A Pattern-driven Security Advisor for Service-oriented Architectures

Maxim Schnjakin
Hasso-Plattner-Institute
Prof.-Dr.-Helmert Str. 2-3
14482 Potsdam, Germany
+49 (0) 331 55 09 514
maxim.schnjakin@hpi.uni-potsdam.de

Michael Menzel
Hasso-Plattner-Institute
Prof.-Dr.-Helmert Str. 2-3
14482 Potsdam, Germany
+49 (0) 331 55 09 516
michael.menzel@hpi.uni-potsdam.de

Christoph Meinel
Hasso-Plattner-Institute
Prof.-Dr.-Helmert Str. 2-3
14482 Potsdam, Germany
+49 (0) 331 55 09 220
meinel@hpi.uni-potsdam.de

ABSTRACT
Service-oriented Architectures (SOA) provide a flexible infrastructure to allow independently developed software components to communicate in a seamless manner. Increased connectivity entails significant higher security risks. To face these risks, a broad range of specifications e.g. WS-Security and WS-Trust has emerged to ensure security in SOA. These specifications are supported by all major Web Service Frameworks and enforced by security modules provided by these frameworks to apply security to ingoing and outgoing messages. In general, a security module is configured declaratively using a security policy e.g. WS-SecurityPolicy that expresses security goals and related configurations. To support a broad range of use cases, these security policy languages offer a variety of settings and options.

However, the complexity of security policy languages leads to an error-prone and tedious creation of security policies. To simplify and support the generation of Web Services, we present an architecture for a security advisor in this paper. This security advisor facilitates the configuration of security modules for service-based systems based on a pattern-driven approach that enables the transformation from general security goals to concrete security configurations. Therefore, we will introduce a security pattern system which is used to resolve concrete protocols and security mechanisms at a technical level.

Categories and Subject Descriptors
K.6 [Management of computing and information systems]: Security and protection; D.2.11 [Software Engineering]: Software Architectures—Protection mechanisms

General Terms
Security, Languages

Keywords

1. INTRODUCTION
Security is an essential aspect of computing and communication systems. A broad range of security mechanisms and protocols has been defined in the last decades to secure data and systems. However, these traditional approaches generally presume a direct interaction of trustworthy participants that are not applicable in a loosely coupled environment that is based on messaging. For instance, Secure Socket Layer (SSL)/Transport Layer Security (TLS) establish a secure channel to realise a point to point data protection. Such an approach is not sufficient in service-oriented environments that facilitate the exchange of messages across intermediary systems and even across trust domains. In the scope of Web Services, WS-Security specifies a framework to ensure the confidentiality and integrity of SOAP-messages and to bind an identity to a message in order to guarantee authenticity. The security requirements for messaging are stated in security policies and enforced by security modules that are integrated in the Web Service Stack that provides the service. There exist several security policies languages that are standardized (e.g. WS-Security-Policy) or specific for a security module (e.g. Axis 2 Rampart Configuration). These policy languages offer a broad range of different settings and options. Depending on the module and the policy language, there might be various alternatives to define security rules in order to achieve specific security goals. Therefore, the configuration of communication systems requires a deep and detailed knowledge about the frameworks, their environments and appropriate technical specifications. This complexity can result in an error-prone creation of security policies. Some frameworks such as the Windows Communication Foundation support an automated generation of WS-SecurityPolicy documents based on selected options and profiles defined in configuration files. This approach simplifies the creation of policies, but still requires a detailed knowledge about the effects of the different configuration options and the relationships and side effects among them.

This paper presents an architecture for a security advisor that supports and simplifies the generation of security configurations for Web Services. Our advisor should support Web Service deployers and developers to set up a service in a secure manner. To represent expert knowledge about
security mechanisms and concepts, we foster the usage of security patterns. Our solution enables users without a strong background in security to configure the underlying framework with a minimal amount of information. Therefore, we introduce a platform-independent security model to gather security information and describe the usage of security patterns to resolve appropriate security protocols and methods.

