March 2, 2007

RT0714

Computer Science; Engineering Technology; Security 8 pages

Research Report

Verifying the Consistency of Security Policies by Abstracting into Security Types

Kouichi Ono, Yuichi Nakamura, Fumiko Satoh, Takaaki Tateishi

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

Verifying the Consistency of Security Policies by Abstracting into Security Types

Kouichi Ono   Yuichi Nakamura   Fumiko Satoh   Takaaki Tateishi
Tokyo Research Laboratory, IBM Research
LAB-S77 1623-14 Shimotsuruma, Yamato-shi, Kanagawa, 242-8502 Japan
{onono,nakamury,sfumiko,tate}@jp.ibm.com

Abstract

——- *The Service-Oriented Architecture (SOA) makes application development easier, because applications can be built from existing services with a bottom-up methodology. However, it is difficult to determine if a desired new service can be built from existing services. Not only the functional consistency of the existing services, but also the consistency of their non-functional (such as security) aspects must be verified. Message protection is an aspect of security. Every service needs an appropriate security policy defining the protection of messages exchanged between the parties to the service. Because of the intricacy of the Web Services Security Policy Language, it is difficult to verify the consistency of the security policies.*

We are developing a method to verify the consistency of security policies by abstracting them. Each security policy is abstracted, and then attached as a security type to the corresponding service in the application model. The security type denotes a security level for message protection. The security developer defines the possible abstraction methods. In this paper, we define the constraint of abstraction methods based on the semantics of the policy language. And also we state verifying the consistemcy of security types by using information flow analysis.

1. Introduction

Many enterprises are developing with the Service Oriented Architecture (SOA) [1] because their business models are changing more frequently. In SOA, services are self-described and can be accessed without regard to the underlying IT infrastructure. Such technology independence allows services to be more easily used for building business processes. Thus, SOA is now widely recognized as having the potential to radically improve business transformations.

SOA makes application development easier because technology-independent services can be coupled over intranets and via the Internet. Meanwhile, the underlying computing environments on which applications are running are becoming more complex, because computers can be networked in complicated topologies, including firewalls and intermediate servers. Therefore, the configuration of non-functional aspects such as security requires a fairly deep understanding of such complex environments.

We believe that security must be unified with the software engineering process from the beginning, and thus security engineering [2][3] is important. Unfortunately, security is considered as an afterthought in most actual development projects, in the sense that security is added after the functional requirements are implemented. It is well known that finding defects downstream greatly increases the costs of removal and repair.

It is difficult to determine if a desired new service can be built from existing services on the underlying computing environments by using a bottom-up methodology, e.g. by Component Business Modeling (CBM) [4]. Not only the functional consistency of the existing services, but also the consistency of their non-functional aspects such as security must be verified. Message protection is an aspect of security. Every service needs an appropriate configuration as a security policy defining the protection of messages exchanged between the parties to the service. Since the Web Services Security Policy Language (WS-SecurityPolicy) [5] is intricate, it is difficult to verify the consistency of the security policies.

In this paper, we propose a verification method for the consistency of security policies by abstracting them. The security developer defines the possible abstraction methods, which are constrained based on the semantics of the policy language. Each security policy is abstracted with the defined abstraction method. Then it is attached as a security qualifier to the corresponding service in the application model. The security qualifier denotes a security level for message protection. If the process flow of the application is given, then the consistency of the security qualifiers can be verified by using information flow analysis.

The rest of this paper is organized as follows: Section 2 describes how to incorporate security into the SOA. In section 3, we describe a verification method for security policies by abstracting them. Section 4 discusses related work and some open issues in our own work. In Section 5, we conclude this paper.

2. SOA and Security

2.1. Service-Oriented Architecture

Applications can be developed based on SOA by using a top-down methodology or a bottom-up methodology as is used here. CBM is the bottom-up methodology we chose, and applications will be built from the existing services by using CBM. In this paper, an example of an application which is built from existing services is used. Fig. 1 shows the example. The example is the Supply Chain Management Application defined by the Web Services Interoperability Organization (WS-I) [6]. The SCM application consists of three sub-systems: Demo System, Retailer System, and Manufacturing System. The application is built from a number of existing services: Logging Service, Retailer Service, Warehouse Service, Warehouse Callback Service, and Manufacturing Service.

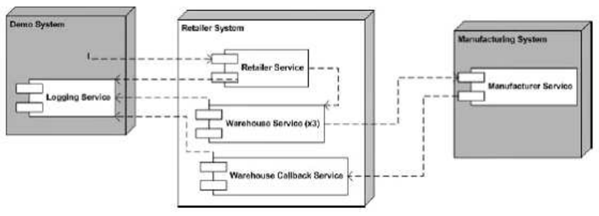
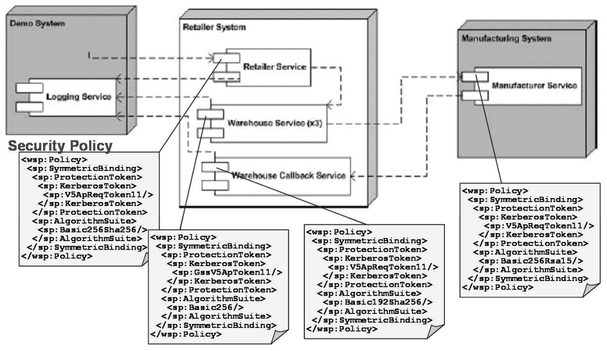


