Security Analysis for Web Services Compositions

Mohsen Rouached
m.rouached@tu.edu.sa
College of Computers and Information Technology
Taif University, Taif, Saudi Arabia

Abstract— As more organizations adopt Web services for increasingly sensitive, mission-critical data the potential impact of breaches of Web services increases both for individuals and organizations. Increasing impacts can result in a worsening of the risk environment for all parties. Web services security and auditing is therefore an important concern. The current trend toward representing Web services orchestration and choreography via advanced business process metadata is fostering a further evolution of current security models and languages, whose key issues include setting and managing security policies, inter-organizational security issues and the implementation of high level business policies in a Web services environment. In particular, the management and maintenance of a large number of Web services needs appropriate authorization policies to be defined so as to realize reliable and secure Web Services. The required authorization policies can be quite complex, resulting in unintended conflicts, which could result in information leaks or prevent access to information needed. In this paper, we discuss the authorization control for Web services compositions and propose a logic based approach to ensure the control access to such compositions.

Index Terms— Web Services Composition, Security requirements, security policies

1 Introduction

Security analysis for Web services has been the subject of a growing interest for the security research community. The preoccupation in anticipating possible security flaws in the SOA's infrastructures is fundamental for increasing the reliability of SOA's, such that it can be widely adopted, enabling the future Internet of Services.
Composed services are the main contribution the SOA's bring to enterprise business process automation considering the fact that no single service can satisfy users' desires in the majority of service oriented scenarios.
Despite the importance of Web service composition, security issues have not been extensively investigated and security concerns become one of the main barriers that prevent widespread adoption of this new technology.
Compared to existing computer systems, providing security for service oriented environments is much more challenging. The security challenges presented by the Web services approach are formidable. Many of the features that make Web services attractive, including greater accessibility of data, dynamic application-to-application connections, and relative autonomy (lack of human intervention) are at odds with traditional security models and controls.
Although providing security for single Web services is a demanding task, securing service composition process seems to be more challengeable.
Service level security include basic aspects such as Authentication, Authorization (Access Control), Non-repudiation, Data Integrity and Confidentiality. Web service can protect SOAP messages sent over insecure transports by embedding security headers. The WS-Security standard defines how such headers may include signatures, cipher texts and security tokens. There are several emerging specifications of Web service security such as WS-Policy, WS-Trust, WS-Privacy, and WS-Federation, covering various facets of security in the context of Web services. They are built on the top of WS-Security and define enhancements to provide security protection to Web service endpoints and the data communication between them.
Composition Level Security focuses on security issues that may arise during Web service composition. In general, a Web service provider may have security concerns regarding the Web services with which it must cooperate during the composition process. Thus, it is fundamental to support a security conscious composition of Web services, that is, a composition of Web services taking into account the security requirements of each Web service provider and composing only those Web services that are compatible with regards to such requirements. WS-Security and other emerging specifications provide the basic security functionalities, but they do not offer enough support to ensure security in Web service composition. For instance, a first challenge is the definition, the verification, and the enforcement of security policies as the complexity of composite Web services grows. To cope with this complexity, it is useful to design a conceptual

model that gives a structured way to think about security policies.

Another challenge is that non-functional concerns should be addressed by external specifications for a better separation of concerns and for more modular composition specification. Furthermore, mixing the specification of the core logic of the composition with specifications of security features and other non-functional concerns into one unit would make the composition specification too complex and hard to maintain and evolve. In order to realize reliable and secure Web services, it is important to authenticate and authorize the users appropriately. For instance, to prevent problems such as an information leak, suitable access control is needed for the users who access the resources through Web services. By using the standard policy description languages such as WS-Policy, WSPL and XACML [10], it is possible to realize complicated access control for Web services. However, the overall structure of these policies can become very complex, reflecting the complexity of the Web services and roles involved. There is an increased risk that an administrator mistakenly defines conflicting policies which, if the wrong choice is made, result in information leak or prevent access to critical information in an emergency situation. Defining and verifying security policies manually is error-prone and cumbersome. An automated analysis is necessary to ensure that the policies are conflict-free when defined at first and as new security concerns are added and removed.

In this paper, we propose a formalism based on the standard Event Calculus (EC)[9] to specify authorisation policies for Web services composition. The paper is structured as follows. Section 2 introduces the different issues associated with security in Web services compositions. In Section 3, we present how we specify the authorization policies using the EC, and how the authorization conflicts can be defined and checked. The encoding of the proposed specification is presented in Section 4. Section 5 is dedicated to related works. Finally, Section 6 concludes and outlines future work.