Most current methodologies for applying security patterns in software development processes focus on early stages of the software life-cycle. In our solution, we define a formal pattern specification to enable reasoning on a set of security patterns. In this way, we bridge the gap between abstract security requirements and specific security policies that are generated based on our approach. In order to prove the applicability of our approach, we introduce a prototype implementation as well.

This paper is structured as follows. Section 2 introduces a domain-independent security model providing a foundation for the pattern-based approach that is described in Section 3. The concepts of Security Policies that are generated by our approach are introduced in Section 4. In the next Section, the architecture and implementation details of our security advisor are described. Section 6 describes related work, while Section 7 discusses and concludes this paper and outlines some suggestions for future work.

2. DOMAIN INDEPENDENT MODEL

In this chapter, we will introduce our domain-independent security model, which enables a generation of security configurations in compliance with the model-driven paradigm. The model allows to generalize from concrete security functions provided by specific software solutions to general security goals. Furthermore, the model allows us to congregate the user requirements in terms of these security goals and to generate domain-specific configurations in various policy languages.

The structure of the model, the description of security requirements in a technical and policy language independent way, and the mapping to concrete security policies has been introduced in [21]. The model provides an abstract security layer, which is conceptual close to the OASIS reference model for SOA [14]. The security layer is designed to cover all security aspects in SOA environment and the relationship among affected entities. Thus, our model describes basic security goals,

- the relationship between specific security properties and mechanisms (in general, these mechanisms could be an algorithm (e.g., AES) or a protocol (e.g., WS-Security),
- the relations among security goals and affected entities by Constraints that are composed in a Security Policy,
- constraint as an articulate technique to enforce and guarantee a specific security goal.

Security constraints are related to a security level that describes the strength of this requirement. As shown in Figure 1, policies are evaluated and enforced by Security Module. Hence, certain Security Patterns support the Security Module and guarantee the related constraints. This security pattern defines the protocols and appointed conditions to enforce specific security goals. Consequently, once the security goal is selected, the resolution of the path through the net of existing patterns occurs automatically.

3. SECURITY PATTERNS

In this section, we will introduce the concept of security pattern that constitutes the foundation of our security advisor. Security patterns have been introduced by Yoder and Barcalow [22] in 1997 and are based on the idea of design patterns as described by Christopher Alexander ‘A pattern describes a problem which occurs over and over again in our environment, and then describes the core of the solution of that pattern’ [1]. As we have mentioned before, security patterns are usually defined in an informal way, generally in the natural language, to enable software and system designer to adopt the solutions described by the pattern to their own specific problem in a particular implementation context [20].

However, our goal is to enable an automated selection of appropriate patterns to gain the required information to generate enforceable security policies that are used to configure security functionality. One of the main steps to achieve this goal is to use a formal pattern specification as introduced
by Marcus Schumacher [19], which enables a reasoning on a set of security patterns. A pattern is composed as follows:

- **problem** - We define a problem as a set of security threats identified by a security goal. Therefore, this part of the pattern structure specifies all the security threats that can be solved by current pattern and relate to a security goal.

- **context** - The context describes the type of the security pattern that identifies its applicability. Buschmann et al. performed a classification of patterns and identified three main classes: Architectural Pattern, Design Pattern and Idioms [5]. The last one is less relevant to our work, since we are referring only to design and architectural patterns. In our approach context describes the security relevant environment in which the pattern operates. We differentiate between local, central and distributed environments. The property local implies that the entire information (needed to solve a particular problem) needs to exist on the local machine. Central indicates that a centralised security infrastructure is used (e.g. Kerberos). Distributed refers to the practice of providing security information as an internet-based service on-demand. For instance, this would imply the establishment of trust relationships.

- **forces** - Forces describe the conditions under which a pattern can be applied. For instance, it might be crucial for the solution of the problem whether two or more participants are involved in the communication process.

- **solution** - A pattern describes a strategy to solve a specific problem that is adapted by concrete security mechanism or security protocol. In our approach, a pattern solution identifies these techniques (e.g. SSL or WS-Security).

