March 17, 2006

RT0646
Network; Security 8 pages

Research Report

Best Practice Patterns and Tool Support for Configuring Secure Web Services Messaging

Michiaki Tatsubori

IBM Research, Tokyo Research Laboratory
IBM Japan, Ltd.
1623-14 Shimotsuruma, Yamato
Kanagawa 242-8502, Japan

Best-Practice Patterns and Tool Support for Configuring Secure Web Services Messaging

Michiaki Tatsubori Takeshi Imamura Yuhichi Nakamura
IBM Tokyo Research Laboratory
tazbori@jp.ibm.com imamu@jp.ibm.com nakamury@jp.ibm.com

Abstract

This paper presents an emerging tool for security configuration of service-oriented architectures with Web Services. Security is a major concern when implementing mission-critical business transactions and such concern motivated the development of Web Services Security (WS-Security). However, the existing tools for configuring the security properties of Web Services give a technology-oriented view, and only assist in choosing the data to encrypt and selecting an encryption algorithm. The users must construct their own mental models of how the security configurations actually relate to business policies.

In contrast, the tool described here gives a simplified, business-policy-oriented view. It models the messaging with customers and business partners, lists various threats, and presents best-practice security patterns against the threats. A user can select among variations on the basic patterns according to the business policies, and then apply them to the messaging model through the GUI. The result of the pattern application is described in the Web Services Security Policy Language (WS-SecurityPolicy).

1. Introduction

Security is one of the major concerns when implementing mission-critical business transactions using Web Services. Since many software vendors have joined in Web Services initiatives such as standardization in W3C and OASIS, the concept of Web Services has been widely accepted over the past few years. With Web Services, applications can be coupled loosely—that is, in a decentralized manner—even beyond the enterprise boundaries. The concept is expected to influence business processes, where security is of critical importance.

There exist new security challenges with Web Services since Web Services allow for applications to interact with each other over the Internet. While most existing technologies are mainly concerned with how to protect applications within a security domain, we must here consider security among multiple security domains. The Web Services Security Model proposed in April 2002 [1] concerned federations among security domains, addressing interoperability among different security infrastructures such as Public Key Infrastructure (PKI) and Kerberos. Specifications proposed and expected to become standards include WS-Security [12], WS-SecureConversation [8], WS-Trust [7], and WS-Federation [10].

A problematic aspect of these specifications is their usability. Although the Web services security concept should provide a sophisticated basis to allow secure application integration even over the Internet, just the standardization and implementation of the concept do not contribute to usability enhancements enough. On the contrary, the usability may become worse since the specifications are growing rapidly so that they can cover variety of security models. One of the usability issues stems from requirements for detailed parameters such as cryptography algorithms and encryption keys, which are specific to particular security infrastructures. We believe most users want to focus on how security affects their business policies rather than worry about technological details. A suitable Web service security abstraction should serve as a bridge to the business level scope.

Leveraging the Web services security model, we have designed a tool to configure security policies. Our prototype tool generates WS-SecurityPolicy [6] descriptions that express security policies and requirements for a service. Since WS-SecurityPolicy is used by service requestors to publish their policies, some of the technical details such as the locations of keys are not included. The goal of our tool is to allow users to use their business scenarios to construct WS-SecurityPolicy descriptions.

In our approach, we prepare a collection of security best-practice patterns, and relate security policy fragment(s) to each of the patterns. Users first construct an application model that represents their business scenario,

