Patterns for Automated Management of Security and Dependability Solutions*

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

Abstract

Current processes for providing security and dependability (S&D) in computing systems require a detailed a priori knowledge about the target systems and their environments. However, in many emergent computing scenarios like ubiquitous computing or ambient intelligence, it is not possible to foresee all possible situations that may arise at runtime so the necessary knowledge is not available at development time. In this paper we present the concept of S&D Pattern and the artefacts that we use to implement it, as the basis for the automated provision of S&D Solutions to running applications in highly dynamic and heterogeneous environments.

1. Introduction

Security and dependability (S&D) are essential aspects of computing and communication systems. The wide spreading of an ever-increasing number of heterogeneous computers and communication channels has resulted in the popularization of many (new and not-so-new) distributed computing paradigms. At the same time, the ubiquity of communication systems and information has fostered the development of many interesting distributed applications in which we rely more and more in our daily lives.

The current processes for developing secure and dependable systems require a detailed a priori knowledge about the systems and their environments. However, in the scenarios depicted above, this knowledge is not available at development time because it is impossible to foresee all possible situations that may arise at runtime. Therefore, we need to overcome this difficulty by delaying the provision of S&D to runtime, when we have enough information to make a sound decision. This approach requires the introduction of automated mechanisms capable of selecting the most appropriate solution based on precise information about the application context and the available solutions.

This paper presents the SERENITY approach to the concept of S&D Pattern and the artefacts that we use to implement it, as the basis for the automated provision of S&D Solutions to running applications.

The rest of the paper is organized as follows. Section 2 presents an example scenario that will be used throughout the paper. The goal of this section is to introduce the main concepts proposed in our work (the artefacts for representing security solutions) to the reader, along with the mechanisms behind the automatic selection and adaptation of these artefacts at runtime. Section 3 presents the SERENITY model for the automated management of S&D Solutions and provides precise descriptions of the three artefacts that we propose for the representation of S&D Solutions. Section 4 describes some relevant related work and finally Section 5 presents conclusions.

2. L.A. Confidential

It is leisure time at “Las Acacias” College and Alice and Bob enjoy a Race Game using their wireless ACME game-consoles. Charlie asks them to join the race using his brand new *BOXX630* SERENITY-enabled game-console. Alice and Bob are willing to accept their friend to join the game, but the ACME consoles require confidentiality for wireless connections to other devices. Basically, this is a pre-configured setting for preventing eavesdroppers from obtaining information about the parties that are interacting and the services they use.

Charlie’s console identifies the requirement (a confidential channel) and looks for the best solution. At design time, the developers of the game identified the need to securely connect players, but because at that stage they could not foresee the possible types of counterparts and the different circumstances under

* Work partially supported by the E.U. through SERENITY project (IST-027587) and by Junta de Castilla La Mancha through the MISTICO-MECHANICS project (PBC06-0082)

which the communication would take place, they decided not to restrict the range of possible solutions to use. One of the SERENITY artefacts called *S&D Class* represents security and dependability services and is especially designed to support system developers in these situations. In particular S&D Classes allow developers to delay the decision about the most appropriate solution to runtime, when the information required to select a specific solution (the context, type and capabilities of the other party, etc.) is available. Thus, they selected and used the S&D Class named "TransmisionConfidentiality.iso.org", which represents confidentiality services and includes a predefined high-level interface. In this way, the game developers were able to use the confidentiality services without knowing which solution will be used to provide them at runtime.

Going back to our scenario, Charlie's console must now select one specific solution to provide the confidentiality services to the game application. At this point the console uses the second of the SERENITY artefacts called *S&D Pattern*, used to represent abstract solutions. The main purpose of this artefact is to guarantee the interoperability of different solutions. A number of different *S&D Patterns* belong to the selected S&D Class. After analysing them, only two are found to be adequate given the current context: Charlie's using an open wireless network susceptible to possible eavesdropping as well as passive and active attacks. The suitable patterns are: "SSL 3.0 Channel" and "TLS Channel". At this point Charlie's device negotiates with the other parties and eventually, the SSL option is selected as the most appropriate.

