Subproject Support Functions

The WS-Security Standard

- An Overview

2004-05-06

Anders Toms

SERVIAM

Table of Contents

1 Introduction

One of the most dominant Web Services standards is SOAP which defines the protocol for accessing Web Services. The early SOAP standard left much to be desired in the area of security. In fact, security issues were completely ignored in the first version. Version 1.2, however, includes an extensibility model that provides possibilities for extending the specification with add-ons that can deal with issues such as security. In 2002, Microsoft & IBM submitted the WS-Security specification to the OASIS standards development organization since they had realised that security issues would need to be dealt with if Web Services would become the technology for the next generation of distributed computing. WS-Security is an add-on to SOAP that describes how the header part of a SOAP-message can be used to include security information and provide confidentiality, integrity, non-repudiation and more. Since 2002 development of the specification has continued and recently the WS-Security specification was voted by the OASIS members to become an official OASIS standard.

Since WS-Security arguably is an important standard for the future development of Web Services, this report will describe this standard and point out some of its benefits and disadvantages. The report is concluded with a short section on products that start to appear and that claim to have support for the WS-Security standard.

2 The WS-Security stack

One of the major shortcomings of the early Web Services initiatives was that security issues were not explicitly addressed by vendors and standards bodies. Lately, this situation has been acknowledged and there is currently a lot of activity going on in different standards development organizations in order to come up with standards that permit secure Web Services to be deployed over public networks. An initiative in this direction was taken by IBM and Microsoft in June 2002 when they submitted their WS-Security specification to OASIS (Organization for the Advancement of Structured Information Standards). The specification is outlined in a document called “Security in a Web Services World: A Proposed Architecture and Roadmap” (2002) which was first released in April 2002. Prior to that, Microsoft gave the first presentation of WS-Security back in October 2001.

The WS-Security specification is actually only one of several specifications that are described in the proposed roadmap. WS-Security is considered the cornerstone of the proposed security architecture, which is why it is the first specification to be handed over to a standards body for acknowledgement as an official standard. However, Microsoft and IBM intend to continue their work to come up with several other specifications that will build upon WS-Security and that will address other, related, issues. Figure 1 shows the other specifications that will build upon the foundation laid by WS-Security.

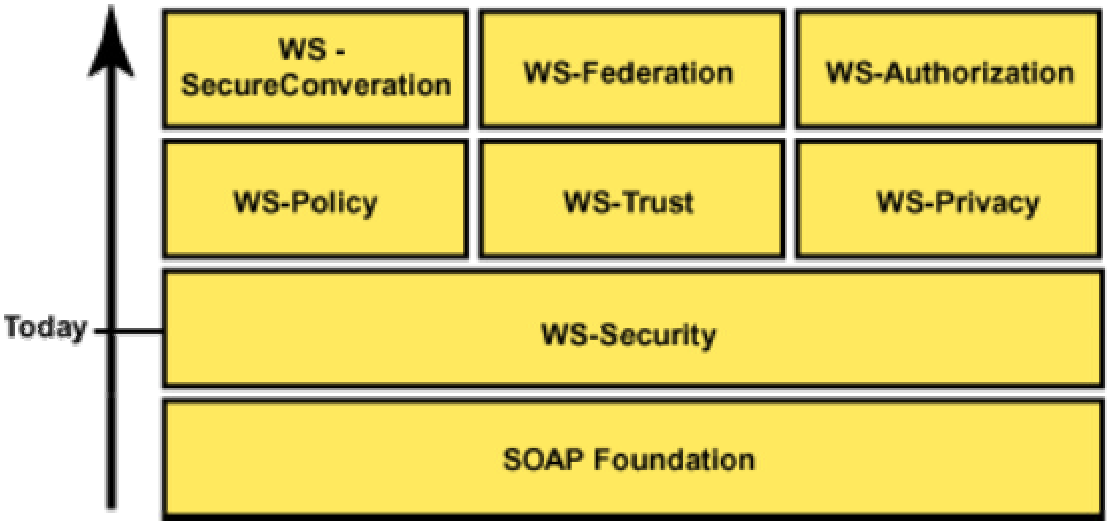

Figure 1. The specifications outlined by IBM & Microsoft (from IBM & Microsoft, 2002).

The SOAP foundation layer consists of core technologies like SOAP, WSDL, XML Signature, XML Encryption and SSL/TLS that are leveraged by the WS-Security specification. The future specifications, building on WS-Security, are briefly described as follows:

- **WS-Policy** Just as the name implies, WS-Policy lets an organisation that exposes a Web Service to specify a policy for that service. The policy details what capabilities and constraints that the service exhibits regarding security, e.g. what type of encryption algorithm it supports and which parameters that need to be encrypted. The policy can be used to discover a service's capabilities and constraints regarding security, similar to how WSDL is used to describe the general characteristics of a service (O'Neil, 2003).

