Security Engineering using Problem Frames

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

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

Abstract. We present a method for security engineering, which is based on two special kinds of problem frames that serve to structure, characterize, analyze, and finally solve software development problems in the area of software and system security. Both kinds of problem frames constitute *patterns* for representing security problems, variants of which occur frequently in practice. We present *security problem frames*, which are instantiated in the initial step of our method. They explicitly distinguish security problems from their solutions. To prepare the solution of the security problems in the next step, we employ *concretized security problem frames* capturing known approaches to achieve security. Finally, the last step of our method results in a specification of the system to be implemented given by concrete security mechanisms and instantiated *generic sequence diagrams*. We illustrate our approach by the example of a secure remote display system.

1 Introduction

Security engineering [1] is a discipline concerned with building secure systems to remain dependable in the face of malice, error and mischance. Tools, processes, and methods are needed to analyze, design, implement, and test secure systems, and to evolve existing systems as their environment evolves. *Security engineers* must be cross-disciplinary experts in cryptography, computer security, formal methods, and software engineering, and they must have knowledge about applied psychology, the law, organizational and audit methods.

Knowing that building security-critical systems is a highly sensitive process, it is important to reuse the experience of commonly encountered challenges in this field. This idea of using *patterns* has proved to be of value in software engineering, and it is also a promising approach in security engineering.

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* [10] are patterns that classify software development *problems*. *Architectural styles* are patterns that characterize software architectures [2]. *Design patterns* [5] are used for finer-grained software design[3], while *idioms* are low-level patterns related to specific programming languages [4].

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 [10] cover a large number of software development

[3] Design patterns for security have also been defined, see Section 8.

problems, because they are quite general in nature. Their support is of great value in the area of software engineering for years. To support software development in more specific areas such as security engineering, however, specialized problem frames are needed. Jackson [10] states: “If you find the problem frame approach helpful you may want to find more frames to add to your personal repertoire. There are several situations that may suggest new problem frames.”

In this paper, we show how to use the problem frames approach in the area of security engineering. We first introduce Jackson’s problem frames in Section 2. Then we discuss a special security problem frame in Section 3 that captures authentication, a software development problem occurring frequently in the area of security engineering. Furthermore, we define a concretized security problem frame in Section 4, that captures known approaches to achieve authentication. We present a *generic security protocol* represented by generic sequence diagrams as a basis for a more concrete specification in Section 5.

We propose a method tailor-made for security engineering using security problem frames, their concretized counterparts, and generic security protocols to proceed from a security problem towards a solution. Initially, a security engineer must describe a security problem by an instantiated security problem frame. Then, concretized security problem frames must be employed to derive a specification given by concrete security mechanisms and instantiated generic sequence diagrams. Section 6 gives an overview of this method.

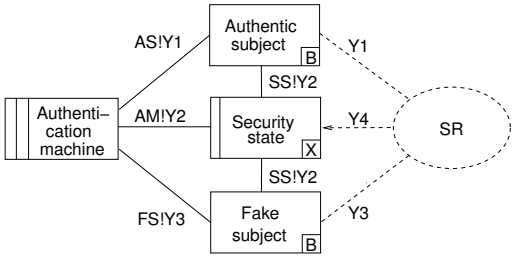
We illustrate our approach by developing a secure remote display system in Section 7. Section 8 discusses related work, and we conclude in Section 9.

2 Problem Frames

Problem frames are a means to describe software development problems. They were invented by Michael Jackson [10], 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 and links between these (see Fig. 1). The task is to construct a *machine* that improves the behavior of the environment it is integrated in.

Plain rectangles denote *application domains* (that already exist), a rectangle with a single vertical stripe denotes a *designed domain* physically representing some information, a rectangle with a double vertical stripe denotes the machine to be developed, and *requirements* are denoted with a dashed oval. The connecting lines represent interfaces that consist of *shared phenomena*. A dashed line represents a requirements reference, and the arrow shows that it is a *constraining* reference.

Furthermore, Jackson distinguishes *causal* domains that comply with some physical laws, *lexical* domains that are data representations, and *biddable* domains that are usually people. *Connection domains* connect two other domains. They represent a communication medium between these domains. Examples are devices that measure vital factors of patients, or a keyboard that is used to type user input. Connection domains have to be considered if connections are unreliable, introduce delays that are an essential part of the problem, convert phenomena, or if they are mentioned in the requirements.



SR: *Security state* represents that access is granted for the *Authentic subject* and that access is denied for the *Fake subject*

Fig. 1. Authentication frame diagram

In the frame diagram of Fig. 1, the "X" indicates that the corresponding domain is a lexical domain. The notation "AS!Y1" means that the phenomena *Y1* are controlled by the biddable domain *Authentic subject*, which is indicated by "B".

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*, which can be explained essentially in the following way [10]: "These descriptions are *indicative* – they indicate the objective truth about the domains, what's true regardless of the machine's behaviour."

