Web-Services Security Quality of Protection

1. Problem statement

WSS allows Web-service providers to implement a security policy.

The term security policy is used in this context to mean: "a statement of the requirements for protecting arguments in a WS API, including:

- how actors are to be authenticated, using what mechanisms and with what parameter value ranges,
- which XML elements are to be encrypted, for what individual recipients, recipient roles or keys, using what algorithms and key sizes,
- which XML elements are to be integrity protected, using what mechanisms, with which algorithms and key sizes, and
- what additional qualifications the service consumer must demonstrate in order to successfully access the API".

This is a relatively restrictive use of the term "security policy". A more comprehensive definition addresses such requirements as:

- privacy (retention period, intended usage, further disclosure),
- trust (initial parameters of the signature validation procedure, including those keys or authorities that are trusted directly, policy identifiers, maximum trust path length), and
- non-repudiation (requirements for notarization and time-stamping).

These are outside the scope of WSS, so we don't address them here.

In some situations (notably RPC-style service invocation), Web service interactions are conducted in a persistent session. In such cases, the provider's security policy may indicate the requirements for authenticity, integrity and confidentiality of the session.

In other situations (notably document-style service invocation), messages must be protected in isolation. In these cases, while the service provider may declare a policy, the service consumer actually creates the service request. So, the consumer actually chooses which security operations to apply to the messages. Therefore, we need a way for the consumer to discover the service provider's policy and choose a set of security operations that are consistent with both its own and the service-provider's security policy.

When a service provider sets its security policy it may do so with little or no knowledge of the mitigating effects of other safeguards that may be in place. For instance, its consumers may be on the same LAN segment or outside the firewall. We assume here that policy is enforced uniformly on all messages of a particular type in order to address the most severe vulnerabilities that may be encountered in a service request.

2. Document outline

In this document we describe an approach to solving the stated problem.

Section 3 describes how this problem is solved today and points out the shortcomings of the existing solutions. Section 4 describes the proposed approach in outline. Section 5 describes a number of process models for applying the proposed approach. Section 6 describes the schema of the proposed element. The appendix contains the schema listing.

3. Existing solutions

Two main solutions to this problem already exist:

1. the consumer and provider agree, in some unspecified manner, which protection mechanisms to apply to which elements, and

2. the provider is capable of accepting, and willing to accept, a broad range of mechanisms, and the consumer chooses which mechanism to use and which elements to apply them to.

The first approach is probably satisfactory when the provider and consumer are governed by the same policy authority. This situation exists most commonly where the provider and consumer are in applications owned and operated by the same corporate entity.

The second approach is probably satisfactory in circumstances where the security policy may be set entirely by the consumer.

Neither of these solutions is satisfactory in a federated environment, where both provider and consumer independently define their own security policies and messages have to be exchanged in a way that satisfies both policies.

4. Basic approach

We propose to extend WSDL to support the publication of the provider's security policy, as illustrated in Figure 1. Specifically, an element called <QoPSpec> is added to both the input and output bindings of each operation in the WSDL instance. The <QoPSpec> element is produced by a security authority in the service provider's domain.

In the case of the input binding, the <QoPSpec> element describes the combinations of security operations that are acceptable under the service-provider's security policy. The security authority in the consumer's domain selects one of the combinations of security operations specified in the input <QoPSpec> element that is acceptable under its own policy and causes those operations to be applied to service request messages.

In the case of the output binding, the <QoPSpec> element identifies the combination of security operations that ***will*** be applied to the output message. The consumer must decide if it can, and is willing to, accept this security policy.

