Encyclopedia of Information Systems and Technology

Phillip A. Laplante, Vladimir Stantchev, Jeff Voas, Anne-Marie Smith, Qiuming Zhu, Morris Chang, Irena Bojanova, David Alan Grier, Jia Zhang, Thomas Costello, San Murugesan, Jim Goldman, Joanna DeFranco

SOA: Service-Oriented Architecture

Publication details
Glenn Cater
Published online on: 29 Dec 2015

How to cite :- Glenn Cater. 29 Dec 2015, SOA: Service-Oriented Architecture from: Encyclopedia of Information Systems and Technology CRC Press
Accessed on: 20 Aug 2023

PLEASE SCROLL DOWN FOR DOCUMENT
SOA: Service-Oriented Architecture

Glenn Cater
Director, IT Risk Consulting, Aon Consulting, Inc., Freehold, New Jersey, U.S.A.

Abstract
The concept of service-oriented architecture (SOA) has been around in various forms for some time, but the SOA model has really become popular of late because of advances in web technology, web services, and standards. Although the concept of an SOA is not tied to a specific technology, in most cases SOA now refers to a distributed system using web services for communication. Other examples of SOA architectures are primarily based upon remote procedure calls, which use binary or proprietary standards that cause challenges with interoperability. Web services solve the problems of interoperability because they are based upon eXtensible Markup Language (XML), by nature an interoperable standard. Significant effort is being put into developing security standards for web services to provide integrity, confidentiality, authentication, trust, federated identities, and more. Those security standards will be the focus of this entry, which will cover XML, XML encryption, XML signature, Simple Object Access Protocol (SOAP), Security Assertion Markup Language (SAML), WS-Security, and other standards within the WS-Security family.

INTRODUCTION
So what is a service-oriented architecture (SOA)? SOA is an architectural model based on independent (or loosely coupled) services, with well-defined interfaces designed in such a way as to promote reuse. SOA fits extremely well with an architecture based on web services, which by nature meet the definition of loose coupling and well-defined interfaces. For instance, as an example of a service in SOA, imagine a user directory that is accessible via web services. In this example, the interface may specify functions, or methods, that include searching the directory (searchDirectory), password resets (resetPassword), updating user information (updateUser), and adding and removing users (addUser, removeUser). As long as the interface is adequately defined, the consumer of the service does not need to know how the service is implemented to use it. Fig. 1 illustrates a simplified SOA.

Fig. 1 shows that each service is reasonably independent and has a well-defined purpose. The idea behind SOA is that, provided the services and their interfaces are designed well, they can be combined in different ways to build different types of applications. For example, the order-processing service may be accessible from both the public website for placing orders and the internal website for sales and marketing purposes. Services expose their functionality through industry standard web service interfaces described using the Web Services Description Language (WSDL), which is discussed later in this entry.

In the simple example mentioned earlier, there is no security shown on Fig. 1. To add security to this picture some of the following need to be addressed:

- Network security, operating system security (server hardening), application security, and physical security
- Transport security, typically via the use of secure sockets layer (SSL)
- Web service security, through the use of the Web Service-Security family (WS-*) of standards for securing web services messages
- Utilizing other WS-Security extensions to provide trust relationships between the company, the payment provider, and the shipping provider

Web services and Web Services Security standards make heavy use of eXtensible Markup Language (XML). XML has revolutionized data exchange because of its simplicity and power. As a simple, human-readable, text format XML has facilitated data exchange between applications and businesses even across dissimilar systems.

The remainder of this entry discusses web services and the methods used to secure applications and data in an SOA environment. XML, XML encryption, XML signature, simple object access protocol (SOAP), security assertion markup language (SAML), and Web Services Security standards will also be covered as part of this entry.
FOUNDATIONS FOR WEB SERVICES AND WEB SERVICES-SECURITY

Web services and Web Services Security are based upon a number of standards that should be understood to some extent by a security practitioner. The idea is to provide an overview of the relevant standards here and how they fit together. Then for some of the more complex standards, we will delve into more detail in later sections.

eXtensible Markup Language

XML is the basic building block on which all the other web services standards and Web Services Security standards are built. XML is a free, open standard recommended by the World Wide Web Consortium (W3C) as a method of exchanging data using a simple text-based format. The fact that XML is a simple, human-readable format and works across heterogeneous systems makes it perfect for web services and SOAs for which the service and the consumer (client) may be on different platforms.

