Automatic Security Policy Composition for Composite Web Services

B. Muruganantham and Dr. K. Vivekanandan

Abstract
Web services contain atomic and composite services. The composite service will invoke other composite or atomic services. The problem here is while composite service invokes other services, the policy agreements and type of security used should be matched with the invoked web services. As a result there will be inconsistency among the web services and also there will be lack of message protection policy. To solve this problem, developers have to define policy of composite service by hand by referring to the policies of the invoked web services in the composite process. However, it is very hard to complete a policy composition without any inconsistencies, because the process definitions and security policies are complex and it is not clear how to compose policies to maintain consistency. So, a proper policy composition engine which can eradicate the inconsistency between web services is needed. Hence in this project, a policy composition engine which is able to verify policies between web services that has different type of access control policy and message protection policy compared to composite web service is proposed. Meanwhile a revised composite policy might have redundant requirements. In order to avoid this problem, the redundant representations are merged or removed in a supervised manner that examines the definitions of all implemented services. The aim of is to provide consistency between atomic web service and composite web service by providing policy composition rule to develop applications by combining different services. Its main purpose is to erase application boundaries and technology differences.

The main issue is to combine these services, as the applications are opened up. Web service composition (WSC) is an ongoing technology to integrate multiple distributed services to fulfill special business object and has gained lots of interest from academic and industry. Security is one of the most critical issues for the enterprise applications. To secure the application, multiple security policies will be necessary such as the exchanged messages should be signed and encrypted; only an authorized employee of the retailer can be able to invoke the service, etc. One service might have multiple policies, and the external services could also have their own security policies. An application developer needs to assure that the retailer service is secure end-to-end, and there are two ways of configuring the security policies for the composite service. When a developer assembles existing services that have their own policies, then the developer could define a new policy for the overall composite service, which is the top-down approach. Alternatively, the security policy

Introduction

A Web service is a software system designed to support interoperable application-to-application interactions over the Internet. Web services rely on a set of XML standards such as Universal Description, Discovery and Integration (UDDI), Web Services Description Language (WSDL), and Simple Object Access Protocol (SOAP). A service-oriented architecture (SOA) is defined as a design pattern in which services are provided by application components over a network, to other components via a communications protocol or it can also be defined as a concept for building applications by assembling services that are the components of business functions. Typically, an SOA application is implemented as a composite service that invokes external services. The processes and service invocations are defined in a process language such as BPEL and developers can rebuild applications by changing only the process definitions without changing the service implementations. One benefit of an SOA application is its flexibility to adapt to changing business processes. It allows various ways...
of the composite service could be created from the security policies of the invoked external services, which is the bottom-up approach. In both cases, the policies of the external services should also be satisfied as long as the composite policy is satisfied. A developer should check that there are no policy inconsistencies among the composite service and the invoked external services.

The security requirements are specified as security policies for the composite service, but there is no clear way to define the policies for the composite service. When an application developer assembles existing services that have their own security policies, the main problem lies here in defining the composite policies to ensure that no inconsistencies exist with the existing policies. There are several kinds of security policies, such as for message protection and for access control, and hence we need to compose these policies separately to create the policies for the composite service. The web services contain atomic and composite services. A composite service can be defined as a service whose implementation calls other services, an atomic service is a service whose implementation is self-contained and does not invoke any other services.

A composite service acts as both a service provider and as a service consumer. It acts as a service provider of the (composite) service and as a service consumer of its child services. The composite service is able to invoke other composite or atomic services. As a result there will be inconsistency among the web services and also there will be lack of message protection policy. To solve this problem, developers have to define policy of composite service by hand by referring to the policies of the invoked web services in the composite process.

The rest of the paper is organized as follows. The next section discusses a brief literature survey of existing theories and work done so far. Section 3 discusses the consistency check between two policies written in WS-Security Policy. Their motivation is a policy composition for different departments, and the composition preferences should be defined by the policy writers. On the other hand, proposed system addresses composition policies that are invoked in a composite process, define the process-aware composition rule, so a composition result depends on a process definition that policies are used.

Proposed system approach is not a static composition of policy representations, and composes security policies from the perspective of protecting a whole composite process. This is one of the features of proposed approach (Bhargavan, 1992), defined a formal semantics of WS-Security, provides a tool for policy validation to prevent security vulnerabilities. The objective is a formal policy validation and their logic representation. Proposed system objective is defining the policy composition rules which can be applied to all composite services. It implements policy composition method declaratively as a prototype. Proposed system addressed the way of policy composition, so defining a security policy formally is not our objective.

