A Security Engineering Process based on Patterns

Denis Hatebur[1,2], Maritta Heisel[2], and Holger Schmidt[2]

[1] ITESYS Institut für technische Systeme GmbH, Dortmund, Germany
d.hatebur@itesys.de
[2] Workgroup Software Engineering, Department of Computational and Cognitive Sciences,
University of Duisburg-Essen, Duisburg, Germany
{denis.hatebur, maritta.heisel, holger.schmidt}@uni-duisburg-essen.de

Abstract

We present a security engineering process based on security problem frames *and* concretized 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. We describe step-by-step how the pattern system can be used to analyze a given security problem and how solution approaches can be found. Further, we introduce a new frame that focuses on the privacy requirement* anonymity.

1 Introduction

When building secure systems, it is instrumental to take security concerns into account right from the beginning of the development process [1]. Hence, *security engineering* should become an integral part of the software engineering process when developing secure software systems.

In this paper, we present a security engineering process that focuses on the early phases of software development. The basic idea is to make use of special patterns defined for structuring, characterizing, and analyzing problems that occur frequently in security engineering. Similar patterns for functional requirements have been proposed by Michael Jackson [6]. They are called *problem frames*. Accordingly, our patterns are named *security problem frames*.

The advantage of using problem frames in requirements engineering is that problems are mapped to well-known problem classes that are practically relevant. Once a problem is successfully fitted to a problem frame, its most important characteristics are known, because these are shared by all problems fitting to the frame.

According to the security engineering process proposed in this paper, in a first step security problem frames must be selected and their variable parts must be instantiated. Furthermore, for each security problem frame, we define a set of *concretized security problem frames* that take into account generic security mechanisms to prepare the ground for solving a given problem. Security problem frames and their concretized counterparts are arranged in a pattern system that makes dependencies between solution approaches and security problems explicit. Our security engineering process incorporates that pattern system. After choosing and instantiating concretized security problem frames in a second step, further iterations of the proposed security engineering process yield to all dependent subproblems that must be solved in order to establish certain solution approaches for security problems.

In the following, we explain how to develop software using problem frames as proposed by Jackson [6] in Sect. 2. In Sect. 3, we introduce the (concretized) security problem frames approach and as a sample frame, we discuss a new (concretized) security problem frame that focuses on the privacy requirement anonymity in Sect. 4. Section 5 presents a pattern system of (concretized) security problem frames. In Sect. 6, we present a security engineering process tailor-made to utilize (concretized) security problem frames. Section 7 discusses related work, and we conclude in Sect. 8.

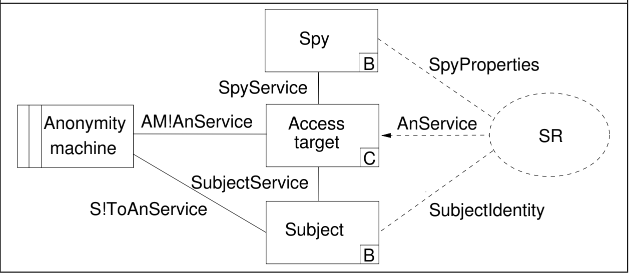
2 Requirements Analysis using Problem Frames

Problem frames are a means to describe software development problems. They were invented by Michael Jackson [6], who 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."* Problem frames are described by *frame diagrams*, which basically consist of rectangles, a dashed oval, and different links between these (see frame diagram in Fig. 1). The task is to construct a *machine* that improves the behavior of the environment it is integrated in.

Plain rectangles denote *domains* that already exist in the application environment, a rectangle with a double vertical stripe denotes the machine to be developed, and *requirements* are denoted with a dashed oval. The connecting

SPF: Anonymity

Frame diagram



Security requirement (SR)

Spy cannot deduce relation between Subject and AnService using AccessTarget.

