situações reais na empresa.

Os valores medidos poderão ser:

- *Throughput* em Mbps, número máximo de novas ligações por minuto, número máximo de pedidos por segundo, número máximo de ligações simultâneas, latência dos pedidos, etc.. Estas medições devem ser efectuadas para mensagens de diferentes tamanhos, diferentes tipos, uso de SSL a terminar na *gateway*, configurações diferentes da *gateway* (configurar transformações XML, determinadas validações mais complexas, ...), etc..
- Ao adicionar mais uma XSG, que benefícios em termos de *performance* é que são obtidos? Muitas vezes, mais vale adquirir um equipamento de outro modelo do que

adicionar mais elementos ao *cluster* pois, seguindo esta segunda opção, não se verificam ganhos de *performance*.

- Em situações de grande tráfego, é possível continuar a gerir a XSG?

Gestão da XSG

- Disponibilização de interface gráfica de fácil utilização.
- Possibilidade de aceder remotamente à interface gráfica através de HTTPS ou protocolo proprietário seguro.
- Possibilidade de aceder remotamente à *gateway* através de SSH. Se não for possível aceder por SSH, em alternativa, aceder através de protocolos “mais fracos” em termos de segurança: HTTP ou Telnet. Também é importante verificar se é possível desactivar esses acessos remotos ou restringir o acesso a determinados endereços IP.
- Possibilidade de aceder localmente à *gateway* através de porta de consola ou ecrã/teclado.
- Possibilidade de utilizar uma *interface* de rede dedicada só à gestão (diferente das *interfaces* de rede usadas para tráfego de “produção”).

9.5 Escolha da XML Security Gateway

Para esta fase foi criada uma tabela de classificação. Essa tabela consiste numa folha de cálculo em que é atribuído um peso relativo a cada um dos cenários listados no ponto anterior (é semelhante à folha de cálculo usada na análise das respostas aos RFI e RFP). Em cada um dos cenários de testes será avaliado um dos requisitos do RFP.

Seguindo as três categorias de classificação dos requisitos (preferencial, desejável e acessório) foram atribuídos diferentes pesos a cada uma delas:

$$K_{\text{preferenciais}} = 5 - \text{constante da categoria “preferenciais”}$$

$$K_{\text{desejáveis}} = 2 - \text{constante da categoria “desejáveis”}$$

$$K_{\text{acessórios}} = 1 - \text{constante da categoria “acessórios”}$$

Dentro de cada uma das categorias de classificação foram atribuídos pesos diferentes a cada um dos requisitos (p – peso do requisito).

Cada um dos requisitos será avaliado numa escala de 0 a 5 (v – avaliação do requisito):

- 0 – Não cumpre o requisito.
- 1 – Cumpre uma pequena parte do requisito (< 50% do esperado).
- 2 – Cumpre parte do requisito (> 50% e < 75% do esperado).
- 3 – Cumpre grande parte do requisito (> 75% e < 100% do esperado).
- 4 – Cumpre totalmente o requisito.
- 5 – Cumpre o requisito e apresenta funcionalidades além das que estávamos à espera).

De forma a ser possível calcular o valor da classificação final da avaliação da solução apresentada por cada um dos fornecedores, é necessário calcular valor “parciais” de cada

uma das três categorias de classificação dos requisitos. Para tal, para cada uma das categorias (x), serão calculados:

$$A(x) = K_x \cdot \sum_{i=1}^n v_i \cdot p_i \qquad B(x) = K_x \cdot \sum_{i=1}^n p_i$$

K_x - constante da categoria x ; n - nº de critérios da categoria; v - avaliação do requisito; p - peso do requisito

Calculados estes valores “parciais” para cada uma das três categorias de critérios, será possível obter a classificação final dessa solução (C) através da seguinte fórmula:

$$C = \frac{A(\text{preferenciais}) + A(\text{desejáveis}) + A(\text{acessórios})}{\max(v) \cdot (B(\text{preferenciais}) + B(\text{desejáveis}) + B(\text{acessórios}))}$$

Além dos parâmetros avaliados durante o piloto, também terão algum peso na escolha final: o custo, suporte pós-venda (tempo de resposta em caso de falha de *hardware*), capacidade de esclarecimento de dúvidas relativas à configuração, resolução de problemas de *software* encontrados, etc.. Nesta avaliação estão incluídos os serviços prestados tanto pelo fornecedor como, caso se aplique, pelo próprio representante local (podem existir casos em que o fornecedor não disponibiliza a possibilidade de obtermos suporte directo relacionado com *software* ou esclarecimento de dúvidas relativas a possíveis cenários de implementação).

10 Proposta de implementação da infra-estrutura de segurança no ambiente SOA do Banco BPI

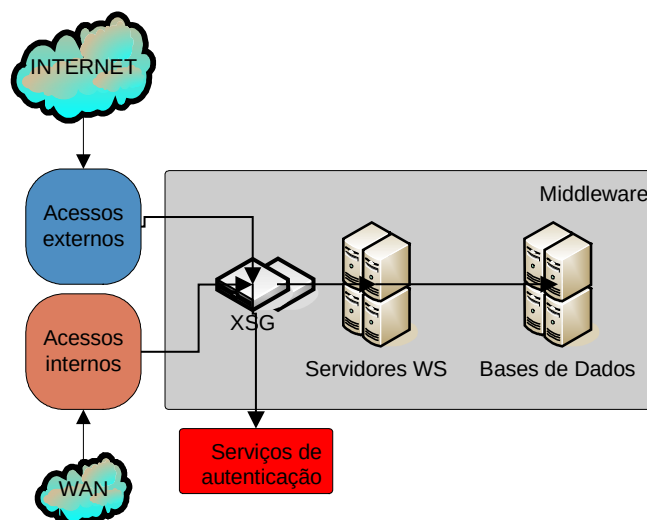


Figura 10.1: Proposta da infra-estrutura de segurança no ambiente SOA do Banco BPI

Na Figura 10.1 está representado o esquema simplificado da proposta da infra-estrutura de segurança onde estarão alojados os *Web Services* do Banco BPI. Esta proposta tem como base nos seguintes tipos de equipamentos:

- *Firewalls* tradicionais que isolam as diferentes zonas do *datacenter*. Nestas *firewalls* serão efectuadas filtragens a nível de IPs e portas.
- Sistemas de detecção/protecção de intrusões (IDS/IPS).
- *XML Security Gateways*, implementadas em *hardware* dedicado e optimizado para o processamento das mensagens XML. Desta forma, parte do processamento relacionado com o SSL e validação XML que actualmente é efectuado a nível de cada um dos servidores, poderá ser efectuado pela *gateway*.
- Equipamentos de rede que permitam garantirem a confiança nas comunicações entre os diferentes intervenientes: *routers*, *switches* e balanceadores de carga.
- Servidores onde estão alojados os WS. – Com a passagem de algumas das operações anteriormente processadas por cada um dos servidores para a XSG, será possível diminuir o número de servidores dedicados a cada WS e consolidar vários serviços num mesmo servidor. Aproveitando esta oportunidade de revisão da arquitectura, também se poderão implementar parte destes WS num ambiente de virtualização dos servidores. Desta forma, serão reduzidos os custos associados à manutenção do parque de servidores (aquisição e contratos de suporte de *hardware*), gastos de energia, ocupação de espaço, portas disponíveis nos equipamentos de rede e necessidade de ter equipas em número suficiente para instalar/manter esses servidores).
- Bases de dados de utilizadores usadas nas tarefas de autenticação/autorização.
- Zona de gestão/monitorização de toda a infra-estrutura (não está representada na Figura 10.1). Só a partir desta zona é que deverão ser efectuadas as tarefas de gestão/monitorização. Por exemplo, os sistemas de gestão das XSG e os servidores

que terão toda a base de dados de registos de acessos aos WS deverão estar nessa zona. Todo o tráfego de gestão da *gateway* deverá ser efectuado através de uma *interface* de rede desse equipamento que estará ligada na infra-estrutura de gestão.

Os equipamentos envolvidos não deverão ser todos do mesmo fabricante e, em alguns casos, as regras de implementação deverão seguir “filosofias” diferentes (ex.: nas regras de *firewalls*, umas poderão ser configuradas para só deixar passar o que estiver listado e negar por defeito, outras poderão listar o que devem negar e deixar passar tudo que não foi listado anteriormente).

Por questões de garantia da continuidade de negócio, em caso de indisponibilidade e, em alguns casos, por questões legais, o Banco BPI tem vários *datacenters*, em diferentes cidades. Sempre que se aplique, a infra-estrutura proposta deverá ser seguida nos vários locais. Algumas das questões relacionadas com balanceamento de WS em locais remotos que foram apresentadas aos fornecedores das XSG estavam relacionadas com a necessidade de manter os vários *datacenters* sempre activos e, em caso de falha, assumirem algumas das funcionalidades dos outros. Além da redundância dos locais, na maior parte dos casos, dentro do mesmo *datacenter*, os equipamentos deverão ser redundantes, tanto para garantir a disponibilidades dos sistemas em caso de falha de *hardware* como para manter alguns valores *de performance*.

Para implementar uma infra-estrutura de segurança no ambiente SOA do Banco BPI, além da utilização dos equipamentos descritos anteriormente, durante a sua configuração deverão ser seguidas as recomendações enumeradas no ponto “7 – Boas práticas de segurança a seguir na implementação de uma Arquitectura Orientada a Serviços”. Algumas das medidas a implementar no Banco BPI deverão ser as seguintes:

- Configurar a XSG de forma a validar todas as mensagens XML (*inbound* e *outbound*) relativamente à sua formatação XML, validar esquemas XML, detectar ataques, detectar vírus nas mensagens/anexos e efectuar alguma filtragem relativa ao conteúdo da mensagem XML. Os WS devem ser implementados de forma a reduzir ao mínimo a referência a recursos disponibilizados por entidades externas.

Por exemplo, validando os esquemas XML antes do processamento da mensagem pelo WS e bloqueando as referências externas existentes nos documentos XML e pedidos SOAP, é possível prevenir que, ataques do tipo “*schema poisoning*” efectuados às entidades externas, afectem estes WS. Também é importante definir, nos próprios esquemas XML, limites mínimos e máximos dos tamanhos das cadeias de caracteres, definir limites máximos nos tamanhos dos documentos (prevenção de ataques “*oversized payload*”), etc..

- Nas *firewalls*, deverá ser efectuada a tradução dos endereços IP públicos atribuídos aos WS virtualizados na *gateway* e que serão acedidos a partir do exterior. Desta forma, não será disponibilizada a informação relativa ao endereçamento interno e outros parâmetros usados internamente na identificação dos serviços. Sempre que se justificar, as mensagens de erro retornadas para os clientes poderão ser modificadas pela *gateway* de forma a não disponibilizar, para o exterior, informação que poderá divulgar alguns dados sobre a infra-estrutura dos WS.
- Enquanto o número de clientes externos dos WS for considerado “reduzido”, não será implementado um sistema UDDI *Registry/Repository* para o exterior. Os dados relativos aos WSDL poderão ser enviados para as empresas clientes dos WS. Se

entretanto for necessário e se a *gateway* escolhida suportar, provisoriamente, esta poderá publicar os WSDL, implementando as medidas de segurança necessárias para que esta publicação seja efectuada de uma forma segura. Mais tarde, se optarmos pela implementação de um sistema UDDI *Registry/Repository* específico, os esquemas dos WS do Banco BPI deverão ser protegidos por sistemas que notifiquem imediatamente os administradores caso seja efectuada alguma alteração a esses ficheiros.

- O acesso dos clientes aos WS deverá ser efectuada através de HTTPS. Os túneis SSL deverão terminar na XSG (se se justificar, os túneis SSL poderão terminar nas zonas de acessos externos e internos). Desta forma, será garantida a confidencialidade e integridade da mensagem entre o cliente e um dos equipamentos da nossa rede (ponto-a-ponto). Em casos excepcionais, poderão ser estabelecidas VPNs entre os clientes e o Banco. Como a implementação de SSL é muito mais flexível e fácil de configurar do que uma VPN, se possível, implementar a primeira opção.

Entre a *gateway* e o WS não é necessário usar SSL pois confiamos na infra-estrutura já existente. Além disso, se o fizéssemos, iríamos deixar de usufruir da vantagem em passar as tarefas de terminação SSL dos WS para a XSG. Entre *datacenters*, estabeleceremos uma VPN e os dados passarão encriptados pela WAN.

- Uso de WS-Security – Sempre que for necessário, poderá ser necessário encriptar/assinar determinados campos das mensagens de forma a garantir a integridade e confidencialidade *end-to-end*. Além disso, a introdução de *timestamps* e *tokens* de segurança são configurações fundamentais para prevenir alguns tipos de ataques e passar as credenciais dos utilizadores. Sempre que se justificar poderão ser usados certificados X.509. Conforme a XSG que for adquirida, ela própria poderá ser uma CA (Certificate Authority) ou poderá ser necessário instalar/utilizar uma externa.
- Gestão centralizada de PKI – Na infra-estrutura proposta é necessário gerir as chaves usadas na encriptação/descodificação das mensagens ou de determinados campos, certificados usados no tráfego https e, possivelmente, também na autenticação dos utilizadores. As chaves deverão ser guardadas em locais seguros, externos à *gateway* ou na própria (existem algumas *gateways* cujo *hardware* já está preparado para implementar formas seguras de armazenamento de chaves). Se o repositório de chaves privadas da empresa for externo à *gateway*, deverá estar localizado numa VLAN protegida por ACLs ou *firewalls* e a comunicação entre a *gateway* e esse sistema deverá ser efectuada através de canais seguros. Além de ser necessário proteger o sistema onde estão guardadas as chaves, a XSG deverá estar configurada para não guardar na sua *cache* essas chaves durante muito tempo pois, podem ser “roubadas”.

Relativamente aos certificados usados no tráfego https, sendo a *gateway* o local onde são terminados os túneis, eles só precisam de ser instalados neste equipamento e os clientes só precisam de confiar nas chaves usadas pela *gateway*.

- Autenticação/Autorização – A proposta relativa ao local onde deverão ser efectuadas estas operações é na XSG. Nesta infra-estrutura existiam duas hipóteses:
 - Em equipamentos localizados nas “zonas de acessos”;
 - Na XSG que se encontra mesmo antes dos WS.

A primeira opção tem a vantagem de conseguir detectar tentativas de acesso falhadas e não as passar para a zona onde estão os WS mas, tem a desvantagem de ser necessário

aceder à base de dados de utilizadores/autorizações a partir de zonas onde se encontram os equipamentos que recebem directamente os pedidos dos clientes.

A segunda opção tem a desvantagem de permitir que, mensagens não autenticadas, consigam passar o nível de protecção do perímetro com sucesso, e serem do tipo das que permitem efectuar alguns ataques (“XML *port scanning*”, acesso não autorizado a determinados recursos, etc.). Esta opção tem como vantagem o facto dos custos de aquisição de equipamentos ser menor.

Uma sugestão possível é a seguinte:

No mínimo, a autenticação deve ser validada na XSG usando WS-Security e SSL/TLS. Se a XSG terminar as ligações SSL dos clientes, este é logicamente o melhor local da arquitectura para serem efectuadas as validações ao nível da autenticação/autorização e rejeitar as mensagens que não cumprem os requisitos relativos a esta validação. Essas autenticações/autorizações deverão ser validadas tanto no acesso aos WS como no acesso a processos de negócio e dados retornados pelo serviço.

Uma das hipóteses de configuração das *gateways* relativamente a estas questões de autenticação/autorização pode ser o uso de SAML. Ou seja, a *gateway* pode efectuar as tarefas normais de autenticação/autorização e, em seguida, inserir *tokens* SAML na mensagem a enviar para o WS, declarando que ele pode aceitar e processar a mensagem pois esse tipo de validações já está efectuado. Nessas mensagens também podem ser definidos períodos de validade dessas declarações de forma a tentar evitar “*replay attacks*”.

- Integração com ferramentas de gestão de identidades/autorizações “*standards*” e as proprietárias do Banco BPI. Estes sistemas devem estar em VLANs protegidas, pelo menos, por ACLs ou *firewalls*. Ou seja, devido à informação constante nestes servidores ser crítica, os acessos com origem/destino nestes servidores deverão ser bastante controlados. Sempre que possível, deverão ser usados protocolos seguros durante a comunicação com estes sistemas, como por exemplo, LDAPS (*Secure LDAP*), HTTPS, etc..
- Registar todos os acessos. – Os registos deverão ser guardados em meios cujos acessos sejam controlados e, se for necessário, deverão ser encriptados e assinados de forma a proteger a sua integridade e confidencialidade. O protocolo de transporte usado para o envio desses registos das XSG/WS para os sistemas de armazenamento deverá ser considerado seguro (ex.: encriptar os dados enquanto estão em trânsito). De forma a facilitar a consulta dos dados (ex.: visualização dos registos através da criação de filtros de dados, criação de relatórios, visualização de gráficos, etc.), os dados devem ser guardados num sistema central. Para garantir que, em caso de falha desse sistema, se continue a ter acesso aos dados armazenados, estes poderão ser replicados para um *datacenter* remoto. Também é importante implementar um sistema de gestão desses dados de forma a, regularmente, serem efectuadas cópias de segurança dos mesmos.
- Registar todas as operações efectuadas nas configurações dos WS, na XSG, *firewalls* e restantes equipamentos da infra-estrutura.
- Definir vários níveis de administração/operação dos equipamentos, atribuídos a determinados conjuntos de utilizadores. Para facilitar a gestão deste tipo de

autorizações, deverá ser consultado um sistema externo a esses equipamentos, protegido de forma semelhante aos sistemas de gestão de identidades/autorizações dos acessos os próprios WS.

- Sempre que possível, utilizar os *standards* WS e implementar a arquitectura de acordo com as sugestões dos perfis WS-I. Exemplo:

Protecção das mensagens em trânsito: uso de WS-Security, WS-Trust, WS-SecurityPolicy e WS-SecureConversation. Estes *standards* podem ser usados para facilitar o estabelecimento de relações de confiança entre o cliente e o fornecedor do WS, especificar como encriptar, assinar e propagar as identidades nas mensagens e criar a noção de contexto de segurança de um grupo de mensagens. Através da criação dos contextos de segurança, é possível estabelecer uma “conversação” segura entre os intermediários de uma forma mais eficiente pois, não é necessário, para cada uma das interacções pertencentes ao mesmo contexto, voltar a efectuar as tarefas relacionadas com a autenticação.

