A Semantic Model to Support Security Matching in Cloud Environments

Giuseppe Di Modica and Orazio Tomarchio
Dipartimento di Ingegneria Elettrica, Elettronica e Informatica, Università di Catania,
Viale Andrea Doria 6, 95125 Catania, Italy

Keywords: Cloud Computing, Security Policies, Semantic, Ontology.

Abstract: Despite its technological advances, cloud computing's adoption is not as wide as expected. Security is still a big concern that prevents many to "cloudify" their applications and put their data in the hands of a cloud provider. Also, interoperable scenarios fostered by SOA technologies exacerbate the security question, as customers have to deal with multiple providers who, in their turn, must establish mutual trust relationships in order to interoperate. In the last few years, policies are being used as a means to build networks of trustiness among cloud providers. Standards and specifications on security management through policies have also appeared. We argue that the main problem with this approach is that policies are expressed through syntactic languages which, if processed by computers, show well-known limitations. We then propose an approach that leverages on the semantic technologies to enrich security policies with semantic contents enabling machine reasoning. The framework we developed caters for the security needs of both customers and providers, and aims at making a smart match between what is requested and what is offered in terms of security. On the customer side, no extra effort is required other than specifying their security policies according to well-established security notations; an automatic procedure is charged of adding semantic content to the policies.

1 INTRODUCTION

Cloud computing has emerged as a flexible, cost-effective and proven delivery platform for providing business services over the Internet, giving organizations the opportunity to increase their service delivery efficiencies, coping with dynamic business requirements. However, when adopting a cloud computing paradigm, an organization has to shift much of the control over the data and operations from its IT department to an external cloud provider. As a result, cloud customers must establish trust relationships with their providers, also being aware of how these providers implement, deploy and manage security on their side. The wide acceptance and adoption of the SOA paradigm (Papazoglou and van den Heuvel, 2007) has posed the basis for the realization of world-wide and cross-domain scenarios of interoperability among services. In a complex scenario where the final service delivered to the end-user is realized by composing services run by different providers in different administrative domains, the end-user security is also affected by trust relationships established among providers.

According to (Phan et al., 2008) policies will be increasingly important to effectively support the "Internet of Service" vision. Though policies have been traditionally adopted to automate network administration tasks, in the last few years policy management has extended its original scope. We are strongly convinced they may serve very well the purpose of granting security. Multiple approaches for policy specification have been proposed that range from formal policy languages, which can be directly processed by a computer, to rule-based policy notation. Some have proposed to represent policies as entries in a table consisting of multiple attributes. The main approach followed today is based on the usage of metadata and languages for the specification of security policies: it permits to abstract from the low-level security mechanisms of the specific provider's infrastructure. The use of languages for the specification of security policies also enables, on the one hand, the providers to expose the security capabilities implemented within their administrative domain and, on the other one, the customers to express their security requirements.

The approach we propose calls on well-known policy and security specifications. We define customers' and providers' security requirements/capabilities within policies which are compliant to well-established specifications. What characterizes our approach is that security requirements and capabilities

Di Modica G. and Tomarchio O..
A Semantic Model to Support Security Matching in Cloud Environments.
DOI: 10.5220/0004375704270436
In *Proceedings of the 3rd International Conference on Cloud Computing and Services Science* (CLOSER-2013), pages 427-436
ISBN: 978-989-8565-52-5

are *semantically* enriched, thus enabling the employment of smarter mechanisms to make the match between what is required and offered in terms of security, respectively on the customer and on the provider side. The contribution of this work consists in the design of an ontology defining main security concepts, a procedure which automatically maps syntactic security requirements into semantically-enriched concepts, and a matchmaker engine which is capable of reasoning on the customer's and the provider's security requirements/capabilities to derive a match level.

The rest of the paper is structured as follows. Section 2 presents background and related work. Then in Section 3 the overall architecture of our framework is presented. Section 4 presents the semantic model we designed. In particular, we thoroughly discuss the developed security ontology, provide details on the mapping procedure and briefly recall the policy matching algorithm. In Section 5 an example is reported to validate our approach. Finally we conclude the work in Section 6.

2 RELATED WORK

The adoption of a policy based-approach for managing a system requires an appropriate language for policy representation and modeling and the design and development of a policy management framework for policy matching and enforcement. Several policy languages and framework have been developed following different approaches, and sometimes have been proposed in different application domains (Tonti et al., 2003). Among the most notably efforts in this domain, worth citing are Ponder (Damianou et al., 2001), a declarative object-oriented language that supports the specification of several types of management policies for distributed object systems, Kaos (Uszok et al., 2003), a policy management framework where concepts and policies themselves are represented using OWL, and Rei (Kagal et al., 2003), a policy framework where OWL is extended with the expression of relations like role-value maps, making the language more expressive than the original OWL. Kaos and Rei are ontology based policy languages that, although not specifically focused on the security domain, are able to express complex security policies: policy matching is also supported by the advanced reasoning capabilities these languages offer. However, when dealing with enterprises and cloud systems that adopt a service oriented paradigm, we believe that an approach compatible with existing standard for policy specification should be followed as far as possible.

