WS-Security policy profile of WS-PolicyConstraints

Working Draft 04, 1 December 2005

Authors:

Anne Anderson (anne.anderson@sun.com)

File name:

ws-security-profile-of-ws-policy-constraints

Abstract:

This document defines predicates for specifying constraints on the message security domain covered by the OASIS WS-Security Standard. These predicates are expressed using the generic policy constraint language WS-PolicyConstraints, which is based on the OASIS eXtensible Access Control Language (XACML) Standard. By expressing constraints using this generic constraint language, any policy processor for WS-PolicyConstraints can verify a message against a WS-Security policy, and can automatically find a mutually acceptable WS-Security policy based on the individual policies of two or more parties. No plug-ins or modifications to the policy processor for WS-PolicyConstraints are required for handling this or any other domain's policy constraints.

The profile defined here is not intended to replace WS-SecurityPolicy. It is a “proof-of-concept” of the WS-PolicyConstraints approach that takes a well-known set of Assertions and demonstrates that they can be expressed using WS-PolicyConstraints. To enable an easy comparison between the two languages, this document has been organized according to the Assertions defined in WS-SecurityPolicy v1.1 because its purpose is to explore various types of Assertions and how they can be expressed using a domain-independent policy assertion language such as WS-PolicyConstraints.

Status:

This version of the specification is a working draft.

Table of Contents

1 Introduction *(non-normative)*

The policy framework currently expressed by *WS-Policy* [WSP] requires the definition of policy "`Assertions`" (predicates) for each domain to which policy is to be applied. Three examples of specifications defining such Assertions have been published to date:

- *WS-PolicyAssertions* [WSPA], defining some general-purpose Assertions,
- *WS-SecurityPolicy* [WSSP], defining policy Assertions for WS-Security [WSS] and other specifications that might cover the same message security space, and
- *WS-ReliabilityPolicy* [WSRP], defining policy Assertions for WS-Reliable Messaging [WSR] and other specifications that might cover the same reliable messaging space.

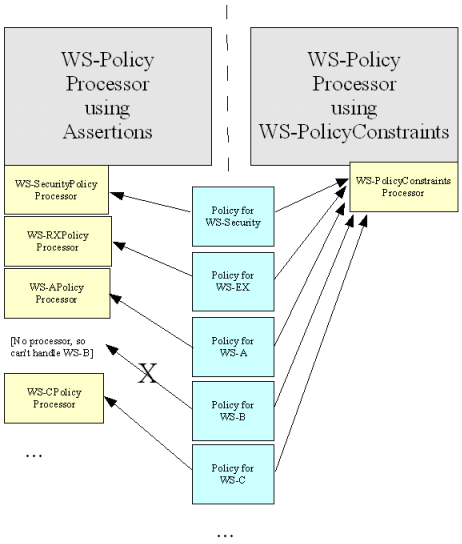
Each of these sets of Assertions is specific to its domain – they do not share syntax or semantics. In order to support each such Assertion, each policy processor must be supplied with an Assertion-specific code module that implements the semantics described in the specification. Such a module must be developed for each platform that is to support the Assertion. Since there is no standard language for the Assertions, the module must be extensively tested for interoperability, as different developers may interpret the specification in different ways. Finally, the module must be deployed on each server that is to support the Assertion. If the Assertion is modified, to support a new secure hash algorithm, for example, this process must be repeated.

As policies are used with more and more domains, the number of domain-specific Assertion modules that must be supported in each policy processor will increase, along with the possibility of interpretation errors, version mismatches, and missing modules. If a customer defines a new type of policy for a new application, the customer must arrange to have modules added to every policy processor for handling the Assertions used the new policy type. It is important to understand that the *WS-Policy* Assertion model requires that each policy processor be configured with code to recognize and implement each Assertion in each domain with which that policy processor will be used.

While the authors of *WS-Policy* suggest that, in the future, policy Assertions should be defined as part of the specification to which these policy Assertions apply, this serves only to reduce the total number of specifications. The implementation of each specification must still include a new module that can handle the Assertions defined in the specification.

An alternative to the Assertions model is specified in *WS-PolicyConstraints* [WSPC]. In this model, a generic language for specifying policy predicates, or "constraints", is defined. This generic language is based on the *OASIS eXtensible Access Control Markup Language* [XACML] functions, as used in the *XACML Profile for web-services* [WSPL]. Expressions in this standard language can then be used to express policy predicates for any domain. Any policy processor that supports the generic language can understand, match, and verify any policy written in the generic language, removing the need to have new code modules for each new type of policy Assertion.

The following diagram illustrates this difference.



In addition to enabling the use of generic policy processors, the *WS-PolicyConstraints* language provides another benefit to web services policy, derived from XACML: many policy predicates can be evaluated directly against a message to confirm that the message conforms to the policy. This is because XACML function arguments can consist of XPath expressions to be evaluated against actual messages, which could be SOAP or other types of messages. For example, assume a policy writer wants to constrain the acceptable values for a username in the "`//S11:Envelope/S11:Header/wsse:Security/wsse:UsernameToken/wsse:UserName`" element of a SOAP message. The policy writer wants to limit the acceptable values to be names that start with the string "Zoe". Using *WS-PolicyConstraints*, the policy writer can specify that the value obtained by evaluating the XPath expression "`/S11:Envelope/S11:Header/wsse:Security/wsse:UsernameToken/wsse:Username/text()`" against actual SOAP messages must match the regular expression string "Zoe.*" using the standard XACML "string-regexp-match" function. In order to constrain the acceptable values for some other element of the SOAP message, the policy writer can use the same function with different XPath expression and regular expression arguments. Using *WS-SecurityPolicy* however, the policy writer must express this policy using an instance of the "`SecurityToken`" element defined in *WS-SecurityPolicy.* The policy writer must correctly use the "`SecurityToken`" element, its "`Claims`" element, its "`SubjectName`" element, its "`MatchType`" XML attribute, and the values for the "`MatchType`" attribute, as defined in the *WS-SecurityPolicy* specification. Likewise, the policy processor must contain a domain-specific module for *WS-SecurityPolicy* that recognizes and correctly interprets all parts of a "`SecurityToken`" element. This module must know that the value contained in the *WS-SecurityPolicy* "`SubjectName`" element is a regular expression that must be matched against the value contained in the "`//S11:Envelope/S11:Header/wsse:Security/wsse:UsernameToken/wsse:Username`" element, even though there is no direct reference to this element in the policy: it is specified only in the text of the *WS-SecurityPolicy* specification. The implementation of all this is specific to the "`Username`" element: the *WS-PolicyConstraints* "`Username`" Assertion can't be used to make regular expression matches against other elements of a message.

By using predicates that can refer to message elements directly, *WS-PolicyConstraints* greatly reduces the number of new elements that must be defined to express policy information. With the current *WS-Policy* Assertions model, however, a new element must be defined for each component of a message for which policy is to be specified.

This document illustrates how the alternative *WS-PolicyConstraints* model could be used to express the Assertions defined in *WS-SecurityPolicy*. It is intended to serve as a proof-of-concept for *WS-PolicyConstraints*, as well as providing examples for the use of *WS-PolicyConstraints* that may be of help to policy developers for other domains. This profile, however, is NOT an attempt to replace *WS-SecurityPolicy*. Certain domain-specific Assertions, such as those in *WS-SecurityPolicy,* have gained acceptance in the industry, and will need to be supported in policy processors. It may also be the case

that some types of new policy Assertions can not be expressed using *WS-PolicyConstraints.* The expectation is that a code module to support policies written using *WS-PolicyConstraints* will exist in parallel with code modules to support some set of domain-specific Assertions, such as those in *WS-SecurityPolicy.* With careful design of the interface between the policy framework layer and the policy Assertions layer, co-existence need not be a problem.

