December 2004

A Business Process Oriented Approachto Secure Web Services

Idir Bakdi
University of Regensburg

Jochen Speyerer
University of Erlangen-Nuremberg

A Business Process Oriented Approach to Secure Web Services

Idir Bakdi
University of Regensburg, Germany
bakdi@forwin.de

Jochen Speyerer
University of Erlangen-Nuremberg, Germany
speyerer@forwin.de

ABSTRACT

Recently, interest in Web Services has grown throughout the IT community. Especially when it comes to application integration, the employment of Web Services seems to be a promising approach. But despite the advantages of this technology, its deployment, particularly in the inter-organizational domain, remains very sparse. As studies show, companies are reluctant to use it mainly due to security concerns. In this paper we show how to improve the security of Web Services protecting them against "semantic" attacks by considering entire business processes instead of single method invocations. We propose a solution consisting of an authorization engine, which makes its decisions about the admissibility of a given call taking the relations between successive requests into account. Further, we sketch an implementation and explain how a modeling formalism such as the Business Process Execution Language for Web Services (BPEL4WS) can help to realize it. Concluding with an analysis of weak points, we pinpoint possible areas of future research activities.

Keywords

Web Services, Security, Business Processes, Authorization.

INTRODUCTION

Web Services are slowly moving beyond the hype and are becoming a serious technology as more and more companies are announcing their adoption. The problem with Supply Chain Management systems, to take an example for inter-organizational application integration, is their inherent need for a time consuming and costly integration of the software used by all participants. Unfortunately, this setback can lead to other problems. Once the integration has been accomplished, the nodes are coupled in a very tight way, leading to a rigid formation of enterprises rather than a flexible, agile and loosely coupled network. Especially for small and medium sized enterprises the decision to join in a network of organizations and the costs associated with it are most likely irreversible and the installed system can therefore thwart the ability to form new partnerships.

One purpose of service-oriented architectures is to overcome the mentioned problems with monolithic systems. With the help of the Web Services Description Language (WSDL) companies can expose their functionalities as well as the corresponding format and semantic aspects. Potential partners of a supply net may rummage Universal Description, Discovery and Integration (UDDI)-registries in order to find a desired service. Method calls between systems are hereby encoded using the Simple Object Access Protocol (SOAP) over standard protocols such as the Hypertext Transfer Protocol (HTTP), the File Transfer Protocol (FTP) or the Simple Mail Transfer Protocol (SMTP).[1] Though the concept of Remote Procedure Calls (RPC) is well known, Web Services present the first common standard that focuses on dynamic business networks (Iyer, Freedman, Gaynor and Wyner, 2003). Besides RPC messaging, Web Services also support document style messaging (McCarthy, 2002; Snell, Tidwell and Kulchenko, 2002). When speaking of method invocations in this paper we include both synchronous and asynchronous messaging style.

Of special interest are the improvements in flexibility and the ease of integration. But despite the clear advantages of Web Services, they are not yet in widespread use. In the majority of cases, the technology has been adopted only for small projects or for feasibility studies. In order to tap the full potential of Web Services, they need to be applied to a wide range of inter-organizational applications. As many studies unveil, the main reasons for the reluctance are security concerns (BEA, 2003; Cap Gemini Ernest & Young, 2002). The results of another survey (META Group, 2003), recently conducted by a consulting company, in which 262 enterprises were questioned about Web Services and their restraints or shortcomings, is shown in Figure 1.

[1] For a comprehensive description of WSDL, UDDI and SOAP see (Newcomer, 2002; Apte and Mehta, 2002; Snell et al., 2002).

