

Security patterns: 10 years later

Koen Yskout Thomas Heyman
Riccardo Scandariato Wouter Joosen

Report CW 514, April 2008



Katholieke Universiteit Leuven
Department of Computer Science
Celestijnenlaan 200A – B-3001 Heverlee (Belgium)

Security patterns: 10 years later

Koen Yskout *Thomas Heyman*
Riccardo Scandariato *Wouter Joosen*

Report CW 514, April 2008

Department of Computer Science, K.U.Leuven

Abstract

Reusing time-tested solutions rather than inventing ad-hoc quick fixes is a well-known security principle. Architectural and design patterns represent proven techniques to package knowledge from software engineering experts in a reusable format. More importantly, the solution proposed by a pattern is known to be sound because it is time-tested—its strengths, weaknesses and possible drawbacks are known in advance. Therefore, in software security engineering, security patterns have been considered to be a very promising means to increase the quality of secure design and make security more accessible to software engineers. However, their adoption does not live up to their potential. To understand why this is so, this paper analyzes the literature of security patterns published over the last ten years and outlines existing gaps.

Keywords : Secure software, Security patterns, Software engineering.

1 Introduction

In the software engineering discipline, patterns represent a well-known technique to package domain-independent knowledge and expertise in a reusable fashion. 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 at development time, e.g., to trade-off among alternative design solutions. Third, patterns establish a common vocabulary that can ease communication between different stakeholders. As such, it may come as no surprise that, almost fifteen years after the publication of the first pattern catalog, design patterns have proven to be successful in software engineering [Gam06]. For instance, the Gang of Four patterns [GHJV94] are used extensively in today's frameworks and products.

In the software security discipline, a recognized security principle is the reuse of community resources to avoid reinventing ad-hoc solutions from scratch [VM02], for two reasons. First of all, inventing a new solution schema (e.g., a new encryption protocol) is risky because of the likelihood of design flaws. Likewise, it is not advisable to re-implement a well-known solution from scratch, because of possible development flaws. Security patterns offer invaluable help in order to enforce this security principle at the architectural and design level, as they encapsulate security expert knowledge in a reusable format. 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) to help automate the implementation phase. In other words, security patterns are means to provide additional assurance that a software artifact is correct.

It is important to note that not every solution that has proven its value and has been reused over time necessarily constitutes a pattern. For instance, the use of a specific encryption protocol implementation, such as OpenSSL, is hardly a pattern, even though such technology has been reused by a large number of projects. However, the abstraction capturing the concept and the rationale of properly instantiating and using a secure communication channel in a two-party communication is likely a pattern. Indeed, the SECURE PIPE pattern [SNL05] has emerged to capture such abstraction. Furthermore, a pattern should be able to answer typical design questions such as “how does this solution impact other aspects of my systems?” For instance, a valid question in the example of the SECURE PIPE would be, “how does a Secure Pipe interact with the auditing infrastructure?”

Security patterns have gained significant attention by the research community, in terms of publications, after the seminal work by Yoder and Barcalow [YB97]. Similarly to design patterns, they have been around for over ten years now and the number of known security patterns is significant. This work has inventorized 218 patterns contained in 38 sources (books, papers, and technical reports) published in the period 1996–2006. Some concepts and methods captured by existing security patterns are well-known and extensively used by practitioners. For example, consider the praxis of enforcing a single point of access to applications in order to reduce the exposed attack surface, or the use

of input filters to prevent injections.

However, it is remarkable that the adoption of security patterns at large is lagging behind, especially when compared to the vast success of design patterns. Despite the extensive literature, security patterns still have an inadequate reputation in the security community. These observations spark the first research question addressed here. This work investigates the reasons that hinder the adoption of security patterns. To this aim, the complete (to the authors' knowledge) set of security patterns that are available in literature is considered and a quantitative analysis is performed according to four major dimensions. The sources that have been used in this study are mentioned in the bibliography and are marked as such. To the best of the authors' knowledge, such an extensive and precise evaluation has not been performed so far. The scope of this analysis is limited to solutions that are intentionally packaged in the format of a pattern, and referred to as being a security pattern by the authors of the respective sources. Nevertheless, the extent of the analyzed sources is non-trivial.