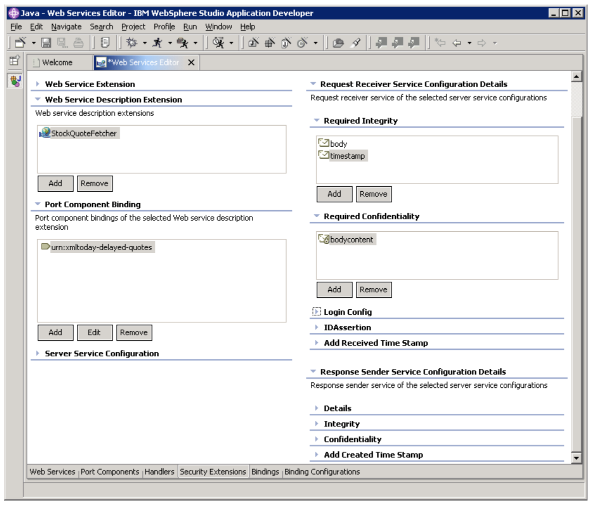

Java - Web Services Editor - IBM WebSphere Studio Application Developer
File Edit Navigate Search Project Profile Run Window Help
Welcome
*Web Services Editor
Web Service Extension
Web Service Description Extension
Web service description extensions
StockQuoteFetcher
Add
Remove
Port Component Binding
Port component bindings of the selected Web service description extension
urn:xmltoday-delayed-quotes
Add
Edit
Remove
Server Service Configuration
Request Receiver Service Configuration Details
Request receiver service of the selected server service configurations
Required Integrity
body
timestamp
Add
Remove
Required Confidentiality
bodycontent
Add
Remove
Login Config
IDAssertion
Add Received Time Stamp
Response Sender Service Configuration Details
Response sender service of the selected server service configurations
Details
Integrity
Confidentiality
Add Created Time Stamp
Web Services
Port Components
Handlers
Security Extensions
Bindings
Binding Configurations

Figure 1. The prototype GUI tool for configuring WS-Security. (Details are not important here.)

and then apply patterns to the model. Since each pattern has security policy fragment(s), the mapping contributes to the construction of a whole security policy. In addition, we introduce a platform model to represent the user's security environment. With the platform model, we can automatically fill out even some of the detailed parameters for the security policy.

The rest of this paper is organized as follows: Section 2 discusses usability issues in security configurations, and investigates how users want to think about security. In Section 3, we give an overview of our tool and introduce its constructs and user interface. Then we briefly review a catalog of patterns for securing Web Services, and show how each pattern addresses threats in Section 4. Section 5 discusses related work, comparing it to this tool. In Section 6, we conclude this paper.

2. Usability Issues and Our Approach

It is hard to properly set up a security configuration. This is especially true for Web services. Here, we discuss why it is hard in the context of Web services. Then we consider some requirements to improve usability.

2.1 Securing Web Services

With the Web services concept, applications can be coupled loosely, that is, in a decentralized manner beyond the enterprise boundary, typically over the Internet. Moreover, each business can have its own security infrastructure and mechanisms, such as Public Key Infrastructure (PKI) or Kerberos. Therefore, we need to interoperate these security systems over different security domains. The Web services security model defines an abstract model that allows federation of the security domains.

The Web services security roadmap document [1] describes not only the abstract security model, but also shows a collection of specifications to be published. If the specifications are properly defined, we can have in some sense complete security infrastructure. However, it would be still hard to configure security. Instead, it is still have !to find a way to leverage the abstract model for improving usability.

As an example, let us take a look at the WebSphere Application Development (WSAD) tool. Figure 1 shows the GUI used to configure WS-Security. We can specify which services are protected in the left pane, and how to protect them in the right pane. In addition to specifying which parts of the message require integrity and confidentiality, we have to specify detailed information such as cryptography algorithms and key locations.

Although the WSAD tool looks simple, users had better have a clear idea about the technology-level details. For example, they have to know if there is a PKI infrastructure upon which participants can establish a trust relationship regarding a certificate authority. Our thesis is that users want to think at the business level rather than at such technological levels.

We illustrate our hypothesis about how users want to think about security by using a sample scenario. Figure 2 shows a book order scenario where Alice orders books, providing her card information. In this scenario, we can imagine the following security requirements:

- Book retailer (Book) needs to authenticate Alice
- Book orders should not be repudiated
- Credit card info should be shown only to the credit card company (Card)

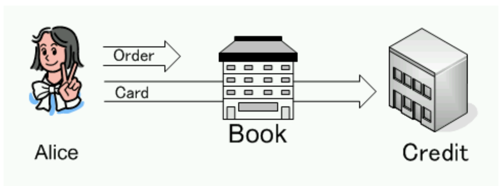


Figure 2. Book Order Scenario.

