www.ijiarec.com

ISSN:2348-2079

Volume-6 Issue-2

International Journal of Intellectual Advancements and Research in Engineering Computations

BPEL based rule generator design and verification of secure service

Jeena. A.J[1], Dr.R.Vijaya Rajeswari[2]

[1]PG Student, Department of Computer Science and Engineering, Mahendra Engineering College (Autonomous), Namakkal-637503.

[2]Associate Professor, Department of Computer Science and Engineering, Mahendra Engineering College (Autonomous), Namakkal-637503.

ABSTRACT

Ensuring the preservation of security is a key requirement and challenge for Service Based Systems due to the use of third party software services not operating under security perimeters. By handling the orchestration, composition and interaction of Web services, the Business Process Execution Language or BPEL has gained tremendous interest. However, such process-based language does not assure a secure environment for Web services composition. The key solution cannot be seen as a simple embed of security properties in the source code of the business logic since the dynamism of the BPEL process will be affected when the security measures get updated. In this context, several approaches have emerged to tackle such issue by offering the ability to specify the security properties independently from the business logic based on policy languages. To mitigate these challenges, we present a framework providing integrity and authentication for secure workflow computation based on BPEL Web Service orchestration. Where as much attention has been dedicated to security issues for Web Services. We propose a novel approach that provides the users with model-based capabilities to specify aspects that enforce the required security policies. On the other hand, it offers a high level of flexibility when enforcing security hardening solutions in the BPEL process. We present a prototype implementation for validating linear workflows and evaluate its performance based on the security aspects in a BPEL process.

INTRODUCTION

Security assurance is important for any software application but acutely so in the case of service-based systems (SBSs), i.e., systems composed of distributed software services, which can be deployed on different and heterogeneous infrastructures and operate without common ownership and centralized control. Assessing and providing assurance about the security of SBSs is a complex problem that has no comprehensive solution to the best of our knowledge. Existing solutions rely on different forms of model checking and theorem proving to verify security properties of service compositions. These approaches typically require the specification of behavioral models of the software services used by the SBS, the component that orchestrates them to provide the SBS functionality, and the security properties that need to be guaranteed in some temporal logic language. There are two main difficulties with such approaches. The first is that creating accurate specifications of for realistic SBSs is a non-trivial and time-consuming task. The second is that, even if SBS specifications are available, performing automated static analysis might be computationally intractable. With the drastic success of service-oriented computing, a wave of innovations has been unleashed to enhance the interaction between customers and service providers. Particularly, Web services composition is currently emerging as a convenient mechanism for automated interaction between distributed applications. BPEL (Business

Author for correspondence:
PG Student, Department of Computer Science and Engineering, Mahendra Engineering College (Autonomous), Namakkal- 637503

Process Execution Language) is so far the most important standard language for Web services composition. However, Web services composition via a BPEL process is still weak at the security level. The common practice used to harden the security of a BPEL process is to embed the security verification code within the business logic of such process. More specifically, when any of the security measures changes, the user should stop the process deployment, make the needed modifications and then redeploy it making all the offered services unavailable during the update procedure. This lack of inability to perform a dynamic update of the security requirements certainly affects the performance of the BPEL process. In this context, some dynamic approaches were proposed to enforce Web services security [1-5].

Workflow describes the automation of a business process. During the process, documents, information, or roles are exchanged among actors in order to complete a task as specified by a well-defined set of rules. A workflow management system allows defining, creating, and managing the execution of a workflow through a software executing on one or more workflow engines. A workflow manager interpreters the formal definition of a process, in order to interact with several actors by managing states and tasks coordination. Tasks, actors, and processes within the workflow can be designed as desired. So it needed to focus on the Web Service technology and on WS-BPEL, the Business Process Execution Language for Web Services. WS-BPEL is an XML-based language for describing the behavior of business process based on Web Services. In other words, WS-BPEL defines a language to describe and manage Web Services orchestration, and to write programs that are Web Services. WSBPEL relies on the concept of activities. These can be either basic or structured [6-10].

In this paper, we present an alternative approach for designing, adapting and verifying the security properties of SBSs, which is based on pattern driven verification. Our approach assumes that an SBS is designed and implemented by a service orchestration process which invokes the individual services that constitute the SBS and may perform various computations upon the data exchanged with these services. To support the verification of security properties, our approach uses secure service composition (SCO) patterns. These patterns encode proven dependencies between service level security properties (i.e., security properties of the individual services of an SBS) and workflow level security properties (i.e., security properties of the entire orchestration/ workflow of the SBS). The encoding of such dependencies in SCO patterns enables the inference of service level security properties, which – if satisfied by the individual services of the SBS – would guarantee the satisfaction of workflow level security properties for it [11-15].