- Se for necessário isolar o acesso de determinados WS relativamente a outros WS, se a XSG suportar a criação de vários contextos, estes poderão ser criados de forma a ser possível isolar esses acessos. Esta funcionalidade de criação de contextos será bastante útil pois, desta forma, com a criação de pelo menos dois contextos diferentes, a mesma *gateway* poderá ser usada para o ambiente de desenvolvimento e para o de qualidade.
- Periodicamente, deverão ser efectuados testes a toda a infra-estrutura de forma a detectar possíveis erros de programação ou configuração dos equipamentos.
- A gestão dos WS deverá ser efectuada a partir de uma estação de gestão central que permitirá a sua visualização, alteração e configuração. Tal como foi referido relativamente aos sistemas onde são armazenados os registos dos acessos, estes sistemas também deverão ser redundantes e deverão permitir gerir as versões das configurações. Esta gestão central das políticas permite criar, com alguma facilidade, mecanismos que garantem que as políticas aplicadas aos WS seguem determinadas regras. Além disso, em termos de carga operacional, como parte das políticas e definições podem ser partilhadas, é muito mais fácil gerir as configurações das mesmas.
- Integração com ferramentas SOA *Governance*.

Implementação de sistemas que permitam monitorizar permanentemente os WS, tanto através da visualização de gráficos/relatórios, como envio de alertas em caso de serem atingidos determinados valores de *performance*, falhas de equipamentos, erros retornados, etc.. Por questões de seguir as normas impostas ao negócio, por exemplo, gerar relatórios que estejam de acordo com a norma PCI DSS(*Payment Card Industry Data Security Standard*).

Com a implementação dos WS, além da monitorização dos fluxos de pacotes, também passa a ser essencial a monitorização dos fluxos de dados e do comportamento dos próprios WS. Como alguns dos ataques só são detectados através da análise de padrões e anomalias nos dados trocados nas mensagens, além dos valores anteriores, é necessário efectuar uma monitorização do tráfego mais completa do que a

disponibilizada só através da monitorização ao nível dos pacotes. Ou seja, é necessário também monitorizar os próprios dados.

- Configuração de SLAs para os WS que precisem de garantir um valor QoS mais elevado. Frequentemente, é necessário dar maior prioridade a:
 - Determinadas operações em que a resposta em tempo real é um dos requisitos fundamentais.
 - Determinados clientes.
 - Mensagens que contenham determinado conteúdo, etc..
- Garantir que, em caso de falha ou diminuição substancial da *performance* das *gateways* ou os servidores WS, a arquitectura SOA não fica num estado “inseguro”. Ou seja, com a sua falha, o próprio serviço, os dados ou os outros elementos da arquitectura não ficam num estado vulnerável que permita serem comprometidos. Portanto, é necessário configurar os WS e equipamentos de forma a serem capazes de se auto proteger contra falhas, detectando que o seu estado poderá mudar e tomarem as medidas necessárias para que o prejuízo seja mínimo – falhar de uma forma “segura”: guardar configurações e dados, abortar operações sem efectuar o seu “*commit*”, enviar mensagens de alerta, registar todas as ocorrências, etc..

Numa situação ideal, nas zonas de acessos internos e externos, deveriam também ser implementadas XSG com as funções de *firewalls* XML. Essas *firewalls* XML fariam alguma validação XML e detecção de ataques. Como nesse caso, o investimento seria elevado (era necessário adquirir esse tipo de *firewalls* em vários *datacenters*), nesta fase inicial não se está a propor a aquisição desse nível de segurança adicional.

11 Conclusões

A segurança é um dos principais factores críticos para a credibilidade de qualquer instituição que disponibilize serviços aos seus clientes.

Após a identificação dos possíveis problemas de segurança existentes numa arquitectura SOA, torna-se evidente que, para a sua implementação de forma segura, devem ser tomadas medidas que permitam garantir a integridade e confidencialidade dos dados/comunicações/transacções, gerir os acessos (autenticação/autorização), registar todas as alterações/operações/acessos efectuados, monitorizar constantemente toda a arquitectura, etc..

Como as arquitecturas SOA já foram adoptadas por um grande número de empresas, actualmente, existem os *standards* WS-*, algum *software* de apoio a estas arquitecturas (Gestão de Identidades, SOA Governance, UDDI Registry/Repository, etc.) e XML Security Gateways que já apresentam um grau de maturidade suficiente para permitir a implementação de uma arquitectura SOA segura. No caso prático do Banco BPI, como existiam alguns WS, algumas das medidas de segurança propostas já estão a ser aplicadas. O novo desafio apresentado – disponibilização de WS para o exterior – irá requerer a implementação de XML Security Gateways. Como, actualmente, existem bastantes fornecedores destes produtos, e estes oferecem bastantes funcionalidades de segurança, a implementação das boas práticas listadas nos pontos anteriores provavelmente será facilitada com a introdução destes equipamentos na infra-estrutura.

No entanto, só a introdução de XML Security Gateways não irá garantir a implementação de uma arquitectura segura. Será necessário cumprir as boas práticas de segurança, efectuar testes periódicos à infra-estrutura e monitorizar permanentemente os serviços e a operacionalidade de todos os equipamentos pertencentes à arquitectura SOA.

Referências e Bibliografia

- [1] A. Buecker, P. Ashley, M. Borrett, M. Lu, S. Muppidi, e N. Readshaw. *Understanding SOA Security – Design and Implementation*. IBM.com Redbooks
- [2] B. Blakley , C. Heath, e membros do “The Open Group Security Forum”. *Security Design Patterns*. Disponível em:
<http://www.opengroup.org/onlinepubs/9299969899/toc.pdf>
- [3] Cisco. Vários documentos disponíveis em: www.cisco.com
- [4] F5 Networks. Vários documentos disponíveis em: www.f5world.com
- [5] Forum Systems. Vários documentos disponíveis em: <http://www.forumsystems.com>
- [6] Gartner. *Best Practices Checklist for Web Services Security, 2006 – ID Number: G00140006*, 12 Maio 2006. Disponível em: www.gartner.com
- [7] Gartner (2007). *Magic Quadrant for Integrated SOA Governance Technology Sets, 2007*. Disponível em: www.gartner.com
- [8] IBM. Vários documentos disponíveis em: www.ibm.com
- [9] IETF – Internet Engineering Task Force. *IETF RFC 2807 – XML Signature Requirements*. Disponível em: <http://www.ietf.org/rfc/rfc2807.txt>
- [10] IETF – Internet Engineering Task Force. *IETF RFC 3275 – Signature Syntax and Processing*. Disponível em: <http://www.ietf.org/rfc/rfc3275.txt>
- [11] R. Kanneganti, e P. Chodavarapu. *SOA Security*, 2008. Manning Publications.
- [12] E. Kuznetsov. *XML Web Services security best practices*, 2004. ZDNet. Disponível em:
http://news.zdnet.com/2100-1009_22-5345253.html
- [13] Layer7 Technologies. Vários documentos disponíveis em: www.layer7tech.com
- [14] C. MacKenzie, K. Laskey, F. McCabe, P. Brown, R. Metz. *Reference Model for Service Oriented Architecture 1.0*, 2006. OASIS Open 2005-2006. Disponível em:
<http://docs.oasis-open.org/soa-rm/v1.0/soa-rm.pdf>
- [15] T. Moses. *eXtensible Access Control Markup Language (XACML) Version 2.0*, 2005. OASIS Open 2004-2005. Disponível em:
http://docs.oasis-open.org/xacml/2.0/access_control-xacml-2.0-core-spec-os.pdf
- [16] OASIS – Organization for the Advancement of Structured Information Standards. Vários documentos disponíveis em: www.oasis-open.org
- [17] Open Group. *Security Design Patterns*. Disponível em:
<http://www.opengroup.org/onlinepubs/9299969899/toc.pdf>
- [18] Open Web Application Security Project. *OWASP XML Security Gateway Evaluation Criteria Project*. Disponível em: www.owasp.org
- [19] D. Paschich. *Deploying the Cisco ACE XML Gateway*. Disponível em: brkapp-2014.pdf no DVD do Cisco Networkers 2008

- [20] D. Patterson. *XML Firewall Architecture and Best Practises for Configuration and Auditing*, 2007. CISSP-ISSEP
- [21] Chris Peiris, e Dennis Mulder. *Pro WCF Practical Microsoft SOA Implementation*. Apress
- [22] *Security at the edge: Rethinking security in light of Web Services*. Disponível em: www.iacis.org
- [23] A. Singhal, T. Winograd, e K. Scarfone. *Guide to Secure Web Services*, 2007. NIST – National Institute of Standards and Technology. Disponível em: www.nist.gov
- [24] L. Vittie. *SOA: Challenges and Solutions*, 2007. F5 Networks. Disponível em: <http://www.f5.com/pdf/white-papers/soa-challenges-solutions-wp.pdf>
- [25] Vordel. Vários documentos disponíveis em: www.vordel.com
- [26] W3C – World Wide Web Consortium. Vários documentos disponíveis em: www.w3c.org
- [27] Web Application Security Consortium. *Web Application Firewall Evaluation Criteria*. Disponível em: www.webappsec.org
- [28] Web Services Interoperability (WS-I) Organization. *Deliverables from the Testing Tools Working Group*. Disponível em: <http://www.ws-i.org/deliverables/workinggroup.aspx?wg=testingtools>

ANEXO A: **Standards de Segurança**

SSL/TLS

SSL (*Secure Socket Layer*) foi desenhado para encapsular outros protocolos (ex. : HTTP). SSL v.3 foi criado em 1996 e, em 1999, foi sucedido pelo TLS (*Transport Layer Security*), uma definição da IETF. SSL é o protocolo de comunicação de dados ao nível do transporte mais usado que fornece:

- Autenticação – A comunicação é estabelecida entre duas partes de confiança.
- Confidencialidade – Os dados trocados são encriptados.
- Integridade da mensagem – Os dados são validados de forma a verificar se existe alguma corrupção dos mesmos.
- Troca de chaves seguras entre o servidor e o cliente.

O SSL pode ser usado em três modos:

- Sem autenticação – Nem o cliente nem o servidor se autenticam um ao outro. Não são enviados nem trocados certificados (só é usada a encriptação e decodificação).
- Autenticação só num dos lados (ou autenticação do servidor) – Só o servidor é que se autentica perante o cliente. O servidor envia para o cliente um certificado de forma a ser possível verificar que o servidor é o “autêntico”.
- Autenticação em ambos os lados (autenticação bilateral) – Tanto o cliente como o servidor autenticam-se um ao outro através do envio de certificados um ao outro. Este modo é usado para prevenir ataques entre o *proxy/gateway* e o fornecedor do WS.

SSL usa a criptografia baseada na combinação de chave secreta/chave pública de forma a tornar as comunicações seguras. São usadas chaves secretas para encriptar e decodificar e chaves públicas para a autenticação mútua das partes envolvidas na comunicação.

XML-Encryption

XML-Encryption é um *standard*, criado pelo W3C, onde está especificado o processo de encriptação dos dados e a representação, em XML, do resultado obtido. Esses “dados” podem ser simples dados (incluindo documentos XML), elementos XML ou o conteúdo de um elemento XML. O resultado deste processo de encriptação é representado na forma de um elemento `<EncryptedData>` que contém ou identifica, através de uma referência URI, os dados encriptados. A estrutura do elemento encriptado é representado pela seguinte estrutura:

```
<EncryptedData ID? Type? MimeType? Encoding?>
  <EncryptionMethod/>?
  <ds:KeyInfo>
    <EncryptedKey>?
    <AgreementMethod>?
    <ds:KeyName>?
    <ds:RetrievalMethod>?
    <ds:*>?
```

```

</ds:KeyInfo>?
<CipherData>
  <CipherValue>?
  <CipherReference URI?>?
</CipherData>
<EncryptionProperties>?
</EncryptedData>

```

No exemplo, “+” representa “1 ou mais ocorrências”, “?” representa “0 ou 1 ocorrência” e “*” representa “0 ou mais ocorrências”.

Um documento XML pode conter zero ou mais elementos `<EncryptedData>`. `<EncryptedData>` não pode ser “pai” ou “filho” de outro elemento `<EncryptedData>`. No entanto, os dados origem que serão alvo do processo de encriptação podem ser de vários tipos, incluindo elementos `<EncryptedData>` ou `<EncryptedKey>`. Neste caso, estamos perante o conceito de “super-encriptação” em que o processo de encriptação deve ser aplicado ao elemento como um todo.

`<EncryptionMethod>` é um elemento opcional que descreve o algoritmo de encriptação usado para cifrar os dados. Exemplo:

- Algoritmo de encriptação de blocos. – Algoritmos criados para encriptar e decodificar dados de tamanho fixo, em múltiplos blocos de octetos. Têm como parâmetros de entrada os dados a serem encriptados/descodificados, a chave e a direcção da operação. Neste tipo de algoritmos é usado um vector de inicialização que é codificado com o texto da cifra. Ex.: *tripledes-cbc*, *aes128-cbc*, *aes256-cbc*, *aes192-cbc* (CBC – *Cipher Block Chaining*).
- Algoritmos de encriptação “stream”. – Algoritmos que, a partir de uma chave, geram uma *stream* de *bytes* aos quais é aplicada a função “XOR” sobre os *bytes* dos dados em texto claro. Caso seja um processo de encriptação, da aplicação do “XOR” é criado o texto cifrado. Se for um processo de descodificação é produzido o texto em claro.
- Algoritmos de “transporte” de chaves. – Algoritmos de encriptação que usam chaves públicas específicos para a encriptação e descodificação de chaves (este algoritmo também pode ser utilizado para encriptar os dados). Ex.: *rsa-1_5*, *rsa-oaep-mgf1p*.
- Algoritmos “*Symmetric Key Wrap*”. – Algoritmos de encriptação que usam chaves secretas partilhadas específicas para encriptar e decodificar chaves simétricas e permitir a verificação da integridade. Ex.: *kw-tripledes*, *kw-aes128*, *kw-aes256*, *kw-aes192*.

`<ds:KeyInfo>` é um elemento também opcional que contém informação relativa à chave usada para encriptar os dados: `<EncryptedKey>`, `<AgreementMethod>` (ex.: DH – Diffie-Hellman, `DHKeyValue`) , `<ds:KeyName>`, `<ds:RetrievalMethod>` (permite representar a ligação para um elemento `<EncryptedKey>` que contém a chave para decodificar o `<CipherData>` associado a um `<EncryptedData>` ou `<EncryptedKey>`), `<ds:*`. Como derivação das chaves, na secção `<AgreementMethod>` podem ser definidos algoritmos de *message digest* (`<DigestMethod>`; ex.: *sha1*, *sha256*, *sha512*, *ripemd160*) a serem usados em conjunto com a encriptação RSA-OAEP na forma de uma função de *hash* e em conjunto com o método de autenticação HMAC das mensagens.

O elemento <CipherData> pode conter os dados já encriptados (<CipherValue>) ou a referência da sua localização através do respectivo URI (<CipherReference>).

<EncryptionProperties> pode conter informação adicional relacionada com a geração do <EncryptedType> (ex.: *timestamp*, número de série do equipamento físico (*hardware*) usado no processo de encriptação, etc.).

XML-Signature

XML-Signature é um *standard* desenvolvido em conjunto pelo W3C e IETF (RFC 2807, RFC 3275) que permite garantir a integridade dos dados, autenticação da entidade que assinou os dados e garantia de “não repúdio”. Este *standard* define a forma como devem ser assinados os conteúdos digitais, incluindo conteúdos XML, recorrendo a assinaturas cuja própria sintaxe é baseada em XML. Esta sintaxe permite representar a assinatura XML de qualquer tipo de dados digitais localizados tanto no documento XML que inclui a assinatura como em outra parte externa (ex.: recursos *Web*, partes das mensagens – algo referenciado por URI). Além disso, este *standard* também define os procedimentos utilizados no cálculo e verificação dessas assinaturas. Ou seja, XML-Signature define um tipo de elemento assinatura XML e uma aplicação de geração/validação de assinaturas XML. Este *standard* também inclui a definição de outros tipos que identificam métodos para referenciar coleções de recursos, algoritmos, informação sobre as chaves e a sua gestão.

A assinatura XML é um método de associação da chave privada do emissor com os dados referenciados (octetos). Ou seja, as assinaturas XML são geradas através da aplicação de uma função *hash* a uma sequência arbitrária de *bytes* (conjunto de referências relativas ao objecto que vai ser assinado) convertendo-os num valor de tamanho fixo denominado *digest*. O *digest* é um processo de transformação unilateral, ou seja, é computacionalmente impraticável recriar a mensagem a partir do *hash* ou encontrar duas mensagens diferentes que produzam o mesmo valor *digest*. O mecanismo de *hash* que normalmente é utilizado é o SHA1 (*Secure Hash Algorithm*), criado pelo governo dos Estados Unidos e considerado como um *standard* desde 1995.

A sintaxe das assinaturas XML deve ser suficientemente extensível de forma a poder suportar qualquer semântica aplicacional/relações de confiança e declarações de assinaturas.

As aplicações que geram/validam assinaturas XML devem estar em conformidade com as seguintes especificações:

- *XML-namespaces* – As aplicações devem escolher algoritmos C14N (canonização) que processam, ou não, *namespaces* dentro do conteúdo XML. Ou seja, alguns algoritmos C14N podem optar por remover todas as declarações de *namespaces*, outros algoritmos podem reescrever as declarações dos *namespaces* para criarem contextos de declarações de cada elemento independentes.
- *Xlink* – Para identificação de cada um dos recursos baseada em URIs simples (sem identificadores dos fragmentos) ou com identificadores dos fragmentos, as aplicações devem utilizar *Xlink locators* para referenciar os recursos assinados.
- *XML-Pointers* – Se as aplicações referenciarem partes dos documentos XML, devem usar *XML-Pointers* dentro do *Xlink locator*.

Seguindo este *standard*, é possível só assinar determinadas partes da árvore XML, ou seja, não é necessário assinar todo o documento. Desta forma, quando um determinado documento XML precisa de ser assinado várias vezes, por várias entidades, é possível garantir a integridade de determinadas partes do documento e deixar a possibilidade de alguma entidade alterar livremente as outras partes do respectivo documento.