Figure 1. WS-I Supply Chain Management

2.2. Security Policy

From the perspective of security, every service is supposed to have a security policy which represents the policy of the security required for the service. This paper focuses on message protection as one of the aspects of security, and it is accepted that the security policy specified in WS-SecurityPolicy is assigned to every service as in Fig. 2. The security policy defines a policy for how incoming messages to the service must be protected. The message protection mainly works using encryption technologies and digital signature technologies. The encryption technologies are related to the confidentiality of the messages, and the digital signature technologies are related to the integrity of the messages.



```
<wsp:Policy>
 <sp:SymmetricBinding>
  <sp:ProtectionToken>
   <sp:KerberosToken>
    <sp:V5ApReqToken11/>
   </sp:KerberosToken>
  </sp:ProtectionToken>
  <sp:AlgorithmSuite>
   <sp:Basic256Sha256/>
  </sp:AlgorithmSuite>
 </sp:SymmetricBinding>
</wsp:Policy>
```

```
<wsp:Policy>
 <sp:SymmetricBinding>
  <sp:ProtectionToken>
   <sp:KerberosToken>
    <sp:GssV5ApToken11/>
   </sp:KerberosToken>
  </sp:ProtectionToken>
  <sp:AlgorithmSuite>
   <sp:Basic256/>
  </sp:AlgorithmSuite>
 </sp:SymmetricBinding>
</wsp:Policy>
```

```
<wsp:Policy>
 <sp:SymmetricBinding>
  <sp:ProtectionToken>
   <sp:KerberosToken>
    <sp:V5ApReqToken11/>
   </sp:KerberosToken>
  </sp:ProtectionToken>
  <sp:AlgorithmSuite>
   <sp:Basic192Sha256/>
  </sp:AlgorithmSuite>
 </sp:SymmetricBinding>
</wsp:Policy>
```

```
<wsp:Policy>
 <sp:SymmetricBinding>
  <sp:ProtectionToken>
   <sp:KerberosToken>
    <sp:V5ApReqToken11/>
   </sp:KerberosToken>
  </sp:ProtectionToken>
  <sp:AlgorithmSuite>
   <sp:Basic256Rsa15/>
  </sp:AlgorithmSuite>
 </sp:SymmetricBinding>
</wsp:Policy>
```

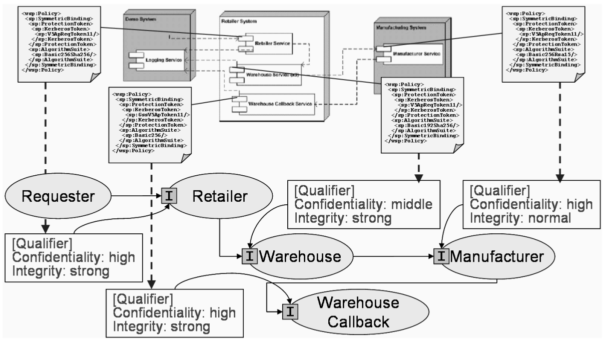
Figure 2. WS-I Supply Chain Management and Security Policies

When developing an application from the existing services with a bottom-up methodology like CBM, every service has its own security policy prior to the service composition. The service composition also defines the flows regarding what data and how the data of the messages is passed from one service to another service. Therefore, the service composition will need to verify the consistency of the security policy for message protection. For example, in the WS-I SCM application, suppose that the encryption of the order information with a strong cipher is specified in the security policy of the Retailer Service, and the encryption of the order information with a weak cipher is specified in the security policy of the Warehouse Service. The order information given by requester must be protected with high confidentiality, because the security policy of the Retailer Service calls for encryption with a strong cipher. However, that order information will be passed from the Retailer Service to the Warehouse Service while protecting it with low confidentiality because the security policy of the Warehouse Service calls for encryption with a weak cipher. For development with a bottom-up methodology like CBM, each service was defined as a service component in advance, and its security policy was also defined prior to the service composition. That means it is possible that the security policies of some of the service components will be inconsistent when an application is built from the service components.

3. Verifying the Consistency of Security Policies

3.1. Abstraction Method

In this section, we propose a method to verify the consistency of security policies. The basic idea is that each security policy is translated into the corresponding security qualifier and the consistency of the qualifiers is verified with a process flow. A security qualifier is a sort of security type, and consists of a number of security levels. In this paper, each security qualifier consists of a confidentiality level and an integrity level. Each set of security levels is totally ordered (or a lattice), and therefore, the set of security qualifiers is a lattice. The translation from a security policy into a security qualifier is called "the abstraction of a security policy" in this paper. Fig. 3 shows the abstraction of security policies of Fig. 2. In Fig. 3, "high", "middle", "strong", and "normal" represent security levels. In addition, a mapping from security policies to security qualifiers is called "an abstraction method for security policies".



```
<wsp:Policy>
 <sp:SymmetricBinding>
  <sp:ProtectionToken>
   <sp:KerberosToken>
    <sp:V5ApReqToken11/>
   </sp:KerberosToken>
  </sp:ProtectionToken>
  <sp:AlgorithmSuite>
   <sp:Basic256Sha256/>
  </sp:AlgorithmSuite>
 </sp:SymmetricBinding>
</wsp:Policy>
```

```
<wsp:Policy>
 <sp:SymmetricBinding>
  <sp:ProtectionToken>
   <sp:KerberosToken>
    <sp:GssV5ApToken11/>
   </sp:KerberosToken>
  </sp:ProtectionToken>
  <sp:AlgorithmSuite>
   <sp:Basic256/>
  </sp:AlgorithmSuite>
 </sp:SymmetricBinding>
</wsp:Policy>
```

```
<wsp:Policy>
 <sp:SymmetricBinding>
  <sp:ProtectionToken>
   <sp:KerberosToken>
    <sp:V5ApReqToken11/>
   </sp:KerberosToken>
  </sp:ProtectionToken>
  <sp:AlgorithmSuite>
   <sp:Basic192Sha256/>
  </sp:AlgorithmSuite>
 </sp:SymmetricBinding>
</wsp:Policy>
```

```
<wsp:Policy>
 <sp:SymmetricBinding>
  <sp:ProtectionToken>
   <sp:KerberosToken>
    <sp:V5ApReqToken11/>
   </sp:KerberosToken>
  </sp:ProtectionToken>
  <sp:AlgorithmSuite>
   <sp:Basic256Rsa15/>
  </sp:AlgorithmSuite>
 </sp:SymmetricBinding>
</wsp:Policy>
```

