Enhance Matching Web Service Security Policies with Semantic

Tuan-Dung Cao and Nguyen-Ban Tran

Abstract. Web service security policy is a way to add some security restrictions to a web service. Matching web service security policies is hard and important to integrate web services effectively. However, the lack of semantics in WS-SecurityPolicy (WS-SP) hampers the effectiveness of matching the compatibility between security policies of different web services. To enhance matching web service security policies, we propose a semantic approach for specifying and matching the security policies. The approach uses the transformation of WS-SP into an OWL-DL ontology, the definition of a set of semantic relations that can exist between the provider and requestor security concepts, and the algorithm to determine the match level of the provider and requestor security policies. We show how these relations and the matching algorithm lead to more correct and more flexible matching of security policies.

1 Introduction

Nowadays, web service becomes a popular standard in information technology industry. To use the existing web services effectively, we need to integrate them by some modern technology and architecture like Service Oriented Architecture (SOA). Discovering a dynamic service and service selection are essential aspects of SOA. To acquire the business requirement, selecting web service must not only take the functional aspects, but also non-functional properties of the services. One of the

Tuan-Dung Cao
School of Information and Communication Technology, Hanoi University of Science and Technology, Vietnam
e-mail: dungct@soict.hut.edu.vn

Nguyen-Ban Tran
Vietnam Joint Stock Commercial Bank for Industry and Trade, Hanoi, Vietnam
e-mail: nguyenbantran@gmail.com

V.-N. Huynh et al. (eds.), *Knowledge and Systems Engineering, Volume 1*,
Advances in Intelligent Systems and Computing 244,
DOI: 10.1007/978-3-319-02741-8_19,

most important non-functional properties of a web service is security. This paper focuses on web service security policy, a non-functional property of web service.

In a big company or organization, SOA is an effective way to provide and integrate its information technology services. Web service is the most adopted implementation of SOA. Message security becomes a major concern when using Web services. Therefore, message security becomes a major problem when we use SOA. Message security mainly means the confidentiality and the integrity of data transmitted through the message. Confidentiality and integrity can be assured by applying security mechanisms such as encryption and digital signature.

WS-SP [1] is widely accepted in the industry and it is currently a popular standard to be aggregated into the Web service architecture. Then matching WS-SP problem becomes more and more important while integrating Web services. However, WS-SP has a big weakness: it only allows syntactic matching of security policies. In fact, security policy matching depends on the policy intersection mechanism provided by WS-Policy [2]. The main step in this mechanism is matching the assertions specified in the service provider and requestor policies. This step only uses syntactic comparison between the security assertions, and does not use the semantics message security. Syntactic matching of security assertions restricts the effectiveness of checking the compatibility between provider and requestor policies. A simple comparison of the syntactic descriptions of security assertions maybe gets fault negative results. For example, consider syntactically different security assertions with the same meaning. Syntactic matching of security assertions only has a strict yes/no matching result. A more flexible matching is needed in order to consider subtle differences that may exist between security assertions and not bypass a potential partnership between a service requestor and a service provider. For example, in the cases when the provider and requestor security assertions have the same type but have some different proper-ties that make the provider assertion stronger, in security concepts, than the requestor assertion.

In the work [6], the authors proposed a semantic way to compare two security assertions, but they didn't compare two all security policies. The goal of this paper is to propose a semantic security approach a matching WS-SP algorithm to compare two security policies. In our approach, an ontology defined in Web Ontology Language (OWL) is used to present Web service security policies. Policy specifications offer semantic information about the requestor and provider. This information can be used to determine policy compatibility and to guarantee interoperability among requestor and provider service, with respect to security aspects

There are some ways to use semantic approach for working with security policy. You can use semantic to work with web service security configurations [7] or to man-age web service security [8]. We use semantic to get the matching level of two web service security policies .