With existing tools like WSAD, users do not see a business scenario as in Figure 2, but only see a service API to add security features. Such tool support is not sufficient in the sense that users cannot think at business level scope.

2.2 Usability Requirements

We assume that the user's thinking involves the following steps:

```
<Policy>
<Integrity>
<TokenInfo>
<SecurityToken>
<TokenType> X509v3</TokenType>
<TokenIssuer>VeriSign</TokenIssuer>
</SecurityToken>
</TokenInfo>
<MessageParts>//OrderInfo</MessageParts>
</Integrity>
<Confidentiality>
<KeyInfo>
<SecurityToken>
<TokenType> X509v3</TokenType>
<TokenIssuer>VeriSign</TokenIssuer>
</SecurityToken>
</KeyInfo>
<Claims>
<SubjectName>Visa</SubjectName>
</Claims>
<MessageParts>//CardInfo</MessageParts>
</Confidentiality>
</Policy>
```

Figure 3. WS-SecurityPolicy Sample

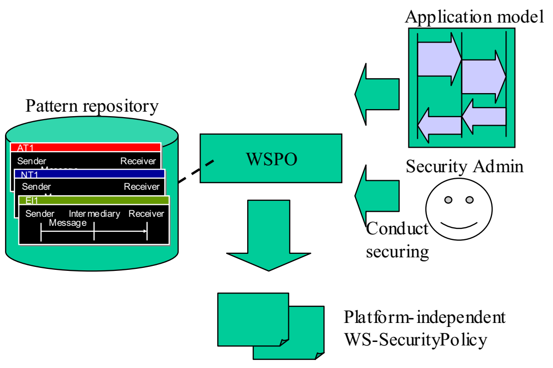


Figure 4. WSPO accepts an application model and generates an abstracted WS-SecurityPolicy description according to the users operations.

1. *Think about the application scenario.* She identifies participants such as Book and Credit and then envisions typical interactions among them. For example, Alice sends order and credit card information to Book.
2. *Hypothesize security policies for her business scenario at a higher abstraction level.* First she evaluates risks and threats and identifies the security requirements. For example, she needs to assess how many orders may be repudiated. Then she constructs an abstract security policy such that orders should not be repudiated
3. *Elaborate the security policy.* She makes the security policy concrete, referring to the partner's security policies and her company's security infrastructure.
4. *Configure the system.* She sets up detailed parameters for the security policies such as key locations, cryptography algorithms, and so on.

We think that the current tools mainly address Step 4, but the earlier steps are more crucial to the users. Therefore, we pay particular attention to support for Steps 1 to 3 in our tool.

In order to allow for users to think at the business level, our tool targets the Web Services Security Policy Language (WS-SecurityPolicy) [6] descriptions. With WS-SecurityPolicy, we can describe the security requirements to access Web services. Figure 3 shows a policy sample for the book order service. Order information requires integrity, and the credit card information requires confidentiality, respectively.

3. Tool Support for Security Configuration

We propose a tool filling the gap between business-level security policies and the concrete WS-Security policies implementing them. When implementing a secure Web Service system, the tool provides a GUI and helps a system administrator who understands the business-level security policies to configure the system so that its messaging operations are performed securely. The proposed tool is called the WS-Policy Organizer or WSPO.

WSPO works with two levels of security configuration: the abstract one and the concrete one. Figure 4 depicts the abstract-level security configuration with WSPO while Figure 5 depicts the concrete-level security configuration.

For the abstract configuration, as shown in Figure 4, WSPO first accepts an application model. Then it shows a GUI for configuring secure messaging on the given application model. According to the operations on the GUI, it generates an abstracted WS-SecurityPolicy description that is platform independent. WSPO uses best-practice patterns for securing Web Services messages in order to provide its users reasonably narrowed options for configuring the system's implementation-level security.