Figure 3. Abstraction of Security Policies

The consistency of the security qualifiers is verified with process flows by using an information flow analysis technique [7][8]. In this paper, we specify process flows in Web Services Business Process Execution Language [9][10] (BPEL for short). Parameter variables in any operation that the process flow describes are typed with a security qualifier, and other variables which are used for invocations of outside services are also typed with the security qualifiers of the outside services. The information flow of the process flow is analyzed statically by using type inference of the variables typed with the security qualifiers.

Fig. 4 shows the systems structure of our method. An abstraction method has to be defined by the security developer prior to the verification. Defining abstraction methods is subject to several constraints based on the semantics of WS-SecurityPolicy. The constraints are discussed in the following section.

3.2. Typing Rules

A security type system is defined as a set of typing rules. A BPEL description (or its subexpressions) will be typed with a security type by the typing rules. It is supposed that exp stands for an expression, v stands for a variable, and A stands for an activity or a sequence of acitivities. $Vars(exp)$ denotes a set of variables used in exp,

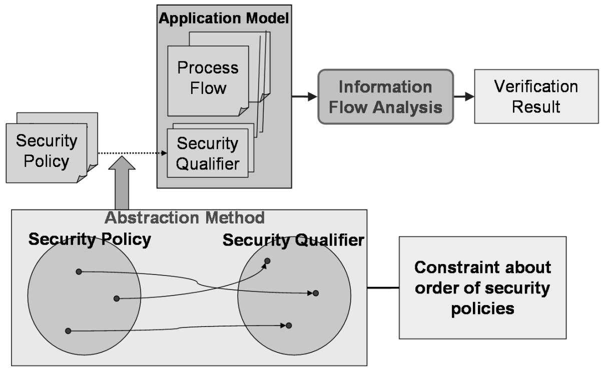


Figure 4. System Structure

$Type(v)$ denotes a security type where the variable v is typed, $Type(S_v)$ denotes a set of security types of $Type(v)$ where variable v in S_v, $UB(S_T)$ denotes the upperbound of security types in S_T, and $LB(S_T)$ denotes the lowerbound of security types in S_T. T_{ub} stands for the upperbound of allsecurity types, and T_{lb} stands for the lowerbound of allsecurity types. In addition, v_{ub} stands for a variable typed as T_{ub} (i.e. $Type(v_{ub}) = T_{ub}, \vdash v_{ub} : T_{ub}$), and v_{lb} stands for a variable typed as T_{lb} (i.e. $Type(v_{lb}) = T_{lb}, \vdash v_{lb} : T_{lb}$). We write $\vdash exp : T$ to mean that the expression exp has type T according to the typing rules. Similarly, $[pc] \vdash A$ means the activity A in the BPEL description is *typable* in the *security context pc*. $T_1 \sqsubseteq T_2$ means that an order relationship is defined on security types T_i, and the security type T_2 is *upper* than T_1. Using those notations, the typing rules for BPEL (a subset of WS-BPEL version 2.0) are defined as follows:

$$(1) \quad \vdash exp : T_{lb}$$

$$(2) \quad \vdash exp : LB(Type(Vars(exp)))$$

$$(3) \quad \frac{\vdash exp : T_u \quad \vdash T_l \sqsubseteq T_u}{\vdash exp : T_l}$$

$$(4) \quad \frac{\vdash v : T_{lb}}{[pc] \vdash \begin{array}{l}\texttt{<copy>}\\ \quad \texttt{<from expression="}exp\texttt{"/>}\\ \quad \texttt{<to variable="}v\texttt{"/>}\\ \texttt{</copy>}\end{array}}$$

$$(5) \quad \frac{\vdash exp : T \quad \vdash v : T}{[T] \vdash \begin{array}{l}\texttt{<copy>}\\ \quad \texttt{<from expression="}exp\texttt{"/>}\\ \quad \texttt{<to variable="}v\texttt{"/>}\\ \texttt{</copy>}\end{array}}$$

$$(6) \quad \frac{[pc] \vdash A_1 \quad [pc] \vdash A_2}{[pc] \vdash \texttt{<assign>}A_1 \quad A_2\texttt{</assign>}}$$

$$(7) \quad \frac{[pc] \vdash A_1 \quad [pc] \vdash A_2}{[pc] \vdash \texttt{<sequence>}A_1 \quad A_2\texttt{</sequence>}}$$