It is important to recognize that *WS-PolicyConstraints* must be used within a policy framework that defines Boolean combinations of Assertions or constraints on individual policy items. This framework could be *WS-Policy or* any other policy framework that addresses the same level of concerns addressed by *WS-Policy,* that is, any framework that defines how to express Boolean combinations of constraints to compose a policy.

1.1 Notation

The following XML Internal Entities are used to make the examples more compact and easier to read:

```
<!ENTITY xsd   "http://www.w3.org/2001/XMLSchema#"/>
<!ENTITY xfunc "urn:oasis:names:tc:xacml:1.0:function:"/>
<!ENTITY xdata "urn:oasis:names:tc:xacml:1.0:data-type:"/>
<!ENTITY x509  "...the URI of the Web Services Security X.509
Certificate Token Profile..."/>
```

The following namespace identifiers are used:

```
wsse http://schemas.xmlsoap.org/ws/2002/12/secext
ds   http://www.w3.org/2000/09/xmldsig#
xenc http://www.w3.org/2001/04/xmlenc#
wsu  http://schemas.xmlsoap.org/ws/2002/07/utility
wsp  http://schemas.xmlsoap.org/ws/2002/12/policy
xsd  http://www.w3.org/2001/XMLSchema
wspc ...a new URI to be defined for WS-PolicyConstraints...
sp   ...a new namespace for security policy elements
```

2 WS-Security

This section serves as a brief introduction to *WS-Security*.

WS-Security is a set of SOAP [SOAP] extensions that "provides three main mechanisms: ability to send security tokens as part of a message, message integrity, and message confidentiality... These mechanisms can be used independently (e.g. to pass a security token) or in a tightly coupled manner..." [WSS Lines 125-132]. The extensions are added to the SOAP envelope header as part of a new `<wsse:Security>` element.

The contents of this `<wsse:Security>` element can include some one or more of the following element types, each of which might occur more than once:

1. A generic ID and reference mechanism: this can be used with other types defined either in *WS-Security* or in other specifications to associate identifiers with elements, and to then reference those identified elements from elsewhere in the `<wsse:Security>` header; a generic ID may also be used in the `<Body>` of the SOAP message envelope,
2. `<wsse:UsernameToken>`: contains a username security token defined in the *WS-Security* specification, and extended in the *Web Services Security Username Token Profile 1.0*,
3. `<wsse:BinarySecurityToken>`: contains a binary security token defined in the *WS-Security* specification. There are specific subtypes defined for various X509 token types in the *Web Services Security X.509 Certificate Token Profile.* There are specific subtypes defined for Kerberos token types in the *Web Services Security: Kerberos Token Profile (draft)*.
4. `<ds:SignedInfo>`: contains a signature conforming to the *XML Digital Signature* specification [XDS],
5. `<xenc:ReferenceList>`: contains a manifest conforming to the *XML Encryption* specification [XENC],
6. `<xenc:EncryptedKey>`: contains an encrypted key conforming to the *XML Encryption* specification,
7. `<wsu:Timestamp>`: contains a time stamp defined in the *WS-Security* specification,
8. `<wsse:SecurityTokenReference>`: contains a reference to a security token, either using the generic ID and reference mechanism, or a `<wsse:SecurityTokenReference>` element defined in the *WS-Security* specification. A `<wsse:SecurityTokenReference>` element may contain a <wsse:KeyIdentifier> element containing a key identifier type that is defined in *WS-Security*.

Each of these elements defines some XML attributes and sub-elements, but is also extensible. New extension elements, such as new token types, etc. may also be added. Several token types are defined in *WS-Security* profiles: *UsernameToken Profile* and *X.509 Token Profile*.

3 Policies about WS-Security

Before starting, it is important to understand the target of a "*WS-Security* policy". In some cases, policy writers want to constrain the content of instances of the "Security" headers defined by *WS-Security.* In other cases, policy writers may want to specify how these "Security" headers are created and processed. As an example of the first type of policy, the policy may specify acceptable values for the "Password" element in a *WS-Security* "UserNameToken". This type of policy constraint can be checked against the actual content of a message containing such a "Password" element. As an example of the second type of policy, the policy may specify that the type of password used in the "Password" element in a "UserNameToken" should be a password digest rather than a plain-text password. This type of policy constraint can't be checked against the content of the message, since there is no element or attribute in the "UserNameToken" that specifies the type of the password. The value of the "Password" element in either case is a string. If the "Password" is incorrect, it is impossible in general to know whether it is because the password value was incorrect or because it was supplied as a plain-text password rather than as a password digest. If they are to communicate successfully, the producer of a "Security" header containing a "Password" element must reach agreement with the consumers of that element about which type of password will be used, but that agreement does not appear explicitly in the "Security" header itself. Another example of policy information that does not appear in the message is a directive that new "Security" headers should be pre-pended to existing ones. Producing messages in such a way aids processing by message consumers, since "Security" headers often need to be verified in a particular order (think of verifying a signature before decrypting the message versus decrypting the message first, and then verifying the signature). But once again, the consumer of a message has no way of knowing whether the producers of the "Security" headers actually pre-pended them: if the messages fail to verify, the consumer has no way of knowing whether the headers were incorrect or whether they were not pre-pended as expected.

Where policies concern information that does not appear in the "Security" header itself, or that can not be expressed easily using direct references to the SOAP message, a new policy element or Attribute needs to be defined to express such information. Such elements or Attributes do not need to be instantiated – they may never appear in a message – but they are needed as a way of talking about how the "Security" header is to be created and consumed – they are identifiers for the associated information. For example, if the producer and the consumer both agree that *PasswordType = "PasswordDigest",* and they produce and consume the message accordingly, then there is no need for *PasswordType = "PasswordDigest"* to appear in the message itself. Nevertheless, it may be useful to convey explicitly in a message the fact that a "PasswordDigest" is being used.

Where new elements or Attributes are found to be needed to specify policies about information not currently specified in a message, it may be an indication that new elements should be added to the underlying specification in a future revision. If such new elements are defined in the underlying specification with default values, this will encourage consistent use of message contents without making messages that conform to the default any longer than currently. Alternatively, the underlying specification could mandate certain processing actions or behaviors to enable consistent usage of instances of the specification's schema.

4 WS-SecurityPolicy

WS-SecurityPolicy defines the domain-specific policy Assertions to be used with *WS-Policy* when writing policies about security information associated with a message. *WS-SecurityPolicy* is described as being generic to any underlying security specification, and not confined to use with *WS-Security.* In practice, *WS-SecurityPolicy* would have to be re-implemented for each underlying security specification if the policies are to be verified, so most implementations can be expected to support only *WS-Security.*

When used with *WS-Security, WS-SecurityPolicy* defines 7 types of *WS-Policy* `<Assertion>` elements that can be used to place constraints on the *WS-Security* `<wsse:Security>` header to be used in SOAP messages:

1. `<SpecVersion>`: indicates support for *WS-Security*, including the Addendum.
2. `<SecurityToken>`: constrains the types and contents of security tokens supplied with a message.
3. `<Integrity>`: places constraints on digital signatures used in the message.
4. `<Confidentiality>`: places constraints on the use of encryption in the message.
5. `<Visibility>`: specifies portions of a message that must be able to be processed by an intermediary or endpoint.
6. `<SecurityHeader>`: constrains aspects of the *WS-Security* `<wsse:Security>` header.
7. `<MessageAge>`: constrains the use of the `<wsse:Timestamp>` header from *WS-Security*

The `<Assertion>` elements defined in *WS-SecurityPolicy* include `wsp:Preference` and `wsp:Usage` XML attributes. Since the most recent draft of *WS-Policy* omits these attributes, these are not described below.

