Building Trust in Web Services with Security Token Service

Jiandong Guo
# Table of Contents

1 Introduction............................................................................................................. 3  
   1.1 WSIT in Brief  
   1.2 Security in WSIT  
   1.3 WS-Trust and Security Token Service  
2 Security Token Service........................................................................................ 6  
   2.1 Overview  
   2.2 STS as a Web Service  
3 Securing Web Services with Security Token Service......................................... 9  
   3.1 Service Providers  
   3.2 Clients  
4 A Typical Example ................................................................................................ 12  
5 WSIT Security Profiles and NetBeans................................................................. 13  
   5.1 Securing Web Services with Issued Tokens  
   5.2 Securing STS  
6 References............................................................................................................. 14
1 Introduction

This paper introduces the features of WS-Trust as supported in WSIT. WSIT is part of Project Metro, the open source Web services stack from Sun. This paper describes how to use WSIT to build an Security Token Service (STS) as a trust authority and how to use STS to secure Web services.

This paper is intended for an audience familiar with Web services and has a basic understanding of WSIT. Some familiarity with Web services security is helpful but not required.

1.1 WSIT in Brief

Web Services Interoperability Technology (WSIT) is an implementation of open Web services technologies that enables interoperability between Java EE and .NET 3.0. Built on top of Java API for XML Web Services (JAX-WS), WSIT addresses key aspects of Web services interoperability such as reliable messaging, atomic transactions, and security.

1.2 Security in WSIT

In WSIT, we support the following Web service security specifications:

➢ **WS-Security (versions 1.0 and 1.1)**

WS-Security provides basic framework for message level security in Web services. Other than the core spec, we also support the username/password, X509 certificate, SAML token profiles.

➢ **WS-SecurityPolicy**

WS-SecurityPolicy enables Web services to specify their security requirements to potential clients in an interoperable manner.

➢ **WS-SecureConversation**

WS-SecureConversation introduces secure sessions on top of WS-Security. It enhances the overall security through key derivations and improves performance by avoiding repeated key exchanges in multi-message exchange scenarios.

➢ **WS-Trust**

WS-Trust defines a protocol framework for security token exchanges. It introduces a model for brokering trust relationships in Web services.
This standards-based approach to security enables interoperability between different platforms. WSIT is tested to ensure interoperability with Microsoft Windows Communications Foundation (WCF) in .NET 3.0. WSIT in general has not been tested for interoperability with other platforms.

1.3 WS-Trust and Security Token Service

WS-Trust defines mechanisms for delegating authentication, authorization and user identity mapping/management to an authority called SecurityTokenService (STS) for a requestor to access a Web Service.

The mechanism defined in WS-Security allows a Web service to require a requestor to provide a set of claims (name, role, authorization code, etc) by associating security tokens with the the messages. The service doesn’t trust the requestor directly, if the requestor cannot provide the necessary security token with the required claims (e.g. authorization code) or the requestor cannot prove the authorized use of the security token. In this case, it can go to an STS as a third party authority to require a security token with required claims. This STS must have trust relationships with both the requestor and the service and brokers trust between them.

The following figure illustrates the generic architecture for a service to use an STS for brokering trust:

The model consists of a Web service client (WSC), a Web service provider (WSP) and a security token service (STS) that can issue tokens. The STS has trust relationships with the client and the service. The client may not have trust relationship with the service and can not provide the token with required claims to access the service. The sequence of actions is as follows:

1. The client sends a MEX request to the service to obtain the service WSDL with SecurityPolicy.
2. If an issued token from an STS is required for the client to access the service, as specified in the service SecurityPolicy, the client sends MEX request to the STS to obtain WSDL of the STS with the SecurityPolicy.

3. The client sends a RequestSecurityToken (RST) to the STS and receives a RequestSecurityTokenResponse (RSTR). The RSTR contains an issued token and the corresponding proof key. The messages between the client and the STS are secured according to the STS security policy.

4. The client sends a request to the service and gets a response. The request contains the issued token from the STS and is secured with the issued proof key. The service authenticates the client and process the request. The response is secured with the issued token and is returned to the client.

Multiple STS can be used in a trust chain across security domains as illustrated by the figure below:

![Diagram](image)

In this case, the service domain STSn has no direct trust relationship with the client or even the STS1 in the client domain. So it may need a chain of STS to broker trust between the client and the service where the client only has trust relationship with STS1, the service only has trust relationship with STSn and each STS only trust the neighboring STS.
2 Security Token Service

2.1 Overview

WSIT provides a general framework for building STS servers. An WSIT-based STS can be an Identity Provider, an Authentication/Authorization Provider or an Attribute Provider.