BACKGROUND

Attacks against secure service systems, on which the infrastructures of modern society and modern economies rely, cause substantial financial damage. Due to the increasing interconnection of systems, such attacks can be waged anonymously and from a safe distance. Thus networked computers need to be secure. The high-quality development of security-critical systems is difficult. Still, many systems are developed, deployed, and used over years that contain significant security weaknesses While tracing requirements during software development is difficult enough, enforcing security requirements is intrinsically subtle, because one has to take into account the interaction of the system with motivated adversaries that act independently. Thus security mechanisms, such as security protocols, are notoriously hard to design correctly, even for experts. Also, a system is only as secure as its weakest part or aspect. Security is compromised most often not by breaking dedicated mechanisms such as encryption or security protocols, but by exploiting weaknesses in the way they are being used. Thus it is not enough to ensure correct functioning of security mechanisms used. They cannot be “blindly” inserted into a security-critical system, but the overall system development must take security aspects into account in a coherent way. In fact, according to 85% of Computer Emergency Response Team (CERT) security advisories could not have been prevented just by making use of cryptography. Building trustworthy components does not suffice, since the

interconnections and interactions of components play a significant role in trustworthiness.

The traditional strategy for security assurance has been "penetrate and patch": It has been accepted that deployed systems contain vulnerabilities. Whenever a penetration of the system is noticed and the exploited weakness can be identified, the vulnerability is removed. Sometimes this is supported by employing friendly teams trained in penetrating computer systems. However, this approach is not ideal: Each penetration using a new vulnerability may already have caused significant damage, before the vulnerability can be removed. It would thus be preferable to consider security aspects more seriously in earlier phases of the system life-cycle, before a system is deployed, or even implemented, because late correction of requirements errors costs up to 200 times as much as early correction. Also, security concerns must be taken into account during every phase of software development, from requirements engineering to design, implementation, testing, and deployment.

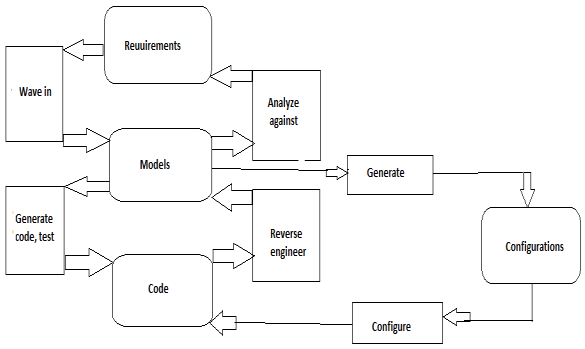

Fig 1.1: Model-based Security Engineering

Academic approaches trying to improve security during development must be tightly integrated with software development approaches already used in industry. Some other challenges for using sound engineering methods for secure systems development exist. For example, the boundaries of specified components with the rest of the system need to be carefully examined, for example with respect to implicit assumptions on the system context. Lastly, a more technical issue is that formalized security properties are not in all approaches preserved by refinement. Since an implementation is necessarily a refinement of its specification, an implementation of a secure specification may, in such a situation, not be secure, which is clearly undesirable. A truly secure software engineering approach thus needs to take both dimensions of the problem into account:

- It needs to integrate the different system lifecycle phases,
- It also needs to take into account the different architectural levels of abstraction of a security critical system in a demonstrably sound, trustworthy, and cohesive way.

SECURE COMPOSITION PATTERNS

SCO patterns encode proven dependencies between security properties of individual services (i.e., service level security properties) and security properties of the entire SBS service workflows (i.e., workflow level properties). The encoding of such dependencies enables the verification that the service workflow of an SBS satisfies certain security properties, and the generation (and adaptation) of an SBS workflow in a way that is guaranteed to satisfy required workflow level security properties. The main contributions are:

- It extends the SCO patterns representation scheme with additional conditions about pattern activity in- puts and outputs, which are

required for matching patterns with SBS workflows, and provides a schema for expressing pattern.

- It introduces the verification algorithm that is based on the extended form of SCO patterns and can verify if a service workflow satisfies specific security properties required of it.
- It presents and discusses the outcomes of an experimental evaluation of our approach.

This fragment (called Checkout) corresponds to the last part a purchasing process realized by an SBS. More specifically, upon receiving a purchasing request consisting of a list of items to be purchased, the credit card details and address of the purchaser, Checkout takes payment for the purchased items, places the order in a purchase repository and creates an order report. The activities Payment, PlaceOrder and WriteReport of Checkout are realized through the invocation of operations of partner services, which are assumed to have identical names with the relevant workflow activities.

In this scenario, a designer might wish to verify whether the Checkout workflow preserves the confidentiality of the credit card and address information of the user. To ensure this, it is necessary to verify that all the services, which are orchestrated by Checkout (i.e., Payment, PlaceOrder and WriteReport), preserve the confidentiality of credit card and address information, as well as the confidentiality of additional information that is exchanged internally within the workflow (i.e., paySuccess and orderSuccess) if the latter also includes any information about the credit card and address information of the user.