Requirements describe the environment, the way it should be, after the machine is integrated. *Assumptions* are conditions that are needed, so that the requirements are accomplishable. Usually, they describe required user behavior. For example, we cannot distinguish a fake user from an authentic user if both have the same credentials. Hence, we must assume that only the authentic user knows the credentials. 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 domain knowledge D, and the assumptions A imply the requirements R ($A \wedge D \wedge S \Rightarrow R$, where $A \wedge D \wedge S$ must be non-contradictory).

Software development with problem frames proceeds as follows: first, the environment in which the machine will operate is represented by a *context diagram* (see upper left-hand side of Fig. 4). Like a frame diagram, a context diagram consists of domains and interfaces. However, a context diagram contains no requirements, and it is not shown who 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, phenomena, interfaces and requirements. The instantiated frame diagram is called a *problem diagram*. 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 [11] for details). The specification is the starting point for the development of the machine.

Successfully fitting a problem to a given problem frame means that the concrete problem indeed exhibits the properties that are characteristic for the problem class defined by the problem frame. Since all problems fitting to a problem frame share the same characteristic properties, their solutions will have common characteristic properties, too. Therefore, it is worthwhile to look for solution structures that match the problem structures defined by problem frames.

3 Security Problem Frames

To meet the special demands of software development problems occurring in the area of security engineering, we developed three security problem frames considering the security problems of authentication, confidentiality, and integrity. For reasons of space, we only present the security problem frame for authentication in this paper. The security problem frames for confidentiality and integrity are presented in [7].

Security problem frames consider *security requirements*. The goal is the construction of a machine that fulfills the security requirements. The security problem frames we have developed strictly refer to the *problems* concerning security. They do not anticipate a solution. For example, we may require the confidential transmission of data without being obliged to mention encryption, which is a means to achieve confidentiality.

Solving a security problem is achieved by choosing generic security mechanisms (e.g., encryption to keep data confidential), thereby transforming security requirements into *concretized security requirements* (see Section 4 for details). 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 supports the reusability of the security problem frames.

In contrast to Jacksons' problem frames, security problem frames and also their concretized counterparts contain patterns for the security requirements, they explicitly involve and describe a potential attacker (threat model), they integrate assumptions, and they also consider non-functional requirements (such as condidentiality).

Security Problem Frame for Authentication
Authentication of users and other systems is an important issue in many security-critical systems. Authentication is the problem to verify a claimed identity that is necessary to control access to data. Accessing data includes not only reading data, but also writing data, executing programs, and creating new data.

The frame diagram in Fig. 1 depicts this security problem. The domain *Authentic subject* in the frame diagram represents an authentic user or another authentic system. In contrast, the domain *Fake subject* represents a fake user or another fake system. The domain *Security state* represents the fact that access is granted to the *Authentic subject* domain and denied to the *Fake subject* domain. The *Security state* domain is externally visible, because the subject must be at least implicitly informed whether the access is granted or denied. The security requirement *SR* is stated according to this description.

4 Concretized Security Problem Frames

Security requirements are often difficult to address. Our approach for dealing with security requirements is to transform them into *concretized security requirements*, which

take the functional aspects of a security problem into account (e.g., using a common secret to distinguish an authentic subject from a fake subject). For this purpose, we concretize the security problem frame introduced in Section 3, using generic security mechanisms.

For transforming security requirements into concretized security requirements, it is important to consider some basic properties of the involved domains, e.g., that the domain representing an authentic subject differs from the domain representing a fake subject. We call this kind of domain knowledge *basic domain knowledge* (D_{Basic}).

In transforming a security requirement SR into a concretized security requirement CSR, new concretized security problem frames evolve from the security problem frames. A detailed description of the transformation process can be found in [7], Section 4.

