A Pattern System for Security Requirements Engineering

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

[1] University Duisburg-Essen, Faculty of Engineering, Department of Computer Science, Workgroup Software Engineering, Germany, email: {maritta.heisel, denis.hatebur, holger.schmidt}@uni-duisburg-essen.de

[2] ITESYS Institut für technische Systeme GmbH, Germany, email: d.hatebur@itesys.de

Abstract

We present a pattern system *for security requirements engineering, consisting of* security problem frames *and* concretized security problem frames*. These are 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. We equip each frame with formal preconditions and postconditions. The analysis of these conditions results in a pattern system that explicitly shows the* dependencies *between the different frames. Moreover, we indicate related frames, which are commonly used together with the considered frame. Hence, our approach helps security engineers to avoid omissions and to cover all* security requirements *that are relevant for a given problem.*

1 Introduction

When building secure systems, it is instrumental to take security concerns into account right from the beginning of the development process. As for functional requirements, a detailed analysis of the security requirements of an envisaged system should be performed. Those security requirements have to be taken into account in all subsequent phases of the system and software development lifecycle. Hence, security engineering should become an integral part of the software engineering process when developing security-critical software systems.

In this paper, we show how such an integration can be achieved for the requirements analysis phase. The basic idea is to define patterns 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, we name our patterns *security problem frames*. They serve to analyze security-related requirements.

Typical security problems concern, for example, confidential data transmission, authentication, or the distribution of secrets. For several such problem classes, we define dedicated security problem frames, which contain a structuring of the environment where the problem arises (including possible attackers), and a schematic expression of the security requirement that can be addressed with the given frame.

To use security problem frames, the variable parts of the patterns are instantiated. Moreover, for each security problem frame, we define several *concretized security problem frames* that take into account generic mechanisms to solve the given problem, for example, to use credentials for authentication. The concretized security problem frames, in turn, are associated with generic security protocols, which capture proven approaches to implement security mechanisms [4]. Thus, security problem frames not only support security engineers in analyzing and structuring security requirements but also lead the way for realizing them.

We equip each (concretized) security problem frame with *preconditions* characterizing the conditions under which it is applicable, and *postconditions* describing the security requirement to be addressed. When applying a (concretized) security problem frame, one must check the corresponding preconditions. If they cannot be assumed to be fulfilled, this means that additional security problems must be solved.

We present a pattern system based on security problem frames and their concretized counterparts. This pattern system helps security engineers to systematically *identify* and *analyze* all the subproblems that must be solved in order to solve a complex security problem. It helps to structure the development process and to avoid confusion and omissions.

We construct the pattern system by analyzing the pre- and postconditions of the different frames, and explicitly identifying *dependencies* between them. To make one frame applicable, another frame should be applied. Thus, subproblems, that are necessary to be considered when solving a given problem, are generated along the dependency relations. The process of generating new subproblems terminates when all preconditions of all applied security problem frames can be proved or assumed to hold.

The contribution of our paper is a formally grounded pattern system for security requirements engineering, which is based on security problem frames and their concretized counterparts. Security engineers can use this pattern system to recognize, structure, characterize, and analyze software

development problems in the area of software and system security. A thorough problem and requirements analysis is crucial for developing adequate solutions to security problems.

In the following, we first present Jackson's problem frames in Sect. 2. In Sect. 3, we introduce security problem frames and in Sect. 4, we present concretized security problem frames. We construct and describe the pattern system in Sect. 5. We illustrate the usage of the pattern system by a secure remote display system in Sect. 6. Section 7 discusses related work, and we conclude in Sect. 8.

2 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 and 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 *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*. 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. For example, if a user types a password to log into an IT-system, this is a phenomenon shared by the user and the system, which is controlled by the user. 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.

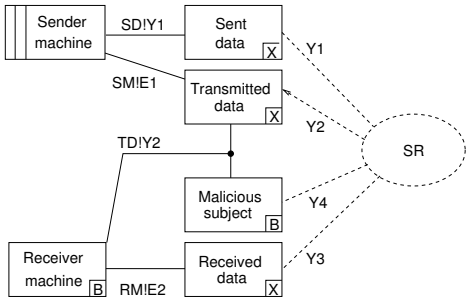
In the frame diagram depicted in Fig. 1, the "X" indicates that the corresponding domain is a lexical domain, and the "B" indicates a biddable domain. The notation "SD!Y1" means that the phenomenon of interface *Y1* is controlled by the lexical domain *Sent data*.

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

SPF: Confidential Data Transmission

Frame diagram



Y1 := {*ContentOfSentData*}
Y2 := {*ContentOfTransmittedData*}
Y3 := {*ContentOfReceivedData*}
Y4 := {*PropertiesOfMaliciousSubject*}
E1 := {*SetTransmittedData*}
E2 := {*SetReceivedData*}

Security requirement (SR)

Malicious subject should not be able to derive *Sent data* and *Received data* using *Transmitted data*.

