Web services are increasingly being provided and consumed in and between cloud environments. Learn how to leverage various interoperable standards to address security challenges in a cloud or distributed Web services architecture.

Web services provide an interoperable mechanism that lets two different systems exchange messages. Securing Web services isn’t any different from securing any other application—it requires authentication, authorization, encryption, digital signatures, and non-repudiation.

However, to highlight the interoperable nature of Web services in addressing these security issues, the Organization for the Advancement of Structured Information Standards (Oasis) established the Web Services (WS)-Security standards. Here, I describe WS-Security and related standards and their roles in cloud computing environments. I also illustrate the importance of such standards and how to integrate them to create a flexible Web services security architecture.

Security Standards for Web Services

WS-Security standards address not only confidentiality and integrity challenges but also challenges related to exchanging various security tokens, such as username and password, X.509, or the Security Assertion Markup Language (SAML). In a nutshell, a WS-Security-compliant message will contain information such as a security token represented as SAML (or some other token type), encrypted data represented as XML Encryption, and digitally signed data represented as an XML Signature.

Although WS-Security might seem straightforward, it’s important to understand the various standards it supports and how to leverage other related security standards to make the integration flexible and interoperable (see Table 1). For example, WS-Security supports SAML as a token type, but when a service provider is configured to require SAML as the authentication token, can the Web service consumer create a SAML token with the necessary information? If not, what standards are available to exchange the consumer’s current token for a SAML token? How can a Web service describe its security policy in an interoperable manner?
Security Assertion Markup Language
SAML referenced directly in WS-Security is widely used as an authentication token along with additional information about the client.
SAML is typically used either when the client needs to exchange attributes other than just basic user information or when an application in a Federated Identity Management infrastructure invokes the Web service. In the latter case, the application typically has a token (such as a cookie or Kerberos protocol) that must be translated to SAML 1.1 or, in some cases, SAML 2.0.

Extensible Access Control Markup Language
XACML specifies XML Schema for authorization decision requests and responses. In XACML, a Policy Enforcement Point (PEP) limits access to various resources. The PEP will interact with a Policy Decision Point using XACML messages to make a decision. PDP will in turn interact with a Policy Administration Point, which stores the policies.
XACML’s SAML profile defines how to protect, transport, and exchange XACML messages. Using the XACML’s SAML profile in WS-Security, Web service providers can implement authorization by leveraging an XACML-compliant PEP.

WS-Trust
The WS-Trust specification enables the exchange of security tokens across various trust domains. An organization can use it internally or with other organizations to help validate, issue, and renew tokens and to translate token formats.
Consider the following cloud computing example. An organization hosts a Web application internally, but the application needs to access Web services hosted by a third party in a cloud. The Web application or Web service consumer can exchange an internal security token (typically a valid single sign-on token) with a WS-Trust-compliant Security Token Service (STS) to obtain a token that the cloud provider trusts.
The security of the Web services defined in WS-Trust can require the incoming message to meet certain security requirements, specified as a policy described in WS-Policy or WS-PolicyAttachment.

<table>
<thead>
<tr>
<th>Standard</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Security Assertion Markup Language (SAML)</td>
<td>Represents identity information in an interoperable manner with additional attributes</td>
</tr>
<tr>
<td>Extensible Access Control Markup Language (XACML)</td>
<td>Describes the policy, requests the authorization decision, and responds with the authorization decision in an interoperable manner</td>
</tr>
<tr>
<td>WS-Trust</td>
<td>A security token service used to change security tokens from one format to another in an interoperable manner</td>
</tr>
<tr>
<td>WS-Policy</td>
<td>Describes the framework or model for expressing the Web services policy in an interoperable manner (for example, describing what parts of the message to sign or encrypt)</td>
</tr>
<tr>
<td>WS-PolicyAttachment</td>
<td>Represents how to attach WS-policy descriptions to subjects such as Web service end points and messages</td>
</tr>
<tr>
<td>WS-SecureConversation</td>
<td>Defines a secure context in which to exchange messages to improve the performance of frequently exchanged messages by using a shared secret or deriving keys from the security context</td>
</tr>
<tr>
<td>WS-SecurityPolicy</td>
<td>Describes the security characteristics of WS-Trust and WS-SecureConversation</td>
</tr>
</tbody>
</table>

Table 1. Various security standards related to the Web Services (WS)-Security architecture.
WS-SecurityPolicy
While WS-Policy describes the constraints and requirements in an interoperable manner, WS-SecurityPolicy aims to describe the assertions to secure the Web services.\textsuperscript{10} WS-SecurityPolicy is mainly designed to describe security characteristics of WS-Trust, WS-SecureConversation,\textsuperscript{11} and WS-Security SOAP Message Security.

WS-SecureConversation
WS-Security defines a mechanism for exchanging security tokens to encrypt and digitally sign messages in an interoperable fashion. However, when exchanging multiple messages, WS-SecureConversation can help define a security context within which messages can be exchanged.

