

Web Services, Authentication, and Identity Management

BY WILLIAM BATHURST
AND GANESH KIRTI

Implementing identity based

Web services security in a service-oriented architecture (SOA) can be a challenge.

There many considerations such as message authentication, authorization, integrity and confidentiality. Web services are applications, but identity is part of the underlying infrastructure. How can identity policy in the form of authentication and authorization be integrated into Web services so the application developer isn't burdened with its details? This article examines mechanisms used to provide this capability.

A Service Oriented Architecture (SOA) can be thought of as a collection of services or components. In the not too-distant past, a SOA could be based on CORBA or DCOM. Today, modern SOAs are built on Web services. When a company begins to implement a SOA, or adopt Web services



for the first time, these key security questions come to mind:

- How will users be authenticated against Web services? Is Web services authentication similar to the authentication process for Web based applications?
- How does authorization work with Web services?
- How will authenticated users be authorized to perform certain activities, and not others?
- How do we integrate Web services security into an existing Identity Management infrastructure?

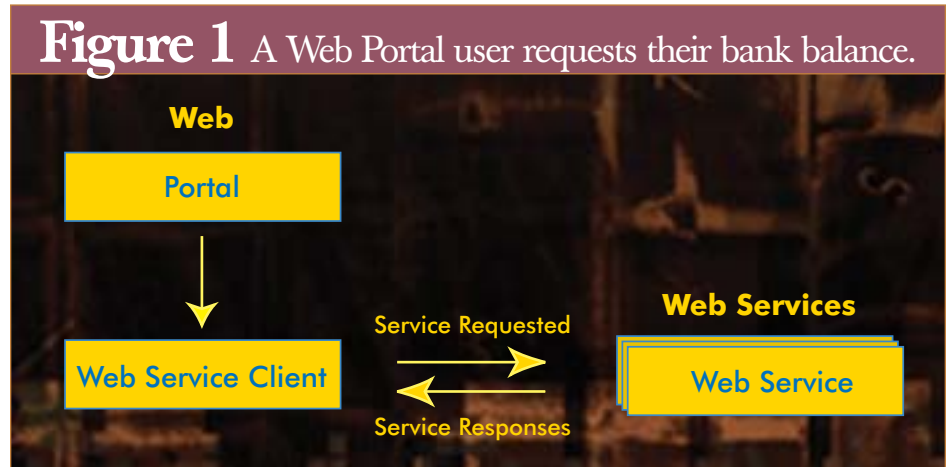
To find the answers, we must first address how Web services work, and then proceed to Web services authentication and authorization. With that understanding, we can then take a look at how Web services would integrate into an existing identity management infrastructure.

Web Services

Web services are essentially decoupled applications. In the simplest use case, a Web service client calls a Web service. For example, consider a banking Web service, which returns an account balance based on account identification. An application would call a Web service client, which in turn invokes the bank Web service.

That application might be part of a presentation layer, such as a bank portal (see Figure 1.) The bank portal might also access a number of Web services besides the account balance Web service. In essence, we have decoupled components that can be accessed by our applications. We may thus think of the application tier as being split into two separate logical domains, an application domain and a Web services domain.

The bank could have other applications that require access to this Web service besides the portal. Examples might be a loan processing application for bank employees, or an administrative application for maintaining customer accounts.



Each application will have its associated user account community. Portal, for example, users would be all the bank customers with checking or savings accounts.

Both Web services and Web applications can require authentication, and each typically authenticates against a common user store. Web-based application users and Web service clients, however, are authenticated by different means.

Authentication

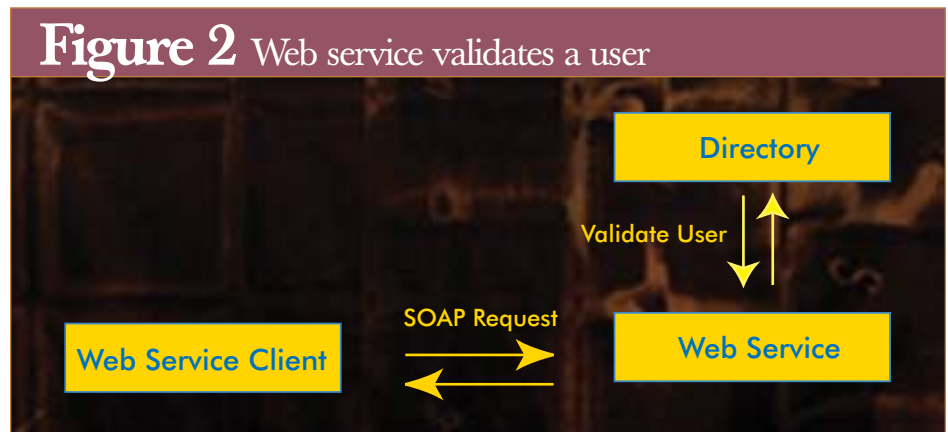
When a user logs in to a portal, or the loan, or administration application, that login is first authenticated. This authentication is usually verified against a directory, where the company keeps track of all of its users. Once authenticated, users can then access their banking functions, such as retrieving a bank balance, or a checking account history.

