2010 International Conference on Availability, Reliability and Security

Threat- and Risk-Analysis During Early Security Requirements Engineering

Holger Schmidt
University Duisburg-Essen, Faculty of Engineering
Department of Computer Science and Applied Cognitive Science, Workgroup Software Engineering
Duisburg, Germany
Email: holger.schmidt@uni-duisburg-essen.de

***Abstract*—We present a threat and risk-driven methodology to security requirements engineering. Our approach has a strong focus on gathering, modeling, and analyzing the environment in which a secure ICT-system to be built is located. The knowledge about the environment comprises threat and risk models. This security-relevant knowledge is used to assess the adequacy of security mechanisms, which are selected to establish security requirements.**

***Keywords*-security requirements engineering; risk analysis; threat analysis; domain knowledge**

I. Introduction

Security describes the inability of the environment to have an undesirable effect on an ICT-system [1]. Expressed from the software development point of view, the software of an ICT-system must be constructed in a way such that the ICT-system is protected from the environment.

From the definition of the term security alone the importance of the *environment* in which an ICT-system is integrated when we deal with security becomes clear. The adequacy of an ICT-system's security mechanisms is strongly influenced by its intended operational environment. Thus, *knowledge* about the environment must be collected to support or even allow the decision for particular security mechanisms. Especially knowledge about the malicious part of the environment, i.e., the part that can attack an ICT-system in order to nullify or bypass its security mechanisms, must be gathered, modeled, and assessed.

The gathering and modeling parts involve *threat analysis techniques* such as Microsoft's STRIDE [2], attack trees by Schneier [3], and the approach by Fernandez et al. [4]. Threat analysis is applied to identify *threats* that exploit *vulnerabilities* of security mechanisms. The assessment part is related to risk analysis techniques such as CORAS by Braber et al. [5]. Risk analysis is used to determine the *probability* that security mechanisms will work correctly in the intended (malicious) environment.

When constructing secure ICT-systems, it is instrumental to take the environment into account right from the beginning of the software development. Consequently, we focus in this paper on *early security requirements engineering*. We consider our *security engineering process using patterns* (SEPP) [6], [7]. There, security requirements are elicited, analyzed, and documented so that security mechanisms adequate to establish the security requirements can be selected. SEPP makes use of special *patterns* called *security problem frames* for security requirements and *concretized security problem frames* for security mechanisms. Thus, security requirements are strictly separated from solutions, i.e., security mechanisms.

Results from applying threat and risk analysis techniques heavily influence the process of selecting security mechanisms to establish security requirements. In this paper, we analyze the impact of threat and risk analysis on security requirement engineering. We extend SEPP by a threat and risk-driven procedure to select adequate security mechanisms. We illustrate the procedure using an example software development.

The rest of the paper is organized as follows: In Sec. II, we introduce a concrete security-critical software development problem, which we use in the course of the paper as a running example. In Sec. III, we demonstrate SEPP based on the example. In Sec. IV, we present the threat and risk analysis extensions to the approach. We consider related work in Sec. V. In Sec. VI, we give a summary and directions for future research.

II. Case Study

We use the following software development problem as a case study to demonstrate the techniques presented in this paper.

> A secure text editor should be developed. The text editor should enable an author to create, edit, open, and save text files. The text files should be stored confidentially.

The informal security requirement (SR) can be described as follows:

> Preserve confidentiality of text file except for its file length for honest environment and prevent disclosure to malicious environment.

Note: We decide to focus on *storing* text files confidentially. The given software development problem can also be

978-0-7695-3965-2/10 $26.00 © 2010 IEEE
DOI 10.1109/ARES.2010.14

interpreted such that the security requirement also covers confidential editing operations, e.g., confidential clipboard copies. To simplify matters, this is not covered in the security requirements analysis presented in this paper. For the same reason, the create and edit functionality of the secure text editor is not covered in our case study. Practically, it is very difficult to develop 100% confidential systems. Hence, as an example, we discuss an SR that allows the secure text editor to leak the text file length.

III. Problem Frames for Security Requirements Engineering

In earlier publications (cf. [8], [9]), we presented special patterns defined for structuring, characterizing, and analyzing problems that occur frequently in security engineering. Similar patterns for functional requirements have been proposed by Jackson [10]. They are called *problem frames*. Accordingly, our patterns are named *security problem frames* (SPF). SPFs consider *security requirements*.

Furthermore, for each SPF, we defined a set of *concretized security problem frames* (CSPF) that take *generic security mechanisms*[1] (e.g., encryption to keep data confidential) into account to prepare the ground for solving a given security problem. Since CSPFs involve first solution approaches, they consider *concretized security requirements*.