As mentioned before, different methods might enforce the same security goal. At the same time, various patterns may exist that specify various solutions to achieve the same security goal. In common with Zimmer [24] there are three general dependencies that might occur between security patterns: **Usage, Refinement or Conflict**.

Based on previous work in the field of security patterns [23], we defined a pattern system that is used by our security advisor. The correct pattern matching happens by means of proper conditions, which are given through forces. Figure 4 clarifies our approach by the example of two security goals: integrity and confidentiality. The figure illustrates two patterns, which are intended to secure data exchange over an insecure channel. The names of these patterns are SecurePipe and MessageConfidentiality. Each of them refers to particular security goal and identifies appropriate security technique.

### 4. SECURITY POLICIES

The increasing complexity of requirements in Service-oriented Architectures is addressed by a declarative approach to manage and define these requirements. This approach allows the communication and negotiation of policies at runtime. Policies are enforced by policy enforcement points that secure the services.

![Figure 2: Rampart policy to secure outgoing messages](image-url)

There are several definitions of security policy. Eckert defines a security policy as a description of rules, user's behaviors and security measures, which are should be taken to achieve the security requirements of a considered system or an organizational unit [6]. We define a security policy as a set of rules (constraints) that reflect a security configuration of security modules regarding a specific security goal. The format of security policies might vary. Even the same security module may support different formats. Apache Axis 2 Web Service Stack [16], for example uses a module called Rampart to provide security functionality [10]. Rampart can be configured using the WS-Policy language or Rampart specific policy assertions. The original Rampart configuration uses a flat list of constraints stated in XML with implementation specific information, while WS-Policy provides a grammar to express and group policy assertions describing a broad range of requirements on a more abstract level [2]. Since WS-Policy is not specific to a problem domain, security assertions are defined in the WS-SecurityPolicy specification providing an implementation independent approach to express constraints for WS-Security, WS-Trust and WS-SecureConversation [5].

In the scope of the native Rampart configuration, a policy is represented by the element 'parameter' with the attribute name='OutflowSecurity' | 'InflowSecurity'. The attribute defines whether the security settings are applied to outgoing or incoming messages. The policy element 'parameter' contains the element 'action' that encapsulates several elements to configure Rampart. Figure 2 shows an exemplary Rampart configuration for outgoing messages regarding the security goal confidentiality.

Confidentiality is enforced by Rampart by providing a policy file containing an 'items' element with the 'encrypt' value. The value 'body' within the element 'encryption-Parts' instructs Rampart to encrypt the whole message body. The values 'user' and 'encryptionUser' refer to the involved service endpoints - these elements reference the name of the involved objects to resolve the associated keys for encryption. Implementation specific details stating how to resolve the keys are declared in a Java property file that is referenced by 'encryptionPropFile'.
Finally, ‘encryptionSymAlgorithm’ and ‘encryptionKeyTransportAlgorithm’ specify the utilized algorithms.

5. APPLICATION ARCHITECTURE

In previous chapters we presented the concepts required to realize the automated and pattern-based policy generation process. In this chapter we introduce the implementation of the automated policy generation tool. We mentioned above, that a manual policy compilation process implies two general problems. The first problem is that it requires expert knowledge to perform the policy generation process. The second problem is that compiling policies for complex systems is a tedious and error prone task. In order to solve these problems, we foster the automated policy generation process by using security patterns. Furthermore, we introduce a special component within our application called security advisor. The security advisor enables users without a strong background in security to generate and deploy policies in order to provide services with the required protection.

A security expert plays an important role in the process of defining a policy. He has the necessary domain-specific knowledge to capture the stakeholders’ security requirements as high-level security goals. Furthermore, the security expert has profound knowledge of the security engineering area, which is required to choose and implement a well-proven method to achieve an abstract security goal. The policy advisor relies on this expert knowledge which has to be present in the application. For this reason, the application needs to be preconfigured by the security expert. The role of the administrator is to identify and specify all information relevant to derive of enforceable security policies (e.g. type and location of a keystore to be used, password needed to access the keystore). After this configuration step has been finished, a service developer can use the security advisor to derive an enforceable security policy based on the general security goals.