The example in Fig. 2 is a snippet of XML describing a person. This simple example shows how XML can be easily read by a human being. The structure of the XML clearly identifies this as data related to a person (see the person element in Fig. 2). So in addition to exchange of data, the XML gives some understanding of what the data represents.

XML extensions

Although not really important for the understanding of how XML relates to Web Services Security, there are some extensions to XML that should be included for completeness.

XML schema is an important extension that allows the structure of XML to be defined similar to the way in which an SQL database schema is defined. Among other things, XML schema specifies what the structure of the XML should be, such as the order in which elements appear, how many of each element is allowed, and the data types. XML schema is useful for creating specifications and for automatically validating the correctness of XML.

XML also has the concept of “XML namespaces.” XML namespaces provide a way to avoid naming conflicts. For example, imagine that there are two different definitions of an employee in XML; to differentiate them XML namespaces can be used. The way this is
done is by prefixing the name with a namespace prefix, for example `<abc:Employee>`, where abc is the namespace prefix that contains a definition of the employee type.

Other extensions exist that provide powerful ways to extract and query data in an XML message. These extensions are called XPath and XQuery. XPath provides a way to reference parts of the XML structure, whereas XQuery is a powerful query language that allows queries to be written against the XML data, similar to SQL, which is the query language for relational databases.

**Simple Object Access Protocol**

SOAP is an important messaging protocol that forms the basis for the web services protocol stack. SOAP messages are designed to be independent of a transport protocol, but are most often transmitted via HTTP or HTTPS when used with web services. SOAP messages are not tied to the HTTP protocol, however, and may also be used in message queuing systems, sent through e-mail, or via other transport mechanisms.

The SOAP standard is based upon XML and defines the structure of messages that can be passed between systems. Messages defined in SOAP have an envelope, a header, and a body as shown in Fig. 3. The SOAP header allows for the inclusion of security elements such as digital signatures and encryption within the message. Although security elements are not restricted only to the header, it is used heavily with WS-S standards to transmit security information with the message.

There are two primary messaging modes used by SOAP—“document” mode and remote procedure call (RPC) mode. Document mode is good for one-way transmission of messages, in which the sender submits the SOAP message but does not expect a response. RPC mode is more commonly used and is a request–response model in which the sender submits the SOAP request and then waits for a SOAP response.

**WSDL and UDDI**

The WSDL and Universal Description, Discovery, and Integration (UDDI) standards allow a consumer of a web service to understand how to find a web service and how to use that service. This includes the following:

- Discovery of basic information about the service such as the service name
- Where to find the service, including network endpoints and protocol used
- How to call the service (the service contract)

WSDL is essentially metadata in the form of XML that describes how to call a web service. There is a security concern with the protection of the WSDL data, because if it falls into the wrong hands it can expose information about your network. The WSDL metadata may be stored as an XML file, but is often available via a URL on the same application server where the web service is hosted. The WSDL should be made available only to authorized users of the service. Later in this entry, we will discuss how security policy requirements are included in WSDL.

UDDI is different in that it defines a standard for a directory of web services. This allows other applications or organizations to discover the WSDL for a web service that meets their need. Businesses publish the WSDL for their web service in the directory so that it can be easily discovered by others. UDDI directories can be hosted publicly on the Internet or internally within corporations to allow services to be discovered dynamically. Security of UDDI directories must be maintained to prevent man-in-the-middle attacks, by which a fake web service could be published in place of a real one. UDDI builds upon other Web Services Security standards to ensure integrity and trust for the data within the directory, which is particularly important for publicly accessible directories.

**XML Signature**

XML signature provides for integrity and authentication of XML data through the use of digital signatures and can be applied not only to XML but to any digital content. The primary use within Web Services Security is to sign XML messages to ensure integrity and to prove the identity of the signer. Fig. 4 shows an informal
representation of the XML signature syntax. The details are removed to reveal the basic structure. Unfortunately, a more complete explanation of how digital signatures work is beyond the scope of this discussion.