For the concrete configuration, as shown in Figure 5, WSPO also accepts a platform model. Then it shows a GUI for specifying the detailed parts of the WS-SecurityPolicy descriptions. The options shown in the GUI are reasonably narrow based on the specified platform model.

With WSPO, users process the configuration of their systems with secure messaging in the following manner:

1. An application developer constructs an application model that models the messages exchanged with the

participants, assuming the business application scenario of the configured system.

2. A security administrator applies the patterns provided in WSPO to each message in the application model. Users must consider their security requirements to choose the several appropriate patterns relevant to their system.
3. WSPO automatically generates formal security policy descriptions (WS-SecurityPolicy) based on the templates in the patterns chosen in Step 2.
4. A system deployer fills in the platform-dependent parts of the generated policies so that the policies can be deployed in the configured system platform. The details of the policies such as encryption algorithms and the locations of the keys are specified in this step.

Users of WSPO can configure their systems in a "top-down" manner. They can start by designing at the business level and then gradually step into more detailed parts of their systems.

3.1 Separation of application models and platform models

An important feature of WSPO is the separate description of an application model and a platform model. This separation allows users to use a GUI specialized for the given descriptions. The platform model of the system can be provided by the person who configured it. The application model can be provided by the application developer without concern about its message-level security. A system integrator (or application deployer) who knows the business-level security policies for the configured system can apply the policies to the application model using the tool.

An application model describes the business messaging models in an application scenario. Here is an example application model description:

```
<?xml version="1.0" encoding="UTF-8"?>
<ApplicationModel ..>
  <name>Bookstore Web Service Server</name>
  <self>uri:Bookstore</self>
  <Entities>
    <Entity name="Bookstore" id="uri:.."/>
    ...
  </Entities>
  <Messages>
    <Message name="Book order request"
id="uri:Bookstore#bookOrderRequest">
      <sender idref="uri:Alice"/>
      <receiver idref="uri:Bookstore"/>
      <MessageParts>
        <MessagePart name="User info."
path="//User" id="uri:Bookstore#user"/>
        ...
      </MessageParts>
    </Message>
    ...
  </Messages>
</ApplicationModel>
```

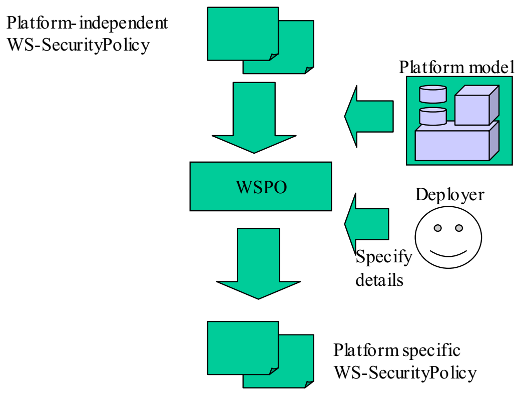

Platform-independent WS-SecurityPolicy
Platform model
WSPO
Deployer
Specify details
Platform specific WS-SecurityPolicy

Figure 5. WSPO accepts a platform model and generates concrete WS-SecurityPolicy description which includes platform-specific configurations.

The first <Message> element specifies the content carried in a request message for a service port. Each <MessagePart> element in the message elements or a message itself is a candidate to be secured.

Platform models describe the platform-specific environmental properties where applications are deployed. They are written in our Platform Description Language or PDL, which is a language defined for this tool. A PDL document contains the information that we used to specify with existing tools, as shown in Step 4 in Section 2.2. For instance, the key locations and the available cryptography algorithms are specified in a PDL document. Following is an example PDL description:

```
<?xml version="1.0" encoding="UTF-8"?>
<PlatformModel ..>
  <name>Bookstore Web Service Server</name>
  <Platforms>
    <Platform name="Bookstore Platform" entity-
IdRef="uri:Bookstore">
      <X509>
        <TrustedCerts>
          <Cert subject="CN=CA,O=VeriSign.."/>
        </TrustedCerts>
        ...
      </X509>
      ...
    </Platform>
    ...
  </Platforms>
</PlatformModel>
```

3.2 Patterns in the repository

The semi-automated generation of WS-SecurityPolicy description by WSPO is based on best-practice patterns for securing Web Service messages, which we have collected for this tool.

