Securing Web services

by M. Hondo
N. Nagaratnam
A. Nadalin

The Web service security challenge is to understand and assess the risk involved in securing a Web-based service today, based on our existing security technology, and at the same time track emerging standards and understand how they will be used to offset the risk in new Web services. Any security model must illustrate how data can flow through an application and network topology to meet the requirements defined by the business without exposing the data to undue risk. In this paper we propose a mechanism for the client to provide authentication data, based on the service definition, and for the service provider to retrieve those data. We also show how XML Digital Signatures and encryption can be exploited to achieve a level of trust.

Two terms have emerged for describing loosely coupled services delivered over the Internet. These interchangeable terms set the stage for a major evolution of application and integration styles. The year 2000 is rapidly becoming the year in which software services emerged as a viable means of flexibly delivering resources to businesses and consumers over the Web . . . Two terms, "e-services" and "Web services," are recognized as describing the phenomenon of delivering software as services in the Internet world.
—Gartner Research Note[1]

To fully support a service model as described by the Gartner Research Note, there will also need to be a model for security, reliable messaging, quality of service, and management for the Web services stack. The Web service security challenge is to understand and assess the risk involved in securing a Web-based service today, based on our existing security technology, and at the same time track emerging standards and understand how they will be used to offset the risk in new Web services. Any security model must illustrate how data can flow through an application and network topology to meet the requirements defined by the business without exposing the data to undue risk. A Web service security model must support protocol-independent declarative security policies that Web service providers can enforce, and descriptive security policies attached to the service definitions that clients can use in order to securely access the service.

Is a Web services security layer *really* required? The industry already has a set of existing and widely accepted transport-layer security mechanisms for message-based architectures, such as SSL (secure sockets layer) and IPSec (Internet Protocol Security); why add another? To answer these questions we examine various components that constitute Web services and also explore some possible approaches to securing Web services.

Web services

The Web service paradigm includes a programming model for application integration that does not discriminate between applications deployed inside and outside the enterprise. Integration and development of Web services can be done in an incremental manner, using existing languages and platforms and adopting existing legacy applications. However, to

0018-8670/02/$5.00 © 2002 IBM

achieve a common program-to-program communication model, it must be built on Web service standards and communicated over standard protocols.

IBM's Web services[2] are intended to complement Java** 2 Platform, Enterprise Edition (J2EE**) and other standards and allow for a full range of tightly coupled distributed and nondistributed applications. Web services provide a method to deploy and provide access to business functions over the Web, and J2EE and other traditional programming standards are technologies for implementing the provider instance of the Web service.

One of the anticipated benefits of Web services is that direct application-to-application communication can replace the human end-user interaction in the current data entry Web applications. Web service standards have been defined to allow enterprise companies to rapidly define simple business-to-business interfaces and exchange messages. Additional requirements for a Web services infrastructure include support for service context, conversations and activities, intermediaries, portal integration, and workflow and service flow management.

XML. Extensible Markup Language (XML)[3] began as an extensible data format to address the limitations of HTML (HyperText Markup Language), but has become a standard for communication between applications. With XML, an application defines markup tags to represent the different elements of data in a text file so that the data can be read and processed by any application that uses XML.

Using XML, it is possible for business persons to define policies and express them as XML documents. These documents can have sections that are encrypted and the documents themselves can be digitally signed, distributed, and then interpreted by the security mechanisms that configure the local software. This allows various implementations to map from the XML description to a local platform-specific policy enforcement mechanism without requiring changes to the infrastructure.[4]

SOAP. Simple Object Access Protocol (SOAP)[5] is a simple, lightweight, and extendable XML-based mechanism for exchanging structured data between network applications on top of widely used Internet standards such as XML and HTTP (HyperText Transfer Protocol). SOAP consists of three parts: an *envelope* that defines a framework for describing the contents of a message, a set of *encoding rules* for expressing instances of application-defined data types, and a convention for representing *remote procedure calls* (RPC) and *responses*. SOAP can be used in combination with a variety of network protocols such as HTTP, RMI/IIOP** (Remote Method Invocation/Internet Inter-ORB** [Object Request Broker] Protocol), SMTP (Simple Mail Transfer Protocol), FTP (File Transfer Protocol), or MQ (Message Queuing).

SOAP can potentially be used in combination with a variety of other protocols; however, the only bindings we discuss here are SOAP in combination with HTTP and HTTP Extension Framework.[6] The usage of a particular protocol does not change the fundamentals of the security model but may change the particular implementation.

The SOAP envelope is defined in XML and enables a large variety of meta-information to be added to the message, such as transaction identifiers, message routing information, and message security. The envelope consists of two parts: header and body. The header is a generic mechanism for adding features to a SOAP message. All immediate child elements of the SOAP header element are called header entries. The body is a container for application data intended for the ultimate recipient of the message. Thus, SOAP can be considered as another layer, between the transport layer (e.g., HTTP) and the application layer (e.g., business data), that is a convenient place for conveying message meta-information.

WSDL. Web Services Description Language (WSDL)[7] is essentially an XML IDL (interface definition language) that provides a way to describe the function and interface of a service. It is a format for describing network services as a set of endpoints operating on messages containing either document-oriented or procedure-oriented information. The operations and messages are described abstractly, and then bound to a concrete network protocol and message format to define an endpoint. Related concrete endpoints are combined into abstract endpoints (services). WSDL is extensible to allow description of endpoints and their messages regardless of what message formats or network protocols are used to communicate; however, the only currently described bindings are for SOAP 1.1, HTTP GET/POST, and MIME (Multipurpose Internet Mail Extensions).

The WSDL service information can be extracted from a UDDI (Universal Description, Discovery, and Integration[8]) business service entry, or may be obtained from other service repository sources.

Regardless of the source, both run-time and development tools can use WSDL to determine run-time bindings to a service. This information can be used to build the logic to access the service either directly from the client or through generated code stubs.