- **WS-Trust** Describes how direct or brokered trust relationships are created. Trust, if looked up in the Cambridge Advanced Learner's Dictionary (2004), is defined as: "to have belief or confidence in the honesty, goodness, skill or safety of a person, organization or thing". In the case of Web Services and security, trust means that you can be confident regarding the authenticity and authorization of an entity if a third party that you trust vouch for this. Examples of common trust models are PKI (a hierarchical trust model) and PGP (a network trust model). The WS-Trust model also allows for delegation and impersonation to be carried out. This means that because a SOAP-message is not sent directly by the end user, but by a software program acting on his or her behalf, a token containing security information that maps back to the end user is inserted into the message. This token is then presented to security controls as the message needs to be authenticated along its path. Thus, the end user is impersonated by the information contained in the token. Delegation and impersonation are essentially consistent with each other, except that delegation provides additional possibilities concerning traceability (IBM & Microsoft, 2002).

- **WS-Privacy** Packaging information regarding an end user and ship that information with the message as it is routed between different nodes may have an impact on end user privacy concerns depending on how sensitive the bundled details are. The WS-Privacy

specification makes it possible for a service provider to express its policy regarding privacy issues and for the service provider to require that incoming requests specify the sender's adherence to the policy. To achieve this, the WS-Privacy specification in turn makes use of a combination of other specifications, including WS-Policy, WS-Security and WS-Trust.

- **WS-Secure Conversation** This specification sets out to make it possible to create sessions spanning several SOAP-messages so that it will not be necessary to evaluate every incoming message with respect to authentication and authorization. The WS-Security does not support sessions which may lead to performance issues. O'Neil (2003) describes WS-Secure Conversation as "SSL at the SOAP level" because SSL will probably stand as a model for how to choreograph and establish the session key. SSL uses the slower public key encryption technology for securely exchanging a session key between the client and the server. Once this key has been established, a much faster private key encryption scheme can be used during the complete session to encrypt the data traffic.

- **WS-Federation** Because different parties involved in consuming and providing a service may use different security technologies, e.g. one party may use Kerberos and another X.509 certificates, it can become necessary to translate security related data between the involved parties. WS-Federation is the specification that will describe how the brokering between parties should be carried out. The specification will operate at a layer above WS-Policy and WS-Trust.

- **WS-Authorization** This specification deals with how to express rules regarding what access is allowed and how these rules can be managed. More specifically, the specification will describe how claims are to be represented in tokens and how these tokens should be interpreted at the endpoint. According to O'Neil (2003) this specification has considerable overlaps with the XACML (Extensible Access Control Markup Language) standard which specifies how to express access control policies in XML-format.

The top three of the six specifications described above are the initial specifications to be developed, according to IBM & Microsoft (2002). Later on, the three specifications at the bottom will be put forward since they build on the previous ones. However, currently, the OASIS Web Services Security Technical Committee (WSS TC) seems to be working only on the WS-Security standard. It is worth pointing out that WS-Security, though it should be studied with the other specifications in mind, can be used separately from all of the other specifications outlined above.

At he time of writing, the WS-Security specification (or as the OASIS WSS TC calls it; SOAP Message Security specification) has just recently been recognised by OASIS as an official standard and more and more organisations are looking closely at WS-Security (CBDI Forum, 2004). The following section will describe in more detail what WS-Security is all about.

3 WS-Security – how does it work?

This section will show some of the basic syntactical constructions that the WS-Security standard uses to include security information in SOAP message headers. To be able to show this we have to study SOAP messages in some detail and this, in turn, requires some basic knowledge of XML and related concepts like e.g. element, attribute and namespace. A very short, but still quite sufficient for the purpose of this section, introduction to XML can be found in Chappell

(2002). A more detailed analysis of XML and related technologies is given in Connolly & Begg (2002). Understanding this section also requires a basic understanding of security technologies like public key cryptography and Kerberos which can be achieved by reading a good security text book, like e.g. Pfleeger & Pfleeger (2003).

Currently, SOAP is one of the protocols that is most tightly associated with Web Services. This is not surprising considering that SOAP is what gets the data from one place on the network (e.g. the service provider) to another (e.g. the service requestor). SOAP uses XML to describe how messages are to be structured so that they can be exchanged between Web Service implementations. Essentially, a SOAP message consists of a mandatory envelope and body and an optional header, as shown in listing 1.

```
                                                                    Envelope
<soap:Envelope xmlns:soap=http://schemas.xmlsoap.org/soap/envelope/>
    <soap:Header>
    </soap:Header>                                          Header
    <soap:Body>
        <GetBalance xmlns=http://www.bigbucksbank.com/bankapp”>
            <Account>8146-12334492</Account>
        </GetBalance>
    </soap:Body>                                                Body
<soap:Envelope>
```

Listing 1. The listing shows an example of a SOAP message where the mandatory envelope and body and the optional header is shown.