Problem frames, SPFs and CSPFs are graphically depicted by *frame diagrams*. Figure 1 shows an instance of the frame diagram of the *CSPF confidential data storage using password-based encryption*. Instances of frame diagrams are called *problem diagrams*.

SPFs and CSPFs as well as instances of these frames make use of the same syntactic elements. They consist of a *machine domain* denoted by a rectangle with two vertical stripes, which represents the software that should be developed in order to fulfill the (concretized) security requirement textually denoted in a dashed oval. According to our case study, the software to be developed is represented by the machine domain Secure text editor. The concretized security requirement CSR derived from SR is stated as follows:

> If password is unknown to malicious environment, then confidentiality of text file except for its file length is preserved for honest environment and disclosure to malicious environment is prevented.

The environment in which the software development problem is located is structured by *domains*, which are graphically denoted by plain rectangles. According to Jackson [10], we distinguish *causal* domains that comply with some physical laws, *lexical* domains that are data representations, and *biddable* domains that are usually people. In the instantiated frame diagram depicted in Fig. 1, a marker "X" indicates that the corresponding domain is a lexical domain, "B" indicates a biddable domain, and "C" indicates a causal domain[2].

The connecting lines between domains represent *interfaces* that consist of *shared phenomena*. According to Jackson, we distinguish *symbolic* phenomena, e.g., data items, *causal* phenomena, e.g., operations, and *event* phenomena, e.g., events and messages. The sets that contain phenomena are named according to their type: sets with symbolic phenomena begin with the letter "Y", those with causal phenomena with the letter "C", and for event phenomena the letter "E" is used. Additionally, the sets have a consecutive number as suffix. Shared phenomena are observable by at least two domains, but controlled by only one domain. The notation "STE!E7" means that the event phenomena of interface E7 between the domains Secure text editor (abbreviated STE) and Encrypted text file are controlled by the Secure text editor domain.

The environment contained in SPFs and CSPFs as well as instances of them is divided into an *honest* and a *malicious* part. The malicious environment of a CSPF is equally modeled to that of the corresponding SPF. Those CSPFs that deal with *confidentiality requirements* basically consider *passive* malicious environments to observe the protected assets the confidentiality requirements refer to. In contrast, the CSPFs that treat *integrity requirements* are equipped with *active* malicious environments to modify the protected assets the integrity requirements refer to.

According to our case study, the domain Author and the interfaces A!E1 and STE!Y1 between this domain and the machine domain represent the honest environment. The domains Malicious user and Operating system together with the interfaces OS!Y4 and MU!E6 between them, the interface MU!E3 between the Malicious user domain and the machine domain, and the interfaces ETF!Y2 and OS!E6 between the domains Operating system and Encrypted text file represent the malicious environment. It is emphasized in Fig. 1 as a hashed area. Moreover, the *asset* is represented by the domain Text file in Fig. 1.

Because of the encryption mechanism used by the machine domain, the interface ETF!Y2 between Secure text editor and Encrypted text file and between Encrypted text file and Operating system contain the phenomenon EncryptedContentOfTextFile. Authors can enter passwords for encrypting and decrypting text files. Therefore, the interface A!E1 between Author and Secure text editor contains the phenomenon Password. Malicious users can also enter passwords under the assumption that they cannot guess passwords of honest users. Therefore, the interface MS!E3 between Malicious user and Secure text editor contains the phenomenon WrongPassword.

A dashed line represents a requirements reference, and the

[1] For reasons of simplicity, we write security mechanism, even if we refer to a *generic* security mechanism.

[2] Showing domain types in frame instances differs from the original notation by Jackson [10], that does not indicate the domain types.

