# An Analysis of the Security Patterns Landscape

Thomas Heyman, Koen Yskout, Riccardo Scandariato, Wouter Joosen
DistriNet, Department of Computer Science, K.U.Leuven
Celestijnenlaan 200A, B-3001 Leuven, Belgium
first.lastname@cs.kuleuven.be

## Abstract

*Architectural and design patterns represent effective techniques to package expert knowledge in a reusable way. Over time, they have proven to be very successful in software engineering. Moreover, in the security discipline, a well-known principle calls for the use of standard, time-tested solutions rather than inventing ad-hoc solutions from scratch. Clearly, security patterns provide a way to adhere to this principle. However, their adoption does not live up to their potential. To understand the reasons, this paper analyzes an extensive set of published security patterns according to several dimensions and outlines the directions for improvement.*

## 1. Introduction

In the software engineering discipline, patterns represent a well-known technique to package domain-independent knowledge and expertise in a reusable way. Architectural and design patterns constitute solid solutions that can be employed out of the box by architects and designers in order to solve known, recurrent problems. A pattern provides three main advantages. First, the solution is known to be sound because it is time-tested. Second, benefits and drawbacks of a pattern are known in advance and they can be taken into account while sketching the solution. Third, patterns establish a common vocabulary that can ease communication between different stakeholders.

In the security discipline, a well-known principle calls for the use of community resources rather than inventing ad-hoc solutions from scratch [21]. For instance, creating a new encryption protocol is risky because of the likelihood of design flaws. Likewise, it is not advisable to implement a well-known protocol from scratch, because of possible coding flaws. Security patterns offer invaluable help in order to enforce this principle at the architectural and design level. First, design glitches can be avoided by applying well-known design solutions. Second, security patterns should include enough detailed information (down to the level of reference code), e.g., to help automate the implementation phase. In other words, security patterns are tools to provide additional guarantees that a software product is correct.

Security patterns have gained significant attention by the research community after the seminal work by Yoder and Barcalow [22]. The authors have surveyed the literature that has been published over the last ten years. Figure 1 (darker line) shows the publication trend of security patterns over this period. Using the Gartner's Hype Cycle terminology [2], it is evident from the graph that we passed the peak of inflated expectations followed by disillusionment. We are now in the "slope of enlightenment", where focused experimentation leads to a true understanding of the applicability, shortcomings and benefits of a technology.

The key issue is that no objective examination has been conducted so far to understand the reasons that hinder the adoption of security patterns. Although there is no lack of both security patterns and catalogs that collect them, security patterns have an inadequate reputation and this clearly hampers their adoption.

Patterns have proven to be successful in software engineering, e.g., the Gang of Four (GoF) patterns are extensively used in today's libraries and frameworks. There is no evident reason why the same should not happen for security patterns. Indeed, the authors strongly believe that they have high potential, especially as an instrument to bridge the knowledge gap between the design phase and secure code.

The contribution of this paper is to provide an overview of the current landscape of security patterns, identify shortcomings and suggest directions for improvement. For this work, the complete (to the authors' knowledge) set of security patterns that are available in literature has been analyzed. In figures, the authors surveyed about 220 patterns published in the period 1996–2006. Accordingly, the major drawbacks in existing literature are pinpointed by answering three main questions, as listed below. For each question, the directions for improvement are outlined constructively.

1. *Are all existing patterns really patterns?* A clear defi-

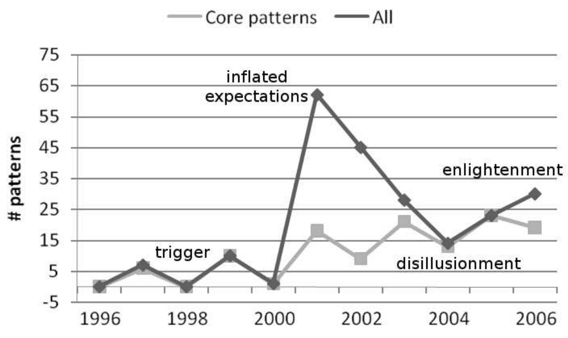

**Figure 1. Patterns by publication year.**

nition of what makes a security pattern is missing. As a consequence, the pattern landscape is heterogeneous in nature and contains several outliers. This paper provides a definition of what a security pattern should be and, accordingly, what it should not. We call this the *soundness dimension*.