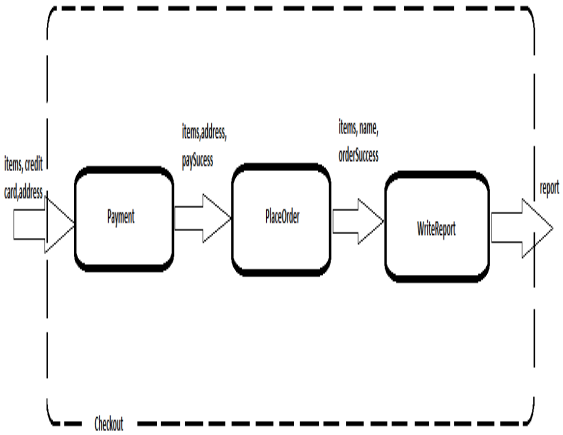

Fig 1.2 Example of SBS service workflow – Checkout.

Furthermore, the preservation of confidentiality will need to be checked against the transmission, processing and storage of any of the information items that need to remain confidential.

EXAMPLE OF SCO PATTERN

One example is an SCO pattern regarding the security property of confidentiality, i.e., a property requiring that no non-authorized disclosure of information should be possible in a system. Confidentiality has been commonly defined based on the concept of information flow (IF). IF-based definitions of confidentiality stratify the users of a system in classes with different access rights to information, and distinguish the information flows within it according to the class of users that they should be accessible to. Typically, the user classes used in IF-approaches are low-level users with restricted access to information, and high-level users having full access. Based on the above principles, there have been several definitions of properties, whose intent is to express the concept

of confidentiality. The definition that we focus on is that of Perfect Security Property (PSP). PSP requires that a low-level user, allowed to access only public information, should not be able to determine anything about high-level (i.e., confidential) information.

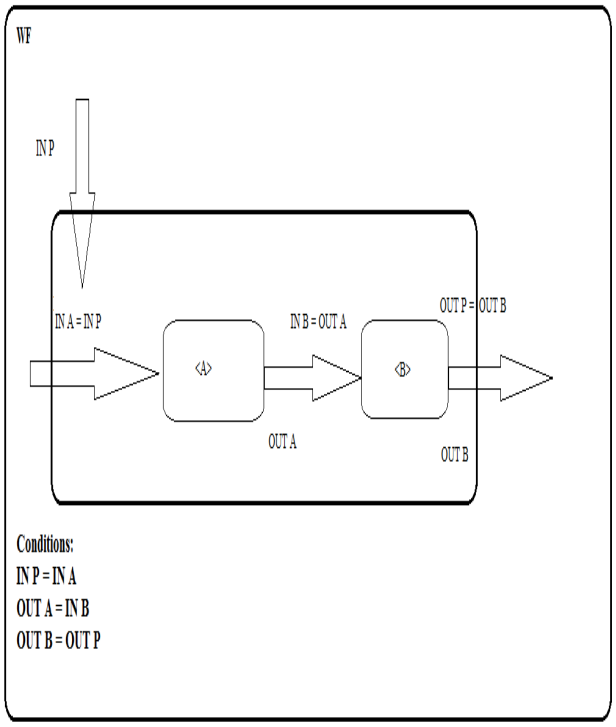

Fig 1.3 PSP SCO Pattern

Figure shows the SCO pattern for preserving PSP on a sequential service workflow P, i.e., a service workflow with two activity placeholders (A and B), in which A is executed before B. The structure of P is shown in the WF part of the figure. Further conditions that define P are specified in the Conditions part. These are: (a) the inputs of A are the inputs of the workflow (IN = IN), (b) the inputs of B are the outputs of A (IN = OUT), and (c) the outputs of P are the outputs of B (OUT = OUT).Let us assume that for each x in {P, A, B}

- I N X and OUT X are the sets of inputs and outputs of x, and E = IN ∪ OUT ;
- VX and C X are two disjoint subsets of E X, which partition it into public parts V (i.e., parts visible to low-level users) and confidential parts C (i.e., parts visible only to high-level users)

Then, as proven in PSP holds on the workflow P if, for all activity placeholders x {A, B}: (a) the actions of x that reveal public information are part of the actions of P that reveal public information and (b) the confidential actions of x that reveal confidential information do not include any action of P that reveals public information. The conditions (a) and (b) are expressed as ASP properties of the pattern, and entail the PSP property on P. The latter is expressed by PSP in the RSP part of the pattern.