As assinaturas digitais XML são representadas pelo elemento assinatura cuja estrutura é a seguinte:

```
<Signature ID?>
  <SignedInfo>
    <CanonicalizationMethod/>
    <SignatureMethod/>
    (<Reference URI? >
      (<Transforms>)?
      <DigestMethod>
      <DigestValue>
    </Reference>)+
  </SignedInfo>
  <SignatureValue>
  (<KeyInfo>)?
  (<Object ID?>)*
</Signature>
```

No exemplo, “+” representa “1 ou mais ocorrências”, “?” representa “0 ou 1 ocorrência” e “*” representa “0 ou mais ocorrências”.

Para poder validar a assinatura é necessário conseguir aceder ao objecto que foi assinado. Portanto, geralmente, a própria assinatura XML deve indicar a localização do objecto original. Essa referência pode ser indicada de uma das seguintes formas:

- Referenciada por um URI contido na assinatura XML. – No caso de serem documentos XML, as assinaturas são relacionadas com os objectos de dados locais através de identificadores dos fragmentos.
- Residir dentro do mesmo recurso que a assinatura XML (a assinatura é “irmã”). – As assinaturas cujo conteúdo é externo ao próprio elemento assinatura (as assinaturas não estão incluídas no conteúdo que elas próprias vão assinar) são relativas a recursos de rede externos ou objectos dos dados locais que residem dentro do mesmo documento XML como elementos “irmãos”.
- Estar embebida dentro da assinatura XML (a assinatura é “mãe”). – A assinatura é incluída dentro do envelope
- Ter a sua assinatura XML embebida dentro de si mesma (a assinatura é “filha”). – A assinatura é considerada como atributo do envelope.

Como o elemento da assinatura, o seu identificador e nome podem coexistir, ou serem combinados com outros elementos num único documento XML, é necessário ter cuidado com a atribuição de nomes de forma a garantir que os identificadores são únicos.

Exemplo da especificação de uma assinatura XML:

```
<ds:Signature ID="MinhaAssinatura" xmlns:ds="http://www.w3.org/2000/09/xmldsig#">
  <ds:SignedInfo>
    <ds:CanonicalizationMethod Algorithm="http://www.w3.org/2001/10/xml-exc-c14n#" />
    <ds:SignatureMethod Algorithm="http://www.w3.org/2000/09/xmldsig#rsa-sha1" />
    <ds:Reference URI="#CorpoMsg">
```

```

        <ds:DigestMethod Algorithm="http://www.w3.org/2000/09/xmldsig#sha1"/>
        <ds:DigestValue>kjklAJKHSJD...</ds:DigestValue>
    </ds:Reference>
</ds:SignedInfo>
<ds:SignatureValue>DJBchm5gK...</ds:SignatureValue>
<ds:KeyInfo>
    <wsse:SecurityTokenReference>
        <wsse:Reference URI="#IDdoMeuToken"/>
    </wsse:SecurityTokenReference>
</ds:KeyInfo>
</ds:Signature>

```

Na secção `<ds:SignedInfo>` são definidas as partes da mensagem que serão assinadas (`ds:Reference URI=...`) e os métodos utilizados (`ds:CanonicalizationMethod` e `ds:SignatureMethod`):

- `CanonicalizationMethod` (ex.: *Canonical XML*, *Canonical XML with comments*) – Algoritmo usado para canonizar o elemento `<SignedInfo>` antes deste ser *digest* como parte da operação de assinatura.
- `SignatureMethod` (ex.: *DSAwithSHA1* (DSS), *RSAwithSHA1*) – Algoritmo utilizado para converter a forma canónica do `<SignedInfo>` no `<SignedValue>`. É uma combinação do *digest algorithm* e do *key dependent algorithm* e, possivelmente, outros algoritmos tais como *padding schemes* (ex.: *RSA-SHA1*). Os próprios nomes dos algoritmos são assinados de forma a resistir a ataques baseados na substituição destes algoritmos por algoritmos mais “fracos”.

As principais validações efectuadas na secção `<SignedInfo>` consistem em dois processos obrigatórios: validação da assinatura e validação de cada referência *digest*.

Na secção `<ds:Reference>`, incluída na `<ds:SignedInfo>`, é definido o `<ds:DigestMethod>` (ex.: *SHA1* – *MD5* não é recomendado) e o resultante `<ds:DigestValue>` calculado para o elemento referenciado nesta secção. Opcionalmente, também pode incluir os `<Transforms>` que produzem as variáveis de entrada para a operação de *digest* – lista de passos aplicados aos dados antes destes serem *digest*, como por ex.: operações de canonização, codificação/descodificação (*Base64*), *XSLT Transform*, *Xpath Filtering*, validação do esquema XML, *XInclude*. Se não forem incluídos os `<Transforms>`, os dados são directamente *digest*. O objecto de dados é assinado através do cálculo baseado no seu valor de *digest* e na assinatura. Mais tarde, a assinatura é verificada através da “*reference*” e validação da assinatura.

Em seguida, já fora da secção `<ds:SignedInfo>`, é definido o `<ds:SignatureValue>` (elemento que contém o valor do “*encritped digest*” do elemento `<ds:SignedInfo>`).

Na secção `<ds:KeyInfo>` é disponibilizada a informação relativa a como encontrar o *token* de segurança associado à assinatura a utilizar (`<wsse:SecurityTokenReference>`) ou qual a chave que deve ser usada para validar a assinatura (certificados, nomes de chaves e algoritmos de negociação de chaves), por exemplo:

```

    <KeyInfo>
        <KeyValue>
            <DSAKeyValue>
                <P>...</P><Q>...</Q><G>...</G><Y>...</Y>
            </DSAKeyValue>
        </KeyValue>
    </KeyInfo>

```

Esta secção é opcional pois a entidade que está a assinar pode não querer revelar a informação relativa às chaves a todas as partes que precisam de processar o documento ou, a informação pode ser já conhecida devido ao contexto do documento e não é necessário representá-la explicitamente nesta secção.

Resumindo, este processo de criação da assinatura pode ser representada da seguinte forma:

```
Para cada ObjectoDeDados,
    DigestValue = DigestMethod (Transformers (ObjectoDados))
    Reference_Element = (ID_ObjectoDados, Transformers (ObjectoDados), DigestMethod, DigestValue)
    SignedInfo_Element = (SignatureMethod, CanonicalizationMethod, Reference_Element(s))
    SignatureValue = SignatureMethod (KEY, CanonicalizationMethod (Reference_Element(s)))
    Signature_Element (SignedInfo, Objets(s), KeyInfo, SignatureValue)
```

O processo de validação pode ser representado da seguinte forma:

Validação das Referências. – Verificação do *digest* contido em cada referência do <SignedInfo>:

```
Aux1 = CanonicalizationMethod (SignedInfo_Element)
Para cada Reference_Element do Aux1,
    ObjectoDados_AUX=SignatureMethodXX(KEY,TransformersXX(Reference_Element_AUX))
    (ou o valor ObjectoDados_AUX está na cache)
    ObjectoDados_AUX_Digested = DigestMethod (ObjectoDados_AUX)
    Se ObjectoDados_AUX_Digested = DigestValue então
        Validação = OK
    Se não,
        Validação = FALHA
```

Validação da assinatura criptográfica calculada através do <SignedInfo>:

1. Obter a informação sobre a chave (Chave**Original**) a partir do KeyInfo ou de uma entidade externa.
2. **Aux** = CanonicalizationMethod (SignatureMethod())
3. Usar o valor de **Aux** para confirmar o SignatureValue do SignedInfo_Element.

Embora este *standard* seja um componente importante para garantir a segurança das aplicações baseadas em XML, só ele não é suficiente para resolver todos os problemas relacionados com a segurança/relações de confiança, nomeadamente no uso, durante as comunicações, de dados XML assinados (ou outros formatos de dados).

XACML (*eXtensible Access Control Markup Language*)

XACML é uma linguagem baseada em XML, desenvolvida pela OASIS, utilizada na escrita das políticas de controlo de acessos a aplicar aos diversos equipamentos e aplicações. O *standard* XACML descreve tanto a linguagem usada para definir as políticas como a usada nas decisões de controlo de acessos dos diversos pedidos/respostas, permitindo a escrita de políticas que determinam quais são requisitos para aceder a determinado recurso.

A linguagem usada na definição de políticas é usada para descrever os requisitos de controlo de acessos e tem algumas extensões que permitem definir novas funções, tipos de dados, lógica combinacional, etc.. A linguagem usada nos pedidos/respostas permitem criar perguntas para verificar se determinada acção é permitida e interpretar o seu resultado. As respostas podem ter um dos quatro valores seguintes: “*Permit*”, “*Deny*”, “*Indeterminate*” (a decisão não pode ser tomada porque ocorreu um erro ou faltam dados) ou “*Not Applicable*” (a resposta não pode ser dada por este serviço).

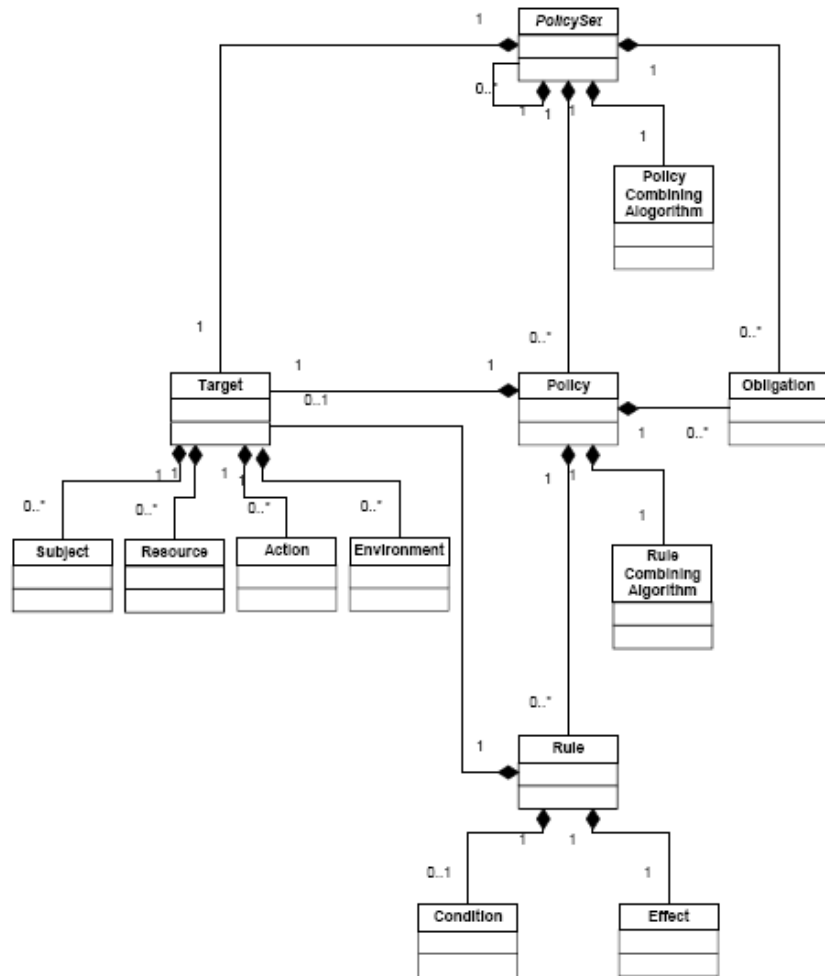


Figura A. 1: Modelo da linguagem das políticas XACML

(Fonte: OASIS [15])

Através do XACML é praticamente possível usar qualquer informação para decidir se o acesso a determinado recurso é ou não possível. As decisões podem ser baseadas nas propriedades dos próprios recursos (seu conteúdo, pertencer a determinado grupo, etc.) ou em factores externos (data, hora, localização, etc.). Também é possível associar acções adicionais (denominadas *obligations*) às decisões, como por exemplo, definir que os dados deverão ser destruídos ao fim de um determinado número de dias.

Este *standard* pode ser utilizado, por exemplo, quando alguém precisa de efectuar uma determinada acção num recurso. Numa primeira fase, é efectuado um pedido ao “componente” – PEP (*Policy Enforcement Point*) – que protege esse recurso (ex.: sistema de ficheiros, servidor *web*, etc.). O PEP criará um pedido baseado num dos atributos do solicitante, no recurso, na acção ou noutra informação relevante para esta solicitação. Esse pedido será enviado pelo PEP para o PDP (*Policy Decision Point*) que irá avaliar o pedido de acordo com uma determinada política que se aplique a esse pedido. Em seguida, o PDP enviará a resposta com a informação sobre se terá ou não acesso. O PEP e o PDP podem pertencer à mesma aplicação ou podem estar distribuídos por vários servidores.

Exemplo da definição de uma política:

A política seguinte é só aplicada a pedidos efectuados a um servidor cujo nome é “xx.banco.pt”. Esta política retorna o valor “*Permit*” só para acções de “*login*” efectuadas entre as 8h30m e as 15h. Em todos os outros casos é retornado o valor “*Deny*”.

Relativamente à sua estrutura, uma política XACML é composta pelo cabeçalho, um texto descritivo dessa política (opcional), identificação do destino (*target*), uma ou mais regras e um conjunto opcional de *obligations*. Neste exemplo, iremos ter uma política com uma só regra mas, em exemplos mais complexos, podem existir grupos de políticas compostos por várias políticas, ou mesmo, por outros grupos de políticas. As próprias políticas podem ser compostas por várias regras (tal como está representado na Figura A. 1).

```
<?xml version="1.0" encoding="UTF-8"?>
<Policy PolicyId="PoliticaExemplo"
  RuleCombiningAlgId="urn:oasis:names:tc:xacml:1.0:rule-combining-algorithm:permit-overrides">
  <Description>
    Esta política só se aplica a servidores cujo nome é xx.banco.pt, etc.... </Description>
  <Target>
    <Subjects>
      <AnySubject/>
    </Subjects>
    <Resources>
      <ResourceMatch MatchId="urn:oasis:names:tc:xacml:1.0:function:string-equal">
        <AttributeValue
          DataType="http://www.w3.org/2001/XMLSchema#string">xx.banco.pt</AttributeValue>
        <ResourceAttributeDesignator DataType="http://www.w3.org/2001/XMLSchema#string"
          AttributeId="urn:oasis:names:tc:xacml:1.0:resource:resource-id"/>
      </ResourceMatch>
    </Resources>
    <Actions>
      <AnyAction/>
    </Actions>
  </Target>
  <!--Regra que verifica se pode ser efectuado login -->
  <Rule RuleId="RegraDeLogin" Effect="Permit">
    <!--Esta regra só sera usada se a acção for de login -->
    <Target>
      <Subjects>
        <AnySubject/>
      </Subjects>
      <Resources>
        <AnyResource/>
      </Resources>
      <Actions>
        <ActionMatch MatchId="urn:oasis:names:tc:xacml:1.0:function:string-equal">
          <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#string">login</AttributeValue>
          <ActionAttributeDesignator DataType="http://www.w3.org/2001/XMLSchema#string"
            AttributeId="ServerAction"/>
        </ActionMatch>
      </Actions>
    </Target>
    <!--Só permite efectuar login se for entre as 8h30m e as 15h -->
    <Condition FunctionId="urn:oasis:names:tc:xacml:1.0:function:and">
      <Apply FunctionId="urn:oasis:names:tc:xacml:1.0:function:time-greater-than-or-equal"
```

```

<Apply FunctionId="urn:oasis:names:tc:xacml:1.0:function:time-one-and-only">
  <EnvironmentAttributeSelector DataType="http://www.w3.org/2001/XMLSchema#time"
    AttributeId="urn:oasis:names:tc:xacml:1.0:environment:current-time"/>
</Apply>
<AttributeValue DataType="http://www.w3.org/2001/XMLSchema#time">08:30:00</AttributeValue>
</Apply>
<Apply FunctionId="urn:oasis:names:tc:xacml:1.0:function:time-less-than-or-equal"
  <Apply FunctionId="urn:oasis:names:tc:xacml:1.0:function:time-one-and-only">
    <EnvironmentAttributeSelector DataType="http://www.w3.org/2001/XMLSchema#time"
      AttributeId="urn:oasis:names:tc:xacml:1.0:environment:current-time"/>
  </Apply>
  <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#time">15:00:00</AttributeValue>
</Apply>
</Condition>
</Rule>
<!-- As regras relativas a outras acções podiam ser incluídas nesta secção -->
<!-- A regra final que, nega (Deny) tudo que não satisfaz as condições das regras anteriores -->
<Rule RuleId="FinalRule" Effect="Deny"/>
</Policy>

```

SAML (*Security Assertion Markup Language*)

SAML (*Security Assertions Markup Language*) é um *standard* XML, desenvolvido pela OASIS, que suporta o *single sign-on*. Ou seja, permite que um utilizador se autentique uma única vez e, a partir desse momento, possa aceder a outros recursos sem ter de voltar a preencher as suas credenciais de autenticação. Como SAML permite incluir o estado da autorização e permissões de autorização numa mensagem, estes dados podem ser passados do primeiro recurso para os seguintes. O *standard* SAML define um vocabulário XML, usado na partilha de declarações de segurança, que especificam quando e como é que uma entidade foi autenticada, informação sobre os atributos de uma entidade ou quando é que uma entidade está autorizada para executar uma determinada acção. Estas declarações permitem a federação de identidades e autorização distribuída dentro de uma infra-estrutura SOA.

SAML foi originalmente criado para responder aos seguintes requisitos:

- Limitações dos *cookies* dos *browsers* da *Internet*. – SAML permite, de uma forma *standard*, criar uma forma de transferir *cookies* entre múltiplos domínios da *Internet*.
- SSO (*Single Sign-On*) proprietário. – SAML fornece um protocolo *standard* para implementar SSO dentro de um domínio ou entre múltiplos domínios.
- Federação. – SAML fornece gestão de identidades (um utilizador pode ter múltiplas identidades na *Internet*).
- Segurança dos WS. – SAML fornece um *token* de segurança *standard* (declaração SAML) que pode ser usado no WS-Security.
- Propagação de identidades. – SAML permite representar um *token* de segurança *standard* que pode ser passado entre os múltiplos passos que compõem um processo de negócio ou transacção.