WSDL has the potential to be extended[9] to include the definition of the context needed by the business execution environment, including security. Without these extensions, users will make assumptions about security in the run-time environment of a Web service. Defining these security assertions in XML will allow us to have a common interpretation of security attributes in different implementations. It will also facilitate searching.

Secure Web services

There are fundamental business reasons underlying the existence of various security mechanisms. The *authentication* of the entity asserting an identity when requesting a service allows businesses to offer different classes of service to different customers. The business reason for *data integrity* is to ensure that each party in a transaction can have confidence in the business transaction. There is also a business and legal need to have an *audit trail* and some evidence of *nonrepudiation* to address liability issues. And more and more businesses are becoming aware of the internal threats to their applications by employees or others *inside* the firewall. Some business transactions require that *confidentiality* be provided on a service invocation or its data (such as credit card numbers). Also businesses on the Internet need to protect themselves from *denial-of-service* attacks. This is the environment in which we need to provide a security service model.

Moving forward from prototypes of Web services will require an open security service model, based on standards, that can serve a heterogeneous "trust domain." This security model should support interfaces for security services that:

1. Use XML data formats as the common representation of various security assertions
2. Accept policy information expressed in XML to configure services (extending WSDL)
3. Use XML messaging as the secure mechanism for exchanging the XML security assertions and also for servicing Web service requests

Security technologies. To evolve from existing applications will initially involve wrapping legacy code with Web services. Some of the security constraints and trust models of existing applications will need to be carried forward and expressed within the early versions of Web services. To accomplish this we will need to incorporate the work begun, in the various Web services toolkits, on security technology from basic authentication to XML digital signature support.

Some of the security constraints of existing applications will need to be carried forward in early versions of Web services.

As these security technologies themselves become services, and workflow becomes the primary application paradigm for dynamic application integration, security services will evolve into core elements of a secure application workflow.

A variety of security technologies are being adopted as standards. Following is a brief overview of these standards and how they can be used.

XML Digital Signatures[10] is a standard for securely verifying the origins of messages. The XML signature specification allows XML documents to be signed in a standard way, with a variety of different digital signature algorithms. Digital signatures can be used for validation of messages and for nonrepudiation.

Security Assertion Markup Language[11] is the first industry standard for secure e-commerce transactions using XML. SAML is being developed to provide a common language for sharing security services between companies engaged in business-to-business and business-to-consumer transactions. SAML allows companies to securely exchange authentication, authorization, and profile information between their customers, partners, or suppliers regardless of their security systems or e-commerce platforms. As a result SAML promotes the interoperability between disparate security systems, providing the framework for secure e-business transactions across company boundaries.

XML Encryption[12] will allow encryption of digital content, such as Graphical Interchange Format (GIF)

images, Scalable Vector Graphics (SVG) images, or XML fragments. XML Encryption allows the parts of an XML document to be encrypted while leaving other parts open, encryption of the XML itself, or the superencryption of data (i.e., encrypting an XML document when some elements have already been encrypted).

As part of the Java Community Process (JCP), there are two Java specification requests (JSRs) that are currently in progress; JSR105 XML Digital Signature [13] and JSR106 XML Digital Encryption. [14] When complete, these two JSRs will define the standards in Java for each technology, thus standardizing the interfaces in each vendor's Web services toolkit.