A pattern consists of:

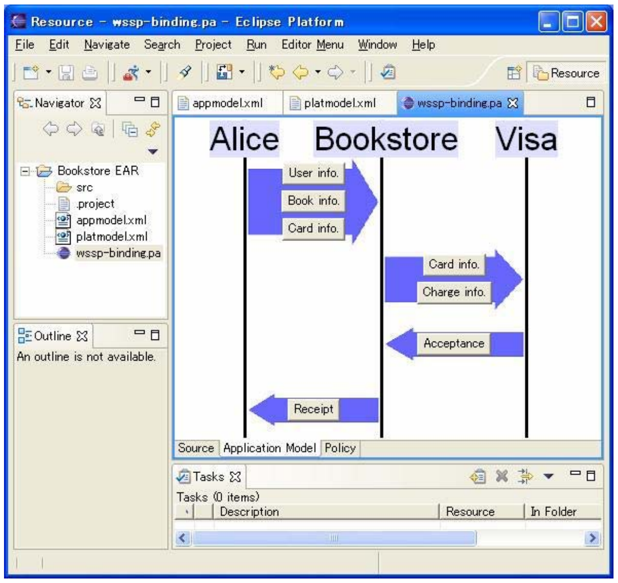


Figure 6. A GUI for an application model in WSPO.

- targeted attacks and threats to protect against and their descriptions,
- the business messaging model to be protected, and
- templates of security policies for implementing the protection.

The narrative descriptions of the best-practice patterns are available at [2]. Also, we will show a simplified version of the patterns in Section 4.

3.3 GUI operations

WSPO shows the view of a given application model that allows users to operate on the view in terms of the application model. The view contains:

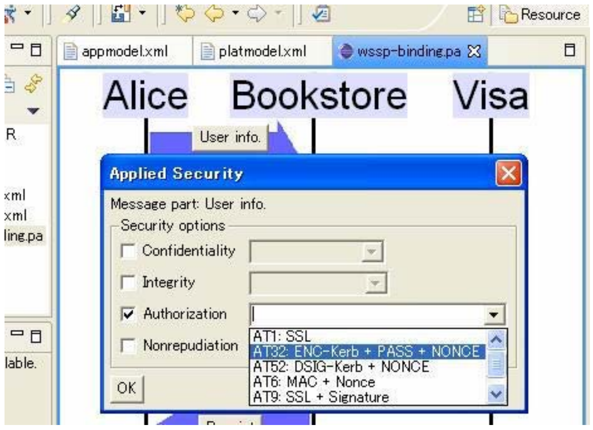
- entities (the company and its business partners), and
- messages sent between the company and its business partners.

Figure 6 shows a screen snapshot of a view for organizing WS-SecurityPolicy description (book-wsp.xml) from a WSDL description (book.wsdl). The system shows the application model represented by the WSDL description.

Through the GUI, a user can select each message to be secured as shown in Figure 7. The options available in the menu are:

- Make confidential
- Give integrity
- Authenticate, and
- Prevent repudiation

When a message is chosen to be secured in some way, WSPO shows the available options for implementing it. For instance, WSPO shows two implementation options for SSL (Protection by Lower Layer) and ENC (Encryp-



Message part: User info.
Security options
Confidentiality
Integrity
[✓ Authorization]
Nonrepudiation
AT1: SSL
AT32: ENC-Kerb + PASS + NONCE
AT52: DSIG-Kerb + NONCE
AT6: MAC + Nonce
AT9: SSL + Signature

Figure 7. A selection dialog in WSPO GUI.

tion for Receiver) when the credit card part of the orderRequest message is specified by a user as confidential. She can choose one of the implementation options or cancel the securing operation itself.