A SAML é baseada em quatro componentes:

- Declarações. – Declarações relacionadas com autenticação (valida a identidade do utilizador), atributos (contém determinadas informações sobre o utilizador) ou autorizações (identifica o que é que o utilizador está autorizado a fazer).
- Protocolos. – Definição da forma como a SAML requisita ou fornece as declarações.
- *Bindings*. – Definem como é que as trocas de mensagens SAML são mapeadas para SOAP, HTTP, SMTP, FTP, etc..
- Perfis. – A forma de combinar os protocolos e *bindings* SAML de forma a suportar determinados casos práticos.

A razão pela qual as declarações SAML são bastante usadas como *tokens* de segurança dentro do WS-Security deve-se ao facto de serem bastante flexíveis, permitindo criar um grande número de combinações de expressões e ajudarem a prevenir ataques *man-in-the-middle* e “*replay attacks*”.

Normalmente, uma declaração SAML relativa a um utilizador ou aplicação inclui:

- Versão SAML.
- Identificador único da declaração.
- Instante em que a declaração foi emitida.
- Identificador do emissor.
- Opcionalmente,
 - o Assinatura. – Uma assinatura XML que permite garantir a integridade da declaração e autentica o emissor dessa declaração.
 - o Sujeito (*subject*). – Informação sobre a entidade à qual esta declaração se aplica (ex.: chave pública).
 - o As condições em que a declaração é válida.
 - o Recomendações. – Informação adicional que, em algumas, pode ser necessária durante o processamento das declarações.
 - o Afirmações. – As declarações SAML podem incluir três tipos de afirmações:
 - Afirmação da autenticação. – Indica que determinado “sujeito” foi autenticado, fornecendo alguns detalhes: qual o método de autenticação utilizado, em que momento é que essa autenticação ocorreu e qual a autoridade de autenticação usada. Ex.: declara que o “sujeito” A foi autenticado através de B no momento M.
 - Afirmação dos atributos. – Emitida por uma autoridade responsável pelos atributos, com base em políticas. Por exemplo, declara que o “sujeito” A está associado aos atributos B, C, etc., com os valores b, c, etc..
 - Afirmação da decisão de autorização (alguns elementos foram descontinuados na SAML v2.0 e agora são suportados pela XACML). – Este tipo de afirmações permitem declarar que, um pedido efectuado por um determinado “sujeito”, para aceder a determinado recurso, resultou

numa determinada decisão de autorização. Opcionalmente, podem ser incluídas determinadas “provas” para suportarem essa decisão. Ex.: uma autoridade de autorização decide quando é que se deve garantir o pedido do “sujeito” A, para executar a acção X (ex.: ler, escrever, etc.) no recurso R (ex.: ficheiro, aplicação, WS, etc.), através da prova P.

Também é possível definir outros tipos de afirmações através da criação de extensões ao esquema.

As declarações SAML podem conter outras declarações SAML. Podem ser assinadas através do uso de XML-Signature e/ou encriptadas através de XML-Encryption.

Exemplo:

```
<saml:Assertion xmlns:saml="urn:oasis:names:tc:SAML:2.0:assertion"
  Version="2.0"
  IssueInstant="2008-04-27T10:25:06Z">
  <saml:Issuer Format="urn:oasis:names:SAML:2.0:nameid-format:entity"> http://emissor.empresa.pt
  </saml:Issuer>
  <saml:Subject>
    <saml:NameID Format="urn:oasis:names:tc:SAML:1.1:nameid-format:x509SubjectName">
      CN=Jose, OU=EmpresaX, O=GrupoEmpresarial,C=Portugal
    </saml:NameID>
  </saml:Subject>
  <saml:Conditions
    NotBefore="2008-04-27T10:25:06Z"
    NotOnOrAfter="2008-04-27T11:25:06Z"
  </saml:Conditions>
  <saml:AuthnStatement
    AuthnInstant="2008-04-27T10:25:06Z"
    <saml:AuthnContext>
      <saml:AuthnContextClassRef>
        urn:oasis:names:tc:SAML:2.0:ac:classes:X509
      </saml:AuthnContextClassRef>
    </saml:AuthnContext>
  </saml:AuthnStatement>
</saml:Assertion>
```

Este exemplo refere-se a uma situação relacionada com a autenticação. Esta declaração indica que o José, da EmpresaX, foi autenticado, no dia 27 de Abril de 2008, através de um certificado X.509. Essa declaração teve a validade de 1 hora.

O *standard* SAML também define o protocolo SAML. É um protocolo, baseado em XML, onde são definidas algumas regras a ter em conta durante o processamento dos pedidos e respostas utilizados na “troca” das declarações SAML. Neste *standard* é descrita a forma como esses pedidos e respostas SAML são embebidos no HTTP e SOAP, permitindo que SAML suporte tanto as tradicionais aplicações *Web* com os WS. Esta especificação define seis protocolos básicos, sendo possível criar algumas extensões a esta especificação:

- Pedido e resposta de declarações. – Permite que os serviços solicitem determinadas declarações a uma autoridade SAML. Os pedidos podem ser feitos com base no identificador único da declaração, ou podem ser solicitadas declarações relacionadas com autenticação, atributos ou decisões de autorização.
- Pedido de autenticação. – Permite que os serviços solicitem às autoridades de autenticação que uma determinada entidade seja autenticada. O “sujeito” pode ser

autenticado antes, durante ou após a autoridade de autenticação receber o pedido de autenticação. O protocolo de pedido de autenticação torna-se bastante útil em situações em que é necessário autenticar utilizadores em cenários onde estão envolvidos serviços públicos e privados. Este protocolo também suporta o pedido de voltar a autenticar a entidade ou pode solicitar métodos de autenticação mais fortes, a serem utilizados em serviços mais críticos.

- Resolução de “artefactos”. – Permite criar um mecanismo em que as mensagens SAML são “transportadas” por referência em vez de ser por valor. Ou seja, o emissor da mensagem SAML, em vez de a enviar através de um protocolo de transporte, envia uma pequena peça de dados denominada “artefacto”. Os “artefactos” deverão ser usados quando o mecanismo de transporte tem restrições relativamente ao tamanho das mensagens ou quando não permite a criação de um canal seguro para o envio das mensagens SAML.
- Gestão do nome do identificador. – Fornece um mecanismo que permite que os fornecedores de identidades SAML e os fornecedores dos serviços possam enviar notificações relativas a novas identificações ou modificação de valores ou formatos. Os fornecedores de identidades disponibilizam o elemento “ManageNameIDRequest” contendo os elementos “NameID” ou “EncryptedNameID” e os elementos “NewID”, “NewEncryptedID” ou “Terminate”.
- Logout simples. – Permite que, através de uma troca de mensagens, todas as sessões relativas a uma determinada sessão de uma autoridade sejam terminadas quase simultaneamente.
- Correspondência do nome do identificador. – Quando uma entidade partilha um identificador com uma entidade, muitas vezes precisa de obter o nome associado a esse identificador num formato especial ou *namespace* de federação. Para tal, pode enviar um pedido ao fornecedor de identidades usando este protocolo.

SAML fornece também alguns perfis para especificar a forma como as mensagens SAML, declarações e protocolos devem ser usados nos vários contextos. Como SAML é um *standard* versátil, existem cinco categorias de perfis:

- SSO. – Este perfil define os protocolos necessários para suportar SSO entre múltiplas aplicações *Web*. Existem perfis que:
 - Definem como suportar SSO SAML quando o cliente usa um *Web Browser*, quando é um cliente SAML ou um *proxy*.
 - São usados para descobrir identidades.
 - São usados para *logout* simples.
 - São usados para gerir identificadores entre múltiplos fornecedores de serviços.
- Resolução de “artefactos”. – Este perfil define a forma como os fornecedores baseados em SAML devem obter as declarações associadas a um “artefacto” SAML.
- Pedidos/Respostas de declarações. – Este perfil descreve a forma de usar o protocolo a aplicar aos pedidos/respostas das declarações SAML, através de protocolos síncronos (ex.: SOAP).

- Correspondência de nomes dos identificadores. – Este perfil descreve a forma como deve ser usado o protocolo de correspondência de identificadores.
- Atributos. – Neste perfil são definidas as formas de representar os atributos nas declarações SAML, tendo como base repositórios comuns de atributos (directórios LDAP ou X.500, sistemas de gestão de bases de dados relacionais (RDBMS), *Active Directory* (AD), etc.).

WS-Security

O *standard* WS-Security foi proposto inicialmente pela IBM, Microsoft e Verisign em 2002, submetido em 2003 como sendo um *standard* OASIS e, em 2004, publicado como *standard* OASIS. Este *standard* permite fornecer segurança ao nível das mensagens SOAP. A protecção da integridade e confidencialidade das mensagens, e definição dos mecanismos utilizados na associação de *tokens* de segurança ao conteúdo das mesmas (tarefas relacionadas com a autenticação), são algumas das funcionalidades descritas neste *standard*. Adicionalmente, também são especificadas as formas de codificar os *tokens* de segurança (definição de uma *framework* para *tokens* baseados em XML), como incluir chaves de encriptação “opacas” (o utilizador da chave não precisa de a analisar), e mecanismos usados na descrição das características dos *tokens* incluídos nas mensagens.

Este *standard* propõe um grupo de extensões SOAP (SOAP11, SOAP12), de uma forma bastante flexível, e foi desenhado para ser usado como base para a segurança dos WS, em conjunto com uma grande variedade de modelos de segurança, tais como: PKI, *Kerberos* e SSL. Ele fornece suporte para múltiplos formatos de *tokens* de segurança, domínios de confiança, formatos de assinaturas e tecnologias de encriptação para garantir a integridade e confidencialidade. Comparando com a versão 1.0, WS-Security 1.1 introduziu novas funcionalidades através da inclusão de secções extra para *Kerberos*, SAML, SOAP com anexos e REL (*Rights Expression Language*).

A vantagem de usar WS-Security em vez de SSL é o facto de ser possível fornecer segurança ao nível da mensagem “*end-to-end*”. Ou seja, a mensagem é protegida mesmo nos casos em que precisa de passar por vários intermediários (*proxies*, *gateways*, balanceadores de carga, sistemas em *outsourcing*, etc.) pois, a segurança do seu conteúdo é garantida desde o ponto de origem até ao ponto de destino (“*end-to-end*”) e não só a nível da segurança da camada de transporte (ver Figura A. 2 e Figura A. 3). Desta forma, já não será necessário que o destinatário da mensagem tenha de confiar totalmente na avaliação a nível de segurança efectuada pelo intermediário. Adicionalmente, WS-Security é independente da camada de transporte do protocolo de comunicação, podendo ser usada por qualquer SOA, ou seja, não só para SOAP sobre HTTP.

Além disso, enquanto que usando SSL e TLS toda a mensagem é encriptada (Figura A. 2), com o WS-Security (Figura A. 3) é possível aplicar a encriptação de uma forma selectiva (só aplicar a determinadas partes da mensagem). Desta forma, por exemplo, se forem utilizadas *firewalls* aplicacionais na infra-estrutura, estas podem inspecionar as partes da mensagem não encriptadas.

É de salientar que não faz parte dos objectivos desta especificação a definição de mecanismos de autenticação e estabelecimento de contextos de segurança, derivação de chaves, publicação e troca de políticas de segurança, estabelecimento de relações de confiança e garantia de “não-repúdio”.



Figura A. 2: Configuração ponto-a-ponto (SSL, ...)

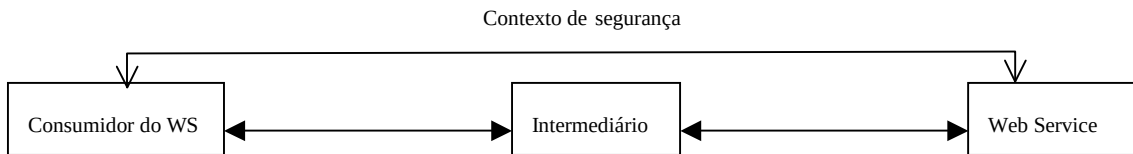


Figura A. 3: Configuração *end-to-end* (WS-Security)

O *standard* WS-Security define o uso de assinaturas e encriptação XML:

- A integridade das mensagens é implementada pela assinatura XML (XML-Signature) em conjunto com os *tokens* de segurança para garantir que, caso as mensagens sejam alteradas, sejam detectadas essas alterações. Estes mecanismos suportam múltiplas assinaturas, criadas por múltiplos *actors/roles* SOAP e são suficientemente extensíveis de forma a poderem vir a suportar formatos adicionais de assinaturas. A própria assinatura da mensagem é usada para verificar, além da integridade da mesma, também a sua origem através da verificação das declarações de um determinado *token* de segurança. Ou seja, se essas declarações se aplicam à entidade que originou essa mensagem.
- A confidencialidade das mensagens é alcançada através da encriptação XML (XML-Encryption) em conjunto com o uso de *tokens* de segurança, de forma a manter a confidencialidade das mensagens SOAP. Estes mecanismos também devem ser criados para suportar múltiplos processos de encriptação e serem operados por múltiplos *actors/roles* SOAP. Este *standard* permite a encriptação de qualquer uma das combinações de: corpo da mensagem, cabeçalho da mensagem e as suas subestruturas. A encriptação pode ser efectuada recorrendo tanto a chaves simétricas partilhadas pela origem e destino como a chaves contidas na mensagem numa forma encriptada.

Relativamente aos *tokens* de segurança, este *standard* divide os *tokens* de segurança em vários tipos e define como é que eles são anexados às mensagens. Portanto, podemos considerar os seguintes tipos de *tokens* de segurança:

- UsernameToken – É o tipo de *token* mais básico do WS-Security. Consiste numa descrição XML do *username* usado nas declarações do serviço que representa. Como, na sua forma básica, estes *tokens* não são assinados, são considerados uma opção “fraca” para ser utilizada na protecção das mensagens.

```

<S11:Envelope xmlns:S11="..." xmlns:wsse="...">
  <S11:Header>
    ...
    <wsse:Security>
      <wsse:UsernameToken>
        <wsse:Username>Sonia</wsse:Username>
      </wsse:UsernameToken>
    </wsse:Security>
  </S11:Header>
  ...
</S11:Envelope>

```

```

...
</S11:Header>
...
</S11:Envelope>

```

Para tornar uma solução que use este tipo de *tokens* mais segura, é possível assiná-los como sendo parte da mensagem e introduzir uma *password*, tanto na forma de *password* em texto claro (opção não muito segura) como na forma de “*password digest*”.

```

<wsse:Security>
  <wsse:UsernameToken>
    <wsse:Username>Sonia</wsse:Username>
    <wsse:Password Type="...#PasswordDigest">adsfhfg</wsse:Password>
  </wsse:UsernameToken>
</wsse:Security>

```

- **BinarySecurityToken** (certificados X.509 e *tickets Kerberos*) – Este tipo de *tokens* são declarados e assinados criptograficamente por uma determinada autoridade. São *tokens* de segurança binários que, primeiro são codificados como binários e depois são representados na forma de documentos XML que são passados entre os serviços. Este tipo de *tokens* permite a integração dos sistemas de gestão de identidades e acessos (ex.: PKI, LDAP, *Active Directory*) nas aplicações SOA.

```

<wsse:BinarySecurityToken wsu:Id=...
  EncodingType=...
  ValueType=.../>

```

ValueType – atributo que indica qual o tipo de *token* (ex.: *Kerberos*, *X509Token*)

EncodingType – define a forma como o *token* de segurança é codificado (ex.: *Base64Binary*)

- **Tokens de segurança XML (SAML)** – Tanto o WS-Security como o SAML fornecem soluções semelhantes em termos de segurança dos WS e, em alguns casos, um pode substituir o outro. WS-Security é capaz de considerar SAML como sendo um tipo de *token* de segurança XML usado na autenticação, autorização e declaração de atributos.

```

<wsse:Security xmlns:wsse="...">
  <saml:Assertion xmlns:saml="..." IssueInstant="2008-01-01T14:28:31.023Z">
    <saml:Issuer>http://exSAMLprovider/</saml:Issuer>
    <saml:AuthenticationStatement>
      <saml:Subject>
        <saml:NameIdentifier NameQualifier="exdominio" >
          uid=ExServiceUser,ou=exOU ,o=exEmpresa
        </saml:NameIdentifier>
        <saml:SubjectConfirmation>
          <saml:ConfirmationMethod>
            urn:oasis:names:tc:SAML:1.0:cm:bearer
          </saml:ConfirmationMethod>
        </saml:SubjectConfirmation>
      </saml:Subject>
    </saml:AuthenticationStatement>
  </saml:Assertion>
</wsse:Security>

```

Em alguns casos, é necessário que o *token* incluído no cabeçalho do `<wsse:Security>` seja encriptado. Nesses casos é utilizado o elemento `<xenc:EncryptedData>`.