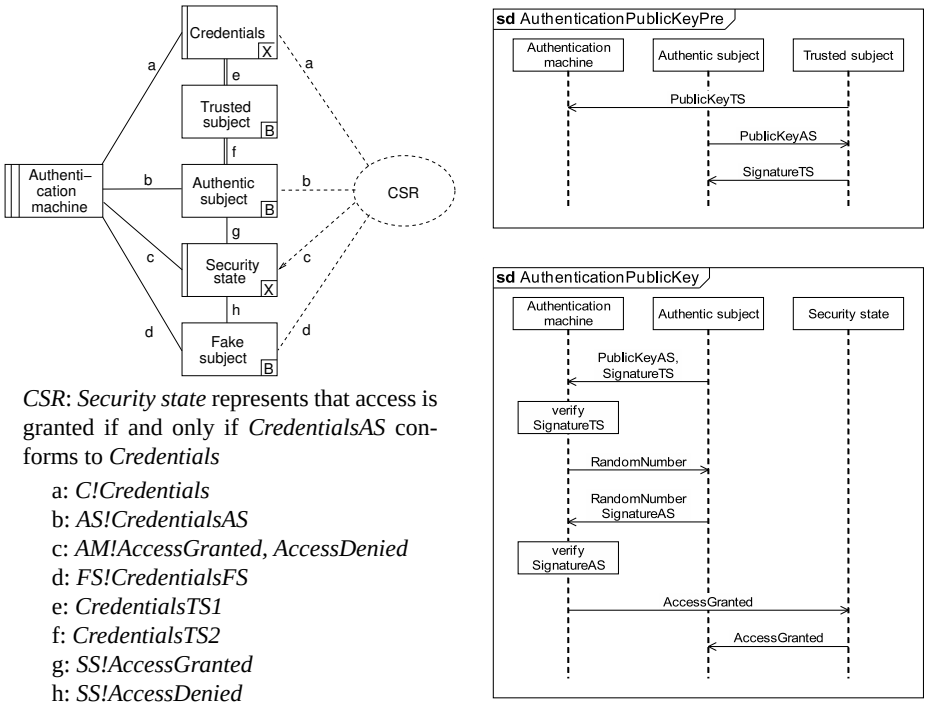
We must explicitly describe any assumptions made (denoted by A). Especially the strength of potential attackers must be characterized. The assumptions are necessary to check if the implication $A \wedge D_{Basic} \wedge CSR \Rightarrow SR$ is fulfilled. By proving this implication, we demonstrate that the concretized security problem frame is sufficient for the security problem frame under the assumptions A.

The security problem frame and its concretized counterpart presented in this paper and the frames considering confidentiality and integrity introduced in [7] are intended to be the first in a more complete collection. To consider other security problems such as availability or non-repudiation, it is necessary to integrate additional security problem frames and concretized security problem frames into the collection. Once a (relatively) complete collection is defined, it will be of considerable help for security engineers. For a new security-critical system to be constructed, the catalogue can be inspected in order to find the frames that apply for the given problem. Thus, such a catalogue helps to avoid omissions and to cover all security aspects that are relevant for the given problem.

Concretized Security Problem Frame for Authentication

The concretized security problem frame for authentication is shown on the left-hand side of Fig. 2. In the course of transforming the security requirement for authentication into a concretized security requirement, we introduce a designed domain *Credentials*, which makes it possible to distinguish between the domains *Authentic subject* and *Fake subject*. The *Credentials* domain must be known by the domain *Authentication machine*.

We introduce an additional domain *Trusted subject* that distributes the credentials to the machine and to the domain *Authentic subject* in the concretized security problem frame. That domain represents trusted subjects such as a system administrator for password-based authentication or a trust center in a public key infrastructure. It ensures that only the domain *Authentic subject* gets the credentials. Hence, we can state the assumption that the credentials distributed to the *Authentic subject* are represented by the phenomenon *CredentialsAS*, whereas a potential attacker represented by the domain *Fake subject* does not know these credentials. Therefore, we assume that it can only submit *CredentialsFS* to the machine. Depending on who generates the credentials represented by the symbolic phenomena *CredentialsTS1* and *CredentialsTS2*, the control direction of the interfaces *e* and *f* must be assigned during instantiation of this concretized security problem frame. In a password-based system, we can decide to let an authentic user (instance of the domain *Authentic subject*) choose a password (instance



CSR: *Security state* represents that access is granted if and only if *CredentialsAS* conforms to *Credentials*

a: *C!Credentials*
b: *AS!CredentialsAS*
c: *AM!AccessGranted, AccessDenied*
d: *FS!CredentialsFS*
e: *CredentialsTS1*
f: *CredentialsTS2*
g: *SS!AccessGranted*
h: *SS!AccessDenied*

Fig. 2. Concretized security problem frame for authentication and generic sequence diagrams for public-key-based authentication

of the domain *Credentials*), or we can decide to let an administrator (instance of the domain *Trusted subject*) choose a password. In the first case, the authentic user controls the phenomenon and in the second case, the administrator controls the phenomenon.

For the authentication problem, we assume *trusted paths* for the interfaces of the domain *Trusted subject* to prevent replay attacks. Trusted paths are confidentiality- and integrity-preserving paths. To distinguish between trusted paths and other paths, trusted paths are depicted as two parallel lines in the frame diagrams.

The security requirement is transformed into a concretized security requirement *CSR* on the basis of the assumption that the domain *Fake subject* has no *CredentialsAS*. If and only if the phenomenon *CredentialsAS* conforms to the domain *Credentials*, the considered subject is an *Authentic subject*.

5 Generic Sequence Diagrams

The generic security mechanisms introduced in the concretized security problem frame only work if certain *generic security protocols* are adhered to. Hence, the concretized security problem frame must be equipped with at least one generic security protocol. We represent such (existing) protocols by generic sequence diagrams. The generic security protocols in this section and in [7], Section 5, are intended to be the first in a more

complete collection. Once a (relatively) complete collection is defined, it helps security engineers to find suitable generic security protocols for the described security problems.