The SOAP message in Listing 1 is retrieving the balance for a fictitious bank account designated by the account number specified in the “Account”-element. The important thing to notice here is the empty header part of the message, boxed in by the dashed line. It is in the header part of the message where security related information is placed by most protocols that deal with security issues.

The SOAP 1.2 specification does not directly specify how to deal with any security related issues such as access control, confidentiality, integrity and non-repudiation. Instead, the specification provides an extensibility model that can be used to build extensions to the original SOAP standard. This possibility to provide extensions for the SOAP protocol is what is used in the WS-Security specification to provide security functions such as authentication, integrity and confidentiality. Hence, WS-Security is basically an add-on to SOAP that provides data protection.

The WS-Security specification adds security to SOAP messages by describing how the header part of the message can be used to pass along security information. Other parts of the message could possibly been used for placing security metadata, but the IBM and Microsoft experts realized early on that the header provided a convenient place (Welsh, 2004). Listing 2 shows the same SOAP message as in Listing 1, except this time security information is included in the header part of the message.

```
<soap:Envelope xmlns:soap=http://schemas.xmlsoap.org/soap/envelope/
     xmlns:wsse=”http://schemas.xmlsoap.org/ws/2002/07/secext”>
     <soap:Header>
          <wsse:Security soap:actor=”http://www.his.se/app/”
               soap:mustUnderstand=”1”>
               ...
          </wsse:Security>
     </soap:Header>
     <soap:Body>
          <GetBalance xmlns=http://www.bigbucksbank.com/bankapp”>
               <Account>8146-12334492</Account>
          </GetBalance>
     </soap:Body>
</soap:Envelope>
```

Security block

Listing 2. The SOAP message with an added security block in the header.

The security information is contained in the *Security* element, sometimes referred to as a *security block*. Two attributes of the security element in Listing 2 are marked in bold. The first attribute, the *actor* attribute, is used to describe which Web Service this particular security block is targeted at. This is necessary because a SOAP message can potentially be routed between several nodes (intermediaries) on its path between the requester and the provider. Consequentially, there can be several different security blocks present in the message targeted to different intermediaries. The actor attribute is not mandatory, but if it is left out, any of the intermediaries may process the block and no intermediary is allowed to remove it. If several security blocks are present in the message, only one block can omit the actor attribute and two security blocks cannot have actor attributes with the same value. Each security block may contain one or more tokens and each of the tokens in a specific security block need to be targeted at the same recipient.

The *mustUnderstand* attribute shows whether the recipient is required to process the security block. In Listing 2 the attribute is set to 1 which means that the recipient in this case have to process the security block and therefore also needs to understand how to process the data. If the recipient cannot process the block, the processing will fail.

3.1 Security tokens

The security blocks do not provide any security by themselves. This is where the security tokens enter the scene. Each security block may contain one or more tokens. The WS-Security specification (WS-Security, 2002) defines a *security token* as a collection of claims. A *claim*, in turn, is defined as a statement that a client makes concerning e.g. name, identity, key, capabilities, privileges or similar. Since a token can handle many different types of claims, tokens fill several purposes. A token can for example include information that can be used to perform authentication decisions or it can include a key to be used for encryption.

There are two basic types of tokens, pure text-based tokens or binary tokens. The user name token is an example of a text-based token and binary tokens may include X.509 certificates or Kerberos tickets which are both represented in binary formats. Listing 3 and 4 describe the syntax of the username token and the binary token in turn.

```
...
<wsse:Security soap:actor="http://www.his.se/app/"
     soap:mustUnderstand="1">
     <wsse:UsernameToken id="..."
          xmlns:wsu="http://schemas.xmlsoap.org/ws/
          /2002/07/utility">
          <wsse:Username>andto</wsse:Username>
          <wsse:Password type="wsse:PasswordDigest">
               FuGb...
          </wsse:Password>
          <wsse:Nonce>YwePq...</wsse:Nonce>
          <wsu:Created>2004-04-15T10:00:00</wsu:Created>
     <wsse:UsernameToken>
</wsse:Security>
...
```

Listing 3. The example shows a text-based token including a username and password which is placed in the security block.