2 Security Requirements for Web services compositions

Service Oriented Architecture allows for considerably more complex interaction models than the classical client/server model, including symmetric peer-to-peer interactions. However, SOA is built on an insecure, unmonitored, and shared environment, which is open to events such as security threats. This may result in conflicts because the open architecture of Web services makes it available to many parties, who may have competing interests and goals. The information processed in Web services might be commercially sensitive, so it is important to protect it from security threats such as disclosure to unauthorized parties. The research area of Web services security is challenging, as it involves many disciplines, from authentication/encryption to access management/security policies. Security concerns and the lack of security conventions are the major barriers that prevent many business organizations from implementing or employing Web services. Such security concerns are also crucial when composing Web services. Similar to the dynamic composition of web services there is a need for a dynamic and consistent composition of the related security policies of all participants. There are several unique security-related characteristics that need to be addressed to develop secure business processes with Web services

To illustrate the security requirements of Web service compositions, [15] considered an example that consists of a car manufacturer with several factories spread across the world. In this scenario, the car maker integrates its IT infrastructure with a supplier and a bank using Web services. Items are ordered from the supplier and consequently the bank offer a payment Web service to pay the transaction. The process is deployed by the IT department of the car manufacturer and the resulting Web service can be accessed from any production site that goes out of stock.

In this scenario, the operations of the partner Web services (supplier, bank) require *authentication* since it is not acceptable that anyone who knows the URL of the service or finds it in a UDDI registry can place orders or perform bank transfers. The supplier and bank Web services must be accessible only to business partners with appropriate credentials. This means that the Web service composer has to know the *security policy* of the partner services. The security policy specifies which authentication mechanisms (username/password pairs, binary certificates), encryption algorithms, digital signatures, etc. are supported by a partner web service. With authentication mechanisms, the partner Web service can be sure of the identity of the caller. The next step is to decide what the caller is allowed to do. This is the focus of *authorization*. Furthermore, it is also important that the factory which passed the order to the supplier cannot deny having done so (*non-repudiation*) and that nobody can claim to be that factory and misuse its identity. A further requirement is *data integrity*; both parties need appropriate support for integrity i.e., if the factory orders one hundred items then the security infrastructure must make sure that nobody can tamper with the data on its way to the target web service and change the order position. Appropriate security mechanisms are also needed to avoid *replay attacks* i.e., if malicious third-party copies the message for ordering car parts from the wire and resends it later, then the order should not be accepted the second time. When invoking the bank's payment Web service, it is essential that nobody can see the sensitive information transferred from the composite service to that partner (*confidentiality*). Both parties have to negotiate and agree on the mecha

isms used to ensure confidentiality.
For instance, if we consider authorizations concerns, we claim that an appropriate authorization framework is needed to smooth the flow of a transaction between multiple services whilst respecting the privacy of the data used. This is a complex task since each individual service may have its own authorization requirements.
The traditional authorization service is not appropriate in this kind of interactions where a coordinating service would need to exchange policy and credential information as well as managing the operation details. Managing these authorization exchanges can lead to processing bottlenecks within the service as well as privacy concerns given that the coordinating service retains visibility and control.
Authorizations may be defined for roles played by subjects that are interacting in the course of a business process. In contrast with conventional models, authorizations may be dynamically (re-)allocated to subjects signifying the fact that they are allowed to produce or consume particular events [6].

Our objective is to support compositions of Web services taking into account the authorization requirements of each Web service provider and composing only those that are compatible regarding these requirements. To fulfil this objective, we need to address two main issues. First, it is mandatory to have a policy language that should be well-defined, flexible enough to allow new policy information to be expressed and extensible enough to add new policy type. Second, we need to check the security of the component Web services to determine whether they are compatible with respect to the specified security requirements. This requires the ability to model the security characteristics of a Web service and to match them according to the specified constraints.

3 Literature review

Various aspects of security policy of web services have been investigated. Some aspects were concerned with how to specify a policy in a machine readable and user friendly way at the same time, how to compose different policies and how to prove that the web service does ensure its policy specification with each request. In what follows, we review the most important approaches for securing Web services compositions.