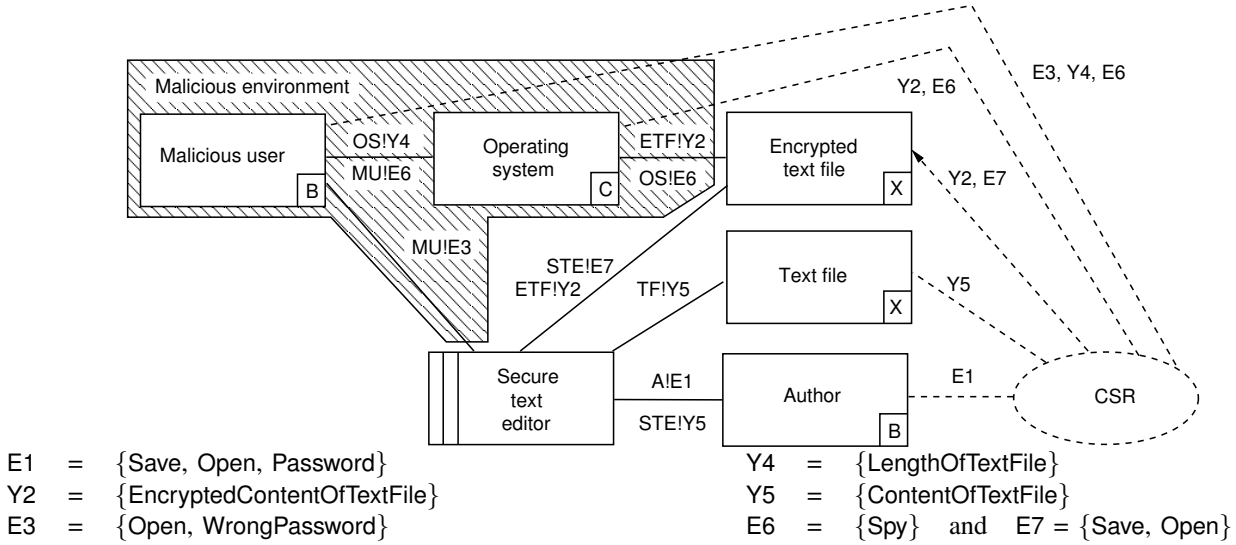

Figure 1. Instantiated CSPF Confidential Data Storage using Password-Based Encryption "Secure text editor"

arrow shows that it is a *constraining* reference. In the instantiated frame diagram depicted in Fig. 1, the CSR refers to the domains Author, Text file, Malicious user, and Operating system. It constrains the Encrypted text file domain, because the text files are encrypted by the machine.

The environment, i.e, the domains (except for the machine domain) and the interfaces with the phenomena they contain, is described by *domain knowledge*. We distinguish two kinds of domain knowledge:

Facts describe fixed properties of the environment irrespective of how the machine is built. For example, it is a fact that the secure text editor is running on a public computer.

Assumptions describe conditions that are needed, so that the requirements are accomplishable. For example, one must assume that the authors do not reveal their passwords.

In contrast to domain knowledge, that describes the environment independently of the machine, *requirements* are expressed by describing the environment after the machine is integrated into it.

SPFs and CSPFs are organized in *pattern catalogs*, which form a *pattern system* [9]. After an SPF is instantiated, the pattern system is used to identify a set of possible CSPFs that can be used to solve the problem characterized by the SPF instance. Moreover, dependencies between security mechanisms (represented by CSPF instances) and security requirements (represented by SPF instances) can be determined.

SPFs are equipped with with a formal description of the security requirement, which we call *effect*. Furthermore, CSPFs are equipped with formally expressed *necessary conditions*, which must be met by the environment for the security mechanism that the CSPF represents to be applicable. If a necessary condition does not hold, the effect described in the according SPF cannot be established. Necessary conditions can be covered by assumptions.

For reasons of simplicity, we describe the necessary conditions of the CSPF instance in Fig. 1 only informally:

- Passwords used by the honest environment must be different from the ones used by the malicious environment. Otherwise, a password-based encryption mechanism is not secure.
- Passwords transmitted by the honest environment to the machine must not be eavesdropped by the malicious environment.
- Passwords used by the honest environment must be transmitted to the machine domain in an integrity-preserving way.

The concepts presented in this section, i.e., SPFs, CSPFs, the pattern system, and the effects as well as the necessary conditions, are the basis of our secure software engineering method SEPP [6], [7].

IV. Selecting Security Mechanisms Based On Security-Relevant Domain Knowledge

We present security-relevant domain knowledge that involves threat and risk analysis in Sec. IV-A. In Sec. IV-B, we present a procedure to collect, model, and assess knowledge about the malicious environment. The approach is then illustrated in Sec. IV-C using the case study.

A. Security-Relevant Domain Knowledge

In the following, we relate terminology from security requirements engineering and the (C)SPF approach presented in Sec. III with terminology from threat and risk analysis. The goal is to provide an understanding and a definition of security-relevant domain knowledge. Figures 4

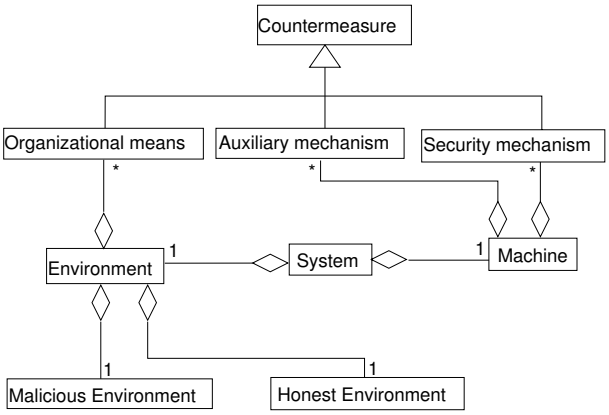

Figure 2. System and Countermeasures

and 2 summarize the terminology and interrelationships using a UML class diagram [11]. They are complemented by Table IV-A that serves to map a part of the notions contained in the class diagrams to the elements of CSPFs.

