SERENITY Pattern-based Software Development Life-Cycle

Francisco Sanchez-Cid, Antonio Maña
Computer Science Department
University of Malaga, Spain
{cid, amg}@lcc.uma.es

Abstract

Most of current methodologies for applying security patterns in software development process focus on early stages of the software lifecycle. Although sound and helpful, these approaches must be complemented with the necessary mechanisms to bridge the gap between the abstract solution described in the pattern and the implementations provided in the application. This paper presents our S&D Patterns-driven life-cycle for effectively incorporating security into applications by: (i) helping S&D Experts to describe solutions using a precise and standard description language, (ii) assisting the SW engineer to integrate the pattern solution into applications and (iii) allowing running applications to react to the changing context by handling those solutions semi-automatically.

1. Introduction

It is now widely accepted that end to end security should be adopted and accomplished as part of the early application design and development process. If addressed at the end of the deployment phase or even considered just before the system testing in a pre-production environment, options for reactive or post-mortem security fixes are very limited.

Moreover, as nowadays threats quickly evolve and make existing solutions obsolete, what was considered a secure application design will probably be insecure in short term. Thus, the challenge faced by our work is to create a quality application security design to ensure that (i) all aspects of application security are considered during the early design stages; (ii) the security and dependability (S&D from now on) requirements are correctly translated from the design to the implementation; and that (iii) once the system has been deployed, it adopts a reactive approach, ensuring service continuity and recovery in case of a security breach or malicious attack.

SERENITY [1] provides a framework supporting the automated integration, configuration, monitoring and adaptation of S&D mechanisms for AmI ecosystems. In what follows, SERENITY approach is presented, by introducing (in section 2) the basic instruments for describing the methodology whose phases are fully depicted in section 3. Finally, section 4 focuses on previous work on security methodologies based on security patterns. We close with conclusion.

2. SERENITY Elements

Before proceeding with SERENITY Methodology phases, we present the roles that different actors in software development process play in SERENITY; then, we introduce the languages used to describe, select and apply solutions; and finally we outline the tools for assisting engineers in each phase.

2.1. The Roles

Serenity distinguishes among several roles depending on the activity performed in the different phases of the SERENITY Methodology, namely:

- **S&D Experts**: They study, analyse and then describe the functionality of S&D Solutions. In turn, S&D Solutions refer to any security or dependability mechanism (e.g. Apache Rampart to secure web services) that realizes an S&D Requirement (e.g. authenticity, integrity…) This role is played in fact by a multidisciplinary team devoted to research and development, probably belonging to some standardization body for S&D technologies. They design S&D Solutions as blocks or services and use the language of S&D Patterns (presented in next section) to represent them. They should also have expertise in verification, validation and certification of S&D Solutions as well as in legal issues related to information technologies.

- **Component Developers**: Once a solution is described in terms of the Language of S&D Patterns, engineers are able to implement it following the strategies defined in the language. As all security mechanisms studied by S&D Experts are described as abstract solutions, both independent of the target
platform and implementation language, component developers can create a variety of implementations of the same solution for different platforms and programming languages. The resulting product is an Executable Component which, in turn, is described using the language of S&D Patterns to facilitate its dynamic application at deployment time.

− Application Developer: this is a wide role that covers (i) the analysts that define the business mandated security requirements of the system; (ii) the software architects that through a well-defined risk analysis propose the candidate architecture and identify candidate patterns that cover the security requirements; and (iii) security-aware developers that implement the design by integrating the patterns identified in (ii). To achieve their goals, Application Developers make use of the three languages proposed in SERENITY. More specifically, the Language of S&D Properties helps them in the process of eliciting the requirements. The Language of S&D Patterns permits them to navigate through the universe of solutions in the search for a candidate. Finally, with End User’s Language they effectively apply the patterns in the application code by incorporating calls to Executable Components in a semi-automatic fashion.

− S&D Engineer: one of the main aspects in S&D Patterns’ specification is the elicitation of the S&D Properties they provide. The definition of these properties along with their formal description is task for S&D Engineers. By means of the Language of S&D Properties, they are able to study the semantic relations established among properties and make inferences based on those relations. This mechanism allows, for instance, checking whether Enterprise A’s definition for integrity is compatible with B’s integrity and thus, whether A’s solution for this requirement is valid for B. In some cases, this role can be assumed by an S&D Expert.

2.2. The Languages

In Figure 2, intersections between vertical and horizontal boxes show the use of the different languages for each of the roles presented above, while the ellipses in the picture represent the tools available for each actor to get advantage of the languages. A precise definition of the three languages for aiding the actors in the different phases of the SERENITY Methodology is given below:

− Language of S&D Patterns: this language is the core mechanism for the representation of S&D Solutions in the form of semantic descriptions. Three are the artefacts that compose the language: S&D Classes, S&D Patterns, and S&D Implementations [2]. S&D Patterns are the means for describing solutions. They include all the information necessary for the selection, instantiation and dynamic application of the solution represented in the S&D Pattern [3]. If two patterns provide the same S&D Property (e.g. confidentiality, non-repudiation…) and comply with a common interface, then we can group them under the same S&D Class. S&D Classes represent abstractions of a set of S&D Patterns characterized for providing the same S&D Properties and complying with a common interface. Finally, for the connection between the abstract solution described as a pattern and the actual Executable Component, S&D Implementations are introduced. They describe the specific context conditions to meet before deploying an Executable Component and provide the link for their deployment.