Declarations

$y1 : Y1;\ e2 : E2;\ sd : SentData;\ rd : ReceivedData$
$td : TransmittedData;\ ms : MaliciousSubject$

Preconditions

$\{(ms, sd), (ms, rd)\} \cap known = \varnothing$
$\{y1, e2\} \subseteq confIf$

Postconditions

$\{(ms, sd), (ms, rd)\} \cap known = \varnothing$
$\{(ms, sd, td), (ms, rd, td)\} \subseteq conf$

Related

SPF: Integrity-preserving Data Transmission
SPF: Confidential Data Transmission

Figure 1. Security problem frame for confidential data transmission

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 user input. Facts describe fixed properties of the problem environment 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: $A \wedge F \wedge S \Rightarrow R$, where $A \wedge F \wedge S$ 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 Fig. 8). 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., instantiate 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 [7]). The specification is the starting point for the development of the machine.

3 Security Problem Frames

To meet the special demands of software development problems occurring in the area of security engineering, we introduced security problem frames [4]. Security problem frames are special kinds of problem frames, which consider *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 Sect. 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 enhances the re-usability of the problem descriptions, since they are completely independent of solution technologies.

3.1 Describing Security Problem Frames

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 frame. It is also specified if it is a security problem frame (SPF) or a concretized security problem frame (CSPF).
- **Frame diagram** This diagram shows the relevant domains and their interfaces, as well as the (concretized) security requirement.
- **Security requirement or concretized security requirement** Here, the security requirement or concretized security requirement to be achieved is stated informally.
- **Declarations** In this section, entities that are necessary for stating the preconditions and postconditions are declared. The 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, i.e., they describe what (concretized) security requirement will be achieved by the machine to be built.
- **Related** Different patterns will often be used in combination. Those frames that are commonly used in combination with the described frame are mentioned here.

We have formalized problem frames as well as (concretized) security problem frames using the object-oriented formal specification language Object-Z. The formalization is not described in this paper. The pre- and postconditions are expressed on the basis of this formalization as logical formulas, and they form the basis for the pattern system described in Sect. 5.

In this section, we present three security problem frames that capture frequently occurring security problems. We have defined further security problem frames, e.g., for establishing integrity-preserving data transmissions [5], which are not presented in this paper.

We discuss the security problem frame for confidential data transmission using the template presented in this section. For reasons of space, we present the rest of the (concretized) security problem frames without their template-based illustration. Instead, we describe the frames using the corresponding frame diagrams.

3.2 Security Problem Frame for Confidential Data Transmission

Many security-critical systems are required to keep data confidential during its transmission. Confidential data transmission means restricting access to transmitted data to those who are privileged to access it.

Figure 1 shows the security problem frame for confidential data transmission. The domain *Sent data* denotes the data that is sent by a sender, represented by the machine domain *Sender machine*. Analogously, the domain *Received data* denotes the data that is received by the domain *Receiver machine*. During its transmission, the data is represented by the domain *Transmitted data*.

Informally speaking, the sender machine generates the transmitted data from the sent data, and the receiver machine generates the received data from the transmitted data.

Thus, the operation *SetTransmittedData* of interface *E1* is controlled by the *Sender machine* domain, and it represents an operation that generates the transmitted data. The operation *SetReceivedData* of interface *E2* represents a similar operation on the domain *Received data*, but it is controlled by the *Receiver machine* domain. The symbolic phenomena of the interfaces *Y1*, *Y2*, and *Y3* represent some sent, transmitted, or received data values. The symbolic phenomenon of the interface *Y1* is controlled by the domain *Sent data*. The symbolic phenomenon of the interface *Y2* between the domain *Transmitted data* and the domains *Receiver machine* and *Malicious subject* is controlled by the domain *Transmitted data*. The symbolic phenomenon of the interface *Y4* is controlled by the *Malicious subject* domain, and it reflects relevant properties of the *Malicious subject* domain, e.g., details about its equipment and strength.

The precondition $\{(ms, sd), (ms, rd)\} \cap \mathit{known} = \varnothing$ of the security problem frame for confidential data transmission expresses that a malicious subject ms does not know sent data sd and received data rd beforehand.

The precondition $\{y1, e2\} \subseteq \mathit{confIf}$ expresses that confidential paths $y1$ and $e2$ between the machine domain *Sender machine* and the domain *Sent data* and between the domains *Receiver machine* and *Received data* are necessary. It describes that data transferred using the interfaces $y1$ and $e2$ is kept confidential.