We used a created prototype for the semantic matching of security assertions then add some relations to extend this completion to get the matching level of WS-SP. Based on the semantic interpretation of the syntactic heterogeneities that may occur between a provider assertion and a requestor assertion, our approach doesn't produce fault negative results and thus supports more correct matching. Besides, it

allows introducing close match and possible match as intermediary matching degrees, which makes security assertion matching more flexible.

The remainder of this paper is organized as follows. Section 2 and 3 introduce about WS-SP and the problem : matching WS-SP. Section 4 presents the ontology and relations in this ontology with semantics. Section 5 presents our algorithm of semantics matching security policies with two cases: a simple case and complex cases. Section 6 discusses about related work. Section 7 concludes the paper with some discussion.

2 Web Service Security Policy

WS-SP is a web services specification, which is created by IBM and used to extend the fundamental security protocols specified by the *WS-Security, WS-Trust* and *WS-SecureConversation* by offering mechanisms to represent the capabilities and requirements of web services as policies. Security policy assertions are based on the WS-Policy framework. Policy assertions can be used to require more generic security attributes like transport layer security *¡TransportBinding¿*, message level security *¡AsymmetricBinding¿* or timestamps, and specific attributes like token types. Policy assertions can be divided in following categories: Protection assertions identify the elements of a message that are required to be signed, encrypted or existent; Token assertions specify allowed token formats (SAML, X509, Username etc.); Security binding assertions control basic security safeguards like transport and message level security, cryptographic algorithm suite and required timestamps; Supporting token assertions add functions like user sign-on using a username token.

Policies can be used to drive development tools to generate code with certain capabilities, or may be used at runtime to negotiate the security aspects of web service communication. Policies may be attached to WSDL elements such as service, port, operation and message, as defined in WS Policy Attachment. WS-SP standard

```
<sp:IssuedToken>
<sp:RequestSecurityTokenTemplate>
<wst:TokenType>...#SAMLV2.0</wst:TokenType>
</sp:RequestSecurityTokenTemplate>
</sp:IssuedToken>
```

Fig. 1 An example of WS-SP

defines a set of security policy assertions for use with the WS-Policy framework. These assertions describe how messages are secured according to the WS-Security protocol. Typically, the published policies are compliant with the WS-Policy normal form. WS-SP assertions mainly describe the token types for security tokens, the cryptographic algorithms, and the scope of protection which means the parts of the SOAP message that shall be encrypted or signed.

3 Matching Web Service Security Policies Problem

When we have many web services and want to integrate them, we have to deal with the compatibility of their security policies. It leads to a problem: how to know whether two different security policies match or not.

Syntactic matching of security assertions restricts the effectiveness of checking the compatibility between them. In order to illustrate this deficiency, suppose that a requestor is looking for a fund transfer Web service that supports the signature of the message body with a symmetric key securely transported using an X509 token. Besides, the necessary cryptographic operations must be performed using Basic256 algorithm suite. This could be formalized, based on the WS-SP standard, by adding the assertions reqAss1 and reqAss2 to the requestor security policy (SP).

Furthermore, suppose that the requestor finds a Web service that fund transfer and whose SP includes provAss1 and provAss2 assertions.

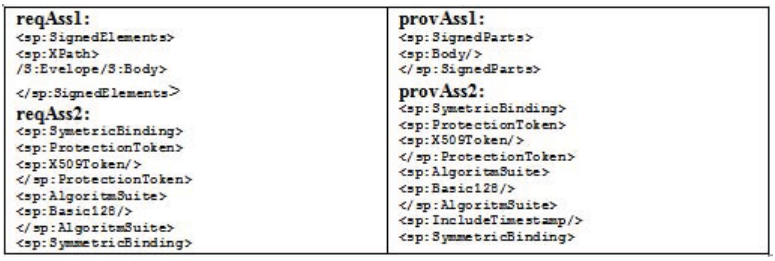

Fig. 2 Example of requestor and provider assertions

It is clear that the assertions specified in the provider SP are syntactically different than those specified in the requestor SP. Syntactic matching will then produce a no match result for these assertions. However, using semantic approach, we get a different matching result. In the above scenario reqAss1 and provAss1 assertions have the same meaning: sign the body of the message. Therefore, matching these two assertions must lead to a perfect match rather than no match. Besides, the only difference between reqAss2 and provAss2 assertions is that the *SymmetricBinding* assertion specified in the provider SP contains an extra child element which is sp:IncludeTimestamp. From security perspective, this strengthens the integrity service ensured by the message signature and makes provAss2 assertion stronger than reqAss2 assertion. Although it is not a perfect match, it is better than no match case.

