Implementing WS-Security in Perl

John Brooke, Mark Mc Keown, Stephen Pickles and **Stefan Zasada**[†]

Manchester Computing, University of Manchester, UK
[†]stefan@zasada.co.uk

Abstract

Computational Grids [4] allow heterogeneous, geographically dispersed resources to be harnessed together to provide huge computing power. Parallel to the development of Grids has been growth of web services and the concept of the Service Oriented Architecture [5]. Much current research is underway into building Service Oriented Grids. Fundamental to this is the Web Services Resource Framework (WSRF [1,25]), the logical convergence of Grid and web services.

Since WSRF is built on web services, underlying WS-Security [2] mechanisms can be used as the basis for secure communications on the Grid. The work presented here examines the implementation of WS-Security for WSRF::Lite [6], a lightweight Perl based Grid middleware.

1. Introduction

Message security and integrity is of fundamental concern in any communication network, and in particular distributed computer systems. One of the latest frameworks for building distributed systems is web services. Until recently web services have relied on their underlying transport protocols to provide secure communication, so called Transport Layer Security (TLS [7]). For example, SOAP [8] messages streamed over HTTP [9] can use SSL [7] to secure communication between the message startpoint and endpoint.

The nature of web services means that a SOAP message (an XML [10] document) may pass through many intermediary sites before it reaches its final destination. Any of these sites could be required to perform an arbitrary amount of processing on the message before it is sent on to its next destination. Using TLS prevents intermediary sites from reading the contents of a message, even if only a small part of the message is sensitive (and thus needs encrypting between its start and end points). By preventing intermediary sites from reading the SOAP messages passing through them it also means that any useful processing of the message that could have been performed by the intermediary site will instead have to be performed by the destination. A solution to this problem is Message Level Security (MLS) where the security is implemented at the SOAP message level.

Another useful feature of message level security is that messages can be stored and used to support repudiation [24]. A service can store a digitally signed message and use it as proof that the user actually sent the message.

2. WS-Security

WS-Security is a recently proposed standard for building message level security into SOAP messages. It allows sensitive parts of a SOAP message to be digitally signed and encrypted, while leaving the rest of the message as machine (and human) readable XML. WS-Security also allows for security tokens to be attached to a SOAP message (such as X.509 certificates [11]) which can be used for sender validation. WS-Security provides mechanisms to ensure message integrity and confidentiality, and is the base component of the WS-Security Stack (fig. 1) providing an extensive framework for building robust, secure applications using web services.

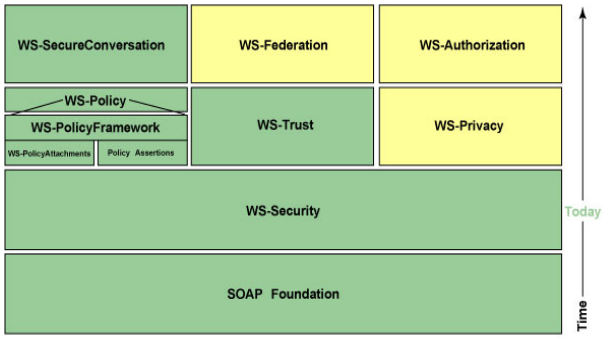

fig 1. The WS-Security Stack

Briefly, WS-Security specifies the process that is used to create a signed and encrypted SOAP message, along with a number of algorithms (transformation, hashing etc.) used in the process and the security tokens (X.509

certificates, Kerberos tickets etc.) used to validate the process. The specification supports signing and encryption using the XML Signature [12] and XML Encryption [13] standards.

By using WS-Security to add a signature to a SOAP message, and also a security token that can be used to verify the signature, the message recipient can determine if the message has been altered in transit and also that the message has come from whom it claims to have.

WS-Security provides a mechanism for encrypting parts of SOAP messages to prevent unauthorised parties from viewing the contents of the message. It uses a flexible framework that allows encryption of any combination of different blocks of the SOAP message, without the limitations of TLS security which leaves the whole SOAP message unreadable.

3. Perl for Web Services

Web services provide a comparatively light-weight way to build distributed systems. Perl is an ideal language with which to develop web services as a) it is highly modular and extendable and b) it is available on a great many different computer architectures, from desktops to supercomputers. The leading Perl web service hosting environment is SOAP::Lite [14], a very light-weight toolkit for building SOAP clients and servers. For example, a SOAP client can be built in as little as five lines of code. Currently SOAP::Lite only supports TLS and does not implement WS-Security at all.

One of the key groups interested in the development of web services has been the Grid community, due to the closely related goals of both web services and Grid computing – building highly flexible, reliable distributed systems from geographically disperse, heterogeneous computer resources spread across multiple organisations. The Open Grid Services Architecture (OGSA [15]) is a proposed framework for building Grid middlewares based on web services. Specifically it includes WSRF, an approach to modelling stateful Grid resources using web services.