Once the abstract solution has been selected, which ensures the interoperability between the different systems, Charlie's console needs to find an implementation (i.e. an instance) of the "SSL 3.0 Channel" S&D Pattern. The third artefact provided by SERENITY comes into play. This artefact is the *S&D Implementation*, used to represent working solutions.

Given Charlie's console context (underlying O.S., running software, other active S&D Solutions, user preferences, etc.), only three *S&D Implementations* are available for that S&D Pattern and that context: *mod_ssl* module from Apache 2.0, Cisco OpenSSL, and Java SSL using JSSE. Java implementation of SSL is selected and activated to provide confidentiality for the game-connection. Charlie is informed of the successful establishment of the confidential connection and he finally joins the game.

2.1. Analysis of the scenario

It goes without saying that our main characters are all but security experts. Consequently, it is important to remark that the provision of a solution for a concrete context should be as transparent as possible for the user. That is, S&D Patterns must be designed for automated processing, so that Charlie's awareness of technical details should be reduced to the minimum.

Each context entails different threats to guard from. Consequently, the S&D Pattern must include information on the attack models considered when the pattern was created. In addition, the scenario reveals another important issue regarding the application context. Bob is trying to connect through a non-trusted network, with a presumably low powered device. As the applicability of S&D Patterns and implementations depends on the context, it is necessary for the patterns to include such applicability conditions.

The S&D Solutions offered by Charlie's console are represented using a three level hierarchy. Figure 1 represents an instantiation of such hierarchy for the artefacts used in the L.A. scenario (note that only the SSL branch is fully expanded).

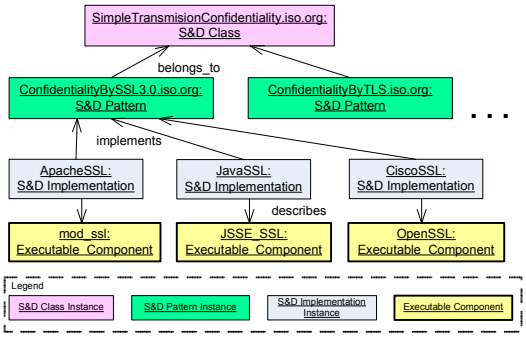

Figure 1. S&D Artefacts' hierarchy

In the figure, both *ConfidentialityBySSL3.0* and *ConfindentialityByTLS* patterns belongs to the same class, *SimpleTransmisionConfidentiality*. In addition, several implementations are available for SSL pattern, namely: *ApacheSSL*, *JavaSSL* and *CiscoSSL*. Finally, each S&D Implementation points to the real Executable Component that realizes the functionality claimed by the pattern. This component is the one to install and configure for the target device. The whole group hierarchy integrates what we call S&D Library.

To end with, the scenario points out the fact that in order to provide and deploy solutions at runtime, some work is necessary at development time. Following our approach, the analysis of the S&D Requirements of a device or application is done during the development of the system but the realization of some of these requirements may be delayed by the developer, until the system is running. However, before the system is running it needs to be populated with all the necessary artefacts: S&D Classes, S&D Patterns and S&D Implementations. In our example, Charlie's game made

use of the *TransmissionConfidentiality* class, but no pattern or implementation was selected at that stage. This is one of the main pillars of our proposal: As all the S&D Patterns belonging to a class provide the class' interface, all of them are candidates at runtime. Extending this concept to S&D Implementations, we achieve the flexibility necessary to adapt the solutions to ever-changing contexts.

3. SERENITY Model

3.1. Representing S&D Solutions

Our main objective is the development of artefacts to represent S&D Solutions for automated processing. Note that for this purpose we do not need to describe the functioning of the solution but its semantics (i.e. properties provided, limitations, etc.). This is an essential difference between our S&D Patterns and the widespread concept of security pattern. These semantic descriptions allow solutions to be automatically selected, adapted, used and monitored at runtime. However, as has been already shown, our approach adopts an integral methodology covering the complete system lifecycle also covering development aspects. Therefore, an additional goal for our artefacts is to support system developers in the development process. With these two purposes in mind, we have developed the following artefacts to capture the different aspects of the S&D Solutions that are necessary at different stages of the system lifecycle.