Approaching model-based security

Use case diagrams describe typical interactions between a user and a computer system (or between different components of a computer system). We use them to capture security requirements. To start with example, Figure 1.4 gives the use case diagram describing the situation to be achieved: a customer buys a good from a business. The semantics of the stereotype <<fair exchange>> is, intuitively, that the actions "buys good" and "sells good" should be linked in the sense that if one of the two is executed then eventually the other one will be (where these actions are specified on the next more detailed level of specification).

Activity diagrams are especially useful to model workflow and to explain use cases in more detail. Following figure 1.5 and 1.6 explains the use case in figure 1.4 in more detail. To demonstrate the connection between this diagram and the one in Figure 1.4, we give two possible diagrams. Both are separated in two swim-lanes describing activities of different parts of a system (here Customer and Business). Round boxes describe actions (such as Request Good) and rectangular boxes describe the object flow (such as Order[filled]). Horizontal bars (synchronization bars) describe a required synchronization between two different strands of activity.

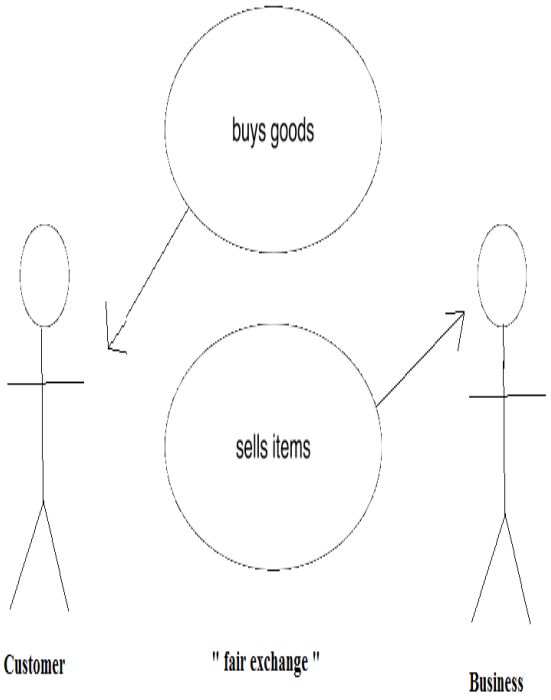

Fig 1.4. Exemplary use case diagram: "fair exchange"

A diamond describes the merging of two strands of activities. The start state is marked by a full circle, the final state by a small full circle within a circle. In each diagram, two tag-value pairs {fair exchange, buys goods} and {fair exchange, sells goods} are used to mark certain actions. Now any such diagram fulfills the security requirement "fair exchange", given in the diagram in Figure 1.4, if for both actions marked with these tag-value pairs it is the case that if one of the actions is executed, then eventually the other will be.

As one can demonstrate on the level of the formal semantics, the left diagram does not fulfill this requirement because the Business may never Deliver Order. Also one can show that the right diagram does fulfill the requirement (intuitively, because the customer may reclaim the payment if the order is undelivered after the scheduled delivery date), assuming that the customer is able to prove having made the payment (indicated by the stereotype <<provable>>), e.g. by following a fair exchange protocol