In this section we present three building blocks of our application:

1. Configuration of security patterns: The aim of this building block is to be able to capture the expert knowledge
2. Administration of the underlying infrastructure: This segment enables a security expert to perform framework specific refinements and to adapt patterns to the underlying infrastructure
3. Security advisor module: An important building block that enables the automated generation of enforceable policies.

Figure 3 illustrates the general structure of our application. First the domain expert knowledge is captured in abstract security patterns. In the next step, an administrator who is configuring the security advisor has to specify all relevant details of the underlying security-infrastructure, e.g. used framework. Finally, the web service deployer can use a graphical interface named “security advisor view” to generate a specific security policy to secure a certain web service. To sum up, we simplify the security configuration process by using security expert knowledge, which can be aggregated and stored in security patterns. In our approach security patterns are formalized to enable the reasoning on them.

Figure 4: Security pattern relationships

5.1 Configuration of Security Components

Prior to using the application, the domain experts should specify a set of patterns, which are specified in domain-specific terminologies. By using the graphical user interface of the application, the domain expert also specifies basic mechanisms and items, which are used (required) to enforce security goals as e.g. confidentiality or integrity. In this context the notion of a Credential represents an important part of our architecture. Credentials are generally used to certify a qualification or identity of a user issued by a third party. A detailed classification of security credentials was presented by Denker et al. [8]. In compliance with this work, we divide credentials in simple credentials (e.g. key, login, certificate) and composed credentials (e.g. SAML or WS-Security Token).

The usage of security patterns for solving security problems is the heart of our approach. In order to enable the reasoning based on a set of security patterns, we defined a formal pattern specification. Each pattern has four general parts, a context, a problem, a solution and forces. The context describes the environment, in which the pattern can be applied. Forces characterize the problem in the context. Finally, the solution proposes a general technology or configuration that provides a solution according to the forces in...
the context. It is important to note that the patterns we use for different abstraction levels have relationships among each other. These relationships reflect the conflicts, compatibilities and dependencies among the patterns. We separate three types of patterns: ‘security patterns’ at the top layer, ‘architecture patterns’ at the specification layer and ‘implementation patterns’ at the application layer.

An abstract security pattern addresses a specific security goal, represents a set of security threats and refers to an architecture pattern which specifies a solution to the given problem. Every architecture pattern comprises an implementation and parent pattern attributes. The former refers to the implementation pattern implying the security functionality of a concrete security framework. The latter refers back to the related pattern at the layer above.

Our current configuration of the application contains two security patterns at the top level: SecurePipe to secure data exchanged over an insecure channel and MessageConfidentiality to secure data exchanged using SOAP messages. Each pattern refers to a certain security goal and identifies an apposite security protocol. Both patterns are capable to solve the security goals message confidentiality and message integrity. The forces attribute of a pattern characterizes the conditions under which the problem can be solved. The forces of the security pattern MessageConfidentiality are described by two attributes: multiple-participants and message-based-handling. The first one indicates the applicability of the pattern to solve security problems in a multi-tenant communication environment. The latter attribute indicates that configuration should be applied on a message level. As possible solution, the WS Security specification provides a mean to protect a message by encrypting and/or digitally signing a body, a header, any combination of them or even parts of the message. On the other side, the security pattern SecurePipe is characterized by the attributes: bidirectional and channel-based-handling, which indicates that this pattern is applicable to enforce security goals in bidirectional communication scenarios. This occurs through establishing a secure channel between communication partners (via SSL protocol).

5.2 Administrating of the SOA Infrastructure

Various frameworks require specific refinements for handling security mechanisms. By using the administration view, the local domain administrator is able to specify relevant security details of the underlying IT infrastructure.

If we assume, that Rampart is applied to enforce message confidentiality, a local domain administrator has to specify the location of the keystore containing all public keys needed to encrypt the outgoing messages and the private key which is used to decrypt incoming data transmissions. To access the keystore, additional information is required, namely the password, provider and the type of the (available) keystore. The apposite template has to be composed in the configuration view. However, the required refinement has to be performed in the administration view of the application. Simply, the keystore template has to specify at least the name of the keystore to be used and the password needed to access the appropriate keystore.