The main aspects under investigation in this study are:

1. *Are there enough patterns?* The first dimension of this analysis investigates whether the existing patterns cover the spectrum of known security problems sufficiently. While the publication of additional patterns does not necessarily improve their adoption, it is possible that the limited adoption of security patterns is due to an insufficient coverage of the security domain. In security, it is quite common to categorize problems according to security objectives (e.g., CIAA – confidentiality, integrity, availability, accountability). Therefore, the distribution of security patterns over the different security objectives they try to fulfill is analyzed. Other categorizations are possible, e.g., according to the abstraction level of the solution (networking, operating system, and so on). However, the chosen categorization is better suited to spot possible gaps with respect to the problem domain. Nonetheless, a categorization of the patterns according to their level of abstraction is indeed useful and is adopted later on in the paper, when improvements to the patterns are suggested.
2. *Are all patterns constructible software solutions?* A clear definition of what makes a security pattern is missing. As a consequence, the pattern landscape is heterogeneous in nature and contains several outliers. A heterogeneous set of patterns can be confusing for the user and may generate frustration during the selection process. This work takes the standpoint of the software developer and provides an operational definition of a security pattern. Accordingly, the paper identifies the patterns that comply with that definition, which are dubbed “core” patterns.
3. *Are the patterns properly documented?* The effectiveness of a pattern to solve a problem is directly connected to its 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.
4. *Are the patterns useful in practice?* It is not hard to provide excellent documentation of a useless pattern. Therefore, the documented “real-life” applications of each pattern should be counted. For instance, for the Gang of Four book, the authors give at least two “known uses” of each pattern.

These are not toy examples, but uses of the pattern in real-life systems, which are strong evidence of its usefulness. To this aim, the “known uses” mentioned in the security pattern documents are analyzed.

By answering the previous questions, the first part of this paper provides an overview of the landscape of security patterns, identifies shortcomings and suggests directions for improvement. The main conclusions can be summarized as follows.

1. The limited adoption is not due to an insufficient coverage of the problem domain. However, the coverage is not uniform.
2. Only about half of the published patterns are core assets from the perspective of a software engineer. Furthermore, in this reduced set there is still significant overlap.
3. The quality of documentation for existing patterns needs to be significantly improved.
4. Only in a few cases, the patterns make a good case in showing the usefulness of the proposed solution in real-life systems.

Overall, this paper aims at providing the reader the following insights: (1) a better understanding of the real potential of security patterns as of now; and (2) a framework for reducing the number of patterns to the core of strictly relevant ones for construction purposes. This paper is part of a broader project investigating the role played by security patterns in the construction of secure software, both from a technical and a methodological perspective. In that respect, improving usability is a key enabler. Additional results can be found in [YHSJ06], [HYSJ07] and [SYHJ08].

The remainder of the paper is structured as follows. In Sections 2 through 5 the results of the quantitative analysis are presented, according to the four above-mentioned dimensions. Section 6 summarizes the identified gaps and addresses the directions for improvement. Section 7 discusses the related work and Section 8 draws overall conclusions.

2 Coverage of security problems

Security, as a software quality, can be further refined into several security objectives. In the first dimension of the analysis, the distribution of security patterns over the security objectives is assessed, in order to outline possible gaps. Indeed, the limited adoption could be due to an inadequate coverage of the security problems spectrum. Accordingly, security patterns can be classified by assigning them to one or more security objectives that they intend to fulfill. However, given the high number of published patterns, no major surprises are expected. It is foreseeable that all major security objectives are covered.