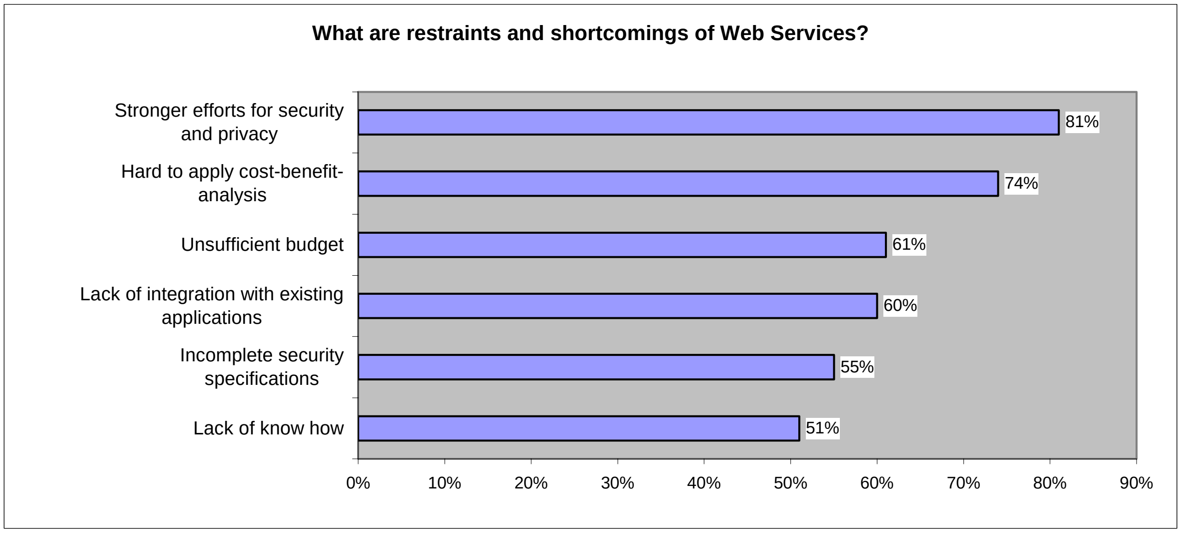

Figure 1. Restraints and shortcomings of Web Services (based on META Group, 2003)

Security concerns surrounding Web Services can be traced back to different facts. In the context of this paper, we identified the following four important reasons:

1. To some extent Web Services are used to deliberately bypass existing security mechanisms. SOAP messages are "tunneled" over protocols such as HTTP or SMTP which are not filtered by firewalls. This opens up a security hole and is directly in contradiction to what firewalls are meant for (Schneier, 2000).
2. Implementing Web Services can only provide noticeable benefit to a company if they can be used to realize parts of the business logic instead of simply receiving stock quotes or exchange rates. But by using them for important and critical business functions, the lack of security may lead to fatal consequences both for the company providing the services and for the legitimate users.
3. Many applications include implementation flaws which can be exploited by an attacker. This danger especially arises in large systems or software that has been enhanced over time to match a changed business environment. A common field of application for Web Services is to use them for wrapping legacy systems, so that they can be called via standardized interfaces and protocols (Beimborn, Mintert and Weitzel, 2002). Appraising the security of such systems is complicated and even if existing weaknesses are known, it is difficult to subsequently secure these applications.
4. One goal of business software is to automate processes without involving human decision makers. This requires an infrastructure capable of maintaining all critical secrets like private keys or other authentication tokens.

When deployed internally Web Services are not as strongly exposed to possible attacks as in inter-organizational settings. This is one reason why the technology is currently used for enterprise application integration rather than for business-to-business integration. But risks still exist in the former case and the considerations of this paper, although focusing on an inter-organizational use, also apply to Web Services employed within a single company.

Our research methodology is based on three pillars: literature review, evaluation of products and the development of a prototype. Due to limited availability of research material on securing Web Services in inter-organizational applications the development of the presented framework has been primarily inductive. Thus, we decided to evaluate the capabilities of commercially obtainable products in addition to a literature review. We discuss discovered gaps and derive the herein proposed concept.

The solution we present in this paper consists of an authorization engine that we develop at the Bavarian Research Network for Information Systems (FORWIN). We start with an overview of existing security standards for Web Services and a summary of products already available. Before discussing an example involving Web Services deployed in an inter-organizational environment, we describe the concept of an authorization engine making its decisions based on the observation of a whole business process. We then sketch a possible implementation of such a solution. Finally, we conclude with a discussion of the strengths and weaknesses of the approach and an outlook on possible areas of future research.

EXISTING SOLUTIONS

In this section we give an overview of existing efforts to protect Web Services. The first subsection contains a short summary of relevant security standards[2] and explains why, taken alone, they do not suffice. The second part describes the concept of "XML application firewalls" and shows in turn their deficiency.

Security Standards for Web Services

Many efforts have already been made to standardize security mechanisms for Web Services. A lot of work has been done by the Organization for the Advancement of Structured Information Standards (OASIS, 2004) on one hand and by the World Wide Web Consortium (W3C, 2004) on the other hand. Table 1 summarizes the main specifications of OASIS and W3C concerning the security of Web Services.