In [15], authors utilized WS-Policy and WS-Security to propose a secure framework for the sake of securing BPEL compositions. In confidentiality, XML-Signature is used to providing integrity, and security token is given to support authentication. The process container which is implemented by a set of aspects in AO4BPEL is the main component of proposed framework. AO4BPEL is an Aspect-Oriented extension for BPEL which supports more adaptable and modular WSs and is implemented as an aspect aware orchestration engine for BPEL.
Davi Böger et al. propose a model in [4] wherein existing standards are combined and tried to provide a practical and consistent solution for secure service composition. According to the approach, WS-Policy is utilized to specify policies and supports not only the orchestration language (WS-BPEL), but also the business processes description language (WS-CDL). an approach presented in [7] is proposed to build processes in accordance with consumer security requirements and provider capabilities. In order to express these characteristics, the suggested approach utilizes Web Ontology Language (OWL) ontology and Web Services Policy Framework (WS-Policy) policies.
[11] presented a framework to execute composite Web service in a decentralized and secure. The main component of the framework is a data structure called container which is passedamong the participating web services in the composition process. The container is encrypted and authenticated so that the execution flow is secured and a set of security requirements are addressed.
In [13], Judith E. et al, presented a policy-driven approach integrated with Authentication and Authorization patterns (AA-patterns) to compose services and restrict service access to only authorized users. The authors point out that the approach is applicable considering both static and dynamic composition of services. According to the approach, UML 2.0 is employed to specify AA-patterns as well as Object Constraint Language (OCL) used for specifications of semantic interfaces annotated with policies.
An RBAC access control model for WSC is proposed in [16], where different constraints are expressed via access control rules. These constraints may include separation of duty constraints and past histories of service invocations constraints which can also be dependent on one or more parameters associated with a WS invocation. In order to represent access control rules, a Pure- Past Linear Temporal Logic Language (PPLTL) is used. In addition, role translations enforce access control and they are defined in a form of a table to map roles among different involved organizations in the composition process. After that, if user having a certain role invokes an operation of the composite Web service, the role translation is carried out through the enforcement system and a composite role is created. A composite role includes a temporally ordered sequence of roles and services involved in the invocation.
An integrated access control model for Web service oriented architecture is presented in [17] wherein Attribute-Based Access Control (ABAC) model is combined with hierarchical RBAC. [18] proposed an extension of this model to support composite service wherein policy is enforced by composite service. This policy is a combination of the policies which protect the operations invoked in the composition process.
A semantic web service composition approach namely

SCAIMO with respect to security issues is presented in [19]. In the SCAIMO framework, a secure task matchmaker based on AI-planning and Web Service Modeling Ontology (WSMO) was introduced to match tasks with operators and methods as well as take cares security requirements of both service provider and requester. To achieve this aim, three different constrains including security related goal, choreography, and orchestration are defined and checked during matchmaking process.
Kuter and Golbeck [20] proposed an approach to generate trustworthy Web service composition. To achieve this goal, they present a new formalism for Web service composition considering available user ratings as well as a novel service composition algorithm called Trusty. Moreover, three trust computation strategies for Trusty are defined; namely overly-cautious, overly-optimistic and average. In their approach, the Hierarchical Task Network (HTN) planner SHOP2 is advanced in order to generate trustworthy service composition by incorporating reasoning mechanisms for social trust. The trust information is used as input for this new procedure and as a result, the most trustworthy composition is produced to solve a service composition problem.

Several approaches are based on WS-Security has been utilized in to address security issues including confidentiality and integrity. Nevertheless, basic security functionalities can only be provided through WS-Security and there is no enough support provided in those approaches to ensure security for Web services compositions. With respect to security policy languages, XACML and WSPolicy languages are employed to specify some Web services security policies. However, WS-Policy and XACML lacks semantics. It in turn impedes the effectiveness of computing the compatibility between the policies.

Some other approaches are based on RBAC model. However, RBAC is insufficient method to use in service composition due to the following reasons: firstly, RBAC as inactive security model cannot dynamically administrate permissions in states executions of working progress. Following this, RBAC suffers from the inability for specifying a fine-grained control in collaborative environments. Next, RBAC provides no abstraction to capture a set of collaborating users which operate in different roles. Lastly, RBAC sometimes faces difficulties for encapsulation of all permissions to perform a job function.
A very recent and important work is addressed in [21]. This paper presents a comparative evaluation of state-of-the-art approaches in service composition. A taxonomy of service composition approaches with respect to security issues is introduced as well. Each classification of the proposed taxonomy including their respective approaches is illustrated in details. They consist of syntactic-based and semantic-based approaches. Moreover, the comparative evaluation of state-of-the-art approaches considering specified criteria is provided for each classification

4. An authorization model for Web services compositions

In the context of Web services a service is seen as a resource that is provided within the system, to which access is controlled. A service can also request other services and is actively involved in computation. In our formal policy model, a Web service can therefore be seen as both object (s_t) and subject (s_s). The type of request made to the Web service is modeled as an action.
To provide an authorization specification that allows expressing hybrid access control policies, we use two Booleans *autho*+ and *autho*− to model positive and negative authorizations respectively. We propose also to extend authorizations policies to express temporal constraints that are of utmost importance for Web service composition languages to keep their promises. Therefore a positive authorization is denoted by *autho+(s, o, a, t)*, where *s*, *o*, *a*, and *t* stand for subject, object, action and timepoint respectively. This authorization holds if the value of *autho+(s, o, a, t)* equals true at time t and does not hold otherwise. Similarly, autho−(s, o, a, t) models a negative authorization. Positive and negative authorizations are used at the specification level to state who is or is not allowed to do what. In case of conflicts, i.e. a subject has both positive and negative authorization; a conflict resolution rule (autho) determines the actual access decision. For instance, R0 shows a conflict resolution rule, stating that a negative authorization takes precedence over a positive authorisation.
R0. autho+s, o, a, t) $\wedge \neg$ *autho-s, o, a, t)* $\rightarrow$ *autho(s, o, a, t)*