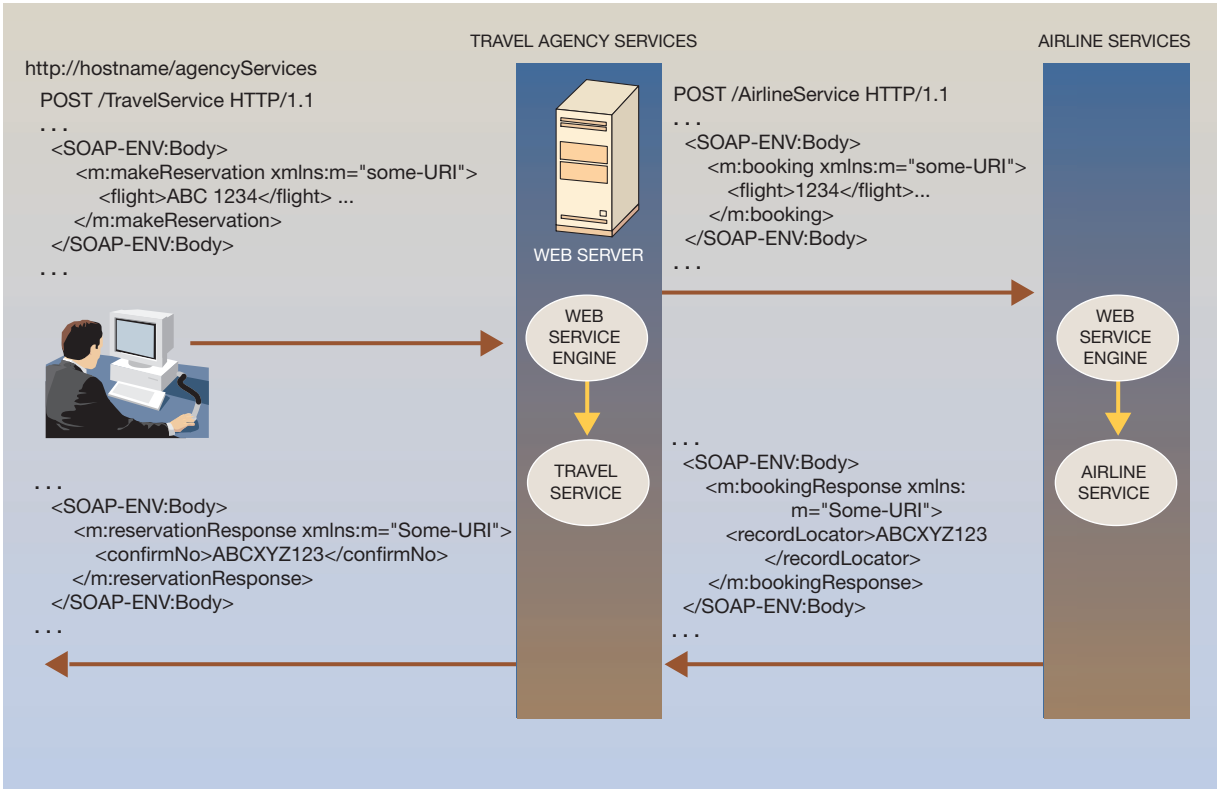
Application patterns

Patterns are available for evolving to Web services from existing Web server applications. One is the browser-to server-pattern. This pattern wraps an existing application as a service, using a SOAP message as the service invocation. The Web server provides a run-time execution container that defines its own security model, with policy information derived from a deployment descriptor configured by the deployer of the Web server application. This pattern typically includes a mechanism for associating the identity of the invoking entity (the browser client) with the executing application instance, and allows the application to continue to function as it did before. With J2EE, an identity is provided as part of authentication, and the container associates that identity to the security context. This pattern uses an *implicit* trust model in the sense that the client and the server rely on the middleware configuration to ensure that the identity is established within a security context provided by J2EE itself. An advantage of this model is that the run-time code maintains the identity mapping, and name assertion life-time constraints and mechanisms for maintaining a valid token can be provided by the middleware. One of the disadvantages is that the model for "delegating" an identity requires that the delegated application-level identity is the same as the invocation identity of the intermediary (and hence the security context). This creates a coupling between middleware and the application-level delegation logic (i.e., run-as element in J2EE) and the requirement for the security context to support cascaded delegation, auditing, and nonrepudiation.

Another pattern involves rewriting the application with a modular design to create smaller tasks that can be combined in different ways to perform more complicated transactions. Each component is able to externalize its output into a message that the next component can use as input. This pattern uses SOAP messages to trigger each event. The messaging agents and message queues can be built into the run-time server below the application level. Sometimes the messaging agents become the "security-aware" part of the run-time code and control the flow of information along its path through components based on security attributes in the header of the message. Sometimes the security attributes get added into the message structure itself, as is the case with digital signatures. The trust model for this type of messaging relies on the specification of an *explicit* trust model along the lines of SAML. In the SAML-type name assertion, the trust will be explicit [15] in the sense that the client and the server rely on coupling the identity information explicitly with the message, rather than on the underlying security context. Of course, this requires that the service handler be able to establish the identity of the caller based on the SAML-type name assertions and on trusting the entity that created these assertion tokens. Thus in a SAML trust model, an authentication/authorization authority has to be known and to digitally sign the assertions at the time of the authentication/authorization event. A certificate associated with the signature can be used to identify the trusted authority and validate the signature, and a time stamp is included to indicate the assertion validity period. An advantage of this type of trust model is that the message can pass through multiple intermediaries. Authorization and delegation decisions can be made in a standard way by the intermediaries without modifying the name assertion of the originator of the message request. If implemented in an "envelope," it is also possible to build audit trails capable of asserting evidence of nonrepudiation, since each intermediary could wrap a message with its own name assertion. A disadvantage of this model is that the last component has to do some additional processing to make sure that the originator name assertion is valid from both a trust and a time standpoint.

Both these patterns implement security in the run-time code and both rely on a mapping of an external form of an identity into a run-time interpretation of that identity and into a set of rules about the identity and its capabilities. The difference has to do with where the mapping is done and whether the information is in an externalized form that can be middleware-independent and persistent.

Figure 1 Usage scenario



A usage scenario

Consider a very simple scenario: a travel agency system contacts an airline reservation system in order to complete a travel transaction request (see Figure 1). The two applications, Web services for a travel agency and an airline, use XML to exchange travel itinerary information and payment details, using a well-understood industry standard specification. The specification requires that applications mark reservations with the <makeReservation> tag.

The travel agency's application is designed to accept a reservation request with a <makeReservation> tag from a customer. Based on the airline details requested, the application will contact the airline's reservation service. The airline's application is designed to perform a transaction under which a reservation for the passenger is made and charged to the credit card system. Note that the airline can even contact the credit card company's charging service in order to perform that operation.

A user searches for the best possible travel deal and decides to purchase the ticket from our travel agency. The user submits a request to access the agency's reservation Web service by specifying the itinerary and credit card details in a SOAP message marked with a <makeReservation> tag. The agency application requests a reservation to be made to the airline reservation system by sending a booking request, providing the agency's details and the passenger details. When the airline system receives the booking request, the application queries the database for the availability of seats for that particular itinerary. Using XML, the airline application returns a response after making the reservation in its back-end system. The response marks the details with record locator number as <recordLocator>ABCXYZ123</recordLocator>. When the travel agency receives the message from airline, the application scans it for the <recordLocator> tag, and upon finding it passes the data into its system to issue a ticket to the end user.

The Web service that provides reservation service in a service network can be combined with other services, such as a service to obtain weather information for the travel destination and a credit service to charge the end user for the travel expense. This way the travel agency can *automatically* issue tickets and schedule ticket delivery with shipping companies through their Web services.

This example describes a simple Web service where two or more functions cooperate through the Web, and the nature of their cooperation adapts to parameters provided by a particular request. As we see, Web services provide a framework for loosely coupled (dynamic) application integration.

Web service provider security

In the previous example we described how loosely coupled applications can be combined to provide a Web service-based e-business solution. Users can invoke Web services using one of the supported protocols. For example, users can submit SOAP requests over HTTP to be processed by the Web service engine. When Web services are used to provide business-related data, those data need to be secured.