Listing 3 shows the use of a username token in a SOAP message. The token is included in the security header block and includes the *Username* and *Password* element. The *id* attribute of the *UsernameToken* element is used to provide a string label for the token which is used to make references to it in other parts of the message. Examples of this use of labels will be shown later. The *type* attribute of the password element describes what type of password that is provided. There are currently two different possible types according to the WS-Security specification; PasswordText or PasswordDigest. In the first case the password is sent in clear text format and in the other case the password is sent as a Base64-encoded SHA1 hash value. The benefit of using the digest is that a one-way cryptographic function is applied to the password which creates a 160-bit digest that is sent with the message. This makes it unnecessary to send the password over the, possibly insecure, network but on the other hand the Base64-encoded password digest provides a fairly weak protection without complementary encryption of the message. A dictionary attack is one technique that could be applied in order to try to uncover the password. Base64-encoding provides no cryptographic protection but can be easily decoded which is why the WS-Security specification strongly recommends that simple username tokens with the password included is not used unless a secure transport is used. The *Created* element in Listing 3 contains a timestamp showing when the message was created. The timestamp is on the format yyyy-mm-

ddThh:mm:ss where T is constant, presumably standing for time and separating the date from the time.

In Listing 3, the *Nonce* element is used to provide a so called nonce, or "number once". A nonce is a value that is inserted into the message and which changes for every message that is sent. This ensures that no two valid messages are identical. If two messages are identical they need to be discarded on the assumption that one of the messages has been intercepted and used in a replay attack (O'Neil, 2003). Another example of a way to provide protection against replay attacks would be to include a timestamp in the message. Both a timestamp and a nonce need also be combined with a digital signature so that attempts at tampering with the timestamp or nonce can be detected. Otherwise, an attacker could change the timestamp or nonce value.

The WS-Security (2002) specification briefly mentions four different approaches for securing messages against replay attacks. Two of these have been mentioned above, although in the WS-Security specification the nonce element discussed above is called *Sequence Number* instead. The two other techniques mentioned are:

- Expirations – this approach requires that each message includes a time after which the message will no longer be processed by the recipient.
- Message Correlation – which means that the server includes information about the request in the reply so that the client can match the reply with the request it once made.

Listing 4 shows the basic structure of a *BinarySecurityToken* element used to include e.g. an X.509 certificate or a Kerberos ticket in a security block. Because attributes are not ordered in XML (and SOAP messages are expressed in XML) it does not matter in what order the attributes of an element are listed. In this case, the *id* attribute is defined first.

```
...
<wsse:Security soap:actor="http://www.his.se/app/"
     soap:mustUnderstand="1">
     <wsse:BinarySecurityToken id=".." encodingType=".." valueType="..">
          ...
     </wsse:BinarySecurityToken>
</wsse:Security>
...
```

Listing 4. The basic structure of a BinarySecurityToken.

The id attribute is used to reference this token from elsewhere in the SOAP message. An example of this use of the attribute will be shown later. The *encodingType* attribute in Listing 4 shows what encoding type is used for the binary data. According to the WS-Security (2002) specification, Base64 or Hex encoding are the current options. Finally, the *valueType* attribute defines what kind of value the token holds, e.g an X.509 certificate is specified using the attribute value *wsse:X509v3*. Other possible values are *wsse:Kerberosv5TGT* for a Kerberos ticket granting ticket or *wsse:Kerberosv5ST* for a Kerberos service ticket.

3.2 Signing and encrypting SOAP messages

The WS-Security standard specifies how techniques developed by the World Wide Web Consortium (W3C) to protect XML, such as XML Signature and XML Encryption, can be applied to SOAP messages to provide confidentiality, integrity and non-repudiation. A detailed description of XML Signature and XML Encryption is out of scope of this text so the interested reader is referred to O'Neil (2003) or the W3C homepage (www.w3c.org) to gain more information. A short overview of how WS-Security provides signature- and encryption possibilities will be provided however. Listing 5 shows an example of XML Signature applied to a SOAP message.

```
<soap:Envelope xmlns:soap=http://schemas.xmlsoap.org/soap/envelope/
     xmlns:wsse="http://schemas.xmlsoap.org/ws/2002/07/secext">
     <soap:Header>
          <wsse:Security soap:actor="http://www.his.se/app/"
               soap:mustUnderstand="1">
               <wsse:BinarySecurityToken id="X509Token"
                    encodingType="wsse:Base64Binary" valueType="wsse:X509v3">
                    DEcVrWPoicAqQkfdmGvBid1Z4+0XwlsFrEim9E3kT7y4...
               </wsse:BinarySecurityToken>
               <Signature xmlns="http://www.w3.org/2000/09/xmldsig#">
                    <SignedInfo>
                         <CanonicalizationMethod
                         algorithm="http://www.w3.org/2001/10/xml-exc-c14n#"/>
                         <SignatureMethod
                         algorithm="http://www.w3.org/2000/09/xmldsig#rsa-sha1"/>
                         <Reference URI="#bodyData">
                              <Transforms>
                                   <Transform
                                   algorithm="http://www.w3.org/2001/10/xml-exc-
                                   c14n#"/>
                              </Transforms>
                              <DigestMethod algorithm=
                              "http://www.w3.org/2000/09/xmldsig#sha1"/>
                              <DigestValue>
                                   CU2ynpWQFL9s4?+7dGnvZ3
                              </DigestValue>
                         </Reference>
                    </SignedInfo>
                    <SignatureValue>
                        0vWuyWMdfOpKSaZcvpLe496SwUi8+Ddfeb8...
                    </SignatureValue>
                    <KeyInfo>
                         <wsse:SecurityTokenReference>
                              <wsse:Reference URI="#X509Token"/>
                         </wsse:SecurityTokenReference>
                    </KeyInfo>
               </Signature>
          </wsse:Security>
     </soap:Header>
     <soap:Body>
          <GetBalance id="bodyData" xmlns=http://www.bigbucksbank.com/bankapp">
               <Account>8146-12334492</Account>
          </GetBalance>
     </soap:Body>
<soap:Envelope>
```