From this analyzing, if the requestor can strengthen his security assertion by the inclusion of a timestamp, the perfect compatibility between both assertions will be ensured. We consider that it is more flexible to decide a possible match for this case in order to not reject a potential partnership between the service requestor and provider. Therefore, these two policies are possible match.

4 Semantic Matching of Web Service Security Assertions

In this section, we will show how to use semantic approach to transform a web ser-vice security policy to a form of ontology.

4.1 WS-SP Ontology

The assertions defined in WS-SP must be augmented with semantic information in order to enable semantic matching. Because this current version of WS-SP doesn't have any semantics presentation, the authors of [6] base on WS-SP and redesign an ontological representation of its assertions in order to obtain a WS-SP-based ontology that can be augmented with new semantic relations. A graphical depiction of the main parts of the ontology is shown in Fig. 3. Because some of these classes aren't designed well, we will use this main part of ontology and propose a way to extend some properties to ensure that this ontology will be used to get the matching level of two policies, not only two assertions. Web service security assertions are specified within security policies of the service provider and requestor. Typically, the structure of these policies is compliant with the WS-Policy normal form. In the normal form, a policy is a collection of alternatives, and an alternative is a collection of assertions. It is in the assertion components that a policy is specialized. Fig. 3 shows

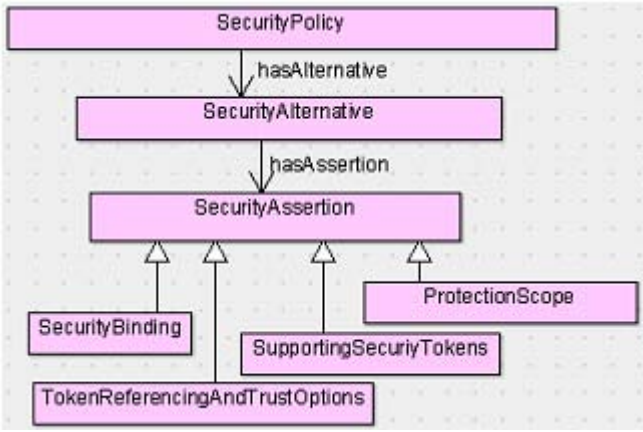

Fig. 3 Main classes of WS-SP-based ontology (from [6])

the main classes of the WS-SP-based ontology. This ontology includes the three main classes: SecurityPolicy, SecurityAlternative and SecurityAssertion in order to present security assertions within security policies. The SecurityAssertion have 4 subclasses: SecurityBinding, ProtectionScope, SupportingSecurityTokens, TokenReferencingAndTrustOptions. The first two classes are detail in reference [6]. We will add some individuals to SupportingSecurityToken to improve its semantic.