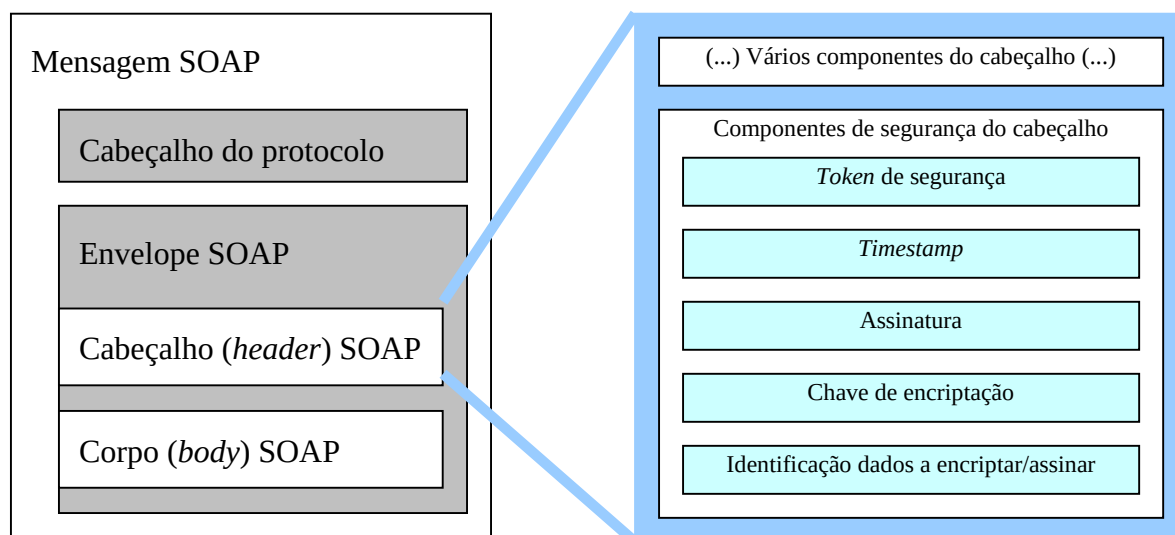


Figura A. 4: Componentes de segurança WS-Security de uma mensagem SOAP

O *standard* WS-Security também define o uso de *timestamps* de segurança de forma a permitir que o receptor da mensagem consiga determinar a “antiguidade” das semânticas de segurança. Muitas vezes, ao detectar que a semântica de segurança já é bastante antiga, pode mesmo decidir ignorá-la. Um dos objectivos do uso do *timestamps* da mensagem no WS-Security é a possibilidade de, através deste campo, poder proteger contra “*replay attacks*”. É preciso ter atenção que está fora do âmbito deste *standard* a descrição dos mecanismos de sincronização da hora.

```
<wsu:Timestamp wsu:Id="...">
  <wsu:Created ValueType="...">...</wsu:Created>
  <wsu:Expires ValueType="...">...</wsu:Expires>
  ...
</wsu:Timestamp>
```

Antes de analisarmos um exemplo, será melhor explicar alguns itens deste *standard* (Figura A. 4):

- As primeiras linhas são relativas à definição do “envelope” da mensagem SOAP. Esta é constituída pelo cabeçalho (*header*) e pelo corpo da mensagem (*body*).
- No cabeçalho, a secção `<wsse:Security>` é utilizada para passar para o receptor da mensagem a informação relativa à segurança da mensagem (passos necessários para a assinatura e encriptação da mensagem). Ou seja, é o mecanismo que permite anexar informação de segurança destinada a determinado destinatário na forma de *actor/role* SOAP. No exemplo seguinte, como não foi especificado o *actor/role* (`S12:actor` ou `S12:role`), esta informação poderá ser processada por qualquer um mas, não poderá ser removida antes de chegar ao seu destino final.
- Na secção `<wsu:Timestamp>` estão os dados relativos ao *timestamp*.
- Na secção `<wsse:BinarySecurityToken ...>` é especificado o *token* criado para a mensagem em questão (no exemplo, é referenciado como sendo “IDdoMeuToken”).
- Na secção `<xenc:EncryptedKey >` é especificada a chave utilizada para encriptar a mensagem. É utilizado o *standard* XML-Encryption declarado no DS *namespace* (“`xmlns:xenc= http://www.w3.org/2001/04/xmenc#`”), definido na linha relativa às características do “envelope” SOAP.
- Na secção `<xenc:DataReference>` são definidas as partes da mensagem que serão encriptadas.

- Na secção <ds:Signature> é especificada a assinatura digital que irá garantir a integridade dos elementos assinados. A assinatura utiliza a especificação XML-Signature declarada no DS *namespace* (“xmlns:ds=<http://www.w3.org/2000/09/xmldsig#>”), definido na linha relativa às características do “envelope” SOAP.
- Na secção <ds:SignedInfo> são definidas as partes da mensagem que serão assinadas (ds:Reference URI=...) e os métodos utilizados (ds:CanonicalizationMethod e ds:SignatureMethod).
- Na secção <ds:KeyInfo> é disponibilizada a informação relativa a como encontrar o *token* de segurança associado à assinatura a utilizar (<wsse:SecurityTokenReference>).

No exemplo seguinte é usado um *token* de segurança, assinatura, encriptação e *timestamp*. O *token* é do tipo certificado X.509 e contém dados binários codificados em base64. O corpo da mensagem é encriptado recorrendo a uma chave simétrica que é passada de uma forma encriptada. O corpo da mensagem é assinado usando uma chave criada através do recurso ao algoritmo RSA-SHA1.

```
<?xml version="1.0" encoding="utf-8"?>
<S11:Envelope xmlns:S11="..." xmlns:wsse="..." xmlns:wsu="..." xmlns:xenc="..." xmlns:ds="...">
  <S11:Header>
    <wsse:Security>
      <wsu:Timestamp wsu:Id="...">
        <wsu:Created ValueType="...">...</wsu:Created>
        <wsu:Expires ValueType="...">...</wsu:Expires>
        ...
      </wsu:Timestamp>
      <wsse:BinarySecurityToken ValueType=...#X509v3 "
EncodingType=...#Base64Binary" wsu:Id=" IDdoMeuToken ">
        HGhjGKJhgjh...
      </wsse:BinarySecurityToken>
      <xenc:EncryptedKey>
        <xenc:EncryptionMethod
Algorithm="http://www.w3.org/2001/04/xmenc#rsa-1_5"/>
        <ds:KeyInfo>
          <wsse:SecurityTokenReference>
            <wsse:KeyIdentifier
EncodingType=...#Base64Binary
ValueType=...#X509v3">JHJhjkHSHJ...
            </wsse:KeyIdentifier>
          </wsse:SecurityTokenReference>
        </ds:KeyInfo>
        <xenc:CipherData>
          <xenc:CipherValue>0ASDFdf...
          </xenc:CipherValue>
        </xenc:CipherData>
        <xenc:ReferenceList>
          <xenc:DataReference URI="#CorpoEncriptar"/>
        </xenc:ReferenceList>
      </xenc:EncryptedKey>
    <ds:Signature ID="MinhaAssinatura">
      <ds:SignedInfo>
        <ds:CanonicalizationMethod
Algorithm="http://www.w3.org/2001/10/xml-exc-c14n#"/>
        <ds:SignatureMethod
Algorithm="http://www.w3.org/2000/09/xmldsig#rsa-sha1"/>
        <ds:Reference URI="#CorpoMsg">
```

```

        <ds:DigestMethod
        Algorithm="http://www.w3.org/2000/09/xmldsig#sha1
        "/>
        <ds:DigestValue>kjklAJKHSJD...</ds:DigestValue>
    </ds:Reference>
</ds:SignedInfo>
<ds:SignatureValue>DJbchm5gK...</ds:SignatureValue>
<ds:KeyInfo>
    <wsse:SecurityTokenReference>
        <wsse:Reference URI="#IDdoMeuToken"/>
    </wsse:SecurityTokenReference>
</ds:KeyInfo>
</ds:Signature>
</wsse:Security>
</S11:Header>
<S11:Body wsu:Id="CorpoMsg">
    <xenc:EncryptedData Type=http://www.w3.org/2001/04/xmenc#Element
    wsu:Id="CorpoEncriptar">
        <xenc:EncryptionMethod
        Algorithm="http://www.w3.org/2001/04/xmenc#tripleDES-cbc"/>
        <xenc:CipherData>
            <xenc:CipherValue>JKHJKhkhjke...</xenc:CipherValue>
        </xenc:CipherData>
    </xenc:EncryptedData>
</S11:Body>
</S11:Envelope>

```

Só a aplicação do *standard* WS-Security não é garantia de implementação dos WS de uma forma totalmente segura. É necessário que, ao serem seguidas estas normas, os resultados obtidos não sejam vulneráveis a outros ataques. Além da utilização destas normas, deverão ser tomadas outras medidas de forma a minimizar os estragos decorrentes de possíveis ataques, nomeadamente, deverão ser também utilizados outros *standards* WS-* que estão actualmente disponíveis.

WS-SecureConversation

O *standard* WS-SecureConversation define um conjunto de extensões ao WS-Security de forma a fornecer mecanismos de estabelecimento e partilha de contextos de segurança e derivação das chaves de sessão (chaves mais eficientes ou troca de novos parâmetros associados às chaves). A necessidade da criação deste *standard* deveu-se ao facto da utilização do WS-Security ter o inconveniente de usar algoritmos de encriptação assimétricos, que são computacionalmente bastante intensivos. Além disso, como uma das vantagens da utilização de SOAP é o facto de ser um protocolo *stateless* (ou seja, simples e em que o consumidor só precisa de trocar uma mensagem com o serviço), as mensagens não são relacionadas com as mensagens anteriores (não existe o conceito de grupos de mensagens relacionadas). Como não existe esse conceito de grupo de mensagens pertencentes à mesma conversação (contexto), existem operações que têm de ser consequentemente repetidas.

O contexto de segurança é uma “abstracção” referente a um estado de autenticação já estabelecido e às chaves negociadas. Com o uso de WS-SecurityConversation é possível aumentar a *performance* e segurança nas trocas seguintes das chaves (como o WS-Security é baseado no modelo de autenticação da mensagem, muitas vezes está vulnerável a determinadas formas de ataques). Através do uso de contextos de segurança, é possível

aumentar a *performance* pois estes permitem mitigar o *overhead* envolvido na troca de múltiplas mensagens. O contexto é definido como sendo um novo tipo de *token* do WS-Security que é obtido através da aplicação do WS-Trust. Ou seja, WS-SecureConversation é criado com base no WS-Security e WS-Trust: usa os tipos de *tokens* do WS-Security e podem ser associados os SCT (*Security Context Token*) para otimizar as trocas complexas de *tokens* de segurança. Os principais objectivos desta especificação são os seguintes:

- Definir a forma como os contextos de segurança são estabelecidos.
- Definir a forma como os contextos de segurança são rectificados.
- Especificar como é que as chaves derivadas são calculadas e distribuídas.

Devido às optimizações efectuadas recorrendo ao WS-SecureConversation, é recomendável aplicar este *standard* em ambientes onde estão envolvidas grandes operações relacionadas com transacções complexas ou que necessitem de bastantes recursos. WS-SecureConversation permite que as múltiplas mensagens dos WS sejam trocadas usando o mesmo contexto, deixando de ser necessário criar um novo contexto de segurança por cada mensagem (ex.: uma assinatura pode ser validada para estabelecer o contexto e esse contexto é criado e válido por um determinado período de tempo ou número de mensagens).

Token do contexto de segurança (SCT – Security Context Token)

Enquanto que a autenticação das mensagens é suficiente para as situações em que estão envolvidas mensagens simples ou mensagens enviadas só num sentido, normalmente as entidades que necessitam de trocar múltiplas mensagens, estabelecem um contexto de segurança no qual elas são trocadas. Esse contexto de segurança é partilhado pelas duas entidades enquanto a sessão da comunicação estiver activa.

No *standard* WS-SecureConversation, o contexto de segurança é representado pelo *token* de segurança <wsc:SecurityContextToken>. Após o estabelecimento do contexto de segurança e do *secret* (autenticação efectuada), podem ser calculadas as respectivas derivações para cada uma das chaves usadas nesse contexto (chaves derivadas).

```
<S11:Envelope xmlns:S11="..." xmlns:ds="..." xmlns:wsse="..." xmlns:wsu="..." xmlns:wsc="...">
  <S11:Header> (...)
    <wsse:Security>
      <wsc:SecurityContextToken wsu:Id="IdMeuToken" xmlns:wsc="..." xmlns:wsu="..." ...>
        <wsc:Identifier>...</wsc:Identifier>
        <wsc:Instance>...</wsc:Instance> (...)
      </wsc:SecurityContextToken>
      <ds:Signature> (...)
        <ds:KeyInfo>
          <wsse:SecurityTokenReference>
            <wsse:Reference URI="#IdMeuToken" />
          </wsse:SecurityTokenReference>
        </ds:KeyInfo>
      </ds:Signature>
    </wsse:Security>
  </S11:Header>
  <S11:Body wsu:Id="BodyDaMensagem"> (...)
</S11:Body>
</S11:Envelope>
```

<wsc:Identifier> – Identificação do contexto de segurança através de um URI.

<wsc:Instance> – Quando os contextos são renovados e são utilizadas chaves diferentes, é necessário identificar as instâncias das diferentes chaves sem ser revelada a chave actual. Este é um elemento opcional que não deve existir no contexto inicial e que deve estar presente nos contextos subsequentes.

É recomendado que o WS, perante elementos ou atributos que não sejam entendidos, retorne um valor correspondente à indicação de uma situação de “falha”. Esse comportamento é especificado na política do serviço (WS-Policy e WS-PolicyAttachment). É necessário ter cuidado ao acrescentar informação aos *tokens* pois é preciso assegurar que as entidades envolvidas conseguem garantir que a informação não foi alterada (a definição do SCT não exige uma forma de proteger o seu conteúdo).

Estabelecimento dos contextos de segurança

Um contexto de segurança, após ser criado e antes de ser usado, precisa de ser partilhado pelas partes intervenientes na comunicação. O *standard* WS-SecureConversation define três formas de estabelecer entre as partes, de uma forma segura, contextos de segurança:

- SCT criado por um serviço de *tokens* de segurança. – O iniciador do contexto de segurança pede a um serviço de *tokens* de segurança para criar um novo SCT. Esse SCT é distribuído aos participantes através de um mecanismo definido no WS-SecureConversation e no WS-Trust.

Ex.: O iniciador do contexto envia um pedido <wst:RequestSecurityToken> para o serviço de *tokens* de segurança. Em seguida é retornada uma <wst:RequestSecurityTokenResponseCollection> contendo a <wst:RequestSecurityTokenResponse>. Nesta resposta estão incluídos o SCT <wst:RequestedSecurityToken> (ou uma referência para ele) e um apontador para o *secret* usado no contexto que foi retornado. Em seguida, é usado o SCT para tornar “seguras” as mensagens relativas a esse serviço.

- SCT criado por uma das partes e propagado com a mensagem. – O iniciador do contexto de segurança cria um SCT e distribui-o para todos os participantes através de um mecanismo definido no WS-SecureConversation e no WS-Trust. Este modelo é usado nas situações em que a entidade que cria o SCT é considerada como sendo de confiança para a criação dos mesmos. Neste cenário, a entidade que inicia o contexto de segurança cria um SCT e emite, para os outros intervenientes, uma <wst:RequestSecurityTokenResponse> assinada, sem ser previamente solicitada. A <wst:RequestSecurityTokenResponse> está incluída no corpo da mensagem ou num bloco do cabeçalho.
- SCT criado através de negociações/trocas. – Quando é necessário negociar ou participar numa sequência de troca de mensagens entre participantes relativas a SCT, tais como “*shared secret*”, WS-SecureConversation permite que as partes troquem informação de forma a estabelecer um contexto de segurança. Neste caso, a entidade que inicia o processo envia um pedido <wst:RequestSecurityToken> para a outra parte, sendo retornado uma resposta <wst:RequestSecurityTokenResponse>.

Rectificação dos contextos

Quando um SCT é criado, é-lhe associado um conjunto de declarações. Como por vezes é necessário rectificar o SCT de forma a acrescentar algumas declarações adicionais, pode ser efectuado um pedido explícito de forma a rectificar as declarações associadas ao SCT em questão. Só após a prova da posse das chaves associadas ao contexto é que este poderá ser rectificado. É aconselhável que esta prova de posse seja efectuada através da criação de uma assinatura sobre o corpo da mensagem e chaves do cabeçalho, através do uso da chave associada ao contexto de segurança.

O pedido das rectificações do SCT é efectuado através de mecanismos definidos no WS-Trust:

```
<S11:Envelope xmlns:S11="..." xmlns:wsse="..." xmlns:wsu="..." xmlns:wst="..." xmlns:wsc="...">
  <S11:Header>
    (...)
    <wsa:Action xmlns:wsa="...">
      http://docs.oasis-open.org/ws-sx/ws-trust/200512/RST/SCT/Amend
    </wsa:Action>
    (...)
  </S11:Header>
  (...)
</S11:Envelope>
```

A resposta é representada da seguinte forma:

```
<S11:Envelope xmlns:S11="..." xmlns:wsse="..." xmlns:wsu="..." xmlns:wst="..." xmlns:wsc="...">
  <S11:Header>
    (...)
    <wsa:Action xmlns:wsa="...">
      http://docs.oasis-open.org/ws-sx/ws-trust/200512/RSTR/SCT/Amend
    </wsa:Action>
    (...)
  </S11:Header>
  (...)
</S11:Envelope>
```

Renovação dos contextos

Normalmente, quando um SCT é criado, também lhe é associada uma data de expiração. Se for necessário prolongar a data de validade do *token*, também se pode recorrer a mecanismos definidos no WS-Trust:

```
http://docs.oasis-open.org/ws-sx/ws-trust/200512/RST/SCT/Renew
http://docs.oasis-open.org/ws-sx/ws-trust/200512/RSTR/SCT/Renew
```

A renovação deve requerer novamente a autenticação das declarações originais pois, estas podem ter uma data de expiração que entra em conflito com a data de expiração do pedido de renovação do contexto. Também é necessário que seja efectuada a prova de posse da chave associada ao contexto de segurança.

Cancelamento dos contextos

Se for necessário cancelar o contexto de segurança, podem ser usados os mecanismos definidos no WS-Trust:

```
http://docs.oasis-open.org/ws-sx/ws-trust/200512/RST/SCT/Cancel
http://docs.oasis-open.org/ws-sx/ws-trust/200512/RSTR/SCT/Cancel
```

Também, neste caso, será necessário provar a posse da chave associada ao contexto. Após o cancelamento de um contexto não será possível utilizá-lo para tarefas de autenticação ou autorização e deixará de ser possível renová-lo.

Derivação das chaves

Um SCT inclui ou contém uma “*shared secret*”. Esta “*shared secret*” deve ser usada para assinar e/ou encriptar as mensagens mas, é recomendado que as chaves derivadas sejam usadas para assinar e encriptar só as mensagens associadas ao contexto de segurança.

Através do uso de uma “*shared secret*” comum, cada uma das partes intervenientes pode definir diferentes chaves derivadas. Desta forma, cada uma das partes pode utilizar chaves diferentes nos processos de assinatura e encriptação. De forma a manter as chaves actualizadas, ainda “dentro do período de validade”, evitando fornecer informação a mais para análise, deverão ser usadas derivações sucessivas. Esta especificação define o *token* <wsc:DerivedKeyToken> como sendo um mecanismo para indicar qual a derivação que está a ser utilizada em cada uma das mensagens.

No mecanismo de derivação das chaves podem ser usados vários algoritmos. O *standard* WS-SecureConversation define um desses algoritmos baseados num subconjunto do mecanismo definido para TLS no RFC 2246: usa a função P_SHA-1 para gerar a sequência de *bytes* que será usada para gerar as chaves de segurança. Esta função tem três argumentos:

- *secret* – “*shared secret*” que é trocada;
- *label* – concatenação do identificador do cliente e do identificador do serviço;
- *seed* – concatenação dos valores *nonce* (*number used once*) que foram trocados entre o emissor e o receptor das mensagens.