1

2

Listing 5. An example of a signed SOAP message.

As is obvious from Listing 5, this is where things start to get nasty, but we will try to explain the basics of the XML code without getting too involved in the nitty-gritty details. Arrow number one points to the *GetBalance* element in the body of the SOAP-message. In the code, the reference is done through the *#bodyData* value of the *URI* attribute in the *Reference* element that points to the corresponding value of the *id* attribute in the GetBalance element. A referring attribute is always immediately preceded by the #-sign.

Arrow one thus means that the GetBalance element of the body part is what is being signed. Therefore, if someone would try to manipulate the account number in order to get the balance for another account, this would be detected by the receiving part. The same signature may contain more than one reference so that different parts of the message can be signed. Looking at the signature element one can see that it contains information about the algorithms used, a digest value of the GetBalance element, a *KeyInfo* element and the value of the digital signature.

A digital signature works by first creating a digest of the data that is to be signed. As was mentioned earlier, a digest is a value created by applying a one-way cryptographic function to a portion of data. One-way means that it is fairly easy to compute the result of the function given some data, but that it is extremely complicated to reverse the computation and retrieve the original data. As an example, it is fairly easy to compute the function $y=x^3$ with a pen and paper, but a lot harder to compute the function $\sqrt[3]{y}$. The digest function has the property that if the message is changed the slightest bit and the function is rerun, it will result in a different digest value (Pfleeger & Pfleeger, 2003). The digest value is typically small (160-bits) and once it has been computed it is encrypted with the sender's private key. This is not the same way public key encryption is used to provide confidentiality, because anyone with access to the sender's public key can then encrypt the message. And the public key is provided for anyone that wants to access it in a public key certificate. The nice thing here is that only the holder of the private key can perform the encryption so that the corresponding public key can decrypt the message, thus we know that the digest really came from the sender defined in the certificate. This hinders an attacker from simply modifying the message, remove the old digest and append a new one. Once the receiver gets the message, he/she decrypts the digest value, computes a new digest from the data using the same algorithm and compares the two results. If the values match, the message has not been altered on the way and its integrity has been guaranteed.

In the case of Listing 5, the Reference element points to the data that has been signed. The Reference element also includes the digest of the data. The KeyInfo section of the Signature element, in turn, points to a binary security token that holds an X.509 certificate, i.e. a public key certificate that was used to encrypt the digest value and create the signature. The actual signature value is kept within the *SignatureValue* element.

In some cases, it may be more convenient to collect a public key certificate for validating the signature from an LDAP directory. The KeyInfo section then holds information about which certificate to retrieve and from where it should be retrieved. One reason for doing this is that currently there is no standardized way for two parties to exchange certificates and therefore the service consumer must already have acquired the provider's certificate prior to calling the service.

Although the actual body data of the message in Listing 5 is protected against tampering, it is not protected from prying eyes. The account number data can easily be read. This is where the encryption possibilities in WS-Security come into play to ensure message confidentiality. Listing 6 gives an example of an encrypted SOAP message.

```
<soap:Envelope xmlns:soap=http://schemas.xmlsoap.org/soap/envelope/
               xmlns:wsse=”http://schemas.xmlsoap.org/ws/2002/07/secext
               xmlns:ds="http://www.w3.org/2000/09/xmldsig#">
    <soap:Header>
        <wsse:Security>
            <xenc:EncryptedKey
                xmlns:enc="http://www.w3.org/2001/04/xmlenc#">
                <xenc:EncryptionMethod
                algorithm="http://www.w3.org/2001/04/xmlenc#rsa-1_5"/>
                <ds:KeyInfo>
                    <ds:KeyName>CN=Anders Toms O=His C=SE
                    </ds:KeyName>
                </ds:KeyInfo>
                <xenc:ChipherData>
                    <xenc:ChipherValue>iuwIUokjOIHsaYGt/6Fhj5...
                    </xenc:ChipherValue>
                </xenc:ChipherData>
                <xenc:ReferenceList>
                    <xenc:DataReference URI="#enc1"/>
                </xenc:ReferenceList>
            </xenc:EncryptedKey>
        </wsse:Security>
    </soap:Header>
    <soap:Body>
        <xenc:EncryptedData
            type="http://www.w3.org/2001/04/xmlenc#Element" id="enc1">
            <xenc:EncryptionMethod
            algorithm="http://www.w3.org/2001/04/xmlenc#3des-cbc"/>
            <xenc:ChipherData>
                <xenc:ChipherValue>OnyTgJERt7Zcbn5gh01Klbe&lpqf9...
                </xenc:ChipherValue>
            </xenc:ChipherData>
        </xenc:EncryptedData>
    </soap:Body>
<soap:Envelope>
```