− End User’s Language: this language presents two sides, each one devoted to a different actor. As described before, the Language of S&D Patterns is a guide on how to correctly implement an S&D Solution. Then Component Developers use the End User’s Language as a mechanism to make the resulting Executable Component compliant with a well-defined interface. This assures that calls to the pattern in the application code will remain the same regardless of the actual Executable Component that realizes the selected pattern. On the other hand, and based on the S&D Requirements, Application Developers look for the most suitable components fulfilling the requirements. Once found, the process of selecting and incorporating the solution into the source code is achieved thanks to the End User’s Language, which provides the necessary access point to Executable Components.

− Language of Properties: this language supports (i) the definition of the formal semantics of different S&D Properties; (ii) the analysis of these definitions in order to find relations between S&D Properties; and (iii) the use of the inferred relations for the flexible selection of S&D Solutions suitable of...
fulfilling the requirements of the system. In order to be able to create definitions for S&D properties and to compare and reason about them, S&D Experts need a precise language that has no room for ambiguity. In this sense, we define the Formal S&D Properties Language (FPL) to provide formal semantics to properties’ definitions and on the other hand, the Operational S&D Properties Language (OPL) for defining semantic relations among them. When looking for a solution, the returned candidates will be those providing the requested requirement along with those providing requirements equivalent to the original one.

2.3. The Tools

Describing S&D Solutions in terms of the Language of S&D Patterns implies the creation of XML documents with a well-defined structure. Thus a Pattern Management Tool is needed for guiding S&D Experts in the creation of such documents, and for making it easy to store and navigate through the different artefacts: from Classes to Patterns and from there to Implementations. Similarly, the Properties Management Tool is provided for S&D Engineers to formally describe S&D Properties, allowing the creation of a repository of properties and the study and elicitation of relations among them.

To end with, the End User’s Language is made available through extensions in well-known CASE tools. For instance, a pre-installed plug-in in Eclipse can allow Application Developers to select S&D requirements from a predefined list, then presenting a list of Patterns fulfilling the mandated requirements.

3. Secure Software Life-Cycle through S&D Patterns

All phases in our development life-cycle (depicted in Fig. 2) are highly influenced by the concept of S&D Patterns. This is the reason why the process of security engineering using SERENITY Methodology is depicted following the life-cycle of an S&D Pattern. The methodology covers from the selection of S&D Solutions at development time to its dynamic adaptation to runtime context.

At development time, two main actors are in place: the security expert, creating S&D Solutions and capturing this knowledge in form of S&D Patterns, and the software developer that takes advantage of this knowledge, selecting and applying patterns during the development. On the other hand, runtime presents a scenario where solutions are in place and applications are already developed and running. In this scenario, it is necessary to monitor the correctness of the applied solutions and react accordingly in case that some fault or security alert is fired. This reaction is possible thanks to the monitoring rules that patterns incorporate and the monitoring mechanisms that build on top of the Serenity Architecture, which is not detailed here in depth for the sake of space.

Now let us focus on each phase of the SERENITY development cycle.

3.1. Creating S&D Patterns

S&D Patterns are made available to S&D Engineers through the Development-Time S&D Library. This library is distributed in several repositories from different providers and stores a complete set of S&D Patterns, S&D Classes and S&D Implementations. A repository is typically managed by an S&D Expert.

S&D Experts start analysing an S&D Solution that can be either a widespread and well-known solution (e.g. a secure pipe using SSL channels) or a brand new one, not yet standardized. The analysis of the experts must conclude with a rich specification of the solution that will be captured using the Language of S&D Patterns. Mandatory elements for this specification are: a well defined interface for accessing the solution features; the preconditions for applying the solution (i.e. a description of the circumstances for the solution to be applicable at runtime); the properties provided once the solution is in place (e.g. confidentiality in HTTP transactions); the static tests performed to provide evidence of robustness under certain threats or attacks; and last but not least, the monitoring rules to capture all the security-specific alerts and issues that may affect the applicability of the solution.

Once a pattern is created, it is time to analyse the possible interactions with other patterns and to check whether it can be grouped along with some other patterns providing the same property. As stated before, S&D Classes provide a high level interface to access its functionality and is used to group S&D Patterns that provide the same S&D Properties. When assigning an S&D Pattern to a class, S&D Experts first create a class adaptor mapping from the class’ interface to the pattern’s interface, and then create a link from the pattern to the class. In no class exists providing the pattern’s properties S&D Experts create a new class and then proceed as before. It is important to remark that during a class’ lifetime, a number of patterns can be linked to this class as long as they provide the same S&D Properties and create the corresponding class adaptor. That way, we assure that given a selected S&D Class, all S&D Patterns belonging to this class will be selectable at runtime.
At this point, security and dependability engineers are in position to take solution descriptions given as S&D Patterns and create Executable Components based on these specifications. Given that patterns describe abstract solutions and are independent of the target platform and implementation, Executable Components can be created for different platforms and in different programming languages. The only restriction is for each executable component to come with a wrapper mapping from the local interface to its counterpart defined at pattern level.