2. *Are the patterns effective to build secure software?* Effectiveness is directly connected to quality and ease of use. In particular, this paper provides figures about how well the patterns are documented, as an overall indicator of their quality. We call this the *quality dimension*.

3. *Is the security spectrum of solutions covered appropriately by the patterns?* Security patterns are key instruments to implement security requirements. In turn, these requirements are linked to security objectives. In order to outline the shortcomings, this paper shows the distribution of security patterns over the different security objectives they try to fulfill. We call this the *coverage dimension*.

This paper is part of a larger project investigating the role played by security pattens in the construction of secure software, both from a technical and a methodological perspective [1]. In that context, some of the aforementioned questions have already been addressed in [23].

The rest of the paper is structured as follows. Section 2 discusses the related work and briefly introduces the main sources that have been used in this study. Sections 3, 4, and 5 analyze the literature according to the above-mentioned dimensions, respectively. Finally, Section 6 presents the conclusions and discusses ongoing and future work.

## 2. Related work

It is hard to cover all the sources that have been analyzed for this study due to space constraints. Only the major ones are mentioned. After the seminal paper by Yoder and Barcalow [22], several works on security patterns have been published at the latest editions of the PLoP (Pattern Languages of Programs) conference, the main venue for the software patterns research community [19, 16, 5, 15, 18]. Concerning pattern catalogues, Blakley and Heath collect an inventory of about fifteen security patterns [3]. In that work, the authors also provide a description of a system of patterns that distinguishes between availability and security concerns. The work by Steel et al. contains a very extensive collection of security patterns, specifically meant for the Java Enterprise platform [20]. A third systematic collection of about fifty security patterns is presented by Schumacher et al. [17]. The work approaches the use of patterns at several levels of abstraction, not limited to architectural and design patterns, but also including patterns for, e.g., performing risk assessment and mitigation. Finally, an analytical body of over thirty patterns can be found in earlier work by Kienzle et al. [9].

Previous work assessing the overall quality of security patterns is very limited, to the best of the authors' knowledge. Halkindis et al. perform a qualitative analysis of patterns contained in the Blakley and Heath inventory [7]. The evaluation criterion they use is the support provided by each pattern to the ten security principles by Viega and McGraw [21]. In a previous study, Konrad et al. apply the same evaluation criteria to the Yoder and Barcalow inventory [11]. Futhermore, Konrad also proposes different classification means to organize patterns, such as the nature of the patterns (creational, structural or behavioral, as defined in GoF) and their abstraction level (network, host, application). Similarly, Rosado et al. map out the relationships between security requirements and security patterns [14]. They also provide a classification into architectural and design patterns.

This work goes one step further by quantitatively assessing a larger set of patterns, according to the dimensions mentioned before.

## 3. What is a security pattern

In the first dimension of the analysis, the focus lies on screening the literature in order to identify appropriate security patterns, i.e., patterns that can be used constructively when architecting or designing an application.

### 3.1. Current state

It has been observed that "because of the popularity of design patterns in the software engineering community, the natural inclination is to assume that anything going by the name security patterns should be described using a UML diagram and include sample source code. While it is true that many interesting security patterns can be presented this way, there are many other important patterns (some procedural, some architectural) that do not fit within these constraints" [8]. We partly agree with the above argumentation, but some clearly stated boundaries must be identified in order to define the real nature of a security pattern.