In fact, the precondition $\{(ms, sd), (ms, rd)\} \cap \mathit{known} = \varnothing$ is also a postcondition, because a malicious subject ms should not know sent data sd and received data rd after the transmission. The postcondition $\{(ms, sd, td),\ (ms, rd, td)\} \subseteq \mathit{conf}$ expresses that a malicious subject ms should not be able to derive the sent data sd as well as the received data rd using the transmitted data td.

The list of related frames contains the frame for integrity-preserving data transmission [5], because in many cases both security requirements are of interest when considering data transmissions. In addition, the list contains the frame itself, because often the feedback to a confidential data transmission should be kept confidential.

3.3 Security Problem Frame for Authentication

Authentication of users and other systems is an important issue for many security-critical systems. Authentication is the problem to verify a claimed identity.

Figure 2 shows the frame diagram of the security problem frame for authentication. The domain *Authentic subject* represents an authentic user or an authentic system. In contrast, the domain *Fake subject* represents a fake user or a fake system. The domain *Authentication state* represents that the *Authentic subject* domain should be authenticated and the *Fake subject* domain should be not authenticated. The *Authentication state* domain is externally visible, because the subjects are at least implicitly informed of their authentication status. The security requirement *SR* is stated

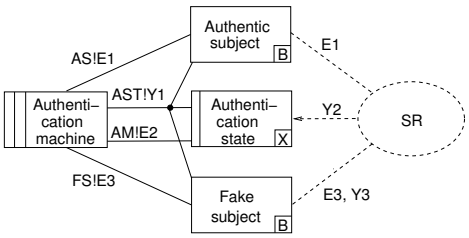


Figure 2. Frame diagram of the security problem frame for authentication

according to this description.

The event *Authenticate*$_{AS}$ of interface *E1* is controlled by the *Authentic subject* domain, and it represents some authentication command. The event *Authenticate*$_{FS}$ of interface *E3* represents a similar command, but it is controlled by the *Fake subject* domain. The operation *SetAuthentication-Status* of interface *E2* is controlled by the *Authentication machine* domain, and it is an operation on the authentication state. The symbolic phenomenon *ContentOfAuthenticationState* of interface *Y1* is controlled by the *Authentication state* domain, and allows the environment to examine the current authentication status. The symbolic phenomena *Authenticated* and *NotAuthenticated* of *Y2* represent the current authentication status, and the symbolic phenomenon *PropertiesOfFakeSubject* of *Y3* reflects relevant properties of the *Fake subject* domain, as in the security problem frame for confidential data transmission (Sect. 3.2).

The precondition of the security problem frame for authentication shown in Fig. 2 is $as \neq fs$, meaning that an authentic subject as must not be a fake subject fs at the same time.

The requirement that an authentic subject as is authentic for an authentication machine am, whereas a fake subject fs is not authentic for an authentication machine am is then expressed using the postconditions $(am, as) \in \mathit{authenticated}$ and $(am, fs) \notin \mathit{authenticated}$.

The list of related frames contains the frame itself, because the security engineer should take authentication in the opposite direction into account.

3.4 Security Problem Frame for Distributing Secrets

In order to apply certain security mechanisms such as encryption, the distribution of secrets is necessary. It is the problem to communicate *matching* secrets to those subjects who are privileged to receive them.

Figure 3 shows the frame diagram of the security problem frame for distributing secrets. It is similar to the frame diagram depicted in Fig. 1. However, this frame focuses on matching of the secrets and requires a trusted path. To express this security requirement, the domain *Secret*$_2$ is constrained, and the control directions of the interfaces *Y1* and

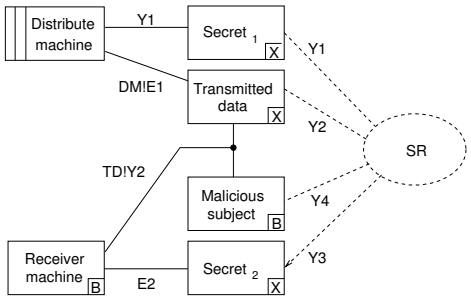


Figure 3. Frame diagram of the security problem frame for distributing secrets

E2 are undefined. The domains *Secret*$_1$ and *Secret*$_2$ represent secrets. The domain *Secret*$_1$ is known to the machine domain *Distribute machine*, whereas the domain *Secret*$_2$ is known to the domain *Receiver machine*. During transmission, the domain *Secret*$_1$ is represented by the domain *Transmitted data*, which can be observed by the domain *Malicious subject*.