XML signature is itself represented as XML, as Fig. 4 shows. The structure contains the following elements:

- **Signature** is the containing element that identifies that this is a digital signature.
- **SignedInfo** contains the references to, and digests of, the data that is digitally signed.
- **CanonicalizationMethod** refers to the way the SignedInfo element is prepared before the signature is calculated. The reason for this is because different platforms may interpret data slightly differently (e.g., carriage returns `<CR>` versus carriage return/line feeds `<CRLF>`), which would cause signatures to compute differently on different platforms.
- **SignatureMethod** refers to the algorithm used for signature generation or validation, for example, `dss-sha1`, which refers to the use of the DSA algorithm with the SHA-1 hashing function.
- **The reference element is complex, but in a nutshell it refers to the data being signed, which is either part of the same XML data, or a uniform resource identifier (URI) that refers to external data, such as a document, web page, or other digital content. In addition, the reference element defines transforms that will affect the content prior to being passed to the digest for computing the hash (via `DigestMethod`). The resultant hash value is stored as `DigestValue`.
- **SignatureValue** is the actual computed signature value. Rather than digitally signing the actual content, the signature is computed over SignedInfo so that all the references, algorithms, and digest values are digitally signed together, which ensures the integrity of the data being signed.
- **KeyInfo** enables the recipient to obtain the key needed to validate the signature, if necessary. This structure is fairly complex and is described in more detail under XML encryption.
- **The object element contains arbitrary XML data that can be referenced within SignedInfo. It can also include a manifest element, which provides an alternate list of references, where the integrity of the list itself is validated, but the integrity of the actual items will not invalidate the signature. The purpose of such a list might be to include an inventory of items that should accompany the manifest. It also defines a `SignatureProperties` element by which other properties of the signature are stored such as the date and time the signature was created.

The XML signature standard defines three types of digital signatures, which are enveloped, enveloping, and detached. An enveloped signature refers to a signature on XML data, whereby the signature element is contained within the body of the XML. Enveloping signatures contain the XML content that is being signed, and this is where the object element is used to contain the data that is signed. Finally, the detached signature type signs content that is external to the XML signature, defined by an URI, which may be external digital content, but can also include elements within the same XML data such as sibling elements. Fig. 5 provides an example of a detached signature.

As discussed earlier, XML signature allows any type of digital content to be signed, and there are uses for XML signature that go beyond the scope of Web Services Security. However, this overview of XML signature is intended to provide a foundation for understanding how it relates to Web Services Security.

**XML Encryption**

By design, XML is a plain text format with no security built in. XML encryption provides data confidentiality through a mechanism for encrypting XML content that relies on the use of shared encryption keys. Fig. 6 shows an informal representation of the XML encryption syntax. The details are removed to simplify explanation of the structure symmetric encryption keys. Standard key exchange techniques based on public-key cryptography provide secrecy for the shared key. Typically the shared
key is included within the XML message in an encrypted form, is referenced by name or URI, or is derived from some key exchange data. Symmetric encryption keys are used to encrypt data for performance reasons because public-key encryption can be very slow in comparison.

Fig. 5 XML signature example.

Fig. 6 shows an informal representation of the XML encryption syntax. The details are removed to simplify the explanation of the structure. Like XML signature, XML encryption is itself represented as XML, as Fig. 6 shows. The structure contains the following elements:

- EncryptedData is the containing element that identifies that this is encrypted data.
- EncryptionMethod defines the encryption algorithm that is used to encrypt the data, such as Triple-DES (3DES). This is an optional element and if it is not present, then the recipient must know what algorithm to use to decrypt the data.
- ds:KeyInfo contains information about the encryption key that was used to encrypt the message. Either the actual key is embedded in encrypted form or there is some information that allows the key to be located or derived.
- ds:KeyId contains an encrypted form of the shared key. As mentioned previously, this key will typically be encrypted using public-key cryptography. There may be multiple recipients of a message, each with their own encrypted key element.
- AgreementMethod is an alternate way of deriving a shared key by using a method such as Diffie–

Fig. 6 Informal XML encryption syntax.
Hellman. Providing key agreement methods means that the key does not need to be previously shared or embedded within the EncryptedKey element.