Security policy composition has been studied in a variety of fields (Li, 2006), studied policy consistency in Web service compositions written in BPEL, with motivations similar to proposed system. This system studied privacy policies and approaches to verify policies in graph transformations (Yang, 2007), analyzed security policy integration between different application domains. This system provides requirements for security integration and patterns. The system targeted policy is access control policy, but they do not mention the concrete policy representation (Srivatsa, 2007), presented an access control model and techniques for specifying and enforcing access control rules for Web service compositions. This system introduces composite roles and principles and specified access control policies using pure-past linear temporal logic (Charfi, 2005), proposed an aspect-oriented approach to specify security policies for Web service compositions. This system implements a set of aspects in AO4BPEL that is an aspect-oriented extension to BPEL. This system specifies authorization information and constraints using XACML eXtensible Access Control Markup Language (XACML) Version 2.0, http://docs.oasis-open.org/xacml/2.0/access_control-xacml-2.0-core-spec-os.pdf, 2011, for the BPEL process, which is similar to proposed approach. There are many related projects discussing security policies for processes, especially access control policies. Also, proposed system discusses how to apply our policy composition to access control policy in (Satoh, 2008).

The policy composition rule in proposed approach is unique and data centric, and our
approach can be applied to other kinds of policies, such as access control policy and specific policy for composite service. This is one of advantages of the proposed system.

Proposed method:
The proposed system derives policy composition rule that can be applied to any composite processes, and offer a security policy composition mechanism. The composite processes and policies are transformed into logic representations, and a composite policy is inferred from the composition rules. It can use two approaches to create a composite security policy: top-down or bottom-up. In the top-down approach, a developer can specify composite policies without regard to the existing policies. The proposed mechanism verifies the specified security policies are consistent with the existing policies, and the developer can confirm that the specified composite policy will work properly. In contrast, the bottom-up policy composition can infer the necessary security requirements from the existing policies of the external services, and a consistent composite policy can be created from the inferred requirements. The system obtains requirements from users and eliminates redundant requirements.

This system consists of two atomic services and a composite service, Hotel reservation web service with its service definitions, access control policy and message control policy as an atomic service and Airline reservation web service with its service definitions, access control policy and message control policy as another atomic service in addition there will be a composite web service called travel reservation the access control policy and message control policy is built on the policies based on atomic web service. Now there will be a policy composition engine that verifies the policies of composite web service with the atomic web service. The policy composition engine checks for the inconsistency between composite and atomic service, there will be a flag if there is inconsistency prevails then the flag is set to “true” if there is no inconsistency then the flag is set to “false”. In case on inconsistency the policy composition engine examines atomic service and composite service then solves its inconsistency, then it examines composite policy for redundant requirements, if there is any redundancy then it will be merged or removed so the resultant composite policy is free from redundant message protection and access control requirements. So after this process the web service are loaded into servers, then the user login to web service application and performs their desired operation.

When a developer assembles existing services that have their own policies, then the developer could define a new policy for the overall composite service, which is the top-down approach. Alternatively, the security policy of the composite service could be created from the security policies of the invoked external services, which is the bottom-up approach. In both cases, the policies of the external services should also be satisfied as long as the composite policy is satisfied. A developer should check that there are no policy inconsistencies among the composite service and the invoked external services.

![System Architecture](image)

Fig. 1: System Architecture.
In the Fig. 1, it represents the system architecture. Here the input is composite web service and atomic web services. The proposed system creates a number of atomic web services and composite web services with policies, and then there will be a policy composition engine which checks for inconsistency and for redundant variables so after this process the user can login to web service and perform their web service operation. The inconsistent variables in the composite web service are fixed and redundant variables are removed. This system takes advantage by implementing policy composition engine, in this process the atomic service policy and composite service policy are matched and according to the atomic service policy the composite service policy is changed. Thus, it provides consistency to web service policy.