WSRF::Lite is an attempt to build a WSRF compliant Grid middleware using Perl and SOAP::Lite. It has been used successfully in projects such as RealityGrid [3] (it is the basis for the RealityGrid middleware). For WSRF::Lite to be used to build fully functioning, interoperable Grid middleware it must be built on a SOAP toolkit that supports WS-Security. This project has looked at implementing WS-Security for SOAP::Lite.

4. WS-Security in Perl

When setting out to implement WS-Security it was decided to implement only a part of the specification, as time was limited (the project was conducted as part of an M.Sc. thesis). The part of the standard that was implemented was support for signing SOAP messages according to the OASIS Web Service Security X.509 Certificate Token Profile [16].

4.1 XML-Canonicalization

The XML Signature working group has defined two algorithms for canonicalizing XML: Inclusive XML Canonicalization (xml-C14N) [17] and Exclusive XML Canonicalization (xml-exc-C14N) [18].

Canonicalization of XML is important because the nature of XML means that a document can be changed in various ways and still be considered equivalent. This is a problem when signing SOAP messages as a receiver's hosting environment could alter a message as it is being processed and then be unable to verify the signature created across the specified part of the message. This becomes an even bigger problem when a message has to travel across numerous intermediate sites, each of which could potentially transform the message as it passes through. Although the message would be equivalent in XML terms, it would be impossible to verify the message signature.

Problems are also encountered if the namespace of an XML element is redefined. Canonicalization takes into account the namespaces of an XML block. The difference between Inclusive and Exclusive Canonicalization is that Inclusive Canonicalization places all namespace declarations currently in force into the canonicalized XML, even if they are outside the scope of the block being canonicalized. It also copies any XML: attributes that are in force and thus guarantees that all declarations made use of are unambiguously specified.

A problem arises if the signed XML is moved into another document with other declarations, even ones outside the canonicalized block, as the declarations will not be in the signature and thus the signature will not be validated. Exclusive Canonicalization tries to address this by only taking into account namespaces that are a part of the XML syntax. Using Exclusive Canonicalization will ensure that an XML block being signed is the same every time.

4.2 WS-Security Signature Algorithm

The WS-Security specification provides a well defined method for creating signatures for SOAP messages, using the algorithms and XML

elements described below to produce and attach the signature to the SOAP message envelope.

WS-Security can be used to sign multiple different parts (data objects) of a SOAP message. Steps 1-3 detailed below are performed for each data object in the message that needs to be signed.

1. For each data object being signed, apply the specified transform to this object (such as Exclusive Canonicalization).
2. Calculate the digest value over the transformed data object using the specified digest algorithm.
3. Create a <ds:Reference> element including: identification of the data object, <ds:Transform> elements identifying the transforms performed, <ds:DigestMethod> identifying the digest algorithm and the <ds:DigestValue> element containing the value of the digest calculated for the object.

Once steps 1-3 have been performed on each data object to be included in the signature, the main signature block is constructed.

4. Create <ds:SignedInfo> element with <ds:SignatureMethod> identifying the signature algorithm used, <ds:CanonicalizationMethod> identifying the canonicalization algorithm used and the <ds:Reference> elements, created previously.
5. Apply the canonicalization algorithm then calculate the <ds:SignatureValue> over the <ds:SignedInfo> block using the specified signature algorithm and sender's private key.
6. Construct <ds:Signature> block with <ds:SignedInfo>, <ds:KeyInfo> and <ds:SignatureValue>. When the signature block has been created it is added to the WS-Security block and then to the SOAP header.
7. The <ds:Signature> block is added to the <wsse:Security> header along with any <wsse:BinarySecurityToken> containing, for example, the senders X.509 certificate.
8. The <wsse:Security> block is added to the SOAP header block and the message is either sent or processed further.

4.3 Implementation

Initially when implementing WS-Security in Perl an analysis was made of the current tools available in Perl that could be used to implement the WS-Security algorithms. While many of the algorithms needed to implement WS-Security are available as Perl modules, it was found that support for XML-Canonicalization was inadequate, so the initial part of the implementation stage of the project focussed on producing a usable implementation of the two XML-Canonicalization algorithms. Perl support for Inclusive and Exclusive XML-Canonicalization was provided by producing a Perl wrapper around the XML-Canonicalization functions of the libxml2 [19] library.

The other algorithms required to produce signed SOAP messages (SHA1, RSA, BASE64) were already available as Perl modules, so the next step to implementing a SOAP signing mechanism in Perl was to implement the WS-Security signature algorithm. A SOAP message envelope is constructed as per the algorithm described in section 4.2. Specifically, in step 1 the data object is transformed with xml-exc-C14N. In step 2 the transformed data object is digested using SHA1. The step is repeated for each piece of XML that needs to be signed. The OASIS X.509 Certificate Token profile recommends that a timestamp and the XML containing the X.509 certificate that is used for the signing go through this process as well.