- **ds:KeyName** provides another way of identifying the shared encryption key by name.
- **ds:RetrievalMethod** provides a way to retrieve the encryption key from a URI reference, either contained within the XML or external to it.
- **ds:*** refers to the fact that there is other key information, such as X.509v3 keys, PGP keys, and SPKI keys that can be included.
- **CipherData** is the element that contains the actual encrypted data, either with **CypherValue** as the encrypted data encoded as base64 text or by using **CypherReference** to refer to the location of the encrypted data, in the XML or otherwise.
- **EncryptionProperties** contains additional properties such as the date and time the data was encrypted.

Fig. 7 shows an example of an XML-encrypted message. The encrypted data is clearly visible in the CipherValue element. This basic overview of the XML encryption standard helps to give some background on how data confidentiality can be achieved with XML; however, there is much more detail than can be covered here.

The XML signature and XML encryption standards together form the basic security building blocks upon which the rest of the WSS standards rely.

**SECURITY ASSERTION MARKUP LANGUAGE**

Security assertion markup language (SAML) is a standard framework based on XML for communicating user identity, user entitlements, and user attributes between organizations or entities in separate security domains. SAML builds upon XML signature and XML encryption to provide integrity, confidentiality, and authentication of SAML assertions.

SAML allows an entity or organization to vouch for the identity of an individual, via a SAML assertion (a portable XML authentication token). The SAML assertion can be presented as proof of identity to another entity provided a trust relationship has been established between the two parties. This can be important for SOAs for which services are located within separate companies or security domains. This concept is really the basis of federated identity, which insulates organizations from the details of authentication and identity management within other organizations.

SAML attempts to solve several problems:

- **Web single sign-on**—by which a user can sign into one website and then later sign into a second related website using the credentials (a SAML assertion) provided by the first site.
- **Delegated identity**—by which credentials supplied to an initial website or service can be utilized by that service to perform actions on behalf of the user. An example is a travel website, which can pass the user identity to other services to perform airline, hotel, and car rental reservations.
- **Brokered single sign-on**—by which a third-party security service authenticates the user. The credentials provided by the third-party security service can then be used to authenticate to multiple websites.
- **Attribute-based authorization**—by which attributes about the user are placed into the SAML assertion. These attributes are then used to make authorization decisions. For example, user “John Doe” has level “director” in the “human resources” department; based upon these attributes he is allowed certain access to the human resources systems.

Within the SAML assertion will be some information about a user’s identity, such as the user’s e-mail address, X.509 subject name, Kerberos principal name, or an attribute such as employee identification number. For privacy purposes, SAML 2.0 introduced the concept of pseudonyms (or pseudorandom identifiers), which can be used in place of other types of identifiers, thereby hiding personal identification information such as an e-mail address. SAML provides two main ways to confirm the subject’s identity. One way is referred to as “holder of key,” where the sender of the message (the subject) typically holds the key that was used to digitally sign the message. The other confirmation method is referred to as “sender vouches,” which means that the
digital signature on the message was created by a trusted third party.

This description of SAML is intended to provide some understanding of where it fits within SOAs. By leveraging trust relationships between service providers, SAML provides loose coupling and independence with respect to user identity. SAML is also referenced by the WS-S standards as a type of security token.

**WEB SERVICES SECURITY STANDARDS**

To gain an understanding of how all the Web Services Security protocols fit together, refer to the illustration in Fig. 8. This diagram shows how XML signature, XML encryption, and SOAP form the foundation of the stack, with the other Web Services Security standards building upon them. Other standards, such as WSDL, UDDI, SAML, WS-Policy, and WS-PolicyAttachment are listed down the right-hand side of Fig. 8 that have relationships to the security standards, but are not specifically security standards themselves.

It is clear from Fig. 8 that the WS-Security protocol suite is complex, which can serve to discourage adoption of these standards into an SOA, particularly for application developers whose job is complicated by these security protocols. This complexity can lead to a reliance on SSL and firewall policies to provide point-to-point security for SOAP messages. Fortunately, tools are available to simplify the integration of security into web services and SOA.

**WS-Security**

The WS-Security standard, also referred to as WSS: SOAP Message Security, specifies extensions to SOAP...
that provide message integrity, message confidentiality, and message authentication. WS-Security leverages XML signature to ensure that the integrity of the message is maintained and XML encryption to provide confidentiality of the message. Security tokens are supported for authentication purposes to provide assurance that the message originated from the sender identified in the message.