Using Zave's and Jackson's terminology [12], a *system* consists of a machine in its environment (see Fig. 2). Adopting a holistic view, we consider security to be a system property. Security can only be regarded as a characteristic of a system. It is not a characteristic of the machine alone.

As depicted in Fig. 2, a *countermeasure* protects at least one asset, and each countermeasure is a part of the system. Security mechanisms (e.g., encryption and access control mechanisms), auxiliary mechanisms (e.g., mechanisms that obfuscate password fields), and organizational means (e.g., user policies) are countermeasures. The latter are countermeasures that are realized by the biddable part of the environment since this part cannot be influenced by the machine. In contrast, the former two types of countermeasures are realized by the machine. As shown in Table IV-A, a CSPF describes a security mechanism. In our case study, the encryption mechanism is represented by the machine domain Secure text editor and the phenomena Password, WrongPassword, and EncryptedContentOfTextFile.

In his PhD thesis, Mayer [13] gives an overview of the different definitions of threat and risk terminology, and he comes up with *consolidated* notions in this field. The terminology related to threats and risk presented in this section is based on these notions.

Figure 3 is adapted from the ISO/IEC 61508 [14] standard for the development of safety-critical electronic devices, and it serves to clarify the different notions of risk in the field of secure software engineering.

According to SEPP, the functionality and security mechanisms are developed concurrently during software development. In the beginning, the system is unprotected, i.e., security is not yet considered. Thus, the system has a number of *vulnerabilities*, which can be exploited by *threats* (see Fig. 4). At this stage, the number of threats to nullify the

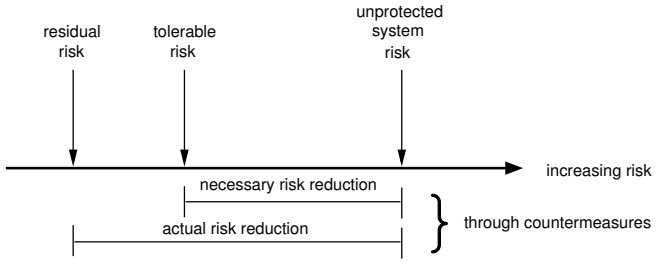

Figure 3. Risk and Risk Reduction (adapted from [14, Part 5, Annex A])

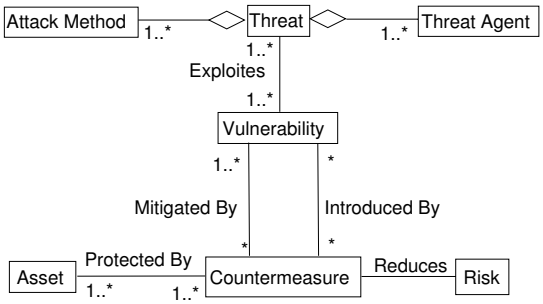

Figure 4. Countermeasures, Assets, and Threat and Risk Analysis Terminology

security requirements and the risk that the threats turn into successful attacks is maximal. As shown in Fig. 3, we refer to this risk as the *unprotected system risk*. The goal of secure software engineering is to reduce this risk by applying countermeasures until a level of *tolerable risk* is reached. Since it is practically very difficult (if not impossible) to develop completely secure systems, there remains a *residual risk*. We consider a system secure in a given environment, if the residual risk is lower than the tolerable risk.

Against this background, we now describe Fig. 4 in detail. We consider an *asset* some piece of information to be protected by countermeasures, which mitigates *vulnerabilities*. Furthermore, a countermeasure possibly introduces a number of vulnerabilities. As shown in Table IV-A, an asset is represented by a set of lexical domains and/or symbolic phenomena. A vulnerability is described by a necessary condition. In general, vulnerabilities are exploited by *threats*. A threat consist of at least one *threat agent*, and at least one *attack method*. As shown in Table IV-A, a threat agent is represented by at least one biddable domain and interface, and an attack method by a set of phenomena. A threat model (e.g., attack trees by Schneier [3]) that describes a threat in more detail can be regarded as domain knowledge associated with the corresponding part of the malicious environment. In our case study, the domain Text file and the phenomenon ContentOfTextFile are protected. A password-based encryption mechanism has several vulnerabilities, e.g., the passwords could be determined using brute force. So, a threat to the mechanism is the possible exploitation of this