The Web application server that hosts the Web service engines can support various protocols. Sometimes, due to the bundling of security with protocols, tasks need to be categorized for performance as either protocol-specific or as protocol-neutral security tasks. A transport protocol-specific handler performs security tasks such as SAML authentication or identity assertions when the authentication information is passed as part of the transport protocol, whereas a transport protocol-neutral authorization handler can perform authorization based on this authentication information regardless of the transport. Once the user is authenticated, the Web service security provider should determine if the invoking user has the authority to invoke the service. This authorization decision is based on the authorization policy associated with the Web service.

Authentication

In order to secure access to a Web service, the user requesting the service needs to be identified through underlying authentication mechanisms (see Figure 2).

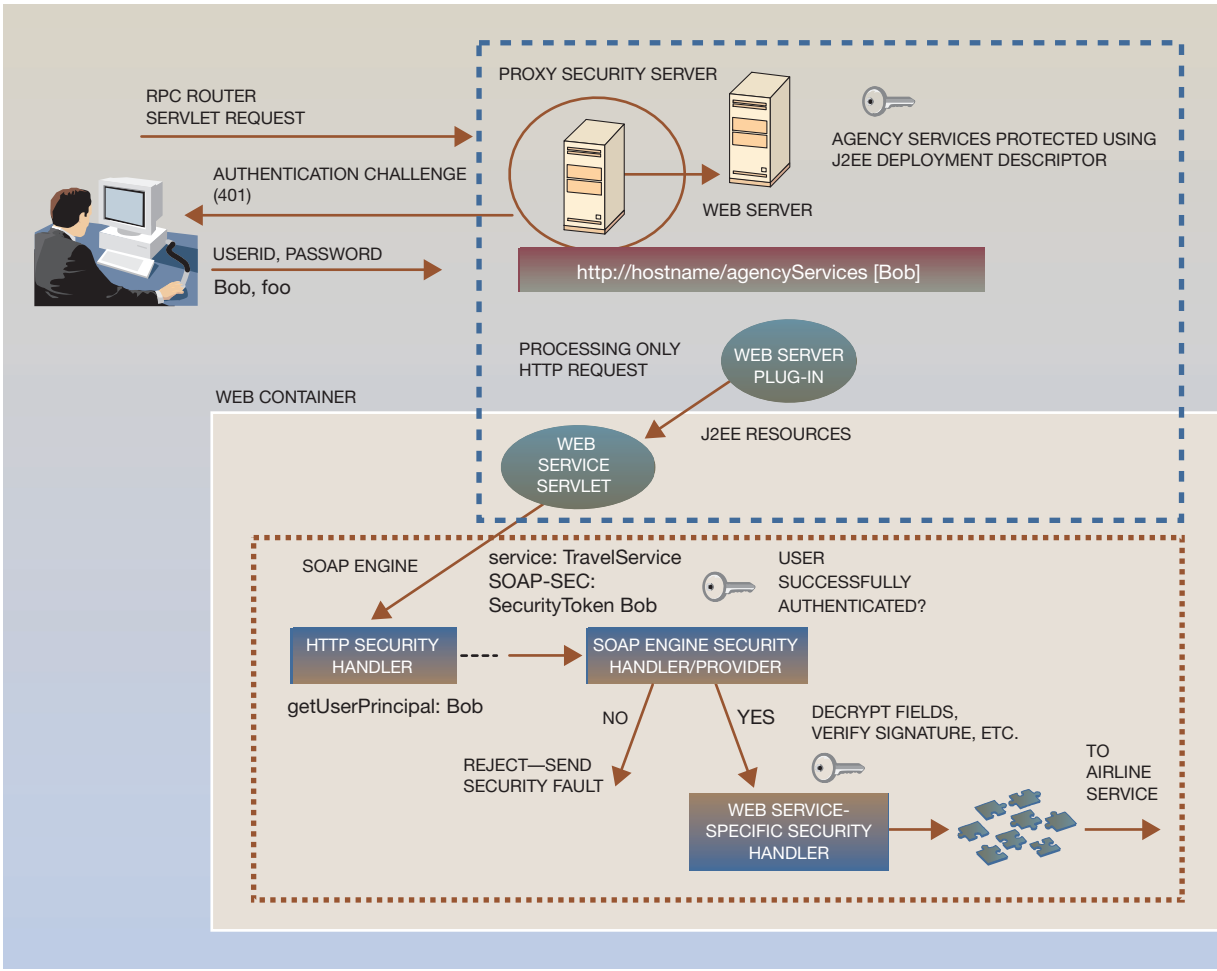
Authenticating the user. An identity needs to be associated with a request in order to enforce authorization policies. A Web service security configuration should specify authentication policies that define how the user credential (or authentication data) is to be retrieved as well as how it is to be verified. In the J2EE security model, the log-in configuration policy specifies how user information is to be retrieved (e.g., HTTP Basic) and the operational environment-specific policies will dictate how it gets authenticated (e.g., Kerberos authentication mechanism). In the Web service security model, the log-in configuration should not only address authentication of immediate clients (clients submitting the request directly to the service), but also address indirect clients (an end client's identity may be part of the request, which can traverse through many intermediaries). This information is necessary for both the client (to submit the credential information in a format understood by the server) and the server (to retrieve the credential information from the transport layer or the message itself).

A user may be authenticated by the Web service engine, or before the Web service engine is given the request.

The WSDL definition of the service will include security constraints on how the credential information is expected to be provided (or retrieved). For example, in the case of an HTTP-bound service, data are expected to be supplied through HTTP Basic authentication. In the case of a service that can be accessed through an intermediary, the WSDL definition should indicate that the credential information is expected as part of the message itself. For example, it could say that the authentication information is expected in the "SOAP-SEC: SecurityToken" header.[16] Based on these two possibilities, we could state that authentication data are supplied using either a transport-specific mechanism (e.g., HTTP Basic), or transport-neutral mechanism (e.g., SOAP-SEC) header but not both.