Declarations

$at : AccessTarget$; $su : Subject$; $tas : ToAnService$;
$sp : Spy$; $originator : \mathbb{P}(Subject \times ToAnService)$

Preconditions

$tas \in confIf$

Postconditions

$(sp, at, originator) \in nonDeduce$

Related

SPF: Integrity-Preserving Data Transmission
SPF: Authentication

Figure 1. Security problem frame for anonymity

lines represent interfaces that consist of *shared phenomena*. Shared phenomena may be events, operation calls, messages, and the like. They are observable by at least two domains, but controlled by only one domain. A dashed line represents a requirements reference, and an arrow indicates that the requirements *constrain* a domain. If a domain is constrained by the requirements, we must develop a machine, which controls this domain accordingly. Furthermore, Jackson distinguishes *causal* domains that comply with some physical laws, *lexical* domains that are data representations, and *biddable* domains that are usually people.

In the frame diagram depicted in Fig. 1, the “C” indicates that the domain Access target is a causal domain, and the “B” indicates that the domains Spy and Subject are biddable domains. The notation AM!AnService means that the phenomena in the set AnService are controlled by the machine domain Anonymity machine.

Problem frames greatly support developers in analyzing problems to be solved. They show what domains have to be considered, and what knowledge must be described and reasoned about when analyzing the problem in depth. Developers must elicit, examine, and describe the relevant properties of each domain. These descriptions form the *domain knowledge*.

The domain knowledge consists of *assumptions* and *facts*. Assumptions are conditions that are needed, so that the requirements are accomplishable. Usually, they describe required user behavior. For example, it must be assumed that a user ensures not to be observed by a malicious user when entering some input. Facts describe fixed properties of the problem environment that hold regardless of how the machine is built.

Requirements describe the environment, the way it should be, after the machine is integrated. In contrast to the requirements, the *specification* of the machine gives an answer to the question: “How should the machine act, so that the system fulfills the requirements?” Specifications are descriptions that are sufficient for building the machine. They are implementable requirements. For the correctness of a specification S, it must be demonstrated that S, the facts F, and the assumptions A imply the requirements R.

Software development with problem frames proceeds as follows: first, the environment in which the machine will operate is represented by a *context diagram*. Like a frame diagram, a context diagram consists of domains and interfaces. However, a context diagram contains no requirements, and it is not shown which domain is in control of the shared phenomena. Then, the problem is decomposed into subproblems. If ever possible, the decomposition is done in such a way that the subproblems fit to given problem frames. To fit a subproblem to a problem frame, one must instantiate its frame diagram, i.e., provide instances for its domains, interfaces, and requirement. The instantiated frame diagram is called a *problem diagram*. For example, a requirement such as “Chat users should be able to send anonymous messages” can be used for instantiating the anonymity frame of Fig. 1.

Since the requirements refer to the environment in which the machine must operate, the next step consists in deriving a *specification* for the machine (see [7] for details). The specification describes the machine and is the starting point for the development of the machine.

3 (Concretized) Security Problem Frames Approach

Security problem frames [4] are special kinds of problem frames, which consider security requirements. They strictly refer to the problems concerning security, without anticipating solutions. For example, we may require that data is kept confidential during transmission without being obliged to mention encryption, which is a means to achieve confidentiality.

The benefit of considering security requirements without reference to potential solutions is the clear separation of problems from their solutions, which leads to a better understanding of the problems and enhances the reusability of the problem descriptions, since they are completely independent of solution technologies. Further, the separation of problems and solutions is helpful to analyze conflicting requirements and the interaction of security and other non-functional requirements [10].

Solving a security problem is initiated by choosing and instantiating a concretized security problem frame. These frames are derived from the security problem frames by considering generic security mechanisms (such as using symmetric or asymmetric encryption for keeping data confidential during transmission).

Both kinds of frames are equipped with preconditions and postconditions expressed as logical formulas [5]. The preconditions express what conditions must be met by the environment for a frame to be applicable; the postconditions are a formal representation of a (concretized) security requirement, i.e., they describe what (concretized) security requirement will be achieved by the machine to be built.

Each (concretized) security problem frame is described according to the following template (see Fig. 1):

Name The name specifies what kind of security problem is addressed by the security problem frame (SPF) or concretized security problem frame (CSPF).

Frame diagram This diagram shows the relevant domains and their interfaces, as well as the (concretized) security requirement.

(Concretized) security requirement The (concretized) security requirement is stated informally.

Declarations Entities that are necessary for stating the preconditions and postconditions formally are declared. These entities are implicitly universally quantified. We use the domains and interfaces of the corresponding frame diagram as data types.

Preconditions Conditions are given that must be met by the environment for the frame to be applicable.