Risk/Threat Notions	CSPF Notions
threat	potential attack to nullify or bypass necessary condition
threat agent	part of malicious environment, i.e., at least one biddable domain and interface
attack method	set of phenomena that are part of the malicious environment
vulnerability	described by necessary condition
security mechanism	described by CSPF
asset	set of lexical domains and/or symbolic phenomena

Table I
RELATION BETWEEN RISK/THREAT NOTIONS AND CSPF NOTIONS

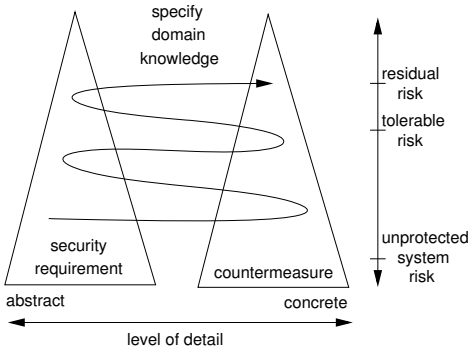

Figure 5. Risk-Driven Twin Peaks Process-Model (adapted from [15])

vulnerability by the attack method brute force.

Since CSPFs introduce security mechanisms, threats to the machine that implements these mechanisms can be considered when a CSPF is instantiated. When instantiating a CSPF, it is possible to model a sophisticated malicious environment, i.e., one that can make use of certain knowledge, e.g., passwords, encryption keys, and ciphertexts.

When a security requirements engineering method such as SEPP is applied, which strictly separates security requirements from solutions to these requirements, then *risk* can be initially considered after first solution approaches are determined. To ensure that a security mechanism works properly, i.e. that it fulfills the security requirements it is intended to fulfill, necessary conditions must be fulfilled. A *risk analysis method* can be used to determine the *probability* that a selected security mechanism will work properly in the intended (malicious) environment. More precisely, such a method helps to decide if necessary conditions can be assumed to be fulfilled or if they constitute new security requirements, which must be treated by further countermeasures. If the risk that a condition does not hold is not tolerable, then it should be considered as a new security requirement. Otherwise, the necessary condition can be considered as an assumption, and hence becomes part of the security-relevant domain knowledge.

In the next section, we make use of the interrelationships presented in this section to define a selection procedure for security mechanisms.

B. Selection Procedure for Security Mechanisms

We present a two-step selection procedure for security mechanisms. Figure 5 describes this procedure based on a spiral process model adapted from [15]: the overall goal is to reduce the risk of the unprotected system until the risk is tolerable. Security requirements are analyzed and then countermeasures are selected in an iterative and incremental way. That is, one goes back and forth between rather abstract and more concrete system modeling. In each iteration, security-relevant domain knowledge is specified based on threat and risk models. It is used to decide on the adequacy of countermeasures. Finally, a system with only a residual risk that is lower than the tolerable risk is developed.

The selection procedure starts given a set of already instantiated SPFs.

Step 1 – Select and Instantiate CSPF: This step must be executed for each SPF instance. One selects a CSPF from the pattern system based on (not necessarily security-relevant) domain knowledge obtained when instantiating the SPF. Afterwards, one assigns concrete values to the domains, interfaces, phenomena, and the concretized security requirement. The result of this step is a set of CSPF instances that represent security mechanisms.

Step 2 – Analyze Dependencies: This step must be executed for each CSPF instance. For each instantiated necessary condition of the CSPF instance, is must be determined if it can be assumed, or if it has to be treated using additional countermeasures:

1) The instantiated necessary condition constitutes a vulnerability of the security mechanism represented by the CSPF instance. Identify threats to the security mechanism based on the instantiated necessary condition. The result of this substep is a set of threats that exploit the vulnerability described by the necessary condition.
2) For each threat to the necessary condition under consideration, determine the risk that the threat turns into a successful attack. Then, consider the following cases:
   - $>$ tolerable risk: If the risk is higher than the tolerable risk, then the threat should be mitigated by countermeasures.
   - $<$ tolerable risk: If the risk is lower than the tolerable risk, then one can assume that the threat does not turn into a successful attack. This means that is not treated by the system at all.

