# Chapter 30
# Developing Secure Software Using UML Patterns

**Holger Schmidt**
*TÜV Informationstechnik GmbH, Germany*

**Denis Hatebur**
*University Duisburg-Essen, Germany, & ITESYS Institut für Technische Systeme GmbH, Germany*

**Maritta Heisel**
*University Duisburg-Essen, Germany*

## ABSTRACT

*This chapter presents a security engineering process based on UML security problem frames and concretized UML security problem frames. Both kinds of frames constitute patterns for analyzing security problems and associated solution approaches. They are arranged in a pattern system that makes dependencies between them explicit. The authors describe step-by-step how the pattern system can be used to analyze a given security problem and how solution approaches can be found. Then, solution approaches are specified by generic security components and generic security architectures, which constitute architectural patterns. Finally, the generic security components and the generic security architecture that composes them are refined, and the result is a secure software product built from existing and/or tailor-made security components.*

## INTRODUCTION

It is acknowledged that a thorough requirements engineering phase is essential to develop a software product that matches the specified requirements. This is especially true for *security requirements*.

We propose a security engineering process that focuses on the early phases of software development covering security requirements and security architectures. The basic idea is to make use of special *patterns* for security requirements analysis and development of security architectures.

Security requirements analysis makes use of patterns for structuring, characterizing, and analyzing *problems* that frequently occur in security engineering. Similar patterns for functional requirements have

DOI: 10.4018/978-1-5225-3923-0.ch030

been proposed by Jackson (2001). They are called *problem frames*. Accordingly, our patterns are named *security problem frames*. Furthermore, for each of these frames, we have defined a set of *concretized security problem frames* that take into account generic security mechanisms to prepare the ground for solving a given security problem. Both kinds of patterns are arranged in a pattern system that makes dependencies between them explicit. We describe how the pattern system can be used to analyze a given security problem, how solution approaches can be found, and how dependent security requirements can be identified.

Afterwards, we develop a corresponding security architecture based on platform-independent *generic security components* and *generic security architectures.* Each concretized security problem frame is equipped with a set of generic security architectures that represent the internal structure of the software to be built by means of a set of generic security components. After a generic security architecture and generic security components are selected, the latter must be refined to platform-specific security components. For example, existing component frameworks can be used to construct a platform-specific security architecture that realizes the initial security requirements.

The rest of the chapter is organized as follows: First, we introduce problem frames and present a literature review. Second, we give an overview of our security engineering process. Then we present the different development phases of the process in detail. Each phase of our process is demonstrated using the example of a secure text editor application. Finally, we outline future research directions and give a summary and a discussion of our work.

## BACKGROUND

In the following, we first present problem frames and second, we discuss our work in the context of other approaches to security engineering.

### Problem Frames

Patterns are a means to reuse software development knowledge on different levels of abstraction. They classify sets of software development problems or solutions that share the same structure. Patterns are defined for different activities at different stages of the software life-cycle. *Problem frames* by Jackson (2001) are a means to analyze and classify software development problems. *Architectural styles* are patterns that characterize software architectures (for details see (Bass & Clements & Kazman, 1998) and (Shaw & Garlan (1996)). *Design patterns* by Gamma, Helm, Johnson, and Vlissides (1995) are used for finer-grained software design, while *idioms* by Coplien (1992) are low-level patterns related to specific programming languages.

Using patterns, we can hope to construct software in a systematic way, making use of a body of accumulated knowledge, instead of starting from scratch each time. The problem frames defined by Jackson (2001) cover a large number of software development problems, because they are quite general in nature. Their support is of great value in the area of software engineering. Jackson (2001) describes them as follows: „A problem frame is a kind of pattern. It defines an intuitively identifiable problem class in terms of its context and the characteristics of its domains, interfaces, and requirement." (p. 76). Jackson introduces five basic problem frames named *required behaviour*, *commanded behaviour*, *information display*, *simple workpieces*, and *transformation*.