

Definition, Application, and Enforcement of WS-SecurityPolicies in Model-Driven SOAs

Meiko Jensen and Jörg Schwenk

Horst Görtz Institute for IT-Security,
Ruhr-University Bochum, Germany
{Meiko.Jensen|Joerg.Schwenk}@rub.de

Abstract. The approach of model-driven security enables highly promising capabilities regarding more reliable security implementations in Web-Services-based business processes. However, the complexity of both the WS-SecurityPolicy standard itself and the additional configuration needs of common WS-SecurityPolicy application and enforcement frameworks leads to a high number of configuration parameters. Hence, this paper illustrates the inevitable need to take the actual enforcement framework into account when setting up a WS-SecurityPolicy-enabled business process using model-driven techniques.

Key words: Model-Driven Security, WS-SecurityPolicy, Security Policy Deployment Descriptor, Web Services, SOA, BPM

1 Introduction

The standards of the Web Services technology [16] provide detailed descriptions for governing digital communication on a global level. Based on the core communication level specifications like XML (eXtensible Markup Language), SOAP, and WSDL (Web Services Description Language), a lot of additional, non-functional properties of Web Services communications have been put into standards of the WS-* family.

Among these, WS-Security [14] and WS-SecurityPolicy [12] are the major specifications for definition and enforcement of security-related properties in Web Services communication. Providing a large set of capabilities in terms of applying cryptography to SOAP messages, the WS-Security specification defines the syntax of security elements in SOAP messages, whereas the WS-SecurityPolicy specification enriches the common WSDL definitions with descriptions on what parts of a Web Service communication are to be protected using what cryptographic primitives (e.g. must be encrypted or digitally signed).

However, despite their standardization and broad set of capabilities, the overall use of these specifications still suffers some incisive issues. These are due to several reasons, with the most important ones among them being the high degree of complexity and missing experience in proper definition and configuration.

In this paper, we focus on the recently developed approach of using techniques from model-driven software development for applying security to Web-Services-based business processes (cf. e.g. [1, 10, 9, 4, 11]). More precisely, we investigate

the core requirements and conditions of properly enforcing WS-SecurityPolicies (generated by a model-driven security development toolset) for SOAP-based communication.

The paper is organized as follows. The next section gives a brief overview on recent approaches in model-driven security development. Section 3 presents a set of issues around the definition, application, enforcement, configuration, and management of WS-SecurityPolicies in service-oriented architectures (SOA), and the paper then concludes with future research directions.

2 Previous and Related Work

The overall idea of the model-driven software development approach consists in providing an abstract, typically graphical system model that can be designed by human developers using appropriate modeling tools (e.g. the ARIS SOA Architect [8]). Once the graphical model is specified, it is automatically transformed into its machine-readable pendant for execution, or it is taken as input for additional high-level verification and validation analyses (e.g. privacy audits). This approach is broadly used for the development of (SOA-based) business processes (see e.g. [15]), and was lately adapted for the task of embedding security mechanisms. Two examples of this approach are described next.

2.1 MDS4WS

Focussing on access control as the premier security aspect, the MDS4WS approach of Alam et al. [1–3] was the first approach to model-driven security for the Web Services domain. That approach addressed the model-based description and generation of XACML files [13] for defining the complex settings of a role-based access control enforcement framework. However, that approach did not address any means of applying security properties like confidentiality or integrity to SOAP messages.

2.2 ARIS Extension

The second approach, described in [10], focusses on proper usage of cryptographic primitives like encryption and digital signatures for the special case of Web-Services-based business processes. Hence, the outcome of the graphical security model transformation obviously consists in a set of WS-SecurityPolicy documents. Each of these corresponds to a certain WSDL description of the Web-Services-based business process itself (which is also generated automatically from the graphical business process model). However, though these WS-SecurityPolicy documents contain all basic information necessary for a proper application and enforcement of cryptography-based security in the business process communication, they still have to be applied to outgoing SOAP messages and enforced for incoming SOAP messages at runtime. Hence, it is necessary to have a closer look at how these enforcement modules work, how far they are

developed by today, and to what extent their special needs and capabilities must be considered in the security model and transformation process.