Abbreviation	*Standard*	*From*	*Description*
SAML	Security Assertion Markup Language	OASIS	Exchange of authentication and authorization data
	WS-Security	OASIS	SOAP extension including XML security protocols
XACML	Extensible Access Control Markup Language	OASIS	Description of access control policies
XCBF	XML Common Biometric Format	OASIS	Exchange of biometrical data
	XML Digital Signature	W3C	Integrity and non-repudiation of digital content
	XML Encryption	W3C	Confidentiality of digital content
XKMS	XML Key Management Specification	W3C	Key management
XrML	Extensible Rights Management Language	OASIS	Digital rights management

Table 1. The main security standards for Web Services

These standards are necessary to assure the authenticity, the confidentiality, the integrity and the non-repudiation of SOAP messages. However, their mere existence is not enough to guarantee an overall protection for Web Services. This is because they only constitute single building blocks, which have to be appropriately employed in conjunction with other application-specific protocols. Besides, existing standards are often not sufficiently deployed and when implemented, serious errors frequently occur. Predetermined security frameworks consisting of encryption and decryption functions and of methods for the generation and verification of digital signatures help to avoid flaws, but unfortunately many application developers do not make use of them. Moreover, logical errors in the Web Services themselves, such as those presented in the example of the next section, can not be repaired by introducing new security standards. A possible way to protect potentially flawed and thus vulnerable Web Services is to use a firewall which verifies incoming messages.

XML Application Firewalls

The concept of firewalls is widely known and frequently used to filter out inadmissible network traffic. Depending on the ISO/OSI[3] layer the firewall is acting at, we distinguish between "packet-filtering" (layer 3) and "circuit-filtering" (layer 4) firewalls. The former filter data packets based on their source and destination address or the port number they are sent to. Circuit-filtering firewalls on the other side observe entire TCP sessions. They prevent, among other attacks, so called "session hijackings", which allow an attacker to take over control of a connection initiated by another computer.

However, to secure Web Services it is not enough to block ports and to filter IP packets. Network traffic looking innocuous at the lower ISO/OSI layers may constitute an attack at the application layer (Middendorf, 2003). Unsuspicious IP packets can carry a SOAP message containing harmful code (Polster, 2003). As the security of the IT infrastructure (i.e. operating

[2] (Polster, 2003) contains a short description of these standards. A more comprehensive presentation can be found in (Hartmann, Flinn, Beznosov and Kawamoto, 2003).

[3] ISO/OSI stands for the International Standards Organization/Open System Interconnection reference model. See e.g. (Forouzan, 1998).

systems, web servers etc.) is getting hardened, more and more attacks appear at the application level. An example for this is the so called “SQL injection”. There are web servers which do not take the precaution of ensuring that requests they get from a client are well formatted. Instead, they just take the received parameters and send them as part of an SQL query to the database. By feeding such a server with fake parameters, an attacker may access protected data or change its content (McDonald, 2004; Middendorf, 2003). Just as web applications can be fooled, Web Services too may be called with flawed parameter values. As the parameters expected by a Web Service are precisely described in WSDL, an attacker could invoke it using values that do not conform to the specified ones.

Many security solution providers have recognized the need for firewalls dedicated to the protection of Web Services. They already offer solutions called “XML application firewalls”. Some examples are XML Filter from Microsoft®, Inc. (Microsoft, 2002), XML Firewall from Westbridge Technology, Inc. (Westbridge, 2004), SOAP Content Inspector™ from Quadrasis, Inc. (Quadrasis, 2004), as well as the products of Reactivity, Inc. (Reactivity, 2004), Vordel (Vordel, 2004), and Check Point™ Software Technologies, Ltd. (Check Point, 2004). (Middendorf, 2003) contains a description of products from some other providers. These solutions differ as to the risks they handle. The main capabilities of XML application firewalls include:

- Validation of SOAP messages to check that they match the specified XML Schemas (syntax verification)
- Authentication and authorization of the caller
- Digital signature to assure the integrity of SOAP messages
- Encryption to assure the confidentiality of SOAP messages
- Protection of the underlying transport layer through TLS/SSL and IP based authentication (SASL)[4]
- Detection of dictionary attacks on passwords and denial of service attacks

The spread of XML application firewalls will increase when more organizations start to deploy Web Services. It is important to note that application level firewalls must not be seen as a substitution for firewalls acting at the lower ISO/OSI layers. The two kinds of firewalls, which some providers already integrate into a single product (Check Point, 2004), should rather be used in a complementary manner to enlarge the spectrum of covered threats.