4.1 Authorization Model

To allow the necessary level of control over the behavior of the Web service composition, authorization policies should be defined in a language flexible enough to allow the specification of conditions that can include multiple triggering events that may take place over time. The EC language seems to be the best basis to start from. We adapt a simple classical logic form of the EC [9], whose ontology consists of (i) a set of time-points,(ii) a set of timevarying properties called fluents, (iii) a set of event types (or actions). The logic is correspondingly sorted, and includes the predicates *Happens*, *Initiates*, *Terminates* and *HoldsAt*, as well as some auxiliary predicates defined in terms of these. *Happens (a, t)* indicates that event (or action) *a* actually occurs at time-point *t*. *Initiates(a, f, t)* (resp. *Terminates(a, f, t)*) means that if event *a* were to occur at *t* it would cause fluent *f* to be *true* (resp. *false*) immediately afterwards. *HoldsAt (f, t)* indicates that fluent *f* is true at *t*.
Then, the complete authorization enforcement model is illustrated in Figure 1.

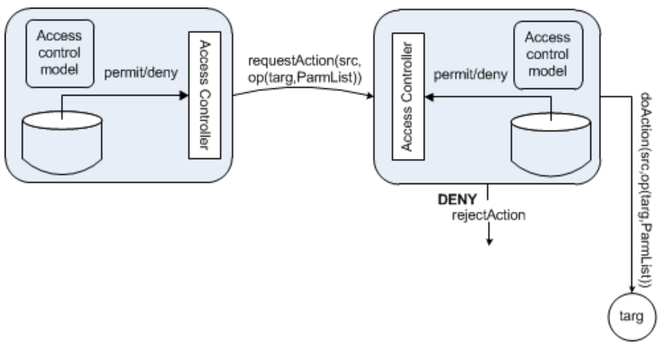

Figure 1 Authorization Enforcement Model

As shown, once the service source makes a request to perform an action on the service target, the target service's access controller processes it. To do this, the access controller evaluates the request by referring to the policy repository and the access control model. If the action is permitted, the access control model will proceed to do the requested action. Otherwise, if the action should be denied, the access control system will reject the action. We precise that the scheme is symmetric, i.e each of the two services could be target, source, or target and source at the same time. As shown in Figure 1, we distinguish two scenarios to represent the enforcement model. The first scenario models the behaviour of the target service's access controller, generating a *doAction* event when an action is permitted. This event would trigger the relevant service behaviour rules thus causing the composition state to change according to the specification. The second one models a target service's access control monitor rejecting the action to prevent a denied operation from being performed. Given the set of parameters values for the operations supported by services (Vp), *DoAction*$(s_s,Op(s_t,Action(Vp)))$ represents the event of the action specified in the operation term being performed by the service s_s for the service s_t. In the same way, *RejectAction*$(s_s,Op(s_t,Action(Vp)))$ is the event that occurs after the enforcement decision to reject the request by a particular source service to perform an action. *RequestAction*$(s_s,Op(s_t,Action(Vp)))$ represents the event that occurs whenever a service source attempts to perform an operation on a target service. Therefore, this is the event that will trigger a permission (or denial) decision to be taken by the target service's access controller.

4.2 Authorization specification

In order to correctly interact with the enforcement model described above, each policy specification rule should initiate the appropriate policy function symbol (permit, deny) for each of the events. So for example, a positive authorization policy rule should specify that the fluent *permit*$(s_s,Op(s_t,Action(Vp)))$ holds when the event *requestAction*$(s_s,Op(s_t,Action(Vp)))$ occurs and the constraints that control the applicability of the policy hold. Additionally, the fluent *permit*$(s_s,Op(s_t,Action(Vp))))$ should cease to hold once the action has been performed thus making it possible to re-evaluate the policy rule on subsequent requests to perform the action. The EC representation of this functionality is indicated by the **autho+** specification as follows.