There are three categories of security tokens that are defined by WS-Security: username tokens, binary security tokens, and XML tokens. Each of the security tokens supported by WS-Security fits within one of these categories. Examples of security tokens are user names and passwords (UsernameToken), Kerberos tickets (BinarySecurityToken), X.509v3 certificates (BinarySecurityToken), and SAML (XML Token). The WS-Security header is designed to be extensible to add additional security token types.

Fig. 9 shows where the WS-Security SOAP extensions appear within the header of the SOAP message.

The example in Fig. 9 shows that the structure of a SOAP message is altered when WS-Security extensions are added. It also shows how the security tokens, XML signature, and XML encryption fit within the WS-Security (wsse) header. The receiver of a message with WS-Security extensions processes the extensions in the order they appear in the header, so in this case the signature is verified on the message body and then the message is decrypted.

The following five types of tokens are discussed in version 1.1 of the standard:

- Username token, which is the most basic type of token. A UsernameToken contains a username to identify the sender and it can also contain a password as plain text, a hashed password, a derived password, or an S/KEY password. Obviously, the use of plaintext passwords is strongly discouraged.
- X.509 token, which is a BinarySecurityToken, identifies an X.509v3 certificate that is used to digitally sign or encrypt the SOAP message through the use of XML signature or XML encryption.
- Kerberos token, which is also a BinarySecurityToken, includes a Kerberos ticket used to provide authentication. Ticket granting tickets (TGT) and service tickets (ST) are supported.
- SAML token, which is an XML token, provides a SAML assertion as part of the SOAP security header.
- Rights expression language (REL) token, which is an XML token, provides an ISO/IEC 21000 or MPEG-21 license for digital content. This type of token is used for communicating the license to access, consume, exchange, or manipulate digital content.

WS-Security allows for the inclusion of time stamps within the SOAP security header. Time stamps can be required (see WS-Policy and WS-SecurityPolicy) to determine the time of creation or expiration of SOAP messages.

In addition, WS-Security defines how to add attachments to SOAP messages in a secure manner by providing confidentiality and integrity for attachments. Support for both multipurpose Internet mail extension (MIME) attachments and XML attachments is provided.

SOAP messages and attachments may be processed by different intermediaries along the route to the final recipient, and WS-Security allows parts of messages to be targeted to different recipients to provide true end-to-end security. There is an important distinction between point-to-point security technologies such as SSL and end-to-end security in which there are multiple intermediaries. A possible scenario is that one intermediary might need to perform some processing on a message before passing the message along; however, some parts of the message are confidential and intended only for the final recipient. SSL would not provide the necessary security in this scenario.

**WS-Policy and WS-SecurityPolicy**

The WS-Policy standard by itself is not directly related to security. Its purpose is to provide a framework for describing policy requirements in a machine-readable way. A policy might describe communication protocols, privacy requirements, security requirements, or any other type of requirement. WS-SecurityPolicy builds upon the WS-Policy framework to define security policies for WS-Security, WS-Trust, and WS-SecureConversation.

The following types of assertions are available within WS-SecurityPolicy:

- Protection assertions (integrity, confidentiality, and required elements), which define which portions of a message should be signed or encrypted and which header elements must be present.
- Token assertions, which specify the types of security tokens that must be included (or not included), such as UsernameToken, IssuedToken (third-party-issued token, e.g., SAML), X509Token, KerberosToken, Spnego-ContextToken (used with WS-Trust), SecurityContext-Token (external), SecureConversationToken (used with WS-SecureConversation), SamiToken, RelToken, Https-Token (requires use of HTTPS).
- Security-binding assertions, which define requirements for cryptographic algorithms, time stamps, and the order of signing and encrypting; whether the signature must be encrypted or protected and whether signatures must cover the entire SOAP header and body.
WS-Security assertions, which indicate which aspects of WS-Security must be supported within the message.

WS-Trust assertions, which define policy assertions related to WS-Trust.

There is a related standard, called WS-PolicyAttachment, that defines attachment points within WSDL at which security policies can be defined. This provides a mechanism for describing the security policy associated with a web service along with the web service interface definition.