Listing 6. An example of an encrypted SOAP message.

As can be seen in Listing 6, the body part of the message has now been encrypted and it is not longer possible to read out the account number from simply looking at the message, as was the case in Listing 5. The example in Listing 6 does not show all details of the encryption possibilities of WS-Security, but rather it shows some of the basic details. The *EncryptedKey* element is where the encrypted symmetric key that is used to encrypt the data is stored. The symmetric key is encrypted using the public key of the recipient. An element called *KeyInfo* is used to point out what public key has used been for this. The *ReferenceList* element contains references to those part of the SOAP message that has been encrypted with this key. In Listing 6, it points out the body

part of the message. Each encrypted element or element content in the original message need to be replaced by a corresponding *EncryptedData* element, as is shown in the body part of Listing 6.

Using WS-Security and XML Encryption together gives the possibility to encrypt selected parts of a message such as individual elements or element content that is particular sensitive. It is also possible to encrypt different parts of a message with different keys, so that one part of the message is directed to one recipient while another part is intended for another.

It is important to point out here that using the functions available in WS-Security for encryption at the application layer does not automatically make lower-layer encryption techniques obsolete. SSL can still be used in conjunction with XML Encryption leading to a "super encrypted" message.

3.3 WS-Security and SAML

Last but not least, a short discussion will be held in this section on how WS-Security relates to another security standard called SAML (Security Assertion Markup Language). SAML defines how trust can be established between different domains (e.g. two corporate computer networks) by using assertions expressed in XML. An assertion in this case can be defined as a claim, statement or declaration often concerning authentication or authorization decisions according to O'Neal (2003). Given that one domain trust the authority (the other domain) that vouches for the claim and the trustworthiness of the claim itself can be guaranteed, the claimed identity can be accepted as true. If something goes wrong, e.g. an unauthorized subject is permitted to access some resource, the other authority can be held responsible. WS-Security specifies how SAML assertions can be inserted into SOAP messages and processed in the same way as any other security tokens.

4 Benefits of WS-Security

So far, a fairly detailed description of how WS-Security includes security tokens into SOAP messages has been given and how XML Signature and XML Encryption can be used in conjunction with WS-Security. This enhancement of the SOAP protocol makes it possible to achieve such security requirements as message integrity, confidentiality and single message authentication (WS-Security, 2002). Requirements such as these are essential if Web Services are to be successfully used for applications accessible over public infrastructures and that require more security than no security at all. However, all the requirements described above can also be achieved by using existing, already well established, standards like SSL/TLS or VPN to protect Web Service deployments. In fact, most Web Service implementations in need of security that are up and running today rely on some of these technologies for protection. The question is, what benefits do the WS-Security standard provide that makes it an important choice for deploying secure Web Services?

One important benefit that the use of the WS-Security standard brings is that a message can be protected from end to end, all the way from the original consumer, across any potential intermediaries on the message's path, to the provider and back again. If e.g. SSL/TLS is used to protect a message and an intermediary would need to perform some processing of a message, this intermediary would need to decrypt the message, perform the necessary processing, encrypt the message again and pass it on to the next receiver. It is obvious that the message is vulnerable when it is decrypted. The threat involved in decrypting the message at a node may be due to a compromised or crooked intermediary.

The problem here is that techniques like SSL/TLS and VPN work at layers below the application layer in the Open Systems Interconnect (OSI) stack. SSL/TLS as an example works at the session layer of the OSI-stack shown in Table 1.

Layer Number	Layer Name	Web Services Technology
7	Application	HTTP, SMTP, SOAP
6	Presentation	Encrypted data, compressed data
5	Session	POP, SSL
4	Transport	TCP, UDP
3	Network	IP Packets
2	Data Link	PPP, 802.11 etc
1	Physical	ADSL, ATM etc

Table 1. The OSI-stack and Web Services technologies at each level (From O'Neil (2002)).