   In both cases, one proceeds. The result of this substep is a set of threats with associated risk values.
3) For each threat with an associated risk higher than

the tolerable risk, find countermeasures to reduce the risk until it becomes tolerable. We must deal with the following three cases:

- case 1 Introduce an auxiliary mechanism to reduce the risk of the threat resulting in a successful attack. Then, one proceeds.
- case 2 Select and instantiate an SPF (which can be identified based on the pattern system). Then, go back to step 1. The CSPF that is then instantiated represents a countermeasure to the threat.
- case 3 Some threats cannot be treated by the machine, because they are threats executed by the malicious environment on the honest environment. Thus, the application of technical countermeasures, i.e., security mechanisms and auxiliary mechanisms, is impossible. One must either deal with the threat by organizational means, or assume that the threats will not turn into successful attacks, or the initially chosen security mechanism is not adequate. In the latter case, one must go back to step 1 and select a different CSPF. Otherwise, one proceeds.

If one cannot find countermeasures sufficient to reduce the risk to a value lower than the tolerable risk, one must go back to step 1 and select a different CSPF.

The results of this step are consolidated sets SPF and CSPF instances, additional auxiliary mechanisms and organizational means, and security-relevant domain knowledge (i.e., results from threat and risk analysis). Note that the result of the procedure possibly leads to the conclusion that the system cannot be realized because the risk cannot be sufficiently reduced.

C. Illustration

We now apply the previously described procedure to the secure text editor case study. SEPP's step that describes the instantiation of the SPF is not shown in this paper, and we consider it as already executed. We proceed with the first step of the procedure presented in the previous section. The instance of the CSPF and the instantiated necessary conditions are presented in Sec. III.

In the second step, we consider the instantiated necessary conditions presented in Sec. III. The first one expresses that passwords used by the honest environment must be different from the ones used by the malicious environment. To bypass this necessary condition, the following threats are found:

- NC1-T1 Malicious environment determines the right passwords (brute force)
- NC1-T2 Malicious environment applies social engineering means to reveal the right passwords
- NC1-T3 Malicious environment guesses the right passwords

For each threat, the risk that it turns out to become a successful attack is determined. In our case study, the brute force threat NC1-T1 is assessed to have a risk higher than the tolerable risk, since the encrypted text files are stored on some *public* storage device. Note, that the statement that public access to the storage device is available is a fact. Moreover, the risk of the social engineering threat NC1-T2 is considered higher than the tolerable risk because the authors of the text files are not aware of social engineering attacks (which is a fact, too). In contrast, it is unlikely that the threat NC1-T3 turns into a successful attack, since passwords are strings and thus, the space of possible passwords is sufficiently large. Therefore, the risk of this threat is lower than the tolerable risk.

For each threat with intolerable risk, countermeasures have to be determined. NC1-T1 can be prevented by introducing a (functional) auxiliary mechanism that blocks the password dialog after a wrong password has been entered a limited number of times.

NC1-T2 threatens none of the machine's interfaces directly. Instead, it is located purely in the biddable environment. Since the biddable part of the environment (in contrast to the lexical and causal parts) cannot be influenced by the machine, countermeasures enforced by the machine are not available. The only available choices to deal with this threat are assuming that the biddable environment is immune against social engineering, installing an organizational policy that instructs authors not to reveal their passwords, or we have to choose a different security mechanism. In our case study, we rely on the mentioned assumption, which becomes a part of the security-relevant domain knowledge.

Since the countermeasure for NC1-T1 prevents brute force attacks and NC1-T2 is assumed to never turn into an successful attack, the overall risk for the fulfillment of the necessary condition is lower than the tolerable risk, i.e., the residual risk is lower than the tolerable risk. For that reason, the necessary condition can be assumed to hold.

The second necessary condition expresses that the passwords are transmitted from the honest environment to the machine in a confidentiality-preserving way. To bypass this necessary condition, the following threat is found:

- NC2-T1 Malicious environment eavesdrops on transmission of password.

The risk for this threat to turn into a successful attack is lower than the tolerable risk, because we expect that the malicious environment has only very basic technical abilities. For that reason, we can assume that the corresponding necessary condition holds. The assumptions on the technical abilities are a part of the security-relevant domain knowledge.

The third necessary condition expresses that the passwords are transmitted from the honest environment to the machine in an integrity-preserving way. To bypass this necessary condition, the following threat is found:

NC3-T1 Malicious environment modifies password during transmission.

The risk for this threat to turn into a successful attack is lower than the tolerable risk, because the primary goal of the malicious environment is to reveal the encrypted text files, and not to make them unavailable for the honest environment. For that reason, we can assume that the corresponding necessary condition holds.

In summary, all three necessary conditions can be assumed to hold. One can be sure that the chosen security mechanism works properly in its operational environment, except for some residual (but tolerable) risk that the considered threats turn into successful attacks.