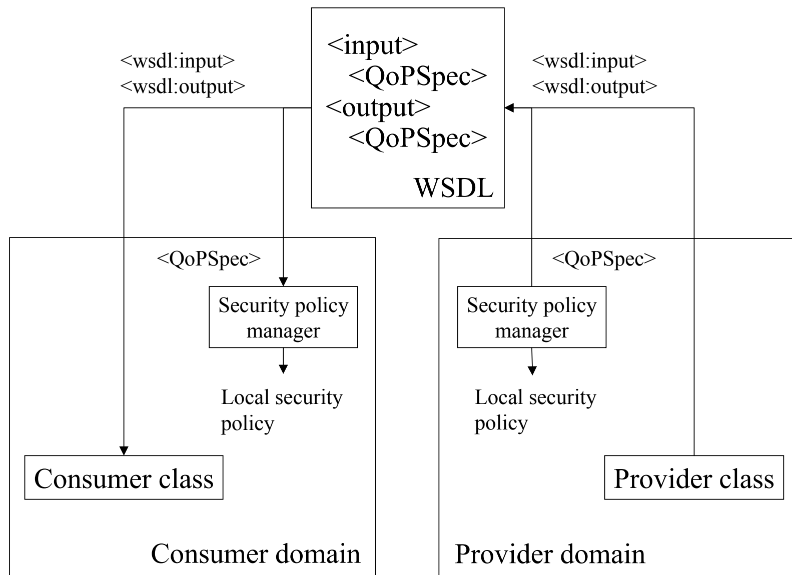

Figure 1 - Basic approach

5. Process models

Three process models are described:

- The development-time model,
- The deployment-time model and
- The run-time model.

In the development-time model, the security policy is discovered by the service consumer at the time at which the consumer class is developed. In the deployment-time model, the policy is discovered by the service consumer at the time at which the consumer class is deployed. And in the run-time model, the security policy is discovered by the service consumer at run-time.

5.1. Development-time model

5.1.1. Service provider

The service provider development environment exports an interface definition in the form of a WSDL instance. In addition, it generates a header that deserializes the input and invokes the service provider class and serializes the class return arguments. See Figure 2.

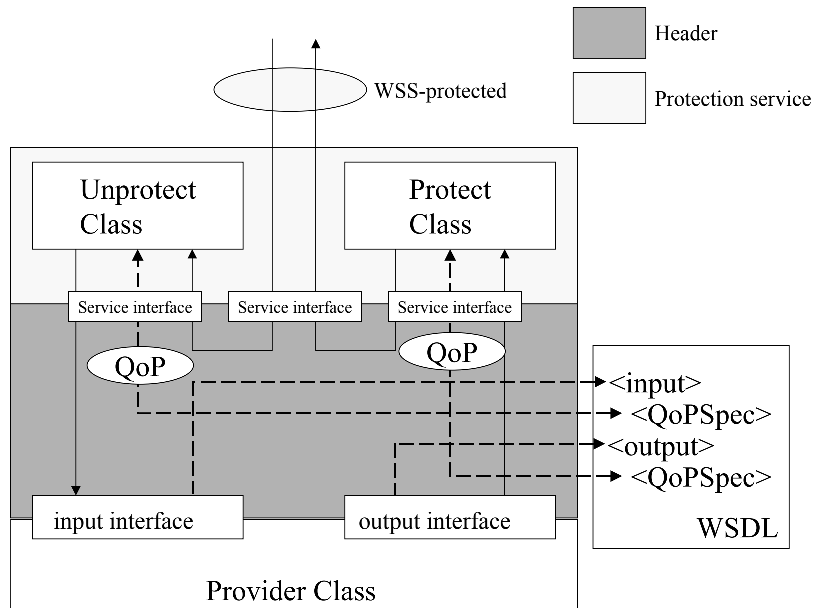

Figure 2 - Service provider development process

In the case where the serialized stream must be WS Security protected, we require additional steps in the header to unprotect the request arguments and protect the response arguments and attach the tokens supplied by the protect class, in accordance with the service-provider's security policy.

Should it be discovered that the input is not protected in accordance with the provider's security policy, an exception may be thrown (see Section 7).

It is worth noting that the <QoPSpec> element serves two functions: it tells the message originator what security operations to apply and it allows the message recipient to identify messages that violate the policy.