2.1 Method

The list of objectives used in this paper is an extension of the well-known CIA triad. According to Avizienis et al. [ALRL04], confidentiality, integrity, and availability (CIA) are defined as:

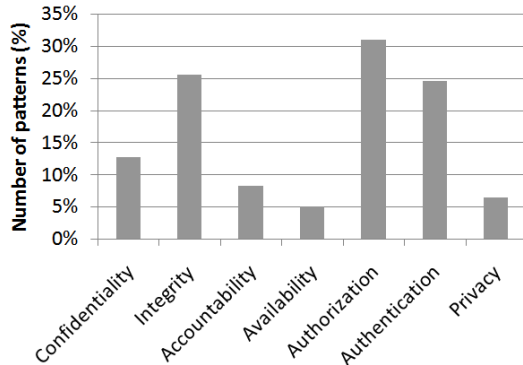


Figure 1: Patterns per security objective.

confidentiality: the absence of unauthorized disclosure of information;

integrity: the absence of improper system alterations;

availability: the readiness for correct service.

Next to the CIA triad, accountability and privacy are often considered.

accountability: the ability to hold users accountable for their actions;

privacy: the ability of an individual or group to control the flow of information about themselves.

In most cases, authentication and authorization are needed as two important support functions to realize the above objectives. They are defined as follows:

identification and authentication: the act of establishing and confirming the digital identity of a principal (person, process, device);

authorization: the act of defining and enforcing privileges to principals.

From a methodological perspective, the objectives are often present as keywords in the pattern’s description. In less straightforward cases, the assignment underwent additional discussion and voting within the authors’ team. Note that a pattern can be associated to more than one objective.

2.2 Results

Figure 1 shows the distribution of the security patterns per security objective. Each bar shows the percentage of patterns assigned to the corresponding objective, out of the total set of 218. Note that, since patterns may belong to multiple objectives, the total exceeds 100%.

In summary, the coverage shows to be adequate but not uniform. More established security objectives, such as integrity, are well represented by the total body of security patterns. By far, authentication and authorization are the best represented. Objectives that recently gained importance—such as privacy, which includes anonymity—are not covered as extensively. Similarly, security

problems that are closer to the dependability area, such as availability, are little covered. This is possibly due to an insufficient knowledge transfer between the two communities.

3 Core security patterns

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

It is not that easy to define a pattern. According to Coplien, it is “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” [Cop]. This definition mentions two important parts: the problem statement (which includes the forces) and the solution part. In order to be useful, the problem statement should be scoped (i.e., of the right granularity) and relevant to the intended user of the pattern. Considering the focus of this work, this implies that the problem statement should be relevant to the software engineer. Patterns that do not fulfill this requirement are said to be *out of scope*. An example of such a pattern is the NATURE OF SAFEGUARDS pattern, from [EH03]¹. Without a solution that resolves the conflicting forces, the pattern is simply a restatement of a security *objective*. This is also the case for patterns whose problem statement is too broad to be adequately resolved. An example is the KNOWN PARTNERS pattern from [SFBH⁺06]. This does not imply that the mere presence of a relevant problem statement and a solution is sufficient for the pattern to be useful to the software engineer. Two other criteria to distinguish between core and non-core patterns are whether the solution is constructible and of a correct level of abstraction.

As far as the constructibility of the solution by the software engineer is concerned, Kienzle et al. [KETE02] observe the following: “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 pattern 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.” The authors agree with the above argumentation, but some clearly stated boundaries must be identified in order to define the scope of this study in a clear-cut fashion. Indeed, existing security patterns cover different levels of abstraction, but not all levels are helpful in terms of constructibility, which is one of the main concerns of this work. 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 [KBZ01] 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., adhere to the principle of least privilege) or on a designer subordinate activity (e.g., identify

¹ This example has already been recognized as a non-pattern by the research community and, for instance, it is cited as a valuable principle (instead of a pattern) in [SFBH⁺06].

assets to be protected). An example of this is the ROLES pattern [YB97], which suggests to use the actors of the use cases as a starting point when defining the roles for a role-based access control system.

As such, some patterns are too abstract (i.e., not implementable by a software engineer) to be considered core patterns. In general, these non-constructable patterns can be subdivided in two categories. First, consider for example the ASSET VALUATION pattern [SFBH⁺06]. This pattern 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 an implementable solution within the scope of the software engineer. Second, consider the above-mentioned ROLES pattern. In this case, the example is closer to a *guideline* or a best practice, rather than a core pattern.