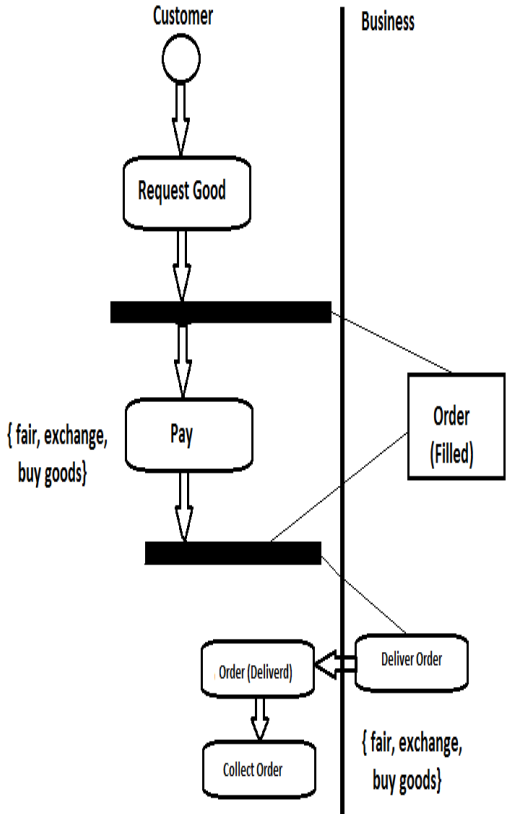

Fig 1.5. Fair Exchange VIOLATED

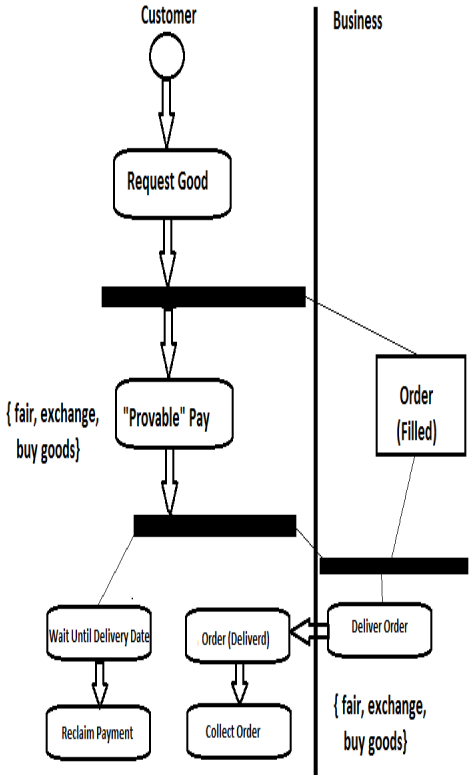

Fig 1.6. Fair Exchange ACHIEVED

MAIN APPROACH

In this section, we present an overview of the proposed approach that aims to provide a model-driven engineering of BPEL-based secure composite Web services. Figure 1.7 depicts our approach schema by illustrating the interaction between its components. As shown in the figure, the security requirements are developed as separated entities and specified at the design level using the Aspect BPEL Profile. In a previous work, IBM has developed a BPEL Profile that allows users to model BPEL processes and generate their corresponding code through an appropriate toolkit. We took advantage of their work to build our Aspect BPEL Profile on top of the proposed BPEL Profile. Our approach offers the user an abstract and user friendly environment to specify aspect-oriented security solutions.

The schema encompasses the following:

- Developing the security requirements within a UML model based on the Aspect BPEL Profile.
- Applying the Aspect BPEL Profile on a part of its elements (BPEL_ Aspect, BPEL_ Location_ Behavior, BPEL_- Insertion_ Point and BPEL_ Location_ Identifier classes) and the BPEL Profile on the rest of it (BPEL_ Behavior_ codeclass).
- Generating the aspect name, the insertion point and the location identifier from the first application (i.e., Applying the Aspect BPEL Profile) and the BPEL code from the second application (i.e., Applying the BPEL Profile).
- Generating automatically the complete Aspect BPEL code using our UML Model To Aspect BPEL Converter.
- Conveying the generated code to the Aspect BPEL with the selected BPEL process in order to produce a secure BPEL process.

The components of approach architecture are the following:

- Aspect BPEL Profile that offers the user a model- based technique to specify the security requirements to be hardened in the BPEL process.
- UML Model To Aspect BPEL Converter that generates automatically the Aspect BPEL aspects according to the Aspect BPEL Profile.
- Aspect BPEL that dynamically enforces security requirements into the BPEL process.

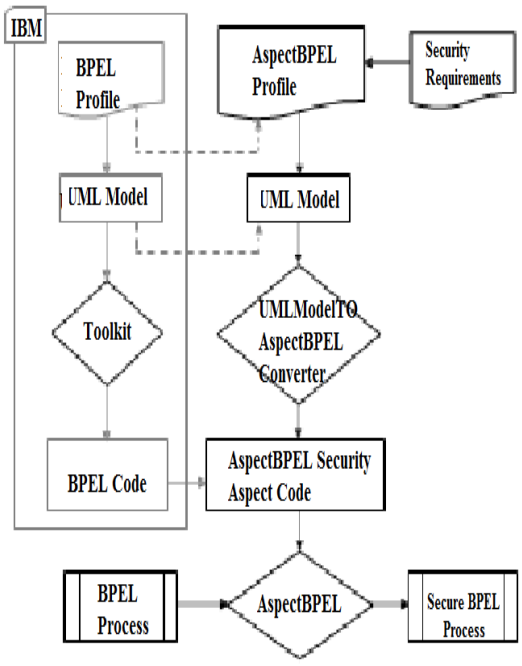

Fig 1.7. Approach Architecture

Our approach spans over three phases. The first phase corresponds to security requirements specification. It allows specifying security concerns in separate components and generates automatically their corresponding aspects expressed in AspectBPEL. In the second phase, AspectBPEL security aspects are generated using the developed converter. Finally, the third phase offers a tool for the dynamic weaving of the generated aspects within the BPEL process.