Furthermore, Rampart uses a callback mechanism to retrieve the password - whereas the service developer needs to implement the password callback class. Hence, Rampart expects information about the location and access to the required password callback class. This framework specific property of Rampart can be managed in the administration view as well.

5.3 Security Advisor

After the domain expert knowledge has been captured in security patterns and refined by a local security administrator afterwards this knowledge can be utilized to generate the enforceable policy. In other words, we enable a web service developer to configure the underlying security module starting by choosing security goals and without having a deep knowledge about the underlying system. For this purpose we utilize a pattern-driven approach discussed in section 2.
On top of this, we define the algorithm to select a pattern and generate enforceable security policies. The algorithm makes use of the building blocks mentioned in the previous paragraphs in order to implement the automated policy generation process.

In Figure 5 we illustrate the basic functionality of our application. First, the application performs a full pattern scan to capture available security goals. After the user selected one or several security goals, it has to be clarified whether several strategies (in terms of patterns) are available to solve the selected problems. This occurs by evaluating the forces attributes of the qualified patterns. Once a single security pattern matches, the user is prompted to select the underlying security module. Depending on the web service framework further information might be required to generate an enforceable security policy. To gain insight into the processing engine of our application, we will use the security goal confidentiality as an example to follow the next steps.

5.3.1 Identify available security goals

To enable the user to select one or more security goals, the application browses through the abstract security patterns catalog.

5.3.2 Select security a goal

The user is asked to select one or more security goals, which should be applied to the underlying security module configuration in order to protect the deployed web service. As mentioned above, we will run through the policy generation process by choosing the security goal confidentiality.

5.3.3 Identify applicable pattern

In the case that the security goal is supported by only one pattern, the selection of the pattern (at the pattern level below) occurs automatically. Otherwise the application relies on the evaluation of the forces attributes of the patterns.

Both pre-configured security patterns SecurePipe and MessageConfidentiality are suitable to enforce the exemplary security goal confidentiality. Based on the fact, that different security patterns may enforce the same security goal, the application has to identify an appropriate protocol in the following step. As mentioned above, the matching of the appropriate pattern is guaranteed by matching the respective forces attributes. Our current solution aims at matching the suitable forces attributes by interacting with the user. We keep the questions and the requested decisions as simple as possible to enable even an "inexperienced" user to characterize the conditions and constraints of the underlying framework. Finally we utilize the user input to infer the optimal security configuration for the underlying web service framework.

In our example, it might be sufficient to identify whether two or more participants are involved in the web service communication in order to determine whether SSL or the WS-Security protocol should be applied to accomplish the exemplary security goal. This is due to the fact, that the SSL protocol is capable of establishing the data protection for only two endpoints. Assuming that the evaluation of the user interaction results in the need to secure parts of the messages exchanged in a multiple participant communication scenario, the application will match the security pattern MessageConfidentiality to ensure the intended purpose.

The implementation attribute of the selected pattern refers to the architecture pattern WS-Security, which implies the equal-named security protocol to be used.

5.3.4 Define framework specific information

The implementation attribute of the security protocol pattern refers to the framework specific implementation patterns (Figure 4). The patterns provide the basic information about the security capabilities of the provided security module. For instance, an implementation pattern can define the appliance of the AES-128 encryption standard to ensure message confidentiality. Another pattern might define the DES algorithm to ensure the same security goal. It cannot be taken for granted that any framework is capable of applying every encryption standard to ingoing or outgoing messages. Therefore, several patterns might be reasonable to ensure the same security goal at the application level.

However, after having selected the security goal and characterized the general conditions of the underlying web service, the user is prompted to select the security module, which is used to secure the deployed web service. The current release of our application provides implementation patterns only for the security module of Apache Axis2 framework - Rampart. Depending on the framework, further information might be required to generate a machine-readable security policy. This information will be discovered by the interaction with the user. In our example, the user has to specify whether Rampart security settings should be applied to incoming or outgoing messages (Figure 7). Furthermore, the user is requested to define the specification language of the resulting policy (either WS-Policy or Rampart specific configuration language).