Conversely, some patterns describe very low-level mechanisms that are implementation *techniques* or *algorithms and protocols* rather than pattern-based solutions. Examples of these are encryption key management protocols (for instance, SESSION KEY EXCHANGE WITH PUBLIC KEYS) and implementation alternatives to manage session data (for instance, KEEP SESSION DATA IN THE CLIENT). 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 [Lea]. 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. As described by Eden and Kazman [EK03], the proposed solution should also be *intensional* in nature, i.e., there should be infinitely-many possible instances of the solution specification. In other words, the solution should contain participants of the pattern (also known as roles) that can be mapped onto software elements. It is this criterion of intensionality that distinguishes architectural and design solutions from code-level solutions. Guidelines on how to map these roles to the elements of an actual design should be devisable as well.

For the remainder of this work, a core pattern is defined as a security pattern with a relevant problem description and a constructible, intensional solution which consists of software entities, arranged in a well-determined spatial and/or behavioral configuration, that interact externally with abstract participants.

3.1 Method

The authors screened the available literature by applying the definition of a core pattern provided in the previous section. In order to make this definition operational and improve the objectivity of the categorization approach, a decision chart was used. This chart, shown in Figure 2, contains questions that allow independent reviewers to categorize security patterns according to the patterns' description. In order to additionally reduce the level of subjectivity inherent to the categorization, the process was repeated independently by three reviewers and decisions were based on majority. In most cases, the assignment of a pattern to a category was unanimous. A short discussion about the cases where voting was required is provided later on.

The categorization process consists of six questions that correspond to the criteria which a core pattern should have. The process starts at question Q1.

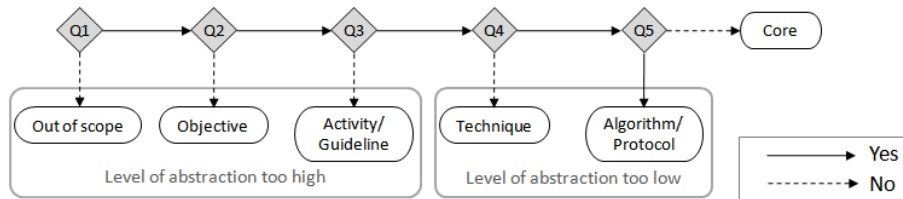


Figure 2: Classifying patterns according to nature.

The questions are (in order):

- Q1** *Is the problem relevant?* Is the described problem relevant in the scope of secure software development from a technical perspective, and not, for instance, a business or management problem? In general, if the problem is not encountered while developing or deploying an application, it is not considered relevant.
- Q2** *Does the pattern offer a concrete solution?* Does the description provide more than a very high-level overview of a possible solution? Patterns that are limited to a problem statement and do not provide an actual solution do not help the application developer, and are mostly rewordings of security objectives.
- Q3** *Is the solution constructible as a software entity?* Is it clear how the described solution can be instantiated as, or integrated in, a software element? If the solution is not implementable, the pattern is not a software security pattern. In the case that the solution is not constructible as a software entity, two cases remain. If the application of the described solution is limited to a specific moment or period within the development process, then the pattern represents an activity. If not, the pattern is a guideline. However, the exact distinction between an activity and a guideline is not important for this work.
- Q4** *Is the solution intensional?* Is it possible to create an infinite number of distinct instances of the solution? More specific, does the solution contain roles that can be mapped to various software entities that vary from one instantiation to another? If the solution is not intensional, then the pattern represents a technique.
- Q5** *Is the solution limited to a behavioral description?* Is there an inherent lack of a structural description in the described solution? If so, the pattern represents an algorithm or protocol. If not, the pattern is a core pattern.