TAS ARCHITECTURE

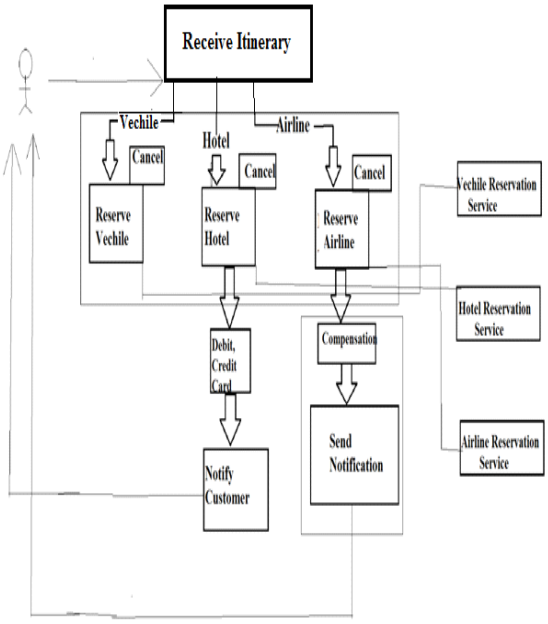

Fig 1.8. Travel Agency System Overview

Travel Agency System Overview (TAS) is one of the most common Web services applications. Therefore, we took it as a case study to demonstrate the usefulness of our approach. Figure1.8. depicts the interaction between the user and TAS including its composition of Web services. TAS consists of a BPEL process, its composition of three Web services and a graphical user interface that includes in its main page the services offered by this system. For instances, Book Flight Ticket, Book Hotel Room, Book Car. The first service allows the user to book a ticket after consulting the Airline Web Service. To reserve a room, the user can use the second service that includes a call for the Hotel Web Service. The user has also the ability to book a car from a Car Web Service.

VERIFICATION PROCESS

The process for verifying if an SBS workflow SBS-WF satisfies required security properties has two main phases. In the first of these phases, all the SCO patterns that can guarantee the RSP property required of SBS-WF are identified and, if their workflow structure matches the structure of SBS-WF, they are used to infer the ASP properties, which if satisfied by the individual services of SBSWF, would guarantee RSP for it. This phase may generate alternative combinations of ASP properties for the services of SBS-WF, which would guarantee RSP. Each of these combinations is what we call a security plan in our approach. In the second phase of the verification process each of the security plans generated in the first phase are used to drive a search process. This process checks if the individual services of SBS-WF referred to in the security plan satisfy the ASP property required of them by the plan. In the following, we present the algorithms that realize the two phases of the verification process.

The algorithm for generating different security plans, i.e., the possible alternative

combinations of security properties of activity placeholders of a workflow WF that would make it satisfy a workflow level security property RSP in the inner security plans algorithm.

Infer security plans Algoritham

Algorithm: INFER SECURITY PLANS (WF, Req, SecPlans)
Input: WF /* workflow */
Req /* security requirement for WF */
Output: SecPlans /* Security plans with ASPs */
INFERECURSION (WF, Req, [ ], SecPlans)
Algorithm: INFERECURSION (PH, Req, InPlans, OutPlans)
Input: PH /* activity placeholder*/
Req /* security requirement for PH */
InPlans /* list of current plans */
Output: OutPlans /* list of inferred security plans */
If there is no pattern P Such that
P.RSP matches Req and P.WF matches WF then
OutPlans: = InPlans
Else
For each pattern P such that P.RSP matches Req and P.WF matches WF do
$SecPlans_p$: = security requirement inferred
By the inference rules of P
Remove Req from InPlans
Add $SecPlans_p$ to InPlans
EndFor
For each R in InPlans where R. Subject is a work-flow do
INFERECURSION (R. Subject, R, InPlans, OutPlans)
EndFor
EndIf
EndFor

The algorithm is invoked having as input a workflow (WF) and a security property (RSP) required of it, which is encoded with the security requirement Req. Based on these two inputs, the algorithm derives the security requirements (i.e., ASP properties) that should be satisfied by partner services that may be bound to the different activity placeholders in WF in order to guarantee RSP. Given WF and a security requirement Req requiring the property RSP, the algorithm tries to apply all the SCO patterns that would be able to guarantee RSP. A pattern P is applied if its workflow (P.WF) matches with the input workflow WF. In this case, the security plan that can be derived from the pattern (through the application of its rules) are computed. These plans replace the initial requirement Req. If the updated list of security plans contains only ASP security properties that are required of individual activities (i.e., not of Orchestration Pattern placeholders), the algorithm terminates. Otherwise, if the security plans include security properties required of activity placeholders that are themselves (sub) workflows, the algorithm attempts to find SCO patterns that match the workflow structure of the sub work- flows and could guarantee the security property required of these sub workflows. This process terminates when a list of security plans includes workflow placeholders matching with no available SCO patterns.

The process of checking if a given workflow satisfies a required security property is realized by the algorithm VERIFYWORKFLOW that is listed below.