5.1.2. Service consumer

The service consumer development environment imports a suitable WSDL instance and generates a stub that is invoked by the consumer class and serializes the output and deserializes the input. See Figure 3.

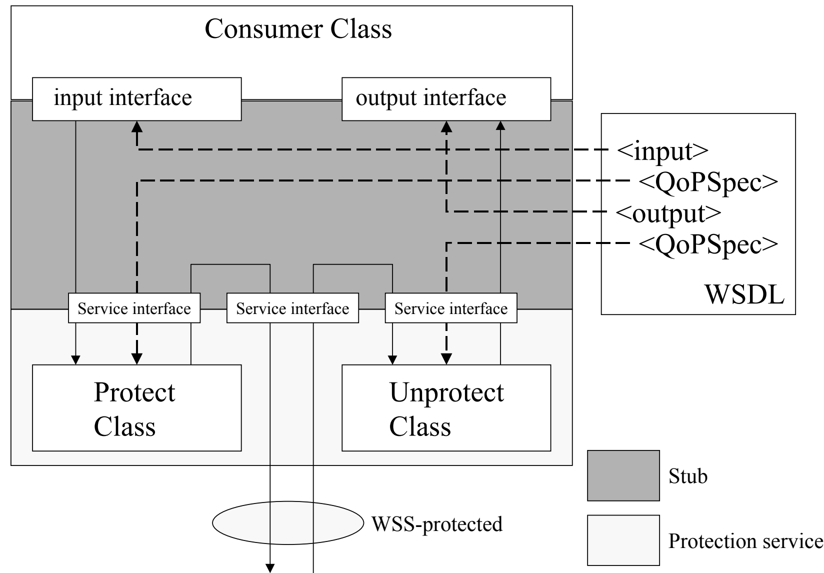

Figure 3 - Service consumer development process

In the case where the serialized stream must be WS Security protected, we require additional steps in the stub to protect the request arguments and attach the tokens supplied by the protect class and unprotect the response arguments in accordance with the service-provider's security policy.

5.1.3. Shortcomings

The main shortcoming with this model is that the security policy is set by the developer, whereas it is common for security policy to be set and managed by a different group, particularly if development is out-sourced or off-the-shelf classes are used. Furthermore, it prevents the modification of security policy during the lifetime of the provider class.

5.2. Deployment-time model

In order to overcome the shortcomings of the development-time model described above, a security policy management authority is introduced. Security policy is set and managed by the authority separately from the activities of the class developer. WSDL remains the mechanism for distributing the <QoPSpec> element. But, in this case, the security policy manager must be capable of appending the <QoPSpec> element to the provider class's WSDL instance.

One further model taxonomy applies. The security service could be implemented "out-of-line" (see Section 5.2.1) or "in-line" (see Section 5.2.2).

5.2.1. Out-of-line model

According to the out-of-line model, the provider's security policy manager creates a <QoPSpec> element and publishes it in the WSDL. Additionally, it makes it available at deployment-time to the provider security service. This does not have to be achieved in a standard fashion, although the <QoPSpec> schema is suitable for the purpose. The provider security service is invoked by the container hosting the provider class every time a message is processed, protecting and unprotecting elements, attaching tokens and identifying policy violations (see Figure 4)

The consumer's security manager obtains the <QoPSpec> element from the WSDL at the time of deployment of the consumer class. It merges its own policy (i.e. it selects one of the security services identified in the <QoPSpec> that is consistent with its own security policy) and makes it available to its

security service. This does not have to be achieved in a standard fashion, although the <QoPSpec> schema is suitable for the purpose. As in the case of the provider class, the consumer security service is invoked by the container hosting the consumer class every time a message is processed.