Sintaxe do *token* <wsc:DerivedKeyToken>:

```
<wsc:DerivedKeyToken wsu:Id="..." Algorithm="..." xmlns:wsc="..." xmlns:wsse="..." xmlns:wsu="...">
  <wsse:SecurityTokenReference>...</wsse:SecurityTokenReference>
  <wsc:Properties>...</wsc:Properties>
  <wsc:Generation>...</wsc:Generation>
  <wsc:Offset>...</wsc:Offset>
  <wsc:Length>...</wsc:Length>
  <wsc:Label>...</wsc:Label>
  <wsc:Nonce>...</wsc:Nonce>
</wsc:DerivedKeyToken>
```

WS-Trust

O objectivo do *standard* WS-Trust é permitir criar modelos de confiança que permitam que os WS interajam, de uma forma segura, através da troca de mensagens SOAP. Ou seja, esta especificação é a responsável pela gestão e estabelecimento de relações de confiança. Para tal, o WS-Trust usa os mecanismos básicos de segurança de mensagens definido no WS-Security para definir primitivas e extensões adicionais usadas na emissão, troca e validação dos *tokens* de segurança necessários para o estabelecimento de relações de confiança entre domínios diferentes. A troca de credenciais de segurança pode ser efectuada directamente pelas próprias entidades ou através de um intermediário. Em ambos os casos, cada uma delas precisa de determinar previamente se pode ou não confiar nas

credenciais da outra parte. Desta forma, as aplicações podem usar estas extensões para implementarem alguma segurança adicional aos WS, nomeadamente, nos serviços de descrição WSDL, UDDI e “bindingTemplates” e nas mensagens [SOAP] e [SOAP2].

O modelo dos WS definido no WS-Trust é baseado num processo em que o WS pode exigir que, relativamente às mensagens que lhe são enviadas, lhe seja apresentado um conjunto de declarações de acordo com o definido nos WS-Policy e WS-PolicyAttachment (ex.: nome, chave, permissões, competências, etc.). Se estas não forem apresentadas, o WS deve ignorar ou rejeitar a mensagem.

A autenticação dos pedidos é baseada na combinação da segurança ao nível das camadas de rede e transporte (IPSec ou TLS/SSL) e na demonstração das declarações mencionadas no parágrafo anterior. A autenticação do consumidor do WS é efectuada através da autenticação dos pedidos e na encriptação dos mesmos usando uma chave conhecida pelo próprio consumidor.

Uma forma de demonstrar que se está autorizado a utilizar um determinado *token* de segurança é incluir uma assinatura digital usando a chave secreta que lhe está associada (prova da posse do *token*). Desta forma, o consumidor pode provar que um conjunto de declarações lhe pertence, associando os *tokens* de segurança (ex.: PKI, certificados X.509, etc.) às respectivas mensagens. De acordo com o referido na política do serviço, se o consumidor não tiver os *tokens* necessários para provar perante o serviço as suas declarações, podem ser contactadas as autoridades apropriadas e serem solicitados os *tokens* necessários para efectuar a respectiva prova das declarações. Essas autoridades, os STS (*Security Token Services*) podem, por sua vez, solicitar o seu próprio conjunto de declarações para autenticar e autorizar o pedido dos *tokens* de segurança. Os STS formam a base de confiança através da emissão de *tokens* de segurança que podem ser usados para negociar as relações de confiança entre os diferentes domínios.

O WS-Trust também define um mecanismo para a troca de mensagens durante o processo de obtenção dos *tokens* de segurança, garantindo que as mensagens não estão desactualizadas e verificando a autorização do uso do respectivo *token*.

Este modelo de segurança (declarações, políticas e *tokens* de segurança) inclui e suporta modelos mais específicos, tais como: autorização baseada na identidade, listas de controlo de acessos e autorizações baseadas nas competências. Desta forma, é possível utilizar algumas das tecnologias já existentes: certificados de chave pública X.509, *tokens* baseados em XML, *tickets kerberos* e mesmo *username/password*. A combinação deste modelo com as primitivas dos WS-Security e WS-Policy é suficiente para construir mecanismos de troca de chaves, autenticação, controlo de acessos baseado em políticas, auditoria e relações de confiança complexas. Ou seja, pelo menos, o WS deve ter uma política aplicada, receber uma mensagem do consumidor que inclua *tokens* de segurança e ter implementada alguma segurança através da aplicação de mecanismos baseados em WS-Security. Seguindo esta especificação, o WS pode processar os pedidos efectuados pelo consumidor já autorizado, se forem verificadas as seguintes condições:

- As declarações incluídas no *token* forem suficientes para que o WS se sujeite à política e que a mensagem está de acordo com a política.
- Os atributos das declarações estarem demonstrados pelas assinaturas. – Em alguns modelos de confiança em que existem intermediários, a assinatura não verifica a identidade das declarações, mas sim a identidade dos intermediários que, por sua

vez, provam a identidade do consumidor. As declarações são provadas ou não com base na política.

- Os emissores dos *tokens* de segurança forem de confiança.

Existem alguns modelos para a obtenção de *tokens* e negociação das relações de confiança:

- Aquisição do *token* através de um pedido:
 - Cenário de relações de confiança directas. – Como parte da troca de mensagens, é pedido ao STS para trocar um *token* de segurança (este pedido pode ser efectuado pelo próprio consumidor ou por um intermediário em seu nome). Se o STS confiar nesse *token* de segurança e o pedido conseguir provar a posse desse *token* de segurança, a troca de mensagens é processada pelo próprio STS.
 - Cenário de relações de confiança delegadas. – Um intermediário efectua o pedido do *token* em nome do consumidor real. Neste caso, o STS responsável pela geração de *tokens* não precisa de confiar na autoridade que emitiu o *token*, que foi fornecido pelo consumidor original, pois ele confia no STS que está envolvido na troca do novo *token*. A base da “confiança” é a relação entre os dois STS.
- Aquisição do *token* sem ser explicitamente pedido. – O *token* é recebido sem ser na forma de uma resposta directa a um pedido SOAP. Nestas situações, as possíveis formas de aquisição do *token* são as seguintes:
 - Uma autoridade envia o *token* para a outra entidade sem este ter sido explicitamente pedido.
 - O *token* pode ser obtido como fazendo parte de um protocolo estabelecido com uma terceira entidade.
 - Por herança.
- “*Trust Bootstrap*”. – Neste caso, um administrador, ou outra autoridade de confiança, definem que todos os *tokens* de um determinado tipo são de confiança. Seguindo estas directrizes, o STS pode comunicá-las aos intervenientes do processo de decisão de forma a estes poderem tomar as suas próprias decisões ou disponibilizar as suas próprias funções, na forma de um serviço, que pode ser utilizado pelos outros serviços nas tarefas de estabelecimento de relações de confiança. Existe vários métodos que podem ser usados para “*trust bootstrap*” de um serviço:
 - Origens de confiança fixas. – Método mais simples, em que existe já definido um conjunto de relações de confiança. Os pedidos são todos avaliados de forma a verificar se contêm *tokens* de segurança pertencentes a uma dessas origens.
 - Hierarquias de confiança. – Através de um mecanismo de origens de confiança, um serviço pode escolher se deve ou não permitir hierarquias de relações de confiança tão extensas como a eventual cadeia de confiança que o conduzirá à origem do *token* em questão.

- Serviço de autenticação. – Este método pode ser considerado como sendo baseado no de “origens de confiança fixas”, em que a entidade só confia no serviço de autenticação. Ou seja, ela reencaminha todos os *tokens* para o serviço de autenticação e este, por sua vez, responde com uma declaração relativa à sua autorização.

Em seguida está um exemplo da representação de uma estrutura com os elementos básicos para a emissão dos *tokens* de segurança:

RequestSecurityToken

```
<S11:Envelope xmlns:S11="..." xmlns:wsu="..." xmlns:wsse="..." xmlns:xenc="..."
  xmlns:wst="...">
  <S11:Header>
    (...)
    <wsse:Security> (...)</wsse:Security>
  </S11:Header>
  <S11:Body wsu:Id="req">
    <wst:RequestSecurityToken Context="..." xmlns:wst="...">
      <wst:TokenType>...</wst:TokenType>
      <wst:RequestType>...</wst:RequestType>
      <wst:SecondaryParameters>...</wst:SecondaryParameters>
      (...)
      <wsp:AppliesTo>...</wsp:AppliesTo>
      <wst:Claims Dialect="...">...</wst:Claims>
      <wst:Entropy>
        <wst:BinarySecret>...</wst:BinarySecret>
      </wst:Entropy>
      <wst:Lifetime>
        <wsu:Created>...</wsu:Created>
        <wsu:Expires>...</wsu:Expires>
      </wst:Lifetime>
    </wst:RequestSecurityToken>
  </S11:Body>
</S11:Envelope>
```

RequestSecurityTokenResponse

```
<wst:RequestSecurityTokenResponse Context="..." xmlns:wst="...">
  <wst:TokenType>...</wst:TokenType>
  <wst:RequestedSecurityToken>...</wst:RequestedSecurityToken>
  (...)
  <wsp:AppliesTo xmlns:wsp="...">...</wsp:AppliesTo>
  <wst:RequestedAttachedReference> (...) </wst:RequestedAttachedReference>
  <wst:RequestedUnattachedReference> (...)</wst:RequestedUnattachedReference>
  <wst:RequestedProofToken>(...)</wst:RequestedProofToken>
  <wst:Entropy>
    <wst:BinarySecret>...</wst:BinarySecret>
  </wst:Entropy>
  <wst:Lifetime>...</wst:Lifetime>
</wst:RequestSecurityTokenResponse>
```

O envio de pedido de emissão do *token*, por parte da entidade solicitante, está descrito na secção <wst:RequestSecurityToken> (normalmente este pedido deve ser assinado pelo solicitante através de *tokens* contidos/incluídos no próprio pedido). Se a política permitir e se os requisitos do receptor forem respeitados, então o solicitante recebe uma resposta (<wst:RequestSecurityTokenResponse>).

O URI opcional “Context” especifica o identificador/contexto do pedido que pode ser utilizado para correlacionar os pedidos e respostas seguintes. O elemento `<wst:TokenType>` permite descrever o tipo de *token* de segurança cuja emissão está a ser pedida e que deverá ser retornado. O elemento `<wst:RequestType>` é usado para indicar, através de um URI, a classe da função que está a ser pedida. O elemento opcional `<wst:SecondaryParameters>` é formado por vários `<wst:RequestSecurityToken>` (sem outros `<wst:SecondaryParameters>`) e é usado nos casos em que existem entidades intermediárias no processo de obtenção do *token* de segurança.

Normalmente, o *token* retornado na `<wst:RequestSecurityTokenResponse>` deverá ser considerado “opaco” para o solicitante. Ou seja, o solicitante não deve precisar de analisar o *token* que foi retornado. Na resposta pode também estar incluída informação específica que tenha sido solicitada, como por exemplo, a data de validade do *token*.

Todos estes elementos são passados como fazendo parte do *payload* do WSDL, sendo implementados pelo STS.

Normalmente, é recomendado que as origens dos pedidos sejam autenticadas. No entanto, nos casos em que os pedidos são anónimos, deverá ser a política do receptor que deverá determinar se o pedido deve ser aceite ou não.

Em alguns casos, os elementos podem incluir chaves que não são encriptadas. Como nesses casos `<xenc:EncryptedData>` não pode ser usado, pode ser usado o elemento `<ws:BinarySecret>`. Este último elemento só deve ser usado quando a mensagem já está protegida (ex.: é usada segurança ao nível do transporte ou o elemento a proteger já está encriptado).

A renovação do *token* de segurança pode ser solicitada através da inclusão do elemento `<wst:RenewTarget>`:

```
<wst:RequestSecurityToken xmlns:wst="...">
  <wst:TokenType>...</wst:TokenType>
  <wst:RequestType>...</wst:RequestType>
  (...)
  <wst:RenewTarget>...</wst:RenewTarget>
  <wst:AllowPostdating/>
  <wst:Renewing Allow="..." OK="...">/>
</wst:RequestSecurityToken>
```

Neste processo de renovação não devem ser alteradas as semânticas das chaves (tamanho, tipo, algoritmos, âmbito, etc.). A prova de autorização do uso do *token* deve ser novamente solicitada/verificada durante este processo de renovação.

A partir do momento em que o *token* já não é necessário, pode ser solicitado o seu cancelamento. Para que este pedido seja atendido, a entidade precisa também de provar que estava autorizada a utilizar esse *token*. A sintaxe do pedido de cancelamento é a seguinte:

```
<S11:Envelope xmlns:S11="..." xmlns:wst="..." xmlns:wsse="...">
  <S11:Header>
    <wsse:Security> (...) </wsse:Security>
  </S11:Header>
  <S11:Body>
    <wst:RequestSecurityToken xmlns:wst="...">
      <wst:RequestType>...</wst:RequestType>
      (...)
      <wst:CancelTarget>...</wst:CancelTarget>
    </wst:RequestSecurityToken>
```

```

    </S11:Body>
</S11:Envelope>

```

A sintaxe da resposta ao pedido de cancelamento é a seguinte:

```

<S11:Envelope xmlns:S11="..." xmlns:wst="..." xmlns:wsse="...">
  <S11:Header>
    <wsse:Security> (...) </wsse:Security>
  </S11:Header>
  <S11:Body>
    <wst:RequestSecurityTokenResponse>
      <wst:RequestedTokenCancelled/>
    </wst:RequestSecurityTokenResponse>
  </S11:Body>
</S11:Envelope>

```

O pedido de validação do *token* é efectuado recorrendo à definição do elemento `<wst:ValidateTarget>` e a resposta é definida no elemento `<wst:Status>`:

```

<wst:RequestSecurityToken xmlns:wst="...">
  <wst:TokenType> http://docs.oasis-open.org/ws-sx/ws-trust/200512/RSTR/Status
</wst:TokenType>
  <wst:RequestType> http://docs.oasis-open.org/ws-sx/ws-trust/200512/Validate
</wst:RequestType>
</wst:RequestSecurityToken>

<wst:RequestSecurityTokenResponse xmlns:wst="...">
  <wst:TokenType> http://docs.oasis-open.org/ws-sx/ws-trust/200512/RSTR/Status
</wst:TokenType>
  <wst:Status>
    <wst:Code> http://docs.oasis-open.org/ws-sx/ws-trust/200512/status/valid
    </wst:Code>
  </wst:Status>
  (...)
</wst:RequestSecurityTokenResponse>

```

WS-Federation

WS-Federation descreve a forma de gerir as negociações das relações de confiança num ambiente heterogéneo de federação, incluindo o suporte de identidades de federação (as funcionalidades de federação incluem as relações de confiança, *single sign-on*, *single sign-off* e gestão dos atributos entre um ambiente de federação). O WS-Federation faz parte da família de três *standards*: WS-Federation, WS-Federation *Passive Client* e WS-Federation *Active Client*.

O *standard* WS-Federation, baseado no WS-Trust, define os mecanismos para trocas e federação de relações de confiança, identidade e declarações. Através do WS-Federation, é possível implementar formas seguras de propagar, entre diferentes domínios de segurança, muitas vezes distribuídos pela *Internet*, informação relacionada com a identidade, atributos, autenticação e autorização. Baseando-se nos modelos definidos pelos WS-Security e WS-Trust, WS-Federation:

- Descreve a forma de gerir as negociações das relações de confiança e troca de *tokens* de segurança.
- Suporta a manutenção da privacidade através da “camuflagem” da informação relacionada com a identidade e atributos.

- Suporta a federação de *sign-out*.

WS-Federation baseia-se no WS-Policy e WS-MetadataExchange para descrever e obter *metadata*.

XKMS (XML Key Management Specification)

XKMS é um *standard* do W3C, baseado em XML, usado na gestão de PKI⁷ (*Public Key Infrastructure*). XKMS especifica protocolos para a distribuição e registo de chaves públicas de forma a utilizá-las em conjunto com os *standards* de assinaturas e encriptação XML.

O XKMS permite a delegação das tarefas de registo e validação de chaves para outros serviços/entidades de confiança, que sejam compatíveis com qualquer sistema PKI, e que possam ser acedidos através de SOAP e XML. Nesses casos, esses serviços/entidades ficam então responsáveis por passar a informação entre eles próprios e os WS. Com esta delegação, a implementação do próprio WS pode ser mantida bastante simples pois, as aplicações ou WS que suportem XKMS, não necessitam de implementar localmente soluções PKI. Nestes casos, eles enviam pedidos XML, em mensagens SOAP, para os componentes PKI instalados externamente (ex.: Verisign) que processam os pedidos XML (pedidos de emissão, retorno ou cancelamento dos certificados) em nome da aplicação ou WS originais.

WS-Reliability

WS-Reliability é um *standard* OASIS, baseada em SOAP, usada na troca de mensagens SOAP, que permite garantir a entrega, não duplicação e ordenação das mensagens. Este protocolo é definido como uma extensão ao cabeçalho SOAP e é independente dos protocolos base.

Como SOAP sobre HTTP não é suficiente quando um dos requisitos a nível aplicacional é que o protocolo usado na troca das mensagens seja de “confiança” e seguro, foi necessário criar este *standard* de forma a definir medidas de “confiança” no contexto dos *standards* WS existentes. Este *standard* foi criado para ser usado em conjunto com outros protocolos complementares (ex.: ebXML *Message Service*). Este *standard* também contém uma secção dedicada só ao *binding* http onde são definidos os elementos do cabeçalho SOAP que deverão ser incluídos nas mensagens que usam o HTTP como protocolo de transporte.

Este protocolo define as seguintes funcionalidades relacionadas com a confiança na troca de mensagens:

- AtLeastOnce – Garantia de entrega da mensagem (entrega uma ou mais vezes).
- At-Most-Once – Garantia da eliminação de mensagens duplicadas.
- ExactlyOnce – Garantia da entrega da mensagem e eliminação de duplicados.

⁷ *Public Key Infrastructure* – Um sistema que usa a criptografia baseada em chaves públicas para encriptar, assinar, autorizar e verificar a autenticação da informação na Internet.

- Num grupo de mensagens, garantir a entrega das mensagens por ordem.