Indeed, existing security patterns cover different levels of abstraction. Some patterns are too abstract to be considered actual patterns. For example, consider the ASSET VALUATION pattern [17], which suggests that without the determination of threats and vulnerabilities, an enterprise is unable to properly assess the risks posed to its assets. Therefore, financial value and impact to the business have to be evaluated for each asset. The above represents a process activity that belongs to the risk analysis phase, rather than a security pattern. A second example is the ROLES pattern [22]. Provided that one has to define the roles for a role-based access control system, the pattern suggests to use the actors of the use cases as a starting point. In this case, the example is closer to a guideline or a best practice, rather than a pattern. Conversely, some patterns describe very low-level mechanisms that are implementation strategies rather than pattern-based solutions, e.g., encryption key management protocols and implementation alternatives to manage session data. By observing similar examples it is possible to come to a *negative* definition, i.e., what a security pattern should not be. Patterns that either provide guidelines, describe high-level process activities, formulate security objectives (e.g., "ensure that access to system resources is restricted to known partners") or state security principles (e.g., "use the least privilege") are of little use for architecting and designing secure systems. The same holds for low-level programming techniques.

It is not that easy to provide a *positive* definition of what makes a pattern. Coplien defines a pattern as "a relationship between a certain context, a certain system of forces which occurs repeatedly in that context, and a certain spatial configuration which allows these forces to resolve themselves" [4]. The definition mentions two important parts: the problem statement (which includes the forces) and the solution part.

As far as the first part of the above definition is concerned, the problem has to be well scoped and limited. Nonetheless, several examples can be observed in which the tackled problem is not at the right level of decomposition, i.e., it is too broad. Note that this criterion of evaluation is independent, to a large extent, from the quality of the documentation of the pattern itself. The limiting factor is not the detail of the description, but rather resides in the granularity of the problem at hand.

The second part of the definition refers to the solution. Often, security patterns describe what should be the outcome of a solution (the what) and not the strategy to achieve the solution itself (the how). As an example, the ROLE BASED ACCESS pattern [10] suggests to create roles that conform to user responsibilities, and assign them an appropriate set of privileges. However, no practical guidance to achieve this goal is provided. In many cases, this is due to the fact that the solution is not implementable as a software artifact, but depends on a designer state of mind (e.g., least privilege) or on a designer subordinate activity (e.g., see the above mentioned ROLES pattern). Contrary to the example, a pattern should describe the solution in terms of constructable software entities and should describe the design steps or rules for constructing those entities, as also observed in [12]. For instance, software entities constituting the solution should be arranged in a well-determined spatial and/or behavioral configuration, and they should interact externally with abstract participants. Algorithmic guidelines about how to map participants of a pattern (a.k.a. roles) to the elements of an actual design should be devisable as well.

A pattern is also a *proven* solution to a problem. That is, the solution proposed by the pattern has been employed extensively in real world applications and survived the test of time. The "known uses" of the solution is the final convincing argument to classify it as a pattern. While this survey currently only considers examples of use provided in the pattern description itself, future work may include analyzing open source software to measure the level of adoption of the patterns.

By applying both the negative definition (patterns vs. concepts and mechanisms) and the positive definition (scoped problem with a time-proven, constructive solution), the authors tried to screen the literature on an objective basis. As a result, only 55% of the 220 analyzed patterns are classified as core patterns, 35% are guidelines and principles, and 10% are process activities.

## 3.2. Possible improvements

The security patterns landscape would greatly benefit from an overall rationalization. As an illustration, it took the authors a full week to skim through the descriptions of the whole set of surveyed patterns. Clearly, this effort can not be expected from every designer whenever a security challenge is encountered. This study provides a starting point and *identifies the candidates for reducing the set of patterns*, based on the definition provided above. To this aim, particular attention must be given to patterns published at the peak of the enthusiastic publication phase. Indeed, Figure 1 compares the overall publication trend (dark line in the upper part) with the trend of core patterns (light line). Major discrepancies can be noted during the publication peak, meaning that high heterogeneity is present during that period. As a side note, the two curves are diverging again at