UML sequence diagrams [15] can be used to express interactions between the different domains occurring in problem diagrams. The shared phenomena are represented by messages, while the involved domains are represented by processes or objects in the sequence diagram. The security protocol and the sequence diagrams presented in this section are called *generic*, because they must be instantiated with the concrete domains and concrete messages between the domains.

The messages in the generic sequence diagrams cannot fit exactly to the symbolic phenomena of the concretized security problem frames of Section 4, because the generic security protocol descriptions require a more detailed view on the communication. The instances of a concretized security problem frame and a generic sequence diagram, however, should use the same names for the shared phenomena and the messages in the generic sequence diagram.

Generic Sequence Diagrams for Authentication

Different generic security protocols for authentication are possible, e.g., biometric protocols, passwords, or public key protocols.

The generic sequence diagrams on the right-hand side of Fig. 2 show authentication sequences using a public-key-based protocol. For reasons of simplicity, we only present one-sided authentication. The presented generic security protocol does not require a trusted connection between *Authentic subject* and *Authentication machine*, because replay attacks and man in the middle attacks respectively are excluded by using random numbers. However, a *Trusted subject* is necessary, and the *Authentic subject* must be able to create a digital signature. The *Trusted subject* must be a trusted third party that can sign the public key of *Authentic subject*. With this signature, the *Authentic subject* can be distuingished from fake subjects. For example, the *Trusted subject* could be instantiated by a *Trust center*. The *Trusted subject* also distributes its own public key to those who want to verify the subjects known by the *Trusted subject*.

The generic sequence diagram on the upper right-hand side of Fig. 2 shows the distribution of the public key (*PublicKeyTS*) of the *Trusted subject*, the public key (*PublicKeyAS*) of the *Authentic subject*, and the signature (*SignatureTS*) of the *Trusted subject*.

After distributing and signing the keys, the following authentication sequence can be performed (see lower right-hand side of Fig. 2). The *Authentic subject* sends its public key (*PublicKeyAS*) and the signature (*SignatureTS*) of the *Trusted subject* to the *Authentication machine*. The *Authentication machine* verifies the signature using the the public key (*PublicKeyTS*) of the *Trusted subject*, which was already distributed (see upper right-hand side of Fig. 2). In case of a valid signature, it sends a random number (*RandomNumber)* to the *Authentic subject*, which uses its private key to calculate a signature *RandomNumberSignatureAS* of the random number that is sent back to the machine. The machine can verify the signature using the public key (*PublicKeyAS*) of the *Authentic subject* that was sent as the first message in the authentication sequence. If the signature is valid, then access is granted (*AccessGranted*). Otherwise, access is denied (not shown in Fig. 2).

If a fake subject sends its public key without a valid signature, access will be denied. If it sends a public key of another subject with the corresponding signature, it will not be

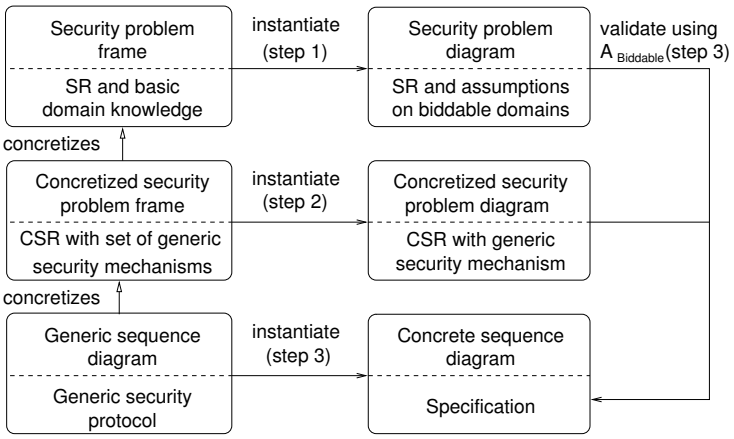


Fig. 3. Overview of our security engineering method

able to calculate the signature of the random number without having the corresponding private key.

The strength of such an authentication protocol depends on the size and quality of the random number, the used keys, and algorithms for signing and verifying the signature. RSA (Rivest, Shamir, Adleman, based on factorization of prime numbers), DSA (Digital Signature Algorithm, based on discrete logarithm) or algorithms based on elliptic curves can be used for signing and verifying (see [13], Chapter 10, page 678).

6 Method for Security Engineering using Problem Frames

In order to give concrete guidance to security engineers in using the concepts introduced so far, we propose a method to proceed from a software development problem in the area of software and system security towards a solution. The presented method constitutes a tailor-made security engineering method using security problem frames, their concretized counterparts, and generic security protocols. Figure 3 shows an overview of that method, which consists of three steps. These are presented one by one in the following. Instantiating security problem frames is the first step of our method, which is presented in Section 6.1. It is followed in the second step by instantiating concretized security problem frames introduced in Section 6.2, and it is completed in the third step presented in Section 6.3 by instantiating generic sequence diagrams and validating the specification with concrete security mechanisms. For the practical relevance of our method it is important that we also support the development of documentation for the *Common Criteria certification process* [8]. This issue is discussed in [7], Section 6.