The operation *SetTransmittedData* of interface *E1* is controlled by the *Distribute machine* domain, and it represents an operation that generates the transmitted data. The operation *SetSecret*$_2$ of interface *E2* represents a similar operation on the domain *Secret*$_2$. The symbolic phenomenon *ContentOfSecret*$_1$ of the interface *Y1* and the symbolic phenomenon *ContentOfSecret*$_2$ of the requirement reference *Y3* represent some sent and received secret data values. The symbolic phenomenon *ContentOfTransmittedData* of the interface *Y2* represents some transmitted data values. The symbolic phenomenon *PropertiesOfMaliciousSubject* of *Y4* reflects relevant properties of the *Malicious subject* domain. The frame diagram does not define which domains choose the secrets. As a consequence, the control directions of the interfaces *Y1* and *E2* are not defined. If the machine domain *Distribute machine* chooses the secrets, then it also controls the interface *Y1*, and the interface *E2* is controlled by the domain *Secret*$_2$. The security requirements constrains the domain *Secret*$_1$. In the case of negotiated secrets, both secrets are constrained by the security requirement, the machine domain *Distribute machine* controls the interface *Y1*, and the domain *Receiver machine* controls the interface *E2*.

The first precondition is $\{(ms, s_1), (ms, s_2)\} \cap known = \varnothing$, and it expresses that the secrets s_1 and s_2 are not known to a malicious subject ms.

The second precondition is $\{y1, e2\} \subseteq trustedIf$, and it expresses that trusted paths $y1$ and $e2$ between the machine domain *Distribute machine* and the domain *Secret*$_1$ as well as between the *Receiver machine* domain and the domain *Secret*$_2$ are necessary. A trusted path is a confidential and an integrity-preserving path. An integrity-preserving path transfers data, which is unchanged or a change is detected (see [5] for details).

The first two postconditions are $\{(dm, s_1), (rm, s_2)\} \subseteq known$ and $\{(ms, s_1), (ms, s_2)\} \cap known = \varnothing$, and they express that the secrets s_1 and s_2 are known to the distribute machine dm and the receiver machine rm, respectively, but that none of the secrets are known to a malicious subject ms. The third postcondition is $(s_1, s_2) \in match$, and it expresses that the secrets s_1 and s_2 match, in the sense of e.g., matching passwords or matching private key and public key of a key pair.

4 Concretized Security Problem Frames

In this section, we present concretized security problem frames for dynamic authentication, confidential data transmission using symmetric encryption, and distribution of secrets based on negotiation and trusted paths. These concretized security problem frames are derived from the security problem frames presented in Sect. 3 by considering generic security mechanisms (such as confidential data transmission using symmetric encryption), thereby transforming the security requirements into concretized security requirements. More information about concretized security problem frames and the transformation process can be found in [4].

In general, the pre- and postconditions of the security problem frames are preserved, and therefore contained in the pre- and postconditions of the concretized security problem frames.

4.1 Concretized Security Problem Frame for Confidential Data Transmission using Symmetric Encryption

One of the concretized security problem frame for confidential data transmission considers symmetric encryption. Its frame diagram is shown in Fig. 4. In transforming the security requirement for confidential data transmission into a concretized security requirement *CSR*, the domains *Secret*$_1$ and *Secret*$_2$ are introduced for the encryption mechanism.

Compared to the interfaces of the security problem frame for confidential data transmission discussed in Sect. 3.2, we preserved all interfaces, and we added the interfaces *Y5* and *Y6*. The symbolic phenomena *ContentOfSecret*$_1$ and *ContentOfSecret*$_2$ of the interfaces *Y5* and *Y6* are controlled by the domain *Secret*$_1$ and the domain *Secret*$_2$, respectively. These symbolic phenomena represent the values of the secrets.

The first two preconditions are preserved from the security problem frame for confidential data transmission shown in Fig. 1. The precondition $\{(sm, s_1), (rm, s_2)\} \subseteq known$ states that the domain *Secret*$_1$ represented by s_1 must be known by the sender machine sm and the domain *Secret*$_2$ represented by s_2 must be known by the receiver machine rm. Moreover, the precondition $\{(ms, s_1), (ms, s_2)\} \cap known = \varnothing$ specifies that the malicious subject ms does not know these secrets. The precondition $(s_1, s_2) \in match$ describes that the two secrets match. In the case of symmetric encryption, the relation $match$ is the equality relation.