WS-Policy (W3C, 2007) is the specification used in service oriented architectures to express policies. A policy is defined as a collection of alternatives and each alternative is a collection of assertions. Assertions specify characteristics that can be used for service selection such as requirements, capabilities or behaviors of a Web service. Policy assertions can represent both requirements on service requestors and capabilities of the service itself. Requirements represent a demand from service requestors to the esrvice provider to behave in a particular way; capabilities are the service providers promise of behaviour. Within the WS-Policy framework, the WS-SecurityPolicy (OASIS, 2012) specification can be used to express security requirements and security capabilities in the form of policies. For example, the use of a specific protocol to protect message integrity is a requirement that a service can impose on requestors. On the other hand, the adoption of a particular privacy policy when manipulating data of a requestor is a service capability.

Policy matching in WS-Policy works at a syntactic level: it offers a domain independent mechanism to find alternatives that are compatible to two policies. Two alternatives are compatible if, for each assertion in one alternative, there is a compatible assertion in the other one. Compatibility of assertions is defined exclusively according to the equivalence of their *qnames*, without any further mechanism involving either their structure or their content.

For this reason, several works in the literature (Sriharee et al., 2004; Verma et al., 2005; Speiser, 2010; Zheng-qiu et al., 2009) have been trying to enhance WS-Policy with semantic annotations. In (Sriharee et al., 2004), WS-Policy assertions refer to policies expressed in OWL: however that work is not focused on policy matching, but on modeling policies as a set of rules, which have to be evaluated using an external rule-based system, requiring reasoning beyond OWL. In (Verma et al., 2005) policies represented in WS-Policy are enhanced with semantics by using a combination of OWL and SWRL based rules to capture domain concepts and rules. In (Speiser, 2010) a lightweight approach to specify semantic annotations in WS-Policy is presented: it combines the syntactic matching with the semantic matching capability provided by OWL.

Our work, as described in next sections, carefully extends WS-Policy by referencing concepts from a security ontology directly in the policy assertions, thus maintaining backward compatibility with existing tools.

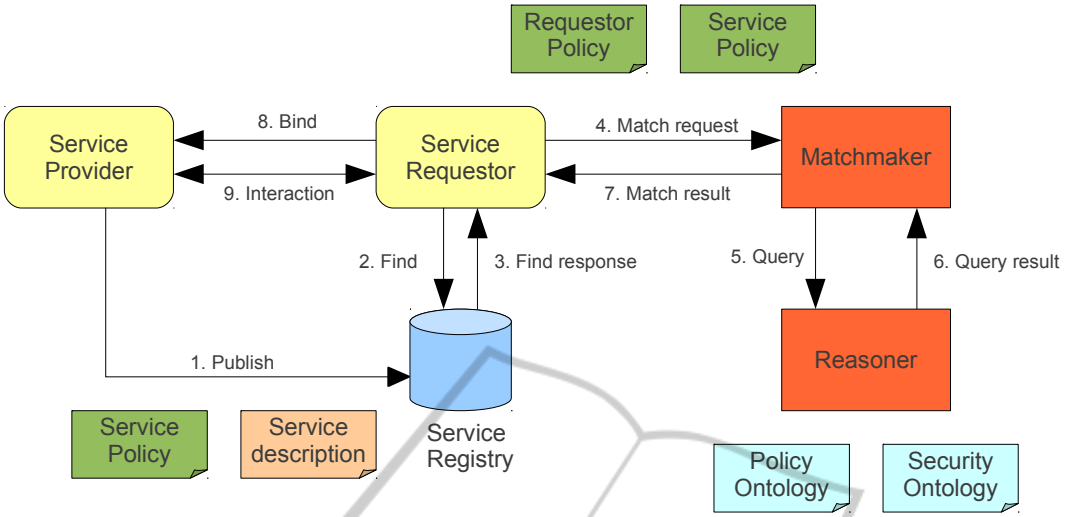

Figure 1: Overall architecture.

3 SYSTEM ARCHITECTURE

In this section we discuss the overall architecture of the semantic security policy matching framework. The basic architecture of the system, depicted in Figure 1, mainly follows that of classic SOA: the requestor, the provider and the registry keep playing their original role in the publish-find-bind cycle. The novelty is the presence of the *Matchmaker* and the *Reasoner* entities, and that of the policy-related information flows.

When a service provider advertises a service, besides providing the service "functional" information in the WSDL document, it also attaches the policy associated with that service. The WS-PolicyAttachment specification defines how to integrate the policy information. This information can be integrated either as an attachment to WSDL description, or by adding it to the UDDI registry as an extra document. When a requestor queries the UDDI registry for a service with given functional features, the registry also replies with the service's attached policy. When selecting the service(s) that match the functional requirements, the requestor may decide to evaluate how well the providers' security capabilities specified within the retrieved services' policies match those specified by its own policy.