The following figure illustrates the high level architecture and the major components of the framework in WSIT for building a STS:

2.1.1 Secure Communication

An STS is a web service itself. Communication between the requestor and the STS is secured as a regular web service. The security of the WS-Trust request and response messages is handled by the SecurityServerPipe built on top of XWSS implementing WS-Security and WS-SecurityPolicy specifications.

2.1.2 STS

BaseSTS is the basic class for creating a Web Service as an STS. It delegates to the corresponding WSTrustContract implementation classes, based on the request type and configurations, for processing the request messages, i.e. RequestSecurityToken (RST) and creating the response messages, RequestSecurityTokenResponse (RSTR).

2.1.3 Security Token Issuance

WSTrustContract is the main interface for processing the WS-Trust
request message RequestSecurityToken and creating the response message with issued security token and proof key. It may be implemented to support multiple protocols (issue, renew, validate, cancel, etc). Concrete implementation classes are provided in WSIT1.0 for issuing SAML assertions of versions 1.0, 1.1, 2.0, with symmetric and asymmetric proof keys.

2.1.4 Authorization
The authorization part allows for an authorization service to be integrated to an STS. The authorization service is used to determine if the requestor can access the target service based on the requestor authentication context established in the authentication process and the appliesTo element, which points to the endpoint of the target service, specified in the RequestSecurityToken message.

2.1.5 User Identity/Attribute Management
This is used to provide the identity and attributes of the requestor to be included in the issued token for the target service. The requestor may have different identities at different services within different trust domains. Additional user attributes may also be required for the target service for authorization purposes. The interface STSAttributeProvider allows for Identity/Attribute services to be integrated into the token issuance for building an STS.

2.1.6 Key Management
The certificates for the target services are required by the token issuance component for encrypting the proof keys and optionally the issued tokens. The issued token must be signed using the private key of the STS.

2.2 STS as a Web Service
An STS is a separate web service. To build a STS, one need to construct a WSDL for the service. This WSDL should contain the configuration for the STS and the security policy assertion for securing communication between the requestor and the STS.

2.2.1 Security Policy
Standard WS-SecurityPolicy assertion is used for specifying the security requirement for the client to communicate with the STS’s. Multiple authentication schemes are supported in WSIT for the client to authenticate to the STS. These include: user name and password; X509 certificate; SAML assertion; issued tokens from the other STS.
The final one, tokens for other STSs, enables a WSIT-based STS to be used in a chain of STSs across trust domains.

2.2.2 STS Configuration
One also needs to configure some features for the STS, as well to maintain federated metadata of trusted service providers. By default this information will be in the form of a local configuration policy assertion. This assertion should be included in the WSDL of the STS in private form so that it will not be exposed externally. (This private assertion is used to configure the STS service but is not needed by clients of an STS.) This information can also be handled at run time through the configuration SPI.

The following example illustrates the general form of this configuration policy assertion:

```xml
<tc:STSConfiguration xmlns:tc="http://schemas.sun.com/ws/2006/05/trust/server"
                       encryptIssuedKey="true" encryptIssuedToken="false">
  <tc:LifeTime>36000</tc:LifeTime>
  <tc:Contract>
    com.sun.xml.ws.security.trust.impl.IssueSamlTokenContractImp
  </tc:Contract>
  <tc:Issuer>SunSTS</tc:Issuer>
  <tc:ServiceProviders>
    <tc:ServiceProvider endPoint="http://localhost:8080/jaxws-fs/simple">
      <tc:CertAlias>bob</tc:CertAlias>
      <tc:TokenType>
        http://docs.oasis-open.org/wss/oasis-wss-saml-token-profile1.1#SAMLV1.1
      </tc:TokenType>
    </tc:ServiceProvider>
    <!-- more service providers -->
  </tc:ServiceProviders>
</tc:STSConfiguration>
```

2.2.2.1 encryptIssuedKey and encryptIssuedToken attributes (line 2)
To specify if the issued proof key or the issued token is encrypte.