6.1 Instantiation of Security Problem Frames (First Step)

According to our method, security engineers start their job by bounding security-critical problems, using context diagrams which show the machines to be developed and their environments. Then, the security-critical problems must be decomposed into subproblems, and these must be fitted to given security problem frames. Successfully fitting a

security problem to a given security problem frame means that the security engineer will then be guided by our method to a specification of an appropriate solution.

Instantiating the security problem frames results in security problem diagrams and textual descriptions of the assumptions and domain knowledge. In the area of software and system security, it is very important to analyze security problems using a *threat model* (see [1], Chapter 10.2). Security engineers must assume a certain level of skill, equipment, and determination that a potential attacker might have. The assumptions concerning biddable domains (e.g., instances of the domain *Fake subject*) are used to integrate threat models into this step. Threat models are important for scaling the strength of a security mechanism with the strength of a potential attacker. One method to describe an attacker is proposed in the *Common Evaluation Methodology* (see [9], Annex B.8). It gives an approach to calculate the attack potential on the basis of a function of expertise, resources, and motivation of the attacker.

Other biddable domains (e.g., *Authentic subject*) must also be described in detail. The corresponding assumptions $A_{Biddable}$ possibly constrain the generic security mechanisms and the generic security protocols to be chosen in the subsequent steps of our method. If we choose a password-based authentication mechanism, then the assumptions made for the *Authentic subject* domain can require us to use a password with, e.g., a good memorability.

6.2 Instantiation of Concretized Security Problem Frames (Second Step)

Concretized security problem frames (as presented in Section 4) constitute the basis for the second step of our method. Such a frame contains a concretized security requirement, which defines a possible generic security mechanism, e.g., asymmetric or symmetric encryption mechanisms. It does not describe the concrete security mechanism, such as DES or AES.

Instantiating the concretized security problem frames is the second step of our method. For instantiating the concretized security problem frames, the same procedure as described in Section 6.1 is applied. In addition to providing instances for domains, phenomena, and interfaces, a security engineer must choose a generic security mechanism. The security engineer must decide if, e.g., a symmetric or an asymmetric mechanism should be used, or which kind of authentication is appropriate for the given context. The domain knowledge and especially the assumptions on the biddable domains, $A_{Biddable}$, gained in the first step of our method are preserved and can completely be reused.

6.3 Instantiation of Generic Sequence Diagrams and Derivation of a Specification with Concrete Security Mechanisms (Third Step)

Instantiating the generic security mechanism and the generic security protocol represented by generic sequence diagrams is the third step of our method. Here we use the assumptions concerning the biddable domains $A_{Biddable}$ gained in the first step, the domain knowledge D (including the basic domain knowledge D_{Basic} described in Section 4, which is used to transform security requirements into concretized security requirements), the instances of the concretized security requirements CSR, and the generic security mechanisms selected in the second step to derive a specification

S. The specification S of the machine to be developed must solve the initially given security problem. It consists of a set of concrete sequence diagrams and concrete security mechanisms. This specification must be validated by demonstrating the following implication: $A_{Biddable} \wedge D \wedge S \Rightarrow CSR$, where $A_{Biddable} \wedge D \wedge S$ must be non-contradictory. The concrete security mechanisms must be chosen by a security engineer according to the following principles:

1. The concrete mechanisms must take assumptions (especially the assumptions about the biddable domains $A_{Biddable}$) and domain knowledge into account.
2. Relative to the domain knowledge and the assumptions, the concrete mechanisms must fulfill the concretized security requirement.
3. The concrete mechanisms must be available at the interface of the machine to be developed.

The procedures of instantiating a generic security protocol and instantiating a generic security mechanism must be performed in parallel. When instantiating the generic sequence diagrams, generic mechanisms like symmetric encryption must be replaced by concrete mechanisms, such as DES or AES. For password-based authentication, e.g., the minimal length of a password must be specified. After that, the instantiated sequence diagrams must be composed. To avoid composing incompatible solutions, we use the concept of expressing dependencies between the different security problem frames. (As it is beyond the scope of this paper, this issue will not be discussed further.)

7 Case Study

We illustrate our method by developing a secure remote display system, which allows its users to view and control a computing desktop environment not only on the desktop computer where it is running, but also from a *PDA* (Personal Digital Assistant) over a Bluetooth connection. After successfully establishing a connection between the PDA and the desktop computer, the desktop computer and the user must both be authenticated. In addition, any data transferred between the PDA and the desktop computer must be kept confidential and must not be modified. We now carry out the steps of our method for this problem.

