DEFINING VIEWPOINTS FOR SECURITY ARCHITECTURAL PATTERNS

David G. Rosado, Carlos Gutiérrez, Eduardo Fernández-Medina, Mario Piattini
ALARCOS Research Group. Information Systems and Technologies Department UCLM-Soluziona Research and Development Institute. Escuela Superior de Informática. University of Castilla-La Mancha.
Paseo de la Universidad, 4 – 13071 Ciudad Real, Spain

Keywords: Security Architectures, Security Patterns, ViewPoints

Abstract: For decades, the security community has undertaken detailed research into specific areas of security, while largely ignoring the design process. Software architecture has emerged as an important sub-discipline of software engineering, particularly in the realm of large system development. This paper describes how security architectural patterns lack of a comprehensive and complete well-structured documentation that conveys essential information of their logical structure, deployment-time, run-time behaviour, monitoring configuration, and so on. Thus we will propose a viewpoints model for describing security architectural patterns. We will investigate security architectural patterns from several IEEE 1471-2000 compliant viewpoints and develop an example that demonstrates how to describe a security architectural pattern with viewpoints. We will make use of well-known language notations such as UML to maximize comprehensibility.

1 INTRODUCTION

In a typical application development environment, architects and developers share similar experiences. They deploy business applications in a highly compressed time frame, making applications work, testing functionality at all levels, ensuring that they meet expected system performance or service levels, and wrapping applications with an attractive client presentation and user documentation. Ensuring the security of the application at all levels has usually been considered at the last phase of the development process (Steel, Nagappan et al. 2005).

For decades, the security community has undertaken detailed research into specific areas of security, while largely ignoring the design process. Security aspects cannot be "blindly" inserted into an IT-system, but the overall system development must take security aspects into account. The result of a well-engineered security system must be an architecture that ensures specific security aspects such as secrecy, integrity and availability (Artelsmair and Wagner 2003).

As we have seen over and over, the software architecture for a system plays a central role in system development as well as in the organization that produces it. Architecture serves as the blueprint for both the system and the project developing it. It defines the work assignments that must be carried out by design and implementation teams and it is the primary carrier of system qualities such as performance, modifiability, and security; none of which can be achieved without a unifying architectural vision. Architecture is an artifact for early analysis to make sure that the design approach will yield an acceptable system. In short, architecture is the conceptual glue that holds every phase of the project together for all of its many stakeholders (Bass, Clements et al. 2003). The architecture must be documented to communicate how it achieves those properties (Bachmann, Bass et al. 2000).

Recently, there has been a growing interest in identifying security patterns in software-intensive systems since they provide techniques for considering, detecting and solving security issues from the beginning of their development life-cycle (Yoder and Barcalow 1997; Schumacher and Roedig 2001; Cheng, Konrad et al. 2003; Schumacher, Fernandez et al. 2005). Security patterns work together to form a collection of coordinated security countermeasures thereby addressing host, network and application security.

This paper describes how security architectural patterns lack of a comprehensive and complete well-structured documentation that conveys essential information of its logical structure, run-time behaviour, deployment-time and monitoring configuration, constraints, elements, and so on. In consequence, we will propose an alternative way for describing security architectural patterns from viewpoints and views, and therefore we can add more information about the pattern in the template used for defining patterns.

The remainder of this paper is organized as follows. Section 2 will discuss the importance of software architectures and the two most important concepts associated with software architecture documentation: view and viewpoint; In Section 3, we will define security patterns and what security architectural patterns are; In section 4, we will describe the viewpoint template defined by the IEEE 1471-2000 standard; In section 5, an overview of the IEEE 1471-2000 compliant Security Subsystem Design viewpoint's template definition will be shown and we will discuss an example of security architectural pattern. Finally, we will put forward our conclusions and future work.

2 SOFTWARE ARCHITECTURE

Software architecture has emerged as an important sub-discipline of software engineering, particularly in the realm of large system development. There are many definitions of software architecture (Garlan and Anthony 2002; Bass, Clements et al. 2003), but what these definitions have in common is their emphasis on architecture as a description of a system, as a sum of smaller parts, and how those parts relate to and cooperate with each other to perform the work of the system.

The architecture must be documented to communicate how it achieves properties such as performance, reliability, security, or modifiability. Fundamentally, architecture documentation can serve three different functions (Bachmann, Bass et al. 2000): a) A means of education. Typically, this means introducing people to the system. b) A vehicle for communication among stakeholders. A stakeholder is someone who has a vested interest in the architecture. c) A basis for system analysis. To support analysis, documentation must provide the appropriate information for the particular activity being performed.