Postconditions These conditions are a formal representation of the (concretized) security requirement.

Related Different patterns will often be used in combination. Those frames that are commonly used in combination with the described frame are mentioned here.

4 (Concretized) Security Problem Frames for Anonymity

Many privacy-critical systems are required to guarantee that a subject using a service via a certain platform should not be identifiable. In such a case, the system must conceal the relation between the subject and the used service. For example, a user of an online chat system should be able to send messages to other chat users without being revealed as the originator of the messages. A more critical example is an e-voting system, which should keep the relation between the vote and its voter confidential. To describe and analyze such privacy problems, we propose the security problem frame for anonymity shown in Fig. 1. The biddable domain Subject denotes a person or a system (e.g., a chat user) that makes use of a service ToAnService (service to be anonymized, e.g., send a message). The service is provided through a certain platform, which is represented by the causal domain Access target (e.g., the Internet). Those subjects who want to reveal the identity of a subject using a service are represented by the biddable domain Spy (e.g., the other chat users). The machine domain is called Anonymity machine, and its task is to prevent that a spy deduces the relation between a subject and a used service. Therefore, the interface AnService is controlled by the machine. It represents the already anonymized service, which is used by the subject. The interface SubjectService between the domains Subject and Access target represents the fact that a subject can directly access the platform that provides the used service. A similar fact is represented by the interface SpyService between the domains Spy and Access target. The control directions of both interfaces are not stated, because both directions are possible for concrete instances of the frame. The requirements reference SpyProperties reflects relevant properties of the Spy domain, e.g., details about its equipment, knowledge, and strength. The requirements reference SubjectIdentity reflects the unique identity of the Subject domain.

The precondition $tas \in confIf$ expresses that a confidential path *tas* between the machine domain Anonymity machine and the domain Subject is necessary. It describes that data transferred using this interface *tas* is kept confidential. Otherwise, a spy could easily read the relation between Subject and ToAnService.

The postcondition expresses that a spy *sp* is not able to deduce the *originator* using the access target *at*.

The list of related frames contains the frames for integrity-preserving data transmission and authentication [4], because in many cases the data transferred using the interface ToAnService must remain unchanged and the subject must be authenticated.

To obtain a concretized security problem frame for anonymity, we introduce the concept of an ID (e.g., a user name or an IP address) to represent a subject's identity. The frame diagram structure of the concretized security problem frame for anonymity is the same as the frame diagram depicted in Fig. 1. The usage of an ID only influences the interfaces of the frame: each service is related to the ID of a subject.

In a similar way, we can construct (concretized) security problem frames for pseudonymity.

5 Pattern System

We have derived a pattern system by matching the preconditions of the concretized security problem frames with the postconditions of the security problem frames [5]. The dependencies between the security problem frames and their concretized counterparts are illustrated graphically. The pattern system contains all available (concretized) security problem frames and the dependencies between them, indicated by arrows pointing from the concretized security problem frames to the security problem frames they depend on. For example, the precondition of the concretized security problem frame for anonymity shown in Fig. 1 contains a formula that expresses a confidential path. For this reason, the frame for anonymity depends on the frame for confidential data transmission. Hence, we draw a line from the box containing the CSPF Anonymity (using IDs) with an arrow pointing at the box SPF Confidential Data Transmission. All dependencies are established following this principle.

The pattern system is self-contained in the sense that for any precondition of a frame covered by the pattern system, there exists at least one frame contained in the pattern system that provides a matching postcondition. Therefore, the (concretized) security problem frames contained in the pattern system can be used to completely analyze a given security problem, whose initial security requirement is covered by one of the frames.

6 Security Engineering Process using Patterns

As we have presented the (concretized) security problem frames approach in [4], and we have extended it in [5] by pre- and postconditions to establish a pattern system, we present in this paper a consolidated variant of a security engineering process tailor-made to utilize (concretized) security problem frames. We call this process *SEPP* (Security Engineering Process using Patterns). Currently, SEPP has a strong focus on security requirements engineering. We plan to extend SEPP to support later phases of the software development life-cycle, incorporating component technology, architectural patterns, and security patterns [11]. An overview of SEPP is illustrated in Fig. 2.