Está fora do âmbito deste *standard*:

- Funcionalidades de encaminhamento das mensagens. – Este *standard* endereça a confiança da troca de mensagens ponto-a-ponto e não em situações em que estão envolvidos intermediários. O mecanismo descrito não especifica técnicas de encaminhamento mas pode ser usado em conjunto com essas regras.
- Transacções. – Mensagens transaccionais garantem a integridade dos *standards* de troca que, possivelmente, envolvem várias mensagens. As condições de falha devem envolver decisões a nível aplicacional baseadas na interpretação do *payload* das mensagens. O *standard* WS-Reliability tem como objectivo garantir a confiança da troca de mensagens individuais, ignorando qualquer interpretação dessas mensagens.
- A definição de limiares de QoS tais como: taxas de erro, latência média, medidas de quantitativas relacionadas com SLAs, etc..

WS-ReliableMessaging

Tal como o *standard* WS-Reliability, WS-ReliableMessaging foi criado para resolver as necessidades de garantia da confiança nas comunicações das soluções compostas por aplicações distribuídas por várias plataformas, muitas vezes heterogéneas. WS-ReliableMessaging foi criado em 2003 pelas empresas BEA Systems, Microsoft, IBM e Tibco. Durante os dois anos seguintes sofreu alguns melhoramentos e, em 2005, foi submetido ao Comité WS-RX da OASIS. A versão actual, WS-ReliableMessaging v1.1, foi aprovada como *standard* OASIS em 2007. Entretanto, antes da criação deste *standard*, já existia um *standard* “concorrente”, também aprovado pela OASIS: WS-Reliability. WS-Reliability era suportado por vários vendedores, nomeadamente: Fujitsu, Hitachi, NEC, Oracle Corporation, Progress Software e Sun Microsystems. Actualmente, a maior parte destes vendedores também já suportam o *standard* WS-ReliableMessaging.

Seguindo este *standard*, é possível a interacção entre plataformas de uma forma credível. Ou seja, é possível garantir que as mensagens enviadas são recebidas e a ordem na qual são processadas. As garantias de entrega que podem ser implementadas pelo *standard* WS-ReliableMessaging são as seguintes:

- AtMostOnce: Entrega sem duplicados mas não trata as mensagens perdidas.
- AtLeastOnce: Entrega uma vez, ou mais. Se não for entregue gera um erro.
- ExactlyOnce: Entregue só uma vez. Nos outros casos gera um erro.
- InOrder: Entregue na mesma sequência que foi enviada. Não trata mensagens duplicadas ou mensagens perdidas.

Para os casos em que ocorrem falhas SOAP, a garantia da entrega das mensagens deve ser assegurada pela implementação da solução e não pelo WS-ReliableMessaging. De forma a ser possível criar uma solução de troca de mensagens completa, WS-ReliableMessaging deve ser usado em conjunto com outros *standards* WS (WS-Security, WS-Policy, WS-Addressing) e protocolos específicos das próprias aplicações.

A definição da duração da mensagem desde que é enviada ou da disponibilidade do sistema, não fazem parte deste *standard*.

WS-Addressing

SOAP não permite especificar, de uma forma *standard*, qual o destino da mensagem ou como as respostas/falhas devem ser retornadas. WS-Addressing fornece uma *framework* XML para identificar os *end-points* dos WS e para garantir a segurança dessa informação de identificação, que é transportada nas mensagens que passam pelos vários intervenientes na troca das mesmas. É óbvio que o último intermediário precisa de saber localizar o *end-point* mas, para que a mensagem seja encaminhada até ele, é necessário preservar essa informação de forma a chegar a esse intermediário sem ter sido alvo de uma alteração propositada que inviabilize a sua entrega.

Como está representado no exemplo seguinte, WS-Addressing permite definir o destino (“*to*”) e acção (“*action*”) a ser executada:

```
<soapenv:Envelope ...>
  <soapenv:Header>
    <wsa:To xmlns:wsa="http://schemas.xmlsoap.org/ws/2004/08/addressing">
      http://localhost:8000/.../exemplo
    </wsa:To>
    <wsa:Action xmlns:wsa=".../ws/2004/08/addressing">
      </wsa:Action>
    ...
  </soapenv:Header>
</soapenv:Envelope>
```

WS-Addressing pode ser usado também para transmitir o endereço a ser usado nas respostas (“*reply-to*”), identificador da mensagem e correlação de identificadores de mensagens. O endereço de resposta é útil nas situações em que são utilizadas mensagens assíncronas como transporte SOAP. O identificador da mensagem e a sua correlação também são úteis nessas situações pois permitem responder a uma mensagem de uma forma assíncrona.

WS-Addressing é independente do modo de transporte. Por exemplo, o pedido pode ser efectuado por JMS e a resposta pode ser via HTTP. Como numa mensagem SOAP são usadas referências de acordo com o WS-Addressing, para representar os sistemas finais, o uso de WS-Addressing é essencial nos casos em que são usados protocolos diferentes de HTTP.

WS-Policy

WS-Policy descreve o modelo usado para definir as competências e restrições das políticas de segurança aplicadas aos consumidores e fornecedores dos WS e respectivos intermediários envolvidos (ex.: *tokens* de segurança, algoritmos de encriptação suportados, políticas de privacidade).

Através deste *standard*, é definida uma gramática flexível e extensível de forma a ser possível exprimir, para um sistema baseado em WS, as suas competências, os seus requisitos e as características gerais da entidade. O WS-Policy define uma *framework* e um modelo para exprimir estas propriedades, na forma de “políticas”, cujas expressões vão desde simples afirmações declarativas até afirmações mais complexas. Algumas das

afirmações especificam os tradicionais requisitos e competências que se manifestam durante a troca da mensagem (ex.: esquemas de autenticação, selecção do protocolo de transporte, encriptação e assinaturas definidas no WS-Security, etc.) e outras especificam os requisitos que, embora ainda não se manifestem nesse momento, são relevantes para a selecção e utilização do serviço (ex.: política de privacidade, características de QoS, etc.). Para auxiliar a compreensão da linguagem usada na WS-Policy, foi criado o *standard* WS-Policy *Primer* onde é efectuada uma descrição introdutória à linguagem utilizada para especificar as políticas WS-Policy, apresentando exemplos de como usar essas expressões.

As declarações constantes nas políticas estão relacionadas com alguns dos *standards* WS-*, incluindo WS-SecurityPolicy, declaração de políticas WS-ReliableMessaging e WS-Addressing WSDL *Binding*. Actualmente, existem três *standards* principais que definem as declarações WS-Policy:

- WS-SecurityPolicy define as declarações relativas à integridade, confidencialidade e informação sobre *tokens* de segurança.
- A política WS-ReliableMessaging define as declarações que podem ser usadas para especificar a forma como o WS usa WS-ReliableMessaging.
- “WS-Addressing WSDL *Binding*” define os elementos que podem ser usados na descrição WSDL para especificar o uso do WS-Addressing.

Exemplo de uma expressão WS-Policy:

```
<wsp:Policy xmlns:wsp=".../policy">
  <wsp:All>
    <wsap:UsingAddressing/>
    <sp:TransportBinding>
      <All>
        <sp:TransportToken>
          <Policy>
            <sp:HttpsToken RequireClientCertificate="true"/>
          </Policy>
        </sp:TransportToken>
        <sp:AlgorithmSuite>
          <Policy>
            <sp:Basic256Sha256Rsa15/>
          </Policy>
        </sp:AlgorithmSuite>
      </All>
    </sp:TransportBinding>
    <wsrm:RMAssertion>
      <All>
        <wsrm:InactivityTimeout Milliseconds="2000"/>
        <wsrm:BaseRetransmissionInterval Milliseconds="5000"/>
        <wsrm:ExponentialBckoff/>
      </All>
    </wsrm:RMAssertion>
  </wsp:All>
</wsp:Policy>
```

A secção <All> é usada para especificar que todas as declarações constantes dentro dessa secção devem ser cumpridas. Se quiséssemos que fosse só escolhida uma das opções declaradas, deveria ser usada a opção “ExactlyOne”.

A expressão “UsingAddressing” permite definir que o consumidor do WS deve incluir a informação WS-Addressing no cabeçalho SOAP. Neste exemplo, na secção

<TransportBinding> é definido que deve ser utilizado um certificado do cliente através de TLS, usado o algoritmo AES com chave de 256 bits para criptografia simétrica, para *hashing* deve ser usado HSA de 256 bits e na criptografia assimétrica RSA v.1.5. A secção <RMAssertion> pode ser usada para definir os parâmetros necessários quando é utilizado o WS-ReliableMessaging para garantir a entrega das mensagens. Neste exemplo, é definido um tempo máximo de inactividade de 2 segundos, o intervalo base de retransmissão de 5 segundos usando o algoritmo exponencial “*backoff*”.

Este *standard* é bastante flexível pois permite definir expressões de políticas dentro de outras expressões de políticas. Este nível de flexibilidade permite aos fornecedores dos WS definirem completamente os requisitos que os consumidores devem cumprir para além dos já descritos no WSDL. Como as expressões das políticas são externas à *metadata* guardada nos UDDI e WSDL, os fornecedores dos WS têm de confiar num mecanismo diferente destes para distribuir a informação relativa a WS-Policy: WS-MetadataExchange ou WS-PolicyAttachment. Uma das funções da especificação WS-MetadataExchange é definir um formato para encapsular *metadata* relativa aos WS (ex.: expressões WS-Policy). Através do WS-PolicyAttachment é possível definir a forma como as políticas são referenciadas nas definições do WSDL, como associar as políticas aos *end-points* e como associar políticas às entradas UDDI.

WS-SecurityPolicy

WS-SecurityPolicy é um *standard* definido pela OASIS que, tal como o WS-Trust e WS-SecureConversation, também faz parte do WS *Secure Exchange* (WS-SX). Este *standard* define um conjunto de declarações de políticas de segurança que podem ser usadas nos documentos WS-Policy para exprimir os requisitos, competências e restrições da implementação de um WS. Em particular, WS-SecurityPolicy permite definir normas para as declarações relacionadas com o uso de WS-Security, WS-Trust e WS-SecureConversation. As declarações do WS-SecurityPolicy descrevem a forma como se pode garantir a segurança das mensagens pertencentes a uma determinada “comunicação”. O objectivo é permitir algum nível de flexibilidade em termos de *tokens*, criptografia e mecanismos usados (incluindo segurança a nível do transporte) mas, garantindo algum nível de especificidade de forma a permitir a interoperabilidade com base nas declarações já definidas. Ou seja, deverá ser flexível o suficiente mas, cumprindo determinados limites relativos ao que está definido nas declarações associadas aos WS.

Conforme o nível em que são aplicadas, as declarações do WS-SecurityPolicy podem ser classificadas em quatro grupos:

- Ao nível do *end-point*:
 - Medidas/*standards* de segurança utilizados: *tokens* necessários, algoritmos e tamanho das chaves, disposição dos elementos no cabeçalho WS-Security, ordem da protecção implementada através do uso de encriptação e assinaturas, outros elementos que devem estar presentes no cabeçalho WS-Security (ex.: *timestamp*).
 - Níveis de conformidade com os padrões WS-Security e WS-Trust.
 - *Tokens* suportados pelo *end-point*.
- Ao nível da operação: *tokens* suportados.

- Ao nível da mensagem:
 - Assinatura e/ou encriptação de partes específicas da mensagem (assinar/encriptar totalmente o corpo da mensagem ou os cabeçalhos identificados através do *namespace* e, opcionalmente, nomes).
 - Assinatura e/ou encriptação de elementos da mensagem identificados através do uso de XPath.
 - Obrigatoriedade da existência de determinados elementos na mensagem.
- Declarações que não podem ser associadas directamente a um assunto, ou seja, declarações que só podem ser incluídas noutras declarações.

No exemplo seguinte está representada a forma de usar o WS-SecurityPolicy num cenário em que deverá ser usado o *token* de nome do utilizador com a *password* em “claro”. O canal de transporte entre os *end-points* deve ser https e é obrigatória a existência dos elementos listados na secção <RequiredElements>:

```
<wsp:Policy xmlns:wsp=".../policy" xmlns:sp=".../securitypolicy">
  <sp:TransportBinding xmlns:sp=".../securitypolicy">
    <wsp:Policy>
      <sp:TransportToken>
        <wsp:Policy>
          <sp:HttpsToken/>
        </wsp:Policy>
      </sp:TransportToken>
    </wsp:Policy>
  </sp:TransportBinding>
  <sp:SupportingTokens>
    <wsp:Policy>
      <sp:UsernameToken/>
    </wsp:Policy>
  </sp:SupportingTokens>
  <sp:RequiredElements>
    <sp:Xpath xmlns:soapenv="..." xmlns:ns1="...">
      /soapenv:Envelope/soapenv:Header/ns1:h1
    </sp:Xpath>
  </sp:RequiredElements>
</wsp:Policy>
```

WS-MetadataExchange

WS-MetadataExchange define um protocolo simples para “questionar” *metadata*, nomeadamente, XSD, WSDL e WS-Policy.

O elemento <Dialect> do pedido <GetMetadata> é usado para indicar qual o tipo de *metadata* que está a ser pedido. No exemplo seguinte, em que é pedido um documento WS-Policy, o elemento <Dialect> toma o valor “tipo URI”.

```
<Envelope xmlns=".../soap/envelope/">
  <Body>
    <wsx:GetMetadata xmlns:wsx=".../mex">
      <wsx:Dialect>
        http://schema.xmlsoap.org/ws/2004/09/policy
      </wsx:Dialect>
    </wsx:GetMetadata>
  </Body>
</Envelope>
```

A resposta do servidor ao pedido anterior toma a seguinte forma:

```
<Envelope xmlns=".../soap/envelope/">
  <Body>
    <wsx:Metadata xmlns:wsx=".../mex">
      <wsx:MetadataSection Dialect=".../policy">
        <wsp:Policy xmlns:wsp="...">
          (... WS-Policy retornada pelo serviço de metadata...)
        </wsp:Policy>
      </wsx:MetadataSection>
    </wsx:Metadata>
  </Body>
</Envelope>
```

WS-MetadataExchange é muito fácil de usar. Por exemplo, através deste protocolo, duas entidades podem trocar políticas sem ser necessário recorrer a repositórios de políticas partilhados.

WS-PolicyAttachment

WS-PolicyAttachment define a forma como as políticas podem ser anexadas às definições WSDL ou entradas UDDI. Este protocolo define quatro níveis para a definição de políticas de um WS: serviço, *end-point*, operação e mensagem. Por exemplo:

- Ao nível do serviço, a política pode definir que todas as trocas das mensagens devem ser registadas devido à existência de normas de auditoria.
- Ao nível do *end-point*, a política pode indicar que um determinado conjunto de algoritmos e chaves são suportadas nas tarefas de encriptação e assinatura dos dados.
- Ao nível da operação, as políticas podem indicar que é necessária uma declaração SAML para indicar os privilégios do cliente.
- Ao nível da mensagem, a política pode identificar quais as partes da mensagem que precisam de ser protegidas através da encriptação e assinatura dos dados.

A política a usar na troca das mensagens é uma combinação da própria política e destes quatro níveis de definição. A nível da sintaxe, no WSDL, a política é dividida nas quatro partes seguintes:

- O nível do serviço é associado ao elemento “wsdl:Service” do WSDL.
- O nível do *end-point* é associado aos elementos “wsdl:Port”, “wsdl:Binding” e “wsdl:PortType” do WSDL.
- O nível da operação é associado aos elementos “wsdl:Binding”/“wsdl:Operation” e “wsdl:PortType”/“wsdl:Operation” do WSDL.
- O nível da mensagem é associado aos elementos:
 - “wsdl:Binding”/“wsdl:Operation”/“wsdl:input”, “wsdl:output”, “wsdl:fault”
 - “wsdl:PortType”/“wsdl:Operation”/“wsdl:input”, “wsdl:output”, “wsdl:fault”
 - “wsdl:Message”

WS-Privacy

As organizações que criam, gerem e usam WS, frequentemente necessitam de especificar as suas políticas de privacidade e exigir que nos pedidos dirigidos aos seus WS estejam incluídas declarações relativas à adesão desses clientes a essas políticas. Através do uso de WS-Policy, WS-Security e WS-Trust é possível especificar e indicar se uma organização está ou não em conformidade com a política de privacidade. A especificação WS-Privacy descreve:

- O modelo usado para incluir a “linguagem de privacidade” nas descrições WS-Policy.
- Como é que WS-Security pode ser usada para associar as declarações de privacidade na mensagem.
- Como os mecanismos WS-Trust podem ser usados para avaliar as declarações de privacidade para as preferências dos utilizadores e declarações das práticas seguidas pela organização.

WS-Authorization

Descreve como gerir e especificar os dados relativos à autorização e as próprias políticas de autorização. WS-Authorization permite descrever como é que essas declarações podem ser especificadas nos *tokens* de segurança e como é que devem ser interpretadas pelos *end-points*. Este *standard* foi criado para ser flexível e extensível relativamente ao formato da linguagem de autorização de forma a poder ser usado no maior número de cenários possíveis.

WS-I Basic Profile (BSP)

As maiores empresas participantes na especificação de *standards* relacionados com os WS juntaram-se, formando a *Web Services Interoperability (WS-I) Organization*, com o objectivo de darem seguimento à definição dos *standards* WS e a que estes sejam implementados de formas que não criem problemas de interoperabilidade entre plataformas, sistemas operativos e linguagens de programação (a combinação errada de dois *standards* pode criar graves falhas de segurança sem que as pessoas que os estão a implementar se apercebam). Em Abril de 2004, WS-I publicou o *Basic Profile* 1.0. Este conjunto de especificações representou a primeira geração dos *standards* dos WS interoperáveis. Neste conjunto de *standards* estavam incluídos: SOAP 1.1, WSDL 1.1, UDDI 2.0, XML 1.0 (2ª Edição), XML Schema Part 1: Structures, XML Schema Part 2: Datatypes, RFC 2246: *The Transport Layer Security Protocol Version 1.0*, RFC 2459: *Internet X.509 Public Key Infrastructure Certificate and CRL Profile*, RFC 2616: *Hyper Text Transfer Protocol 1.1*, RFC 2818: *HTTP over TLS*, RFC 2965: *HTTP State Management Mechanism* e *The Secure Sockets Layer Protocol Version 3.0*. Em Abril de 2006, o comité WS-I actualizou o *Basic Profile* para a versão 1.1 com algumas alterações e correcções efectuadas à errata anteriormente publicada (actualmente no WS-I estão a trabalhar nas versões 1.2 e 2.0 do *Basic Profile*).