Xml-exc-C14N is used again to transform the SignedInfo block in step 5, after which the signature is created using RSA.

5. Integrating SOAP Security into WSRF::Lite

WS-Security support was added to WSRF::Lite by overriding the functions that serialize and de-serialize the XML from the SOAP::Lite toolkit. First the SOAP envelope is created, it is then passed through the canonicalization and digesting steps before the relevant elements are digitally signed by an X.509 certificate, the WS-Security Header element is created and the SOAP message is reconstructed. By default WSRF::Lite will try to sign the SOAP Body, the WS-Security timestamp, the X.509 certificate used to sign the message and any WS-Addressing [20] headers it finds in the message. If any of these elements are missing from the message then WSRF::Lite will simply ignore them. WS-Security is turned on by the setting of an environmental variable. The default elements of a message that are to be signed are declared in a Perl hash, a user can customise what elements of a message are to be signed by simply adding elements to or removing elements from this hash.

6. Conclusion

Adding WS-Security to the SOAP::Lite toolkit is a necessary step for making WSRF::Lite useful Grid middleware. The work presented here has provided a basis on which further functionality can be built. The signed SOAP messages produced by this implementation of WS-Security have been shown to conform to XML Signature Core Validation using tools

from the IBM XML Security Suite [21]. They have also been shown to interoperate with secure web services created with Microsoft's .NET framework [22] and with services developed with the Sun Java Web Service Developers Pack (JWSDP [23]). We consider this testing for interoperability to be a key element in the software development cycle. It is especially important that implementation of features concerned with security are tested in this manner.

7. Acknowledgements

Thanks to Savas Parastatidis for helping with interoperability testing between our implementation and .NET. Mark Mc Keown would also like to acknowledge the support of OMII [26] through the RAHWL [27] project.

8. References

[1] A. Nadalin, et al. Web Service Security: Soap Message Security 1.0. Technical report, OASIS, March 2004.

[2] K. Czajkowski, et al. "From Open Grid Services Infrastructure to WS Resource Framework: Refactoring and Evolution." Technical report, IBM, May 2004.

[3] RealityGrid, http://www.realitygrid.org

[4] I. Foster, C. Kesselman and S. Tuecke, "The Anatomy of the Grid" , http://www.globus.org/research/papers/anatomy.pdf, 2001.

[5] D. Booth et al. "Web Services Architecture", W3C Working Group Note 11 February 2004.

[6] WSRF::Lite, http://www.sve.man.ac.uk/Research/AtoZ/ILCT

[7] T. Dierks et al. "The TLS Protocol Version 1.0", IETF RFC 2246, 1999.

[8] M. Gudgin et al. "SOAP Version 1.2 Part 1: Messaging Framework", W3C Recommendation 24 June 2003.

[9] Fielding et al. "Hypertext Transfer Protocol – HTTP/1.1", IETF RFC 2616, 1999.

[10] T. Bray et al, "Extensible Markup Language (XML) 1.0 (Third Edition)", W3C Recommendation 04 February 2004.

[11] S. Chokhani and W. Ford, "Internet X.509 Public Key Infrastructure Certificate Policy and Certification Practices Framework", IETF RFC 2527, 1999.

[12] D. Eastlake et al., "XML-Signature Syntax and Processing", W3C Recommendation 12 February 2002.

[13] D. Eastlake et al. , "XML Encryption Syntax and Processing", W3C Recommendation 10 December 2002.

[14] SOAP::Lite, http://soaplite.com/.

[15] I. Foster et al., "The Open Grid Services Architecture, Version 1.0", GGF Technical Report, 2005.

[16] P. Hallam-Baker et al., "Web Services Security: X.509 Token Profile V1.0", OASIS Standard, 2004.

[17] J. Boyer, "Canonical XML Version 1.0", W3C Recommendation 15 March 2001.

[18] J. Boyer, "Exclusive XML Canonicalization Version 1.0", W3C Recommendation 18 July 2002.

[19] libxml2, http://xmlsoft.org/

[20] M. Gudgin et al., "Web Services Addressing 1.0 – Core", W3C Working Draft 31 March 2005.

[21] IBM XML Security Suite, http://www.alphaworks.ibm.com/tech/xmlsecuritysuite

[22] Microsoft's .NET Framework, http://www.microsoft.com/net/

[23] Sun Java Web Service Developers Pack, http://java.sun.com/webservices/jwsdp/index.jsp

[24] J. Schwartz et al. "Security Challenges, Threats and Countermeasures Version 1.0" Technical Resport, WS-I, May 2005

[25] M. McKeown and J.Vicary, "Build WS-Resources with WSRF::Lite", http://www.ibm.com/developerworks/edu/gr-dw-gr-wsrflite-i.html

[26] Open Middleware Infrastructure Institute, http://www.omii.ac.uk/

[27] Robust Application Hosting in WSRF::Lite, http://www.omii.ac.uk/mp/mp_wsrf_lite.htm