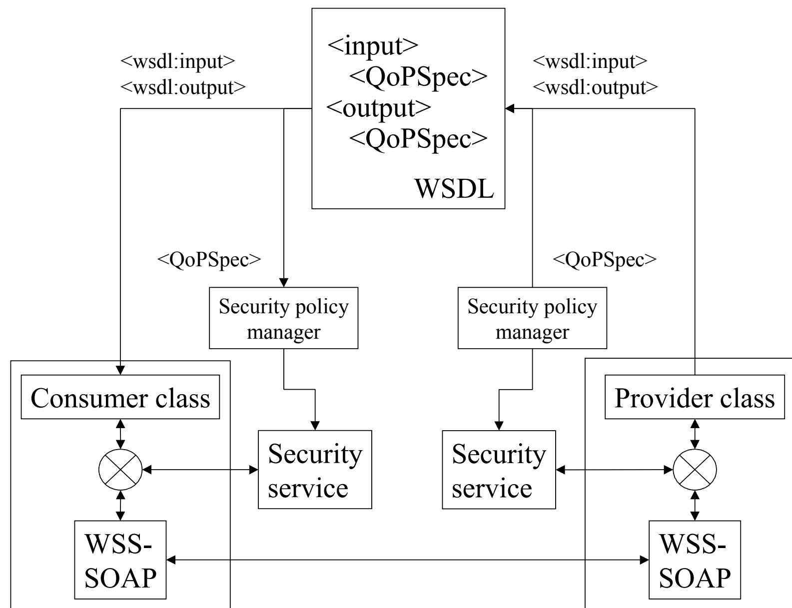

Figure 4 - Out-of-line model

The <QoPSpec> element could be supplied to the security service by the consumer class, either in the WSDL or separately. Alternatively, the security service can obtain the WSDL and extract the <QoPSpec> element on its own, in which case the consumer class should ignore the <QoPSpec> element in the WSDL instance.

5.2.2. In-line model

The in-line model works similarly, with the exception that the containers hosting the consumer and provider classes emit a SOAP message, which is intercepted by the security service. Likewise, the consumer and provider classes could provide the <QoPSpec> element, as a child element of the WSS security header element, with the "security service" identified as the target role. Alternatively, the security service could obtain the <QoPSpec> element on its own (see Figure 5).

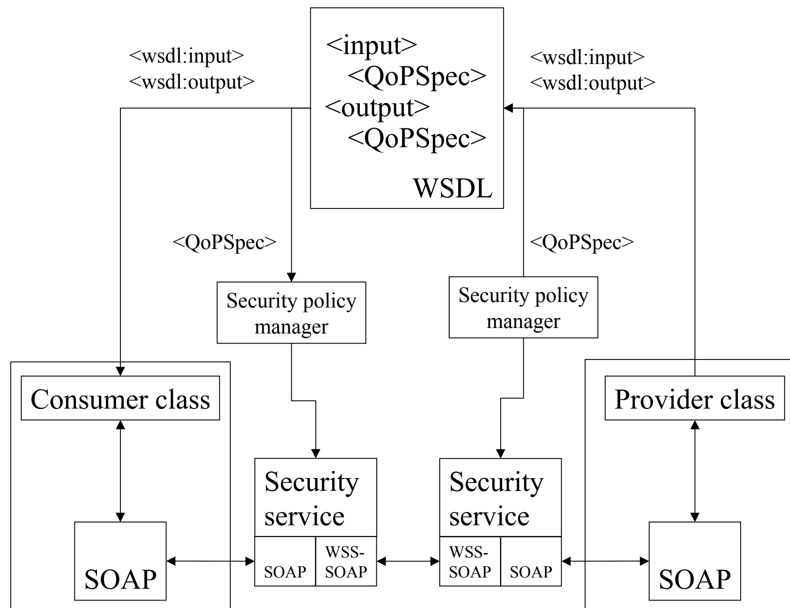

Figure 5 - In-line model

5.3. Run-time model

The only thing that distinguishes the run-time model from the deployment-time model is that the service's WSDL instance and its <QoPSpec> element are retrieved at the time the message is prepared. This approach may be useful if the consumer may obtain the service from any one of a number of providers, each with differing security policies. It may also be useful if the <QoPSpec> element is generated automatically by a security-policy negotiation protocol.