3.1.1 S&D Patterns

To start with, we define S&D Solutions as well-defined mechanisms (i.e. security protocols, encryption algorithms, etc.) that provide one or more S&D Properties (i.e. confidentiality, availability, etc.). Hence, *S&D Patterns* are detailed descriptions of abstract S&D Solutions that contain all the information necessary for the selection, instantiation and adaptation, and dynamic application of the solution represented in the S&D Pattern. One important aspect of the solutions represented as S&D Patterns is that they can contain a description of the results of any static analysis performed on them. Such descriptions provide a precise foundation for the informed use of the solution and enhance the trust in the model. Despite of that, the limitations of the current static analysis tools introduce the need to support the dynamic validation of the behaviour of the described solutions by means of monitoring mechanisms.

S&D Patterns represent not only simple solutions, but also complex ones. In fact, a special type of S&D Patterns, called Integration Scheme, is used to represent solutions that are built by combining other S&D Patterns. For the sake of space, no extended explanation is given here, but readers can find a more detailed description of this artefact in [1].

3.1.2. S&D Classes

S&D Classes represent abstractions of a set of S&D Solutions characterized for providing the same S&D Properties and having compatible interfaces. We could describe this artefact as an extension of the "interface" concept, with some semantic information, in a similar way as proposed in [2]. This artefact is mainly used at development time by system developers. The main purpose of introducing this artefact is to facilitate the dynamic substitution of the S&D Solutions at runtime while facilitating the development process.

Given that interoperability is a key issue at this level, with this approach it is possible for developers to create an application bound to a specific S&D Class given that this artefact only defines the high-level interface. At runtime all S&D Patterns (and their respective S&D Implementations) belonging to this S&D Class will be selectable. S&D Patterns that belong to an S&D Class can have different interfaces, but they must describe how these specific interfaces map into the S&D Class interface. Figure 2 shows how this correspondence is captured in a component of the S&D Pattern called "Interface Adaptor". In the representation, the Interface Adaptor specifies how to map the *SendConfidential()* function (at Class level) to the sequence *{GetKey(); Encrypt();* and *Send()}* (at Pattern level).

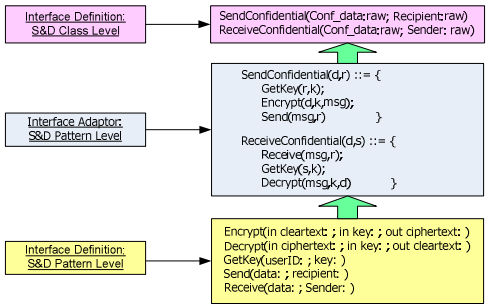

Figure 2. Interface Mapping process

3.1.3. S&D Implementations

S&D Implementations represent the components that realize the S&D Solutions. All S&D Implementations of an S&D Pattern must conform directly to the interface, monitoring capabilities, and any other characteristic described in the S&D Pattern. However, they may have differences, such as the specific context conditions that must be met before deploying it, their performance, target platform, programming language

or any other feature not fixed yet by the pattern. A specific component providing encryption services or a web service providing time stamping services are susceptible to become S&D Implementations.

We must emphasize that S&D Implementations are not the actual components but their representation. The actual components are made accessible to applications thanks to the SERENITY Runtime Framework (presented in next section), who maps from the S&D Implementations to the actual executable components.

3.2. Automated management of S&D Patterns

The scenario presentation suggested some entity in charge of deploying and monitoring the pattern selected for Charlie's device. This entity takes form in this section and is what we call *SERENITY Runtime Framework*. The SERENITY Runtime Framework (SRF from here onwards) is in charge of negotiating the terms of the dialogue and navigating throughout the S&D Artefacts' hierarchy. Figure 3 shows a simplified structure with the main components of the framework.