$$\frac{\vdash exp : T \quad [T] \vdash A}{[T] \vdash \begin{array}{l}\texttt{<if>}\\ \quad\texttt{<condition>}exp\texttt{</condition>}A\\ \texttt{</if>}\end{array}} \tag{8}$$

$$\frac{\vdash exp : T \quad [T] \vdash A_1 \quad [T] \vdash A_2}{[T] \vdash \begin{array}{l}\texttt{<if>}\\ \quad\texttt{<condition>}exp\texttt{</condition>}A_1\\ \quad\texttt{<else>}A_2\texttt{</else>}\\ \texttt{</if>}\end{array}} \tag{9}$$

$$\frac{\vdash exp : T \quad [T] \vdash A \quad \vdash T_e \sqsubseteq T \quad [T_e] \vdash A_e}{[T_e] \vdash \begin{array}{l}\texttt{<if>}\\ \quad\texttt{<condition>}exp\texttt{</condition>}A\\ \quad A_e\\ \texttt{</if>}\end{array}} \tag{10}$$

$$\frac{\vdash exp : T \quad [T_l] \vdash A \quad \vdash T_l \sqsubseteq T}{[T_l] \vdash \begin{array}{l}\texttt{<elseif>}\\ \quad\texttt{<condition>}exp\texttt{</condition>}A\\ \texttt{</elseif>}\end{array}} \tag{11}$$

$$\frac{\vdash exp : T \quad [T_l] \vdash A_1 \quad [T_l] \vdash A_2 \quad \vdash T_l \sqsubseteq T}{[T_l] \vdash \begin{array}{l}\texttt{<elseif>}\\ \quad\texttt{<condition>}exp\texttt{</condition>}A_1\\ \quad\texttt{<else>}A_2\texttt{</else>}\\ \texttt{</elseif>}\end{array}} \tag{12}$$

$$\frac{\vdash exp : T \quad [T_l] \vdash A \quad \vdash T_e \sqsubseteq T_l \sqsubseteq T \quad [T_e] \vdash A_e}{[T_e] \vdash \begin{array}{l}\texttt{<elseif>}\\ \quad\texttt{<condition>}exp\texttt{</condition>}A\\ \quad A_e\\ \texttt{</elseif>}\end{array}} \tag{13}$$

$$\frac{\vdash exp : T \quad [T] \vdash A}{[T] \vdash \begin{array}{l}\texttt{<while>}\\ \quad\texttt{<condition>}exp\texttt{</condition>}A\\ \texttt{</while>}\end{array}} \tag{14}$$

$$\frac{\vdash exp : T \quad [T] \vdash A}{[T] \vdash \begin{array}{l}\texttt{<repeatUntil>}\\ \quad A\texttt{<condition>}exp\texttt{</condition>}\\ \texttt{</repeatUntil>}\end{array}} \tag{15}$$

$$\frac{\vdash e_s : T_s \quad \vdash e_f : T_f \quad \vdash T = LB(\{T_s, T_f\}) \quad [T] \vdash A}{[T] \vdash \begin{array}{l}\texttt{<forEach counterName="}v\texttt{">}\\ \quad\texttt{<startCounterValue>}e_s\\ \quad\texttt{</startCounterValue>}\\ \quad\texttt{<finalCounterValue>}e_f\\ \quad\texttt{</finalCounterValue>}\\ \quad\texttt{<scope>}A\texttt{</scope>}\\ \texttt{</forEach>}\end{array}} \tag{16}$$

$$\frac{[T_u] \vdash A \quad \vdash T_l \sqsubseteq T_u}{[T_l] \vdash A} \tag{17}$$

The rules (1)-(3) are for typing an expression, (4)-(6) are for assignment, (8)-(13) are for conditional branch, and (14)-(16) are for iterations (loops). The rule (17) is a *subsumption* rule. It denotes that if a BPEL description (or an activity) is typable in a security context T_u then it is typable in a *lower* security context T_l.

The rules are used for typing not only security levels of confidentiality but also security levels of integrity. In the context of confidentiality, a formula $T_1 \sqsubseteq T_2$ represents "the confidentiality of security type T_2 is *higher* than T_1. However in the context of integrity, the same formula $T_1 \sqsubseteq T_2$ represents "the integrity of security type T_2 is *weaker* than T_1. The meanings of order predicates $\sqsubseteq$ and $\sqsupseteq$ are not intuitive in the context of integrity because the information flow direction is the exact opposite of the direction in the context of confidentiality.

3.3. Constraints on Abstraction

The specification of WS-SecurityPolicy specifies the syntax and semantics of security assertions. For example, a `<sp:AlgorithmSuite>` assertion designates an algorithm suite which will be applied to targeted elements and/or parts for their encryption and/or digital signatures. In an `<sp:AlgorithmSuite>` assertion, an algorithm suite name is specified, which represents a suite of message digest function for digital signature, encryption, symmetric/asymmetric key wrap algorithms, and so on. These algorithms have intrinsic strengths, and therefore, there exist some orders about the confidentiality level and integrity level between arbitrary algorithm suites. Table 1 shows the list of algorithm suites defined in the WS-SecurityPolicy specification (extracted from the specification). In this table, the "Algorithm Suite" row is the names of algorithm suites, the "Enc" row is the names of encryption algorithms, the "Dig" row is the names of secure hash functions (message digest functions) for digital signatures, the "Sym KW" row is the names of key wrap algorithms for symmetric keys, and the "Asym KW" row is the names of key wrap algorithms for asymmetric keys.