WS-Trust

WS-Trust builds upon WS-Security and WS-Policy to define mechanisms for issuing, renewing, and validating security tokens. The WS-Trust model has many similarities to Kerberos, and there are direct analogies such as the delegation and forwarding of security tokens. Of course WS-Trust is designed to work over web services and with many types of security tokens, such as X.509, Kerberos, XML tokens, and password digests. WS-Trust can also extend to trust relationships over the Internet, whereas Kerberos is more suited to providing trust within intranet-type scenarios. WS-Federation, discussed later in this entry, builds upon these principles and adds mechanisms to provide a framework for implementing identity federation services.

In the WS-Trust model shown in Fig. 10, the web service has a policy that defines what security tokens are required to use the service (via WSDL). To access the web service, the requester needs a valid security token that the web service understands. To obtain a valid security token, the requester may directly request a token from the security token service (STS), via a RequestSecurityToken request. Assuming the requester adequately proves its claims (via digital signatures) to the STS and meets the STS policy, the STS will respond with a RequestSecurityTokenResponse containing a new token signed by the STS. This new token will be in a format the web service understands, even if the client and web service support different authentication mechanisms. For example, say the client understands X.509 certificates only and the web service understands SAML only, then the STS can issue a SAML token for the requester to present to the web service.

WS-Trust addresses the issue of trust in the security tokens by providing mechanisms for brokering trust relationships through the use of one or more STSs. Trust is established through relationships between the requester and an STS, between the web service and an STS, and between STSs. So the web service need not directly trust the requester or the STS it uses to accept security tokens, as long as there is a trust relationship between the requester’s STS and the web service’s STS.

WS-SecureConversation

The WS-SecureConversation standard builds upon WS-S and WS-Trust to define the concept of a security context, or session between services. Establishing a security context aims to alleviate some of the potential security problems with WS-S, such as message replay attacks and support for challenge–response security protocols.

There are three different ways to establish the security context.

- An STS (see WS-Trust) is used, whereby the initiator requests the STS to create a new security context token.
- The initiator is trusted to create the security context itself and sends it along with a message.
- A new security context is created via a negotiation between participants, typically using the WS-Trust model.

An advantage of WS-SecureConversation is that it optimizes multiple secure web service calls between services by performing the authentication step only once for the conversation, by reducing message size with the use of a small context identifier, and by performing only fast symmetric cryptography (using the shared secret keys). WS-SecureConversation uses public-key cryptography to derive shared secret keys for use with the conversation.

WS-Federation

WS-Federation builds upon WS-Security, WS-Policy, WS-SecurityPolicy, WS-Trust, and WS-Secure Conversation to allow security identity and attributes to be shared across security boundaries. As its name suggests, WS-Federation provides a framework for implementing federated identity services.

WS-Federation defines certain entities.

- Principal is an end user, an application, a machine, or another type of entity that can act as a requester.
STS, as defined in WS-Trust, issues and manages security tokens such as identity tokens and cryptographic tokens. The STS is often combined with an identity provider role as STS/IP.

Identity provider is a special type of STS that performs authentication and makes claims about identities via security tokens.

Attribute service provides additional information about the identity of the requester to authorize, process, or personalize a request.

Pseudonym service allows a requester (a principal) to have different aliases for different services and optionally to have the pseudonym change per service or per log-in. Pseudonym services provide identity mapping services and can optionally provide privacy for the requester, by utilizing different identities across providers.

Validation service is a special type of STS that uses WS-Trust mechanisms to validate provided tokens and determine the level of trust in the provided tokens.

Trust domain or realm is an independently administered security space, such as a company or organization. Passing from one trust domain to another involves crossing a trust boundary.

These services can be arranged in different ways to meet different requirements for trust, from simple trust scenarios through to quite complex trust scenarios. The example in Fig. 11 illustrates a fairly complex scenario in which the requester first requests a token from the STS/IP it trusts: 1) The security token is then presented to the resource’s STS to request a token to access the resource; 2) Assuming the requester’s token is valid, the resource’s STS will issue a new token, which is then presented to the web resource to request access; 3) The web service resource at some point needs to perform work on behalf of the principal, so it queries another STS/IP in a separate security domain to obtain a delegated security token; 4) Assuming the web service has the appropriate proof that it is allowed to perform delegation, the STS/IP will issue a security token; 5) This delegated security token is then presented to the resource on behalf of the principal. The chain of trust between the requester and the resource in trust domain C can be followed in Fig. 11.