(autho+)$isValidComp(s_s,Op(s_t,Action(Vp)))\wedge Constraint \rightarrow Initiates(requestAction(s_s,Op(s_t,Action(Vp))), permit(s_s,Op(s_t,Action(Vp))), t)$

(autho+) $isValidComp(s_s,Op(s_t,Action(Vp)))\rightarrow Terminates(doAction(s_s,Op(s_t,Action(Vp))), permit(s_s,Op(s_t,Action(Vp))), t)$

This also shows how each of the other policy types would be represented by rules in the formal notation. For each rule, the terms s_s, s_t, *Action* and *Constraint* can be directly mapped to the source service, target service, action, constraint and event clauses used when specifying policies.

The *isValidComp* predicate checks if the members of the $(s_s,Op(s_t,Action(Vp)))$ tuple are consistent with the specifi cation of the Web service composition.

The *Constraint* predicate is introduced to specify the pre- and post-conditions for each operation. It can be represented by a combination of *HoldsAt* terms. The **autho−** specification represents a negative authorization policy by stating that, if the *Constraint* holds and the event requesting the action happens, the action is denied. It is specified as follows.

(autho+)$isValidComp(s_s,Op(s_t,Action(Vp)))\wedge Constraint \rightarrow Initiates(requestAction(s_s,Op(s_t,Action(Vp))), deny(s_s,Op(s_t,Action(Vp))), t)$

(autho+) $isValidComp(s_s,Op(s_t,Action(Vp)))\rightarrow Terminates(doAction(s_s,Op(s_t,Action(Vp))), deny(s_s,Op(s_t,Action(Vp))), t)$

The second part of the rule shows how the *deny* fluent will be terminated once the decision to reject that action has been taken, thus allowing the specification to be reevaluated on subsequent requests. Note that the termination parts for these policies do not have any constraints and can be generically specified for the whole service composition.

4.3 Case study

Access to electronic patient records underlies certain r strictions. The concrete requirements are sophisticated and often dependent on circumstances that are not under the control of the entities that are involved in the access. Although we restrict ourselves here to a few requirements, more complex policies can be specified using the model.

Scenario: *Patients can always access their own medical records, but not append any information. The clinician that created a medical record is responsible for that record. The responsible clinician can read and append to records he/she is* *responsible for. In case of a national emergency situation (e.g. epedemic disease) the protection of personal information stored in health-records is relaxed. To ensure that a sufficient service can be provided, all health-care professionals are allowed to read information stored on electronic records.*

The requirements expressend in EC are given by the following rules:

(R1)*autho+(P,R,read,t)=isValidComp(R,Op(read(P,R)))∧Happens(requestAction(S,Op(read(S,R)),t)∧Initiates(requestAction(S,Op(read(S,R)),equalTo1(S, P),t) → Initiates(requestAction(P,Op(read(P,R))), permit(P,Op(read(P,R))), t)*

(R2)*autho−(P,R,append,t)=isValidComp(R,Op(append(P,R)))∧ Happens(requestAction(S,Op(append(S,R)), t)∧ Initiates(requestAction(S,Op(append(S,R)),equalTo(S, P), t) → Initiates(requestAction(P,Op(append(P,R))), deny(P,Op(append(P,R))), t)*

(R3) *∀(t1, t2)Happens(doAction(C,Op(create(C,R), t1) ∧ t2 > t1 → autho+(C,R, read, t2)*

(R4) *∀(t1, t2)Happens(doAction(C,Op(create(C,R), t1) ∧ t2 > t1 → autho+(C,R, append, t2)*

(R5) *Happens(requestAction(C,Op(read(C,R)), t) ∧ Initiates(requestAction(C,Op(read(C,R)), equalTo(C, hcProf), t) → autho+(C,R, read, t)*

According to these rules, the owner *P* of a medical record *R* can read his/her record, but not append ((R1),(R2)). The responsible clinian *C* can read and append to records *R* that he/she created. *C* is responsible, if he/she created the resord in the past.)((R3),(R4)). In an emergency situation, health-care professionals can read information stored in electronic patient records ((R5)).

The rules ((R1)...(R5)) are the basic elements that compose the overall authorisation policy. Let P = {(R1), (R2)} now be the policy that applies normally, Q ={(R5)} be the policy that applies in under emergency conditions, R = {(R0), (R3), (R4)} (R0 introduced in Section 3.1) the simple policy that applies over the whole policy composition. R defines the conflict resolution and that the responsible clinician can access the record. The latter is necessary, to ensure that the temporal reference in the rules applies over the whole policy composition:

$$S = R \,||\, < emergency() > P; [emergency()](P \,||\, Q))+$$

The operators used are explained as follows:

- P||Q: Both, policy P and policy Q apply at the same time.
- P+ : defines an iteration of policy P
- P;Q: Sequential composition of two policies. The system is first governed by policy P and then by policy Q.
- < w > P: The system is governed by policy P unless w holds. The state formula w can here indicate the happening of an event.
- [w]P: The system is governed by policy P as long as w holds.