3.2 Results

The results of the classification process are summarized in Figure 3. In total, 38 sources containing 218 security patterns have been analyzed. Half of the indexed patterns are classified as core patterns, about 20% are guidelines and activities, another 20% are algorithms, protocols or techniques. The remaining 10% is either out of scope, or represents an objective and does not contain a

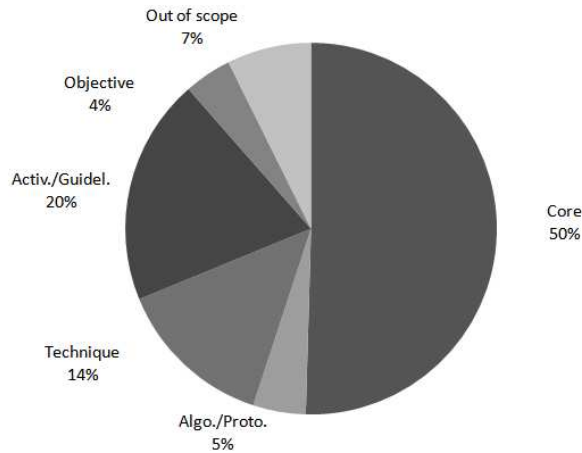


Figure 3: Patterns by abstraction level.

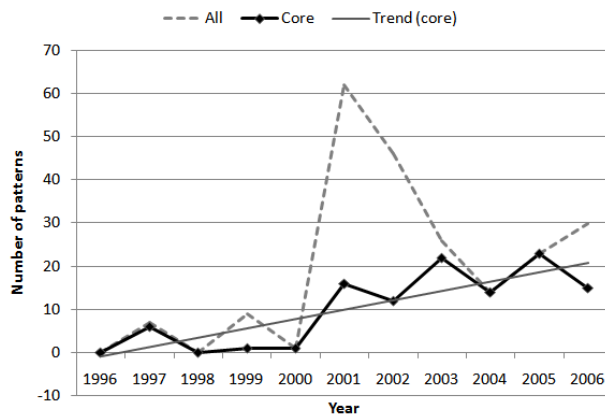


Figure 4: Published core patterns.

sufficient solution. Of the initial 38 sources, only 28 contain core patterns. From the viewpoint of the software engineer, these numbers constitute a substantial reduction: the amount of sources to be analyzed is reduced by one fourth, while the amount of patterns to be considered is halved.

Figure 4 contains the number of published core patterns per year over the last ten years. Note that the amount of published core patterns per year is increasing over time. This can be explained by the reasoning that, as the domain of computer security matures, more and more proven solutions are preserved in a domain-independent pattern format.

In the figure, the overall curve of published patterns (including non-core) is represented as a dashed line. It is worthy to highlight that the core pattern trend is not impacted by the publication peak that occurred during the period 2000–2003 (see the remarkable divergence of the two curves during that period). These observations appear to corroborate the validity of the operational definition of

‘core pattern’.

Finally, as a remark on the objectivity of the provided decision chart, the reviewers only had to vote on the categorization of 12 patterns (5.5% of all patterns) as being either a core pattern, or a non-core one. All other patterns (206) were unanimously assigned to one of both sets. Other points of minor disparity were the distinction between a technique or an algorithm/protocol (9 conflicts of opinion, or 4.1% of the total amount of patterns), a technique or a guideline (another 9 conflicts, or 4.1%) and a guideline or an objective (8 conflicts, or 3.7%). For the other categories, the reviewers agreed unanimously on more than 97% of the patterns under consideration. These numbers show that the definitions used for the different categories are crisp enough to avoid major uncertainty, and hence instrumental to weed out subjectivity.

4 Quality of documentation

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

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 Blakley et al. [BHmoTOGSF04], 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 instantiating 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 software engineer.

4.1 Method

The analyzed patterns are assigned scores (\mathcal{S}) 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

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.

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 [BHmoTOGSF04], the solution proposes to implement a workload manager in order to distribute the 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 [BHmoTOGSF04] is assigned a score of 2 for the same element, as it refers to additional design patterns to help the developer in implementing the solution.

In order to obtain an overall indicator of a pattern documentation quality, the following formula is used:

$$Quality = \sum_i \mathcal{W}_i \frac{\mathcal{S}_i}{max}$$

The quality is a normalized weighted sum over the scores of all description elements, expressed as a percentage (where 100% indicates a perfect score on all elements). In the equation, max is the maximum score for an element (i.e., 2). The weights (\mathcal{W}) are given in Table 1. The rationale for assigning the weights is as follows. 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.