Depending on the network topology, a user may be authenticated by the Web service engine or before the Web service engine is given the request. For example, when a user submits a request to the servlet that dispatches the Web service request, the user

Figure 2 Web service authentication

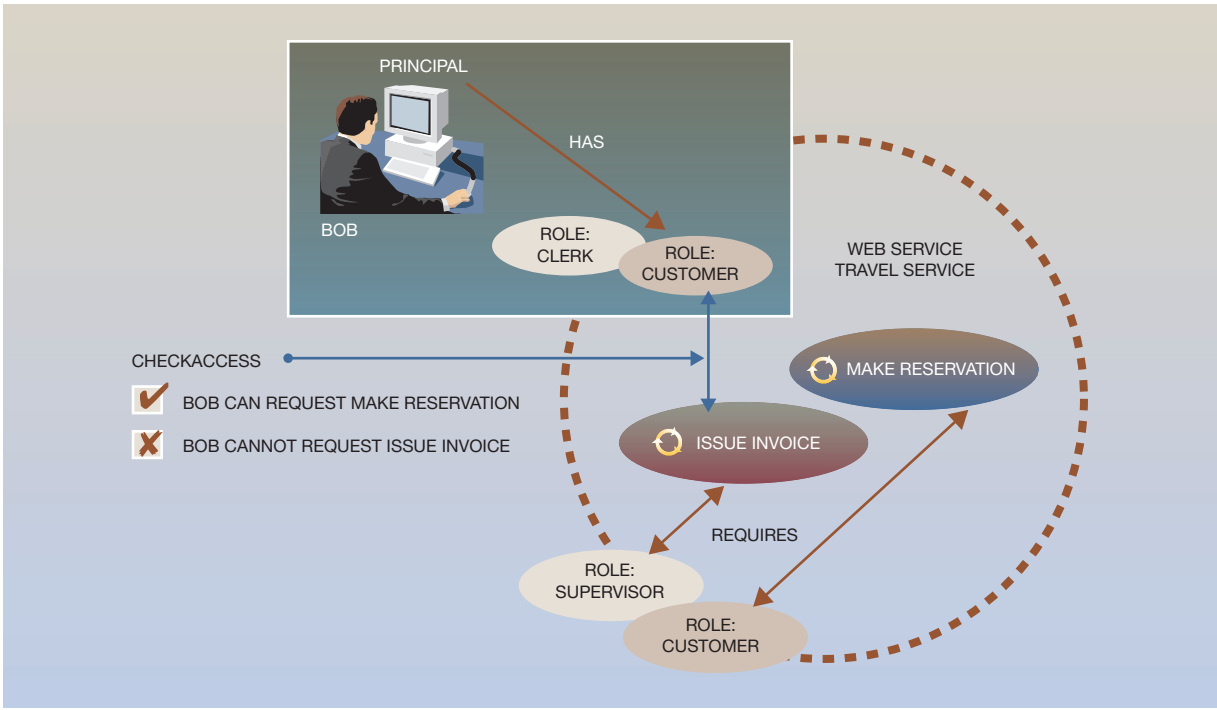


might be challenged by a front-end reverse proxy server, using the HTTP 401 challenge mechanism, so that the user can submit his or her credentials (user identifier and password).

In order to handle the authentication data that get passed around through different means, a Web service engine will use transport-specific handlers as well as transport-neutral handlers. For example, in a J2EE environment where HTTP is the transport protocol used, the HTTP-specific handler (i.e., servlet) can invoke the getUserPrincipal API (application programming interface) to retrieve the identity of the user. If the user is not already authenticated as far as the underlying system is concerned, the handler should authenticate the user based on the credentials sent over the transport (e.g., user identifier and password in the HTTP header). This transport-dependent approach to perform authentication will allow the authorization and downstream processing of the call to be transport-neutral. If identity information is present in the message itself, the security handler will retrieve those data and validate the information.

Asserting the user identity. Once authenticated, the identity of the user must be securely associated with the context of execution to be used in the downstream requests. To do so, the authentication handler asserts the identity of the user within the request. For example, a special header, containing the asserted identity of the user, digitally signed by the security service, can be attached to the request. The

Figure 3 Authorization scenario



user name could be replaced by the Kerberos credential of the user. By establishing the user identity within the execution context, the Web service runtime component (e.g., J2EE container) can perform authorization based on the user identity.

For example, a J2EE application server can challenge user "Bob" when he tries to access a Web service. When the Web service is invoked, the HTTP security handler will call the getUserPrincipal routine to validate "Bob." In the case of a SOAP binding, the authentication handler may insert a "SOAP-SEC: SecurityToken Bob" header asserting that the calling principal is "Bob."

Authorization enforcement

Based on the authorization policies specified for the services, the J2EE container will enforce the authorization policies. For example, user Bob would be allowed to access our travel agency service only if he is authorized to do so.

In order for the Web service provider to enforce security constraints on Web services, the security policies must be declaratively conveyed to the run-time authorization policy endpoint to ensure portability of the service from one Web application server to another. One approach is for the service *provider* infrastructure to take care of defining the security policies. Then the Web service *developers* need not know how to programmatically enforce all possible authorization policies for the different servers in which the service may be deployed.

In order to maintain a level of abstraction of policies during service installation and to give the ability to specify application security policies during Web service development, Web service security policies should be expressed in terms of roles. As in J2EE, a security role is a semantic grouping of permissions that a given type of user must have in order to successfully use the Web.

As shown in Figure 3, a user who is a registered customer to our travel agency is granted the permission to make travel reservations using a travel service.

In order for an authorization check to succeed, the user invoking the service must have at least one of

Table 1 Principal role-mapping example

Subjects	Roles		
	Clerk	Customer	Supervisor
AdminGroup	X		
CustomerGroup		X	
GoldCustomers		X	
ManagerGroup			X
Bob	X	X	

the roles required to access the service. The mechanisms by which users and user groups are granted application roles are specific to the operational environment.