While the approach ensures that the <QoPSpec> element is fresh, since it is always retrieved dynamically, the impact on performance may be unacceptable in many situations.

6. QoPSpec

6.1. <QoPSpec> element

Each <wsdl:input> element and each <wsdl:output> element is extended with one <QoPSpec> element. The <QoPSpec> element specifies the full security policy for the <wsdl:input> element or <wsdl:output> element to which it is attached. A <QoPSpec> element contains one or more <Contract> elements. Security operations must be applied to a <wsdl:input> element by the service consumer, and to a <wsdl:output> element by the service provider, in the order in which they appear in the <QoPSpec> element, and they must be removed from the <wsdl:input> element by the service provider, and from the <wsdl:output> element by the consumer in the reverse order.

```
<xs:element name="QoPSpec" type="QoPSpecType"/>
<xs:complexType name="QoPSpecType">
    <xs:element ref="Contract" maxOccurs="unbounded"/>
</xs:complexType>
```

6.2. <Contract> element

Each <Contract> element defines a set of payload nodes and a single security operation that applies to those nodes. A <Contract> element contains a <Clients> element that identifies the payload nodes and a <Service> element that identifies the security operation to be applied to the payload nodes.

```
<xs:element name="Contract" type="ContractType"/>
<xs:complexType name="ContractType">
    <xs:sequence>
        <xs:element ref="Clients"/>
        <xs:element ref="Service"/>
    </xs:sequence>
</xs:complexType>
```

6.3. <Clients> element

A <Clients> element identifies a set of payload nodes. It contains one or more <Client> elements, each of which identifies a single payload node. The originator must apply the corresponding operation to all the payload nodes.

```
<xs:element name="Clients" type="ClientsType"/>
<xs:complexType name="ClientsType">
    <xs:element ref="Client" maxOccurs="unbounded"/>
</xs:complexType>
```

6.4. <Client> element

A <Client> element identifies a payload node by id and (optionally) assigns an id to the resulting element, so that further operations can be applied if necessary.

```
<xs:element name="Client" type="ClientType"/>
<xs:complexType name="ClientType">
    <xs:sequence>
        <xs:element name="ClientRef" type="xs:string"/>
        <xs:element name="ClientId" type="xs:string" minOccurs="0"/>
    </xs:sequence>
</xs:complexType>
```

The <ClientRef> child element carries location information about the payload. Consequently, this could be a URI representing a MIME contentId, href or XPath/Xpointer. The <ClientRef> element could also point to a remotely located payload component that may need specific protection schemes. The <ClientId> element is used to identify the element that may be created as a result of applying the security operation, so that it can be identified as the target of subsequent security operations in their <ClientRef> elements.

6.5. <Service> element

A <Service> element identifies a single security operation. It contains a <ServiceId> element that identifies the security operation and an optional <ServiceParameters> element containing a set of parameters for the operation. Services with the same ServiceId represent alternatives; representing all the services of that type that are acceptable to the service-provider. There is no significance to the order. The service-consumer SHALL choose one.

```
<xs:element name="Service" type="ServiceType"/>
<xs:complexType name="ServiceType">
    <xs:sequence>
        <xs:element ref="ServiceId"/>
        <xs:element ref="ServiceParameters" minOccurs="0"/>
```

```
    </xs:sequence>
</xs:complexType>
```

Note: Some security operations may not require any parameters.

6.6. <ServiceId> element

A <ServiceId> element identifies a security operation by URN. Some common security operations will be defined and named in the specification.

```
<xs:element name="ServiceId" type="xs:anyURI"/>
```

6.7. <ServiceParameters> element

A <ServiceParameters> element contains a set of parameters associated with the parent security operation. It contains one or more <ServiceParameter> elements.

```
<xs:element name="ServiceParameters" type="ServiceParametersType"/>
<xs:complexType name="ServiceParametersType">
    <xs:sequence>
        <xs:element ref="ServiceParameter" maxOccurs="unbounded"/>
    </xs:sequence>
</xs:complexType>
```