WS-SecureConversation also describes a mechanism for deriving session-key information. It has a specific binding described for WS-Trust so that when requesting a token, the WS-Trust consumer can request a Security Context Token, and the STS (WS-Trust) will respond with one.

Web Services Security Architecture
A Web services security architecture should consider not only the various combinations of where a Web service or Web service consumer can reside but also available interoperable standards. Web services security can be architected to accommodate various security tokens, enforce access control, and maintain transaction confidentiality and integrity.

In designing security for Web services, we must address various questions. For example, how should we describe what security tokens

**Figure 1. Services interacting in a cloud.** This exemplifies interactions between a Web service provider and a service consumer in the infrastructure as a service (IaaS), software as a service (SaaS), or platform as a service (PaaS) models along with the corporate environment.
are acceptable and what parts of the message should be encrypted and digitally signed?

Also, when the service is invoked, how should we validate the security token, translate it into an internal token for validation, and enforce access control before executing the service or operation? Figure 2 shows how the various standards can help answer these questions.

The Web services security architecture in Figure 2 leverages the interoperable standards to

- define what security tokens to accept and what parts to sign or encrypt using WS-Policy or WS-PolicyAttachment;
- validate the token and obtain a SAML token along with XACML information—the Web service interacts with the STS (WS-Trust) to accomplish this;
- define the security for the STS using WS-Policy (or WS-PolicyAttachment or WS-SecurityPolicy);
- increase performance using WS-SecureConversation for frequent message interactions; and

The design of the consumer application, on the other hand, should have the flexibility to translate any client application token into the token the service provider requires. It should also be able to digitally sign and encrypt the message or parts of it.

Figure 3 shows the interactions between a Web service consumer and an STS (WS-Trust) to translate a client application token.

**Web Services Security Walkthrough**

Figure 4 shows a deployment scenario in which a cloud provider offering IaaS deploys a Web service that authorized business partners can access. An organization subscribes to a SaaS provider, so its business partners receive access to the Web services hosted at the IaaS cloud provider. The organization deploys an internal application that consumes services from the SaaS provider.

The organization has clearly leveraged interoperable standards to secure Web services interactions. Consider its use of the IaaS. It uses WS-Policy to describe its Web services security policy, and it has a dedicated virtual private network (VPN) tunnel for its corporate network, which leverages the STS based on WS-Trust to translate tokens. It also exchanges an incoming username and password (invoked by a business partner) token for a SAML token with an XACML profile. Finally, it uses an XACML-compliant PEP to enforce access control for service requests.
For any internal applications that the organization developed or owns, the organization deploys (according to WS-Trust standards) an STS for Web services’ and consumers’ use. Each internal application also invokes a Web service exposed by the SaaS provider with a SAML token. Furthermore, each application (for the Web service or consumer) exchanges its internal application token for a SAML token through the STS. Finally, the client application invokes the SaaS Web service with the SAML token.

One advantage of this architecture is the use of STS, SAML, XACML, and WS-Policy. By adhering to the interoperable security standards, organizations can not only make their services available across various platforms but also ensure a higher level of security.

In the example, the organization can deploy services in the cloud (IaaS) or use services exposed by a cloud provider (SaaS) and still control who accesses the services and what they can do. Also, the STS maintains control over which users or business partners can access the service and what operations they can perform (by sending XACML information). It also controls who can access the SaaS Web service, even though access is through an internal application (it translates the internal application token to the required SAML token).

**Security Token Service**

Figure 4 shows how to leverage an STS deployed by a cloud provider (IaaS) in an internal corporate network with a dedicated VPN tunnel. However, you could deploy an STS at any number of places based on the need for a service provider or consumer to support multiple tokens.

For example, there could be one STS for every group of Web services (typically grouped by either the organization or cloud service model) that can translate the tokens received from various service consumers. The same concept can apply to service consumers. Each service consumer or set of service consumers can trust an STS to translate client credentials into the required token.

**Figure 4. A sample Web services security architecture. The organization leverages interoperable standards to secure Web services interactions.**
The standards I mentioned here aren’t the only ones that address Web services security. Other standards exist, including the Oasis Digital Signature for signature processing, the XML Key Management Specification\(^\text{12}\) for distributing and registering public keys, and the WS-Metadataexchange\(^\text{13}\) (proposal) for exchanging metadata for Web services security. However, the standards mentioned here address the most common challenges experienced when securing a service.

The architecture I outlined will be more beneficial in cases where there’s a mix of heterogeneous systems and cloud service models, and a need to control access from one place (authentication and authorization) and express policies in an interoperable manner. However, when the deployment scenario is very limited, as long as appropriate risks are understood, a variety of alternate options—such as such HTTPS mutual authentication and Internet Protocol Security—are available.

References


Sitaraman Lakshminarayanan is a lead systems engineer for GE Energy and is responsible for the smart grid security architecture. His research interests include systems security architecture, service-oriented architecture, and identity and access management. Lakshminarayanan received his MSc in computer science from Madurai Kamaraj University. Contact him at sitaraman.lakshminarayanan@ge.com.
This article was featured in computingnow.computer.org.

Top articles, podcasts, and more.

computingnow.computer.org