V. Related Work

While Mayer [13] relates the threat and risk terminology to the security requirements engineering terminology, his approach is not environment-centric and it does not consider the transition from security requirements to security mechanisms. The environment is not explicitly covered because Mayer mainly refers to security requirements engineering methods such as the goal-based Secure Tropos [16], [17] and KAOS [18], [19] approaches, and SQUARE [20] that are rather machine-centric. Consequently, Mayer does not explicitly consider security-relevant domain knowledge.

Asnar et al. [21] propose the Tropos Goal-Risk Framework, an extension of earlier work [22], to assess risk based on *trust relations* among actors. Trust, combined with the concept of *delegation* of the fulfillment of a goal, enables the modeling of responsibility transfer from one actor to another. The authors propose qualitative risk reasoning techniques to support the analyst in evaluating and choosing among different possible sub-goal-trees. Compared to our approach, Asnar et al. do not model threats explicitly. Moreover, the notion of trust is comparable to the assumptions on biddable parts of the environment that we summarize together with facts under the term security-relevant domain knowledge.

Lin et al. [23] 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 authors state that the purpose of anti-requirements and abuse frames is to analyze security threats and derive security requirements. Based on a set of functional requirements and security objectives, the authors propose an iterative threat analysis method which is comparable to our approach to the extent that they identify and tackle vulnerabilities iteratively based on their abuse frames. It is our impression that abuse frames rather compare to problem diagrams than to problem frames. Problem diagrams are problem models that do not constitute patterns. Moreover, risk is not covered by the abuse frames approach, and threat analysis results are not explicitly considered as security-relevant domain knowledge.

Houmb et al. [24] present a cost-benefit trade-off analysis supported using Bayesian Belief Networks (BBN). According to the authors the "... framework separates security concerns from core functionality using aspects. Each treatment strategy is modeled as an aspect model, and then composed with the primary model." The treatment strategies are comparable to the countermeasures in our approach. The properties of the treatment strategies are estimated by variables, which are the input for the BBN. The security level of a system is compared to an acceptance level comprised of the budget, security acceptance criteria, law and regulations, business goals, and policies. This security level is similar to the level of tolerable risk of our approach. In summary, the approach by Houmb et al. supports to justify design decisions based on risk analysis and system threats derived from successfully exploited vulnerabilities.

A comprehensive comparison of security requirements engineering approaches with respect to threat, risk, and further issues can be found in [25].

Note that we are aware of other approaches to threat and risk analysis (e.g., Microsoft STRIDE [2] and SDL [26]). We do not consider them in detail here due to space limitations.

VI. Conclusions and Future Work

The paper at hand relates security requirements engineering to threat and risk analysis terminology to elaborate an understanding of security-relevant domain knowledge.

The presented approach is embedded into SEPP, which is extended by a method driven by security-relevant domain knowledge to select adequate security mechanisms. The security-relevant domain knowledge comprises facts and assumptions on the operational environment, especially threat and risk models.

In the future, we would like to extend our SEPP's later phases, i.e., architectural design phases, by an approach similar to the one presented in this paper. We believe that threat and risk analysis must be repeatedly applied throughout the complete software development lifecycle. Moreover, we consider an analysis of the interactions between security requirements and different other kinds of quality requirements such as safety, usability, and performance requirements as a key feature of a software engineering approach to develop highly secure software.

Acknowledgments

I thank Maritta Heisel for her extensive and valuable feedback on my work.

References

[1] L. Røstad, I. A. Tøndel, M. B. Line, and O. Nordland, "Safety vs. security," in *Proceedings of the International Conference on Probabilistic Safety Assessment and Management (PSAM)*, M. G. Stamatelatos and H. S. Blackman, Eds. ASME Press, New York, 2006.

[2] S. Hernan, S. Lambert, T. Ostwald, and A. Shostack, "Uncover security design flaws using the stride approach," November 2006, http://msdn.microsoft.com/de-de/magazine/cc163519.aspx. [Online]. Available: http://msdn.microsoft.com/de-de/magazine/cc163519.aspx

[3] B. Schneier, "Attack trees," Dr. Dobb's Journal, 1999, http://www.schneier.com/paper-attacktrees-ddj-ft.html. [Online]. Available: http://www.schneier.com/paper-attacktrees-ddj-ft.html

[4] E. B. Fernandez, D. L. la Red M., J. Forneron, V. E. Uribe, and G. Rodriguez G., "A secure analysis pattern for handling legal cases," in *Latin America Conference on Pattern Languages of Programming (SugarLoafPLoP)*, 2007, http://sugarloafplop.dsc.upe.br/wwD.zip.

[5] F. Braber, I. Hogganvik, M. S. Lund, K. Stølen, and F. Vraalsen, "Model-based security analysis in seven steps – a guided tour to the CORAS method," *BT Technology Journal*, vol. 25, no. 1, pp. 101–117, 2007.

