A SECURITY DESIGN PATTERN TAXONOMY BASED ON ATTACK PATTERNS
Findings of a Systematic Literature Review

Andreas Wiesauer and Johannes Sametinger
Department of Business Informatics - Software Engineering, Johannes Kepler University of Linz
Altenbergerstrasse 69, Linz, Austria

Keywords: Security design pattern taxonomy, Security design patterns, Attack patterns, Literature review.

Abstract: Security design patterns are proven solutions to security problems in a given context with constructive measures of how to design certain parts of a software system. The literature contains numerous definitions, examples, and taxonomies of such patterns. There are also a few quality criteria for them. We suggest a new taxonomy based on attack patterns in order to enhance applicability of security design patterns especially for non-experts in software security. We further suggest a combined consideration of attack patterns, security design patterns and test cases for the validation and evaluation of security design patterns.

1 INTRODUCTION

Patterns provide a convenient way for encapsulating and reusing knowledge. Their purpose is to communicate proven solutions in a certain domain. The use of patterns has emerged from architecture and has been applied to software engineering by Gamma et al. (Gamma et al., 1995). Starting with Yoder and Barcalow, the pattern concept has been transformed to the field of software security (Yoder and Barcalow, 1997). The idea of providing security knowledge as patterns seems obvious. Numerous security design patterns have been published over the last years.

The term ”security pattern” is used in various contexts. Security patterns may be reflected directly in software. This is also true for patterns known from the seminal work of (Gamma et al., 1995). In this case, patterns typically have an impact on the architecture and the design of software. Other security patterns deal with administrative means for securing software, e.g., with fire-walls or with demilitarized zones. The term security pattern is also used for patterns that influence activities of software life-cycle phases other than the design, e.g., blueprints for security requirement specifications or test case definitions. In this paper we focus on patterns that are reflected in software architecture and design. We use the term ”security design pattern” for that purpose.

In this paper, we present the results of a systematic literature review aimed at identifying security design patterns and classification schemes. Further, we suppose a classification scheme of security design patterns that is based on attack patterns. The paper is structured as follows: In Section 2 we introduce design patterns. Objectives and the methodology of our approach, a systematic literature review, is presented in Section 3. Results of the literature review are discussed in Sections 4, 5 and 6, i.e., security pattern definitions, quality criteria, and taxonomies, respectively. A new taxonomy that is based on attack patterns is introduced in Section 7. Conclusions are drawn in Section 8.

2 DESIGN PATTERNS

Design patterns can be described as a problem/context/solution triple (Buschmann et al., 2007; Gamma et al., 1995). ”A design pattern systematically names, motivates, and explains a general design that addresses a recurring design problem in object-oriented systems” (Gamma et al., 1995). Design patterns are abstract ideas that can be illustrated in different ways and that can be instantiated in many ways. They can be illustrated, for example, using class diagrams or using role models. Design patterns provide a common design vocabulary, a documentation and

Wiesauer A. and Sametinger J. (2009).
A SECURITY DESIGN PATTERN TAXONOMY BASED ON ATTACK PATTERNS - Findings of a Systematic Literature Review.
In *Proceedings of the International Conference on Security and Cryptography*, pages 387-394
DOI: 10.5220/0002232503870394

learning aid, an adjunct to existing methods, and a target for refactoring.

Gamma et al. first proposed a uniform description template which includes the name of the pattern, its intent and motivation, applicability structure and other patterns that are related to it (Gamma et al., 1995). Buschmann et al. have developed an extension to this template (Buschmann et al., 2007). It includes:

- Name:
  a name and a short summary of the pattern
- Also Known As:
  other names of the pattern
- Example:
  a real world example that demonstrates the purpose and the benefit of the pattern
- Context:
  situations in which the pattern may apply
- Problem:
  a description of the problem the pattern addresses including its associated forces
- Solution:
  a description of how the problem can be solved
- Variants:
  a reference to other patterns that are variants or specialization of this pattern
- Consequences:
  an outline of the benefits and potential tradeoffs of the pattern

Starting in 1997, design patterns had been used for security-specific issues, see (Yoder and Barcalow, 1997). They used a similar description scheme with a similar understanding of what a design pattern is. Subsequently, most authors have relied on the traditional definition of design patterns as a foundation.