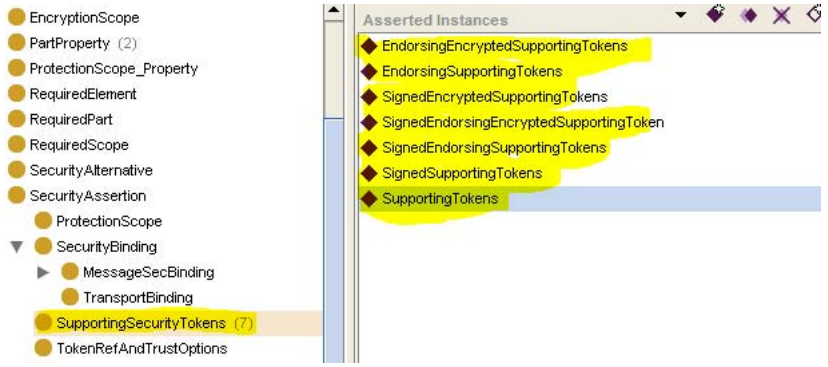

Fig. 4 Add individuals into SupportingSeuciryTokens

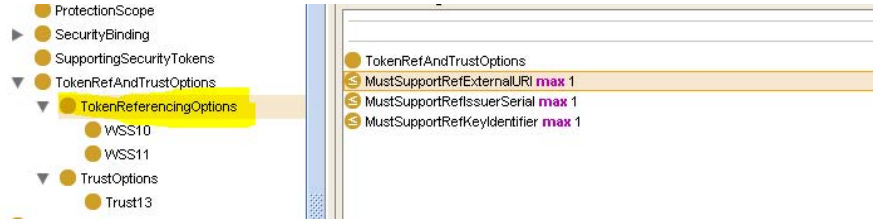

Fig. 5 Subclasses of *TokenRefAndTrustOptions* class

Fig.4 shows 7 individuals we added into SupportingSecurityTokens class. These individuals are used when there is a supporting token in security policy. By using these individuals, we can compare two SupportingSecurityToken more conrectly.

With *TokenRefAndTrustOptions*, we move related classes such as *TokenReferencingOptions, TrustOptions,* to become subclass of it.

4.2 Adding Semantic Relations

We used the WS-SP-based ontology with new semantic relations at the level of the security assertions and their properties. From the relation between assertions, we will compare two security policies to get the matching level of them. In this subsection, we epitomize all relations which are used. After we have the matching level of all assertions, we can get the matching level of a requestor policy with a provider policy.

To specify a relation between two assertions, we need to transform it into an ontology form. Subject to the security concepts, we will get the matching result, take security concepts of *ProtectionScope, SecurityBinding*, and *AlgorithmSuite*, for example. Next, we present the semantic relations.

isIdenticalTo relation: This relation allows to specify that a security concept specified by the provider is identical to a security concept specified by the requestor (no syntactic heterogeneity exists between them). There are two main cases for this relation in Table 1.

Table 1 isIdenticalTo relation

Case	isIdenticalTo
Properties of ProtectionScope, SecurityBinding	- If two simple properties point to the same security concept and have equal values. - Two complex properties point to the same security concept and all their child properties is identical.
Assertions of ProtectionScope, SecurityBinding	Point to the same security concept and all their properties are identical.

isEquivalentTo relation: This relation allows to specify that a security concept specified by the provider is equivalent to a security concept specified by the requestor. There are two main cases of security connects in Table 2.

Table 2 isEquivalentTo relation

Case	isEquivalentTo
Encrypted part and encrypted element or signed part and a signed element	XPath expression of message part is equivalent to the value of element. Example: '/S:Enveloppe/S:Body' and 'Body'.
Two algorithms suites	Algorithm suites defined in WS-SP are not disjoint and two algorithm suites can include many common elements. Example: Basic256 and Basic256Sha256.

isLargerThan relation. This relation only concerns the protection scope concept in SOAP message (in Table 3). *isSmallerThan* relation. This relation is to spec-

Table 3 isLargerThan relation

Case	isLargerThan
Part of SOAP message or XML elements	A part of message or a XML element is larger than any elements that belongs to it.
Encryption scopes, signature scopes, and required scopes Protection scope	If a scope is larger than other scope. If a scope has at least one property that larger than a property of other scope, and different scopes are identical or equivalent.

ify that the protection scope (in the SOAP messages) specified by the provider is smaller than the protection scope specified by the requestor. It is the opposite of *isLargerThan* relation and occurs in three cases just in an opposite manner to *isLargerThan* relation.

isStrongerThan relation. This relation is to specify that a security concept specified by the provider is stronger, in the security perspective, than a security concept specified by the requestor. There are three main cases for this relation (in Table 4).

isWeakerThan relation. This relation is to specify that a security concept specified by the provider is weaker, in the security perspective, than a security concept specified by the requestor. It is the opposite of *isStrongerThan* relation and occurs in three cases just in an opposite manner to *isStrongerThan* relation.