Table 1. Algorithm Suite (extracted)

Algorithm Suite	Enc	Dig	Sym KW	Asym KW
Basic256	Aes256	Sha1	KwAes256	KwRsaOaep
Basic192	Aes192	Sha1	KwAes192	KwRsaOaep
Basic128	Aes128	Sha1	KwAes128	KwRsaOaep
TripleDes	TripleDes	Sha1	KwTripleDes	KwRsaOaep
Basic256Rsa15	Aes256	Sha1	KwAes256	KwRsa15
Basic192Rsa15	Aes192	Sha1	KwAes192	KwRsa15
Basic128Rsa15	Aes128	Sha1	KwAes128	KwRsa15
TripleDesRsa15	TripleDes	Sha1	KwTripleDes	KwRsa15
Basic256Sha256	Aes256	Sha256	KwAes256	KwRsaOaep
Basic192Sha256	Aes192	Sha256	KwAes192	KwRsaOaep
Basic128Sha256	Aes128	Sha256	KwAes128	KwRsaOaep
TripleDesSha256	TripleDes	Sha256	KwTripleDes	KwRsaOaep
Basic256Sha256Rsa15	Aes256	Sha256	KwAes256	KwRsa15
Basic192Sha256Rsa15	Aes192	Sha256	KwAes192	KwRsa15
Basic128Sha256Rsa15	Aes128	Sha256	KwAes128	KwRsa15
TripleDesSha256Rsa15	TripleDes	Sha256	KwTripleDes	KwRsa15

Those algorithms have influences on the strength of message protection. The message digest function has an influence on the integrity level, the encryption has an influence on the confidentiality level, and the key wrap algorithms have influences on both of these levels. However, note that the key wrap algorithm has no influence on the integrity level if no `<sp:SignedParts>` assertions and no `<sp:SignedElements>` assertions are specified in the `<wsp:Policy>` element when the `<sp:AlgorithmSuite>` assertion is specified in the `<wsp:Policy>` element. Also, it has no influence on the confidentiality level if no `<sp:EncryptedParts>` assertions and no `<sp:EncryptedElements>` assertions are specified.

Every algorithm suite has an encryption algorithm: AES-256 (“Aes256” in Table 1), AES-192 (“Aes192”), AES-128 (“Aes128”), or Triple-DES (“TripleDes”). In general, the strengths of those algorithm are ordered from strongest to weakest as AES-256, AES-192, AES-128, and Triple-DES. Therefore, from the viewpoint of the strength of confidentiality about encryption algorithm, the algorithm suites defined in Table 1 are ordered as follows.

$$\begin{aligned}
& \mathrm{Basic256} = \mathrm{Basic256Rsa15} \\
& = \mathrm{Basic256Sha256} = \mathrm{Basic256Sha256Rsa15} \\
\sqsupseteq\ & \mathrm{Basic192} = \mathrm{Basic192Rsa15} \\
& = \mathrm{Basic192Sha256} = \mathrm{Basic192Sha256Rsa15} \\
\sqsupseteq\ & \mathrm{Basic128} = \mathrm{Basic128Rsa15} \\
& = \mathrm{Basic128Sha256} = \mathrm{Basic128Sha256Rsa15} \\
\sqsupseteq\ & \mathrm{TripleDes} = \mathrm{TripleDesRsa15} \\
& = \mathrm{TripleDesSha256} = \mathrm{TripleDesSha256Rsa15}
\end{aligned}$$

On the other hand, every algorithm suite has a message digest function using either SHA-1 (“Sha1” in Table 1) or SHA-256 (“Sha256”). In general, SHA-256 is stronger than SHA-1. Therefore, from the viewpoint of the strength of the integrity of the message digest function, the algorithm suites defined in Table 1 are ordered as follows:

$$\begin{aligned}
& \mathrm{Basic256} = \mathrm{Basic192} \\
& = \mathrm{Basic128} = \mathrm{TripleDes} \\
& = \mathrm{Basic256Rsa15} = \mathrm{Basic192Rsa15} \\
& = \mathrm{Basic128Rsa15} = \mathrm{TripleDesRsa15} \\
\sqsupseteq\ & \mathrm{Basic256Sha256} = \mathrm{Basic192Sha256} \\
& = \mathrm{Basic128Sha256} = \mathrm{TripleDesSha256} \\
& = \mathrm{Basic256Sha256Rsa15} \\
& = \mathrm{Basic192Sha256Rsa15} \\
& = \mathrm{Basic128Sha256Rsa15} \\
& = \mathrm{TripleDesSha256Rsa15}
\end{aligned}$$

The same thing as encryption algorithm holds for the key wrap algorithms for symmetric/asymmetric keys. Every algorithm suite has a symmetric key: AES-256 (“KwAes256” in Table 1), AES-192 (“KwAes192”), AES-128 (“KwAes128”), or Triple-DES (“KwTripleDes”). From the viewpoint of the strength of integrity and confidentiality of the symmetric key wrap algorithm, the algorithm suites defined in Table 1 are ordered as follows.