Examples of security design patterns are:

- Secure Pipe Pattern
  Prevent eavesdropping and tampering of client transactions caused by man-in-the-middle attacks (Steel et al., 2005).
- Secure Logger Pattern
  Application events and related data must be securely logged. Log data should be secured from unauthorized access (Steel et al., 2005).
- Single Access Point Pattern
  Systems with external access should have a single point of access that grants or denies access after checking a client (Schumacher et al., 2006).
- Checkpointed System Pattern
  Design a system in a way that its state can be recovered and restored to a known valid state in case a component fails (Halkidis et al., 2004).
- Limited View Pattern
  Users should only see those parts of a system that they have access to (Yoder and Barcalow, 1997).

Additional patterns can, for example, be found at www.securitypatterns.org/patterns.html.

3 OBJECTIVES AND METHODOLOGY

According to Barnum and Sethi, only a minority of software architects and developers is well educated in security issues (Barnum and Sethi, 2006). It is doubtful that people with minor security knowledge will be able to apply security design patterns in a correct, an effective and an efficient manner. We need pattern selection criteria that are based on security requirements and that are applicable in practical contexts. This leads us to the following questions:

- How are security design patterns defined?
- Which different kinds of security design patterns have been proposed and how are they best described and documented?
- Which classification schemes for security design patterns exist?
- Have any quality criteria been suggested for evaluating security design patterns?

The application of systematic approaches for the assessment and the aggregation of research outcomes is needed in order to gain a balanced and objective summary for a particular research topic (Brereton et al., 2007). Conducting a systematic literature review is such an approach. According to Kitchenham (Kitchenham, 2004), systematic literature reviews aim at ”identifying, evaluating and interpreting all available research relevant to a particular research question, or topic area or phenomenon of interest”. When conducting such a review, research studies that are analyzed are called *primary studies*, the analysis itself is called *secondary study*. Fig. 1 shows the steps that make up a systematic literature review. The activities are grouped in three main phases: planning, conducting the review and reporting. For further details see (Brereton et al., 2007) and (Kitchenham, 2004).

We have followed these steps and have started by defining search queries and using various scientific search engines, e.g., computer.org, acm.org and springerlink.com. The search has led us to an ample number of papers as well as books, conference tracks and web sites. We have identified many definitions, quality criteria and taxonomies. These will be discussed in Sections 4, 5 and 6, respectively.

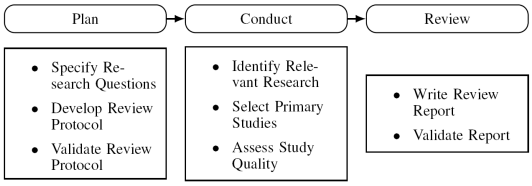

Figure 1: Literature review (Brereton et al., 2007).

4 DEFINITIONS

Yoder and Barcalow have been the first to apply design patterns to security issues (Yoder and Barcalow, 1997). They have used a similar description scheme to Gamma et al. with a similar understanding of design pattern definition. Schumacher has defined a security design pattern as a way of documenting proven solution to recurring problems in a well-structured manner (Schumacher, 2002). As patterns are written by experts, Schumacher emphasizes that they provide an effective way to learn from experts. This is especially important in the area of security, since there are only few software architects and designers that have experienced knowledge concerning security issues. The definition of security patterns has been further concretized: ”A pattern describes a particular recurring security problem that arises in specific contexts, and presents a well-proven generic solution for it” (Markus Schumacher, 2005). The solution part of a pattern consists of a set of interacting roles that can be arranged into multiple concrete design structures.