The Matchmaker is the component that the requestor can rely on to perform such an evaluation. It is modelled as a web service, and receives requests with two inputs, i.e., the policies specifying the requirements and the capabilities of, respectively, the service requestor and the service provider. The specification of both policies is done by extending WS-Policy. Since the specification does not impose limits on the kind of allowed assertions, as already said, for the definition of the policies we have decided to adopt semantically enriched terms. This, of course, will enable semantic-based procedures of policy comparisons.

In order to cope with existing policy specifications, we developed a module able to transform a standard WS-Policy containing WS-SecurityPolicy assertions into a policy expressed by using semantic concepts belonging to our ontology. This way, we are able to integrate in our framework existing systems.

The requestor's and the provider's policies are compared by overlapping the requirements defined in one policy on the capabilities expressed in the other, and vice versa. This way every possible requirement-capability pair is assigned a match level and, in the end, the overall match level between the requestor and the provider is obtained. Since requirements and capabilities are expressed in a semantic form, the Matchmaker will invoke the Reasoner component to perform the semantic match. To accomplish its task, the Reasoner makes use of the *Security Ontology*, which is an ontology that describes concepts from the security domain, expressly created to let the Reasoner evaluate the relationships among the semantic concepts that populate the policies.

4 SEMANTIC MODEL

In this section the semantic model proposed by our approach is described. An ontology was developed to represent main concepts from the security domain.

Such ontology will be used to define concepts of security within the policies. Also, we developed a procedure which is able to automatically map security concepts expressed in the WS-SecurityPolicy specification onto concepts of our security ontology. This procedure enables providers and customers adopting the WS-SecurityPolicy specification to seamlessly use our framework. Finally, to help customers finding the providers that best suite their security requirements, a semantic discovery engine was developed which is capable of reasoning on customers' requests and suggesting highly matching providers.

4.1 Security and Policy Ontology

To support the semantic description of security requirements and capabilities two ontologies have been defined: a *security ontology*, that describes security related concepts like protocol, algorithm, credential, and a *policy ontology* that defines the concept of the policy and its characterization in term of requirements and capabilities. In the following more details about the ontologies are provided. We point out that it is not among the objectives of this work to provide an exhaustive view of all concepts that populate the domain of security.

The policy ontology is a very short ontology that defines the concept of policy in the context of our architecture. A policy is nothing but a list of requirements and capabilities. In the OWL formalization, the Policy Class and the related properties *hasRequirement* and *hasCapabilities* have been created.

For what concerns the security ontology we takes inspiration from the solutions proposed in literature. As an example, the ontology proposed in (Garcia and Felgar de Toledo, 2008), that makes explicit references to the WS-Security's nomenclature, addresses the problem of security when messages are exchanged among web services. In (Kim et al., 2005) the proposed ontology covers most of the concepts of the security domain: despite it was defined to address security aspects at a very high levels, there are some constructs expressly designed to semantically annotate web services. In this work we extended and refined a preliminary ontology (Di Modica and Tomarchio, 2011a), in order to adequately represent security concepts from the WS-SecurityPolicy specification.

A graphic overview of our proposed security ontology is depicted in Figure 2: the main concepts are *Objective, Protocol, Algorithm, Credential, Security scope* and *Target*.

A security *Objective* is a high level abstraction by which we mean a particular security service offered by the system. The following services can be

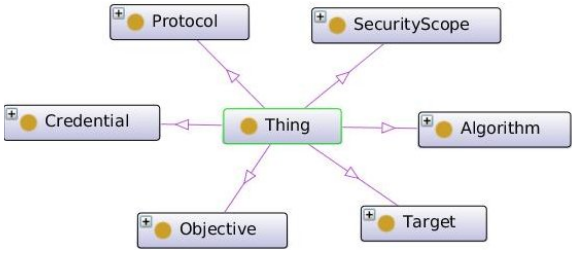

Figure 2: Security ontology: top level classes.

offered: authentication, authorization, confidentiality, integrity and non-repudiation.

A *Protocol* is a set of rules agreed by parties that need to communicate to each other. In the context of security a protocol makes use of tools, like algorithms and credentials, in order to accomplish an objective. In Figure 3, the different protocol categories are represented: we may distinguish encryption, signature, transport, authentication, access control and key management protocols. Each of these categories may contain several individuals, reflecting different security protocols such as, for example, XML-Enc, XML-Dsig, SAML, XACML, and so on.

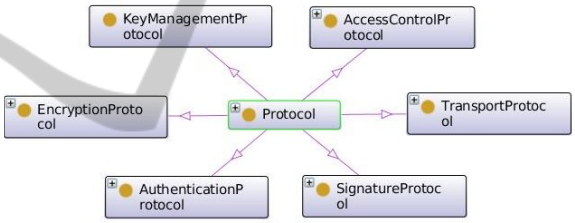

Figure 3: The Protocol class.

An *Algorithm* is a procedure for solving problems that employs a finite number of steps. In the literature several security algorithms have been proposed. We can cite, for instance, the category of encryption and that of authentication algorithms. For the former, another categorization can be proposed according to the type of key used for the encryption: symmetric (e.g. DES, 3DES, AES) and asymmetric (e.g. RSA). Among the authentication algorithms, instead, we identify those that employ hash functions (e.g. SHA, MD5) and those using MAC functions (e.g. HMAC, CBC-MAC). We also included the categories of Key derivation and Key wrapping algorithms. The resulting security algorithm branch is depicted in Figure 4.