3.4 WS-SecurityPolicy Generation

WSPO generates a WS-Policy file for each message specified as secure. The generated policy for an orderRequest message in the application model given in Section 3.1, with an authentication option of the implementation variation AT3 (a combination of the PASS, NONCE, and ENC idioms) applied to the book-info part, would be as follows:

```
<Policy>
  <SecurityToken>
    <TokenType>UsernameToken</TokenType>
    <Claims>
      <UsePassword Usage="Required"/>
      <UseNonce Usage="Required"/>
    </Claims>
  </SecurityToken>
  <Confidentiality>
    <KeyInfo>
      <SecurityToken>
        <TokenType>${TOKEN_TYPE}</TokenType>
        <TokenIs-
suer>${TOKEN_ISSUER}</TokenIssuer>
        <Claims>
          <SubjectName>CN=Book</SubjectName>
        </Claims>
      </SecurityToken>
    </KeyInfo>
    <Mes-
sageParts>//UsernameToken</MessageParts>
  </Confidentiality>
</Policy>
```

4. Idioms and Best-Practice Patterns with WS-Security

The best-practice pattern language we developed for our tool has two levels of conceptual components. They are technical idioms and combinational patterns. First we present several idioms that abstract technologies extracted from Web Services specifications like WS-Security and WS-SecurityPolicy. Then, we show several patterns that clarify how we can combine these idioms to counter security threats. An early draft, but with more comprehensive descriptions of the patterns we developed, is available at [2].

4.1 Idioms in WS-Security

Before pigeonholing the best-practice patterns for securing Web Service messaging, it is better to abstract from the somewhat prosaic protocol specifications and define common notions for representing the functional features defined in the specifications. In this section, first we present several "idioms", which are used when implementing secure messaging with WS-Security. Each idiom is just a building block and does not provide a complete solution.

Protection by Lower Layer (SSL)

Protects a channel between a sender and a receiver by using security protocols provided by lower layers, such as SSL/TLS provided by the transport layer and IPSec provided by the network layer. The sender is either an initial sender or an intermediary. The receiver is either an intermediary or an ultimate receiver.

Encryption for Receiver (ENC)

A message is encrypted using a receiver's key. The receiver is the ultimate receiver of the content to be sent. The key may be obtained beforehand or when sending the message. If the key is a public key, it is obtained from a security token service (STS) using WS-Trust. If the key is a secret key, it is received from the receiver using WS-SecureConversation. The key-bearing message is encrypted using WS-Security.

Digital Signature by Sender (DSIG)

A digital signature is attached to a message using the sender's private key. The sender is either an initial sender or an intermediary. The digital signature is attached using WS-Security.

MAC by Sender (MAC)

Attaches a message authentication code (MAC) to a message using a sender's secret key. The sender is either an initial sender or an intermediary. The key needs to be shared with the receiver.

The key is shared between the sender and the receiver using WS-SecureConversation. The key may be exchanged beforehand or when sending the message. The MAC is attached using WS-Security.

Password of Sender (PASS)

An initial sender's username and its password are attached to a message. The pair consisting of a username and a password is attached using WS-Security.

Nonce (NONCE)

A nonce is attached to a message. The nonce may be attached to part of a message being processed, or if a digital signature or a MAC is attached, to a manifest of the digital signature or the MAC. The nonce can be attached using WS-Security if the data is only a username or a pair consisting of a username and password, or a timestamp can be used as the nonce.

4.2 A Best-Practice Patterns Catalog

We focused on four types of threats to Web Services: eavesdropping, falsification, masquerade, and repudiation. Each idiom presented in the previous section provides an implementation for protection from some of these threats. A single idiom may protect against a single threat or a combination of idioms may protect against several threats.

This section presents protection patterns where technical idioms are combined to counter certain threats. We use the idioms presented in the previous section to describe the implementations of the protections.

Confidential Message

Synopsis

Provide a confidential message.

Context

Threat: eavesdropping

Solution

Make it impossible for attackers to get or read any message content by encrypting it and transmitting an encrypted message instead of the original message.

Implementation Options

SSL (ET1) or ENC (ET2)

Message with Integrity

Synopsis

Provide a message with integrity.