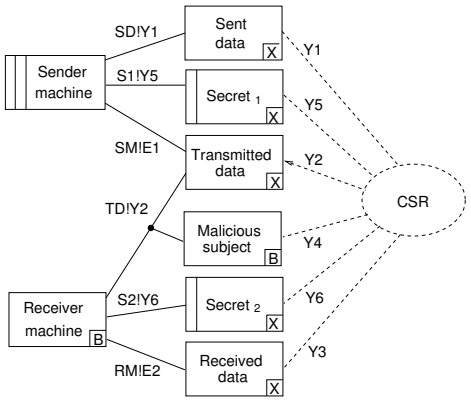


Figure 4. Frame diagram of the concretized security problem frame for confidential data transmission using symmetric encryption

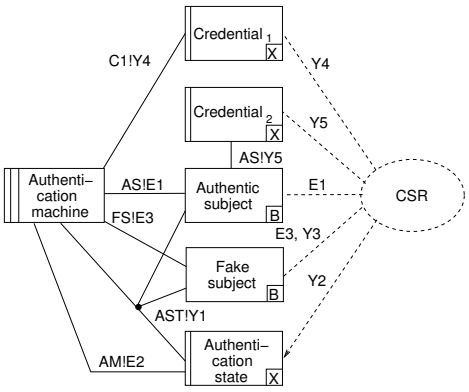


Figure 5. Frame diagram of the concretized security problem frame for dynamic authentication

Moreover, trusted paths $y5$ and $y6$ between the domains *Sender machine* and *Secret*$_1$ are necessary, as well as between the domains *Receiver machine* and *Secret*$_2$. This is expressed using the predicate $\{y5, y6\} \subseteq trustedIf$.

The postcondition is the same as the postcondition of the security problem frame for confidential data transmission presented in Sect. 3.2.

4.2 Concretized Security Problem Frame for Dynamic Authentication

One of the concretized security problem frames for authentication considers dynamic mechanisms. Its frame diagram is shown in Fig. 5. In transforming the security requirement for authentication into a concretized security requirement *CSR*, two designed domains *Credential*$_1$ and *Credential*$_2$ are introduced, which make it possible to distinguish between the domains *Authentic subject* and *Fake subject*.

Compared to the interfaces of the security problem frame for authentication discussed in Sect. 3.3, we preserved all interfaces except for the interfaces *E1* and *E3*. The phenomenon *Authenticate*$_{AS}$*(*a_{AS}*)* of interface *E1* is an authentication operation, where the parameter a_{AS} represents some authentication data of the authentic subject. The phenomenon *Authenticate*$_{FS}$*(*a_{FS}*)* of interface *E3* is also an authentication operation, where the parameter a_{FS} represents some authentication data of the fake subject. We added the interfaces *Y4* and *Y5*. The symbolic phenomenon *ContentOfCredential*$_1$ of interface *Y4* is controlled by the domain *Credential*$_1$, whereas the symbolic phenomenon *ContentOfCredential*$_2$ of interface *Y5* is controlled by the domain *Authentic subject*.

The first precondition is preserved from the security problem frame for authentication discussed in Sect. 3.3. The second precondition $\{(am, c_1), (as, c_2)\} \subseteq known$ expresses that the domain *Credential*$_1$ represented by c_1 must be known by the authentication machine am and the domain *Credential*$_2$ represented by c_2 must be known by the authentic subject as. The third precondition expresses that c_1 is not known by the fake subject *fs* domain. The fourth precondition states that c_1 and c_2 match.

The postcondition is stated similarly to the postconditions of the security problem frame for authentication presented in Sect. 3.3, with the difference that the predicates $(c_1, a_{AS}) \in match$ and $(c_1, a_{FS}) \notin match$ require that the authentication data a_{AS} match the c_1, while the fake authentication data a_{FS} does not match c_1 before the authentication is established.

4.3 Concretized Security Problem Frames for Distributing Secrets

In this section, we introduce two concretized security problem frames for distributing secrets, one using negotiation and another one using trusted paths. The frame diagram of the former frame is shown in Fig. 6.

Compared to the interfaces of the security problem frame for distributing secrets discussed in Sect. 3.4, we preserved all interfaces except for the interfaces *E1* and *Y2*. The phenomenon of interface *E1* is renamed to *SetNegotiationData*$_{DM}$, and the phenomenon of interface *Y2* is renamed to *NegotiationData*$_{DM}$. We added the interface *Y5*. It contains a symbolic phenomenon named *NegotiationData*$_{RM}$. We added the interface *E3*. It contains an operation named *SetNegotiationData*$_{RM}$. The symbolic phenomena of the interfaces *Y2* and *Y5* represent the different negotiation data of the receiver machine and the distribute machine. The operation of the interface *E1*

determines the phenomena visible via the interface *Y2*. The operation of the interface *E3* determines the phenomena visible via the interface *Y5*. Additionally, we renamed the domain *Transmitted data* to *Negotiation Data*.

The first two preconditions are preserved from the security problem frame for distributing secrets shown in Fig. 3.