Algorithm:
Step 1: Create Hotel Reservation Web Service as follows:
if(hotel_reservation_web_service==null)
{
  <atomicpolicy>
  <wsp:Policy id="Hotel_signature_policy">
    <sp:SignatureMode>
      <sp:Digest>
        <sp:SignatureAlgorithm>SHA1withRSA
      </sp:SignatureAlgorithm>
    </sp:Digest>
    </sp:SignatureMode>
  </wsp:Policy>
  <wsp:Policy id="Airline_encryption_policy">
    <sp:EncryptionMode>
      <sp:ASymmetric>
        <sp:EncryptionAlgorithm>AES
      </sp:EncryptionAlgorithm>
    </sp:ASymmetric>
  </sp:EncryptionMode>
  </atomicpolicy>

  i. Create service definitions (Request, Booking, Response etc.).
  ii. Define Access Control Policy predicates
  iii. Define Message Protection Policy.
}

Step 2: Create Airline Reservation Web Service as follows:
if(airline_reservation_web_service==null)
{
  <atomicpolicy>
  <wsp:Policy id="Airline_signature_policy">
    <sp:SignatureMode>
      <sp:Digest>
        <sp:SignatureAlgorithm>SHA1withRSA
      </sp:SignatureAlgorithm>
    </sp:Digest>
    </sp:SignatureMode>
  </wsp:Policy>
  <wsp:Policy id="Airline_encryption_policy">
    <sp:EncryptionMode>
      <sp:ASymmetric>
        <sp:EncryptionAlgorithm>AES
      </sp:EncryptionAlgorithm>
    </sp:ASymmetric>
  </sp:EncryptionMode>
  </atomicpolicy>

  i. Create service definitions (Request, Booking, Response etc.).
  ii. Define Access Control Policy predicates
iii. Define Message Protection Policy.
}

Step 3: Create Travel Reservation Composite Service as follows:
if(travel_reservation_composite_service==null)
{
<compositepolicy>
-service name="flighta">
<wsp:Policy id="FlightA_signature_policy">
 .
 .
 </wsp:Policy>
 <service name="flightb">
 <wsp:Policy id="FlightB_signature_policy">
 .
 .
 </wsp:Policy>
 <service name="hotela">
 <wsp:Policy id="HotelA_signature_policy">
 .
 .
 </wsp:Policy>
 <service name="hotelb">
 <wsp:Policy id="HotelB_signature_policy">
 .
 .
 </wsp:Policy>

Step 4: Handling Redundant Policy Requirements:

i. The composite policy is examined to find its redundant requirements.

ii. The redundant requirements are now merged or removed and the resultant composite policy is free from redundant message protection and access control requirements.

iii. Now the web services are loaded into the servers. The web pages for all the external and composite web services are defined.

iv. Now the user accounts are created. Then the user login to the web service application and performs travel reservation operations.

In the first step, first the service definitions for Hotel Reservation web services are created. The service definitions include Request, booking and response. The message protection policies are created for the messages handled by the hotel reservation service. It describes about types of signature methods and encryption used. The encryption methods vary according to the security levels of the hotels. Access control policies are defined which depicts eligibility criteria of persons to invoke the service. Then in the next step, the service definitions for Airline reservation web services are created. It consists of service descriptions for request messages, booking and response. The message protection policies for various types of messages handled by this service are defined. The message protection
policies describe about encryption methods and signature methods needed to protect the messages. Access control policies are defined that depict eligibility criteria of persons to invoke the service. These fields are included in the policy definition. Now in step 3, the service definitions of composite travel reservation services are created. The message protection policies of the service are created based on its Hotel Reservation and Airline Reservation counterpart. Then its access control policies are created with respect to its Hotel Reservation and Airline Reservation counterpart. Then policy composite engine is designed to verify the inconsistencies between policies of composite and atomic web services. The inconsistency is identified if the flag is set to ‘true’. Then the inconsistent variables of composite service are fixed. Here in step 4, the composite policy is examined for redundant requirements. If it has any redundant requirements then it creates problems in SOAP message generation. So such requirements are merged or removed from the composite policy. Now the web services are loaded into web servers. The users register with web service provider and access the services. Then the web services are loaded into the servers and the user logins to the web service application, and perform operations.

Conclusion:
The proposed system takes advantage by implementing policy composition engine, in this process the atomic service policy and composite service policy are matched and according to the atomic service policy the composite service policy is changed. Thus, it provides consistency to web service policy. During this process there might be a chance of redundant variables in composite web service policy so as an enhancement in order to eliminate those redundant variables, the atomic service policy and composite service policy are analyzed and the redundant variables in the composite service policy is eliminated. So the resultant composite policy is free from redundant access control policy and message control policy requirements. Then the web services are loaded into servers, the user now login to web service application and performs their desired operations.

In future, the techniques used in policy composition engine will be improved to support dynamic requirements of web services stored in the repository.

REFERENCES