Instances of SRF can be embedded in any type of device with a minimum computational power (Charlie's game console in the example scenario). Every SRF instance acts like a dynamic S&D provider, providing solutions to applications and monitoring the correctness of the provided solutions. For that purpose, each SRF instance has an S&D Library containing the artefacts that describe the available security and dependability solutions. This library is searched by the SRF for the best pattern to meet the requirements. After selecting a solution, the SRF uses the information provided by the S&D Implementations and dynamically deploys the corresponding *Executable Component*.

Two elements of Figure 3 are also worth mentioning here: the *Context Manager* and the *S&D Manager*. A brief example will help us to understand their purpose.

After some racing Bob decides to check his agenda to confirm when he is expected to send his Personal Progress Report for the Software Engineering Group Projects class. While Alice's console was only SERENITY-aware, Bob's new console is fully SERENITY-enabled. His game-console has a web browser so he just connects to the University Virtual Campus using the college private LAN. Being a trusted network, a simple authentication pattern is used to connect to the Intranet of the University. As lunch time is approaching, Bob takes his game console with him (as usual) and goes to his favourite restaurant, just down the street. While having his "Burrito Deluxe", he suddenly has a great idea for the final presentation of his project and tries to connect to the group forum in the intranet to post-it before losing the idea.

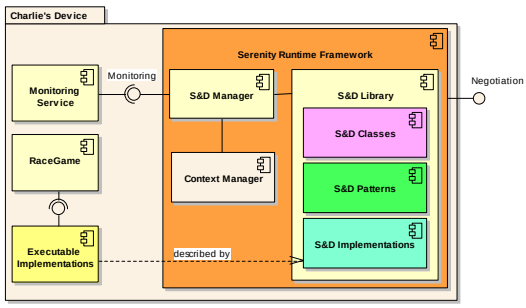

Figure 3. Simplified perspective of SRF

At this point, the *Context Manager* realizes that the browser is trying to connect to the intranet from an untrusted network. The S&D Pattern that was active for providing a confidential channel is no longer valid and the system must be reconfigured using a new pattern. The *S&D Manager* analyses the context information coming from the Context Manager along with the current S&D Requirements and triggers a query to find the better solution available in the S&D Library. This solution is then activated and connected to the browser. If there is no appropriate solution in the S&D Library matching the requirements and able to handle the new situation, the SRF instance denies the access to the browser and informs the user.

SRF instances present interfaces to communicate with other systems. When Charlie tried to join the game, the *Negotiation Interface* was used to mediate in order to reach an agreement on the parameters and restrictions for the communication. This step helped to discriminate between TLS and SSL channels. It is important to note that as long as non SERENITY-enabled systems (e.g. Alice's device) implement this interface, they will also be able to communicate with SERENITY-enabled nodes (Bob's and Charlie's ones).

As SERENITY faces runtime scenarios in which the context and requirements can evolve on time, a *Monitoring Interface* is also provided. Each S&D Pattern includes specific information on how to monitor its behaviour. The monitoring interface accesses that information and forwards it to an appropriate monitor, which is responsible for checking whether the component is acting as predicted or not. Monitoring rules defined in S&D Pattern allows checking relevant aspects such as the expected size of a transmission, the allowed resources to be accessed by a component, expected latency rates, and so on.

4. Related work

The concept of security pattern was introduced to support system engineers in selecting appropriate security solutions. But currently most security patterns

are expressed in textual form, as informal indications on how to solve some particular security problem [3, 4]. However, there is an increasing interest in proposing more formal and precise descriptions to enhance the special needs of secure-ware systems with high dependency on the environment in which these systems are deployed. Konrad et al. in [5] study the security patterns proposed by *Gamma et al.* in [6] and uses UML to represent both the structural and behavioural information. *Behaviour* or *Supported Principles* are two new fields that convey essential information that has not been necessary in the general design patterns but appears as mandatory in the new security context. Other proposals describe more precise representations based on UML diagrams, but they do not include enough semantic information for automating their processing [7].

In an ambitious paper, Eduardo B. Fernandez follows in [8] the track initiated in [7] (here the author combines for the first time the idea of multiple architectural levels with the use of design patterns) and proposes a methodology for using security patterns at every stage of the software lifecycle. Following this approach, Wassermann and Cheng present in [9] a revision of most of the patterns from [8] and [10] and categorise them in terms of their abstraction level. However, none of these approaches face the possible change of requirements at runtime, and the consequent need of adapting or changing the patterns in use.