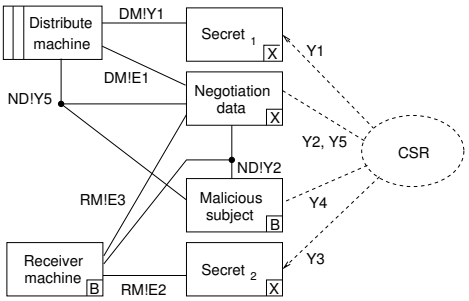


Figure 6. Frame diagram of the concretized security problem frame for distributing secrets using negotiation

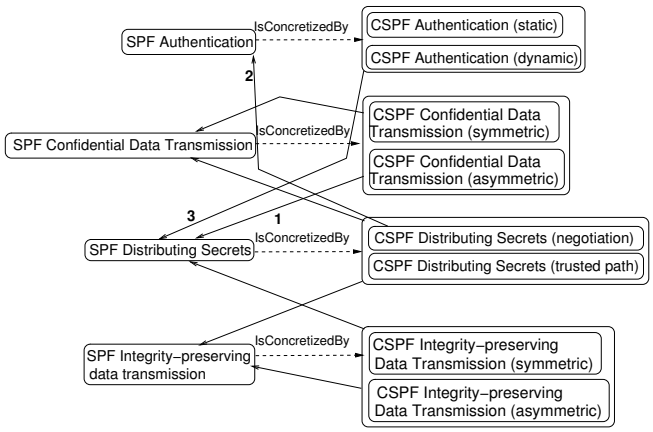


Figure 7. Pattern system of (concretized) security problem frames

Asymmetric mechanisms ensure that the secrets cannot be derived by a malicious subject using the negotiation data. Nevertheless, authentication of the receiver is required to prevent a man in the middle attack. Therefore, the other preconditions $(dm, rm) \in authenticated$ and $(dm, ms) \notin authenticated$ require to take an authentication subproblem between distribute machine dm and receiver machine rm into consideration.

The concretized security problem frame for distributing secrets using trusted paths abstains from such an authentication subproblem. Instead, it requires a trusted path. Thus, its preconditions contain the predicate $y2 \in trustedIf$, which describes a trusted path $y2$ between the machine domain *Distribute machine* and the domain *Receiver machine*.

The postconditions of both frames are the same as the postconditions of the security problem frame for distributing secrets presented in Sect. 3.4.

5 A Pattern System for Security Engineering

In this section, we explicitly represent the dependencies and relations of the frames in detail, yielding a pattern system, which is shown in Fig. 7.

The pattern system is constructed by analyzing the preconditions and postconditions of the different security problem frames and their concretized counterparts. We check the preconditions of a concretized security problem frame and then, we syntactically match them with the postconditions of at least one security problem frame. For example, the preconditions of both concretized security problem frames for distributing secrets presented in Sect. 4.3 contain formulas that describe trusted paths. Therefore, confidential and integrity-preserving paths are necessary. The postconditions of the security problem frame for confidential data transmission presented in Sect. 3.2 contain a formula that describes a confidential path. For this reason, the frames for distributing secrets depend on the frame for confidential data transmission. Hence, we draw a line from the outer box containing the *CSPF Distributing Secrets (negotiation)* and *CSPF Distributing Secrets (trusted path)* with an arrow pointing at the box *SPF Confidential Data Transmission*. All dependencies in Fig. 7 are established following this principle.

Besides the frames introduced in Sections 3 and 4, Fig. 7 shows the concretized security problem frames for static authentication, confidential data transmission using asymmetric encryption, and the security problem frame for integrity-preserving data transmission including its concretized counterparts. The (concretized) security problem frames depicted in Fig. 7 form a self-contained pattern system: 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.

Our security requirements engineering process proceeds as follows: when developing a secure system, a security engineer starts with the elicitation of an initial set of security requirements using, e.g., the CREE method [2]. Then, each elicited security requirement must be compared to the informal descriptions of the security requirements of the security problem frames. After appropriate security problem frames are identified 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.

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.

The process continues with choosing appropriate concretized security problem frames. After the concretized

counterparts have been chosen, the newly introduced domains, phenomena, interfaces, pre- and postconditions, and the concretized security requirement must be instantiated.

Afterwards, the preconditions of the instantiated concretized security problem frames and the pattern system must be inspected. To guarantee that the preconditions hold, two alternatives are possible: either, they are *assumed* to hold, or they have to be *established* by using another security problem frame whose postconditions match the preconditions to be established. Such a frame can be determined using the pattern system shown in Fig. 7 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.

What assumptions are reasonable depends on the threats the system should be protected against. Moreover, some assumptions cannot be avoided, because otherwise, a certain (concretized) security requirement cannot be achieved. 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.