The schema for *WS-SecurityPolicy* [WSSP-Sch] is a collection of element definitions, many of which are freely extensible. The `Assertion` elements in general are not structured in the schema itself, but depend on using the extensibility of their parent elements. Putting the elements together in a meaningful way requires studying the *WS-SecurityPolicy* specification.

Except for `<SpecVersion>`, `<SecurityHeader>`, and `<MessageAge>` the *WS-SecurityPolicy* Assertions, are not specific to *WS-Security*, and might be used to apply to any security parameter specification mechanism. [Note: `<MessageAge>` could have been generic, but the *WS-SecurityPolicy* specification specifically ties it to the *WS-Security* `<Timestamp>` element]. The *WS-SecurityPolicy* engine used to process and enforce policies using these Assertions must include specific code modules to support the application of the Assertions to each specification mechanism, however, so the engine must be modified to support *WS-Security*. If the Assertions are used to apply to some other specification mechanism, the *WS-SecurityPolicy* engine must be modified again to support the new specification. Just being non-specific to *WS-Security* does not automatically make *WS-SecurityPolicy* work with any security parameter specification mechanism. Its semantics are still domain-specific.

5 Using WS-PolicyConstraints for WS-Security Policies

Many of the following *WS-PolicyConstraints* predicates are written directly against the *WS-Security* specification. Where new elements or attributes are needed to specifying information that does not appear in a message instance directly, the predicates are written against the element defined for this purpose in the *WS-SecurityPolicy* specification. Note that using *WS-SecurityPolicy* in this way does not mean that domain-specific policy processing modules are needed. The predicates are still expressed using the generic *WS-PolicyConstraints* language and can be handled by a generic policy processor. Using the element defined in *WS-SecurityPolicy* is only one possible approach. Using *WS-PolicyConstraints* to express most policy constraints against the message itself means that additional policy elements can usually be much simpler than those currently defined in *WS-SecurityPolicy.* In most cases, a simple Attribute could be used instead.

Where predicates are written directly against the *WS-Security* specification, the predicates can be directly enforced using an XACML Policy Decision Point engine, although the engine must be extended to support some new functions and datatypes. It is expected that the number of such extensions will be limited, since they are needed only for expressing policies about legacy data that is not in XML, such as the contents of public key certificates. Future information appears likely to be defined in XML, and can then be referenced using the existing standard XACML functions.

If security-related predicates are to be written using *WS-PolicyConstraints* against some schema other than *WS-Security,* then the predicates would need to be reformulated. Since two parties in a message exchange must agree on how they will specify their security information, however, it does not seem overly restrictive to require that the actual policies be written against the specific format especially since one of the payoffs is the ability to directly verify the policy against the messages. If more abstract policies are needed, then new abstract elements can be defined (or re-use the ones defined in *WS-SecurityPolicy)* and associated with sets of specific predicates for particular security header schemas.

5.1 Multiple constraints on a single nodeset

Frequently, a policy will require a single nodeset in a <wsse:Security> header to satisfy multiple conditions. In these cases, the *WS-PolicyConstraints "*&wspc;function:limit-scope" may be used to enclose the predicates that must be satisfied within a single nodeset. This function is defined as an extension to XACML using XACML's extensible function capabilities. For example, if a particular canonicalization method and a particular signature method must be used in a single <ds:Signature> element, the following *WS-PolicyConstraints* predicate would be used.

```
<Apply FunctionId="&wspc;function:limit-scope">
   <AttributeValue
DataType="&xsd;string">//S11:Envelope/S11:Header/wsse:Security/ds:Signa
ture/ds:SignedInfo</AttributeValue>
   <Apply FunctionId="&xfunc;anyURI-equal">
      <AttributeSelector DataType="&xsd;string" RequestContextPath=
"/ds:CanonicalizationMethod/@Algorithm".>
      <AttributeValue DataType="&xsd;anyURI">
          http://www.w3.org/2001/10/xml-exc-c14n"</AttributeValue>
   </Apply>
   <Apply FunctionId="&xfunc;anyURI-equal">
      <AttributeSelector DataType="&xsd;anyURI" RequestContextPath=
"/ds:SignatureMethod/@Algorithm".>
      <AttributeValue DataType="&xsd;anyURI">
          http://www.w3.org/2000/09/xmldsig#hmac-sha1</AttributeValue>
   </Apply>
</Apply>
```

Only individual predicates will be shown below, but it should be remembered that the

313 `&wspc;function:limit-scope` function may be used to restrict any set of predicates to a single
314 nodeset.

315 5.2 Limited XPath expressions

316 In order for policy predicates in two different policies to be compared, it is necessary to be able to tell
317 whether the two predicates refer to the same underlying policy vocabulary item. Using full XPath, there
318 are multiple ways to refer to the same element or attribute in schema instances, and it is not possible to
319 determine the same element or attribute is being referenced.

320 *WS-PolicyConstraints* needs to use a subset of XPath such that it can be determined whether any two
321 XPath expressions refer to the same nodeset or nodesets. No such subset has been defined as far as
322 we know.

323 A proposed subset is used in these examples. This subset uses only absolute XPath expressions (i.e.
324 all start with //<doc root>), does not use any numbered elements (e.g. x[1]), does not use query functions
325 in the XPath expressions (e.g. [@PasswordType="PasswordDigest"]), and references only text elements
326 or the values of XML attributes in the terminal element of the XPath expression.

327 This subset may be inadequate and is probably overly constrained, but further research is needed to
328 specify an optimal subset that retains as much expressivity as possible while preserving the ability to
329 match the potential nodesets specified by each XPath expression.

6 Actual WS-Security policy predicates

This section contains actual predicates specified using *WS-PolicyConstraints* for each predicate defined in a *WS-SecurityPolicy* Assertion.

6.1 Specification version

WS-SecurityPolicy uses the `<wsp:SpecVersion>` Assertion to indicate 'support for *WS-Security* including the Addendum". The example provided is:

```
<wsp:SpecVersion URI="http://schemas.xmlsoap.org/ws/2002/07/secext"/>
```

The semantics of this Assertion are not entirely clear: is it intended to specify that a particular version of *WS-Security* must be used, or that there must be an instance of a *WS-Security* `<wsse:Security>` header using this version of the specification?

WS-PolicyConstraints can express either or both of these policy requirements precisely as follows.