2.2.2.2 LifeTime (line 3)
To specify the lifetime of the issued token.

2.2.2.3 Contract (lines 4 to 6)
The implementing class of the WSTrustContract for use with an STS implementation.

2.2.2.4 Issuer (line 7)
An unique identifier of this STS.

2.2.2.5 ServiceProviders (lines 8 to 16)
All the services who have trust relationships with this STS.
2.2.6 ServiceProvider (lines 9 to 14)

To configure a service provider, one must specify the endpoint of the service (line 9). This will be used with the value of AppliesTo element in the RequestSecurityToken message from the client. The CertAlias of the service certificate in the keystore must also be specified (line 10). The service certificate is required for encrypting the issued token and the issued proof key in the issued token. One may also specify the token type (lines 11 to 13) and the key type to be issued for the client to use with the service. The token type, key type and the key size can also be specified in the RequestSecurityTokenTemplate element in the IssuedToken assertion in the SecurityPolicy for the target service. This information, contained in the RequestSecurityTokenTemplate, are copied into the RequestSecurityToken message on the client side and passed to the STS.

2.2.3 WSDL for the STS

To build a WSIT-based STS one needs to have a WSDL that contains the SecurityPolicy assertion, the STS configuration and the local security configuration policy.

The NetBeans WSIT plug-in can be used to build STS with some easy configurations for some common security profiles. See section 5 for more details.

2.2.4 STS

To develop a STS as a Web service, one needs to extend the com.sun.xml.ws.security.trust.sts.BaseSTSImpl to include the WSDL location. Then the standard JAX-WS tools can be used to deploy an STS as a Web service. As in 2.2.3, this process can be automated with NetBeans WSIT plug-in.

3 Securing Web Services with Security Token Service

Using an issued security token from a STS for authentication and security on the service and client sides follows the general framework of WS-Security and WS-SecurityPolicy.

3.1 Service Providers

3.1.1 Issued Token Requirement
A Service Provider uses the standard policy assertion IssuedToken defined in WS-SecurityPolicy to specify its requirement for an issued token from an STS.

The following example illustrates this assertion:

```xml
<sp:IssuedToken
    sp:IncludeToken="http://schemas.xmlsoap.org/ws/2005/07/securitypolicy/IncludeToken/AlwaysToRecipient">

    <Issuer xmlns="http://schemas.xmlsoap.org/ws/2005/07/securitypolicy">
        <Address xmlns="http://www.w3.org/2005/08/addressing">
            http://localhost:8080/jaxws-s5-sts/sts
        </Address>
        <Metadata xmlns="http://www.w3.org/2005/08/addressing">
                <wsx:MetadataSection>
                    <wsx:MetadataReference>
                        <Address xmlns="http://www.w3.org/2005/08/addressing">
                            http://localhost:8080/jaxws-s5-sts/sts
                        </Address>
                    </wsx:MetadataReference>
                    </wsx:MetadataSection>
                </Metadata>
            </Metadata>
        </Issuer>

    <sp:RequestSecurityTokenTemplate>
        <t:TokenType xmlns:t="http://schemas.xmlsoap.org/ws/2005/02/trust">
            http://docs.oasis-open.org/wss/oasis-wss-saml-token-profile-1.1#SAMLV1.1
        </t:TokenType>
        <t:KeyType xmlns:t="http://schemas.xmlsoap.org/ws/2005/02/trust">
            http://schemas.xmlsoap.org/ws/2005/02/trust/SymmetricKey</t:KeyType>
        <t:KeySize xmlns:t="http://schemas.xmlsoap.org/ws/2005/02/trust">256</t:KeySize>
        </sp:RequestSecurityTokenTemplate>
    </wsp:Policy>
</sp:IssuedToken>
```

3.1.2 Issuer (lines 3 to 18)
The optional Issuer assertion specifies the location of the STS including the endpoint and/or the Metadata address of the STS.

3.1.3 RequestSecurityTokenTemplate (lines 19 to 26)
This assertion is used to specify the service requirement for the type of the issued token, type of the proof key as well as the size of the proof key. It may also include required claims in the issued token. This information is passed to the STS in the RequestSecurityToken message from the client.

3.2 Clients

An embedded STS client in WSIT is called automatically by the SecurityClientPipe for the client to the target service. This STS client is secured
the same way as a client to a regular service. The figure below illustrates what happens on the client side when an STS is involved:

1. The request message to the service is intercepted by the SecurityClientPipe to add security information.
2. The SecurityClientPipe will check the security requirement of the service as specified in the SecurityPolicy. If an issued token from an STS is required, it calls TrustPlugin which will create a JAX-WS Dispatch client for the STS.
3. The RST sent by the Dispatch client is intercepted by the SecurityClientPipe for the STS client to add on security information according to the security policy of the STS.
4. The response from the STS is intercepted by the SecurityClientPipe of the STS client for processing Security information.
5. The TrustPlugin process the RSTR and sets a security context with the issued token and the proof key.
6. The request message to the service is secured with the security context established in 5.