3 SECURITY PATTERNS

Security patterns are proposed as a means of bridging the gap between developers and security experts. Security patterns are intended to capture security expertise in the form of worked solutions to recurring problems. The benefits of using patterns are: they can be revisited and implemented at anytime to improve the design of a system; less experienced practitioners can benefit from the experience of those more fluent in security patterns; they provide a common language for discussion, testing and development; they can be easily searched, categorized and refactored; they provide reuseable, repeatable and documented security practices; they do not define coding styles, programming languages or vendors (Berry, Carnell et al. 2002).

An architectural pattern expresses a fundamental structural organization schema for software systems. It provides a set of predefined subsystems, specifies their responsibilities, and includes rules and guidelines for organizing the relationships between them (Buschmann, Meunier et al. 1996).

We define security architectural patterns at several levels of detail depending on the different potential consumers who see different characteristics, functionalities, connections and behavior of a same pattern. If we define security patterns from different perspectives, we are adding more relevant information to the template used for describing security patterns.

4 VIEWPOINTS APPROACH

We attempt to extend the template by adding new information from the stakeholders' viewpoint following as a reference the "4+1" view model (Kruchten 1995).

Obviously, since the 4+1 views preceded IEEE 1471, they do not meet the definition of views as specified in the standard. The 4+1 views describe a collection of representations that provide guidance for software architects. The viewpoints we discuss are within the spirit of the 4+1 views.

ANSI/IEEE 1471-2000 (IEEE 2000) provides guidance for choosing the best set of views to document, by bringing stakeholder interests to bear. It prescribes defining a set of viewpoints to satisfy the stakeholder community. For describing viewpoints and views, IEEE 1471 standard defines a set of elements or sections (template) that are showed in (IEEE 2006) and that we will see later.

4.1 ViewPoints Catalogue

We are defining a library of viewpoints of security that allow us to document security architectural patterns according to IEEE 1471-2000. By definition, these viewpoints are reusable for any software system, thus we can document security patterns, security architecture, software architecture, etc., based on our viewpoint's library.

A number of viewpoint catalogues already exist, but we have found that all of them do not consider aspects of security, they are only applied to the development of architectures for large information systems and they are not applied in the context of security. In response, we have developed a set of viewpoints for the security architect and the security engineers, that build up and extend the "4+1" set, identified by Philippe Kruchten (Kruchten 1995) and Nick Rozanski and Woods (Rozanski and Woods 2005). Our catalogue contains seven core security viewpoints: Logical, Process, Development, Physical, Deployment, Operational and Misuse Cases viewpoints as we can see in Figure 1.

The security logical viewpoint describes the objects or object models within the security architecture that support security behavioral requirements. The security process viewpoint describes the security architecture as a logical network of secure communicating processes. This viewpoint assigns each method of the object model to a thread of execution and captures concurrency and synchronization aspects of the security design. The security physical viewpoint maps software onto hardware and network elements and reflects the distributed aspect of security architecture. The security development viewpoint focuses on the static organization of the software in the security development environment and deals with issues of configuration management, security development assignments, security responsibilities, and countermeasures. The security deployment viewpoint describes the security environment which the system will be deployed into, including the fact of capturing the dependencies the system has on its runtime environment. The aim of the Security Operational viewpoint is to identify security system-wide strategies for addressing the operational concerns of the system's stakeholders and to identify solutions that address them.

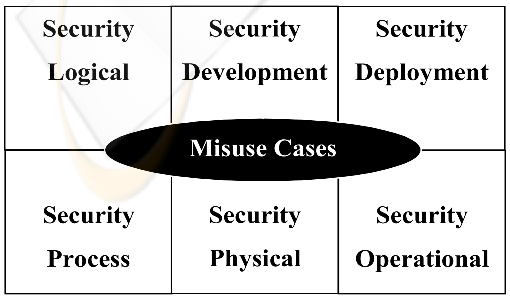

Figure 1: Our approach of Security Viewpoints.