7.1 Instantiating Security Problem Frames (First Step)

Figure 4 shows on the upper left-hand side the environment in which our machine must operate, expressed as a context diagram. The machine to be developed is called *Desktop, PDA, bluetooth network*. The context diagram also contains a *Malicious user* domain and a *Malicious subject* domain. They represent potential attackers, which must be described in detail. We use the method proposed in the Common Evaluation Methodology [9] for that description. The domain *Malicious user* represents an attacker who wants to make the machine believe that the domain *Malicious user* is the domain *Authentic user*. The domain *Malicious subject* represents another computer that tries to act as an authentic desktop or to intercept and modify the communication.

The Common Evaluation Methodology defines the attack potential or the strength of a potential attacker as a function of time, expertise, knowledge, and equipment. It

Factor	Identification value	Exploit value	Sum
Elapsed time (< 1 day)	2	3	5
Expertise (Proficient)	2	2	4
Knowledge of TOE (Public)	2	2	4
Access to TOE (< 1 day)	2	4	6
Equipment (Standard)	1	2	3
Sum	9	13	**22**

Table 1. Example calculation of the attack potential according to the Common Evaluation Methodology [9]

also identifies two numeric values for each of these factors. The first value is for identifying and the second one is for exploiting a *vulnerability*. In our system (named as *TOE* (Target of Evaluation)), we must consider the vulnerabilities of authentication, confidentiality, and integrity. For reasons of simplicity and instead of calculating three (possibly different) vulnerabilities, we only calculate one vulnerability and use the resulting value for all vulnerabilities to be considered. We assume that a potential attacker needs less than one day for exploiting a vulnerability, a proficient expertise of the attacker, a public known TOE, less than one day access to the TOE, and standard attack equipment. Using Table 3 in the Common Evaluation Methodology [9], we look up the corresponding numeric values for the domains *Malicious user* and *Malicious subject*, as shown in Table 1. Thus, we derive from the sum 22 of the ten values that the attack potential is rated as ”Moderate“.

After the context is described, the problem must be decomposed into subproblems, instantiating the appropriate security problem frames. The machine to be developed has to solve a *user authentication subproblem*, a *desktop authentication subproblem*, a *confidentiality subproblem for the user input*, a *confidentiality subproblem for the screen content*, an *integrity subproblem for the user input*, and an *integrity subproblem for the screen content*.

For reasons of space, we concentrate on the *desktop authentication subproblem*. The diagrams for the other subproblems are given in [7].

The security problem diagram for desktop authentication is shown on the right-hand side of Fig. 4. We instantiate the authentication frame (see Fig. 1) using the domain instances *Desktop*, *Desktop auth machine*, *PDA security state*, and *Malicious Subject*. These domain instances are also used to instantiate the security requirement SR.

7.2 Instantiating Concretized Security Problem Frames (Second Step)

In this step, we must choose appropriate generic security mechanisms, and we must instantiate the corresponding concretized security problem frames.

For the desktop authentication, we choose a public-key-based mechanism. Therefore, we must instantiate the concretized security problem frame for authentication (see left-hand side of Fig. 2). We reuse the domain instances introduced in the first step of our method, and we must additionally introduce the domain instances *Trust Center* and *Public key of trust center*. For the instantiation of the *CSR*, we assume that the *Mali-*

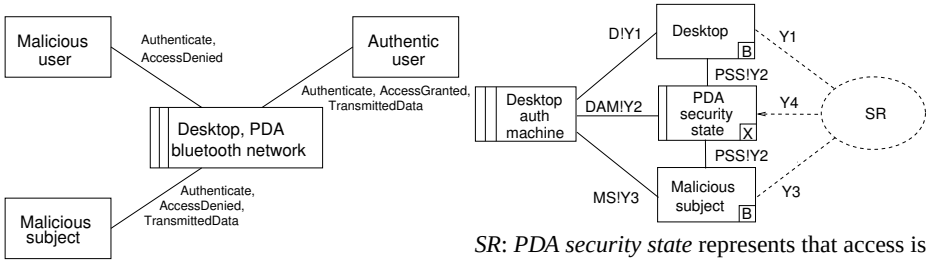


Fig. 4. Context diagram of a secure remote display system and security problem diagram for desktop authentication