Context

Threat: falsification

Solution

Make it impossible for attackers to get any messages, or make it possible for the receiver to detect any changes to the messages by attaching digital signatures to a message.

Implementation Options

SSL (ST1), DSIG (ST2) or MAC(ST3)

Authenticated Message Source

Synopsis

Authenticate the message source.

Context

Threat: masquerade

Table 1. Mapping threats to implementation variations

Threat	Variation	SSL	ENC	DSIG	MAC	PASS	NONC
Eavesdropping	ET1	X					
	ET2		X				
Falsification	ST1	X					
	ST2			X			
	ST3				X		
Masquerade	AT1	X				X	
	AT3		X			X	X
	AT5			X			X
	AT6				X		X
	AT9	X		X			
	AT10	X			X		
Repudiation	NT1			X			X
	NT2	X		X			

Solution

Perform authentication and make it impossible for attackers to get or reuse any authentication information.

Implementation Options

PASS + SSL (AT1), PASS + NONCE + ENC (AT3), DSIG + NONCE (AT5), MAC + NONCE (AT6), DSIG + SSL (AT9) or MAC + SSL (AT10)

Note that there are two ways to validate authentication information: by myself or by a proxy. In the latter case, validation is requested using WS-Trust. Validation may be done using WS-Federation as well. Validation of gateways can be regarded as a variation of this case.

Non-Repudiated Message

Synopsis

Provide a message that cannot be repudiated.

Context

Threat: repudiation

Solution

Add versions for every message to be sent and attach digital signatures to messages using a private key.

Implementation Options

DSIG + NONCE (NT1) or DSIG + SSL (NT2)

4.3 Patterns against Threats

Table 1 shows the mapping between threats and implementation variations. From the table, we can see what variation to use to counter each threat. We can also see what variations to combine to counter some of the threats efficiently. This table is used to find and analyze combinations of variations that meet the security requirements.

For example, suppose we want to counter all the threats from both third parties and intermediaries. Using the NT1 variation, we can counter repudiation as well as falsification and masquerade, because that variation is constructed from the DSIG idiom and the NONCE idiom, and from those idioms, the ST2 variation and the AT5 variation are constructed. Therefore, using the NT1 variation in combination with the ET2 variation, we can counter all of the threats.

5. Discussion

5.1 Related Work

Yoder et al. proposed patterns for an architecture for making an application secure [15]. By combining some of the patterns and implementing the combination, a secure application can be built. Yoder et al. focused on the architecture of an application and classified its security aspects, which is different from what we focus on, messaging between applications. Their patterns are orthogonal to ours and would be useful for building an application.

Braga et al. proposed patterns for messaging between applications [16]. Our situation calls for patterns of this sort. However, each pattern is too abstract to use in tools proposed in this paper. Also, it is assumed that communication is directly between sender and receiver, and is protected using application layer security mechanisms. We should consider that one or more intermediaries may exist between sender and receiver in the context of Web Services. Moreover, Braga et al. targeted the patterns only while we target policies and configurations as well.

The Basic Security Profile Working Group of WS-I studied the security of Web Service scenarios [17]. Though they targeted the same area as Braga et al. did, they studied it in more detail, especially considering intermediaries and transport-layer security mechanisms. Their work does not consider the possibility that some of the intermediaries are malicious and it does not well address how to combine application and transport layer security mechanisms. However, it would be useful for refining some part of our patterns.

5.2 Limitations of Our Approach

Here, we discuss some limitations of our tool, envisioning how to enhance it. First, the actual descriptions of WS-SecurityPolicy are not abstract enough to eliminate technical details completely. As in Figure 1, two levels are explicitly distinguished in IBM WebSphere: the service and binding levels. As for signature, which parts require integrity should be defined at the service level, and signature algorithms and key types should be defined later at the binding level. On the other hand, these different levels of information are mixed in the WS-SecurityPolicy descriptions. It seems possible to define a classification hierarchy of policy descriptions on the basis of the key elements of WS-SecurityPolicy such as Integrity and Confidentiality.