Additionally, the process to create the mentioned design structure is considered to be an important part of the solution. The motivation of using security patterns originates from the facts that patterns codify knowledge in an understandable way, that the representation of patterns is familiar to software developers and system engineers, and that patterns help to expand the security focus from low-level implementation to higher-level architectures (Markus Schumacher, 2005). The latter is especially important since around 50% of all security vulnerabilities arise from design flaws (McGraw, 2006).

Similarly, Fernandez and Yuan define security patterns as an aid for designers with little security knowledge (Fernandez and Yuan, 2007). Also, Hafiz and Johnson point out the reusability aspect of patterns (Hafiz and Johnson, 2006). They use the general term security pattern rather than security design pattern. However, they state that security patterns influence software design decisions, suggesting that they do understand them as security design patterns in our sense. They also mention the problem of managing security patterns and diagnose a need for a classification or taxonomy. Heyman et al. differ slightly as they first view a pattern as a ”well-known technique to package domain-independent knowledge and expertise in a reusable way” (Heyman et al., 2007). In contrast to other definitions, they explicitly accentuate domain-independence as a major characteristic of a security design pattern. The definition is substantiated by elements that a pattern should comprise: a description of a scoped problem and a constructive solution. They further outline that patterns cover different abstraction levels and that some patterns are too abstract to be considered as real patterns, e.g., roles patterns which suggest to use actors of use cases as a starting point when defining roles within a role-based access control system. They also propose a negative definition of security design patterns: Mere guidelines, high-level process activities or statements of security principles cannot be considered as patterns (Heyman et al., 2007). A pattern definition should further contain a ”known uses” element, which gives an indication of whether a pattern is real or not. By applying their positive and negative definition they analyzed 220 pattern. Only 55% of these were considered to be patterns, whereas 35% had been classified as guidelines and 15% as process activities.

Romanosky presents a catalogue of eight security design patterns and follows the definitions already discussed (Romanosky, 2001). He views security design patterns as a means for identifying and formulating all security practices needed in a given environment. High-level security requirements should be transformed into security policies and further into security procedures. Patterns assist in implementing security procedures into the software product. Slightly different, Steel et al. treat security design patterns as ”an abstraction of business problems that address a variety of security requirements and provide a solution to the problem” (Steel et al., 2005). This definitions express that security design patterns should be aligned with security requirements. Such patterns can appear as architectural patterns or as defensive design strategies that depict how secure code should be written according to a given design. Kienzle et al. have presented a security pattern repository and distinguish between structural patterns and procedural patterns (Kienzle et al., 2002). Structural patterns can be implemented in an application in the sense of security design patterns, but they can be applied at the architectural as well as at the implementation level. Procedural patterns aim at improving the process of developing security-critical software.

Yoshioka et al. have a broader understanding of security patterns (Yoshioka et al., 2008). They have carried out a survey on security patterns and

have classified their results by the software life-cycle phases. They present patterns for the requirements phase, the design and implementation phase, the testing phase, etc. For example, a *security needs pattern* in the requirements phase helps to identify assets of a system and information in a system that has to be considered as an asset. Yoshioka et al. do not only classify procedural aspects as patterns, e.g., guidelines for carrying out a certain activity, but also secure programming guidelines or refactoring measures, i.e., patterns in the implementation phase.

Horvath and Dörges criticize the usefulness of patterns in the security area (Horvath and Dörges, 2008). They outline that the concept of patterns has two major shortcoming in the area of security. On the one hand, patterns are only informal descriptions of what to do. On the other hand, patterns are often not suitable to describe complex architectures. This is important, because security has to be applied to the whole system. In their opinion, patterns cannot fulfill their expectations. They therefore suggest the application of more formal approaches like Petri nets.