The following predicate requires that a particular version of *WS-Security* be used. This type of predicate is called a "*WS-Security* version predicate" in the subsequent discussion.

```
<Apply FunctionId="&xfunc:anyURI-is-in">
   <AttributeValue DataType="&xsd;anyURI">
       http://schemas/xmlsoap.org/ws/2002/07/secext</AttributeValue>
   <AttributeSelector
RequestContextPath="//S11:Envelope/S11:Header/wsse:Security/@xmlns"
DataType="&xsd;anyURI"/>
</Apply>
```

If more than one version of *WS-Security* is supported, then multiple instances of a "WS-Security version predicate" may be used as follows:

```
<wsp:ExactlyOne>
   <...WS-Security version predicate #1.../>
   <...WS-Security version predicate #2.../>
</wsp:ExactlyOne>
```

In general, when using *WS-PolicyConstraints,* no predicate used merely to require the presence of a `<wsse:Security>` header will be needed, since other predicates that require the header to contain particular contents will occur and can only be satisfied if the header itself is present, as in the "*WS-Security* version predicate" above. Nevertheless, if it is actually the case that some form of security header must be present, but without any constraints on its contents, the following *WS-PolicyConstraints* predicate can be used. This predicate is called "Security header present predicate" below.

```
<Apply FunctionId="&wspc;function:must-be-present">
   <AttributeValue
DataType="&xsd;string">/S11:Envelope/S11:Header/wsse:Security</Attribut
eValue>
</Apply>
```

If the `<wsse:Security>` header is optional, then "Security header present predicates" may be used as follows, where "other predicates" are other predicates that must be satisfied in addition to the "Security header present predicate":

```
<wsp:ExactlyOne>
   <wsp:All>
      ...other predicates...
      <...Security header present predicate.../>
   </wsp:All>
   <wsp:All>
      ...other predicates...
   </wsp:All>
</wsp:ExactlyOne>
```

Alternatively, the *WS-PolicyConstraints* function "`&wspc;function:must-not-be-present`" can be used. This may be especially attractive if the number of other, independent predicates is large. The policy above would then look as follows:

```
<wsp:ExactlyOne>
   <wsp:All>
      ...other predicates not depending on Security header...
      <wsp:ExactlyOne>
         <wsp:All>
            <...Security header present predicate.../>
            ...other predicates depending on Security header...
         </wsp:All>
         <Apply FunctionId="&wspc;function:must-not-be-present">
            <AttributeValue
DataType="&xsd;string">//S11:Envelope/S11:Header/wsse:Security</Attribu
teValue>
      </wsp:ExactlyOne>
   </wsp:All>
</wsp:ExactlyOne>
```

6.2 Security tokens

WS-SecurityPolicy uses the `<wsp:SecurityToken>` Assertion to "describe what security tokens are required and accepted by a Web service. It can also be used to express a Web Service's policy on security tokens that are included when the service sends out a message (e.g., as a reply message)."

WS-Security does not define any standard elements or attributes for a security token: all security tokens are defined in profiles. *WS-PolicyConstraints* examples are provided below for each token type listed in Appendix I of *WS-SecurityPolicy*. If new token types are defined, the *WS-SecurityPolicy* specification would need to be amended to support them, whereas corresponding *WS-PolicyConstraints* predicates require only knowledge of the new security token profile schema, and can be written using the standard *WS-PolicyConstraints* language.

There are a number of sub-elements included in a `<wsp:SecurityToken>` Assertion. We will first describe those common to various security token profiles, then those specific to particular profiles, each with its corresponding *WS-PolicyConstraints* predicate. Then we will show how the *WS-PolicyConstraints* "limit-scope" function can be used to require that a collection of predicates must all be true for a single security token in the `<wsse:Security>` header.

6.2.1 Common Token Profile Elements

/wssp:SecurityToken/wssp:TokenType

There are actually three token types defined in the *Web Services Security X.509 Certificate Token Profile.* The *Profile* states that these URI fragments are relative to the URI for "this specification", but that URI is not clearly stated anywhere. The "Document Location" on the title page is shown as "http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-x509-token-profile-1.0", so perhaps that is what is meant. We use the XML Internal Entity "&x509;" to refer to the correct URI.

1. `&x509;#X509v3`: a single X.509 v3 signature-verification certificate.
2. `&x509;#X509PKIPathv1`: an ordered list of X.509 certificates packaged in a PKIPath.
3. `&x509;#PKCS7`: a list of X.509 certificates, and (optionally) CRLs packaged in a PKCS#7 wrapper.

WS-PolicyConstraints allows specification of each token type precisely:

```
<Apply FunctionId="&wspc;must-be-present">
   <AttributeValue
DataType="&xsd;string">//S11:Envelope/S11:Header/wsse:Security/&x509;#x
509v3</AttributeValue>
</Apply>
```

where the desired X.509 or other security token type would be specified in the XPath expression used in the <AttributeValue>.

/wssp:SecurityToken/wssp:TokenIssuer

In an actual X.509 certificate, the Issuer will be embedded in the binary data of the certificate itself. If the value in the certificate is required, then *WS-PolicyConstraints* will require that a new function must be written to extract that value from a certificate reference. Note that any implementation of *WS-SecurityPolicy* in which this Assertion will be applied to an actual X.509 certificate will require implementation of an equivalent function internally.

If, rather than the value in an X.509 certificate, the policy refers to the value specified in an instance from the *XML Digital Signature* standard that includes the issuer name, then no new function is required. The following example shows a predicate that uses the *XML Digital Signature* element <ds:X509IssuerName>, that must be present in the <ds:KeyInfo> in the <wsse:Security> header. This example requires that the certificate issuer name be "DC=ACMECorp, DC=com".

```
<Apply FunctionId="&xfunc;x509Name-match">
   <Apply FunctionId="&xfunc;x509Name-one-and-only">
      <AttributeSelector
RequestContextPath="//S11:Envelope/S11:Header/wsse:Security/ds:KeyInfo/
wsse:SecurityTokenReference/ds:X509Data/ds:X509IssuerSerial/ds:X509Issu
erName/text()" DataType="&xdata;x509Name"/>
   </Apply>
   <AttributeValue DataType="&xdata;x509Name">DC=ACMECorp,
DC=com</AttributeValue>
</Apply>
```

Token issuer constraints can similarly be constructed for other types of security tokens.

/wssp:SecurityToken/wssp:Claims/wssp:SubjectName

As with the previous "TokenIssuer" Assertion element, a "SubjectName" Assertion might refer to an actual X.509 certificate included in a message, or to a description of such a certificate specified using *XML Digital Signature.* As before, if the policy refers to the contents of an actual X.509 certificate, then *WS-PolicyConstraints* will require that a new function be written to extract that value from a certificate reference. Any implementation of *WS-SecurityPolicy* in which this Assertion element will constrain an actual X.509 certificate instance will require implementation of an equivalent function.

The following *WS-PolicyConstraints* predicate assumes that the subject name from the contents of the X509 certificate is provided in the *XML Digital Signature* description contained in the Security header. This example requires that the certificate subject's name be "cn=Anne.Anderson, DC=Sun, DC=com".

```
<Apply FunctionId="&xfunc;x509Name-match">
   <Apply FunctionId="x509Name-one-and-only">
      <AttributeSelector
RequestContextPath="//S11:Envelope/S11:Header/wsse:Security/ds:KeyInfo/
wsse:SecurityTokenReference/ds:X509Data/ds:X509SubjectName/text()"
DataType="&xdata;x509Name"/>
   </Apply>
   <AttributeValue DataType="&xdata;x509Name">CN=Anne.Anderson, DC=Sun,
DC=com</AttributeValue>
</Apply>
```

Subject name constraints can similarly be constructed for other types of security tokens.