Moreover, we are defining a new viewpoint's template extending the aforementioned template of IEEE 1471-2000 and we have added new sections in the context of security that are described as follows:

- Security properties to be addressed by the security policy on the basis of the security viewpoint's elements. We consider that the complete security policy of a security pattern is the aggregation of the security policies defined for each security viewpoint.
- Security metrics to be taken into account.
- Security procedures to be taken into consideration from this viewpoint; for instance, from the physical viewpoint, procedures to restore the physical node in which the security services defined by the pattern are running, or from the logical viewpoint, how to carry out the off-line exchange of key material between the involved parties.
- Best practices: for example, from the developer's viewpoint, techniques for secure programming, or from the physical viewpoint, topologies of secure networks.

5 SECURITY DESIGN SUBSYSTEM VIEWPOINT

Each aforementioned viewpoint can be divided into different viewpoints satisfying the interest of a particular stakeholder. A series of viewpoints is then used to elaborate the details of the general viewpoint. Selecting the security design subsystem viewpoint and considering the template IEEE 1471, we have defined this viewpoint as presented in Figure 2.

❶ **Abstract**. This viewpoint shows security module decomposition and the use between systems of software system. Each security module interprets itself as a subsystem to develop; therefore it is an entity in construction time, and can communicate with others security subsystems for completing its functionality. The decomposition continues until that each module or subsystem of security is allocated to a unique responsible of development or team.

❷ **Stakeholders and their concerns addressed.** Secure applications will be developed by (at least) three different roles: i) Application software developers that focus on the business logic; ii) Security providers that focus on the design and implementation of reusable frameworks of security logic; iii) Security engineers that implement the security policy for a particular application and focus on how the system is implemented from the perspective of security, and how security affects the system properties. • Project managers, who must define work assignments, form teams, and formulate project plans and budgets and schedules; Maintainers, who are tasked with modifying the software elements; Testers and integrators who use the modules as their unit of work.
❸ **Elements, Relations, Properties, and Constraints.** • Security modules are units of implementation, and their decomposition in shorter modules, just as use dependencies exist between them. • Relations between security modules can have the semantic associated 'is-part-of' or 'utilize'. • The last level of subsystems called security design subsystems, defined in the views according to this viewpoint: i) Must be set of products of work of design assigned to different develop teams; ii) Security subsystems will correlate with the construction directories that will be developed, tested and handed over respective teams of development; iii) Following modality origins, the security subsystems must exhibit high cohesion and low coupling; iv) These subsystems will be the lower level entities for which the software architects team will need to define the interface.
❹ **Language(s) to Model/Represent Conforming Views.** • The representation language used will be the UML and extensions for security aspects such as UMLSec (Jürjens 2001; Jürjens 2002) and SecureUML (Lodderstedt, Basin et al. 2002). • Each module or subsystem of security will represent itself as a stereotyped UML packet with the reserved word <<subsystem>>, the use relations will show with <<uses>> and decomposition relations with nesting of UML packets. • The interfaces that implement each system are modeled as UML interfaces and the name of the service to be included in each interface corresponds with the names of the use cases defined at the abstraction level of "Goal Summarize" (CockBurn 2000) for each subsystem. • The design subsystem included into views according to this viewpoint will declare a realization of one or more interfaces whose methods correspond with use cases at the abstraction level "User Goal" specified in the model of use cases of this design subsystem.
❺ **Applicable Evaluation/Analysis Techniques and Consistency/Completeness Criteria**. Revision checking of the fact that the different development groups of form understand the context of the subsystem that they are going to develop (where system comes from) so as the interfaces with other design subsystems. Some analysis and evaluation methods are described by Ronald Wassermann (Cheng, Konrad et al. 2003) and Jan Jürgens (Deubler, Grünbauer et al. 2004).
❻ **Viewpoint Source**. Viewpoint of Design Subsystem (Garlan and Anthony 2002)

Figure 2: Security Design Subsystem Viewpoint.

6 EXAMPLE: 'QoP' PATTERN

Due to space constraints and because actually we are working and researching in this issue, we will do a brief description of the 'QoP' pattern from security logical view that includes a views' packet with the information of decomposition in security design subsystem, attempting describe the main object from the security design subsystem viewpoint.

From this description of viewpoint, we will attempt to describe, following the viewpoint of design subsystem, the 'QoP' security pattern that offers Quality of Protection security service which address message confidentiality, message integrity and message authentication.

This view allows the security software architects to communicate security development team boundaries, communicate and negotiate interfaces between security development teams, and communicate with security project management.

Each subsystem must implement an interface that is used by the rest of elements that need the service that the owner subsystem of the interface offers. This subsystem implements the interface 'QoPSecurityService' with the methods protect and verifyProtection, as we can see in Figure 3.