As shown in Figure 3 and Table 1, the makeReservation method on the travel agency Web service requires the role "Customer." For example, if user Bob makes a request, his identity is associated with the context of execution. The security run-time component will make the authorization decision based on the identity of the caller invoking the service. The ability to specify authorization policy declaratively helps decouple the application security policy from organizational security policies, while still effectively enforcing overall security requirements.

Trust. The IETF (Internet Engineering Security Task Force) security glossary[17] contains several definitions of trust. One of them asserts: "Generally, an entity can be said to 'trust' a second entity when it (the first entity) makes the assumption that the second entity will behave exactly as the first entity expects. This trust may apply only for some specific function."

Trust is emerging as one of the central issues of all communications in the Internet age. How do we evaluate and find that which is trustworthy and discard that which is not? Trust in itself is a very fuzzy concept. There are ways in which we implicitly trust, such as trusting friends to help if your car breaks down, or explicitly trust, such as contracting for work to be done on your house. The evolution of XML-based vocabularies for e-commerce raises the question of how the trust in these new documents will be defined and enforced.

XML Digital Signatures offer an interesting framework for working with this question. Individuals create signatures that can be applied to "any digital content (data object)." Digital signatures can provide integrity, signature assurance, and in some cases evidence of nonrepudiation of Web data. This is especially important for documents that represent commitments, such as contracts, price lists, and manifests. The work on XML Digital Signatures addresses the digital signing of documents (any Web resource addressable by a uniform resource indicator [URI]) using XML syntax. This capability is critical for a variety of electronic commerce applications, including tools for payment.

The W3C XML DSIG specification[10] addresses the signing of parts of an XML document as well as the entire document. When combined with SOAP (as recommended to the W3C by IBM and Microsoft)[16] XML DSIG can be used by the sender of the message to generate a signature over the body of the SOAP message. The receiver is able to parse the signature element in the SOAP header and either extract the certificate needed to verify the signature from the message itself, or find a reference to a key[18] that was used to sign the body of the SOAP message.

Establishing trust. When a Web service delegates part of its work to another service, it may establish a broad mutual trust relationship, or it might need to establish trust on every request submitted. In any kind of trust relationship, there can be different levels of granularity at which a request (or response) needs to have trust asserted. In a paper-based workflow, it may be sufficient to seal and sign the letter envelope that contains many documents. Similarly, in a Web services-based workflow, it may be sufficient to sign the SOAP header to assert that the documents attached to the SOAP body can be trusted and are not tampered with.

Web services can use the XML digital signature capability in different ways to achieve different levels of trust. In our previous example, the travel agency and the airline establish a broad mutual trust relationship and agree to digitally sign requests and responses. The SOAP engine used by the travel agency signs the request and sends it across the HTTPS (HTTP over SSL) connection. The request may contain several documents. Figure 4 illustrates an example of how XML DSIG is used to sign a request to issue tickets.

If the two entities involved in this trust relationship were concerned about internal security, additional trust, including the signing of individual documents or fields and not just the envelope, may be required. The two entities may also require that all requests

Figure 4 Travel service example

```
<SOAP-ENV:Envelope xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/">
  <SOAP-ENV:Header>
    <SOAP-SEC:Signature
xmls:SOAP-SEC="http://schemas.xmlsoap.org/soap/security/2000-12">
      <ds:Signature xmlns:ds="http://www.w3.org/2000/09/xmldsig#">
         . . .
         <ds:Reference URI="#BODY"../>
         . . .
      <ds:Signature>
 <SOAP-SEC:Signature>
<SOAP-ENV:Header>

<SOAP-ENV:Body xmlns:SOAP-SEC="http://schemas.xmlsoap.org/soap/security/2000-12"
       SOAP-SEC:id="Body">
    <m:IssueTickets xmlns:m="some-URI">
    <m:company>Travel-R-Us</m:company>
         <m:ticketholder>Joe Someone</m:ticketholder>
         <m:customerCC>Visa</m:customerCC>
         <m:CC#>111-222-3333</m:CC#>
    . . .
    </m:IssueTickets
  </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

contain encrypted data elements,[12] such as credit card number, as well as signed elements. In this approach, even if the individual documents are extracted from the envelope and used later in a different context (e.g., stored for later reference), the documents contain persistent security attributes, which can be verified at any time. In a paper-based workflow, this would be the equivalent of requiring that individual letters in an envelope be signed.

Figure 5 illustrates how XML DSIG could be used by the airline to sign each "IssueTicket" content of the response it sends to the travel agency. In this scenario, the XML element, not the SOAP message or body, is the object of the signature. In addition to these signed data, the two companies would probably need to use the HTTPS protocol to provide a secure authenticated communication channel over which these signed XML elements can be exchanged.

A further evolution of a simple travel workflow scenario would be possible if the travel industry defines standard XML elements (like signed Ticket elements) and a distributed workflow to replace current point-to-point implementations as in this example.[19]

Trusted third parties. If there are only two parties engaged in an exchange, a trust relationship can be established between them. However, as Web services enable the discovery of and dynamic binding to new partners, a third-party trust relationship may be needed. Two entities involved in a contract may not know each other but may each trust a common third party. In such cases, they may want the third party to be the one to sign and verify that the other party is trustworthy. In a paper-based workflow, a neutral third party or notary may be needed as a witness to a transaction. Similarly, in a Web services environment, a third party may be required to assert the validity of a request, or a response, or even the credentials of the requestor or the target.

In the previous example above, both the travel agency and the airline might agree to use public key cer-

Figure 5 Data signing example