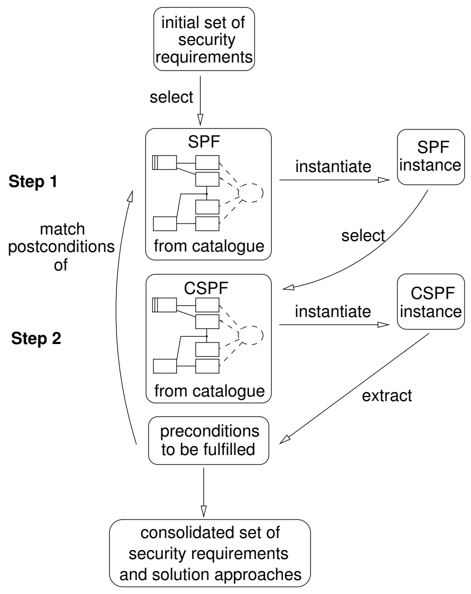


Figure 2. Security engineering process using patterns

Step 1 – Select and Instantiate Security Problem Frames Developing a secure system using (concretized) security problem frames starts after the security goals and an initial set of security requirements are elicited using, e.g., the MSRA (formerly known as CREE) method [2]. We provide a catalogue of security problem frames and their concretized counterparts. Each elicited security requirement must be compared to the informal descriptions of the security requirements of the security problem frames contained in the catalogue. For example, if users must access certain services while hiding their real identity, the security problem frame for anonymity (or pseudonymity) presented in Sect. 4 is applicable. After an appropriate security problem frame is determined for each given security requirement, these frames must be instantiated. When instantiating a security problem frame, the domains, phenomena, interfaces, pre- and postconditions, and the security requirement must be assigned concrete values.

To instantiate the domains that represent potential attackers (e.g., the domain Spy in Fig. 1), a certain level of skill, equipment, and determination of the potential attacker must be assumed. Via these assumptions, threat models are integrated into the method.

After the security problem frames are instantiated, a security engineer proceeds with checking the “Related” sections of the used frames, which mention those frames that are commonly used in combination with the described frame. This helps to find missing security requirements right at the beginning of the security requirements engineering process.

Step 2 – Select and Instantiate Concretized Security Problem Frames To solve a security problem characterized by an instance of a security problem frame, the process continues with choosing a solution approach (e.g., using an anonymous proxy to browse the web), thereby instantiating appropriate concretized security problem frames. Afterwards, the preconditions of the instantiated concretized security problem frames must be inspected. If the instantiated preconditions are already considered in a previous iteration of SEPP’s second step, the problems described by the respective preconditions need not be taken into account again. Otherwise, two alternatives are possible to guarantee that these preconditions hold: either, they can be *assumed* to hold, or they have to be *established* by instantiating a further security problem frame whose postconditions match the preconditions to be established. Such a frame can easily be determined using the pattern system by following the arrow(s) pointing from the instantiated concretized security problem frame under consideration to the security problem frames that can be used to address dependent subproblems, see Fig. 2.

What assumptions are reasonable depends on the threats the system should be protected against. Moreover, some assumptions cannot be avoided completely, because it may be impossible to achieve a security requirement. For example, we must assume that an administrator can distinguish a fake user from an authentic user when creating a user account and providing user name and password.

Only in the case that preconditions *cannot* be assumed to hold, one must instantiate further appropriate security problem frames, and the procedure is repeated until all preconditions of all applied concretized security problem frames can be proved or assumed to hold. Therefore, the SEPP process results in a set of consolidated security problems and solution approaches that additionally contains all dependent security problems and corresponding solution approaches, some of which may not have been known initially.

The next step in the software development life-cycle is to derive a specification, which describes the machine and is the starting point for the development of the machine. As it is beyond the scope of this paper, this issue will not be discussed in detail.

7 Related Work

To elicitate security requirements, the threats to be considered must be analyzed. Lin et al. [8] use the ideas underlying problem frames to define so-called anti-requirements and the corresponding *abuse frames*. The purpose of anti-requirements and abuse frames is to analyze security threats and derive security requirements. Hence, abuse frames and security problem frames complement each other.

Gürses et al. [2] present the MSRA (formerly known as CREE) method for multilateral security requirements analysis. Their method concentrates on confidentiality requirements elicitation and employs use cases to treat functional requirements.