[6] D. Hatebur, M. Heisel, and H. Schmidt, "A security engineering process based on patterns," in *Proceedings of the International Workshop on Secure Systems Methodologies using Patterns (SPatterns)*. IEEE Computer Society, 2007, pp. 734–738.

[7] ——, "Analysis and component-based realization of security requirements," in *Proceedings of the International Conference on Availability, Reliability and Security (AReS)*. IEEE Computer Society, 2008, pp. 195–203.

[8] ——, "Security engineering using problem frames," in *Proceedings of the International Conference on Emerging Trends in Information and Communication Security (ETRICS) (LNCS 3995)*, ser. LNCS 3995, G. Müller, Ed. Springer Berlin / Heidelberg / New York, 2006, pp. 238–253.

[9] ——, "A pattern system for security requirements engineering," in *Proceedings of the International Conference on Availability, Reliability and Security (AReS)*. IEEE Computer Society, 2007, pp. 356–365.

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

[11] UML Revision Task Force, *OMG Unified Modeling Language: Superstructure*, Object Management Group (OMG), November 2007, http://www.omg.org/spec/UML/2.1.2/. [Online]. Available: http://www.omg.org/spec/UML/2.1.2/

[12] P. Zave and M. Jackson, "Four dark corners of requirements engineering," *ACM Transactions on Software Engineering and Methodology*, vol. 6, no. 1, pp. 1–30, 1997.

[13] N. Mayer, "Model-based management of information system security risk," Ph.D. dissertation, University of Namur, April 2009, http://nmayer.eu/publis/Thesis_Mayer_2.0.pdf.

[14] International Organization for Standardization (ISO) and International Electrotechnical Commission (IEC), "Functional safety of electrical/electronic/programmable electronic safty-relevant systems," ISO/IEC 61508, 2000, http://www.iec.ch/61508/. [Online]. Available: http://www.iec.ch/61508/

[15] B. Nuseibeh, "Weaving together requirements and architectures," *Computer*, vol. 34, no. 3, pp. 115–117, 2001.

[16] P. Giorgini and H. Mouratidis, "Secure Tropos: dealing effectively with security requirements in the development of multiagent systems," in *Safety and Security in Multi-Agent Systems – Selected Papers*, M. Barley, F. Masacci, H. Mouratidis, and P. Scerri, Eds. Springer Berlin / Heidelberg / New York, 2006.

[17] ——, "Secure tropos: A security-oriented extension of the tropos methodology," *International Journal of Software Engineering and Knowledge Engineering*, vol. 17, no. 2, pp. 285–309, 2007.

[18] A. van Lamsweerde, "Elaborating security requirements by construction of intentional anti-models," pp. 148–157, 2004.

[19] ——, "Engineering requirements for system reliability and security," in *Software System Reliability and Security*, ser. NATO Security through Science Series - D: Information and Communicarion Security, J. G. M. Broy and C. Hoare, Eds. IOS Press, 2007, vol. 9, pp. 196–238.

[20] N. R. Mead, E. D. Hough, and T. R. Stehney II, "Security quality requirements engineering (SQUARE) methodology," Carnegie Mellon Software Engineering Institute, Tech. Rep. CMU/SEI-2005-TR-009, 2005.

[21] Y. Asnar, P. Giorgini, F. Massacci, and N. Zannone, "From trust to dependability through risk analysis," in *Proceedings of the International Conference on Availability, Reliability and Security (AReS)*. IEEE Computer Society, 2007, pp. 19–26.

[22] Y. Asnar, P. Giorgini, and J. Mylopoulos, "Risk modelling and reasoning in goal models," University of Trento, Tech. Rep. DIT-06-008, 2006.

[23] L. Lin, B. Nuseibeh, D. Ince, and M. Jackson, "Using abuse frames to bound the scope of security problems," in *Proceedings of the IEEE International Requirements Engineering Conference (RE)*. Los Alamitos, CA, USA: IEEE Computer Society, 2004, pp. 354–355.

[24] S. H. Houmb, G. Georg, R. France, J. Bieman, and J. Jürjens, "Cost-benefit trade-off analysis using bbn for aspect-oriented risk-driven development," in *Proceedings of the IEEE International Conference on Engineering of Complex Computer Systems (ICECCS)*. IEEE Computer Society, 2005.

[25] B. Fabian, S. Gürses, M. Heisel, T. Santen, and H. Schmidt, "A comparison of security requirements engineering methods," *Requirements Engineering*, to appear.

[26] M. Howard and S. Lipner, *The Security Development Lifecycle*. Redmond, WA, USA: Microsoft Press, 2006.