Table 4 isStrongerThan relation

Case	isStrongerThan
Security binding properties	Have influence on security strength, then we have *isStrongerThan* relation at the level of these properties.
Security binding assertion	If they point to the same type of security binding, and A provider assertion has at least one properties is stronger than the corresponding property of requestor assertion.
Protection scopes	If the provider scope has at least one extra scope, and the other scopes are identical, equivalent or have isLarger than relations.

hasTechDiffWith relation. In addition to concepts that allow to specify how a SOAP message is to be secured (confidentiality, integrity,), WS-SP-based ontology also includes concepts to describe technical aspects concerning how adding and referencing the security features in the message. At the level of these technical concepts, we define *hasTechDiffWith* relation to state that any mismatch between the provider concept properties and the requestor concept properties must be considered as a technical mismatch rather than a security level mismatch. *isMoreSpecificThan* relation. According to WS-SP standard, many security properties are optional to specify in a SP and WS-SP doesn't attribute default values for them. Therefore we define *isMoreSpecificThan* relation that occurs when a security concept specified by the provider is more specific (i.e., described in more detail) than a security concept specified by the requestor.

isMoreGeneralThan relation. This relation occurs when a security concept specified by the provider is more general (i.e., described in less detail) than a security concept specified by the requestor. It is the opposite of *isMoreSpecificThan* relation.

isDifferentFrom relation. This relation occurs when the security concepts specified by the requestor and the provider are semantically disparate.

5 Algorithm of Semantic Matching Web Service Security Policies

In this section, we propose an algorithm for matching provider and requestor security policies in a simple policy case and complex policies cases. This matching algorithm uses the matching level of assertions [6], which will be compared through the way mentioned below. After comparing two assertions, we will compare two security policies. There are four matching level: perfect match, close match, possible match, and no match in decreasing order of matching

5.1 Compare Two Simple Assertions

Simple assertion is a security assertion which contains only one alternative. To compare two simple assertions, we need to transform them into ontology form which bases on WS-SP ontology [6]. We will compare two assertions in ontology form, we will have many relations between them. We consider these relations to get the matching level between two assertions.

Perfect match. A perfect match occurs when *provAss* and *reqAss* are connected through *isIdenticalTo* or *isEquivalentTo* relations.

Close match. A close match occurs when *provAss* and reqAss are connected through *isMoreSpecificThan* relation. Possible match. A possible match is decided in three main cases:

Case 1. *provAss* and *reqAss* are connected through *isMoreGeneralThan* relation. This means that the information available cannot ensure that provAss can perfectly match reqAss. We assume that a potential partnership between the requestor and the provider can take place if the requestor can obtain additional information or negotiate with the provider.
Case 2. *provAss* is connected to *reqAss* through *isLargerThan, isStrongerThan*, or *hasTechDiffWith* relations. This means that the incompatibility between the two assertions doesn't negatively affect the security services and levels required in *reqAss*. We assume that a potential partnership between the requestor and the provider can take place if the requestor can strengthen his policy assertion or change some technical properties of his assertion.
Case 3. *provAss* and *reqAss* point to the same security concept and have at least one *isMoreGeneralThan, isLargerThan, isStrongerThan*, or *hasTechDiffWith* relation at the level of their properties, and their remaining properties are linked with semantic relations of type *isIdenticalTo, isEquivalentTo*, or *isMoreSpecificThan*. For example, suppose that *provAss* and *reqAss* point to the *SymmetricBinding* concept, but *provAss* has a protection token that is more general than the protection token specified in *reqAss*. In addition, *provAss* has an algorithm suite that is identical to the algorithm suite specified in *reqAss*. And finally, *provAss* has a *Timestamp* property that is stronger than the *Timestamp* property of *reqAss*. The two assertions have two heterogeneities that don't rule out the possibility of a match, so it is a possible match case.