Each Web service may also require authentication. In that case, the Web service client sends credentials to the Web service as part of its SOAP (Simple Object Access Protocol) request. Depending on the Web service security requirements, this can be done via HTTP Security, or WS-Security.

HTTP Security

HTTP Security provides a basic level of application security by supporting user authentication. A Web service client authenticates against the Web Service by using Basic, Digest, or certificate-based schemes. Basic authentication, the simplest to implement, is the most common and is used in this example. The user's credentials are sent over the network to the Web service, which validates the username-password pair against the directory (see Figure 2:)

If the Web service were deployed in a J2EE architecture, then a JAAS (Java



Authentication and Authorization Service) login module could be used in the Web service to authenticate the user against the directory. This logic would be added to the Web service code. Microsoft also provides a similar login mechanism in its .NET framework.

HTTP Security requires an extra layer of security, since the password is either passed in the clear or weakly encrypted. There are other problems too, such as “man in the middle attacks” where hackers sniff the contents of the message payload during network transmission. The standard solution has been to use SSL (Secure Socket Layer) to encrypt all communications between the Web service client and the Web service.

WS-Security

The OASIS WS-Security specification is an open standard for Web services security. Its goal is to let applications secure SOAP message exchanges by providing encryption, integrity, and authentication support. WS-Security offers a general-purpose mechanism for associating security tokens with message content. The specification defines three approved token types:

- UsernameToken Profile
- X.509 Certificate Token Profile
- SAML (Security Assertion Markup Language) Token Profile

Each of these profiles defines how to use its token type within the WS-Security specification. For example, the UsernameToken Profile describes how a Web service client can supply a UsernameToken as a way to identify

Figure 3 Sample WS-Security Username Token

```
<wsse:UsernameToken wsu:Id="Example">
  <wsse:Username> ... </wsse:Username>
  <wsse:Password Type="..."> ... </wsse:Password>
  <wsse:Nonce EncodingType="..."> ... </wsse:Nonce>
  <wsu:Created> ... </wsu:Created>
</wsse:UsernameToken>
```

the requestor by a username and optionally by supplying a password. The XML snippet in Figure 3 shows a sample WS-Security UsernameToken:

To authenticate using WS-Security, you’d need to add a SOAP header containing the WS-Security token information to the SOAP envelope, as shown in Figure 4:

In general, a Web Service client doesn’t directly manipulate the SOAP envelope to add authentication details. Instead, the client has a security interceptor that intercepts the SOAP envelope, and adds the WS-Security authentication details.

A security interceptor could be an XML firewall, a WS-Security JAX-RPC Handler, or a similar agent. The idea behind security interceptors is to make security policy transparent to Web service clients.

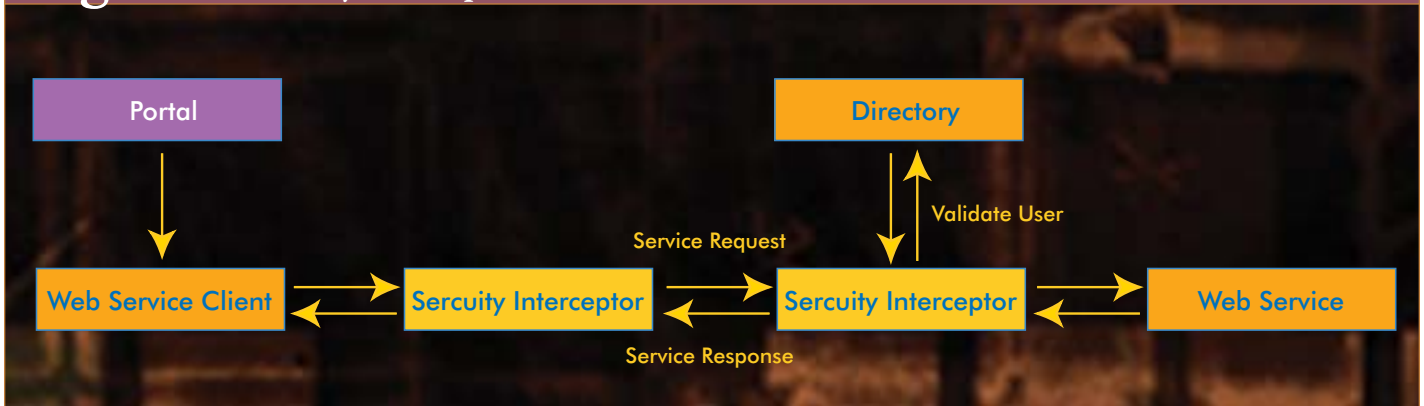
The SOAP Web service acts in a similar fashion. Here a security interceptor intercepts the incoming SOAP envelope before it arrives at the Web service, processes the WS-Security element, and validates the username and password. As with HTTP Security, a JAAS login module can be used for validation.