WS-Federation introduces models for direct trust, direct brokered trust, indirect brokered trust, delegated trust, and federated trust relationships. Other services can be added to the picture, such as attribute and pseudonym services for attribute-based authorization, role-based authorization, membership, and personalization. Pseudonym services store alternate identity information,
which can be used in cross-trust domain scenarios to support identity aliases and identity mapping.

WS-Federation also describes a way for participants to exchange metadata such as the capabilities, security requirements, and characteristics of the web services that form the federation. This exchange of metadata is achieved through the use of another standard called WS-MetadataExchange, which builds primarily upon WSDL and WS-Policy.

**WS-Authorization and WS-Privacy (Proposed Standards)**

As these standards are not yet published, they are mentioned here just for completeness. WS-Privacy is a proposed standard language for describing privacy policies for use with web services. The standard is intended for use by organizations to state their privacy policies and to indicate their conformance with those policies. WS-Authorization is a proposed standard for how to describe authorization policies for web services using a flexible authorization language. The standard will describe how authorization claims may be specified in a security token and validated at the endpoint.

**WS-I Basic Security Profile 1.0**

With the large number of WS-* security standards, vendors are implementing them at different times, and not all of the options are common from one vendor’s system to the next. WS-I Basic Security Profile 1.0 is intended to provide a baseline for WS-Security interoperability among different vendor’s products. The idea is that if the products conform to the Basic Security Profile 1.0, then they should be interoperable at least to some level. This can be important when implementing SOAs with products from different vendors, such as Sun’s Java J2EE, BEA Weblogic, and Microsoft’s .NET platform.

The Basic Security Profile 1.0 supports a good number of security extensions, including Kerberos, SAML, X.509, and username tokens and support for SSL transport-layer security (HTTPS).

**PUTTING IT ALL TOGETHER**

Now that we have covered the suite of Web Services Security standards, we can apply this knowledge to the problem of securing an SOA based on web services.

It is important to note that traditional security principles should form the basis of a secure SOA. The environment in which the systems are running should be managed appropriately to ensure that the organization’s security policies are satisfied, and that regulatory requirements placed on the organization are being met. This includes attention to network security, operating system security, application security (including the web services infrastructure), and physical security. Security risk assessments, threat analysis, vulnerability scanning, and penetration testing techniques should be used to validate the security of the SOA services, platforms, and related systems.

To perform a thorough security assessment, the following types of questions should be asked:

- What does the overall SOA look like?
- Who are the intended consumers of the service (s)?
- How are the services discovered by consumers? Is WSDL or UDDI used?
- What interactions occur between consumers and services and between services?
- Are any of the services or consumers on untrusted networks?
- What types of data are passed between consumers and services at various points?
- Is data integrity or confidentiality required at any point within the SOA?
- Does data flow through multiple intermediaries?
- Is there a need to provide end-to-end security for certain types of data?
- What are the authentication and authorization requirements for each of the services?
- Is the authorization based upon roles or attributes?
- Is data privacy a concern?
- What security technologies, such as X.509, Kerberos, or SAML, are available?
- Are multiple security domains involved? Is there a need for cross-domain trust relationships?
- Are there different web services technologies, such as J2EE, Weblogic, or .NET, in use that might cause issues with protocol support or interoperability? If so, is the WS-I Basic Security Profile 1.0 supported?
- Threat analysis—what potential threats are there to the infrastructure, such as malicious attacks, insider threats, information disclosure, disasters, message replay attacks, or denial-of-service (DoS)?

The following summarizes the types of threats that apply to SOA and mechanisms to mitigate the threats:

- Information disclosure (confidentiality)—Use of XML encryption within WS-Security can provide data confidentiality. End-to-end message confidentiality can also be handled with XML encryption.
- Message tampering—Message tampering could be used to remove XML, add XML, or otherwise alter data or cause some unintended behavior within the application. XML signatures can be used to ensure the integrity of messages.
- Message injection—Message injection may be used to cause some unintended behavior within the
application. Authentication mechanisms and input validation within the service can help to mitigate this issue.