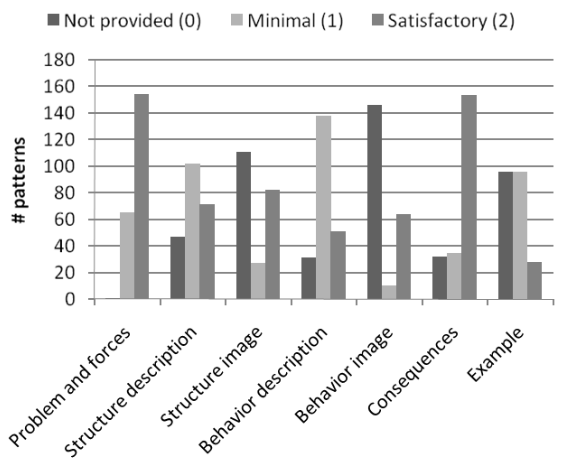

**Figure 2. Scores of the description elements**

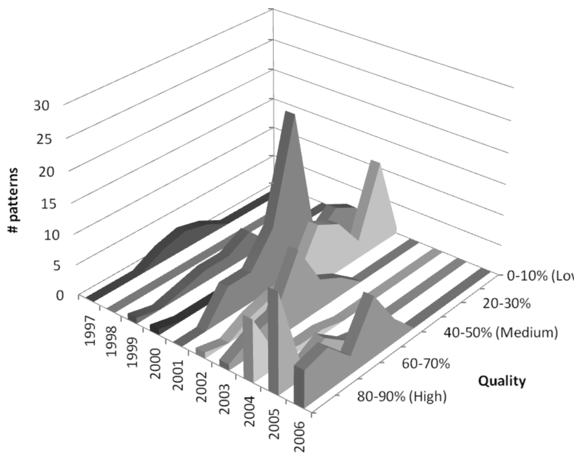

**Figure 3. Number of published patterns in function of the quality and year of publication.**

present time.

Finally, the simplification process should also *remove significant overlaps in published patterns*. Often, patterns from different sources have different names, but similar content. This observation comes from the authors' experience in working with patterns. A quantitative analysis hereof is a very large effort and is still work in progress.

## 4. Quality analysis of the patterns landscape

In the second dimension of the analysis, the study focuses on the quality of patterns. In this section, the authors only refer to the *quality of the documentation*; whether or not this documentation actually describes a good and useful pattern is covered in Section 3, and is not taken into account here.

### 4.1. Current state

A security pattern without a clear and appropriate description is of limited use, if any. Moreover, poorly described patterns will hamper the adoption of security patterns in general.

Given a good pattern description, it should be obvious to determine whether a pattern is applicable to a particular situation and how the actual instantiation of the pattern should be done. According to the authors, and mainly corresponding to [3], a good description of a security pattern should, at least, contain the following elements:

- The *problem and forces* that describe the context from which the pattern emerged. This part should clarify if the pattern is applicable to a specific situation.
- The *solution* offered by the pattern, comprising both its structure and behavior. The *structure* depicts (at least textually, and preferably also graphically) the different actors that play a role in the pattern, and the relationships among them. The intended *behavior* of the actors defines (again in a textual and graphical way) the collaborations among the different actors.
- The *consequences* of implementing the pattern, highlighting both its strengths and weaknesses.
- An *example* of the pattern in an easily understood software setting. This example makes the pattern concrete and tangible. It also helps to clarify the pattern's intent, its use and its consequences to the user.

The encountered patterns are assigned scores on all of the above-mentioned description elements. A score of 0 is assigned for 'not provided', 1 for 'provided, but minimal' (i.e., the provided information is illustrative, but not sufficiently constructive) and 2 for 'provided and satisfactory' (i.e., the provided information aids the developer in understanding and implementing the pattern). For example, in the REPLICATED SYSTEM pattern [3], the solution consists of implementing a workload manager in order to distribute load among different replicas of the system. However, insufficient guidance on how to implement this manager is provided. Therefore, the description of the solution structure is assigned a score of 1. On the other hand, the STANDBY pattern [3] is assigned a score of 2 for the solution structure, as it refers to additional design patterns to help the developer in implementing the solution.