3 Security Modeling for SOAs

The specific characteristics and requirements of the common WS-SecurityPolicy application and enforcement implementations pose a set of troubles to all technology adopters that want to use WS-SecurityPolicy in a correct and reliable way within their applications and software products. Thus, it is necessary to investigate the general settings of WS-SecurityPolicy adaptation, the commonalities of the implementation requirements, and the derivations for a model-driven security development process for these.

3.1 WS-SecurityPolicy Usage Scenario

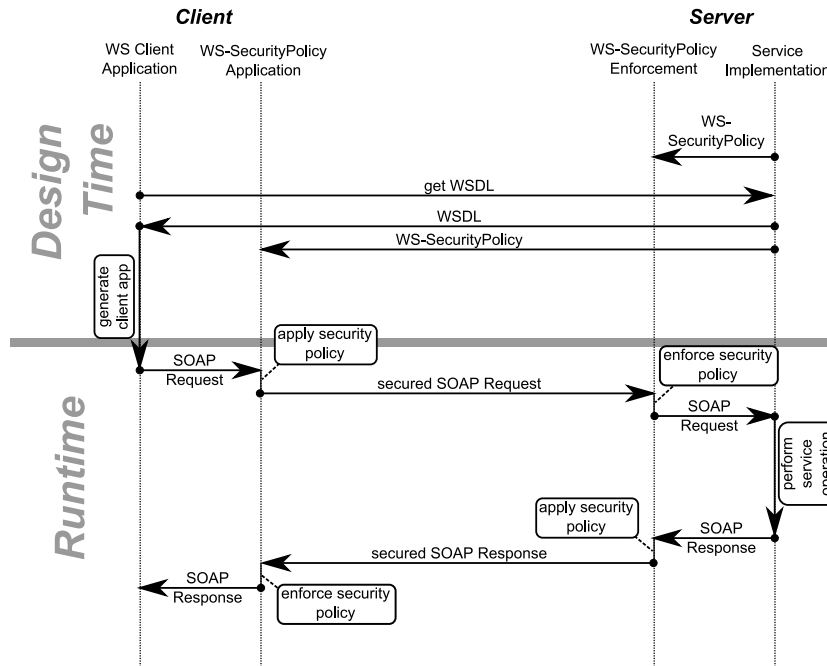


Fig. 1. WS-SecurityPolicy usage scenario

Figure 1 shows a common example of a WS-SecurityPolicy-enhanced Web Service communication. At design time, the client requests the WSDL document in order to determine the particular Web Service operation he intends to use. Along with that, the client additionally receives the WS-SecurityPolicy

document, determines the contained security assertions, and applies the specified encryptions, digital signatures, and other operations on its outgoing SOAP request message. Then, the request message is sent to the server, which immediately has to verify the proper and sufficient application of the cryptographic primitives, i.e. has to enforce the correct use of the WS-SecurityPolicy. This step is required, as a Web Service server cannot determine the origin of a SOAP request message immediately, hence has to protect itself from unsecured SOAP requests sent by malicious clients, e.g. on the Internet. Once the WS-SecurityPolicy enforcement is done, a valid SOAP request message is processed by the Web Service server. If the particular operation happens to have a result, an appropriate SOAP response message is created, which is to be sent back to the invoking client. Therefor it must first be processed according to the WS-SecurityPolicy for outgoing SOAP messages at the server side. Then, in the same manner as described before, the response message is validated for properly fulfilling the assertions of the WS-SecurityPolicy at the client side, before it is processed by the client side application logic.

3.2 Definition of WS-SecurityPolicies

One of the most challenging issues with the WS-SecurityPolicy-based security enforcement architecture consists in the proper definition of the WS-SecurityPolicy assertions. As such assertions can target all of a Web Service's endpoints, operations, or even single messages (e.g. requiring encryption just for the request, keeping the response unencrypted), the level of what a security assertion refers to is one of the most important characteristics of WS-SecurityPolicy design. If a Web Service endpoint has three operations, each consisting of request and response message, the same overall requirement (e.g. "encrypt the SOAP body") can be expressed in many different ways. It can be defined once only for the endpoint itself, thus automatically applying to all messages in all operations. Alternatively, it may be specified for each and every single message, here resulting in 6 single assertions of identical characteristics but different WSDL message references.

For the specific use of WS-SecurityPolicy as output to a security model transformation, it would be a valid approach to have all security assertions be attached to the particular SOAP messages only. The resulting WS-SecurityPolicy would cause the very same enforcement characteristics, but it would be much bigger and thus much more complex to apply and enforce correctly. Hence, it is necessary to post-process the outcome of such a security model transformation of a model-driven security tool in a way that best-possibly supports the runtime application and enforcement implementations.