Essencialmente, estes perfis WS-I disponibilizam casos de uso para os WS, assim como as melhores formas de os implementar, garantindo a interoperabilidade. Muitas empresas implementam os seus WS tendo como base estes perfis e, alguns fornecedores de produtos

de segurança relacionados com os WS, garantem que os seus produtos e tráfego gerado/tratado pelos mesmos são compatíveis com os perfis WS-I. A necessidade da definição destes perfis deve-se ao facto de, com base nos *standards* associados aos WS, existirem várias soluções para resolver o mesmo problema. Estes perfis permitem restringir essas soluções a um pequeno conjunto (perfil) e assim, os responsáveis pela implementação dos WS têm uma maior garantia de suporte da solução que foi validada previamente como estando de acordo com os perfis WS-I.

Algumas das regras constantes no WS-I *Basic Profile* são as seguintes:

- Eliminação de escolhas desnecessárias. Ex.: o WS-I *Basic Profile* proíbe o uso de SSL 2.0 pois apresenta alguns problemas que foram corrigidos no SSL 3.0.
- Estabelecimento de boas práticas. Ex.: o WS-I *Basic Profile* proíbe o uso de mais do que um cabeçalho WS-Security com o atributo “actor” omitido.
- Não aconselha o uso de determinadas hipóteses de implementação. Ex.: Em relação às cifras usadas nos processos de encriptação, não aconselha o uso de Diffie-Hellman anónimos (são vulneráveis a ataques *man-in-the-middle*), MD5 (algoritmo com algumas falhas conhecidas), chaves de 40 ou 56 bit (facilmente são comprometidas através de ataques “*brute force*”), etc..
- É obrigatório definir os tipos de dados. O WS-Security não exige a presença de tipos de dados quando diferentes tipos de dados podem ser usados num determinado contexto. O WS-I *Basic Profile* garante a interoperabilidade desses casos através da exigência da definição dos tipos de dados. Ou seja, na referência a um *token* de segurança deve ser especificado o tipo do valor.

ANEXO B: RFI (*Request for Information*) elaborado pelo Banco BPI**1. Requirements**

As Banco BPI is interested in choosing a XML Security Gateway (XSG) to secure its Web Services, below are listed the preferential, desirable and accessory requirements that will be relevant to choose this type of appliances.

1.1. Preferential Requirements

- Hardware optimized to:
 - o Accelerate Secure Sockets Layer (SSL) operations
 - o Accelerate Cryptographic operations
 - o Accelerate XML validation operations(XML parsing, schema validation)
 - o Accelerate XML transformation operations
 - o Message or attachment compression
 - o Cache XML traffic
- SSL termination (XSG has to be able to terminate the tunnel SSL from the client, process the message and initiates other tunnel to the WS server).
- XML Intrusion/Threat Prevention/Detection (e.g.: XML Denial of Service, SOAP attachment viruses, buffer-overflow attempts, application-level attacks including SQL injection e XPath Injection, service scanning, and brute-force "flooding" denial-of-service attacks (nested elements, false schema invocations, attempts at recursion, ...)).
 - o Please list some of the attacks detected.
 - o IDS/IPS Signatures Automatic Update
- XML validation
 - o XML well-formedness checks
 - o Schema validation (internal and external schema repository)
 - o XML parsing
- Signature validation
- Source IP filtering
- Web services virtualization
 - o Stylesheet Language Transformation (XSLT)
 - o Message routing based on:
 - SLA
 - XML message contents (please give some examples of these type of contents)

- Web services' servers load, number of requests, round robin, etc. (methods used by the XSG to implement high availability/load balancing of the WS servers)
- Source IP
- Standards' support:
 - o WS-Security (message level signature and encryption; XSG has to be able to encrypt and sign some fields of the XML message using an internal PKI)
 - o WS-SecureConversation
 - o WS-Trust
 - o WS-Federation
 - o XACML
 - o WS-Policy
 - o WS-PolicyAttachment
 - o WS-MetadataExchange
 - o WS-AtomicTransaction
 - o WS-Coordination
 - o WS-ReliableMessaging
 - o WS-Addressing
 - o WS-SecureExchange (WS-Trust, WS-SecureConversation, WS-SecurityPolicy)
 - o BPEL
 - o WS-BPEL
 - o WS-CDL
- SLA definitions
 - o Maximum message size
 - o Define of traffic limits
 - o Define service-level contracts with multiple performance indicators (e.g.: time of execution, etc.)
- Logging/Reporting
 - o Central log repository
 - o High availability of central log repository
 - o Fields logged – Please give some examples of the fields, related to XML traffic processing, that are logged.
 - o Logging with explore capability: log all traffic and give tools to analyze it.
- Monitoring
 - o Metrics for XSG operation: Network and Hardware (ex.: Performance, Latency, memory, CPU).

- o Web Services' traffic monitoring
 - o Online diagnostic: Possibility to use filters for traffic analysis
- Alerts
 - o Send alerts using SMTP, SNMP or Syslog.
 - o XSG operational alerts: hardware failure, x% memory utilization, number of simultaneous connections, attacks, etc.
 - o Alerts related to WS traffic: send an alert when some SLA value isn't granted, etc.
- XSG's administration
 - o Administration/Operation auditing: ability to know who made what changes to the XSG
 - o High availability of the XML Gateway Manager: cluster installation or possibility to have other manager with a replica of the configuration or other proprietary option.
- Integration with some external products:
 - o SOA Governance – Please list products
 - o UDDI registry/repository – Please list products
 - o Identity Management Systems and Access Control Systems: AD – Active Directory or other LDAP systems (kerberos), ADFS – Active Directory Federation Services, STS – Security Token Service (SAML), SSO – Single Sign On Systems (SSO Session Cookies), CA – Certificate Authority (X.509), Banco BPI's Proprietary Systems (username), ... (Please list other products)
 - o Reverse http/https proxies – Please list products
- XSG High availability
 - o XSG'S type of operation: active/active or active/passive?
 - o Support to stateful and stateless failover and load balancing
 - o Load Balancing: XSG cluster manages its traffic distribution or it's necessary to use an external load balancer to distribute the traffic between the XSG cluster's members?

1.2. Desirable Requirements

- Logging/Reporting
 - o Automatic Log Management Tools available (rotation of logs, backups, ...)
 - o Statistics Reports. Give some examples.
- XSG's administration
 - o Administration Delegation: we want to allow operation team to manage filtering policies without full administration permissions.

- o Definition of roles of administration consulting external Identity Management Systems: Microsoft Active Directory, Cisco ACS, etc. (list some other products).
- o User friendly interaction
- o Management of configuration's versions and backups
- o Central Management: Centrally, manage two sites with various WS (these sites are at different cities).

1.3. Accessory Requirements

- Software version of the XML Security Gateway to be used at development, quality or pre-production environments: Windows installation and support of installation in virtual machines.
- Possibility to virtualize the XML Security Gateway (at the same XSG I can create some contexts with different policies/configurations applied).
- Possibility to install XML Gateway Manager in a virtual machine.
- Possibility to install Central Logging System in a virtual machine.
- WSDL publication
- VPN termination
- Antivirus for SOAP attachments
- Logging/Reporting
 - o Integration with some external products: Cisco MARS, Microsoft Operations Manager, other examples.
 - o Customizable Reports
 - o Send reports automatically by email
- Automatically firmware update
- Remote Load Balancing of the WS servers – we have two datacenters and some WS are implemented at both datacenters
- Simulate configuration applications: we want to simulate a new configuration and see if some test messages are blocked/transformed/etc.
- XSG High availability
 - o High availability: Local and Remote – As BPI has two main Sites, which is the better architecture to this case? Can I have a XSG cluster with members at two Sites (different cities)?

2. Partners

Contact of your partners in Portugal (or for Portugal).

ANEXO C: Lista de requisitos constantes no RFP (*Request for Proposal*) elaborado pelo Banco BPI

1. Requisitos Preferenciais

- P1 – *Hardware* otimizado para aceleração de operações SSL.
- P2 – *Hardware* otimizado para aceleração de operações criptográficas.
- P3 – *Hardware* otimizado para operações de validação XML (*parsing* do XML, validação de esquemas).
- P4 – *Hardware* otimizado para operações de transformação XML.
- P5 – Terminação SSL (possibilidade de também poder iniciar ligação SSL para os WS). – A *XML Security Gateway* deverá poder terminar túneis SSL estabelecidos com o cliente, processar a mensagem e iniciar um outro túnel SSL com o servidor WS.
- P6 – XML IDS/IPS. – Listar alguns dos ataques detectados.
- P7 – Validação da formatação do XML.
- P8 – Validação dos esquemas XML contra repositórios tanto internos como externos.
- P9 – Validação de assinaturas.
- P10 – Filtragem por IP de origem.
- P11 – Suporte de transformações XSLT.
- P12 – Encaminhamento das mensagens com base em SLAs.
- P13 – Encaminhamento das mensagens com base no conteúdo das mensagens. – Listar alguns exemplos de tipos de “conteúdos” das mensagens que podem ser usados nas regras de encaminhamento.
- P14 – Encaminhamento das mensagens com base na carga dos WS, nº pedidos, *round-robin*, etc.. – Listar alguns exemplos de tipos de “conteúdos” das mensagens que podem ser usados nas regras de encaminhamento.
- P15 – Encaminhamento das mensagens com base no IP de origem.
- P16 – Suporte do standard WS-Security. – A XSG deverá ser capaz de encriptar e assinar alguns dos campos das mensagens XML usando uma PKI interna. Também deverá suportar o uso de *tokens username, kerberos, X.509* e SAML).
- P17 – Suporte do *standard* WS-SecureConversation.
- P18 – Suporte do *standard* WS-Trust.
- P19 – Suporte do *standard* WS-Federation.
- P20 – Suporte do *standard* XACML.
- P21 – Suporte do *standard* WS-Policy.
- P22 – Suporte do *standard* WS-PolicyAttachment.
- P23 – Suporte do *standard* WS-ReliableMessaging.
- P24 – Suporte do *standard* WS-Addressing.

- P25 – Suporte do *standard* WS-SecureExchange (WS-Trust, WS-SecureConversation e WS-SecurityPolicy).
- P26 – Definição de SLAs em função do tamanho máximo das mensagens.
- P27 – Definição de SLAs em função da definição de limites de tráfego.
- P28 – Definição de SLAs em função de múltiplos indicadores de *performance*: tempo de execução, etc..
- P29 – *Logs/Relatórios* – Repositório central de *log*. – Consolidação dos *logs* das várias XSG, localizadas em *datacenters* distintos, num repositório central.
- P30 – *Logs/Relatórios* – Campos registados. – Listar alguns dos campos registados.
- P31 – *Logs/Relatórios* – Disponibilização de ferramentas de análise dos *logs*. – Existência de ferramentas para exploração de *logs* que permitam um diagnóstico em tempo real e em diferido, com capacidade de definição de filtros.
- P32 – Monitorização – Disponibilização de métricas relacionadas com a operação das XSG (*performance*, latência, memória, etc.).
- P33 – Monitorização do tráfego dos WS.
- P34 – Monitorização *online* para diagnóstico com a possibilidade de usar filtros nessa análise.
- P35 – Alertas – Possibilidade de enviar alertas por SMTP, SNMP ou Syslog.
- P36 – Alertas relativos à operacionalidade da XSG em situações de falhas de *hardware*, utilização da memória, nº de ligações simultâneas, ataques, etc.
- P37 – Alertas relativos ao tráfego dos WS: envio de alertas se determinado valor de SLA não é cumprido, etc.
- P38 – Administração da XSG – Auditoria de actividades de administração/operação. – Capacidade de registo e exploração das actividades de administração relativas a configurações e visualização de *logs*.
- P39 – Possibilidade de integrar com produtos externos de SOA *Governance*. – Listar alguns desses produtos.
- P40 – Possibilidade de integrar com produtos externos de UDDI *registry/repository* (suporte UDDI v.2 ou v.3). – Listar alguns desses produtos.
- P41 – Possibilidade de integrar com sistemas de gestão de identidade e controlo de acessos. – Listar alguns desses produtos.
- P42 – Suporte de LDAP.
- P43 – Suporte de ADFS.
- P44 – Suporte de *kerberos*.
- P45 – Suporte de SAML.
- P46 – Suporte de X.509.
- P47 – Suporte do sistema proprietário do Banco BPI.
- P48 – Integração com *Microsoft Active Directory*.

P49 – Disponibilidade da XSG – Suporte de activo/activo e/ou activo/passivo?

P50 – Disponibilidade da XSG – Suporte de failover *stateful* e/ou *stateless*? – Garantia de serviço sem intervenção humana, nas situações em que ocorre falha de um dos equipamentos.

P51 – Disponibilidade da XSG – Suporte de balanceamento *stateful* e/ou *stateless*? – Garantia de serviço nas situações em que, uma das XSG está com bastante carga e os pedidos que estão a ser processados por essa XSG são passados para outra XSG pertencente ao mesmo *cluster*.

2. Requisitos Desejáveis

D1 – *Logs/Relatórios* – Possibilidade de enviar *logs* directamente para bases de dados SQL, Oracle, etc..

D2 – *Logs/Relatórios* – Possibilidade de implementar uma solução com redundância dos *log servers*.

D3 – *Logs/Relatórios* – Possibilidade de gestão automática dos *logs* (rotação de *logs*, *backups*, etc.).

D4 – *Logs/Relatórios* – Possibilidade de gerar relatórios, já predefinidos, com as estatísticas de utilização.

D5 – Administração da XSG – Possibilidade de instalar a estação de gestão das XSG de forma redundante. – Instalação da estação de gestão das XSG em *cluster*, suportar uma estação de gestão de *backup* ou outra solução proprietária.

D6 – Administração da XSG – Possibilidade de delegar determinadas tarefas de configuração/consulta das AXG sem ser necessário atribuir permissões de administração a esses grupos de utilizadores.

D7 – Administração da XSG – Definição de papéis de administração diferentes, consultando sistemas de IAM externos (Microsoft AD, Cisco ACS, etc.). – Definição de diferentes de privilégios de acordo com as responsabilidades de cada equipa, ex.: administração, operação, monitorização e auditoria.

D8 – Administração da XSG – *Interface* gráfica intuitiva e fácil de usar.

D9 – Administração da XSG – Gestão automática das versões das configurações das XSG e respectivos *backups*.

D10 – Administração da XSG – Possibilidade de gerir, a partir de um único local central de gestão, XSGs localizadas em *datacenters* localizados em cidades diferentes. – Administração de todos os equipamentos a partir de um único local, com replicação de configurações.

D11 – Suporte de *Single Sign-On*.

3. Requisitos Acessórios

A1 – *Hardware* optimizado para permitir compressão de mensagens e anexos XML.

A2 – *Hardware* optimizado para permitir *cache* do tráfego XML.

A3 – Actualizações automáticas das assinaturas XML IDS/IPS.

A4 – Suporte do *standard* WS-MetadataExchange.

A5 – Suporte do *standard* WS-AtomicTransaction.

A6 – Suporte do *standard* WS-AtomicTransaction.

A7 – Suporte do *standard* BPEL.

A8 – Suporte do *standard* WS-BPEL.

A9 – Suporte do *standard* WS-CDL.

A10 – Disponibilidade da XSG – As XSG são capazes de balancearem o seu próprio tráfego (distribuição de carga das XSG) ou o balanceamento tem de ser efectuado recorrendo a equipamentos externos?

A11 – Existência de uma versão por *software* da XSG (para ser usado em ambientes de desenvolvimento e qualidade).

A12 – Possibilidade de criar vários contextos de XSG no mesmo *hardware*.

A13 – Administração da XSG – Possibilidade de instalar a estação de gestão numa máquina virtual.

A14 – *Logs/Relatórios* – Possibilidade de instalar o sistema de gestão dos *logs* numa máquina virtual.

A15 – *Logs/Relatórios* – Integração com sistemas de *syslog* ou *snmp* externos.

A16 – *Logs/Relatórios* – Possibilidade de criar relatórios personalizados.

A17 – *Logs/Relatórios* – Possibilidade de enviar relatórios por *mail*.

A18 – *Logs/Relatórios* – Possibilidade de exportar relatórios nos formatos pdf, csv ou MSWord.

A19 – Publicação de WSDL na própria XSG.

A20 – Terminação de VPNs na XSG.

A21 – Antivírus para mensagens XML e anexos. – A própria XSG já tem incorporado um sistema de antivírus? É possível integrar com sistemas de antivírus externos?

A22 – Actualizações automáticas do *firmware*. – É possível criar sistemas de actualização automática e remota do *firmware* das XSG?

A23 – Balanceamento entre *Sites* – Possibilidade de balancear WS existentes em mais do que um *datacenter* a partir da mesma XSG (numa das situações a XSG estará remota, num dos outros *datacenters*).

A24 – Arquitectura de alta disponibilidade proposta numa situação em que existem mais do que um *datacenter*. É possível ter um *cluster* de XSG com membros em *datacenters* (cidades) diferentes?

ANEXO D: Ferramentas disponíveis para testar a forma como as XSG detectam algumas situações de ataques

Ataques de intrusão destinados aos WS

- WSBang (<http://www.isecpartners.com/wsbang.html>)
- WSMaP (<http://www.isecpartners.com/wsmap.html>)
- WSDigger (<http://www.foundstone.com/us/resources/proddesc/wsdigger.htm>)
- Vordel SOAPBox (<http://www.vordel.com/products/soapbox/index.html>)
- OWASP Wfuzzer
(http://www.owasp.org/index.php/Category:OWASP_Wfuzzer_Project)
- OWASP SQLiX (só ataques “SQL Injection”)
(http://www.owasp.org/index.php/Category:OWASP_SQLiX_Project)
- SIFT Web Method Search Tool (<http://www.sift.com.au/73/171/sift-web-method-search-tool.htm>)

Proxy HTTP/HTTPS (incluem algumas ferramentas que podem ser utilizadas em ataques dirigidos aos WS)

- OWASP WebScarab
(http://www.owasp.org/index.php/Category:OWASP_WebScarab_Project)
- Achilles (<http://www.mavensecurity.com/achilles>)

Produtos de diagnóstico dos WS (verificam se os WS estão de acordo com determinadas especificações e testam algumas vulnerabilidades)

- SOAPSonar (<http://www.crosschecknet.com>)