/wssp:SecurityToken/wssp:Claims/wssp:SubjectName/@wssp:MatchType

The preceding example required the subject name to match the supplied value exactly. *WS-SecurityPolicy* provides an XML attribute to be used for specifying either an exact match or a match where the specified value must be the prefix of the value in the certificate. In *WS-PolicyConstraints,* this information is provided via the matching function that is used. For example, if a prefix match is desired,

then the following *WS-PolicyConstraints* predicate may be used. In this example, the subject's X500 name must begin with "CN=Anne.Anderson," where "CN" may be capitalized or not.

```
<Apply FunctionId="&xfunc;x500Name-regexp-match">
   <AttributeValue DataType="&xsd;string">^[Cc][Nn] *=
*Anne.Anderson,.*</AttributeValue>
   <Apply FunctionId="x500Name-one-and-only">
      <AttributeSelector
RequestContextPath="/S11:Envelope/S11:Header/wsse:Security/ds:KeyInfo/w
sse:SecurityTokenReference/ds:X509Data/ds:X509SubjectName/text()"
DataType="&xdata;x500Name"/>
   </Apply>
</Apply>
```

6.2.2 X.509v3 Token

/wssp:SecurityToken/wssp:Claims/wssp:X509Extension

This element in a *WS-SecurityPolicy* `<SecurityToken>` specifies the value, OID, and (optionally) the criticality of a required X509Extension in the certificate. The *XML Digital Signature* specification does not contain elements for describing extensions contained in an X509 certificate other than the `SubjectKeyInfo` extension, so new functions must be written to extract this type of information from a referenced certificate in a message.

The following example uses a new XACML extension function called "`hexBinary-getCertExtensionValue`". It takes as input a string value interpreted as the OID of the desired extension, a string value indicating required criticality (with acceptable values of "`Critical`", "`NotCritical`", and "`CriticalOrNot`"), and a reference to an ASN-encoded X.509 certificate, encoded in ASN.1 with data type "`&xml;hexBinary`". It returns a bag containing the values of all extensions in the referenced certificate that match the requirements. The values are returned with data type "`&xml;hexBinary`".

The following example requires the value of the extension having OID "`1.2.840.113549.1.1.5`", without regard to criticality, to match the value "0xFFABC123" in hex binary form. In this case there may be multiple certificates in the message, and by used of the "`&xfunc;any-of`" function, it is required that at least one of them have this extension with this value.

```
<Apply FunctionId="&wspc;function:limit-scope">
   <AttributeValue DataType="&xsd;string">
//S11:Envelope/S11:Header/wsse:Security/wsse:BinarySecurityToken
   </AttributeValue>
   <Apply FunctionId="&xfunc;anyURI-equal">
      <AttributeValue DataType="&xsd;anyURI">#X509v3</AttributeValue>
      <AttributeSelector DataType="&xsd;anyURI"
          RequestContextPath="/@ValueType"/>
   </Apply>
   <Apply FunctionId="&xfunc;anyURI-equal">
      <AttributeValue
DataType="&xsd;anyURI">#hexBinary</AttributeValue>
      <AttributeSelector DataType="&xsd;anyURI"
          RequestContextPath="/@EncodingType"/>
   </Apply>
   <Apply FunctionId="&xfunc;any-of">
      <Apply FunctionId="&xfunc;hexBinary-match">
      <AttributeValue
DataType="&xsd;hexBinary">FFABC123</AttributeValue>
      <Apply FunctionId="&wspc;function:hexBinary-
getCertExtensionValue">
         <AttributeValue DataType="&xsd;string">
              1.2.840.113549.1.1.5</AttributeValue>
         <AttributeValue DataType="&xsd;string">
```

```
                CriticalOrNot</AttributeValue>
            <AttributeSelector RequestContextPath="/text()"
DataType="&xsd;hexBinary"/>
         </Apply>
      </Apply>
</Apply>
```

6.2.3 Kerberos Token

6.2.3.1 /wssp:SecurityToken/wssp:Claims/wssp:ServiceName

This element in a *WS-SecurityPolicy* `<SecurityToken>` is used with a Kerberos token to specify the service's `PrincipalName` (`sname` field defined in RFC1510). The draft *Web Services Security Kerberos Token Profile 1.0* specifies that a Kerberos ticket is included in a `<wsse:Security>` header using the `<wsse:BinarySecurityToken>` described in *WS-Security.* Since this token is not in an XML format, a special function must be written to extract the `sname` field.

The following *WS-PolicyConstraints* version of this constraint uses a new XACML extension function called "`string-getKrb5SName`". It takes as input a reference to a Kerberos ticket in a `<wsse:BinarySecurityToken>`. It returns the service's `sname` as a string.

The following example requires the value of the service name to match the regular expression "`.*\.WORLD`". In this case there may be Kerberos tickets in the message, and it is required that at least one of them have this service name.

```
<Apply FunctionId="&wspc;function:limit-scope">
   <AttributeValue DataType="&xsd;string">
      //S11:Envelope/S11:Header/wsse:Security/wsse:BinarySecurityToken/
   </AttributeValue>
   <Apply FunctionId="&xfunc;anyURI-equal">  <!-- valueType -->
      <AttributeValue DataType="&xsd;anyURI">
http://www.docs.oasis-open.org/wss/2004/07/oasis-000000-wss-kerberos-
token-profile-1.0#Kerberosv5_AP_REQ
      </AttributeValue>
      <AttributeSelector DataType="&xsd;anyURI" RequestContextPath=
         "/@ValueType"/>
   </Apply>
   <Apply FunctionId="&xfunc;anyURI-equal">  <!-- encodingType -->
      <AttributeValue DataType="&xsd;anyURI">#base64Binary
      </AttributeValue>
      <AttributeSelector DataType="&xsd;anyURI" RequestContextPath=
         "/@EncodingType"/>
   </Apply>
   <Apply FunctionId="&xfunc;any-of">
      <Apply FunctionId="&xfunc;string-regexp-match">
         <AttributeValue
DataType="&xsd;string">.*\.WORLD</AttributeValue>
         <Apply FunctionId="&wspc;function:string-getKrb5SName">
            <AttributeSelector RequestContextPath=
                "/text()" DataType="&xsd;base64Binary"/>
         </Apply>
      </Apply>
   </Apply>
</Apply>
```

6.2.4 Username Token

6.2.4.1 /wssp:SecurityToken/wssp:Claims/wssp:UsePassword

This element in *WS-SecurityPolicy* specifies the requirements on the `<wsse:Password>` element in the

<wsse:UsernameToken>. The requirements are included in a Type XML attribute that may have the value wsse:PasswordText or wsse:PasswordDigest. The *Web Services Security Username Token Profile 1.0* extends the <UsernameToken> element defined in *WS-Security* with a "Password" element having a "Type" attribute.

Using *WS-PolicyConstraints,* requirements on the values for this "Type" attribute can be expressed as follows.

```
<Apply FunctionId="&xfunc;anyURI-is-in">
   <AttributeValue
DataType="&xsd;anyURI">#PasswordText</AttributeValue>
   <AttributeSelector
      DataType="&xsd;anyURI"
      RequestContextPath=
"//S11:Envelope/S11:Header/wsse:Security/wsse:UsernameToken/wsse:Passwo
rd/@Type"/>
</Apply>
```

6.3 Integrity Assertion

WS-SecurityPolicy uses the <wssp:Integrity> Assertion to indicate required signature formats.

The following elements and XML attributes are used in this Assertion and can be expressed using the *WS-PolicyConstraints* predicates described below.

6.3.1 /wssp:Integrity/wssp:Algorithm/@wssp:Type

This XML Attribute is used in a <wssp:Integrity> Assertion to indicate an algorithm type, where the values can be wsse:AlgCanonicalization, wsse:AlgSignature, or wsse:AlgTransform.

In *WS-PolicyConstraints* the algorithm type will be specified by the path selected for the algorithm identifier in the next example.