$$\begin{aligned}
& \mathrm{Basic256} = \mathrm{Basic256Rsa15} \\
& = \mathrm{Basic256Sha256} = \mathrm{Basic256Sha256Rsa15} \\
\sqsupseteq\ & \mathrm{Basic192} = \mathrm{Basic192Rsa15} \\
& = \mathrm{Basic192Sha256} = \mathrm{Basic192Sha256Rsa15} \\
\sqsupseteq\ & \mathrm{Basic128} = \mathrm{Basic128Rsa15} \\
& = \mathrm{Basic128Sha256} = \mathrm{Basic128Sha256Rsa15} \\
\sqsupseteq\ & \mathrm{TripleDes} = \mathrm{TripleDesRsa15} \\
& = \mathrm{TripleDesSha256} = \mathrm{TripleDesSha256Rsa15}
\end{aligned}$$

An algorithm suite has an asymmetric algorithm: RSA-OAEP (“KwAesOaep” in Table 1) and RSA-1.5 (“KwRsa15”). In general, the strengths of those algorithm are ordered from strongest to weakest as RSA-OAEP and RSA-1.5. Therefore, from the viewpoint of the strength of the integrity and the confidentiality of the asymmetric algorithms, the algorithm suites defined in Table 1 are ordered as follows.

$$\begin{aligned}
& \mathrm{Basic256} = \mathrm{Basic192} \\
& = \mathrm{Basic128} = \mathrm{TripleDes} \\
& = \mathrm{Basic256Sha256} = \mathrm{Basic192Sha256} \\
& = \mathrm{Basic128Sha256} = \mathrm{TripleDesSha256} \\
\sqsupseteq\ & \mathrm{Basic256Rsa15} = \mathrm{Basic192Rsa15} \\
& = \mathrm{Basic128Rsa15} = \mathrm{TripleDesRsa15} \\
& = \mathrm{Basic256Sha256Rsa15} \\
& = \mathrm{Basic192Sha256Rsa15} \\
& = \mathrm{Basic128Sha256Rsa15} \\
& = \mathrm{TripleDesSha256Rsa15}
\end{aligned}$$

In addition, `<sp:ProtectTokens>`, `<sp:OnlySignEntireHeadersAndBody>`, `<sp:IncludeTimestamp>`, and `<sp:EncryptSignature>` assertions also have influences on the integrity and/or confidentiality levels. Each assertion is optional, and it has no child nodes. Each assertion has a influence on the integrity level. The strength of the integrity of a security policy is weaker or equal to a security policy which is the same except when one of the above assertions is added.

The above security assertions are independent of each other, and the order relationships cannot be defined among them. For example, there are no constraints on the order of the integrity level between a security policy where an `<sp:IncludeTimestamp>` assertion is specified but an `<sp:ProtectTokens>` assertion is not and a security policy where an `<sp:IncludeTimestamp>` assertion is not specified but an `<sp:ProtectTokens>` assertion is specified.

Supporting tokens assertions, such as

- `<sp:SupportingTokens>` assertion
- `<sp:SignedSupportingTokens>` assertion
- `<sp:EndorsingSupportingTokens>` assertion
- `<sp:SignedEndorsingSupportingTokens>` assertion,

are used to specify message protection with additional security tokens. Therefore, the `<sp:AlgorithmSuite>` assertion specified in the `<wsp:Policy>` element of a supporting tokens assertion has an influence on the integrity and/or confidentiality levels. In addition, the supporting tokens assertions for endorsing, such as

- `<sp:EndorsingSupportingTokens>` assertion

`<sp:SignedEndorsingSupportingTokens>` assertion,

improve the strength of the integrity, because those assertions sign the digital signature of the message. When one of the signed supporting tokens assertions such as

`<sp:SignedSupportingTokens>` assertion `<sp:SignedEndorsingSupportingTokens>` assertion,

is used for signing, it also improves the strength of the integrity, because the security token of the assertion is signed with the signature key of the message.

The security developer has to define an abstraction method for the security policies that satisfies the constraints described in this section. The security developer has the freedom to define the mapping as an abstraction if it is not subjected to the constraints.

Suppose that the following security policies are given, and Policy 1 is for the Retailer service, and Policy 2 is for the Warehouse service. Note that the policy descriptions are simplified for this explanation.

```
[Policy 1]
<wsp:Policy>
  <sp:SymmetricBinding>
    <wsp:Policy>
      <sp:AlgorithmSuite>
        <wsp:Policy>
          <sp:Basic256Sha256/>
        </wsp:Policy>
      </sp:AlgorithmSuite>
      <sp:IncludeTimestamp/>
      <sp:ProtectTokens/>
    </wsp:Policy>
  </sp:SymmetricBinding>
  <sp:SignedParts>
    <sp:Body/>
  </sp:SignedParts>
  <sp:EncryptedParts>
    <sp:Body/>
  </sp:EncryptedParts>
</wsp:Policy>

[Policy 2]
<wsp:Policy>
  <sp:SymmetricBinding>
    <wsp:Policy>
      <sp:AlgorithmSuite>
        <wsp:Policy>
          <sp:Basic256/>
        </wsp:Policy>
      </sp:AlgorithmSuite>
    </wsp:Policy>
  </sp:SymmetricBinding>
  <sp:SignedParts>
    <sp:Body/>
  </sp:SignedParts>
  <sp:EncryptedParts>
    <sp:Body/>
  </sp:EncryptedParts>
</wsp:Policy>
```

The security levels of confidentiality about these security policies are the same because the same encryption algorithm and the same symmetric key wrap algorithm are used in the specified algorithm suite: Basic256Sha256 and Basic256. On the other hand, the security levels of the integrity about these security policies are different, because the message digest function of Basic256Sha256 and Basic256 are SHA-256 and SHA-1. And also an `<sp:IncludeTimestamp>` assertion and an `<sp:ProtectTokens>` assertion are specified in Policy 1. That means the integrity level of Policy 1 is stronger than Policy 2 (and, the security type of Policy 2 is *upper* than the security type of Policy 1).