In Figure 2, the results of the grading are shown. The graph clearly shows that in most pattern descriptions, the

| | | |
|---|---|---|
| **Problem** | | **19%** |
| Problem description | 19.0% | |
| **Solution** | | **57%** |
| Structure description | 19.0% | |
| Behavior description | 19.0% | |
| Structure image | 9.5% | |
| Behavior image | 9.5% | |
| **Others** | | **24%** |
| Example | 9.5% | |
| Consequences | 14.5% | |

**Table 1. Weights assigned to the various elements of description.**

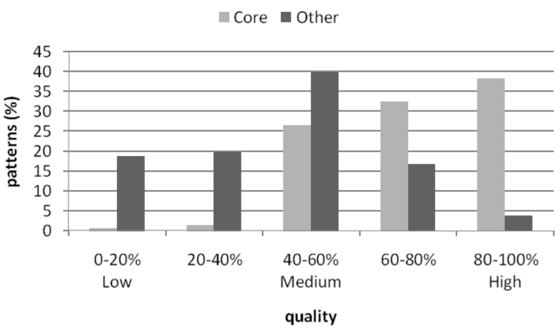

**Figure 4. Percentage of patterns per quality level.**

'problem and forces' section is satisfactory. However, the quality of the actual solution description for these problems (which are given by the structural and behavioral text and images) is not equally high. The lack of well-described solutions indicate that a substantial amount of the analyzed patterns are more problem-oriented than solution-oriented.

For each pattern, an overall quality indicator is obtained by calculating a weighted average of the individual scores on each description element. The weights used for the different elements are given in Table 1. The problem description and the structural and behavioral description of the pattern are deemed equally important and get the highest weight. The addition of an image is helpful, but may be omitted if a good and clear textual description of the solution is given. Hence the images only get half the weight of the description. This same weight is assigned to the example, while the consequences are valued in between the example and each of the images.

In Figure 3, the number of published patterns is plotted in function of the publication year and the average quality of the pattern description (calculated using the previously described method). The figure shows that, while the number of patterns between 2001 and 2003 (the inflated expectations phase) is quite high, the quality of their description is, on average, only mediocre. During the following years, the quality shifts to the better percentages.

Figure 4 presents the quality distribution for both the core patterns (as discussed in Section 3) and the others. It shows that the quality of the non-core patterns (dark color) is centered around the medium to low percentages, while in general the core patterns (light color) have higher ratings. Note that, as mentioned before, the quality in this section only denotes the quality of the *description* of the pattern; no score on the actual contents or value of the pattern itself was given. Patterns with a low rating may therefore be as valuable as patterns with a high one, even though they are documented in a less rigorous way, and vice versa. However, this graph could indicate a correlation between the soundness and documentation quality of a pattern. That is, the elements considered important are well-suited to describe core patterns, in contrast to guidelines, best practices, process activities and others.

### 4.2. Possible improvements

The average documentation quality of security patterns is improving. This shows that, as the discipline matures, the community acknowledges the importance of an adequate description. This is also illustrated by the increase over time of the length of pattern descriptions, which has evolved from about 500 to 1500 words on average. In general, quality and length appear to be correlated, even though high variance exists. To consistently achieve a good quality level and ensure that the description is complete, *the use of a standardized template for describing the patterns* would be beneficial. Possible templates are already outlined and used in [8, 20, 17, 23].

For instance, patterns that successfully describe a significant problem are quite common. Further, since many of these belong to the core patterns category, they are relevant. However, because they lack a sufficiently detailed description of the solution, their current form limits their usability. Therefore, there is a need for substantial reworking in order to extend the way they are articulated. A template provides the foundation to efficiently realize this goal.

## 5. Coverage analysis of the patterns landscape

In the third dimension of the analysis, the distribution of security patterns over the security objectives is assessed, in order to outline possible gaps.

### 5.1 Current state

Security as a non-functional requirement can be refined in different security objectives, such as confidentiality and