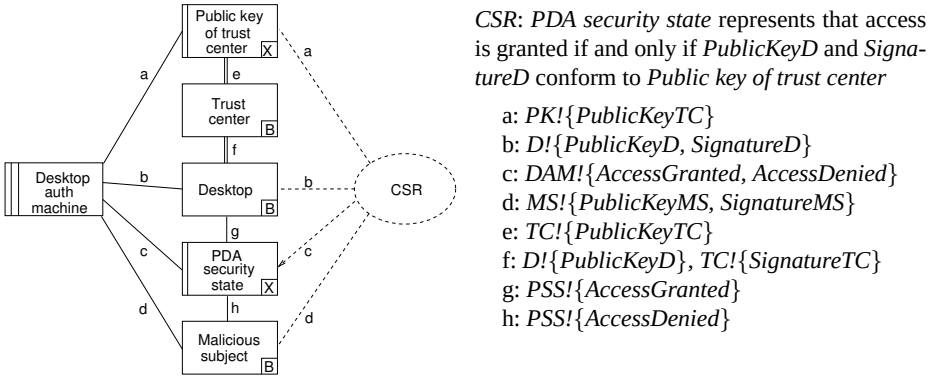


Fig. 5. Concretized security problem diagram for desktop authentication

cious subject cannot send *SignatureD*. Instead, it sends a *SignatureMS*. The phenomena *SignatureD*, *SignatureMS*, and *SignatureTC* represent the usage of the private keys for creating digital signatures. The usage of the public keys is represented by the domain *Public key of trust center* and the phenomena *PublicKeyD*, *PublicKeyMS*, and *PublicKeyTC*. Figure 5 shows on the left-hand side the concretized security problem diagram for desktop authentication.

7.3 Instantiating Generic Sequence Diagrams and Deriving a Specification with Concrete Security Mechanisms (Third Step)

In the third step of our method, we must instantiate generic security protocols (see Section 5) and the generic security mechanisms chosen in the second step of our method. The concrete security mechanisms should be selected in such a way that the CSR is fulfilled using the assumptions concerning the biddable domains $A_{Biddable}$.

We calculated in the first step of our method (see Section 7.1) that the assumed attack potential of the *Malicious subject* is rated as ”Moderate“. The assumptions on

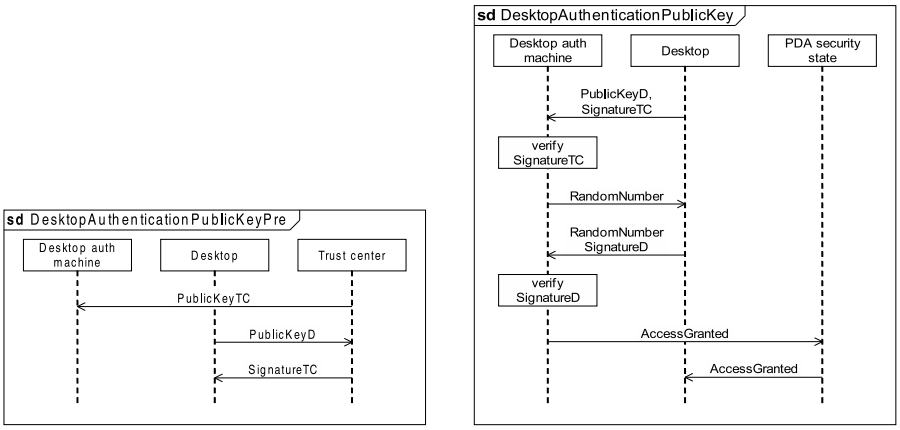


Fig. 6. Instantiated sequence diagrams for public-key-based desktop authentication

the biddable domains $A_{Biddable}$ are represented by this attack potential of the domain *Malicious subject*, whereas the assumptions on the other biddable domains *Trust center* and *Desktop* are neglected for reasons of simplicity. In the second step, we chose a public-key-based mechanism for the desktop authentication. For the *Desktop auth machine*, we conclude in this step that the public-key-based mechanism RSA (see [14], Chapter 4.3 for details) with 768 bits is appropriate according to the assumptions concerning the biddable domains $A_{Biddable}$. Furthermore, we use the generic sequence diagrams *AuthenticationPublicKeyPre* and *AuthenticationPublicKey* on the right-hand side of Fig. 2 as patterns. The domains in the generic sequence diagrams are instantiated as described in Section 7.2. The instantiated sequence diagrams for public-key-based desktop authentication are shown in Fig. 6.

It is improbable that the *Malicious subject* can guess or calculate *SignatureD* and *RandomNumberSignatureD*. RSA is based on factorization of prime numbers. Until now, nobody succeeded in factorizing a prime number with a length greater than 576 bit (see [16] for details). Therefore, the 768 bit RSA used in our example is secure (not only for attackers of moderate strength) until further notice. Thus, we have demonstrated that the specification S (defined by the instantiated sequence diagrams and the 768 bit RSA) suffices to fulfill the concretized security requirement CSR: $A_{Biddable} \wedge D \wedge S \Rightarrow CSR$.

Hence, our original problem will be solved if the derived specification is correctly implemented.

8 Related Work

In the first step of our method, security requirements must be expressed, using a threat model. Lin et al. [12] take a different approach to use the ideas underlying problem frames in security. They define so-called anti-requirements and the corresponding *abuse frames*. An anti-requirement expresses the intentions of a malicious user, and an abuse frame represents a security threat. 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.

Seperating security problem frames and concretized security problem frames enhances the so-called security frames introduced in [6]. We now carefully distinguish the problem description using security problem frames and the preparation of a solution using concretized security problem frames.