Suppose that the security developer defines the security levels of the confidentiality as **high**, **middle**, **low**, and also defines the security levels of the integrity as **strong**, **normal**, **weak**. Now the security developer defines an abstraction method for the security policies. Suppose that it is defined that Policy 1 is mapped to [confidentiality: **high**, integrity: **strong**] (SQ1) and Policy 2 is mapped to [confidentiality: **high**, integrity: **normal**] (SQ2).

3.4. Verifying the Consistency of Security Qualifiers

Each security policy of a service component is translated into a security qualifier as shown in Fig. 3. The consistency of the security qualifiers is verified with the process flows. Fig. 5 defines a process flow for the operation `submitOrder` of the Retailer service, and Fig. 6 shows its BPEL description. The operation receives an order `PartsOrder` submitted by the `Customer`, and invokes the three Warehouse services A, B, and C. When a Warehouse service has enough stock for the order, then the operation `submitOrder` returns a response of `OrderResponse`. If any Warehouse service does not have sufficient stock, the operation returns a response of `OrderResponse` with the value 0. Note that the process flow does not consider the partial shipping.

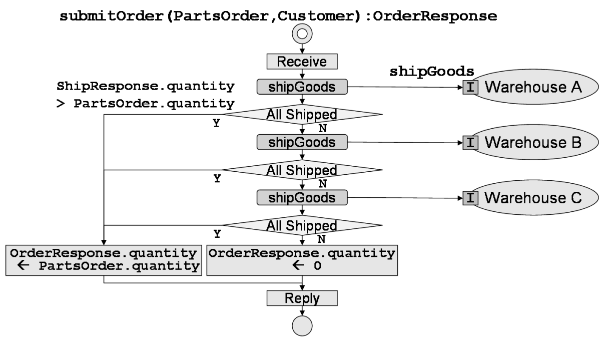


Figure 5. A Process Flow

```
<process>
 <sequence>
  <receive variable="PartsOrder"/>
  <flow>
   <sequence>
    <invoke partnerLink="A" operation="shipGoods"
      outputVariable="ShipResponse"/>
    <if>
     <condition>
      ShipResponse.quantity > PartsOrder.quantity
     </condition>
     <assign>
      <copy>
       <from expression="PartsOrder.quantity"/>
       <to variabe="OrderResponse.quantity"/>
      </copy>
     </assign>
     <else>
      <invoke partnerLink="B" operation="shipGoods"
        outputVariable="ShipResponse"/>
      <if>
       <condition>
        ShipResponse.quantity > PartsOrder.quantity
       </condition>
       <assign>
        <copy>
         <from expression="PartsOrder.quantity"/>
         <to variabe="OrderResponse.quantity"/>
        </copy>
       </assign>
       <else>
        <invoke partnerLink="C" operation="shipGoods"
          outputVariable="ShipResponse"/>
        <if>
         <condition>
          ShipResponse.quantity > PartsOrder.quantity
         </condition>
         <assign>
          <copy>
           <from expression="PartsOrder.quantity"/>
           <to variabe="OrderResponse.quantity"/>
          </copy>
         </assign>
         <else>
          <copy>
           <from expression="0"/>
           <to variabe="OrderResponse.quantity"/>
          </copy>
         </else>
        </if>
       </else>
      </if>
     </else>
    </if>
   </sequence>
  </flow>
  <reply/>
 </sequence>
</process>
```

Figure 6. A Process Flow in BPEL

Variables in the process flow are typed with security types defined as the security qualifiers. As described above, the Retailer services has the security qualifier SQ1 and the Warehouse service has the security qualifier SQ2. Therefore, the variables `PartsOrder`, `Customer`, and `OrderResponse` are typed with SQ1, and `ShipResponse` (which is the return value of `shipGoods` in the Warehouse service) is typed with SQ2. By using type inference, the information flow of the process flow is analyzed. As the result of the analysis, an inconsistency in the integrity is found in the process flow. The security type for the integrity of the conditional expression

```
<condition>
 ShipResponse.quantity > PartsOrder.quantity
</condition>
```

is **normal** because the security type for the integrity of the variable `PartsOrder` is **strong**, but the variable `ShipResponse` is **normal**. It is the consequence of rule (2). The program context for the integrity of this branch is **normal** because the conditional expression is **normal**. This is an implicit flow. The rule (9) holds in the description. The branch of the condition reaches the assignment

```
<assign>
 <copy>
  <from expression="PartsOrder.quantity"/>
  <to variabe="OrderResponse.quantity"/>
 </copy>
</assign>
```

The security type for the integrity of this assignment is **strong**. It is the consequence of rule (5) and (9). That causes the inconsistency in the integrity.

This inconsistency means that there is a certain level of risk that the messages exchanged between the Retailer service and the Warehouse service could be tampered with. It follows that the assignment in the unintended branch may be executed by an attacker to tamper with the response message from the Warehouse service to the Retailer service.