It is also necessary to mention that all these operators are then expressed using the same EC formalism.

4.4 Conflicts

In order to detect conflicts involving authorization policies, i.e. those that arise when it exist two policies defined for the same source, target and action: one being an authorization and the other one being a prohibition, we introduce the *authConflict* predicate that holds if an authorization conflict is detected. This predicate is defined as:

HoldsAt (permit (s_s, Op (s_t, Action(Vp))), t) ∧
HoldsAt (deny (s_s, Op (s_t, Action(Vp))), t)= ∧
HoldsAt (authConflict (s_s, Op (s_t, Action(Vp))), t)

Example: Let us consider a typical example of authorization conflict, which arises when the same service is assigned to two roles that have opposite authorization permissions.

To enable a complete specification of the different conflict cases that may arise, we introduce a further set of predicates, events, and fluents. First, we introduce the predicate *ContradictoryRoles(r1, r2, t, a)* to describe that two roles *r1* and *r2* have opposite permissions for processing an action *a* at timepoint *t*. Here, we just recall that the role can be either s_s or s_t. Then, the events introduced are *AssignServiceRole(s, r)* that denotes a request of a service *s* for assignment to a role *r*, *RolePermitAction(r, a)* that specifies a request for permission of an action *a* for a role *r*, and *RoleDenyAction(r, a)* that defines a request for denial of action *a* for a role *r*.

Finally, three fluents are specified: *Assigned(s, r)* indicates that service *s* is assigned to a role *r*, *RoleHavePermission(r, a)* defines that a role *r* is permitted to process action *a*, and *AuthorizationConflict(r1, r2)* denotes that there is an authorization conflict in the composition (a service is assigned to contradictory roles).

Considering the elements described above, it is possible to define rules that can be used to recognise conflicting situations in the authorization policy specification. These rules are shown as follows.

(C1) *Happens(RolePermitAction(r,a),t)∧ ¬HoldsAt(RoleHavePermission(r, a), t) → Initiates(RoleHavePermission(r, a),RolePermitAction(r, a), t)*

(C2) *Happens(RoleDenyActivity(r, a), t)∧HoldsAt(RoleHavePermission(r,a),t)→Terminates(RoleHavePermission(r, a),RoleDenyActivity(r, a), t)*

(C3)*Happens(AssignUserRole(s,r1),t)∧ ¬HoldsAt(AuthorizationConflict(r1, r2), t) → Initiates(Assigned(s, r1),AssignUserRole(s, r1), t)*

(C4)*HoldsAt(RoleHavePermission(r1,a),t)∧ ¬HoldsAt(RoleHaveP ermission(r2, a), t)|HoldsAt(RoleHaveP ermission(r2, a), t)∧(¬HoldsAt(RoleHaveP ermission(r1, a), t) → ContradictoryRoles(r1, r2, t, a)*

(C5)*HoldsAt(Authorized(s,r2),t)∧Happens(AuthorizeRequest(r1,s),t)∧ContradictoryRoles(r1,r2, a, t) → Happens(conflictEvent, t)∧Initiates(AuthorizationConflict(r1, r2),*

Figure 3 Conflicts Specification

conflictEvent, t)
The first rule initiates the fluent *RoleHavePermission(r, a)* when the event *RolePermitAction(r, a)* happens if this fluent is currently not true. The second rule implements deny for role *r* to process the action *a* as a termination of fluent *RoleHavePermission(r, a)* when *RoleDenyActivity(r, a)* event happens. The third rule assigns service s to the role *r* when *AssignUserRole(s, r)* event happens if *AuthorizationConflict(r1, r2)* between the role *r1* and some other role *r2* is not present in the composition process. The fourth rule defines two roles, one of which has and another one does not have permission for some action. Here we note that we not fix which role has positive permission and which role has negative permission.Thus, ContradictoryRoles is symmetrical regarding *r1* and *r2*. Finally, the fifth rule defines a notion of authorization conflict: the user requested the assignment for the second of two contradictory roles. These rules are generic and can be composable to obtain a general constraint about the composition process.