We can summarize that there are differing definitions of the term security patterns. The focus of such patterns is set quite differently, i.e., procedural vs. structural and depending on the software life-cycle phase. The term security design pattern gives the most unanimous view of how a pattern is defined. The main benefit of a security design pattern is also seen uniformly as a mean for documenting knowledge and transforming experience to people with little security knowledge. This benefit is scrutinized by some authors because they see a lack on useful classifications and taxonomies as well as formalisms of the patterns. Also there is a difficulty of pattern application when being confronted with complex architectures.

We use the term security design pattern for proven solutions to security problems in a given context. These solutions have to contain constructive measures of how to design certain parts of a software system, i.e., security design patterns have to be directly reflected in the software design.

In contrast to the definitions, the scientific community mainly has a uniform opinion on how security patterns should be described and documented. Most authors stuck to the template proposed by (Buschmann et al., 2007) with only minor modifications, see (Fernandez and Pan, 2001; Fernandez et al., 2008; Kienzle and Elder, 2001; Romanosky, 2001; Schumacher et al., 2006; Yoder and Barcalow, 1997; Yoshioka et al., 2008). Some authors complement this template by graphical notations, e.g., UML class or sequence diagrams, see (Fernandez and Pan, 2001; Steel et al., 2005). A strategy item has additionally been added to the description template by (Steel et al., 2005). The strategy outlines different possibilities to implement and deploy a security pattern.

5 QUALITY CRITERIA

A pattern is a means to encapsulate knowledge about proven solutions and transfer (externalize) the expertise to others with less knowledge. Security design patterns should help to mitigate security risks by applying them to software designs. Therefore, a top-level quality aspect is whether a security design pattern actually helps to mitigate the risks, i.e., whether it effectively does what it pretends to do. A second quality aspect is whether a pattern indeed accomplishes the transfer of knowledge, i.e., whether it is documented precisely enough to be used in a practical context. Heyman et al. have carried out a study on the quality of security design patterns (Heyman et al., 2007). They conclude that patterns at first need a clear and appropriate description.

They have defined quality scores for each description element like problem description, solution, or consequences: 0 for ”not provided”, 1 for ”minimal” and 2 for ”satisfactory”. Each description element additionally has a certain weighting, e.g., problem description 19%, consequences 14.5%. For more details on the weighting see (Heyman et al., 2007). An overall quality indicator for security design patterns is obtained by using the quality score and its weighting. Using this quality indicator, it turned out that between 2001 and 2003 the number of patterns published was high but the quality was only mediocre. The quality level has increased in the following years together with a decline of the number of published patterns.

Halkidis et al. have performed a qualitative evaluation on security design patterns by a set of criteria (Halkidis et al., 2004):

- How well do they conform to the guiding principles of (Viega and McGraw, 2001)?
- How well do they help to deter a programmer from building a system that suffers from security holes like buffer overflows?
- How well does a specific security pattern respond to attacks described in the STRIDE model (Howard and Lipner, 2006)?

Viega and McGraw have introduced ten guiding principles for building secure software (Viega and McGraw, 2001). Examples are principle of least privilege and the principle of secure fail. The principle of least privilege states that any operation in a program should be carried out with the least privilege

necessary. The principle of secure fail means that a system should continue to work in secure mode in case of a failure. For three typical software development problems, Halkidis et al. have analyzed whether patterns help to avoid security holes (Halkidis et al., 2004). The problems were buffer overflows, poor access control mechanisms and race conditions. The STRIDE model groups possible attacks into different categories. STRIDE is an acronym for Spoofing identity, Tampering with data, Repudiation, Information disclosure, Denial of service, and Elevation of privileges. Halkidis et al. have analyzed a number of security design patterns and used an ordinal scale for their judgement, e.g., whether a pattern does fulfill one of the guiding principle or whether it negatively affects a principle. They have further based their judgement on whether a pattern ”possibly” helps to avoid an attack like a buffer overflow or whether it delivers protection or ”improved protection” against one of the STRIDE attacks.

Both approaches can serve as a useful guideline when selecting security patterns, but they are built upon a subjective basis. The authors do not state objective criteria for judging a description element of a pattern as ”minimal” or ”satisfactory”, or whether it ”possibly” helps to prevent specific attacks. Further, the weighting of the description elements is only based on heuristics and seems to be quite arbitrary.