Second, the federation of security domains should be taken into consideration for security configurations. Although WS-SecurityPolicy can support federation, our pattern representation and platform model do not take it into account. Since it is important for Web service security, we have to enhance our models as soon as possible.

Finally, our tool has not been integrated with an actual development process. For now, we assume that users construct application models from scratch, referring to WSDL. However, the application development trend is for the Model Driven Architecture (MDA). In MDA, users construct an abstract model for the business process, and refine the model repeatedly. This suggests that we want to reuse a model at some abstraction level when creating the application model. In addition, our tool would be used during the model construction process. Actually, the Business Process Modeling Language (BPML) [20] defines a business process notation which is quite similar to our application model. In this way, we want to seek how to leverage existing MDA efforts for our tool.

6. Concluding Remarks

This paper proposed a tool giving a simplified, business-policy-oriented view to its users, who are configuring secure Web Services for their systems. It models the messaging with customers and business partners, lists various threats to the messaging, and offers best-practice patterns for the threats. A user can select among variations on the basic patterns according to the business policies, and then apply them to the messaging model using the GUI. The process of configuration with the tool is in a "business-friendly" manner. That is to say they can start with designing at the business level and then gradually step into more detailed parts of their systems.

Acknowledgment

We thank Tim Ebringer and Shannon Jacobs for their advice on this work and English proofreading. We also thank anonymous reviewers of this paper for their valuable comments, and they are reflected on the final version of this paper.

References

[1] Security in a Web Services World: A Proposed Architecture and Roadmap, Apr 7, 2002.

[2] Takeshi Imamura and Michiaki Tatsubori: Patterns for Securing Web Services Messaging, *OOPSLA 2003 Workshop on Web Services and Service Oriented Architecture Best Practice and Patterns*, Anaheim, California, USA, November 26-31, 2003.

[3] Extensible Markup Language (XML) 1.0, W3C Recommendation, Oct 6, 2000. http://www.w3.org/TR/REC-xml

[4] SOAP Version 1.2, W3C Recommendation, June, 2003. http://www.w3.org/TR/soap12

[5] Web Services Policy Framework (WS-Policy), May 28, 2003. http://www-106.ibm.com/developerworks/library/ws-polfram

[6] Web Services Security Policy Language (WS-SecurityPolicy), Dec 18, 2002. http://www-106.ibm.com/developerworks/library/ws-secpol/

[7] Web Services Trust Language, Draft, Dec 18, 2002. http://www-106.ibm.com/developerworks/library/ws-trust

[8] Web Services Secure Conversation, Draft, Dec 18, 2002. http://www-106.ibm.com/developerworks/library/ws-secon

[9] Web Services Description Language (WSDL) 1.1, W3C Recommendation, Mar 15, 2001. http://www.w3.org/TR/wsdl,

[10] Web Services Federation Language, Jul 8, 2003. http://www-106.ibm.com/developerworks/webservices/library/ws-fed

[11] Bruce Schneier, *Secrets and Lies: Digital Security in a Networked World*, John Wiley & Sons, 2001.

[12] Web Services Security: SOAP Message Security, OASIS WSS TC Working Draft, Aug 27, 2003.

[13] Web Services Security: UsernameToken Profile, OASIS WSS TC Working Draft, Aug 11, 2003.

[14] Web Services Security: X.509 Certificate Token Profile, OASIS WSS TC Working Draft, Aug 19, 2003.

[15] J. Yoder and J. Barcalow. Architectural Patterns for Enabling Application Security, *PLoP '97*.

[16] A. Braga, C. Rubira, and R. Dahab. *Tropyc*: A Pattern Language for Cryptographic Software, *PLoP '98*.

[17] WS-I Security Scenarios, WS-I Basic Security Profile Working Group Draft, Feb. 14, 2004.

[18] Frankel, D: *Model Driven Architecture*, Addison-Wesley (2003).

[19] Baker, J. and Ghalimi, I: *BPML 101 Implementing the BPML Specification* (2001).

[20] Business Process Modeling Language (BPML), Business Process Management Initiative, Mar 8, 2001, http://www.bpmi.org/bpml.esp