5 Validation

In our study we utilize theorem proving for verifying that a given policy is conflict-free and proving that add and remove operations do not introduce conflicts. In this section, we describe a method for representing EC in the SPIKE language. The SPIKE induction prover has been designed to verify quantifier-free formulas in theories built with first order conditional rules. SPIKE was chosen for the following reasons: (i) its high automation degree, (ii) its ability on case analysis (to deal with multiple operations), (iii) its refutational completeness, (to find counter-examples), and (iv) its incorporation of decision procedures. SPIKE proof method is based on the so called cover set induction: Given a theory SPIKE computes in first step induction variables where to apply induction and induction terms which basically represent all possible values that can be taken by the induction variables. Given a conjecture (rule or a policy) to be checked, the prover selects induction variables according to the previous computation step, and substitutes them in all possible way by induction terms. This operation generates several instances of the conjecture that are then simplified by rules, lemmas, and induction hypotheses.
The ingredients of our encoding are shown in what follows;
Data. All data information manipulated by the system is ranged over a set of sorts. This data concerns generally the argument types of events and fluents.
Events. We consider that all events of the system are of sort *Event*, where the event symbols are the constructors of this sort. These constructors are free as all event symbols are assumed distincts.
Fluents. The sort *Fluent* respresents the set of fluents. All fluent symbols of the systems are the constructors of sort Fluent, that are also free.
Time. We use the sort of natural numbers, Nat, which is reflected by constructors 0 and successor succ(x) (meaning x + 1).
Axioms. We express all predicates used in EC as boolean function symbols. The signatures of these functions symbols and others additional functions are as follows:
$Happens : Event \times Nat \rightarrow Bool$
$Initiates : Event \times Fluent \times Nat \rightarrow Bool$
$Terminates : Event \times Fluent \times Nat \rightarrow Bool$
$HoldsAt : Fluent \times Nat \times Nat \rightarrow Bool$
$Clipped : Fluent \times Nat \times Nat \rightarrow Bool$
HoldsAt and *Clipped* are defined within a time range. For instance, *HoldsAt(f, t1, n)* is defined within the range [t1, t1 + n].
Finally, the EC axioms necessary to do the verification process are expressed in conditional equations as follows:
(A1) $event \neq Noact \wedge Happens(p(event, t1)) = true \wedge Initiates(event, f, t1) = true \rightarrow HoldsAt(f, t1, 0) = true$
(A2) $HoldsAt(f, t1, t) = true \wedge Clipped(f, t1 + t, s(0)) = false \rightarrow HoldsAt(f, t1, s(t)) = true$
(A3) $event \neq Noact \wedge Happens(p(event, t1)) =true \wedge Terminates(event, f, t1) = true \rightarrow Clipped(f, t1, s(0)) = true$
(A4) $event 6= Noact \wedge Happens(p(event, t1 + t + s(0))) = true \wedge Terminates(event, f, t1 + t + s(0)) = true \rightarrow Clipped(f, t1, s(s(t))) = true$
(A5) $Happens(p(Noact, t1 + t + s(0))) = true \rightarrow Clipped(f, t1, s(s(t))) = Clipped(f, t1, t + s(0))$
Authorization rules. In the same way, we can express the autho+ and the autho− rules in equational form. For instance, the requirement (R1) in Section 4 is written as follows:
(R1) $isValidComp(R,Op(read(P,R))) = true \wedge Happens(requestAction(S,Op(read(S,R)), t) = true \wedge Initiates(requestAction(S,Op(read(S,R)), equalTo(S, P), t) = true \rightarrow Initiates(requestAction(P,Op(read(P,R))),permit(P,Op(read(P,R))), t) = true$
Finally, we build an algebraic specification from EC specification. Once building this specification, we can check all authorization rules by means the powerful deductive techniques (rewriting and induction) provided by SPIKE.
All the generated axioms can be directly given to the SPIKE prover, which automatically orientes these axioms into conditional rewrite rules. Then, given as inputs the specification of the composition expressed in algebraic equations and the authorization rules to be checked, when SPIKE is called, either the authorization rules proof succeed, or the SPIKE 's proof-trace is used for extracting all scenarios which may lead to potential deviations. There are two possible scenarios. The first scenario ismeaningless because conjectures are valid but it comes from a failed proof attempt by SPIKE . Such cases can be overcome by simply introducing new lemmas. The second one concerns cases corresponding to real deviations. The trace of SPIKE gives all necessary informations

(events, fluents and timepoints) to understand the inconsistency origin. Consequently, these informations help designer to detect policies problems in the composite Web service.

6 Conclusion

Despite the importance of Web service composition, securityissues have not been extensively investigated and security concerns become one of the main barriers that prevent widespread adoption of this new technology.

In this paper we have discussed the security requirements and challenges of securing Web services compositions. We have also reviewed the most important efforts that addressed this problem statement by exposing their strengths and limits. As a contribution to this important concern, we have presented a framework for managing authorization policies for Web service compositions. The methodology was supported by a formal representation of conflicts scenarios that may arise during the composition process. There are several directions for future work to further improve the presented work. One thread in our future work will focus on the policies refinement and the generalization of the reasoning technique to handle other security properties. We plan also to study forensics aspects for services composition. Indeed, investigations of breaches of security or suspicious events in, or transaction auditing of SOAs, would employ digital evidence in an effort to reconstruct the events under investigation. Such evidence would be helpful to financial fraud investigators, business arbiters, potential investors, and judicial actors. However, unlike traditional forensics implementations, applying forensics to service oriented infrastructures introduces novel problems such as platform independence, need for neutrality and comprehensiveness, and reliability issues because of interdependencies between services and the ability to build global services using Web service compositions processes.