We have not been able to identify studies that provide objective quality measures or even metrics for security design patterns.

6 TAXONOMIES

A huge number of security design patterns has been published. This makes it difficult to select patterns that are appropriate for a specific software system. Additionally, many patterns with similar intent and similar solution are published under different names by different authors. Most authors see the need for a classification or a taxonomy of patterns, see for example (Schumacher et al., 2006; Hafiz et al., 2007; Weiss and Mouratidis, 2008). A classification helps users to navigate through available patterns and also allows them to recognize similar patterns.

Authors have proposed classifications based on different criteria. Hafiz et al. created an overview of various approaches. Among others, they were able to identify the following classification criteria (Hafiz et al., 2007; Hafiz and Johnson, 2006):

- Applicability
  They differentiate between patterns for protected systems and patterns for available system. Patterns for protected systems deal with protecting resources against unauthorized use. Patterns for available systems deal with providing predictable and uninterrupted access to services and resources.
- Application Context
  ”Core security” includes patterns for security mechanisms within a system. ”Perimeter security” includes patterns that deal with authentication, authorization and security filtering at the system’s entry points. ”Exterior security” includes patterns that concern data transmission and communication protocols.
- Logical tiers
  Patterns are classified according to the logical tier of a system in which they are used, e.g., in the user interface layer or in the business logic layer.
- Security concepts
  ISO-13335 defines confidentiality, integrity, availability and accountability as key concepts of security. Patterns can be classified according to the key concept that they support.
- Viewpoints and interrogatives
  Patterns are classified according to different viewpoints, e.g., business architecture, integration architecture or application architecture, and to interrogatives, e.g., purpose (why?), data (what?) or function (how?). Business architecture covers business and management perspective of an application. Integration architecture deals with the integration of internal and external systems. The application architecture includes system and software elements.
- STRIDE
  Patterns are classified according to attacks from the STRIDE model, see Section 5.

The classification based on viewpoints and interrogatives originates from the Zachman framework and was developed by Microsoft (Trowbridge et al., 2004). Within one of the architectures, one can further differentiate between different stakeholders, e.g., software designers or implementers. This classification does not only comprise security design patterns in our sense, but also, for example, procedural patterns.

All classification schemes have several disadvantages. The security concept based classification, for example, has the major drawback that many patterns belong to more categories - the classification is not concrete enough (Hafiz et al., 2007). Classification based on logical tier brings order into the landscape of pattern, but it does not help in selecting appropriate patterns, since one already has to know where and

for which purpose a pattern should be applied. The tabular classification based on viewpoints and interrogatives delivers a good overview and allows users to navigate through different patterns, but it is not aligned with security requirements.

7 A NEW TAXONOMY BASED ON ATTACK PATTERNS

An architect or developer should be able to select appropriate patterns in order to fulfill security requirements given to her. Classification based on attacks of the STRIDE model is a first step towards this direction. The overall security requirement is to resist against certain attacks. When planning a secure system, one should elicit against which attacks the system has to resist and therefore which security requirements the system has to fulfill. Given this set of possible attacks, architects and developers should be able to select patterns. We will introduce a taxonomy based on attack patterns that enables architects and developers to select appropriate patterns according to possible attacks.

7.1 Attack Patterns

An attack is an act of carrying out an exploit of a vulnerability (Barnum and Sethi, 2006). An attack pattern is a blueprint for creating a kind of an attack - a ”blueprint for disaster” (Hoglund and McGraw, 2004). The motivation behind using attack patterns is the same as with security design patterns: If software developers have little knowledge in determining vulnerabilities and possible attacks, attack patterns should encapsulate and transfer this knowledge. Attack patterns are a means for teaching the software development community how to exploit software in reality and to implement adequate ways to avoid attacks (Barnum and Sethi, 2006). Barnum has developed a description and classification schema for attack pattern (Barnum, 2008). This schema includes:

- Name and unique ID of the attack pattern
- Related weaknesses and vulnerabilities
- Methods of attack and attack examples
- Describing and diagnosing information
- Related attack patterns

”Describing information” includes injection vector and activation zone. An injection vector describes the format of an input-driven attack and its playload, see (Hoglund and McGraw, 2004)). The activation zone is the area within the target where the payload gets executed, see (Hoglund and McGraw, 2004) ”Diagnosing information” includes techniques to probe a certain target to determine vulnerabilities and indicators for attacks, i.e., conditions that indicate that an attack is in progress or has occurred. The schema additionally includes an element ”related design patterns”, in order to specify design patterns that help to avoid a successful attack or that may lead to a successful attack. Barnum therefore sees a relationship between attack patterns and design patterns (Barnum, 2008). The MITRE Corporation hosts a website at capec.mitre.org that provides a publicly available catalog of attack patterns. The website is sponsored by the U.S. Department of Homeland Security and is technically assisted by Cigital Inc. The attack patterns in the catalog are documented according to the scheme described above. Additionally, the website provides a classification tree for attack patterns. It classifies attack patterns into categories like spoofing, resource depletion or exploitation of authentication. Figure 2 shows an excerpt of this classification tree.

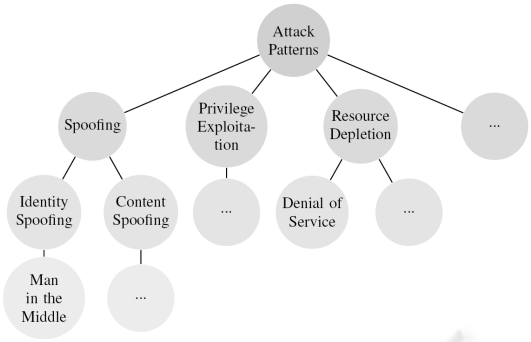

Figure 2: Attack Pattern Classification Tree (excerpt).

The attack pattern description schema explicitly includes related security design patterns. However, only a few attack patterns of the catalog actually mention design patterns.

7.2 Taxonomy

We have decided to align design patterns with attack patterns of the CAPEC catalog and to suggest a taxonomy based on the description of attack patterns and the purpose and intent of security design patterns. This taxonomy can be considered as a concretion of the design pattern classification based on the STRIDE model, since the attack patterns catalog together with its classification tree is a concretion thereof.

The advantage of such a taxonomy is the fact that users can easily see relevant security design patterns when identifying specific attack patterns. It further helps to identify similar design patterns. Security

design patterns in the same attack pattern category may be similar or even redundant. Thus, our taxonomy will help to rationalize the security design pattern landscape. We will provide a few examples to emphasize this fact.

The CAPEC catalogue defines a man-in-the-middle attack (attack pattern ID 94). This attack aims at the communication channel between a server and a client. Attackers try to eavesdrop data transmitted between server and client. The intent of the secure pipe pattern (Steel et al., 2005) and the secure channel pattern (Schumacher et al., 2006) aim at securing the communication channel between different entities. Both patterns are similar and help to prevent successful man-in-the-middle attacks. If designers fear a planned system might be susceptible to such attacks, the suggested taxonomy enables them to select appropriate patterns easily.

Another example is an SQL injection attack (attack pattern ID 66). This attack targets at software that creates SQL statements out of unchecked user input. When user input is not validated carefully, it may be possible to create user input by appending SQL command fragments that may cause unexpected harmful database queries like modification, disclosure or deletion of data. The interception validator pattern (Steel et al., 2005) aims at checking and validating user input and therefore helps to avoid SQL injection attacks. The client input filter pattern (Kienzle et al., 2002) follows the same purpose. Furthermore, attackers performing a dictionary-based password attack try to guess user passwords by using words in a dictionary (attack pattern ID 16). The account lockout pattern (Kienzle et al., 2002) helps to avoid such attacks by limiting the number of incorrect login attempts.