3.3 Application and Enforcement

In the very same line as the design of a WS-SecurityPolicy document impacts its enforcement's complexity and thus performance, the application and enforcement implementations themselves are highly critical parts of the overall security

architecture. Though both application and enforcement components rely on the very same WS-SecurityPolicy document as input, their internal processes and requirements differ completely.

On the WS-SecurityPolicy *application* side, a given WS-SecurityPolicy document can easily be transformed into a normalized representation, then into an appropriate automaton for processing SOAP messages that pass by (cf. [7]). Hence, on security assertion application side it is sufficient to identify one valid configuration out of the set of configurations allowed by the WS-SecurityPolicy, and apply that configuration to all outgoing SOAP messages. Nevertheless, the choice of optimization criteria for determining the best among a set of allowed configurations is one example of a security policy application configuration parameter.

On the WS-SecurityPolicy *enforcement* side however, the implementation must be more flexible. Even though it most likely will be confronted with SOAP messages that originate from a few client implementations only, thus having the very same security policy application used repeatedly, the enforcement implementation nevertheless has to allow *all* configurations that are validly fulfilling the given WS-SecurityPolicy. Hence, the task of building an appropriate verification automaton is way more complex (cf. [6]). Additionally, reality has shown that commonly used WS-SecurityPolicy enforcement implementations pose some restrictions to incoming SOAP messages on their own, which in some cases even are not described in the WS-SecurityPolicy document nor elsewhere. These are only due to some developer's configurational choice, but are not proclaimed visibly to the Web Service clients. For example, the order of a digital signature header element and a timestamp header element within a SOAP header is not restricted by any XML Schema nor WSDL nor WS-SecurityPolicy documents. Nevertheless, for a long time the commonly used WS-Security framework called Apache Rampart [5] is likely to drop a SOAP request as faulty if that order is considered incorrect. Another example consists in that common .NET Web Services are not capable of receiving SOAP messages beyond a (configurable) message size (in bytes). Though this is a common type of restriction, the actual value of this size limitation again is not contained in any of the Web Service's descriptions.

Thus, in order to enable an optimal yet flexible WS-SecurityPolicy enforcement implementation, a considerable approach consists in using the very same implementation provider for both application and enforcement side, and deactivate other configurations if the scenario's settings do not require additional external clients (which is a very common setting for business process realizations). Otherwise, a more flexible enforcement implementation must be used, allowing more deviations in WS-SecurityPolicy fulfillment to the cost of higher computational needs. This again must be considered at the business process design level, hence it must be included in the high-level security model.

3.4 Configuration and Management

In order to express such implementation-specific parameters of WS-SecurityPolicy application and enforcement, each such implementation must be extended with an implementation-specific *security policy deployment descriptor (SPDD)* to mediate between the WS-SecurityPolicy specification itself and the corresponding implementations. This is a very common approach for setting up non-secured Web Services already; these implementations are always accompanied with a general deployment descriptor that contains all implementation-specific parameters that are not part of the Web Services descriptions themselves. Hence, the SPDD can be seen as an extension to the common deployment descriptor, just for expressing the specific parameters and configurations of the used WS-SecurityPolicy framework.

However, the approach of using an explicit SPDD is only useful if it can be created or processed automatically. The task of processing its contents is a framework-specific one, hence each framework has to implement an appropriate deployment configurator reading SPDDs on its own. The SPDD generation process, however, can be supported by the model-driven approach described earlier. For instance, the security modeling toolset can be extended with a choice for a certain target framework (e.g. Apache Rampart or .NET), and generate its appropriate SPDD along with the (generic) WS-SecurityPolicy document. Later on, if the framework changes (e.g. due to becoming a performance bottleneck), the security model can be adapted pretty easily: the WS-SecurityPolicies remain identical (as they only contain general information relevant to all existing frameworks), and only the SPDD must be re-generated for the new target framework.

Using such an approach of a general WS-SecurityPolicy document in conjunction with a framework-specific SPDD, the overall task of setting up a proper security policy for complex Web Service compositions can be supported by far, reducing the risk of accidental misconfigurations that cause vulnerabilities.