6.3.2 /wssp:Integrity/wssp:Algorithm/@wssp:URI

This XML Attribute is used in a <wssp:Integrity> Assertion to indicate the URI associated with an algorithm. The following *WS-PolicyConstraints* predicate can be used to indicate that a canonicalization algorithm with the URI "http://www.w3.org/2001/10/xml-exc-c14n" is required.

```
<Apply FunctionId="&xfunc;anyURI-is-in">
   <AttributeValue DataType="&xsd;anyURI">
       http://www.w3.org/2001/10/xml-exc-c14n</AttributeValue>
   <AttributeSelector DataType="&xsd;anyURI"
RequestContextPath="//S11:Envelope/S11:Header/wsse:Security/ds:SignedIn
fo/ds:CanonicalizationMethod/@Algorithm"/>
</Apply>
```

6.3.3 /wssp:Integrity/wssp:TokenInfo/wssp:SecurityToken

This element "indicates a supported security token format or authority previously described". This element is under-specified in *WS-SecurityPolicy* so it is not possible to determine for sure what the contents of of such a <wssp:SecurityToken> element should be. If the value is a <wsse:Reference> to a previously specified <wssp:SecurityToken> element, then in *WS-PolicyConstraints,* this Assertion would be replaced with a predicate that references the <wsse:SecurityTokenReference> element in the signature's <ds:KeyInfo> element, as follows.

```
<Apply FunctionId="&xfunc;anyURI-is-in">
   <AttributeValue DataType="&xsd;anyURI">#X509Token</AttributeValue>
   <AttributeSelector DataType="&xsd;anyURI" RequestContextPath=
"//S11:Envelope/S11:Header/wsse:Security/s:Signature/ds:KeyInfo/wsse:Se
curityTokenReference/wsse:Reference/@URI"/>
```

```
</Apply>
```

6.3.4 /wssp:Integrity/wssp:Claims

This element "contains data that is interpreted as describing general claims that must be expressed in the security token." In *WS-SecurityPolicy,* the specific claims would have to be specified in some extension document, along with their interpretation, matching, and verification semantics.

In *WS-PolicyConstraints,* claims are just constraints. These can be created without requiring any extensions. For example, if some particular string value is required in a UsernameToken, this could be expressed as follows:

```
<Apply FunctionId="&xfunc;string-is-in">
   <AttributeValue DataType="&xsd;string">...required
value...</AttributeValue>
   <AttributeSelector DataType="&xsd;string"
RequestContextPath="//S11:Envelope/S11:Header/wsse:Security/wsse:Userna
meToken/...path to location of required value..."/>
</Apply>
```

For stating constraints about contents of BinarySecurityToken types, new XACML extension functions will usually be needed to extract the desired field from the encoded token. Any implementation of *WS-SecurityPolicy* must also have such extraction functions implemented internally.

6.3.5 /wssp:Integrity/wssp:MessageParts

This element's text contents "is an expression that specifies the targets to be signed. The evaluation of the expression is determined by the optional dialect attribute."

In *WS-PolicyConstraints*, this type of Assertion would be replaced with a predicate that constrains the contents of the `ds:Reference/@URI` XML attribute in the `<ds:SignedInfo>` element. For example, a requirement that the "Body" of the message must be signed would be expressed as follows:

```
<Apply FunctionId="&xfunc:anyURI-is-in">
   <AttributeValue DataType="&xsd;anyURI">#body</AttributeValue>
   <AttributeSelector DataType="&xsd;anyURI"
RequestContextPath="//S11:Envelope/S11:Header/wsse:Security/ds:Signatur
e/ds:SignedInfo/ds:Reference/@URI"/>
</Apply>
```

No specific predicate is needed in *WS-PolicyConstraints* to express the `<wssp:MessageParts[@dialect]>` XML attribute, since *WS-PolicyConstraints* always uses XPath in its `<AttributeSelector>` elements.

6.3.6 /wssp:Integrity/wssp:MessageParts/@Signer

In *WS-SecurityPolicy,* "this optional attribute contains a list of one or more URI references that indicate which nodes must provide a signature", where the pre-defined value is the *WS-Security* URI associated with the role or actor XML attribute in the `<wsse:Security>` header element.

In *WS-PolicyConstraints,* this component of the `<wssp:Integrity>` Assertion can be expressed as follows.

```
<Apply FunctionId="&wspc;function:limit-scope">
   <AttributeValue DataType=&xsd;string">
      //S11:Envelope/S11:Header/wsse:Security/</AttributeValue>
   <Apply FunctionId="&xfunc;anyURI-is-in">
      <AttributeValue DataType="&xsd;anyURI">
          ...Actor URI...</AttributeValue>
      <AttributeSelector RequestContextPath=
"//S11:Envelope/S11:Header/wsse:Security/@S11:actor"/>
   </Apply>
   <Apply FunctionId="&wspc;function:must-be-present">
```

```
      <AttributeValue
DataType="&xsd;string">/ds:Signature</AttributeValue>
   </Apply>
</Apply>
```

This predicate would usually be used as part of one or more of the other predicates placing constraints on the <ds:Signature> element for the specified actor.

6.4 Confidentiality Assertion

The <wssp:Confidentiality> Assertion places constraints on the use of encryption within the message.

In *WS-PolicyConstraints,* the components of this Assertion would be replaced with predicates identical to those specified above for the <wssp:Integrity> Assertion, except that the <AttributeSelector> would select elements or XML attributes in the //S11:Envelope/S11:Body/xenc:EncryptedData node. These predicates are not spelled out here as they are directly comparable to those shown above.

6.5 Visibility Assertion

The <wssp:Visibility> Assertion describes parts of the message that must either be unencrypted, or must be encrypted for a particular wsse:actor or wsse:role.

In *WS-PolicyConstraints,* a requirement that parts of the message NOT be encrypted would be expressed in a predicate such as the following, which requires that nothing in the body of the message be encrypted.

```
<Apply FunctionId="&wspc;function:must-not-be-present">
   <AttributeSelector
RequestContextPath="//S11:Envelope/S11:Body/xenc:EncryptedData"/>
</Apply>
```

The *WS-SecurityPolicy* specification does not describe how a <wssp:Visibility> element specifies an encryption target for a particular actor. The example claims to require that the body be visible to the "http://www.fabrikan123.com" endpoint, but that detail seems to have been omitted.

A requirement that a particular part of the message be encrypted for a particular intermediary would be expressed in a predicate such as the following, which requires that the message contain an encrypted key intended for recipient "http://www.fabrikan123.com".

```
<Apply FunctionId="&xfunc;string-is-in"/>
   <AttributeValue DataType="&xsd;string">
        http://www.fabrikan123.com</AttributeValue>
   <AttributeSelector DataType="&xsd;string" RequestContextPath=
"//S11:Envelope/S11:Header/enc:EncryptedData/ds:KeyInfo/enc:EncryptedKe
y/@Recipient/>
</Apply>
```

This could be combined with a constraint indicating that this key is used to encrypt the <S11:Body> of the message.

```
<Apply FunctionId="&wspc;function:limit-scope">
   <AttributeValue DataType="&xsd;string">
        //S11:Envelope/S11:Header/enc:EncryptedData</AttributeValue>
   <Apply FunctionId="&xfunc;string-is-in">
      <AttributeValue DataType="&xsd;string">
          http://www.fabrikan123.com</AttributeValue>
      <AttributeSelector DataType="&xsd;string" RequestContextPath=
         "/ds:KeyInfo/enc:EncryptedKey/@Recipient"/>
   </Apply>
   <Apply FunctionId="xfunc;anyURI-is-in">
      <AttributeValue DataType="&xsd;anyURI">#Body</AttributeValue>
      <AttributeSelector Datatype="&xsd;anyURI" RequestContextPath=
```

```
            "/enc:CipherData/enc:CipherReference/@URI"/>
      </Apply>
   </Apply>
```

6.6 Security Header Assertion

WS-SecurityPolicy uses the <wssp:SecurityHeader> Assertion to indicate requirements on the <wsse:Security> elements in the header in a *SOAP* message.