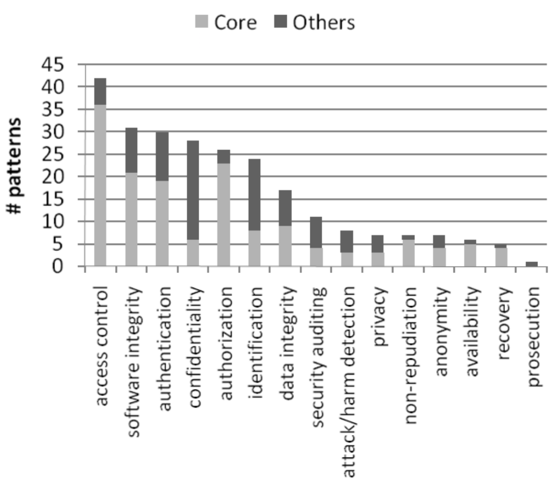

**Figure 5. Amount of patterns per security objective.**

integrity. Accordingly, security patterns can be classified by assigning them to one or more security objectives that they intend to improve. The description is based on the list of security objectives found in [6]. This list includes objectives on access control, attack and harm detection, integrity, non-repudiation and privacy. Note that, since in this work security is considered from a software engineering viewpoint, physical protection, hardware integrity and personnel integrity are out of the scope of this survey. For completeness, availability is also added.

Figure 5 shows the amount of patterns per security objective. Each bar shows the total amount of patterns, while the light gray part shows the fraction of core patterns. Note that, since patterns may belong to multiple objectives, the total amount exceeds the number of indexed patterns.

More established security objectives, such as integrity and confidentiality, are well represented by the total body of security patterns. By far, the most well-represented objective is access control and its sub-objectives, i.e. identification, authentication and authorization. Objectives that recently gained importance—such as privacy, non-repudiation and anonymity—are not covered as extensively.

It must be noted that objectives such as confidentiality, identification and data integrity, although covered by a substantial amount of patterns, only contain a limited number of core patterns—most patterns for these objectives are comprised of guidelines and best practices. This is especially true for confidentiality and identification. While the non-core patterns for these objectives provide a valuable exploration of the problem space, they do not offer concrete assistance to the architects and designers of a secure system.

## 5.2 Possible improvements

Improving the coverage of the security landscape is possible in two dimensions.

First, *additional security patterns for lesser covered security objectives* (as shown at the right-hand side of Figure 5) can be introduced. According to [4], patterns capture solutions with a track record, not theories or speculation. For these patterns to emerge, the state of the practice of their selected security areas needs to be mature enough. For example, in privacy management (as one of the more recent security fields), one of the emerging patterns is the grouping of personal information (or attributes) around *pseudonyms*. This approach is commonly used by identity management systems (IDMs). By representing the user by one or more pseudonyms (aliases), the IDM retains control on disclosing personal information of a user to third parties. Furthermore, third parties can not deduce that multiple sets of personal information belong to the same person (represented by multiple pseudonyms), or multiple persons, without cooperation of the IDM. This architectural approach is proving its value as a privacy management strategy, and is already implemented in multiple identity management systems such as Liberty and Shibboleth. Eventually, a time-tested implementation of this approach may be captured in a security pattern.

The second dimension for improvement is highly related to the quality issue described in Section 4, and lies in *converting guidelines to core patterns*. Some of the more stable security objectives such as confidentiality or identification are mainly covered by guidelines (the dark parts of the bars in Figure 5). If proven methods of implementing solutions for these guidelines are available, it is possible to incorporate them in the pattern to form a core pattern (thereby converting the dark parts of the bars to light gray). For example, the SECURE COMMUNICATION pattern [13] states that secrets should be transmitted encrypted to prevent eavesdropping, without providing concrete guidance on implementing it. In order to come to a core pattern, this guideline could be extended with a description of how to implement a secure connection in Java by leveraging the *SSLSocketFactory*.

## 6. Conclusions and future work

This paper presents an analysis of existing literature on security patterns. The study draws conclusions according to three main dimensions:

1. The number of patterns needs to be reduced by removing candidates that do not fit into the definition of a pattern.

2. The quality of documentation for existing patterns needs to be improved. Furthermore, patterns should

provide better assistance in bridging the gap from design to implementation, e.g., by providing reference code and by supporting automation.

3. Some areas of security, like privacy and non-repudiation, have limited support in the pattern landscape.

To overcome the observed shortcomings, the following research challenges must be addressed. Some of them have already been tackled by the authors, at least in an initial shape [23], others are subject to current and future work.