In a certain way, firewalls can be seen as a kind of authorization engines that have to decide whether a given action is admissible or not. In order to make appropriate decisions it is helpful to consider the whole chain of actions, instead of just looking at the current request. A deficiency of existing XML application firewalls is that they only examine single method invocations. The relations between successive calls are not considered. Yet, many attacks consist of a combination of requests, which differs from those appearing in regular use. SQL injection attacks succeed because systems are called with parameters they were not intended to receive. Similarly, one could imagine attacks on Web Services in which several methods are invoked in a combination and with parameters that the implementation was not designed to handle. This sort of “semantic” attack is what the approach we propose is aiming to protect against.

NEW APPROACH

We present a solution which improves the security of Web Services by imposing additional obstacles on a possible attacker. Firstly, we describe our approach which consists of an authorization engine making its decisions based on the observation of an entire business process. In the third subsection we illustrate our idea using an example involving Web Services deployed in an inter-organizational setting. Finally, we sketch a possible implementation.

Description

Usually, a Web Services provider offers a number of methods to be remotely invoked. They can be called in an arbitrary combination and with any parameters. But only a very small fraction of these possibilities is sensible from the application’s point of view. When buying an item for example, one first asks for its price and pays after that, not the other way around. The meaningful combinations can be modeled as business processes. Now, the idea is to execute Web Services only in the context of an instance of a business process. If an activity is not currently planned according to the business process, it will not be carried out. This way random access to potentially vulnerable Web Services can be avoided. The approach is meant to improve security by limiting access to Web Services in the same way that “design by contract” (Meyer, 1992) is meant to improve the reliability of software by imposing preconditions on method invocations to prevent erroneous calls.

[4] For a description of TLS/SSL and SASL see (Rescorla, 2000; Joseffson, 2004).

Figure 2 shows where an authorization engine enforcing this restriction would be situated in the overall architecture. It is important to note that it constitutes an additional security measure and is not meant as a replacement of other mechanisms.

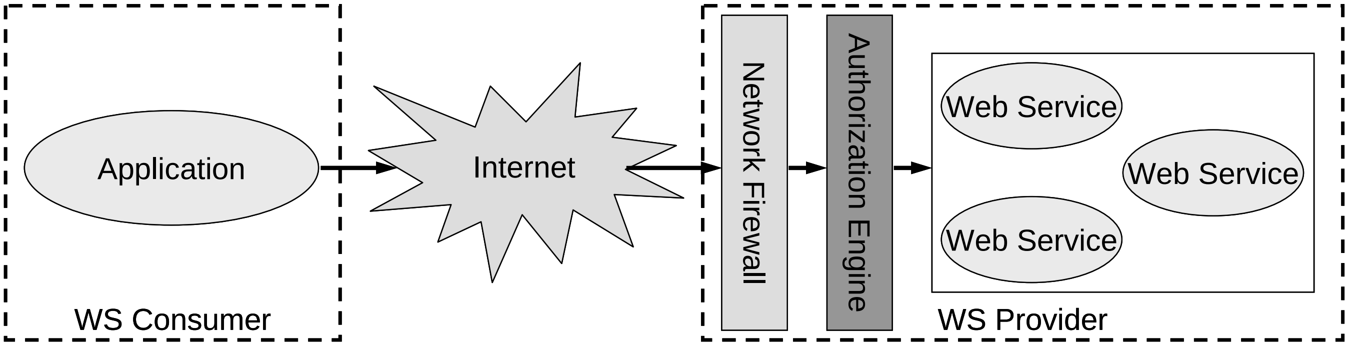

Figure 2. The authorization engine's position in the overall architecture

The approach resembles somehow that of intrusion detection systems (IDS), which monitor IT systems in order to detect anomalies indicating an attack. The difference consists in the way authorization decisions are made. Because it is impossible to model all admissible scenarios explicitly when protecting computer networks or operating systems, other methods are used to determine unauthorized activities. These can be divided into two categories. On one hand, there are IDS based on stochastic models of a system's normal behavior. Such models are created by observing the (legitimate) interactions taking place during a certain period. If significant deviations to the modeled behavior occur, an alarm is triggered. On the other hand, methods exist which are able to detect predefined attack patterns. They permanently monitor the IT system and send out a warning as soon as one of the patterns appears. Considering Web Services, it is not possible to model attack patterns, because it is unknown which weaknesses in the services can be combined to form a threat. But in general, it is possible to precisely define the business processes the Web Services are intended to participate in.