No match. No match is decided in cases: *provAss* and *reqAss* are connected through *isDifferentFrom, isSmallerThan,* or *isWeakerThan* relations; *provAss* and *reqAss* point to the same security concept, and they have at least one *isDifferentFrom, isSmallerThan,* or *isWeakerThan* relation at the level of their properties.

In this subsection, we present the way to compare two simple assertion, so we create a function named $Compare\_Simple(A,B)$ to compare two simple assertion with the return values are: no match, possible match, close match, and perfect match. This result is sorted by increasing of value.

5.2 Compare Two Complex Assertions

A complex assertion has more than one alternative. In this subsection, we will proposed an algorithm to compare two complex assertions. We have a requestor A and provider B; assertions are reqA and provB. The matching level between reqA and provB is defined as the highest match of any alternative of provB with any alternative of reqA. For example, reqA includes alternatives: reqA1 and reqA2, provB includes alternatives: provB1 and provB2, with reqA1, reqA2, provB1,

provB2 are simple assertions. We assume that provB1 and reqA1 have relation isIdenticalTo, and provB2 and reqA2 have relation *isDifferentFrom*. So, reqA and provB have the matching level *isIdenticalTo*.

The algorithm: Let A, B are two complex assertions. Assertion A contains alternatives: A1, A2, , An and assertion B contains alternatives: B1, B2, , Bm. Because A, B are complex assertions, so Ai or Bj maybe a complex assertion.

We present this algorithm in pseudocode:

Compare_Assertion(Assertion A, Assertion B) {
If (A is a simple assertion and B is a simple assertion) then return Compare_Simple(A, B).
/ A contains alternatives: A1, A2, , An */*
/ B contains alternatives: B1, B2, ., Bm */*
Return (Max(Compare_Assertion(Ai, Bj) with 0 <i <n+1 and 0 <j <m+1)).
}

This algorithm can compare a complex assertion with a simple assertion. In this case, a simple assertion X have one alternative is X itself.

5.3 Compare Two Security Policies

In above subsections, we compared two assertions in general case (simple case or complex case). Now, we will use this matching assertion algorithm to build a matching security policy algorithm. The matching process consists in checking to what extent each security assertion reqAss specified in the requestor SP is satisfied by a security assertion provAss specified in the provider SP. The matchmaker has to perform three main tasks. Firstly, it must create all possible semantic relations at the level of each pair of provider and requestor assertions and get matching level of all assertion pairs. Secondly, based on the created semantic relations, it must decide the appropriate matching degree for each provAss-reqAss pair. The final matching degree for a requestor assertion is the highest level of matching level against all of the checked provider assertions. Thirdly, after all requestor assertions have a matching level, the matching degree of requestor and provider policies is the lowest matching degree of all assertions in requestor policy.

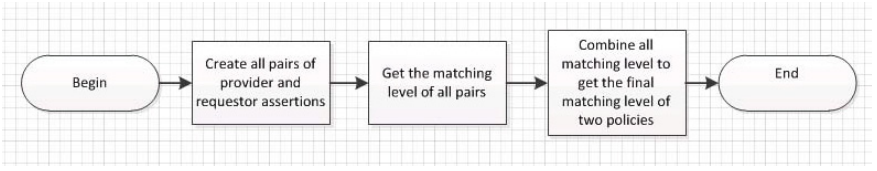

Fig. 6 Matching security policy algorithm

Matching algorithm: Let X, Y are two security policy. Policy X contains assertion X1, X2, , Xn and policy Y contains assertion Y1, Y2, , Ym. Because we only need to

know a requestor policy is satisfied or not by a provider policy. Then, this algorithm compare requestor policy X with provider policy Y. With each assertion Xi, we will get the highest satisfied matching level of it with all assertion of policy Y. *Relation_Assertion_Policy(Xi, B) = max {relation(Bj, Ai)with 0 <j <m+1)}*

With this definition, we will get the matching level of two policy is the lowest matching level of all assertion in requestor policy. Relation_policy(A, B) = min{relation_assertion_policy(Ai, B)), with 0 <i <n+1)}.