Applications must log events and actions in order to guarantee accountability and non-repudiation and for forensic purposes. When a target system does not properly the control the log file, an attacker might manipulate or forge it (Log injection-tampering-forging attack, pattern ID 93). This has an impact on the accountability and non-repudiation of actions and may lead to incorrect forensic analysis. The secure logger pattern (Steel et al., 2005) provides instructions for properly handling logging facilities.

7.3 Validation

In a first iteration, we have been able to assign some 40 security design patterns to attack patterns. So far, our assignments rely on semantic analysis of the attack pattern descriptions and on security design pattern descriptions. We will have to validate the correctness and usefulness of the classification, i.e., to test whether a security design pattern actually helps to deter from a successful attack described by an attack pattern. It is also necessary to check whether non-experts actually will be able to select appropriate patterns in specific situations.

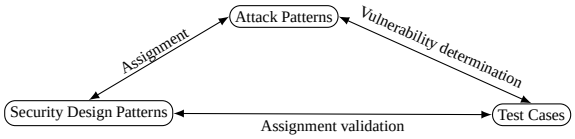

Figure 3: Assignment Validation Triplet.

We envision an approach for the validation of our assignments based on test cases, see Fig. 3. For each attack pattern one or more test cases should be defined. These test cases can be used to check whether the attack described by the pattern can be executed successfully on a particular software system. The assignment is valid if the application of the security design pattern prevents such an attack from being performed successfully, or at least increases the resistance of the system against such attacks.

Our goal is to have a list of attack patterns with corresponding security design patterns with test cases.

8 CONCLUSIONS

We have introduced design patterns and presented an overview of security design patterns. In the literature, there are many different security design pattern definitions and specifications. Also, different taxonomies for security design patterns are available. We argue that these taxonomies are not adequate for non-experts in order to select patterns in specific situations. We have therefore introduced a new taxonomy that is based on attack patterns.

In the literature, there are several approaches on how to determine the quality of security design patterns. But there is a lack on sufficiently impartial quality measures. The two most important aspects in this context are documentation and repeatability. First, security design patterns have to be documented precisely enough to let non-experts understand and apply them. Second, test cases are needed to let non-expert users find out whether the pattern had been applied correctly, i.e., attacks will indeed be prevented.

In the future, we plan several projects for the assignment validation as well as for a security pattern evaluation based on test cases. These projects additionally aim on determining whether non-experts are able to apply security design patterns or further operationalisation of security knowledge is required.

REFERENCES

Barnum, S. (2008). *Common Attack Pattern Enumeration and Classification (CAPEC) Schema Description*. Cigital Inc., http://capec.mitre.org/about/ documents.html.

Barnum, S. and Sethi, A. (2006). Introduction to attack patterns. Technical report, U.S. Dept. of Homeland Security, https://buildsecurityin.uscert.gov/daisy/bsi/articles/knowledge/attack/585-BSI.html.

Brereton, P., Kitchenham, B. A., Budgen, D., Turner, M., and Khalil, M. (2007). Lessons from applying the systematic literature review process within the software engineering domain. *Journal of Systems and Software*, 80(4):571–583.

Buschmann, F., Henney, K., and Schmidt, D. C. (2007). *Pattern-Oriented Software Architecture Volume 4: A Pattern Language for Distributed Computing*. Wiley & Sons.

Fernandez, E. B., Fonoage, M., VanHilst, M., and Marta, M. (2008). The secure three-tier architecture pattern. In *Proc. of International Conference on Complex, Intelligent and Software Intensive Systems*, pages 555–560, Los Alamitos, CA, USA. IEEE Computer Society.

Fernandez, E. B. and Pan, R. (2001). A pattern language for security models. In *Proceedings of PLoP 2001 Conference*.

Fernandez, E. B. and Yuan, X. (2007). Securing analysis patterns. In *ACM-SE 45: Proceedings of the 45th annual southeast regional conference*, pages 288–293, New York, NY, USA. ACM.