Credentials play an important role in information systems that require authentication. Again, there are several categories of credentials, among which we can cite the biometric (fingerprint, voice), electronic (login, password, encrypted keys, certificates), and physical (smartcard, passport, identity card).

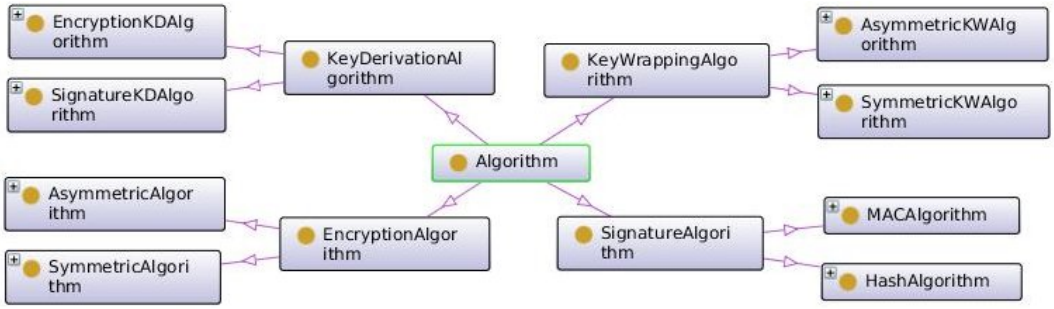

Figure 4: The Algorithm class.

The *Security scope* class allows to differentiate between security concepts related to a single host and security in communication (like message, transport or network level security).

Finally, the *Target* class allows to model the target to which the specific security mechanism is to be applied: currently the only subclass is *message parts* which models all of the elements that can be found in a message according to the WS-Security specification.

The concepts are related to each other within the ontology, and some properties (not shown in figures for the sake of clarity) have been also defined:

- a protocol supports one or more security objective (*hasObjective* property).
- a protocol makes use of one or more security algorithms (*hasAlgorithm* property);
- a protocol requires one or more credentials (*reqCredential* property);
- a protocol may be applied to a security scope (*hasSecurityScope* property);
- a protocol may protect a specific target (*protectsPart* property).

A policy written according to our ontology has a standard WS-Policy structure: the wsp:Policy, wsp:ExactlyOne and wsp:All tags are present and have the same meaning than the standard specification. The only constraint in order to allow a correct working of our matching algorithm is that the policy should be normalized: it has to contain only one wsp:ExactlyOne element which, in turn, may contain any number of wsp:All elements. Each element contained in an assertion is semantically annotated with concepts of our security ontology: in addition it should be specified if the element represents a requirement or a capability. In the following we give an example of a policy containing only one alternative:

```
<wsp:Policy
    ...namespace definition...
  <wsp:ExactlyOne>
    <wsp:All>
      <security:LoginProtocol rdf:ID="requirement0" >
        <security:requiresCredential
           rdf:resource="http://www.semanticweb.org/
              ontologies/security.owl#OneTimePassword"
            security:isMainCredential="false"/>
      </security:LoginProtocol>
      <security:Https rdf:ID="requirement1" >
        <security:hasSecurityScope
           rdf:resource="http://www.semanticweb.org/
             ontologies/security.owl#
               TransportLevelSecurity"/>
        <security:requiresCredential
           rdf:resource="http://www.semanticweb.org/
             ontologies/security.owl#Timestamp"/>
        <security:hasEncryptionAlgorithm
           rdf:resource="http://www.semanticweb.org/
             ontologies/security.owl#AES-256"/>
        <security:hasSignatureAlgorithm
           rdf:resource="http://www.semanticweb.org/
             ontologies/security.owl#SHA-1"/>
      </security:Https>
    </wsp:All>
  </wsp:ExactlyOne>
</wsp:Policy>
```

The policy file starts with the wsp:Policy element. The next element is wsp:ExactlyOne, which contains just one wsp:All element. The policy is already normalized and contains one assertion. Two requirements are contained in this assertion: security:LoginProtocol, which specifies a login protocol, and security:Https, which specifies a protocol for the communication. The latter is better described by specifying some of the used algorithms, the provided security level and the need for a timestamp. All these properties are specified using concepts defined in the previously described security ontology.

4.2 Policy Mapping

We developed a framework module, called **Transformer**, which is capable of transforming a policy, written in WS-Policy and containing assertions specified through the WS-SecurityPolicy, into an equivalent policy whose assertions are semantic concepts

defined through the above discussed security ontology.

Let P_{WSSP} be a WS-Policy compliant policy also containing WS-SecurityPolicy assertions. The Transformer implements a procedure that iterates on P_{WSSP}'s alternatives. For each iterated alternative, its semantic equivalent is returned in the case that the alternative is expressed in WS-SecurityPolicy; otherwise, the alternative is returned unchanged. The Transformer is fed with a Map whose keys are the basic WS-SecurityPolicy keywords and values are their equivalent semantic concepts: the Map will be used to semantically enrich the policy document.