Security-aware developers implement the solution according to the pattern’s interface making use of the End User’s Language. Using the End User’s Language ensures that all implementations comply with the standard format and parameters, providing a common access point to different Executable Components. Once a component is created, it is described by means of an S&D Implementation, which encloses information on how to deploy it, and serves as link between the actual component and the source pattern. The Runtime S&D Library stores the components, where they can be requested from the applications.

Eventually, the solution has been described as an S&D Pattern, the Pattern catalogued under an S&D Class, implemented as an Executable Component and the latter described as an S&D Implementation. The next logic step is to populate the Development Time S&D Library with the artefacts created.

3.2. Creating Applications

In order to build secure applications, S&D requirements of the system are analysed and identified. This includes the elicitation of requirements based on industry regulations, corporate policies, and other business-specific needs. Having the list of requirements specified by the analyst, security designers identify the set of S&D Patterns that covers them. Following the example in previous section, the designer will use the SERENITY plug-in available in his favourite CASE tool, and will navigate through the patterns that match the requirements. The plug-in presents a front end for accessing the Development Time S&D Library, to look for and, eventually, select a solution.

Then, security-aware developers must include in the code the actual requests for the solution (i.e. the pattern) that has been selected. At this level, the request is made for an S&D Pattern by incorporating calls to functions defined in the pattern’s interface. In order for the developer to easily specify these calls, the End User’s Language provides an access point to the interface of the selected pattern. At deployment time, the previous request will be translated into a request for the actual Executable Component.

It is at deployment time when the knowledge of the actual context conditions and applicability restrictions are known. Thus, it is natural to wait until the deployment of the application for selecting the actual component that will realize the S&D Pattern selected at development time. All Executable Components that implement a pattern comply with the same pattern’s interface. Consequently, they are interchangeable as long as their preconditions hold for applying them.

3.3. Running System

Once the application has been created and deployed in the target system, a close interaction between the application and the SERENITY Runtime Architecture
starts. The very moment that the system is deployed, applications send a request for the required Executable Components. Monitoring mechanisms have been already activated to recover relevant information about the context conditions, characteristics of the target system, and so forth. This information is contrasted with the preconditions coming from the S&D Patterns and S&D Implementations. When an artefact is found whose preconditions hold, its Executable Component is recovered from the Runtime S&D library, deployed, and its handler returned to the application. This way, the Executable Components are effectively linked to the application.

Finally, it starts the process of monitoring solutions deployed. The monitoring rules from the corresponding S&D Patterns are extracted, activated and sent to a monitoring service that periodically checks the captured events. If some event is found that breaks any of the rules, the Executable Component is immediately stopped and the application is informed of the contingency. Then, the application can decide to look for another applicable component sending a request for a new Executable Component.

4. Related work

The integration of security aspects in the software development life-cycle has attracted the attention of different researchers. However, they usually consider only specific aspects (access control, etc.), phases (requirements elicitation, etc.) or activities (modeling, programming, etc.) in the development life-cycle. Examples of these include proposals for modelling systems, such as UMLsec [4], which is proposed as an extension of the Unified Modelling Language for modelling security properties of computer systems. UMLsec focuses on the modelling activity and only addresses a few specific security requirements. Tropos methodology has formulated similar proposals [5].

Only a few proposals provide a more extensive support for the development life-cycle. Model Driven Security, is a model driven architecture approach to security engineering [6]. More recent proposals, also based on the MDA paradigm, have been introduced for specific contexts such as Service Oriented Architectures (SOA). In particular, recent work based on the concept of security patterns is presented in [7] describing an MDA-based process for SOA applications. Also based on MDA, the proposal described in [8] focuses on Business Processes.

The main difference of our proposal with those described above is the consideration of the complete development life-cycle. We consider the development of the security solutions and their characterization as patterns with the objective of facilitating their reusability, interoperability and dynamic context-based selection and substitution. Furthermore, our approach supports application developers in selecting and integrating the most appropriate solutions, thus avoiding implementation errors caused by the lack of security expertise of the application developers.

5. Conclusions

The use of the language of S&D Patterns and the adoption of the Serenity Methodology helps S&D Experts to describe solutions in a systematic and standardized way, going from Classes to Patterns to S&D Implementations. Next steps include the creation of an S&D Library with a wide catalogue of solutions. Once grouped, classified, and related to other existing S&D solutions, it is easy to navigate through them. In addition, S&D Patterns allow Component Developers to count on the skeleton and guidelines for creating Executable Components that faithfully follow the original S&D Solution. The existence of a common interface also promotes interoperability and reusability.

We have implemented a prototype for S&D Experts to create and edit S&D Artefacts, also permitting the management of these artefacts in S&D Libraries. We are currently testing the integration of a first set of S&D Artefacts with the Serenity Runtime Architecture.

6. References