The following elements and XML attributes are used in this Assertion and can be expressed using the *WS-PolicyConstraints* predicates described below.

6.6.1 /SecurityHeader/@MustPrepend

This XML attribute is used to specify that, when new <wsse:Security> elements are added to a SOAP message header, they must be prepended. This is helpful when processing order is important, such as whether encryption is being done before or after signature generation.

This constraint is a requirement on the creation process for a *WS-Security* header rather than a requirement on its content. In order to express this policy, a new policy variable or vocabulary item must be created about which policy can be stated. As an example, we will use the new vocabulary item defined in *WS-SecurityPolicy* for this, but this item could as easily be a new XACML Attribute or an XML element simpler than the one defined in *WS-SecurityPolicy.* Note that, even though an element from *WS-SecurityPolicy* is used in stating the policy, the policy processor does not need to understand anything about *WS-SecurityPolicy,* and needs no new code in order to process this constraint.

```
<Apply FunctionId="&xfunc;boolean-is-in">
   <AttributeValue DataType="&xsd;boolean">true</AttributeValue>
   <AttributeSelector DataType="&xsd;boolean" RequestContextPath=
        "//wsp:SecurityHeader/@MustPrepend"/>
</Apply>
```

This requirement can not be enforced by a policy processor, as the receiver has no way of knowing whether the operations were actually done in the order indicated by the header elements. One symptom of failure to perform operations in the specified order is that signatures will fail to verify and encryption cannot be successfully decrypted, but unless the receiver tries all alternative orderings, there is no way to distinguish this case from the case in which an invalid signature or encryption has been done. Even though the requirement can not be enforced, it is important for the sender and the receiver to be able to agree on their policy.

Since header order can be important, it may be appropriate for a future revision or profile of *WS-Security* specify that <wsse:Security> header elements must occur in the order in which they are to be processed.

6.6.2 /SecurityHeader/@MustManifestEncryption

This XML attribute "indicates that only encryptions listed or referenced from the <Security> header will be processed; any encryptions in the message not referenced will be ignored. If false (the default), then the processor MUST search the message for applicable encryptions to process."

As with "@MustPrepend" above, this is a requirement on the creation and processing of a SOAP message and its <wsse:Security> headers rather than a requirement on the content of the message itself. In order to express this policy, a new policy variable or vocabulary item must be created about which policy can be stated. As an example, we will use the new vocabulary item defined in *WS-SecurityPolicy* for this, but this item could as easily be a new XACML Attribute or an XML element simpler than the one defined in *WS-SecurityPolicy.* Note that, even though an element from *WS-SecurityPolicy* is used in stating the policy, the policy processor does not need to understand anything about *WS-SecurityPolicy,* and needs no new code in order to process this constraint.

```
<Apply FunctionId="&xfunc;boolean-is-in">
   <AttributeValue DataType="&xsd;boolean">true</AttributeValue>
```

```
   <AttributeSelector DataType="&xsd;boolean" RequestContextPath=
        "//wsp:SecurityHeader/@MustManifestEncryption"/>
</Apply>
```

This requirement can not be enforced by a policy processor, as the receiver has no way of knowing whether all encrypted elements in the body that the sender intended to have processed were actually referenced form the `<wsse:Security>` header. Even though the requirement can not be enforced, it is important for the sender and the receiver to be able to agree on their policy.

6.7 MessageAge Assertion

WS-SecurityPolicy uses the `<wssp:MessageAge>` Assertion to indicate requirements on the `<wsse:TimeStamp>` element in the `<Security>` Header in a *SOAP* message.

The following elements and XML attributes are used in this Assertion and can be expressed using the *WS-PolicyConstraints* predicates described below.

6.7.1 /MessageAge/@Age

This XML attribute "specifies the actual maximum age timeout for a message expressed in seconds."

In *WS-PolicyConstraints,* such requirements can be specified by placing constraints on the value of the `<wsse:TimeStamp>` element itself. For example, the following constraint requires a maximum age of 3600 seconds. We use the existing XACML Attribute for "current time".

```
<Apply FunctionId="&xfunc;dateTime-greater-than-or-equal">
   <Apply FunctionId="&func;dateTime-add-dayTimeDuration">
      <Apply FunctionId="&func;dateTime-one-and-only">
         <AttributeSelector
RequestContextPath="/S11:Envelope/S11:Header/wsse:Security/wsu:Timestam
p/wsu:Created/text()" DataType="&xsd;dateTime"/>
      </Apply>
      <AttributeValue
DataType="&xqo;dayTimeDuration">3600S</AttributeValue>
   </Apply>
   <xacml:EnvironmentAttributeDesignator
AttributeId="urn:oasis:names:tc:xacml:1.0:environment:current-time"
DataType="&xsd;dateTime"/>
</Apply>
```

7 Lessons learned and future work

This exercise in translating the `Assertion`s defined in *WS-SecurityPolicy* for use with *WS-Security* has yielded some new perspectives on the problem of expressing policies, and has also reinforced some others.

7.1 XPath intersections

Most importantly, we need a subset of *XPath* to use in referring to nodes in instances of a schema, such as the schema for *WS-Security*, such that it is possible to detect when two different policies refer to the same node. Use of *XPath* expressions that do not use functions and or special node selectors such as "`<path>[2]`" seems to be adequate for the requirements of expressing *WS-SecurityPolic*y constraints, but more verification is needed.

7.2 New functions

The following new functions were found to be needed and have been added to Draft 5 of the *WS-PolicyConstraints* specification.

1) `&wspc;function:string-to-uri:` converts a value of type `string` (that can be interpreted as a valid URI) to a value of type `anyURI`. XACML has a `url-string-concatenate` function, but this is not capable of composing a URI from fragments that are themselves not valid URIs (#body). More flexible composition of URIs may be needed for matching references in the body of the SOAP message to the corresponding elements in the SOAP header. This draft managed to express all requirements from *WS-PolicyConstraints* without this function, but it may be needed for more specific correlation requirements.
2) "`&wspc;hexBinary-getCertExtensionValue`". It takes as input a string value interpreted as the OID of the desired extension, a string value indicating required criticality (with acceptable values of "`Critical`", "`NotCritical`", and "`CriticalOrNot`"), and a reference to an ASN-encoded X.509 certificate, encoded as `&xsd;hexBinary`. A corresponding function that takes a certificate encoded in `&xsd;base64Binary` may also be needed. Functions that return the extension value as `&xsd;string` or `&xsd;integer` may also be useful.
3) "`&wspc;string-getKrb5SName`". It takes as input a reference to a Kerberos ticket in a `<wsse:BinarySecurityToken>` that is encoded in `&xsd;base64Binary`. It returns the service's `sname` as a string. Two versions of this function may actually be needed, one for the case where the ticket is encoded in `&xsd;base64Binary` and the other where the ticket is encoded in `&xsd;hexBinary`.