Example

Throughout this section we will explain our concept of an authorization engine to secure Web Services based on a fictitious example. Let's consider a company providing IT services for travel agencies and the necessary steps to arrange and book a journey. There are seven different Web Services available to cover the business logic of this task: *Get Airfare Quotes*, *Get Hotel Quotes*, *Choose Flight*, *Choose Hotel*, *Book Trip*, *Payment Received* and *Send Tickets*. By using the first two services, the travel agencies are able to inform their clients about available flights and hotel rooms. *Choose Flight* and *Choose Hotel*, are then used to select the desired itinerary and accommodations, respectively. By executing the *5th* Web Service the system books the chosen flights and hotels. The method *Payment Received* notifies the server about the incoming payment and the last service is finally used to trigger a mailing of the documents to the customer.

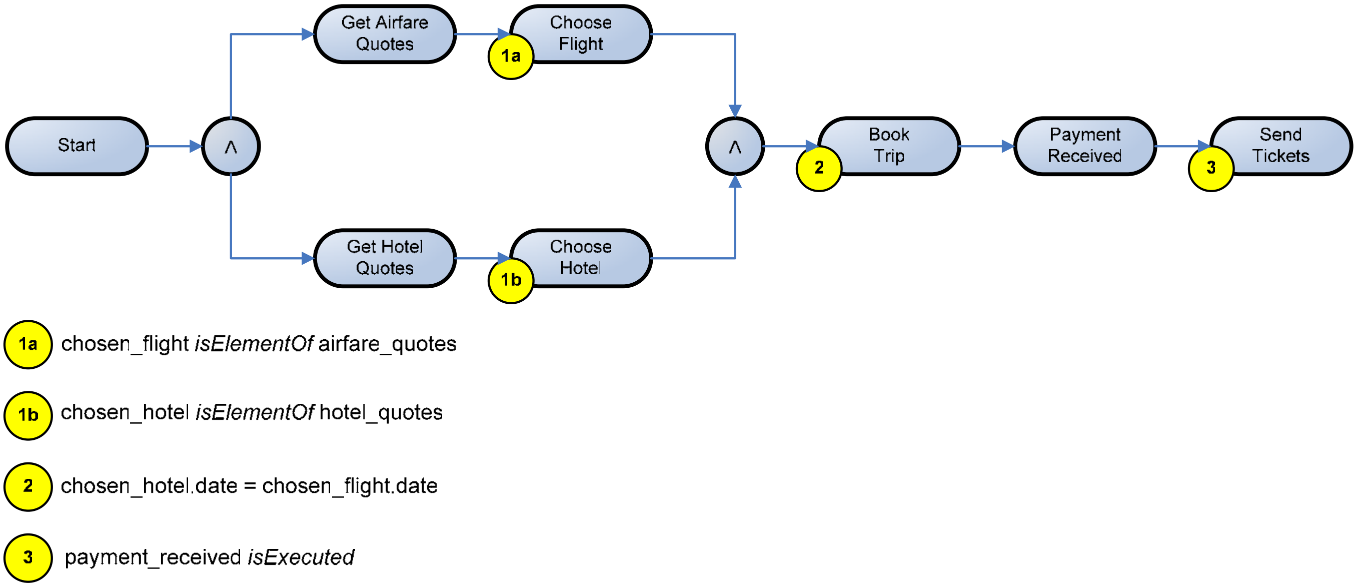
Get Airfare Quotes
Choose Flight
1a
Start
^
^
2
Book Trip
Payment Received
3
Send Tickets
Get Hotel Quotes
Choose Hotel
1b
1a chosen_flight *isElementOf* airfare_quotes
1b chosen_hotel *isElementOf* hotel_quotes
2 chosen_hotel.date = chosen_flight.date
3 payment_received *isExecuted*

Figure 3. Business process "journey booking"

By using an XML application firewall one can assure the integrity of exchanged messages, the confidentiality of data about customers, the authentication and authorization of the connected travel agencies as well as the bindingness of a booked trip. Nevertheless, an authorization engine can help in further increasing security as shown in the following. Consider the business process depicted in Figure 3. In order to protect the system against "semantic" attacks, which may include method calls with malicious parameters or the invocation of several Web Services in an improper combination, the following rules could be specified:

1. The parameter for the Web Service *Choose Flight* must match one of the results from the previous call to *Get Airfare Quotes*. The same is true for *Get Hotel Quotes* and *Choose Hotel*. Applying this rule prevents a potential attacker from exploiting possible implementation flaws in the *Choose Flight* and *Choose Hotel* Web Services, which may otherwise lead to unforeseen effects, for example, if dates are used during which the hotel or the flight is not available or sold out.
2. Before calling *Book Trip*, the system needs to check that the dates for the flight and the hotel correlate. This rule can avert the following attack: Assume that the applicable rates are determined on the basis of the selected flight. Further, consider the case where the Web Service used to cancel already booked trips is inconsistently implemented such that the season and therefore the amount of the back payment is based on the chosen hotel. In such a situation and without the described examination of the method invocations, an attacker would be able to book a trip by selecting a flight in the post season and a hotel in the peak season. If s/he is canceling the tour afterwards and if the difference in prices is high enough, the attacker would make a profit, although cancellation fees may be charged.
3. Only after the *Payment Received* service was executed successfully, the system may allow a call to *Send Tickets*. This prevents fraudulent use which is possible if the authorization check of the last Web Service is vulnerable.

Possible Implementation

The attacks we are protecting against concern malicious invocations of Web Services. Given a call to a method an authorization engine has to decide, based on the history of the current instance of the business process and on the parameters of that call, whether it should be executed or not. In this section we give a brief overview of a possible implementation for the proposed solution. After describing the main components of the system we show how a modeling formalism such as BPEL4WS can help to realize it.

System Components

First of all, we must distinguish between design-time and run-time issues. The components of the system can be categorized into two classes as follows:

1. Components concerning both design-time and run-time:

    a. *Service (In, Out):*

    Services are the objects to protect. Each service has got two parameters. The parameter "In" represents all the input data transmitted to the service when invoked. "Out" stands for the result data returned by the service after completion. A service potentially contains (implementation) flaws.

    b. *Role:*

    By "role" we mean a role in the sense of access control systems (Ferraiolo, Kuhn and Chandramouli, 2003). That is, only subjects belonging to a certain role are authorized to invoke a given service. We will not further consider the act of associating subjects with roles. Assertions concerning this association are received as SAML documents from the authentication process. The methods used for the authentication might be faulty, too.

    c. *Business Process (Constraints):*

    Hereby we mean a business process (BP) modeled from the application's point of view, not from a technical one. The atomic tasks which make up a BP are the services. A BP determines the temporal and logical order as well as the combination in which the services have to be executed. Together with the BP, security constraints may be specified, which determine conditions for granting access rights to the services. The simplest (degenerate) BP would consist of a single service. A constraint could indicate the roles allowed to invoke this service.

2. Components concerning run-time only:

    a. *Business Process Instance:*

    At runtime there may be one or more incarnations of a BP. One such incarnation is called a "business process instance" (BPI). Comparing this to an object oriented programming language, BPs would correspond to classes and BPIs to objects.

    b. *Business Process Instance ID:*

    Each BPI is uniquely identified by a so called "business process instance ID" (BPI-ID). Every time a caller invokes a service s/he uses such a BPI-ID to specify the BPI s/he is referring to. That is, services can only be executed within the context of a running BPI.

    c. *Invocation (inVal, outVal, contextVal):*

    When a service is invoked with the parameter value "inVal" and returns "outVal" as the result, we speak of an "invocation (inVal, outVal, contextVal)", where "contextVal" sums up all relevant context information of the service call. contextVal consists of the calling time, the role of the caller, the BPI-ID of the business process instance the call belongs to, etc.

    d. *History:*

    The history is a repository for all the invocations executed so far along the belonging inVal, outVal and contextVal.

    e. *Security Rules:*

Security rules are (automatically) generated from the modeled BP and the corresponding constraints. They determine in which context a given invocation is admissible.

f. *Entities:*

Every variable (role, input and output parameters, BPI-ID, time, etc.) used to decide whether a call to a service should be executed or not is called an "entity". Entities are the variables the security rules refer to.

g. *Evidence:*

At run-time the entities are assigned concrete values. The sum of this information we call "evidence".

h. *Authorization Engine:*

The "authorization engine" decides whether a given call to a service is admissible or not. To do so, the currently available evidence (which comes from inVal, contextVal and the history) is assigned to the appropriate entities. Then the generated security rules are evaluated in order to make a decision.