4. Related Work

A method to the verify security policies by introducing an abstract link specification language was proposed in [11]. A high-level link specification describes the intended secrecy and authentication goals for messages flowing between SOAP processors. The link specification is compiled into WS-SecurityPolicy configuration files. Then it is verified to check whether or not the security goals of the link specification are met by a given set of WS-SecurityPolicy files by using π-calculus. The method is not targeting the WS-SecurityPolicy files written by the security developer directly, but the WS-SecurityPolicy files generated from the abstract link specifications. Our method can verify the WS-SecurityPolicy files written by the security developer directly.

In [12] a framework for securing Web service compositions in BPEL by using aspects is proposed. The framework consists of three major components: A security service, a process container and a deployment descriptor. The process container is implemented as a set of aspects that are specified in AO4BPEL [13], an aspect-oriented extension to BPEL offering more modularity and adaptability. These aspects are generated from a generic aspect library at development time according to the deployment descriptor, which specifies the security requirements of BPEL activities along with the required security parameters such as keys and certificates. This method is an extension of BPEL interpretation framework, and the security is tackled by the generated aspects at the execution time. Our method verifies the security policies statically at the time of development by abstracting them.

5. Concluding Remarks

By using a bottom-up methodology like CBM, applications can be built up from existing service components in an SOA environment. It is necessary to determine if the application can be built from existing service components. However this is difficult because not only the functional consistency of the existing services, but also the consistency of their non-functional aspects such as security must be verified. Since every service component has its own security policy given in advance, verifying the consistency of the security policies is called for. However, since the Web Services Security Policy Language is intricate, it is difficult to verify the consistency of the security policies directly.

In this paper, we have proposed a method to verify the consistency of security policies by abstracting them. The possible abstraction method must be defined by the security developer in advance. Defining an abstraction method is constrained based on the semantics of the policy language. Every security policy is abstracted with a defined abstraction method, and translated into the corresponding security qualifier. The security qualifiers are attached to the service components in the application model in order to verify their consistency with the process flow in BPEL by using an information flow analysis technique.

Since applications are becoming more complex in the SOA environment, it is becoming harder to develop them. A bottom-up methodology like CBM makes the development easier by reusing existing service components. As SOA-based applications are being more widely adopted, their security is becoming more important. The verification method we have proposed will decrease the difficulty at the time of building up applications from existing service components.

References

[1] CBDI Forum Ltd. A CBDI Report Series - Guiding the Transition to Web Services and SOA, `http://www.cbdiforum.com/bronze/downloads/ws_roadmap_guide.pdf`, 2003.

[2] P. Devanbu and D. Stubblebine. Software Engineering for Security: a Roadmap, *Proc. of the 22nd International Conference on Software Engineering (ICSE 2000)*, 227–239, Limerick, Ireland, 2000.

[3] R. J. Anderson. *Security Engineering: A Guide to Building Dependable Distributed Systems*, John Wiley & Sons, Inc., 2001.

[4] IBM Corporation. Component-based approach to strategic change, `http://www.ibm.com/services/us/igs/cbm/html/cbm_jump.html`, 2005.

[5] G. Della-Libera, M. Gudgin, P. Hallam-Baker, M. Hondo, H. Granqvist, C. Kaler, H. Maruyama, M. McIntosh, A. Nadalin, N. Nagaratnam, R. Philpott, H. Prafullchandra, J. Shewchuk, D. Walter, and R. Zolfonoo. Web Services Security Policy Language (WS-SecurityPolicy) Version 1.1, `http://specs.xmlsoap.org/ws/2005/07/securitypolicy/ws-securitypolicy.pdf`, 2005.

[6] Web Services Interoperability Organization (WS-I). Supply Chain Management Sample Application Architecture, `http://www.ws-i.org/SampleApplications/SupplyChainManagement/2003-12/SCMArchitecture1.01.pdf`, 2003.

[7] D. Volpano, C. Irvine, and G. Smith. A Sound Type System for Secure Flow Analysis, *Journal of Computer Security*, 4(2-3):167–187, 1996.

[8] A. Sabelfeld and A. C. Myers. Language-Based Information-Flow Security, *IEEE Journal on Selected Areas in Communications*, 21(1):5–19, 2003.

[9] T. Andrews, F. Curbera, H. Dholakia, Y. Goland, J. Klein, F. Leymann, K. Liu, D. Roller, D. Smith, S. Thatte, I. Trickovic, and S. Weerawarana. Business Process Execution Language for Web Services (BPEL4WS) Version 1.1, `http://www-128.ibm.com/developerworks/library/specification/ws-bpel/`, 2005.

[10] OASIS Open. Web Services Business Process Execution Language Technical Committee (WS-BPEL TC), `http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=wsbpel`

[11] K. Bhargavan, C. Fournet, and A. D. Gordon. Verifying Policy-Based Security for Web Services, *Proc. of the 11th ACM Conference on Computer and Communications Security (CCS'04)*, 268–277, Washington DC, U.S.A., 2004.

[12] A. Charfi and M. Mezini. Using Aspects for Security Engineering of Web Service Compositions, *Proc. of the IEEE International Conference on Web Services (ICWS'05)*, 59–66, Orlando, Florida, U.S.A., 2005.

[13] A. Charfi and M. Mezini. Aspect-Oriented Web Service Composition with AO4BPEL, *Proc. of the European Conference on Web Services (ECOWS'04)*, LNCS 3250, Springer-Verlag, 168–182, Erfurt, Germany, 2004.