6.8. <ServiceParameter> element

A <ServiceParameter> element contains a single parameter for a security operation. It contains a single <ParameterId> element and a single <ParameterValue> element.

```
<xs:element name="ServiceParameter" type="ServiceParameterType"/>
<xs:complexType name="ServiceParameterType">
    <xs:sequence>
        <xs:element ref="ParameterId"/>
        <xs:element ref="ParameterValue"/>
    </xs:sequence>
</xs:complexType>
```

6.9. <ParameterId> element

A <ParameterId> element contains an identifier for a security parameter in the form of a URN. Some common parameters will be defined and named in the specification. Some security operations may require more than one parameter of a particular type.

```
<xs:element name="ParameterId" type="xs:anyURI"/>
```

6.10. <ParameterValue> element

A <ParameterValue> element contains a single parameter value.

```
<xs:element name="ParameterValue" type="ParameterValueType"/>
<xs:complexType name="ParameterValueType">
    <xs:any/>
</xs:complexType>
```

Note: a structured element, such as <ds:KeyInfo>, could be an identified parameter.

7. Security and privacy considerations

If the <QoPSpec> element were to be distributed without integrity/authenticity protection, an adversary could successfully substitute a weaker policy, thereby tricking the consumer into protecting its messages weakly. This situation is avoided if the consumer domain enforces its own security policy. Alternatively, or additionally, the <QoPSpec> element could be signed by the provider and verified by the consumer.

Naturally, despite having stipulated a policy, a provider may receive a message that violates the policy. The provider's policy SHOULD specify how to address this situation. It could process the message anyway; it could be argued that, in the case of confidentiality, any damage is already done and the situation cannot be mitigated by a refusal to process the message. However, in order to avert a repeat of this failure, it is RECOMMENDED that the provider issue an error, and refuse to process the message.

Appendix A - Schema

Following is the schema for the <QoPSpec> element.

```
<xs:element name="QoPSpec" type="QoPSpecType"/>
<xs:complexType name="QoPSpecType">
    <xs:element ref="Contract" maxOccurs="unbounded"/>
</xs:complexType>
<xs:element name="Contract" type="ContractType"/>
<xs:complexType name="ContractType">
    <xs:sequence>
        <xs:element ref="Clients"/>
        <xs:element ref="Service"/>
    </xs:sequence>
</xs:complexType>
<xs:element name="Clients" type="ClientsType"/>
<xs:complexType name="ClientsType">
    <xs:element ref="Client" maxOccurs="unbounded"/>
</xs:complexType>
<xs:element name="Client" type="ClientType"/>
<xs:complexType name="ClientType">
    <xs:sequence>
        <xs:element name="ClientRef" type="xs:string"/>
        <xs:element name="ClientId" type="xs:string" minOccurs="0"/>
    </xs:sequence>
</xs:complexType>
<xs:element name="Service" type="ServiceType"/>
<xs:complexType name="ServiceType">
    <xs:sequence>
        <xs:element ref="ServiceId"/>
        <xs:element ref="ServiceParameters"" minOccurs="0"/>
    </xs:sequence>
</xs:complexType>
<xs:element name="ServiceId" type="xs:anyURI"/>
<xs:element name="ServiceParameters" type="ServiceParametersType"/>
<xs:complexType name="ServiceParametersType">
    <xs:sequence>
        <xs:element ref="ServiceParameter" maxOccurs="unbounded"/>
    </xs:sequence>
</xs:complexType>
<xs:element name="ServiceParameter" type="ServiceParameterType"/>
<xs:complexType name="ServiceParameterType">
    <xs:sequence>
        <xs:element ref="ParameterId"/>
        <xs:element ref="ParameterValue"/>
    </xs:sequence>
</xs:complexType>
<xs:element name="ParameterId" type="xs:anyURI"/>
<xs:element name="ParameterValue" type="ParameterValueType"/>
<xs:complexType name="ParameterValueType">
    <xs:any/>
</xs:complexType>
</xs:schema>
```