```
<Signature Id="Travel-R-Us01" xmlns="http://www.w3.org/2000/09/xmldsig#">
  <SignedInfo>
    <CanonicalizationMethod Algorithm="http://www.w3.org/TR/2000/CR-xml-c14n-20001026"/>
    <SignatureMethod Algorithm="http://www.w3.org/2000/09/xmldsig#dsa-sha1"/>
    <Reference URI="#IssueTicket" />
      <Transforms>
        <Transform Algorithm="http://www.w3.org/TR/2000/CR-xml-c14n-20001026"/>
      </Transforms>
      <DigestMethod Algorithm="http://www.w3.org/2000/09/xmldsig#sha1"/>
      <DigestValue>YWJjZGxma3NqamRlZmZnaGlvcztkbGZramFzZGw7Cg==</DigestValue>
    </Reference>
  </SignedInfo>
  <SignatureValue>YWJjZGxma3NqamZmZnaGlvcztkbGZramFzZGw7Cg==</SigntureValue>
  <KeyInfo>
    <KeyValue>
      <DSAKeyValue>
        <P>...</P><Q>...</Q><G>...</G><Y>...</Y>
      </DSAKeyValue>
    </KeyValue>
  </KeyInfo>
</Signature>
<IssueTicket>
    <PassengerName>...</>
    <ShippingAddress>...</>
    <BillingInfo>...</>
    . . .
</IssueTicket>
```

tificates and key pairs generated by a trusted third party, a "travel better business bureau." In this pattern, each party checks that the other is a legitimate member of this trust domain by contacting the third party, which could perform an on-line validation[20] of the certificates for certified travel businesses as a value-added Web service. Once the travel agent gets the okay from the travel better business bureau, it accepts the signed requests of the airline.

Auditing and nonrepudiation

An audit capability can capture a tamper-resistant record of security-related events in order to evaluate the effectiveness of security policies and mechanisms. This paper does not discuss requirements for auditing security-relevant events in Web services, or APIs for Web service containers to generate audit records. A future version of the Web services security specification may include such requirements.

Nonrepudiation as a legal term implies providing legal evidence that a user performed some action, such that the user cannot reasonably deny having done so. In some cases, it is claimed that XML Digital Signatures can be used to provide signed evidence of the XML messages exchanged between the services. But an end-to-end solution to nonrepudiation is a difficult problem and involves trust issues. Therefore, this paper does not propose a solution for nonrepudiation, which should be addressed in the future.

Summary

A Web service security model should address security issues involved in a request from an end client to a target service, including the intermediary services that route the service requests. This paper proposes a mechanism for the client to provide authentication data based on the service definition and, at

the same time, for the service provider to retrieve those data.

Because of the necessity for and complexity in established trust in the Web services model, this paper also proposes how XML Digital Signatures and encryption can be exploited to achieve a level of trust. We show that as part of its evolution, the Web services requirements for application development can be seen as an opportunity to introduce a method of coupling security technologies (authentication, authorization, digital signatures, etc.) with business trust issues (public key infrastructure policy, role-based access control, firewalls, etc.) and workflow into the creation of core Web security services configured through policies expressed in XML.

As the base Web service technology evolves, more complex scenarios need to be considered and handled in the future.

Future

As we begin to secure Web services, we can also evaluate service offerings through the perspective of risk assessment. No Web service is impenetrable to attack. Every offering of a service implies risk, whether on the Internet or on an intranet, because there is always a risk when a service interface is externally exposed. The question is: what is the scope of the exposure, how can the exposure be offset by the potential value of the service, and who is responsible for the enforcement of any countermeasures used to prevent the service interface from being exploited?

The ability to supply a complete Web service environment, in which risk assessment and policy enforcement are an integral component, will depend on several initiatives continuing to evolve as *de facto* standards. First, the workflow needs an integrated security model as part of its processing model. Second, analysis is needed to determine whether XML schemas can be used to formalize security models through the definition of security types. Third, Web Services Description Language, Web Services Flow Language,[21] or Web Services Endpoint Language need to be extended to contain security attributes and policy information related to Web services. In particular, the specification of PKI policies will be important for interoperability in the more dynamic Web services involving late binding. Finally, the emerging W3C activities to define the processing rules and key management services for XML applications must be well integrated with the other Web services specifications.

Appendix A: Glossary

access To access is to interact with a system entity in order to manipulate, use, gain knowledge of, or obtain a representation of, some (or all) of a system entity's resources.

access control Access control protects resources against unauthorized access; it is a process by which use of resources is regulated according to a security policy and is permitted by only authorized system entities according to that policy.

assertion An assertion is data, produced by a SAML authority, constituting a declaration of identity, or attribute information, or authorizations.

asserting party Formally, an asserting party is the administrative domain hosting SAML authorities that are issuing assertions. Informally, it is an instance of an assertion-issuing SAML authority.

attribute An attribute is a distinct characteristic of an object. An object's attributes are said to describe the object. Objects' attributes are often specified in terms of their physical traits, such as size, shape, weight, and color, etc. for real-world objects. Objects in cyberspace might have attributes describing size, type of encoding, network address, etc. Which of the object's attributes are salient is decided by the observer.

attribute assertion An attribute assertion declares that the specified subject has the specified attribute(s). Attributes may be specified by means of a URI (uniform resource indicator) or through an extension schema that defines structured attributes.

authorization assertion An authorization assertion declares that a subject has been granted specific permission to access one or more resources.

credential A credential is information that is transferred to establish a claimed principal identity.

decision assertion A decision assertion reports the result of the specified authorization request.

group A group names a collection of principals to which permissions may be granted.