Data that is sent between two computers using the OSI-model is padded with information as it travels down the layers in the model and is unpadded at each layer in the receiving computer. This can be thought of as similar to how the famous Russian dolls (or matrioshka) are nested with a smaller doll inside each bigger doll. Continuing the analogy, if a doll somewhere in the middle would be locked with a padlock, it would be impossible to reach the innermost doll without first unlocking the padlock. In the case of Web Services over SSL/TLS, it is impossible to make sense of data at the layers above the session layer without first decrypting it at the session layer.

Perhaps the biggest benefit by using the XML Encryption functionality supported by WS-Security is that different parts of the message can be encrypted with different keys. This makes it possible to distinguish between parts of the message that should be accessible by different recipients.

WS-Security does not require a special transport layer technique to be used. It is not bound to only using HTTP for transport, although most often this is the transport protocol used today. Just as SOAP is independent of transport, so is WS-Security. It seems intuitive to use a security technique for SOAP that provides this property; otherwise some of the benefit of SOAP is lost. SSL on the other hand, is by far most commonly used in combination with HTTP as in HTTPS (although SSL can also be used to protect e.g. SMTP according to Garfinkel & Spafford (2002)).

Another benefit of using WS-Security and the fact that parts of a message can be selectively encrypted is that it can have an impact on performance, especially if large amount of data is sent because encryption is a resource intensive operation. This advantage is likely to become less significant over time when hardware accelerators for encryption are built into appliances and since hardware performance is constantly improving. Still, only the parts of the message that need protection need to be protected. Using SSL/TLS for protection is instead an all or nothing solution. A message encrypted or signed at the SOAP layer using WS-Security is also protected when it is stored on disk on a server (i.e. WS-Security provide *persistent security* as compared to only "*in-transit security*"). If e.g. a SOAP message is written to a database it is still protected, but SSL/TLS and VPN only provide protection when a message is in transit. A quotation from Schneier (2000, p.168) exemplifies this fact:

"Also, SSL does nothing to protect the data at the server. In early 2000, there were many cases of hackers breaking into Web sites and stealing information: credit card numbers, personal account information and more. SSL does nothing to prevent this."

To conclude the discussion related to SSL/TLS and VPN it can be stated that these techniques only provide point-to-point protection and not necessarily end-to-end. If routing is not a requirement, however, SSL/TLS and VPN may very well be a pragmatic solution.

A truly important benefit of WS-Security is that the specification provides a standard way to handle identity management spanning across multiple, heterogeneous systems in different organisations (Bloomberg, 2003a; Bloomberg, 2003b; Bloomberg & Schmelzer, 2004). In the same manner that Web Services provide an abstract layer for communication between systems implemented in different programming languages and running on different platforms, so does WS-Security provide an abstract layer for communicating security related data between systems that makes use of different security technologies. One system may e.g. use Kerberos for authentication while the other system uses public key certificates. WS-Security also provides for extensibility, meaning that it can be extended to include new technologies as they show up.

5 Disadvantages of WS-Security

One disadvantage of using WS-Security is that it can actually hamper performance if multiple messages are sent between nodes compared to if a lower level encryption technique like SSL is used (Heinonen & Sörensen, 2004). This is because for each encrypted message a new symmetric key is generated and used to encrypt the message and then this symmetric key is asymmetrically encrypted and included in the message. WS-Security also does not currently have any support for sessions spanning several messages and every message thus need to be evaluated separately with respect to authentication and authorization. Naturally, these procedures consume precious resources. (WS-Secure Conversation, which was described in section 2, is the later specification that will deal with these problems.) On the other hand, XML Encryption makes it possible to only encrypt a small part of a large message, so poor performance is not to be taken for granted. Another disadvantage that was mentioned earlier is that there is still no commonly agreed upon or standardized way to exchange certificates between service consumer and provider (Heinonen & Sörensen, 2004).

Lastly, the WS-Security standard is still new and, as always with new technology, problems are most likely to show up as time goes by. A much more mature standard like SSL or VPN has an immediate advantage on this point.

6 WS-Security product support

Currently there are several products on the market claim to have support for the WS-Security standard. However, as always, it is probably wise to take on a sceptical attitude towards what product vendors claim and what the products really live up to. As an example, a recent test of XML Gateway products (MacVittie & Forristal, 2003) revealed that several vendors claimed support for the WS-Security standard but still did not fully comply with it. The test included six different products from vendors like Verisign, Forum Systems and Reactivity.

XML Gateways are a good example of products for which the WS-Security standard is important. WS-Security makes it possible for an XML Gateway to restrict which clients can

access which services and individual methods for each service. A WS-Security Gateway can also provide data integrity and encryption but XML Gateway products are still evolving and are not a panacea to Web Services security. In their article, MacVittie & Forristal (2003) point out validation against an XML Schema as an example of a common function in XML Gateways that is still somewhat problematic.