WORKFLOW VERIFICATION ALGORITHM

Algorithm: VERIFYWORKFLOW (WF, Req, VPlan)
Input: WF /* workflow */
Req /* security requirement to verify */
Output: VPlan /* verified security plan for WF */
INFRASECURITYPLANS (WF, Req, SecPlans)
VPlan := nil
For each plan in SecPlans do
If VERIFYREOUIREMENT (WF , Plan) then
VPlan:= Plan
Exit
EndIf
EndFor

GENERATION OF SECURE WORKFLOWS

When an existing workflow does not satisfy a required RSP security property due to a particular partner service (or a fragment of it), it might be possible to replace the responsible service (or workflow fragment) in order to restore the required RSP. This modification is handled by the

algorithm listed in below. This algorithm starts by trying to find appropriate workflows based on a query (Q) expressing structural, behavioral and security requirements for the service or process fragment that should be modified. To do this, initially it tries to identify abstract (i.e., not instantiated) workflows that can provide the requested functionality by searching for appropriate functional workflows in a repository of reference workflows. This repository contains abstract workflows encoding reference process models providing standardized functionalities in different domains. Examples of such reference process models exist for several domains including, for example, financial services (e.g., SWIFT) and electronic data interchange in fields such as manufacturing, logistics and telecommunications (e.g., RosettaNet Partner Interface Processes (PIPs) and IBM Industry Packs. The abstract workflow matching process is based on a structural matching algorithm described in secure workflow generation algorithm. ASPs for the individual partner services of the matching workflow or for the services of sub workflows that could be used to replace them. Following the identification of the alternative security plans (if any), GENERATESECURE- WORKFLOWS tries to discover individual services that can be bound to the abstract service in question. This is realized through the call of the algorithm SERVICEDISCOVERY. If such services can be identified for all the abstract partner services of an AW, GENERATESECURE- WORKFLOWS replaces the abstract services in AW with concrete services and/or workflows and returns the instantiated workflow as part of the possible solutions list.

DISCOVERY PROCESS

The discovery process is realized by the algorithm SERVICEDISCOVERY. This algorithm takes as input the specification of an activity A in an abstract workflow and a security plan SPlan and finds concrete services (Servs) that can be matched with A from a structural and a behavioral point of view, whilst also satisfying the security properties specified for A in SecPlan. The description of A includes the different inputs and outputs of the activity in the abstract workflow. For the activity Payment in , for example, the inputs are items, credit Card and address and the outputs are items, address, pay Success. The activity description in the workflow also specifies the types of these inputs and outputs. The first step of the algorithm is to construct a query expressed in A-SeRDiQueL, i.e., an XML based query language developed in ASSERT4SOA. A query in this language has four parts:

- A parameter part defining generic parameters of the query process (e.g., the matching algorithm that will be used for structural and behavioral matching, the maximum distance threshold for accepting candidate services, whether service composition should be triggered in cases where no single candidate service can match the query).
- A structural part specifying the interface, i.e., the set of operation signatures of A and the data types of the parameters of these operations.
- A behavioral part specifying behavioral conditions regarding A that candidate services should match.
- A constraints part that specifying the security properties and any other constraints, which services that can substitute for A should satisfy. Constraints are specified by logical expressing defining atomic or complex conditions over the contents of service descriptors in service registries. In the case of security constraints, these conditions refer to service security certificates specified according to the ASSERT4SOA security certificates schema.

In Checkout process example, let us assume that the service bound to the activity Payment does not satisfy the confidentiality property for customer information, i.e., the customer's credit card and address information. To restore this property there are two options: to find an alternative service for Payment for which there is a security certificate indicating that the service satisfies the property, or to generate a workflow of services that would satisfy the property. In Payment could, for example, be substituted for by a PayPal like service, which does not require

credit card details. Assuming, however, that the latter service is based on a workflow like the Express Checkout Purchase of PayPal Payment in Checkout would need to be replaced by a work-flow of three activities: Set Express Check Out, Express- Check Out Payment and Prepare Order, as shown in Fig. 5. As in PayPal, the former of these activities creates a transaction token that is used to take payment from a customer without passing on his/her credit card details.

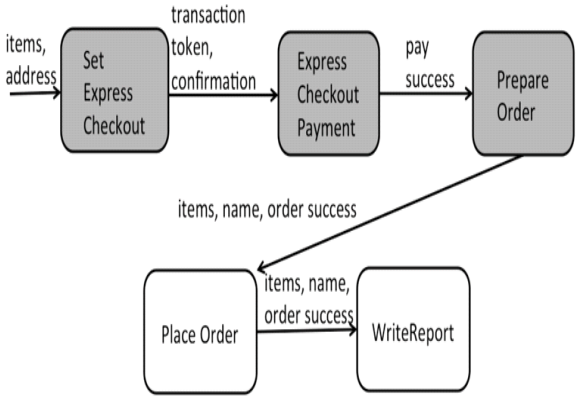

Fig 1.10. Modified Checkout process

The change in the Checkout workflow shown in Fig. 1.9 is realized as follows. When the algorithm GENER- ATESECUREWORKFLOWS is called with a query Q for replacing the service Payment in the original Checkout process, it retrieves the PayPal like payment services workflow shown by the grey activities in Fig. 1.9. This workflow matches structurally and behaviorally with Payment Subsequently, GENERATESECURE-WORKFLOWS infers the alternative security plans for this workflow using the confidentiality property that Payment failed to satisfy, and tries to find services that (a) match with the abstract PayPal workflow and (b) satisfy the security properties in one security plan.