permission Permission refers to a set of activities (a set of one or more operations on some set of one or more resources) that is the target of an authorization decision.

principal A principal is a system entity; its identity can be authenticated.

principal identity A principal identity is a way to represent a principal's identity, typically an identifier.

relying party A system entity that is making a decision contingent upon information or advice from another system entity is a relying party; e.g., a system entity that is relying upon various

SAML assertions about some other party(ies), made by yet other party(ies).

role A role is a set of permissions that may be granted to a principal.

security assertion A security assertion is typically scrutinized in the context of a security policy.

Security Assertion Markup Language (SAML) SAML is a specification describing a set of security assertions that are encoded using XML, the request/response protocols used to obtain the security assertions, and the bindings of these protocols to various transfer protocols (e.g., SOAP, BEEP [Blocks Extensible Exchange Protocol], HTTP, etc.).

security domain A security domain is an environment or context that is defined by security policies, security models, and security architecture, including a set of resources and set of system entities that are authorized to access the resources. An administrative domain may contain one or more security domains. The traits defining a given security domain typically evolve over time.

security policy A security policy is a set of rules and practices that specify or regulate how a system or organization provides security services to protect resources. Security policies are components of security architectures. Significant portions of security policies are implemented via security services, using security policy expressions.

security service A security service is a processing or communication service that is provided by a system to give a specific kind of protection to resources, which may reside with the system or with other systems, e.g., an authentication service or a PKI-based document attribution and authentication service. Security services include authentication, authorization, and accounting (AAA) services. Security services typically implement portions of security policies, and are implemented via security mechanisms.

**Trademark or registered trademark of Sun Microsystems, Inc. or the Object Management Group.

Cited references and notes

1. D. Plummer and D. Smith, *Web Services and Software E-Services: What's in a Name?* Application Integration and Middleware Strategies Research Note COM-12-0101, Gartner Group (October 30, 2000).
2. Web services architecture; see http://www-106.ibm.com/developerworks/webservices/library/w-ovr/.
3. W3C Recommendation, Extensible Markup Language (XML) 1.0 (Second Edition); see http://www.w3.org/TR/2000/REC-xml-20001006.html
4. SAML (Security Assertion Markup Language) is an emerging standard, from the OASIS organization, that provides the definition of XML tokens, such as name assertions that can be used to map identities between administrative domains.
5. D. Box, D. Ehnebuske, G. Kakivaya, A. Layman, N. Mendelsohn, H. F. Nielsen, S. Thatte, and D. Winer, *Simple Object Access Protocol (SOAP) 1.1*, W3C Note (May 8, 2000); available at http://www.w3.org/TR/SOAP/.
6. See http://www.w3.org/Protocols/HTTP/ietf-http-ext/.
7. E. Christensen, F. Curbera, G. Merideth, and S. Weerawarana, *Web Services Description Language (WSDL) 1.1*, W3C Note (March 15, 2001); see http://www.w3.org/TR/wsdl.html
8. See http://www.uddi.org/ and http://www.uddi.org/faqs.html#who.
9. R. Cover, *Web Services Flow Language (WSFL)*; see http://xml.coverpages.org/wsfl.html.
10. W3C XML Digital Signatures, see http://www.w3.org/Signature and http://www.w3.org/TR/2000/CR-xmldsig-core-20001031/.
11. Security Assertion Markup Language; see http://www.oasis-open.org/committees/security/docs/draft-sstc-use-strawman-03.html.
12. W3C XML Encryption Syntax and Processing, see http://www.w3c.org/Encryption/2001/03/12-proposal.html.
13. XML Digital Signature APIs, see http://www.jcp.org/jsr/detail/105.jsp.
14. XML Digital Encryption APIs, see http://www.jcp.org/jsr/detail/106.jsp.
15. It is expected that the trust model will be specified in SAML.
16. A. Brown, B. Fox, S. Hada, B. LaMacchia, and H. Maruyama, *SOAP Security Extensions: Digital Signature*, W3C Note (February 6, 2001); see http://www.w3.org/TR/SOAP-dsig/.
17. See http://www.ietf.org/rfc/rfc2828.txt.
18. XKMS is an emerging W3C specification for key management services. It can be used in combination with the keyref element in the keyinfo block of the XML digital signature to retrieve keys and certificates. See http://www.verisign.com.
19. The ebXML organization (see http://www.ebxml.org) worked with "vertical" industries, such as travel, to attempt to specify such core components, as well as issuing specifications to sign XML messages as indicated in this example.
20. The IETF PKIX working group has defined a specification for Online Certificate Status Protocol; see http://www.ietf.org/internet-drafts/draft-ietf-pkix-ocspv2-02.txt.
21. F. Leymann, Web Services Flow Language guide, available at http://www-4.ibm.com/software/solutions/webservices/pdf/WSFL.pdf.

Accepted for publication December 7, 2001.

Maryann Hondo *IBM Software Group, One Charles Park, Cambridge, Massachusetts 02142 (electronic mail: mhondo@us.ibm.com).* Ms. Hondo joined Lotus Development Corporation in 1996 and is currently the Web services security standards lead in software strategy for the IBM Software Group. In this role she has chaired the ebXML security team, participated in the development of the UDDI specifications, and is a member of the Oasis SAML working group. Previous IBM roles include security architect for emerging technology, manager of the IBM/Iris Jonah team (IETF PKIX reference implementation), and security architect for Lotus e-Suite. Her background outside IBM includes working for Hewlett-Packard Co., Digital Corporation, and Bell Labs.

Nataraj Nagaratnam *IBM Application & Integration Middleware Division, 3039 Cornwallis Road, Raleigh, North Carolina 27709-2195 (electronic mail: natarajn@us.ibm.com).* Dr. Nagaratnam is the lead security architect for the IBM WebSphere® product. He received his Ph.D. degree from Syracuse University; his thesis addresses secure delegation in distributed object environments. He has authored and edited books on Java networking and JavaBeans™ and has published his research work in numerous journals and conference proceedings.

Anthony Nadalin *IBM Software Group, 9442 Capitol of Texas Highway North, Austin, Texas 78759 (electronic mail:*

drsecure@us.ibm.com). Mr. Nadalin is the lead architect for the IBM Java security project. As senior architect, he is responsible for infrastructure design and development across IBM. He serves as IBM's primary security liaison to Sun Microsystems' JavaSoft division for Java security design and development collaboration.