Haley et al. [3] present a framework for security requirements engineering. It defines the notion of security requirements, considers security requirements in an application context, and helps answering the question whether the system can satisfy the security requirements. Their definitions and ideas overlap our approach, but they do not use patterns and they do not give concrete guidance to identify and elicit *all* requirements.

Popp et al. [9] apply extended use cases in the field of security-critical system development. Use cases extended by security information are used to develop the specification of security-critical systems, whereas SEPP focuses on identifying and analyzing requirements beforehand.

Security patterns [11] are applied later, in the phase of detailed design. The relation between our concretized security problem frames, which still express problems, and security patterns is much the same as the relation between problem frames and design patterns: the frames describe problems, whereas the design/security patterns describe solutions on a fairly detailed level of abstraction.

8 Conclusion and Perspectives

In this paper, we have presented SEPP, a security engineering process based on *security problem frames* and *concretized security problem frames*. These special kinds of problem frames are arranged in a pattern system, which serve to structure, characterize, analyze, and finally solve software development problems in the area of software and system security. SEPP supports to obtain a complete set of security requirements by analyzing the preconditions of the used problem frames and deciding if they can be assumed or must be established by applying more security frames.

Compared to the papers [4] and [5], this paper presents a new security problem frame that allow us the take the the privacy requirement of anonymity into account. Furthermore, it contains an elaborated version of the security engineering process and its underlying pattern system.

In the future, we intend to find new patterns to extend the catalogue of security problem frames and concretized security problem frames. Additionally, we plan to elaborate more on the later phases of software development. For example, we want to investigate how to integrate component technology in the development process. Finally, we plan to provide tool support for SEPP.

References

[1] E. Fernandez. A methodology for secure software design. In *Proc. Int. Conference on Software Engineering Research and Practice*, 2004.

[2] S. Gürses, J. H. Jahnke, C. Obry, A. Onabajo, T. Santen, and M. Price. Eliciting confidentiality requirements in practice. In *CASCON '05: Proceedings of the 2005 conference of the Centre for Advanced Studies on Collaborative research*, pages 101–116. IBM Press, 2005.

[3] C. B. Haley, J. D. Moffett, R. Laney, and B. Nuseibeh. A framework for security requirements engineering. In *SESS '06: Proceedings of the 2006 international workshop on Software engineering for secure systems*, pages 35–42, New York, NY, USA, 2006. ACM Press.

[4] D. Hatebur, M. Heisel, and H. Schmidt. Security engineering using problem frames. In G. Müller, editor, *Proceedings of the International Conference on Emerging Trends in Information and Communication Security (ETRICS)*, LNCS 3995, pages 238–253. Springer-Verlag, 2006.

[5] D. Hatebur, M. Heisel, and H. Schmidt. A pattern system for security requirements engineering. In *Proceedings of the International Conference on Availability, Reliability and Security (AReS)*, IEEE Transactions, pages 356–365. IEEE, 2007.

[6] M. Jackson. *Problem Frames. Analyzing and structuring software development problems*. Addison-Wesley, 2001.

[7] M. Jackson and P. Zave. Deriving specifications from requirements: an example. In *Proceedings 17th Int. Conf. on Software Engineering, Seattle, USA*, pages 15–24. ACM Press, 1995.

[8] L. Lin, B. Nuseibeh, D. Ince, and M. Jackson. Using abuse frames to bound the scope of security problems. In *Proceedings of 11th IEEE International Requirements Engineering-Conference (RE'04)*, pages 354–355, 2004.

[9] G. Popp, J. Jürjens, G. Wimmel, and R. Breu. Security-critical system development with extended use cases. In *APSEC*, pages 478–487, 2003.

[10] H. Schmidt and I. Wentzlaff. Preserving software quality characteristics from requirements analysis to architectural design. In *Proceedings of the European Workshop on Software Architectures (EWSA)*, volume 4344/2006, pages 189–203. Springer Berlin / Heidelberg, 2006.

[11] M. Schumacher, E. Fernandez-Buglioni, D. Hybertson, F. Buschmann, and P. Sommerlad. *Security Patterns: Integrating Security and Systems Engineering*. Wiley & Sons, 2005.