4.2 Results

Figure 5 shows the results of the grading per description element. The graph clearly shows that the ‘problem and forces’ section is always provided and, for most patterns, 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, which can be cumbersome from the perspective of a software engineer.

Figure 6 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

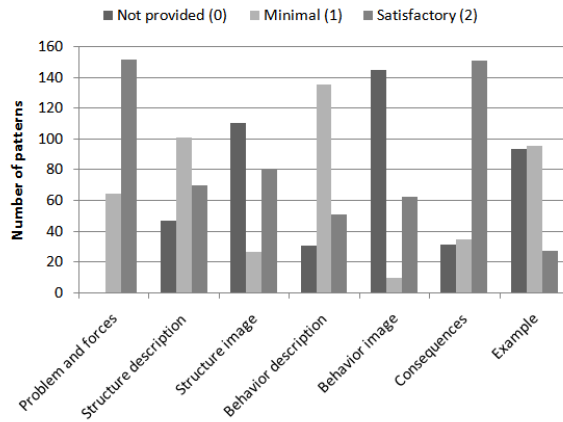


Figure 5: Scores of the description elements

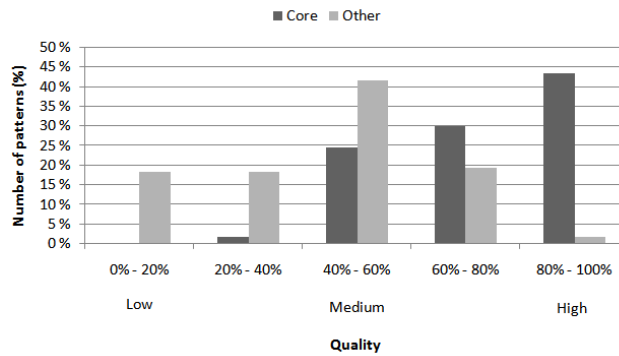


Figure 6: Percentage of patterns per quality level.

patterns (light color) is centered around the medium to low percentages, while in general the core patterns (dark 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 core patterns and a good description. That is, the elements considered important are well-suited to describe core patterns, in contrast to guidelines, techniques, process activities and others.

In Figure 7, the number of published patterns is plotted in function of both the publication year and the average quality of the pattern description. The figure shows that, while the number of patterns between 2001 and 2003 is quite high, the quality of their description is, on average, only mediocre. During the following years, the quality improves. This improvement can be explained by the establishment of an active security patterns community, stimulating discussions on what is important for a good security pattern.

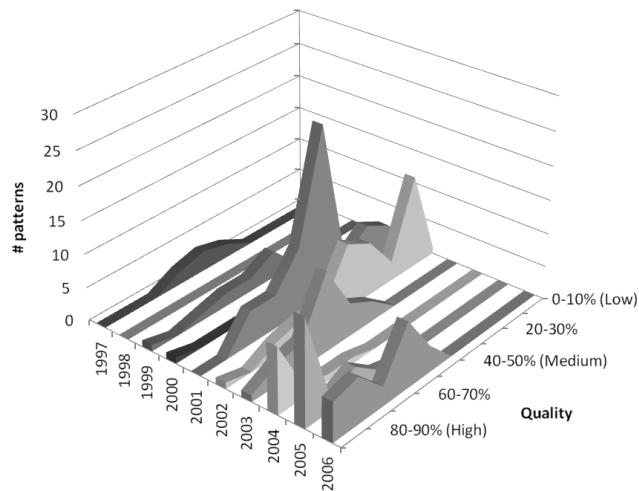


Figure 7: Number of published patterns in function of the quality and year of publication.

Another interesting observation can be made by comparing the various types of sources of security patterns, as presented in Figure 8. Four types of sources are distinguished, i.e., books, PLoP conferences, other workshops (e.g., EuroPLoP), and other (unreviewed) documents. The graph shows the average quality of each analyzed source (the black diamond), together with the minimum and maximum score (represented by the bars) of the patterns in that source. Also, for each source type, the average of the average scores is plotted (using a dashed line).