BPEL4WS

BPEL4WS is a programming abstraction that allows developers to compose multiple synchronous and asynchronous Web Services into an end-to-end business flow. It provides for the modeling of business processes made up of service invocations. BPEL4WS is an extensible language permitting the introduction of new elements to satisfy special needs. We think that it can be used even without such an extension to help implement the above presented system. All the described components can be expressed using constructs contained in BPEL4WS.

The history (i.e. the current state of the BPI) can be traced using „variables". In BPEL4WS variables are state variables permitting the maintenance of the state of a business process. Variables associated with message types can be specified as input or output variables. This is how to store the InVal and OutVal parameter sets mentioned above. By assigning concrete values to the variables during execution (using BPEL4WS' assign activity) evidence is gathered. More evidence lies implicitly in the path taken by the business process instance. Recall that a BP may contain different paths from which the BPI may choose one according to preset conditions at run-time.

Correlations are suitable for implementing the concept of BPI-IDs. In BPEL4WS "(global) correlation sets" are used as aliases for the identities of business process instances. Using them the protocol can figure out which BPI a given message belongs to. The In and Out parameters can be represented by "message properties". In WSDL each operation of a port type has got input and/or output messages.

The constraints leading (together with the structure of the BP) to rules could be modeled through "join conditions". In BPEL4WS each activity has a join condition. If this Boolean expression evaluates to true, the corresponding activity is executed else it is not. Due to the use of XPath 1.0 to specify join conditions in the current version of BPEL4WS the expressiveness of constraints would however be restricted, as, for example, it is only possible to do calculations with integers (i.e. divisions are not permitted) and strings can only be checked for equality but not arbitrarily manipulated. But in the future, richer languages like XQuery (Brundage, 2004) are expected to be used in conjunction with BPEL4WS enabling the formulation of more complex constraints. In practice, the modeling of a BP can be supported by appropriate tools, which could make suggestions about the constraints and group them hierarchically to facilitate their administration.

It is important to note that using BPEL4WS is just one option. The components we defined abstractly in the last subsection could also be implemented using other languages. For example if a business process management engine is already in place its formalism should be used in order to avoid modeling the same workflows twice.

CONCLUSION

The standards surrounding Web Services still have to be seen a work in progress, and some aspects need further developments. Nevertheless, we clearly see advantages of using Web Services in the area of flexible integration and therefore the necessity of securing them.

This paper presents an authorization engine that implicitly restricts potential attackers rather than explicitly forbidding them to execute certain operations. The proposed solution will not always thwart a misuse of the services. A denial of service attack for instance has to be prevented by other means. However, especially the danger of a "semantic" attack is reduced, which improves the overall security.

The main drawback of our approach is the additional overhead it causes. The study we mentioned at the beginning (META Group, 2003) states that enterprises are concerned about the costs of securing Web Services. The costs introduced by our approach are mainly due to the necessity of modeling the business processes. Each of their facets, such as the relationships between exchanged massages, has to be specified in detail. This disadvantage is however relativized by the fact that a neat modeling of the business processes is useful, independent of security considerations. It helps optimizing the processes and improving their quality.

As with most research fields there are many areas for improvement and future study. So far we concentrated on securing a set of Web Services hosted by a single server on which the authorization engine runs. It would be interesting to examine how to extend this solution to the case of multiple Web Services providers. A future goal could also be the development of a tool which can be used as a plug-in in integrated development environments like Microsoft's Visual Studio® (Microsoft, 2004) or the Eclipse project (Eclipse, 2004). Such a tool could support the modeling of business processes and the specification of security constraints and could use the results to automatically generate the rules applied by the authorization engine.