Note that all functions other than the first are for dealing with non-XML data embedded in the *WS-Security* instance. Implementations of *WS-SecurityPolicy* must supply similar functions internally in order to deal with these types of `Assertion`s.

If the solution to the problem of matching a URI reference and target is to create a `&wspc;string-url-concatenate` function, then *WS-PolicyConstraints* must support embedding this function in a constraint.

7.3 Constraints on the message processor

Using only direct references to instances of *WS-Security,* it is not possible to create constraints that impose requirements on the creation or processing of security aspects of a SOAP message. An example is a requirement that `<Security>` elements in a header must be prepended, such that the order in which they occur in the document is the reverse of the order in which the corresponding operations were performed. This is an example of where some new policy element, such as the one defined in *WS-SecurityPolicy* must be defined. In this draft of this profile, we have used the elements defined in *WS-SecurityPolicy*, but new XACML Attributes or new simple XML elements could be used

instead. Since much of the *WS-SecurityPolicy* schema is vocabulary items that can actually be referenced directly in the SOAP message, the *WS-SecurityPolicy* schema could be considerably simplified and reduced if *WS-PolicyConstraints* were used. Also, if *WS-PolicyConstraints* is used, no special processor will be needed for the *WS-SecurityPolicy* (or XACML) vocabulary items because they are used simply as vocabulary item identifiers, and have no special semantics that the policy processor must understand.

Since there is no way to verify this requirement in general, and since the typical use of the requirement is to ensure correct processing of messages, it seems more reasonable to require use of a *WS-Security* profile (or future revision of *WS-Security* itself) that always requires that `<Security>` elements in the header be prepended.

This is a constraint on the message processor, and could also be handled by the creation of a schema for describing characteristics of the message processor, with an instance of this schema included in the *SOAP* `<S11:Header>`. Then *WS-PolicyConstraints* could be used to specify constraints directly on such instances.

7.4 Overly constrained policies

In several cases, *WS-PolicyConstraints* requires that a *WS-Security* instance be constructed in a particular way, when alternative expressions would be possible and equally valid. Multiple *WS-PolicyConstraints* constraints could be OR'ed together to capture all such allowable alternatives, but it might be simpler to define a profile of *WS-Security* that specifies one way of expressing a requirement where multiple alternatives exist. Such a profile could be defined for use with *WS-Security* instances that are to be used with *WS-Policy.* An example is various ways in which signature or encryption information in the `<wsse:Security>` header might be referenced from the body of the SOAP message.

7.5 Cost and value of abstraction

WS-SecurityPolicy defines abstract ways to specify constraints. These abstract specifications are in many cases easier to compare than the more specific constraints expressed in *WS-PolicyConstraints.* Two considerations place this difference in a different perspective, however.

The first is that each abstraction in *WS-SecurityPolicy* must be resolved into specific forms in order to enforce a given policy. In order to determine that a given instance of *WS-Security* conforms to a *WS-SecurityPolicy* policy, the engine that implements *WS-SecurityPolicy* must be able to recognize all the specific ways of expressing *WS-Security* and all the variations that are equally valid. A *WS-PolicyConstraints* policy, however, can be used not only to match policies, but can also be used directly to enforce the policy against a particular instance. It would probably be a good idea for the tools used to create policies to support more abstract terms. The tool should then translate these terms into the specific *WS-PolicyConstraints* predicates actually required to negotiate and verify policy.

The second is that, while it may be simpler to express an abstract constraint using *WS-SecurityPolicy*, the processing of these abstract constraints is still complex: in fact, new processor code must be created to handle every possible abstract constraint as applied to every possible concrete message format. With a standard predicate language such as *WS-PolicyConstraints,* every possible constraint can be handled by one standard processor. The simplicity should be provided at the policy authoring level, and not at the concrete policy instance level.

8 References

[RFC2119] S. Bradner, *Key words for use in RFCs to Indicate Requirement Levels*, IETF RFC 2119, March 1997, http://www.ietf.org/rfc/rfc2119.txt.

[SOAP] M. Gudgin, et al., eds., *SOAP Version 1.2 Part 1: Messaging Framework,* W3C Recommendation, 24 June 2003, http://www.w3.org/TR/2003/REC-soap12-part1-20030624/.

[WSP] S. Bajaj, et al., *Web Services Policy Framework (WS-Policy)*, September 2004,http://www.ibm.com/developerworks/library/specification/ws-polfram/

[WSPA] T. Nadalin, ed., *WS-PolicyAssertions,* 28 May 2003, http://www.ibm.com/developerworks/library/ws-polas

[WSPC] A. Anderson, *XACML-Based Web Service Policy Constraint Language (WS-PolicyConstraints),* Working Draft 05, 27 June 2005, http://research.sun.com/projects/xacml/ws-policy-constraints-wd-05.pdf.

[WSPL] T. Moses, ed., *XACML profile for Web-services*, OASIS Access Control (XACML) TC, Working Draft 04, 29 September 2004, http://www.oasis-open.org/committees/download.php/3661/draft-xacml-wspl-04.pdf.

[WSR] K. Iwasa, et al., eds., *WS-Reliability v1.1*, OASIS Web Service Reliable Messaging TC, OASIS Standard, 15 November 2004, http://www.oasis-open.org/committees/download.php/9330/WS-Reliability-CD1.086.zip.

[WSRP] Stefan Batres, ed., *Web Services Reliable Messaging Policy Assertion (WS-RM Policy),* February 2005, http://specs.xmlsoap.org/ws/2005/02/rm/WS-RMPolicy.pdf

[WSS] T. Nadalin, et al., eds., *Web Services Security: SOAP Message Security 1.0 (WS-Security 2004),* OASIS Web Services Security TC, OASIS Standard 200401, March 2004, http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-soap-message-security-1.0.pdf.

[WSS-Sch] T. Nadalin, et al. eds., *WS-Security schema*, OASIS Web Services Security TC, OASIS Standard, March 2004, http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=wss.

[WSSP] T. Nadalin, ed., *Web Services Security Policy Language (WS-SecurityPolicy),* Version 1.0, 18 December 2002, http://www.verisign.com/wss/WS-SecurityPolicy.pdf

[XACML] T. Moses, ed., *eXtensible Access Control Markup Language (XACML) Version 2.0*, OASIS Access Control (XACML) TC, OASIS Standard, 1 February 2005, http://docs.oasis-open.org/xacml/2.0/access_control-xacml-2.0-core-spec-os.pdf.

[XDS] Mark Bartel, et al., eds., *XML-Signature Syntax and Processing*, W3C Recommendation, 12 February 2002, http://www.w3.org/TR/2002/REC-xmldsig-core-20020212/

[XENC] Donald Eastlake, et al., eds., *XML Encryption Syntax and Processing*, W3C Recommendation, 10 December 2002, http://www.w3.org/TR/xmlenc-core/.

A. Revision History

Rev	Date	By Whom	What
01	4 April 2005	Anne Anderson	Initial version: Replacing WS-SecurityPolicy with WS-PolicyConstraints. File name: Example-WS-SecurityPolicy
02	30 May 2005	Anne Anderson	Added much more introductory material.
03	28 June 2005	Anne Anderson	Changed title: WS-Security profile of WS-PolicyConstraints. Made examples consistent with "lessons learned" from previous versions. File name: ws-security-profile-of-ws-policy-constraints
04	1 December 2005	Anne Anderson	Clarified relationship with WS-SecurityPolicy: not a replacement, but a "proof-of-concept".

B. Notices