These two methods are used by the elements for applying the service to outbound/inbound messages according to policy established for this service. This subsystem has relations with others subsystems or interfaces, as it can be the relation with the subsystem Security Token Manager, through its interface *SecurityTokenManager* that manages security tokens and they can be implemented using WS-Trust, XKMS with PKI infrastructure, etc., established in its security policy; it has relation with the subsystem Message Confidentiality Manager, through its interface *ConfidentialityManager* that cipher or decipher the message offering message confidentiality service using XML Encryption; and it has relation with the subsystem Message Integrity Manager, through its interface Integrity*Manager* that check and protect the message offering message integrity service using XML Digital Signature. Also it can have relation with the subsystem Security Policy Service, this is optional, where it would manage all policies associated to the services offered by the system. We have said that this is optional because policies can be managed and implemented into of the own subsystem (i.e. 'QoP' subsystem) without to be relation with this subsystem.

Other possible relation, non obligatory, is the relation between 'QoP' Security Service subsystem and an Alarm subsystem, using a common protocol of alarms (CAP, Common Alerting Protocol) establishing an alarms system in the application, communicating elements with others, indicating an event or an alarm generated and sent to others subsystems.

7 CONCLUSIONS

It is important to document a software architecture because first of all it serves to introduce people to the system; secondly, it serves as a vehicle for communication among stakeholders, and finally it is use as a basis for system analysis. Moreover, a documented architecture is crucial for understanding its main characteristics, its functionality, its components and connections, its behaviour, and so on. It will be important to describe and define the main characteristics of architectural patterns for stakeholders to be able to use and analyze the pattern at the time of integrating it into either the design of the application, or the design of the whole architecture.

In this paper, we have described an architectural pattern from viewpoints attempting to provide a wider vision of its main characteristics, its design, connections, elements, interfaces, implementation, classes and behavior, putting the description of the pattern conforming to the template as future work.

Our intention is not only to define security architectural patterns by means of a views template and a viewpoint template but also to recommend ANSI/IEEE 1471-2000 (IEEE 2000), that provides guidance for choosing the best set of views to document. We have defined a viewpoints' catalogue and we have added and we are adding new elements or sections to the viewpoint template of IEEE 1471-2000 standard. Our research concentrates in defining

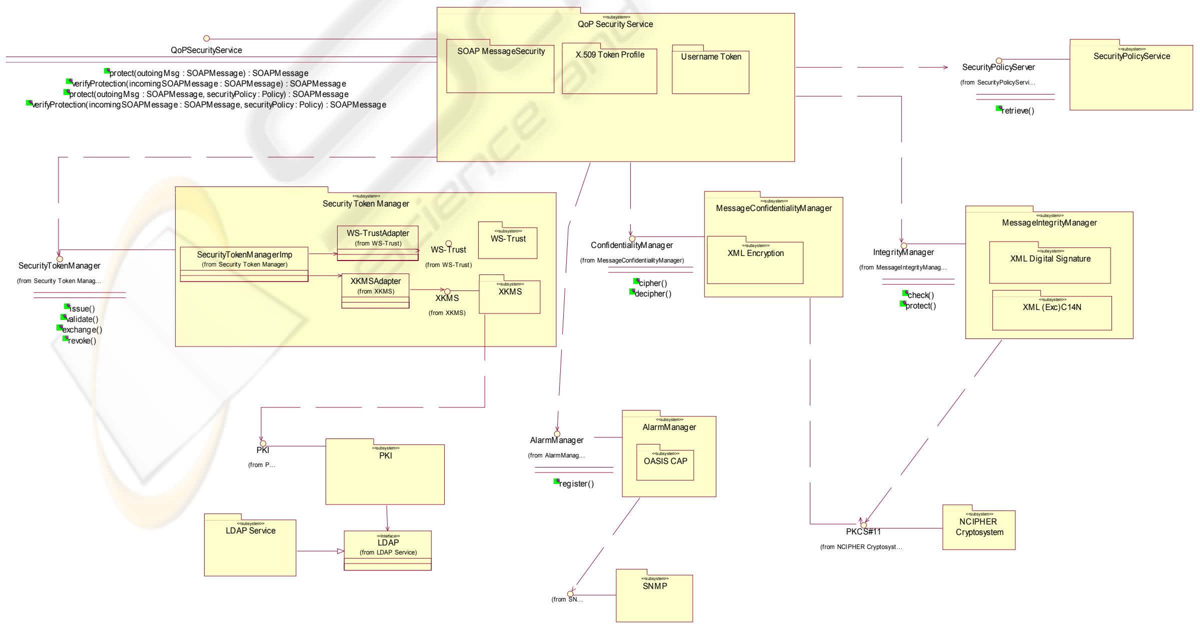

Figure 3: Relations between subsystems and interfaces of the 'QoP' Pattern.

a library of viewpoints adhered to IEEE 1471-2000 whose instances are the views that we can define following the documentation IEEE 1471-2000 (IEEE 2006). In this way, we could have a library of viewpoints to document security architectural patterns.