Security patterns [3] 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. Moreover, design and security patterns are applicable only in an object-oriented setting, while problem frames and our security problem frames are independent of a particular programming paradigm.

9 Conclusions and Future Work

In this paper, we have presented new kinds of problem frames tailored for representing security problems, called security problem frames and concretized security problem frames. They are patterns for software development problems occurring frequently when security-critical software has to be developed.

Security problem frames consider security requirements in order to increase the potential for reuse by carefully distinguishing security problems from their solutions. The security requirements are stated as patterns to be instantiated. In transforming security requirements into concretized security requirements, new concretized security problem frames evolve from the security problem frames. The concretized security problem frames introduce generic security mechanisms, which only work if certain generic security protocols are adhered to. Hence, we equip each of the concretized security problem frames with such generic security protocols, described by generic sequence diagrams. The instances of generic security protocols are *solutions* to the initially given security problems.

Both kinds of security problem frames and the generic security protocols presented in this paper are intended to be the first in a more complete collection. Once a (relatively) complete collection is defined, it is of considerable help for security engineers. For a new security-critical system to be constructed, the catalogue can be inspected in order to find the frames and protocols that apply for the given problem. Thus, such a catalogue helps to avoid omissions and to cover all security aspects that are relevant for the given problem.

While the frames themselves “only” help to comprehend, locate and represent problems, our method supports security engineers to *solve* the problems fitted to security problem frames step-by-step. The instantiation of the security problem frames is the first step. Here, we gain important information about the problem environment. Moreover, a threat model is defined about the assumed capabilities on potential attackers. The method proceeds in the second step with the instantiation of concretized security problem frames. In this step, the principles of the envisaged solution are fixed. The third step consists of selecting concrete security mechanisms and deriving a specification of the security-critical system on the basis of instantiated generic security protocols represented by sequence diagrams.

With the concept of security problem frames and the associated method based on concretized security problem frames and generic sequence diagrams (in addition to security patterns), security engineers can hope to cover large parts of the development of security-critical systems with a pattern-based approach.

In the future, we intend to extend this work by formalizing assumptions, domain knowledge, and requirements. Second, the compositionality of the security problem frames will be considered in more detail, by performing interaction analyses. Third, we intend 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.

References

[1] R. Anderson. *Security Engineering*. Wiley, 2001.

[2] L. Bass, P. Clements, and R. Kazman. *Software Architecture in Practice*. Addison-Wesley, 1998.

[3] B. Blakley and C. Heath. *Technical Guide: Security Design Patterns*. The Open Group, April 2004. `http://www.opengroup.org/publications/catalog/g031.htm`.

[4] F. Buschmann, R. Meunier, H. Rohnert, P. Sommerlad, and M. Stal. *Pattern-Oriented Software Architecture: A System of Patterns*. John Wiley & Sons, 1996.

[5] E. Gamma, R. Helm, R. Johnson, and J. Vlissides. *Design Patterns – Elements of Reusable Object-Oriented Software*. Addison Wesley, Reading, 1995.

[6] D. Hatebur and M. Heisel. Problem frames and architectures for security problems. In B. A. Gran, R. Winter, and G. Dahll, editors, *Proceedings of the 24th International Conference on Computer Safety, Reliability and Security (SAFECOMP)*, LNCS 3688, pages 390–404. Springer-Verlag, 2005.

[7] D. Hatebur, M. Heisel, and H. Schmidt. Using problem frames for security engineering. Technical report, Universität Duisburg-Essen, 2006. `http://swe.uni-duisburg-essen.de/intern/seceng06.pdf`.

[8] International Organization for Standardization (ISO) and International Electrotechnical Commission (IEC). Common criteria 2.3. ISO/IEC 15408, 2005. `http://www.commoncriteriaportal.org`.

[9] International Organization for Standardization (ISO) and International Electrotechnical Commission (IEC). Common evaluation methodology 2.3. ISO/IEC 18405, 2005. `http://www.commoncriteriaportal.org`.

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

[11] 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.

[12] L. Lin, B. Nuseibeh, D. Ince, M. Jackson, and J. Moffett. Introducing abuse frames for analysing security requirements. In *Proceedings of 11th IEEE International Requirements Engineering Conference (RE'03)*, pages 371–372, 2003. Poster Paper.

[13] C. P. Pfleeger. *Security in Computing*. Prentice Hall, third edition, 2003.

[14] G. Schäfer. *Security in Fixed and Wireless Networks*. John Wiley & Sons, Ltd, Chichester, 2003.

[15] UML Revision Task Force. *OMG Unified Modeling Language: Superstructure*, August 2005. `http://www.uml.org`.

[16] E. W. Weisstein. RSA-576 factored. *MathWorld Headline News*, 2003. `http://mathworld.wolfram.com/news/2003-12-05/rsa/`.