Some authors propose formal characterizations of patterns. The idea of precisely specifying a given class using class invariants and pre- and post-conditions for characterizing the behaviours of individual methods is the basis of the design by contract [11]. Evolutions of that approach appear in [12], where logic formalism is proposed with an associated graphical notation to specify rich structural properties. Also using contracts, in [13] authors try to preserve the design integrity of a system so that it continues to be faithful to the patterns used in its initial design even as it evolves to meet changing requirements. Following the formal methods approach, Mikkonen in [14] introduces classes, relations and actions to formalize patterns representation and to allow complex specifications by the combination of patterns.

5. Conclusions and Future work

In this paper we have presented the artefacts that we use to implement the concept of S&D Pattern. We have shown using an example scenario how these artefacts serve are used in the process of providing S&D to applications used in highly dynamic, heterogeneous and distributed environments.

Our current work is focused on the further development of these artefacts to cover additional aspects, and on the development of support tools. We have already developed tools for the creation of these artefacts, and our current work is aimed at the tools for automated selection, adaptation and management of the solutions at runtime.

6. References

[1] Francisco Sanchez-Cid, Antonio Muñoz, Daniel Serrano, and M.C. Gago, “Software Engineering Techniques Applied to AmI: Security Patterns”. *In Proceedings of the First International Conference on Ambient Intelligence Developments*, Springer, Sept. 2006, pp. 108-124.

[2] C. Canal, L. Fuentes, E. Pimentel, J.M. Troya, and A. Vallecillo, “Adding Roles to CORBA Objects”, *IEEE Transactions on Software Engineering 29(3)*, Mar. 2003.

[3] Kienzle, D.M., Elder, M.C., “Final Technical Report: Security Patterns for Web Application Development” *Available at http://www.scrypt.net/~celer/securitypatterns/final%20report.pdf*.

[4] IBM's Security Strategy team, “Introduction to Business Security Patterns. An IBM White Paper”, *Available at http://www-3.ibm.com/security/patterns/intro.pdf*, 2004.

[5] Konrad, S., B.H.C. Cheng, Campbell, Laura. A., and Wassermann R., “Using Security Patterns to Model and Analyze Security Requirements”, *Proc. Requirements for High Assurance Systems Workshop*, 2003.

[6] Gamma, E., Helm, R., Johnson, R., and Vlissides, J., “Design patterns: Elements of Reusable Object-Oriented Software”, *Addison-Wesley*, 1994.

[7] E.B. Fernandez, and Pan, Rouyi, “A pattern language for security models”, *PLoP’01 Conference*, 2001.

[8] E.B. Fernandez, “Security patterns”, *Procs. of the Eigth International Symposium on System and Information Security*, Keynote talk, Sao Jose dos Campos, Brazil, 2006.

[9] R. Wassermann, and B.H.C. Cheng, “Security Patterns” *Technical Report MSU-CSE-03-23*, Aug. 2003.

[10] Yoder, J. and Barcalow, J., “Architectural Patterns for Enabling Application Security”, *Pattern Languages of Program Design*, Reading, MA: Addison Wesley Publishing Company, 2000, pp. 301-336.

[11] Hallstrom, J. O., Soundarajan, N., and Tyler, B., “Monitoring Design Pattern Contracts”, *In Proc. of the FSE-12 Workshop on Specification and Verification of Component-Based Systems*, 2004, pp. 87-94.

[12] Allenby, K., and Kelly, T., “Deriving Safety Requirements Using Scenarios”, *In Proc. of the 5th IEEE International Symposium on Requirements Engineering*, 2001.

[13] Hallstrom, J. O., and Soundarajan, N., “Pattern-Based System Evolution: A Case-Study”, *In the Proc of the 18th International Conference on Software Engineering and Knowledge Engineering*, 2006.

[14] Mikkonen, T., “Formalizing design patterns”. *In Proc. of 20th ICSE*, IEEE Computer Society Press, 1998, pp. 115-124.