In a preliminary step the Transformer parses P_{WSSP} and creates a tree of nodes representing the WS-SecurityPolicy alternatives. Secondly, the tree is visited to map combinations of alternatives onto new "equivalent" alternatives. Finally, the original WS-SecurityPolicy terms contained in the just created alternatives are replaced by semantic concepts according to the instructions provided by the previously mentioned Map. In the following, an excerpt of that Map is reported in Table 1.

4.2.1 Mapping Examples

The Transformer's main task, then, is to find the correspondence between WS-SecurityPolicy assertions and security ontology's concepts. At the current stage, not all elements contained in an alternative have a corresponding semantic concept. The reason is twofold. First, the developed ontology focuses just on main security concepts (integration of other concepts is planned in the near future); second, some WS-SecurityPolicy elements deal with technical aspects which are not specifically bound to security (e.g., the layout order). WS-SecurityPolicy terms that have no correspondence in the ontology are simply ignored by the Transformer.

Starting point for the mapping procedure is the *binding* assertions. Depending on the token found on those assertions, it is possible to derive the protocols to be used. For instance, if the token is one of wsp:ProtectionToken, wsp:InitiatorToken or wsp:RecipientToken then both cipher and digital sign protocols will have to be used. According to the key-value map introduced earlier, semantic individuals such as *XML-Dsig* and *XML-Enc* will be chosen to semantically represent this option. Tokens such as wsp:EncryptionToken, wsp:InitiatorEncryptionToken or wsp:RecipientEncryptionToken only express the need for a cipher protocol, and will be mapped onto *XML-Encryption*, while for tokens like wsp:SignatureToken, wsp:InitiatorSignatureToken and wsp:RecipientSignatureToken the *XML-DigitalSignature* protocol will correspond. Finally, wsp:TransportToken will be mapped onto the *Https* protocol.

The symmetric and asymmetric bindings guarantee protection at the message level, and will be mapped on the *hasSecurityScope* property set to the MessageLevelSecurity value. The transport binding provides protection at the transport level: in that case, that assertion will be mapped on the *hasSecurityScope* property whose value will be set to TransportLevelSecurity.

In the case of wsp:ProtectionToken or wsp:InitiatorToken, the specific type of token used to guarantee protection is also specified. Among all possible types, the mapping process supports the following: "X509", "SAML", "Username" and "Https". The latter, in particular, is the only one allowed as a transport token. The first three tokens specify that authentication credentials must be provided. While "SAML" and "Username" have a direct correspondence with protocols described by the ontology (*SAML* and *login* protocol respectively), "X509" does not have a direct correspondence; so instead of mapping it onto a protocol it is mapped onto the authentication security Objective. Furthermore, in the case of "X509" the property indicating that an X509 certificate credential is required must be added to the used protocols. In the case of "Username", instead, a password may be required in the assertion: if so, the *OneTimePassword* credential will be used. Finally, for a "SAML" token we must add to the protocols set a credentials to require the use of SAML as a guarantee on the identity. An important assertion is the AlgorithmSuite, which specifies the algorithms adopted in the security environments. Depending on the assertion's settings, those algorithms must be added to the corresponding protocols. For instance, if the assertion is set to "Basic256", 1) the hasEncryptionAlgorithm property must be added to the XML-Enc protocol, with its value set to AES-256, and 2) the hasSignatureAlgorithm property must be added to the XML-Dig protocol, with the value set to SHA-1. Likewise, if there are assertions specifying the use of key derivation within the token, the corresponding algorithm will be added.

4.3 Semantic Discovery

We implemented a semantic discovery engine capable of searching for correspondences between security requirements and capabilities. Suppose that both the customer (requestor) and the provider (service) have expressed their requirements and capabilities in their own policies. The engine operates this way: a) the

Table 1: Mapping between WS-SecurityPolcy terms and Semantic Concepts.