- Message replay—WS-SecureConversation provides mechanisms to prevent this kind of attack, but otherwise, message identifiers or time stamps can be used to prevent message replay.
- Authentication—Authentication is provided by XML signatures and security tokens such as Kerberos, X.509 certificates, and SAML, or even username tokens. These methods are supported by WS-Security and WS-Trust.
- Authorization—Authorization can be role based or attribute based. The web services platform will typically provide some form of authorization capability, but for more advanced authorization needs, the application will have to include explicit authorization checks.
- Service availability—Disasters, whether natural or human-made, need to be planned for by ensuring that an adequate disaster recovery strategy is in place. Other malicious attacks such as DoS can affect the network, operating system, or application. Dealing with DoS attacks is beyond the scope of this entry, however.
- Token substitution—Attempts to substitute one security token for another can be prevented by ensuring that digital signatures provide integrity over all the security critical portions of the message, including security tokens.

Once a risk assessment is completed, and the security requirements are understood, decisions need to be made about how to secure the SOA environment. Risks are normally rated in terms of impact and likelihood and should be prioritized, for example, into high-risk, medium-risk, and low-risk categories. Security measures can then be chosen to mitigate the risks and meet security requirements, based on a cost–benefit analysis.

General security principles should be followed when choosing security measures, such as:

- Ensuring the confidentiality, integrity, and availability of data and services
- Defense in depth
- Principle of least privilege
- Minimizing the attack surface
- Promoting simplicity rather than complexity

At the network level, firewall policies can be applied to limit access to web services, because SOAP messages are transmitted via HTTP, typically on Transmission Control Protocol (TCP) port 80, or via HTTPS on TCP port 443. Internet-facing servers should have access restricted just to the port that the service is listening on. Firewall policies can form the first line of defense by reducing the available attack surface. Other standard techniques, including DMZ architecture, security zones, and intrusion detection/prevention, can reduce risk at the network level and provide defense in depth.

At the transport level, web services are often secured through the use of SSL, via the HTTPS protocol, and policies can be applied through WSDL to ensure that web services are secured with SSL. The use of SSL should definitely be considered, particularly because it is a well-understood protocol, although it is important to understand that SSL provides only point-to-point encryption and that other techniques need to be applied if the security of the SOAP messages is to be maintained beyond the SSL session.

At the message level, XML is by nature a text-based standard, so data confidentiality and integrity are not built in. SOAP messages and attachments may be processed by different intermediaries along the route to the final recipient, and WS-Security allows parts of messages to be targeted to different recipients. This is an important distinction between point-to-point security technologies, such as SSL, and end-to-end security, which WS-Security supports. XML encryption can provide end-to-end data confidentiality via public-key cryptography and shared symmetric-key cryptography, whereas XML signature can meet data integrity and message authentication needs.

Other issues exist when dealing with trust relationships and cross-domain authentication. The WS-Trust and WS-Federation standards provide a technical foundation for establishing trust for SOAs. Organizational security policies and regulatory requirements should define the security requirements that need to be placed on interactions with customers and business partners. These security requirements can be used as a basis for determining the security mechanisms that need to be used to provide an appropriate level of trust, such as encryption strength or method of authentication (X.509 digital certificates, SAML, Kerberos, etc.). However, trust between organizations goes beyond technical implementation details and also needs to be addressed by contractual obligations and business discussions.

CONCLUSION

The WS-S family provides an essential set of standards for securing SOAs; however, the number and complexity of the standards is a definite problem. This complexity can serve to discourage the adoption of these standards into an SOA, particularly for application developers, whose job is complicated by security needs. These standards are also evolving and new security standards are being developed, so expect the SOA security landscape to evolve over time.
Fortunately vendors are providing new tools to simplify the integration of WS-Security standards into web services. These tools can help by hiding many of the lower-level details from security practitioners and architects. Expect these tools to evolve over time as SOA and web services become more mature. At this time, however, it is still not an easy task to integrate WS-Security standards into web services.

For the security practitioner, standard security principles can be leveraged to assist in guiding architects and developers in selecting appropriate mechanisms to secure SOAs.

BIBLIOGRAPHY