ACKNOWLEDGEMENTS

This research is part of the following projects: DIMENSIONS (PBC-05-012-2) and MISTICO (PBC06-0082) financed by FEDER and by the "Consejería de Ciencia y Tecnología de la Junta de Comunidades de Castilla-La Mancha" (Spain), and CALIPO (TIC2003-07804-C05-03) granted by the "Dirección General de Investigación del Ministerio de Ciencia y Tecnología" (Spain).

REFERENCES

Artelsmair, C. and Wagner, R. (2003). Towards a Security Engineering Process. The 7th World Multiconference on Systemics, Cybernetics and Informatics, Orlando, Florida, USA.

Bachmann, F., Bass, L., et al. (2000). Software Architecture Documentation in Practice: Documenting Architectural Layers**: Pgs.** 46.March 2000

Bass, L., Clements, P., et al., Eds. (2003). Software Architecture in Practice, Addison-Wesley.

Berry, C. A., Carnell, J., et al. (2002). Chapter 5: Patterns Applied to Manage Security. J2EE Design Patterns Applied.

Buschmann, F., Meunier, R., et al. (1996). Pattern-Oriented Software Architecture: A System of Patterns, John Wiley & Sons.

CockBurn, A. (2000). Writing Effective Use Cases, Addison-Wesley Professional.

Cheng, B. H. C., Konrad, S., et al. (2003). Using Security Patterns to Model and Analyze Security Requirements. High Assurance Systems Workshop (RHAS 03) as part of the IEEE Joint International Conference on Requirements Engineering (RE 03), Monterey Bay, CA, USA.

Deubler, M., Grünbauer, J., et al. (2004). Sound Development of Secure Service-based Systems. Second International Conference on Service Oriented Computing (ICSOC), New York City, USA, ACM Press.

Garlan, J. and Anthony, R. (2002). Large-Scale Software Architecture, John Wiley & Sons.

IEEE (2000). Recommended Practice for Architectural Description of Software-Intensive Systems (IEEE Std 1471-2000). New York, NY, Institute of Electrical and Electronics Engineers**: Pgs.** 29.01-May-2000

IEEE. (2006, last saved: March 21, 2006). "Software Architecture Document (SAD)." from www.sei.cmu.edu/architecture/SAD_template2.dot.

Jürjens, J. (2001). Towards Secure Systems Development with UMLsec. International Conference of Fundamental Approaches to Software Engineering (FASE/ETAPS), Genoa, Italy, Springer-Verlag.

Jürjens, J. (2002). UMLsec: Extending UML for Secure Systems Development. 5th International Conference on the Unified Modeling Language (UML), 2002, Dresden, Germany, Springer.

Kruchten, P. (1995). "Architectural Blueprints - The "4+1" View Model of Software Architecture." IEEE Software **12**(6): 42-50.

Lodderstedt, T., Basin, D., et al. (2002). SecureUML: A UML-Based Modeling Language for Model-Driven Security. 5th International Conference on the Unified Modeling Language (UML), 2002, Dresden, Germany, Springer.

Rozanski, N. and Woods, E. (2005). Software Systems Architecture: Working With Stakeholders Using Viewpoints and Perspectives, Addison Wesley Professional.

Schumacher, M., Fernandez, E. B., et al. (2005). Security Patterns, John Wiley & Sons.

Schumacher, M. and Roedig, U. (2001). Security Engineering with Patterns. 8th Conference on Patterns Lnaguages of Programs, PLoP 2001, Monticello, Illinois, USA.

Steel, C., Nagappan, R., et al. (2005). Core Security Patterns, Prentice Hall PTR.

Yoder, J. and Barcalow, J. (1997). Architectural Patterns for Enabling Application Security. 4th Conference on Patterns Language of Programming, PLop 1997, Monticello, Illinois, USA.