WS-SecurityPolicy term	Semantic Concept	
Https	http://www.semanticweb.org/ontologies/security.owl#Https rdf:ID requirement/capability	
XML-Enc	http://www.semanticweb.org/ontologies/security.owl#XML-Enc rdf:ID requirement/capability	
.....	.....	
Login	http://www.semanticweb.org/ontologies/security.owl#LoginProtocol rdf:ID requirement/capability	
Authentication	http://www.semanticweb.org/ontologies/security.owl#Authentication rdf:ID requirement/capability	
.....	.....	
TransportLevel	http://www.semanticweb.org/ontologies/security.owl#hasSecurityScope http://www.semanticweb.org/ontologies/security.owl#TransportLevelSecurity	rdf:resource
AES256	http://www.semanticweb.org/ontologies/security.owl#hasEncryptionAlgorithm http://www.semanticweb.org/ontologies/security.owl#AES-256	rdf:resource
AES192	http://www.semanticweb.org/ontologies/security.owl#hasEncryptionAlgorithm http://www.semanticweb.org/ontologies/security.owl#AES-192	rdf:resource
AES128	http://www.semanticweb.org/ontologies/security.owl#hasEncryptionAlgorithm http://www.semanticweb.org/ontologies/security.owl#AES-128	rdf:resource
.....	.....	
EncryptedBody	http://www.semanticweb.org/ontologies/security.owl#encryptsPart http://www.semanticweb.org/ontologies/security.owl#Body	rdf:resource
SignedBody	http://www.semanticweb.org/ontologies/security.owl#signsPart http://www.semanticweb.org/ontologies/security.owl#Body	rdf:resource
RequiredBody	http://www.semanticweb.org/ontologies/security.owl#requiresPart http://www.semanticweb.org/ontologies/security.owl#Body	rdf:resource
.....	.....	
SignSignature	http://www.semanticweb.org/ontologies/security.owl#signsPart http://www.semanticweb.org/ontologies/security.owl#Signature	rdf:resource
SignSignatureToken	http://www.semanticweb.org/ontologies/security.owl#signsPart http://www.semanticweb.org/ontologies/security.owl#SigningToken	rdf:resource
SignSupportingToken	http://www.semanticweb.org/ontologies/security.owl#signsPart http://www.semanticweb.org/ontologies/security.owl#SupportingToken	rdf:resource

requirements of the service are compared to the capabilities of the requestor; b) the capabilities of the service are compared to the requirements of the requestor. In order for the comparison process to have a positive outcome, the two following conditions must hold:

- the capabilities expressed in the service's policy must meet the requirements expressed in the requestor's policy;
- the requirements expressed in the service's policy must be met by the capabilities expressed in the requestor's policy.

The first step of the matchmaking process has to assign a match level to each requirement-capability pair. The comparison will leverage on the well-known concepts of *exact*, *subsume*, *plugin*, and *no match* (Paolucci et al., 2002), and has been carried out by means of techniques that have already been explored by authors in the context of supply-demand matchmaking in cloud markets (Di Modica and Tomarchio, 2012).

In the second step the overall match between the two policies is evaluated: the objective is to find the capability that matches at best for each requirement. The overall match is then defined to be the minimum among the individual match levels evaluated in the first step for each requirement-capability pair. Further details on the policy matching can be found in a previous work (Di Modica and Tomarchio, 2011b), even though in that work a preliminary version of the ontology is employed.

5 CASE STUDY

In this section, an simple case study of the overall behaviour of the semantic security framework is presented. The overall scenario is the classic scenario (see also Figure 1) where a client (service requestor) is looking for a service that provides a specific functionality (expressed through the WSDL specification), with a specific security policy which has been expressed using WS-SecurityPolicy assertions; in addition, the client expresses also security requirements (within its own security policy) that the service provider must meet. The client searches the UDDI registry in order to retrieve service information. The registry, besides providing the information necessary to invoke the service (the WSDL descriptor), also returns the provider's associated security policy. The Transformer transforms this policy into a semantically annotated policy.

At this point, the Matchmaker and the Reasoner come into play, allowing the client to verify the fulfillment of its security requirements. The Matchmaker is fed with the client's and the provider's policies. It

then runs the matching algorithm and, with the help of the Reasoner, evaluates the match level between the policies. Finally the client, based on the obtained value, decides whether to invoke the service or not.

5.1 Security Policies Definition

The service provider expresses its security policy by using the standard WS-Policy framework, adopting WS-SecurityPolicy terms. The specific policy has been taken by the example policies of WS-SecurityPolicy specification (OASIS, 2012). The scenario provides for a requestor that wants to access a service anonymously, and a service provider that presents its credentials. In order to guarantee message confidentiality and authentication, the security protocol will use symmetric ephemeral keys, generated by the provider and exchanged through asymmetric encryption.

Provider Policy. The provider policy is composed by multiple items: one at the endpoint level, and two for input and output messages respectively. At the upper level the the WS-SecurityPolicy elements are: SymmetricBinding, EncryptedParts, SignedParts and Wss11. Each element, in its turn, is characterized by several properties (not described here for space reason) better detailing the requirements and the capabilities of the provider.

The Transformer module takes as input this policy and provides a corresponding policy annotated with semantic concepts. The resulting policy, which is shown in Listing 1, contains two requirements and three capabilities. In order to improve the readability of the listing, the namespace *http://www.semanticweb.org/ontologies/security* was removed from each rdf:resource term.