Figure 5 illustrates the interceptor concept with a Web service client invoking

Figure 4 SOAP Header containing WS-Security Token

```
<?xml version="1.0" encoding="utf-8"?>
<S11:Envelope xmlns:S11="..." xmlns:wsse="..." xmlns:wsu="..." xmlns:ds="...">
  <S11:Header>
    <wsse:Security xmlns:wsse="...">
      <wsse:UsernameToken wsu:Id="Example">
        <wsse:Username> ... </wsse:Username>
        <wsse:Password Type="..."> ... </wsse:Password>
        <wsse:Nonce EncodingType="..."> ... </wsse:Nonce>
        <wsu:Created> ... </wsu:Created>
      </wsse:UsernameToken>
    </wsse:Security>
  </S11:Header>
  <S11:Body wsu:Id="MsgBody">
    <tru:getBalance xmlns:tru="http://samplebank.com/ws">
      65712356
    </tru:getBalance>
  </S11:Body>
</S11:Envelope>
```

Figure 5 Security Interceptors



a Web Service. A security interceptor catches the outbound SOAP envelope and adds authentication details to the SOAP header. The interceptor may get those details from the Portal from a callback handler, or from JAX-RPC properties. The Web service also has a security interceptor that catches the inbound SOAP envelope, and verifies the username-password pair in the UsernameToken. If they are valid, access is allowed to the Web service.

If an X.509 Certificate Token were used instead of the Username Token, the validation process would be similar – the security interceptor sitting in front of the Web service would validate the requestor’s digital certificate. If the certificate were found to be valid, then the

user would be allowed access to the Web service.

There are several advantages to using WS-Security over HTTP Security:

WS-Security is more flexible – it allows multiple ways to authenticate users against a Web Service.

WS-Security is more secure – it allows for messages to be digitally signed to prevent tampering, or selectively encrypted to provide confidentiality.

By using WS-Security, security can be enforced at different places, such as a JAX-RPC handler, proxy agent, or an XML firewall. It doesn’t have to be handled in your Web service client or

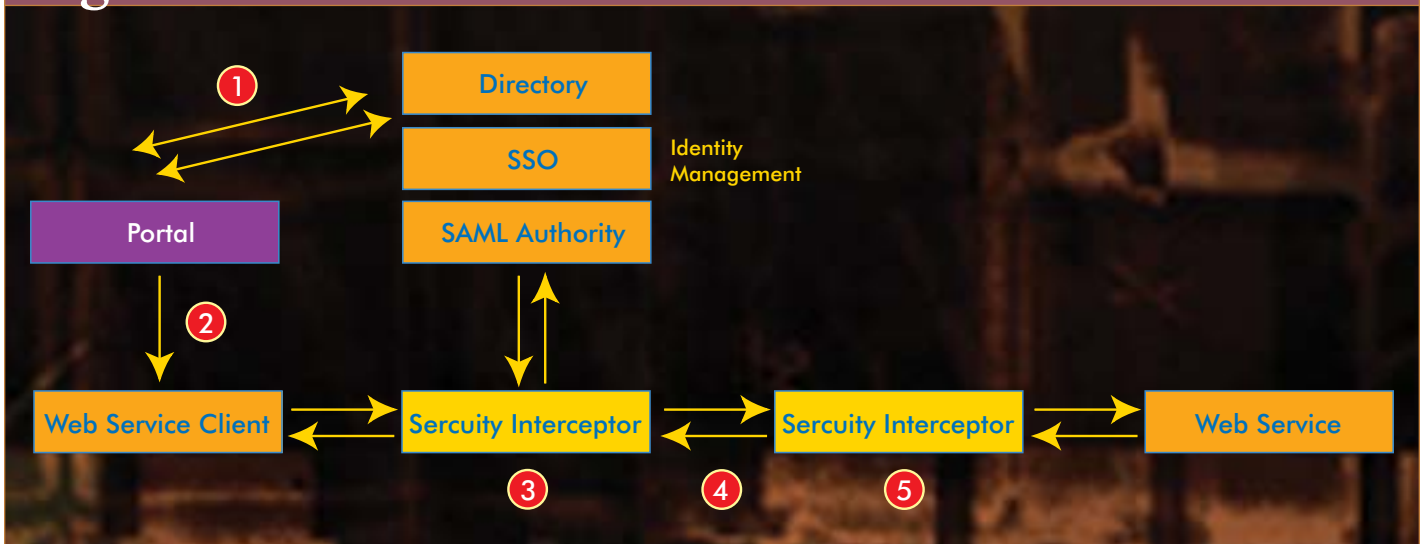
Web service application code. This removes complexity, while adding flexibility in your SOA design.