Our above proposed algorithm can help us get the more flexible and correct matching level of a requestor policy and a provider policy. This algorithm not only supports in the simple case of security policy, but also the complex case of security policy.

6 Related Work

WSPL (Web Services Policy Language) [4] is another syntactic policy language based on XML that can be used to specify a wide range of policies, including authorization, quality of service and reliable messaging etc. WSPL is of particular interest in several respects, for example, it supports merging two policies and policies can be based on comparisons other than equality allowing policies to depend on fine-grained attributes. In essence, a policy of WSPL is a sequence of rules, where each rule represents an acceptable alternative. A rule contains a number of predicates, which correspond to policy assertions in WS-Policy. Because WSPL is still based on syntactical domain models, its shortcomings are similar to WS-Policy. In a different work [5], they were interested in the work to conflict resolution semantic naming type when aligning SP in distributed multi-systems security policies. Their solution is based on ontology mediation for SP cooperation and understanding. They propose a helpful process for security experts, this process permit to mask heterogeneity and resolve conflicts between SP, using different steps.

In another work [10], the author proposed a simple ontology to compare two security policies, and then build an algorithm to compare security policies. This algorithm bases on ontology and it uses the semantic reasoner to get the final result. However, the algorithm is simple, and the comparing algorithm isn't detail.

In the work [6], the authors proposed a WS-SP-based ontology and some relations to compare two security assertions. That paper show how to get the matching level of two simple security assertions, but it lacks the comparing of all two polices and the processing in the complex security assertions case.

In addition to being compatible to WS-SP and the work [6], our approach is better than previous approaches: we extend the WS-SP-based ontology with additional semantic relations that support more correct and more flexible semantic matching of Web service security policies, not only simple assertions.

7 Conclusion and Discussion

In this paper, we used an approach to provide a semantic extension to security assertions of Web services, then from this result we compared two web service security policies. The approach is based on the transformation of WS-SP into an OWL-DL ontology, and using the relations in this ontology to get matching level of two security policies.

Our semantic approach supports more correct and more flexible security policy matching compared to syntactic matching of previous works that combined ontology and Web service security properties because our approach also compares complex security policies to get the final matching degree between them, which contain several alternatives. Besides, we plan to develop a tool that automatically transforms a WS-SP policy into our ontological representation, and show a security policy which is compatible with two requestor and provider, or print out no match result.

References

[1] OASIS: WS-SecurityPolicy 1.3, http://www.oasis-open.org/specs/
[2] W3C: WS-Policy 1.5, http://www.w3.org/TR/ws-policy/
[3] Verma, K., Akkiraju, R., Goodwin, R.: Semantic matching of Web service policies. In: Proceedings of the Second Workshop on Semantic and Dynamic Web Processes, pp. 79–90 (2005)
[4] Anderson, A.H.: An Introduction to the Web Services Policy Language. In: Fifth IEEE International Workshop on Policies for Distributed Systems and Networks, POLICY 2004 (2004)
[5] Benammar, O., Elasri, H., Sekkaki, A.: Semantic Matching of Security Policies to Support Security Experts
[6] Brahim, M.B., Chaari, T., Jemaa, M.B., Jmaiel, M.: Semantic matching of WS-Security Policy assertions (2011)
[7] Bhargavan, K., Fournet, C., Gordon, A.D., O'Shea, G.: An Advisor for Web Services Security Policies (2005)
[8] Garcia, D.Z.G., de Toledo, M.B.F.: Web Service Security Management Using Semantic Web Techniques (2004)
[9] Bhargavan, K., Fournet, C., Gordon, A.D.: Verifying Policy-Based Security for Web Services
[10] He, Z.-Q., Wu, L.-F., Zheng, H., Lai, H.-G.: Semantic Security Policy for Web Service. In: IEEE International Symposium on Parallel and Distributed Processing with Applications (2009)