One aspect that makes checks against a schema problematic is that the same code can be expressed in many different ways in XML. A schema for checking valid input also need to be quite detailed in order to be used for input validation. Simple pattern-matching against known attacks may turn out to be insufficient. MacVittie & Forristal (2003) describes a test in which an SQL-injection attack aimed at the stored procedure *xp_cmdshell* went through undetected simply by changing the name to *Xp_CmDsHeLl*.

Except for XML Gateways, support for many of the WS-Security details has already been included in a number of development tools. Examples of such tools include Systinet Web Applications and Services Platform (WASP[1]), Microsoft's Web Services Enhancements (WSE[2]) and Sun's Java Web Services Developer Pack (JWSDP[3]).

Concluding from the presentation of a couple of vendors that provide products they claim have support for the WS-Security standard, it seems as if the standard is gaining a fairly widespread support and acceptance. Also, given that the standard was voted an official standard by the OASIS members on April the 6th 2004 (www.oasis-open.org) this development seems likely to continue.

7 Summary

This report has provided an overview of the WS-Security standard which adds security functions such as confidentiality, integrity and non-repudiation to SOAP message headers. A brief insight has also been given to the details of the WS-Security syntax. Fortunately, most developers will be shielded from directly handling SOAP messages and security headers through tools like Microsoft's Web Services Enhancements (WSE). Section six provided a few examples of tools and products that claim to support the WS-Security standard. To sum things up, it can be stated that the WS-Security standard seems to have the potential to become an important tool to protect public Web Services in the future, especially once routing and intermediaries starts to become more commonly used in Web Service deployments.

[1] http://www.systinet.com/products/wasp_jserver/overview
[2] http://www.microsoft.com/sverige/pr/articles/Nyhetsnotis_661.asp
[3] http://java.sun.com/webservices/jwsdp/overview.html

8 References

8.1 Books

Chappell, D. (2002) *Understanding .NET – A Tutorial and Analysis.* US: Addison-Wesley. ISBN: 0-201-74162-8.

Connolly, T. & Begg, C. (2002) *Database Systems – A Practical Approach to Design, Implementation, and Management.* 3rd edition. US: Addison-Wesley. ISBN: 0-201-70857-4.

Garfinkel, S. & Spafford, G. (2002) *Web Security, Privacy & Commerce.* 2nd edition. US: O'Reilly & Associates. ISBN: 0-596-00045-6.

O'Neil, M. (2003) *Web Services Security*. US: McGraw-Hill. ISBN: 0-07-222471-1.

Pfleeger, C. P. & Pfleeger, S. L. (2003) *Security in computing.* 3rd edition. US: Prentice Hall, Pearson Education, Inc. ISBN: 0-13-035548-8.

Schneier, B. (2000) *Secrets & Lies, Digital Security in a Networked World.* US: Wiley Publishing Inc. ISBN: 0-471-45380-3.

8.2 Articles & Web References

Bloomberg, J. (2003a) Context & Identity, the Linchpins of Web Services Security. ZapThink White Paper. Available on the Internet (040210): www.zapthink.com

Bloomberg, J. (2003b) Building Security Into a Service Oriented Architecture. ZapThink White Paper. Available on the Internet (040210): www.zapthink.com

Bloomberg, J. & Schmelzer, R. (2004) A Guide to Securing XML and Web Services. ZapThink White Paper. Available on the Internet (040210): www.zapthink.com

Cambridge Advanced Learner's Dictionary. Available on the Internet (040331): http://dictionary.cambridge.org/

CBDI Forum (2004) Commentary, Web Service Standards Checkpoint. Wednesday 14th of April 2004. Available on the Internet (040415): http://www.cbdiforum.com/public/news/index.php3?id=1387

Heinonen, M. & Sörensen, C. M. (2004) Connecting Systems with Secure and Interoperable Web Services. Masters Thesis in Information Technology, Linköping Institute of Technology. Linköping, February 27, 2004.

IBM & Microsoft. (2002) Security in a Web Services World: A Proposed Architecture and Roadmap. Version 1.0. April 7, 2002. Available on the Internet (031207): http://www-106.ibm.com/developerworks/webservices/library/ws-secmap/

MacVittie, L. & Forristal, J. (2003) Enemy at the gateway. Network Computing, 031016. Available on the Internet (040413): http://img.cmpnet.com/nc/1421/graphics/1421f2_file.pdf

O'Neil, M. (2002) Is SSL Enough Protection for Web Services? EAI Journal, December 2002. Available on the Internet (040506): http://www.bijonline.com

Welsh, T. (2004) Web Services Security: The Problem. Web Services Strategies, Vol. III, No.1, January 2004.

WS-Security (2002). Specification, version 1.0. Available on the Internet (040415): http://schemas.xmlsoap.org/specs/ws-security/ws-security.htm