After the preconditions of the instantiated concretized security problem frames are inspected, often additional security problems must be considered. As a consequence, one must instantiate appropriate security problem frames. Again, a security engineer proceeds with choosing and instantiating appropriate concretized security problem frames. Again, the preconditions of the instantiated concretized security problem frames and the pattern system must be inspected, and identified dependencies must be resolved by assuming that the preconditions are fulfilled or by considering additional security problems. As a consequence of the latter case, one must instantiate appropriate security problem frames. The process of generating new subproblems terminates when all preconditions of all applied concretized security problem frames can be proved or assumed to hold. Therefore, the security requirements engineering process will result in a *complete* set of security problems (and solution approaches), some of which may not have been known initially.

The explicit knowledge of the dependencies between the security problem frames and their concretized counterparts increases the value of our approach. The guidance provided by the dependency relations of the pattern system helps to structure the security requirements process, to avoid confusion, and to analyze security problems and their solution approaches in depth.

6 Case Study

We illustrate the usage of our pattern system by a secure remote display system, which allows its users to view and control a computing desktop environment not only on the *PC* (Personal 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 PC, any data transferred between the PDA and the PC must be kept confidential.

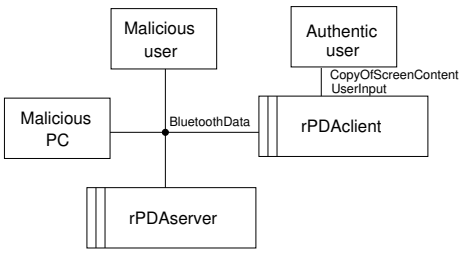


Figure 8. Context diagram of a secure remote display system

Figure 8 shows the environment in which our machine must operate, expressed as a context diagram. The machines to be developed are called *rPDAserver* and *rPDAclient*. The context diagram also contains an *Authentic user* domain, a *Malicious user* domain, and a *Malicious PC* domain. The first domain represents the authentic PDA user, while the latter two domains represent attackers who want to spy the Bluetooth data transmission in order to obtain the user input or the screen content, respectively.

After the context is described, the problem must be decomposed into subproblems, fitting to appropriate security problem frames. The machines to be developed have to solve a *confidentiality subproblem for the user input* and a *confidentiality subproblem for the screen content*. For reasons of space, we concentrate on the *confidentiality subproblem for the screen content*, and we only depict one instantiated frame diagram (see Fig. 9). Instead of showing the other instantiated frames, we present the instantiated pre- and postconditions. In combination with the pattern system, they guide us through the security requirements engineering process.

To instantiate the security problem frame for confidential data transmission (see Sect. 3.2), some domains are factored out of the machine domain *rPDAserver*. The domain *Screen content* is an instance of the domain *Sent data*, *Bluetooth data* is an instance of the domain *Transmitted data*, and *Copy of screen content* is an instance of the domain *Received data*.

After having chosen an appropriate security problem frame, we proceed by selecting a security mechanism suitable to tackle the problem. This corresponds to choosing one of the concretized security problem frames associated with the selected security problem frame. Because of the probably high data exchange between the PDA and the PC, we decide to choose a symmetric encryption mechanism for the screen content confidentiality subproblem. The decryption of the screen content is intentionally left out, because only encryption is needed to ensure confidentiality. Decryption is often used to ensure integrity.

Figure 9 shows the problem diagram for confidential

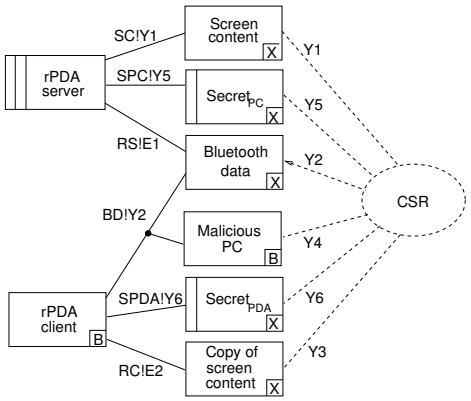


Figure 9. Problem diagram for confidential screen content transmission using symmetric encryption

screen content transmission using symmetric encryption. We only have to instantiate the new domain *Secret*$_1$ by *Secret*$_{PC}$ and the new domain *Secret*$_2$ by *Secret*$_{PDA}$. We *assume* confidential paths ($\{y1, e2\} \subseteq$ *confIf*) between the machine domain *rPDAserver* and the domains *Screen content* and *Copy of screen content*, because we do not suspect a malicious subject being inside the machine itself, such as a trojan horse. For the same reason, we *assume* that the domains *Screen content* and *Copy of screen content* are not known to the *Malicious PC* domain, and we *assume* trusted paths ($\{y5, y6\} \subseteq trustedIf$) between the machine domain *rPDAserver* and the domain *Secret*$_{PC}$ as well as between the domains *rPDAclient* and *Secret*$_{PDA}$.