REFERENCES

1. Apte, N. and Mehta, T. (2002) UDDI: Building Registry-Based Web Services Solutions, Prentice Hall, New Jersey.
2. BEA Systems (2003) Umfrage unter europäischen CIOs: Web Services werden das Geschäftsleben revolutionieren, *http://de.bea.com/presse/2002/020926_a.jsp*, download 2003-07-07.
3. Beimborn, D.; Mintert, S. and Weitzel, T. (2002) Web Services und ebXML, *WIRTSCHAFTSINFORMATIK*, 44, 3, 277-280.
4. Brundage, M. (2004) XQuery – The XML Query Language, Addison-Wesley Professional, Boston.
5. Cap Gemini Ernest & Young (2003) Der Markt für Web-Services - Erwartungen, Treiber, Investitionsabsichten, *http://www.de.cgey.com/servlet/PB/-s/qhm78e1ugl851mfjda9jbh53w2b0404/show/1004620/Web-Services.pdf*, download 2003-08-29.
6. Check Point™ Software Technologies, Ltd. (2004) Securing Web Services, *http://www.checkpoint.com/products/fp3/webservices_overview.html*, download 2004-01-23.
7. Eclipse Project (2004), *http://www.eclipse.org*, download 2004-04-05.
8. Forouzan, B. (1998) Introduction to data communication and networking, McGraw-Hill, Boston.
9. Ferraiolo, D. F.; Kuhn, D. R. and Chandramouli, R. (2003) Role-Based Access Control, Artech House, Boston, London
10. Hartmann, B.; Flinn, D., J.; Beznosov, K. and Kawamoto, S. (2003) Mastering Web Services Security, Wiley Publishing, Inc., Indianapolis.
11. Iyer, B.; Freedman, J.; Gaynor, M. and Wyner, G. (2003) Web Services: Enabling dynamic business networks, *Communications of the Association for Information Systems*, 11, 525-554.
12. Josefsson, S. (2004) SASL Overview, *http://www.sens.buffalo.edu/UBiquity/software/gnu/doc/web/share/doc/gsasl/html/SASL-Overview.html*, download 2004-01-23.
13. Meyer, B. (1992) Applying "Design by Contract", Computer, 25, 10, 40-51.
14. McCarthy, J. (2004) Reap the benefits of document style Web services, *http://www-106.ibm.com/developerworks/webservices/library/ws-docstyle.html*, download 2004-04-05.
15. McDonald, S. (2004) SQL Injection: Modes of attack, defence, and why it matters, *http://www.sans.org/rr/papers/3/23.pdf*, download 2004-01-22.
16. META Group Deutschland AG (Ed.) (2004) Web Services gewinnen an Fahrt, aber Sicherheits- und ROI-Bedenken bremsen das Wachstum, *http://www.metagroup.de/presse/2003/pm16_07-07-2003.htm*, download 2004-02-04.
17. Microsoft®, Inc. (2002) Ensuring Trusted Web Services, *http://www.microsoft.com/presspass/features/2002/feb02/02-18xmlfilter.asp*, download 2004-01-23.
18. Microsoft® Visual Studio® (2004), *http://msdn.microsoft.com/vstudio/*, download 2004-04-05.
19. Middendorf, S. (2003) *Sicher bedient – Sicherheit von Webservices*, iX, 3, 88.
20. Newcomer, E. (2002) Understanding Web Services – XML, WSDL, SOAP, and UDDI, Addison-Wesley, Boston.
21. Organization for the Advancement of Structured Information Standards (OASIS) (2004), *http://www.oasis-open.org*, download 2004-02-11.
22. Polster, R. (2003) IT-Sicherheit auf dem Prüfstand – Neue Herausforderungen durch Web Services, *OBJEKTspektrum,* 5, 42-49.

23. Quadrasis, Inc. (2004) SOAP Content Inspector, *http://www.hi.com/solutions/products/easi_product_packages/easi_soap.htm*, download 2004-01-23.

24. Reactivity, Inc. (2004) Reactivity XML Firewall, *http://www.reactivity.com/products/index.html*, download 2004-01-23.

25. Rescorla, E. (2000) SSL and TLS: Designing and Building Secure Systems, Addison-Wesley, Boston.

26. Schneier, B. (2000) SOAP, *Crypto-Gram Newsletter, http://www.counterpane.com/crypto-gram-0006.html#SOAP*, download 2004-01-25.

27. Snell, J.; Tidwell, D. and Kulchenko, P. (2002) Web Services with SOAP, O'Reilly & Associates, Inc., Sebastopol.

28. Vordel (2004) Firewalls and Web Services - myths and facts, *http://www.vordel.com/ knowledgebase/vordel_view3.html*, download 2004-01-23.

29. Westbridge Technology, Inc. (2004) XML Firewall, *http://www.westbridgetech.com/appfirewall.html*, download 2004-01-23.

30. World Wide Web Consortium (W3C) (2004), *http://www.w3.org*, download 2004-02-11.