Gamma, E., Helm, R., Johnson, R., and Vlissides, J. (1995). *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley.

Hafiz, M., Adamczyk, P., and Johnson, R. E. (2007). Organizing security patterns. *IEEE Software*, 24(4):52–60.

Hafiz, M. and Johnson, R. E. (2006). Security patterns and their classification schemes.

Halkidis, S. T., Chatzigeorgiou, A., and Stephanides, G. (2004). A qualitative evaluation of security patterns. In *Proceedings of the 6th International Conference on Information and Communications Security (ICICS)*, pages 132–144, Malaga, Spain. Springer.

Heyman, T., Yskout, K., Scandariato, R., and Joosen, W. (2007). An analysis of the security patterns landscape. In *SESS '07: Proceedings of the Third International Workshop on Software Engineering for Secure Systems*, page 3, Washington, DC, USA. IEEE Computer Society.

Hoglund, G. and McGraw, G. (2004). *Exploiting Software - How to Break Code*. Addison Wesley.

Horvath, V. and Dörges, T. (2008). From security patterns to implementation using petri nets. In *SESS '08: Proceedings of the fourth international workshop on Software engineering for secure systems*, pages 17–24, New York, NY, USA. ACM.

Howard, M. and Lipner, S. (2006). *The Security Development Lifecycle*. Microsoft Press.

Kienzle, D. M. and Elder, M. C. (2001). Final technical report: Security patterns for web application development. Technical report, http://www.scrypt.net/čeler/securitypatterns/final

Kienzle, D. M., Elder, M. C., Tyree, D., and Edwards-Hewitt, J. (2002). Security patterns repository version 1.0. Technical report, http://www.scrypt.net/˜celer/securitypatterns/ repository.pdf.

Kitchenham, B. (2004). Procedures for undertaking systematic reviews. Technical report, Computer Science Department, Keele University (TR/SE-0401) and National ICT Australia Ltd (0400011T.1).

Markus Schumacher, Eduardo Fernandez-Buglioni, D. H. F. B. P. S. (2005). *Security Patterns. Integrating Security and Systems Engineering (Wiley Series in Software Design Patterns)*. Wiley & Sons.

McGraw, G. (2006). *Software Security: Building Security In*. Addison-Wesley.

Romanosky, S. (2001). Security design patterns. Technical report, http://www.cgisecurity.com/lib/ securityDesignPatterns.pdf.

Schumacher, M. (2002). Security patterns. *Informatik Spektrum*, Juni 2002:220–223.

Schumacher, M., Fernandez-Buglioni, E., Hybertson, D., Buschmann, F., and Sommerlad, P. (2006). *Security Patterns : Integrating Security and Systems Engineering (Wiley Software Patterns Series)*. John Wiley & Sons.

Steel, C., Nagappan, R., and Lai, R. (2005). *Core Security Patterns: Best Practices and Strategies for J2EE(TM), Web Services, and Identity Management*. Prentice Hall PTR.

Trowbridge, D., Cunningham, W., Evans, M., Brader, L., and Slater, P. (2004). *Describing the Enterprise Architectural Space*. Microsoft, http://msdn.microsoft.com/enus/library/ ms978655.aspx.

Viega, J. and McGraw, G. (2001). *Building Secure Software: How to Avoid Security Problems the Right Way*. Addison-Wesley Professional.

Weiss, M. and Mouratidis, H. (2008). Selecting security patterns that fulfill security requirements. In *IEEE International Conference on Requirements Engineering*, pages 169–172. IEEE Computer Society.

Yoder, J. and Barcalow, J. (1997). Architectural patterns for enabling application security. In *Proceedings of the 4th Conference on Patterns Language of Programming (PLoP'97)*.

Yoshioka, N., Washizaki, H., and Maruyama, K. (2008). A survey on security patterns. *Progress in Informatics*, (5):35–47.