3.2.1 Client Configuration

One may also specify the location of the STS on the client side instead of using the Issuer in the IssuedToken policy assertion on the service side. This may be used in the case of a local STS which is in the same domain as the client and for privacy reasons. The following assertion
should be in the wsit-client.xml file for the configuration of the client for the target service:

```
<tc:PreconfiguredSTS xmlns:tc="http://schemas.sun.com/ws/2006/05/trust/client"
    endpoint="http://localhost:8080/jaxws-s5-sts/sts"
    metadata="http://localhost:8080/jaxws-s5-sts/sts"/>
```

### 4 A Typical Example

This is a typical sample for brokering trust across domains.

Here the client must get an issued token from the STS in the service domain to access the service. This issued token must contain user authorization information or any other user attributes as required by the Service. In turn, the STS in the service domain cannot authenticate the client directly with user credentials like username/password. Instead it only has a trust relationship with the STS in the client domain which serves as an Identity provider in the client domain. This trust relationship is configured out of band on both sides. So the client must first go to its local STS to be authenticated with its username/password pair. Then it will be issued
a SAML token which contains user identity/attributes that can be consumed by the STS in the Service domain.

The Service should specify that an issued token from the Service side STS is required. This means that the Security Policy in the WSDL for the service must have an IssuedToken assertion which contains an Issuer element with the endpoint and the MEX endpoint of the STS in the service domain. The WSDL for STS itself also has an IssuedToken assertion but with no Issuer element. This means that it require an issued token from an STS in the client domain. Clients may come from different domains, so each client must be configured to know its associated STS. More precisely, the client configuration policy of the client for the service domain STS should contain the PreConfiguredSTS policy assertion pointing to the local STS.

Here is the flow of the actions:
1. The client obtains the service WSDL with security policy using a MEX call.
2. The client obtains the service side STS WSDL with security policy using a MEX call. The service STS endpoint and the MEX endpoint were retrieved from the service WSDL obtained in step 1.
3. The client obtains the local STS WSDL with security policy using MEX call. The local STS endpoint and the MEX endpoint are obtained locally through an PreConfiguredSTS policy assertion.
4. The client sends an RST to the local STS with username/password. The local STS authenticates the user and issues a security token with user identity for the service domain. An STSAttributeProvider provides the identity mapping service. An RSTR is returned to the client with issued token and/or the associated proof key.
5. The client sends an RST to the service STS with the issued SAML token from step 4. The service STS authenticates the user and issues an SAML assertion with authorization information for the service. An RSTR is returned to the client with the SAML assertion.
6. The client sends the service request to the Service with the issued SAML token from step 5. The service retrieves the authorization information and processes the request. A response or a fault is then returned to the client.

5 WSIT Security Profiles and NetBeans

Several security profiles are defined in WSIT for use with NetBeans to configure the security on both the service and the client sides. These profiles cover some of the most common use cases in securing Web services. You may find detailed information about this in WSIT tutorial.

5.1 Securing Web Services with Issued Tokens
Three profiles are defined for securing Web services with issued tokens from an STS.

- **STS Issued Token**
  This is the case that an issued token as well as an associated proof key from an STS are required. The issued token and proof key are used to secure the messages between the client and the service.

- **STS issued Token with Service Certificate**
  In this case, the service certificate is available to the client for securing messages. The issued token is used for authentication. Therefore no associated proof key is required.

- **STS issued Endorsing Token**
  In this case, the service certificate is also available to the client. But a proof key is required to sign the message primary signature.

### 5.2 Securing STS

All security profiles can be used with an STS. In particular we can use the issued token based profiles for securing STS to create STS chain scenarios.

Check the WSIT tutorial [chapter 6](#) and [chapter 7](#) for more information.

### 6 References

- GlassFish Community: [https://glassfish.dev.java.net/](https://glassfish.dev.java.net/)
- Project Metro: [https://metro.dev.java.net/](https://metro.dev.java.net/)
- WSIT: [https://wsit.dev.java.net/](https://wsit.dev.java.net/)
- WSIT tutorials: [https://wsit-docs.dev.java.net/](https://wsit-docs.dev.java.net/)