References

[1] M. Abadi, M. Burrows, B. W. Lampson, and G. D. Plotkin. A calculus for access control in distributed systems. In CRYPTO ’91: Proceedings of the 11th Annual International Cryptology Conference on Advances in Cryptology, pages 1–23, London, UK, 1992. Springer-Verlag.

[2] G.-J. Ahn and R. Sandhu. The rsl99 language for rolebased separation of duty constraints. In RBAC ’99: Proceedings of the fourth ACM workshop on Rolebased access control, pages 43–54, New York, NY, USA, 1999. ACM Press.

[3] E. Bertino, S. Jajodia, and P. Samarati. Supporting multiple access control policies in database systems. In SP ’96: Proceedings of the 1996 IEEE Symposium on Security and Privacy, page 94, Washington, DC, USA, 1996. IEEE Computer Society.

[4] Boger, D., et al., A Model to Verify Quality of Protection Policies in Composite Web Services. 2009 IEEE Congress on Services, ed. L.J. Zhiang. 2009. 629-636.

[5] W. K. Edwards. Policies and roles in collaborative applications. In CSCW ’96: Proceedings of the 1996. ACM conference on Computer supported cooperative work, pages 11–20, New York, NY, USA, 1996. ACM Press.

[6] W.-J. V. D. Heuvel, K. Leune, and M. P. Papazoglou. Efsoc: A layered framework for developing secure interactions between web-services. Distrib. Parallel Databases, 18(2):115–145, 2005.

[7] Garcia, D.Z.G. and M.B. Felgar de Toledo. Ontology-Based Security Policies for Supporting the Management of Web Service Business Processes. in Semantic Computing, 2008 IEEE International Conference on. 2008.

[8] H. Koshutanski and F. Massacci. An access control framework for business processes for web services. In XMLSEC ’03: Proceedings of the 2003 ACM workshop on XML security, pages 15–24, New York, NY, USA, 2003. ACM Press.

[9] R. Kowalski and M. J. Sergot. A logic-based calculus of events. New generation Computing 4(1), pages 67–95, 1986.

[10] T. Moses. Extensible access control markup language (xacml) version 2.0 3, Feb 2005.

[11] Biskup, J., et al., Towards secure execution orders for composite web services. 2007 IEEE International Conference on Web Services, Proceedings, ed. L.J. Zhang, et al. 2007. 489-496.

[12] M. Siponen, R. Baskerville, and T. Kuivalainen. Integrating security into agile development methods. hicss, 07:185a, 2005.

[13] Rossebø, J. and R. Bræk, A policy-driven approach to dynamic composition of authentication and authorization patterns and services. Journal of Computers, 2006. 1(8): p. 13.

[14] T. Woo and S. Lam. Authorization in distributed systems: a new approach. Journal of Computer Security, 2:107–136, 1993.

[15] Charfi, A.; Mezini, M.; , "Using aspects for security engineering of Web service compositions," *Web Services, 2005. ICWS 2005. Proceedings. 2005 IEEE International Conference on* , vol., no., pp. 59- 66 vol.1, 11-15 July 2005

[16] Paci, F., R. Ferrini, and E. Bertino, Identity Attribute-based Role Provisioning for Human WS-BPEL processes. 2009 Ieee International Conference on Web Services, Vols 1 and 2, ed. E. Damiani, J. Zhang, and R. Chang. 2009. 535-542.

[17] Srivatsa, M., et al. An access control system for web service compositions. in ICWS. 2007.

[18] Emig, C., et al. An access control metamodel for Web Service-oriented architecture. in ICSEA. 2007.

[19] Zhu, J.Q., Y. Zhou, and W.Q. Tong, Access control on the composition of Web services. International Conference on Next Generation Web Services Practices, Proceedings, ed. A. Abraham and S.Y. Han. 2006. 89-93.

[20] Kuter, U. and J. Golbeck, Semantic Web Service Composition in Social Environments. Semantic Web – ISWC 2009, Proceedings, 2009. 5823: p. 344-358.

[21] Homa Movahednejad, Suhaimi Bin Ibrahim, Mahdi Sharifi, Harihodin Bin Selamat, and Sayed Gholam Hassan Tabatabaei. Security-aware web service composition approaches: state-of-the-art. In *Proceedings of the 13th International Conference on Information Integration and Web-based Applications and Services* (iiWAS '11). ACM, New York, NY, USA, 112-121. 2011.