5.3.5 Generate Policy

Having specified the security module requirements, the application executes the software component, which compiles the underlying implementation patterns according to the specified details to an enforceable policy. Figure 8 depicts the outcome of the described steps as a Rampart configuration for outgoing messages regarding the security goal confidentiality.

6. RELATED WORK

Figure 7: Rampart specific configuration
The concept of security pattern was introduced to support system engineers in choosing appropriate security solutions [18]. In the recent years, various security patterns have been defined that are expressed in an informal way - usually using natural language [13]. However, there is an increasing need in the description of patterns in a more formal way to enable their usage in the development life-cycle. In [17] Konrad et al. use UML to represent the patterns proposed by Gamma et al. [9]. UML diagrams describe structural and behavioral information, but they do not imply enough semantics for automating their processing.

Model-driven security and the automated generation of security enhanced software artifacts and security configurations have been a topic of interest in recent years. For instance SecureUML [12] is a model driven security approach for process-oriented systems focusing on access control. Similar to SecureUML, Juerjens presented the UMLSec extension for UML in order to express security relevant information within a system specification diagram. One focus of UMLSec lies on the modeling of communication-based applications, both do not lead for establishing the link between security intentions and model-driven generation of related security requirements.

In [11] Sanchez-Cid et al. presents security and dependability patterns which are used within a scope of SERENITY Project to incorporate security into applications. Sanchez-Cid uses a pattern-driven approach to allow running applications to react to the changing context by handling those solutions semi-automatically. Hence, the focus of the mentioned framework is the adaptation of security and dependability mechanisms for ambience intelligence ecosystems.

Bandara et al. propose a policy refinement process, which is based on the abductive reasoning technique [3]. In their work, they assume that the abstract policies are the consequences and the refined policies are the hypotheses. As a formal foundation of this approach, they use Event Calculus to specify the policies [15]. For each refinement of an abstract policy, the policy administrator consults the refinement patterns, which are defined in the paper by Darimont and Lamsweerde [7]. Generally, the refined policies are still abstract and not concrete enough, so that a set of enforceable policies cannot be generated from these refined policies. Therefore, the refinement process is repeated until all abstract policies are refined into a set of concrete policies.

In [4] Bhargavan et al. present an implementation of a plug-in for Microsoft Web Services Enhancements (WSE) which helps users to detect incorrect uses of WS-Security in SOAP processors. They propose a static rule-based tool for detecting typical errors in WSE configuration and policy files. The WSE Policy Advisor evaluates queries against a policy file and creates a textual report indicating what kind of security vulnerability may exist. In addition, the tool suggests, also in a textual manner, modifications to the original policy to reduce the detected vulnerability, but without giving strong security guarantees. As our work is neither limited to a specific framework nor any XML-based domain-specific languages, such a static checker could be used to find problems with policies generated by our security advisor and help to improve them.

7. CONCLUSIONS AND FUTURE WORK

In this paper we presented an architecture and an implementation for a security advisor that simplifies the generation of security policies in Service-oriented Architectures. The strength of our concept lies in its generic description of the security goals and the abstraction from technical details. In particular, the change in high-level policies can be immediately propagated to the managed system.

In the course of this paper a pattern system has been introduced, which is used to resolve concrete security protocols and mechanisms with regard to specific preconditions. The gathered information is to be mapped to an arbitrary framework or technical specification. We achieve our goal by formalizing the considered system and formalizing the description of the pattern.

A further strength of our solution is a simple integration and the extensibility of the software to include additional security modules. It is sufficient to implement a simple software component to map existing implementation patterns into a new policy language. As a proof of concept we implemented a software component to map abstract security patterns to the configuration of Axis 2 Rampart module.

Our current work is focused on the further development of the application to cover additional security goals and aspects like access control or monitoring. Furthermore, we intend to describe and to implement the existing security goals in different application domains. While we described our model-driven approach, we involve the user to set some framework specific information. In the next step we will address the process of automated identification of the environment parameters.

8. REFERENCES