*Construction of an inventory.* The large amount of security patterns and their heterogeneity make the selection of "the right pattern for the job" a daunting task. By following the definition of what makes a security pattern in Section 3, an inventory can be built that includes patterns at the same level of abstraction. Also, significant overlap between existing security patterns exists and should be removed. Furthermore, the inventory can be extended with new patterns to improve the coverage of security objectives, as outlined in Section 5.

*Adoption of a documentation template.* By adopting a common template, the overall quality of patterns would greatly benefit, as shown in Section 4.

*Definition of a system of patterns.* Security patterns should not be isolated entities. They may interact with one another, in either beneficial or detrimental ways. In order to select the best pattern for a given design challenge, these inter-pattern relationships must be made explicit to the user.

*Articulating a methodology for pattern-based secure software design.* Patterns should be organized so that their selection during different design phases is simplified. Furthermore, the possible effects of the selection of a pattern should be made explicit to facilitate trade-off decisions during design. Finally, a guiding methodology is necessary to assist developers in bridging the gap between requirement definition, design, and beyond.

## References

[1] Software security for network applications – SoBeNet. http://sobenet.cs.kuleuven.be.

[2] Understanding hype cycles. http://www.gartner.com.

[3] B. Blakley and C. Heath. *Security Design Patterns*. The Open Group Security Forum, 2004.

[4] J. O. Coplien. A pattern definition. www.hillside.net/patterns/definition.html.

[5] N. Delessy-Gassant, E. B. Fernandez, S. Rajput, and M. M. Larrondo-Petrie. Patterns for application firewalls. In *PLoP*, 2004.

[6] D. Firesmith. Specifying reusable security requirements. *Journal of Object Technology*, 3(1), January 2004.

[7] S. T. Halkidis, A. Chatzigeorgiou, and G. Stephanides. A qualitative evaluation of security patterns. Malaga, Spain, October 2004. International Conference on Information and Communications Security (ICICS).

[8] D. Kienzle, M. Elder, D. Tyree, and J. Edwards-Hewitt. Security patterns template and tutorial. February 2002.

[9] D. M. Kienzle, M. C. Elder, D. Tyree, and J. Edwards-Hewitt. Security patterns repository, version 1.0, 2006.

[10] S. R. Kodituwakku, P. Bertok, and L. Zhao. Aplrac: A pattern language for designing and implementing role-based access control. In *EuroPLoP*, 2001.

[11] S. Konrad, B. H. Cheng, and L. A. Campbell. Using security patterns to model and analyze security requirements. In *IEEE International Conference on Requirements Engineering (RE)*, Monterey Bay, CA, USA, 2003.

[12] D. Lea. Checklist for writing great patterns. www.hillside.net/patterns/writing/writingpatterns.htm.

[13] S. Lehtonen and J. Pärssinen. A pattern language for cryptographic key management. In *EuroPLoP*, 2002.

[14] D. G. Rosado, C. Gutiérrez, E. Fernández-Medina, and M. Piattini. Security patterns related to security requirements. In *4th International Workshop on Security in Information Systems (WOSIS)*, 2006.

[15] T. Saridakis. Design patterns for fault containment. In *PLoP*, 2003.

[16] M. Schumacher. Firewall patterns. In *EuroPLoP*, 2003.

[17] M. Schumacher, E. Fernandez-Buglioni, D. Hybertson, F. Buschmann, and P. Sommerlad. *Security Patterns*. Wiley, 2006.

[18] P. Sommerlad. Reverse proxy patterns. In *PLoP*, 2003.

[19] K. E. Sorensen. Session patterns. In *PLoP*, 2002.

[20] C. Steel, R. Nagappan, and R. Lai. *Core Security Patterns: Best Practices and Strategies for J2EE, Web Services, and Identity Management*. Prentice Hall, 2005.

[21] J. Viega and G. McGraw. *Building Secure Software: How to Avoid Security Problems the Right Way*. Addison-Wesley, 2002.

[22] J. Yoder and J. Barcalow. Architectural patterns for enabling application security. In *PLoP*, 1997.

[23] K. Yskout, T. Heyman, R. Scandariato, and W. Joosen. An inventory of security patterns. Technical Report CW-469, Katholieke Universiteit Leuven, Department of Computer Science, 2006.