Listing 1: Semantically enriched Policy.

```
<wsp:Policy>
 <wsp:ExactlyOne>
 <wsp:All>
  <security:Authentication rdf:ID="capability0
      "/>
  <security:XML-Enc rdf:ID="capability1">
   <security:hasKeyDerivationAlgorithm
       rdf:resource="#PSha1L192"/>
   <security:requiresCredential rdf:resource="
       #X_509Certificate"/>
   <security:hasSecurityScope rdf:resource="#
       MessageLevelSecurity"/>
   <security:requiresCredential rdf:resource="
       #Timestamp"/>
   <security:hasEncryptionAlgorithm
       rdf:resource="#AES-256"/>
   <security:encryptsPart rdf:resource="#Body"
       />
  </security:XML-Enc>
  <security:XML-Enc rdf:ID="requirement0">
   <security:hasSecurityScope rdf:resource="#
       MessageLevelSecurity"/>
   <security:requiresCredential rdf:resource="
       #Timestamp"/>
   <security:hasEncryptionAlgorithm
       rdf:resource="#AES-256"/>
   <security:encryptsPart rdf:resource="#Body"
       />
  </security:XML-Enc>
  <security:XML-Dsig rdf:ID="capability2">
   <security:hasKeyDerivationAlgorithm
       rdf:resource="#PSha1L256"/>
   <security:requiresCredential rdf:resource="
       #X_509Certificate"/>
   <security:hasSecurityScope rdf:resource="#
       MessageLevelSecurity"/>
   <security:requiresCredential rdf:resource="
       #Timestamp"/>
   <security:hasSignatureAlgorithm
       rdf:resource="#SHA-1"/>
   <security:signsPart rdf:resource="#Body"/>
  </security:XML-Dsig>
  <security:XML-Dsig rdf:ID="requirement1">
   <security:hasSecurityScope rdf:resource="#
       MessageLevelSecurity"/>
   <security:requiresCredential rdf:resource="
       #Timestamp"/>
   <security:hasSignatureAlgorithm
       rdf:resource="#SHA-1"/>
   <security:signsPart rdf:resource="#Body"/>
  </security:XML-Dsig>
 </wsp:All>
 </wsp:ExactlyOne>
</wsp:Policy>
```

The requirements are the XML-Encryption and the XML-DigitalSignature protocols, which are also described with some properties imposing the specific encryption algorithm to be used and the need to encrypt and sign the Body of the message.

The capabilities are the XML-Enc, XML-Dsig and the Authentication assertions. The first two are similar to what already expressed into the requirements, except for some specific algorithms (the latter represents a security objective ensured by the X.509 certificate).

Requestor Policy. The service requestor expresses its policy in terms of requirements and security capabilities too. In order to show the effectiveness of the semantic matching algorithm, it was manually created from the service provider policy, by inverting requirements with capabilities. Then, some elements and properties have been modified in order to show the reasoning capabilities of the framework. In particular:

- the first requirement was changed into the SAML protocol, that is, into a more specific requirement;
- in the second requirement, the hasSecurityScope property was changed into CommunicationSecurity;
- third requirement and first capability were left unchanged;
- in the second capability, the hasSignatureAlgorithm property was changed into hasAlgorithm, that is, into a less specific property.

5.2 Matching Results

The matching algorithm tries to combine the client requirements with the server capabilities (and vice versa) for each couple of policy alternatives. Since in this example there is only one alternative, there are no iterations among alternatives. The match level found by the reasoner are the following:

Service Requirements versus Requestor Capabilities

- *Possible Match*: it derives from the comparison between the Authentication and SAML assertions, since the requirement is more specific than the capability
- *Close Match*: it derives from the comparison between the XML-Enc assertions which are identical in both policies, but have different properties. In particular the requirement property CommunicationSecurity is more general than the MessageLevelSecurity property of the capability.
- *Perfect Match*: it derives from the comparison between the XML-Dsig assertions which are identical in both policies, and also have identical properties.

Service Capabilities versus Requestor Requirements

- *Perfect Match*: it derives from the comparison between the XML-Enc assertions which are identical in both policies, and also have identical properties.
- *Possible Match*: it derives from the comparison between the XML-Dsig assertions which are identical in both policies, but have different properties. In particular, the requirement property hasSignatureAlgorithm is more specific than the capability property hasAlgorithm.

The policies' overall match level is given by the lowest among the found levels: in this case it is a *Possible match*. The Matchmaker communicates to the client that the selected service may satisfy its security requirements. Since the match is not Perfect, the client is required to provide additional information or negotiate with the service provider before actually invoking the service. Further details on the negotiation phase that may carried out at this step can be found in (Liccardo et al., 2012).

It should be noticed that, if a traditional (syntactic) approach had been used for the policy comparison, the result would have led us to consider the two policies as incompatible.

6 CONCLUSIONS

Security has been acknowledged to be one of the primary concerns preventing a wide adoption of cloud computing among enterprises. In interoperable contexts like those fostered by SOA, where networks of trustiness must be set up among service providers, defining, modeling and matching service security policies is a crucial problem that needs to be faced. Existing proposals leveraging on a syntactic approach like the WS-Policy are not very well suited for these heterogeneous and dynamic scenarios. In this paper we proposed to integrate the WS-Policy model with semantic annotations, thus enabling semantic matching capabilities, allowing to go beyond the strict syntactic intersection of policy assertions.

We then developed a security ontology which allows the relationships among main security concepts to be modeled. Leveraging on this ontology, we designed and implemented a framework for the matchmaking of what is requested on the customer side and what can be provided on the provider side in terms of security features. The framework is also compatible with syntactic policies defined through the WS-SecurityPolicy specification. A procedure was devised to transform the policy's syntactic terms into semantic concepts of our security ontology. A simple use case example was also proposed to show the viability of the approach and the actual limits of the pure syntactic-based approaches.