In addition, we must consider three preconditions concerning the relations *known* and *match*. These preconditions are not assumed to be fulfilled. For this reason, we must consider them as subproblems that have to be analyzed by applying further security problem frames. Following the arrow annotated "1" in Fig. 7, we decide to choose the security problem frame for distributing secrets presented in Sect. 3.4, because this frame provides postconditions sufficient to fulfill these preconditions.

From here on, we only discuss preconditions that cannot be assumed to be fulfilled.

Considering the instantiated security problem frame for distributing secrets, we conclude that there does not exist a trusted path between the domains *rPDAserver* and *rPDAclient*, because it is possible that a malicious subject eavesdrops on the (public) Bluetooth traffic. Therefore, we choose the concretized security problem frame for distributing secrets using negotiation presented in Sect. 4.3, because this frame abstains from using such a trusted path.

Its preconditions contain the relation $authenticated$; therefore, this frame requires us to take an authentication subproblem into consideration. We cannot assume these preconditions to be fulfilled. Thus, we have identified one more subproblem. Following the arrow annotated "2" in Fig. 7, we decide to choose the security problem frame for authentication presented in Sect. 3.3, because this frame provides postconditions sufficient to fulfill the given preconditions.

To concretize that problem, we decide to choose a dynamic authentication mechanism, because a mechanism using digital signatures is appropriate for the authentication between machines. Therefore, we must instantiate the concretized security problem frame for dynamic authentication presented in Sect. 4.2.

Its preconditions contain the relations $known$ and $match$. Unfortunately, we cannot *assume* these preconditions to be fulfilled.

We have to solve a subproblem considering the distribution of secrets, thus following the arrow annotated "3" in Fig. 7. We must consider the security problem frame for distributing secrets presented in Sect. 3.4, because this frame provides postconditions sufficient to fulfill these preconditions.

Because we have to provide digital signatures, the distribute machine is instantiated by a trust center, which represents a certification authority in a public key infrastructure. The trust center is responsible for the distribution of the digital signatures. We may *assume* a trusted path between the trust center and the PC. Hence, we choose the concretized security problem frame for distributing secrets using trusted paths presented in Sect. 4.3.

Now, no more preconditions are left to be established. Hence, we have identified and analyzed in detail all subproblems that are necessary to solve the *confidentiality subproblem for the screen content*. Following the dependencies of our pattern system helped us to systematically and completely set up all the necessary subproblems. A more detailed presentation of the case study can be found in [5].

The next step in solving the problem would be to derive specifications of all the machines contained in the different subproblems. All in all, our original problem will be solved when the derived specifications are correctly implemented. In [4], we show the next steps of the development process, e.g. how the specification is represented using UML sequence diagrams.

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*. 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.

Gürses et al. [2] present the CREE method for multilateral security requirements elicitation. Their method concentrates on confidentiality requirements and employs use cases to treat functional requirements. The CREE method

is useful to be applied in a phase of the security requirements engineering process that precedes the application of our approach.

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 our procedure focuses on identifying and analyzing requirements beforehand.

Security patterns [1] 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 Conclusions and Future Work

In this paper, we have presented a pattern system that supports security requirements engineering. It is based on security problem frames and their concretized counterparts. These special kinds of problem frames serve to structure, characterize, analyze, and finally solve software development problems in the area of software and system security.

Since realistic security problems must usually be decomposed into several subproblems, our pattern system is of considerable help for security engineers. Important classes of security problems are characterized and represented by security problem frames. A security engineer can inspect the security problem frame catalog and identify all the frames relevant for the given problem. Then, each subproblem is further elaborated by selecting a concretized security problem frame associated with the previously chosen security problem frame. In the next step, the associated preconditions must be inspected. They can either be assumed as true, or they can lead to one or more subproblems to be solved. These subproblems can be identified easily, following the dependency relations of our pattern system. That process terminates, when all preconditions that are left can be assumed to hold. Moreover, we indicate related frames, which are commonly used together with the considered frame.

Thus, our pattern system leads the security engineer to a complete description of all the subproblems that must be solved in order to solve a complex security problem. Our pattern system integrates well with software requirements engineering using Jackson's problem frames.

In the future, we intend to find new patterns to extend the catalog 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 our security engineering method.

Acknowledgment We thank Bozhan Ivanov for his constructive comments on this work.

References

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

[2] S. F. 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 Müller, G., 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. Using Problem Frames for Security Engineering. Technical report, Universität Duisburg-Essen, 2006. `http://swe.uni-duisburg-essen.de/intern/seceng06.pdf`.

[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, 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.

[9] G. Popp, J. Jürjens, G. Wimmel, and R. Breu. Security-Critical System Development with Extended Use Cases. In *APSEC*, pages 478–487, 2003.