CONCLUTION

In this paper, we have described a framework for verifying the security of SBS work flows based on patterns. The verification process is realized by inferring security properties of the individual services of the workflow (ASP) which, when satisfied, would guarantee workflow level security properties (RSP), and checking if the individual services have such properties.

The results of an initial evaluation of our approach were positive: even for workflows with 100 services and large SCO patterns sets (100 patterns) verification was performed in less than 0.3 seconds. This efficiency comes at the expense of completeness in verification. More specifically, our approach is not complete since it will only be able to verify an RSP property if: (a) there is a SCO pattern P that can guarantee RSP; (b) the abstract workflow structure of P, i.e., matches the workflow of interest; and (c) the partner services the workflow of interest that match the activity placeholders of the pattern workflow satisfy the ASP properties required of them by P. Hence, the identification and development of comprehensive sets of SCO patterns is a pre-requisite for the effectiveness and applicability of our approach. Identifying comprehensive pattern sets requires further research focusing not only on finding new patterns but also on establishing the right methodology for doing and for evaluating the sufficiency of the pattern sets developed using it.

REFERENCES

[1]. Albanese, M., Jajodia, S., and Molinaro, C. A Logic Framework for Flexible and Security- Aware Service Composition. IEEE 10th Int. Conf. on Autonomic and Trusted Computing, 2013, 337-346

[2]. Bartoletti, M., Degano, P., and Ferrari, G. L. Types and effects for secure service orchestration. In 19th IEEE Computer Security Foundations Workshop. 2006, 57-69.

[3]. Brucker, A. D., Compagna, L., and Guilleminot, P. Compliance Validation of Secure Service Compositions. In Secure and Trustworthy Service Composition, 2014, 136-149. Springer Int. Pub.

[4]. Charfi, A., and Mezini, M. Using aspects for security engineering of web service compositions. In IEEE International Conference on Web Services, Proceedings. 2005, 59-66.

[5]. Chinnici, R., Moreau, J. J., Ryman, A., & Weerawarana, S. Web services description language (wsdl) version 2.0 part 1: Core language. W3C recommendation, 26, 2007, 19.

[6]. Deubler, M., et al. Sound development of secure service- based systems. In Proc. of the 2nd Int. conference on Service oriented computing, 2004, 115-124.

[7]. Dijkman, R. M., Dumas, M., & Ouyang, C. Semantics and analysis of business process models in BPMN. Information and Software technology, 50(12), 2008, 1281-1294.

[8]. Dong, J., Peng, T., and Zhao, Y. Automated verification of security pattern compositions. Information and Software Technology, 52(3), 2010, 274-295. DOI: 10.1016/j.infsof.2009.10.001

[9]. Drools. Available from: http HYPERLINK "http://www.drools.org/": HYPERLINK "http://www.drools.org/"//www.HYPERLINK "http://www.drools.org/". HYPERLINK "http://www.drools.org/"drools HYPERLINK "http://www.drools.org/". HYPERLINK "http://www.drools.org/"org/

[10]. Dufour, J. M., etal. Simulation-based finite-sample tests for heteroscedasticity and ARCH effects. Journal of Econometrics, 122(2), 2004, 317-347.

[11]. Forgy, C. L. Rete: A fast algorithm for the many pattern many object pattern match problem. Artificial intelligence, 19(1), 1982, 17-37. DOI: 10.1016/0004-3702(82)90020-0

[12]. Frankova, G., Massacci, F., and Seguran, M. From Early Requirements Analysis towards Secure Workflows. In Trust Management: Proceedings of IFIP Joint ITrust and PST Conferences on Privacy, Trust Management and Security, Springer 2007.

[13]. Gürgens, S., Ochsenschläger, P., & Rudolph, C. On a formal framework for security properties. Computer Standards & Interfaces, 27(5), 2005, 457-466.

[14]. IBM, IBM Industry Packs. Available from: http://w HYPERLINK "http://www-/"ww HYPERLINK "http://www-/"-01.ibm.com/software/integration/business-process-manager/industry-packs/library/documentation/

[15]. Lelarge, M., Liu, Z., and Riabov, A. V. Automatic com- position of secure workflows. In Proceedings of the Third inter- national conference on Autonomic and Trusted Computing, 2006, 322-331. Springer-Verlag. DOI: 10.1007/11839569_31.