Future works will be aimed to test the framework with interoperable scenarios (B2B scenarios) and to interact with even more complex policies.

ACKNOWLEDGEMENTS

The work described in this paper has been partially supported by the *"Cloud4Business"* project funded by the PO FESR 2007/2013 programme of Sicily region.

REFERENCES

Damianou, N., Dulay, N., Lupu, E., and Sloman, M. (2001). The ponder policy specification language. In *Proceedings of the International Workshop on Policies for Distributed Systems and Networks*, POLICY '01, pages 18–38, London, UK. Springer-Verlag.

Di Modica, G. and Tomarchio, O. (2011a). Semantic annotations for security policy matching in WS-Policy. In *SECRYPT 2011 - Proceedings of the International Conference on Security and Cryptography*, pages 443–449, Seville (Spain).

Di Modica, G. and Tomarchio, O. (2011b). Semantic Security Policy Matching in service oriented architectures. In *Proceedings - 2011 IEEE World Congress on Services, SERVICES 2011*, pages 399–405, Washington DC (USA).

Di Modica, G. and Tomarchio, O. (2012). A semantic discovery frame work to support supply-demand matchmaking in cloud service markets. In *CLOSER 2012 - Proceedings of the 2nd International Conference on Cloud Computing and Services Science*, pages 533–541, Porto (Portugal).

Garcia, D. Z. G. a. and Felgar de Toledo, M. B. (2008). Ontology-Based Security Policies for Supporting the Management of Web Service Business Processes. In *2008 IEEE International Conference on Semantic Computing*, pages 331–338. Ieee.

Kagal, L., Finin, T., and Joshi, A. (2003). A policy language for a pervasive computing environment. In *Proceedings of the 4th IEEE International Workshop on Policies for Distributed Systems and Networks*, POLICY '03, pages 63–, Washington, DC, USA. IEEE Computer Society.

Kim, A., Luo, J., and Kang, M. (2005). Security ontology for annotating resources. In *On the Move to Meaningful Internet Systems 2005: CoopIS, DOA, and ODBASE*, pages 1483–1499. Springer.

Liccardo, L., Rak, M., Di Modica, G., and Tomarchio, O. (2012). Ontology-based Negotiation of Security Requirements in Cloud. In *Computational Aspects of Social Networks (CASoN), 2012 Fourth International Conference on*, pages 192 –197, Sao Carlos (Brasil).

OASIS (2012). WS-SecurityPolicy 1.3. OASIS Standard. Available at http://www.oasis-open.org/specs/.

Paolucci, M., Kawamura, T., Payne, T. R., and Sycara, K. P. (2002). Semantic matching of web services capabilities. In *ISWC '02: Proceedings of the First International Semantic Web Conference on The Semantic Web*, pages 333–347, London, UK. Springer-Verlag.

Papazoglou, M. P. and van den Heuvel, W.-J. (2007). Service Oriented Architectures: approaches, technologies and research issues. *VLDB Journal*, 16(3):389–415.

Phan, T., Han, J., Schneider, J., Ebringer, T., and Rogers, T. (2008). A survey of policy-based management approaches for Service Oriented Systems. In *Software Engineering, 2008. ASWEC 2008. 19th Australian Conference on*, pages 392–401. IEEE.

Speiser, S. (2010). Semantic Annotations for WS-Policy. In *IEEE International Conference on Web Services (ICWS 2010)*, pages 449–456. IEEE.

Sriharee, N., Senivongse, T., Verma, K., and Sheth, A. (2004). On using ws-policy, ontology, and rule reasoning to discover web services. In *Intelligence in Communication Systems*, number May 2004, pages 246–255. Springer.

Tonti, G., Bradshaw, J., Jeffers, R., Montanari, R., Suri, N., and Uszok, A. (2003). Semantic Web languages for policy representation and reasoning: A comparison of KAoS, Rei, and Ponder. In *International Semantic Web Conference (ISWC2003)*, pages 419–437, Florida (USA). Springer.

Uszok, A., Bradshaw, J., Jeffers, R., Suri, N., Hayes, P., Breedy, M., Bunch, L., Johnson, M., Kulkarni, S., and Lott, J. (2003). Kaos policy and domain services: Toward a description-logic approach to policy representation, deconfliction, and enforcement. In *Proceedings of the 4th IEEE International Workshop on Policies for Distributed Systems and Networks*, POLICY '03, pages 93–, Washington, DC, USA. IEEE Computer Society.

Verma, K., Akkiraju, R., and Goodwin, R. (2005). Semantic matching of Web service policies. In *Semantic Web Policy Workshop (SDWP 2005)*.

W3C (2007). Web services policy 1.5 - framework. W3C Recommendation. Available at http://www.w3.org/TR/ws-policy/.

Zheng-qiu, H., Li-fa, W., Zheng, H., and Hai-guang, L. (2009). Semantic Security Policy for Web Service. In *2009 IEEE International Symposium on Parallel and Distributed Processing with Applications*, pages 258–262. Ieee.