Web Services and Single Sign-On

There is another means of WS-Security authentication: the SAML (Security Assertion Markup Language) Token Profile. SAML is an XML-based security framework for exchanging authentication and authorization information. It’s the enabling glue for distributed and single sign-on architectures.

In our banking example, SAML could be used to provide single sign-on capabilities between the bank’s Web applications and its Web services.

Figure 6 SAML Token Authentication





Let's return to our portal application scenario to illustrate using a SAML Token for authentication. In this case, a user has logged into the Portal and is trying to get his bank balance. The Portal calls the Web service client to get this data from the Web service, as in Figure 6:

Let's examine what happens in this example:

1. When the user logs into the portal, their credentials are checked. If they're valid, the user is allowed to access the portal.
2. The user tries to access their bank account balance. The portal calls the Web service client on the user's behalf.
3. A Security Interceptor intercepts the outgoing message to add a WS-Security SAML authentication token (SAML assertion) to the message header. This token asserts that the user has already authenticated, and further logins are not required. The Interceptor will request a token from a centralized SAML authority on behalf of the portal user. It then adds that SAML authentication token to the WS-Security element in the SOAP envelope.
4. The SOAP message is then sent to Web the service.
5. The receiving Security Interceptor verifies the SAML token. If it's valid, the interceptor allows access to the Web service.

The idea behind this process is that once a user has been authenticated, he shouldn't have to authenticate again. The beauty of SAML is that it can combine the power of Web application and Web service based single sign-on in an interoperable, standard way.

Authorization

Web services authorization is the process of granting or denying access to a Web service resource, such as a Web service method. This authorization is typically based on a two-step process: First, the user is authenticated, ensuring that the

user is who he claims to be. Secondly, the user is authorized, and thus allowed access to resources based on his identity. An example of authorization could be a bank Web service that has three methods: getBalance, creditBalance, and debitBalance. In this example both a bank employee and a bank customer may be able to access the Web service, but the bank customer is only authorized to access the getBalance method. A bank employee may be able to access all three of these methods.

In Web services, the security interceptor will enforce authorization at the method level before allowing the incoming SOAP request to access the Web service method. Authorization decisions are based on the user's identity, which is located in the security token. The identity is then typically passed into the Web service, which may directly use standards based programmatic security APIs such as JAAS.

Integrating with Identity Management

Most companies with large application user communities have an Identity Management infrastructure (IDM) that provides all the security needs for Web application and Web services. A typical IDM already has all the requirements necessary to support Web services authentication and authorization, including:

- Single Sign-on
- Centralized LDAP directory of users
- SAML authority
- Security Interceptors
- Certificate Authority
- Delegated Administration Services (DAS)

Both J2EE and .NET let you authenticate against an existing IDM. The case of SAML is a bit more complex, as it requires a SAML authority to issue

authentication assertions. Most IDM vendors are moving towards supplying this functionality.

Conclusion

In this article we've looked at how authentication and authorization work behind the scenes for Web services. We've shown how a user's identity can be pushed from the front-end application tiers to the back-end Web service tier. This gives the appearance of coupled security, although the applications are decoupled. WS-Security allows companies to centralize their security, while still being able to support a distributed application environment. ■

William Bathurst is a senior product manager at Oracle Corporation with 18 years of industry experience. He currently leads product management for the J2EE platform security and Web services security.

Ganesh Kirti is a software development manager at Oracle Corporation with 11 years of industry experience. He currently leads development of Oracle's Java platform security including Web services security and J2EE security.

Resources

Sun's JAAS homepage:
<http://java.sun.com/products/jaas>

Sun's Java security homepage:
<http://java.sun.com/security/>

WS-Security Specification
http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=wss

OASIS WS-Security Specification
www.oasis-open.org/committees/tc_home.php?wg_abbrev=wss

OASIS SAML page
www.oasis-open.org/committees/tc_home.php?wg_abbrev=security

HTTP Basic and Digest Authentication
www.ietf.org/rfc/rfc2617.txt