4 Conclusion and Future Work

Based upon previous work in the field of Model-Driven Security for Web-Service-based business processes, we have illustrated the issues of adapting the WS-SecurityPolicy specification in terms of definition, application, enforcement, configuration, and management. We have shown that the model-driven security approach can be used to support all of these issues, and we have stressed the need for an explicit security policy deployment descriptor that mediates between a general WS-SecurityPolicy document and its real-world enforcement framework.

Actual work in progress consists in defining an XML Schema for such an SPDD, taking some experimental SPDDs we manually crafted for widespread WS-SecurityPolicy frameworks as input. Once this prerequisite is addressed, the existing model-driven security toolset is to be extended with the SPDD capabilities.

References

1. M.M. Alam, Ruth Breu, and Michael Breu. Model driven security for Web services (MDS4WS). *Proceedings of INMIC 2004. 8th International Multitopic Conference.*, 2004.
2. Muhammad Alam. Model driven security engineering for the realization of dynamic security requirements in collaborative systems. In Thomas Kühne, editor, *Models in Software Engineering, Workshops and Symposia at MoDELS 2006, Genoa, Italy, October 1-6, 2006, Reports and Revised Selected Papers*, volume 4364 of *Lecture Notes in Computer Science*, pages 278–287. Springer, 2006.
3. Ruth Breu, Michael Hafner, Barbara Weber, and Andrea Novak. Model Driven Security for Inter-organizational Workflows in e-Government. *E-Government: Towards Electronic Democracy*, Volume 3416/2005:122–133, 2005.
4. Sven Feja, Ralph Herkenhöner, Meiko Jensen, Andreas Speck, Hermann de Meer, and Jörg Schwenk. Towards modeling and transformation of security requirements for service-oriented architectures. In *Proceedings of the 1st EuroNF Workshop on Future Internet Architecture (EuroNF-FIA), Paris, France, 2008*.
5. Ruchith Fernando. Secure Web Services with Apache Rampart. Technical report, WSO2 Oxygen Tank, 2006.
6. Nils Gruschka and Ralph Herkenhöner. WS-SecurityPolicy Decision and Enforcement for Web Service Firewalls. In *Proceedings of the IEEE/IST Workshop on Monitoring, Attack Detection and Mitigation*, 2006.
7. Nils Gruschka, Meiko Jensen, and Torben Dziuk. Event-Based Application of WS-SecurityPolicy on SOAP Messages. In *Proceedings of the 2007 ACM Workshop on Secure Web Services*, 2007.
8. IDS Scheer AG. IDS Scheer AG - Country Site DE: ARIS Software. http://www.ids-scheer.de/de/ARIS_Software_Software/7796.html. 2008-06-05.
9. Meiko Jensen and Sven Feja. *Model-Driven Integration Engineering*, chapter Modellierung von Sicherheitsaspekten in orchestrierten integrierten Anwendungssystemen, pages 301–311. Leipziger Beiträge zur Informatik, Band XI, 2008.
10. Meiko Jensen and Sven Feja. A security modeling approach for web-service-based business processes. In *Proceedings of the 7th IEEE Workshop on Model-Based Development for Computer-Based Systems*, 2009.
11. Jan Juerjens. *Secure Systems Development with UML*. SpringerVerlag, 2003.
12. Chris Kaler and Anthony Nadalin. Web Services Security Policy Language (WS-SecurityPolicy) 1.1. 2005.
13. Tim Moses. eXtensible Access Control Markup Language (XACML) Version 2.0. *OASIS Standard*, 2005.
14. Anthony Nadalin, Chris Kaler, Ronald Monzillo, and Phillip Hallam-Baker. Web Services Security: SOAP Message Security 1.1 (WS-Security 2004). *OASIS Standard*, 2006.
15. Sebastian Stein, Stefan Kühne, Jens Drawehn, Sven Feja, and Werner Rotzoll. Evaluation of OrViA Framework for Model-Driven SOA Implementations: An Industrial Case Study. In *6th International Conference on Business Process Management*, 2008.
16. Sanjiva Weerawarana, Francisco Curbera, Frank Leymann, Tony Storey, and Donald F. Ferguson. *Web Services Platform Architecture: SOAP, WSDL, WS-Policy, WS-Addressing, WS-BPEL, WS-Reliable Messaging, and More*. Prentice Hall PTR, 2005.