As a first observation, published security pattern books have the highest average score. This is most likely due to the rigorous review process for a book. Also, books often collect patterns that have been discussed extensively, which could explain the increase in documentation quality. For example, the Security Patterns book [SFBH⁺06] is the accumulation of numerous discussions between members of the security patterns community.

Further, conference and workshop papers that contain a single pattern (denoted with an asterisk in the figure) have higher scores than most of the sources collecting multiple patterns. A likely cause for this observation is the possibility that a publication with a single pattern will focus more on the adequate description of that single pattern, whereas a collection of patterns might place more focus on the relations between the patterns and less on the pattern descriptions themselves.

Finally, the PLoP conferences have a higher average than other workshops (e.g., EuroPLoP) and other, unreviewed publications.

5 Known uses

The fourth and final dimension of the analysis concerns the known uses of the security patterns. Since a pattern is a *proven* solution to a problem, the solution proposed by the pattern should have been employed extensively in real world applications and survived the test of time. Therefore, similarly to the patterns in

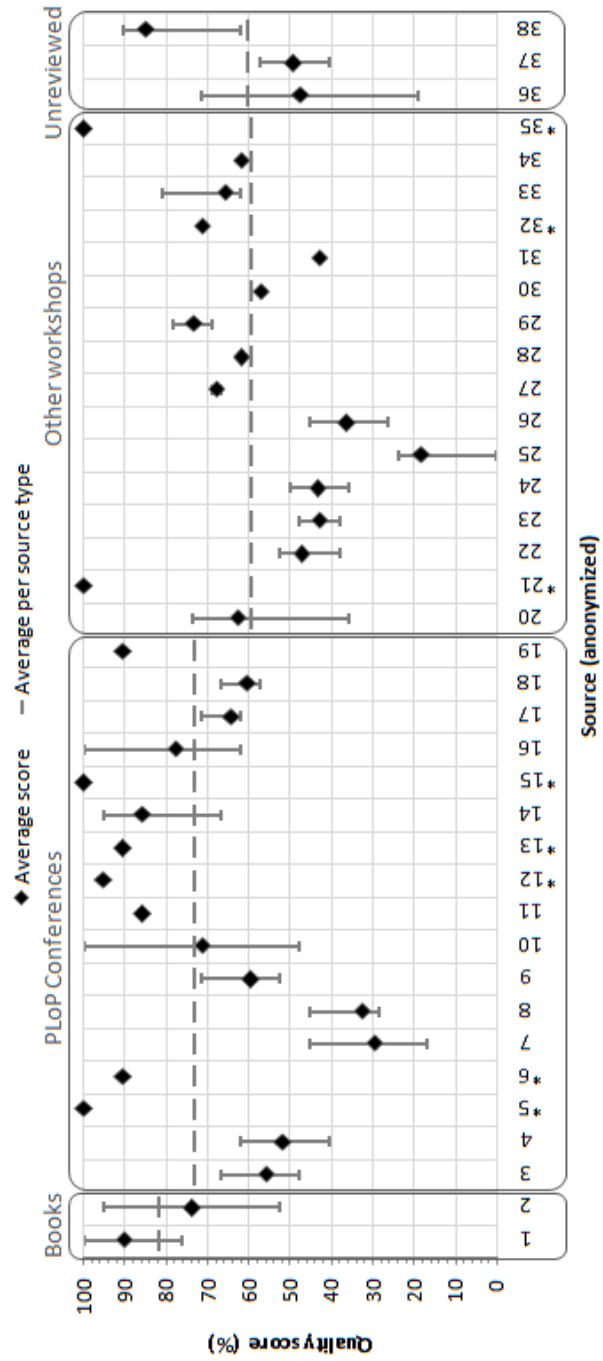


Figure 8: Average (diamond), minimum and maximum (bars) quality score per (anonymized) source, and average quality per source type (dashed line).

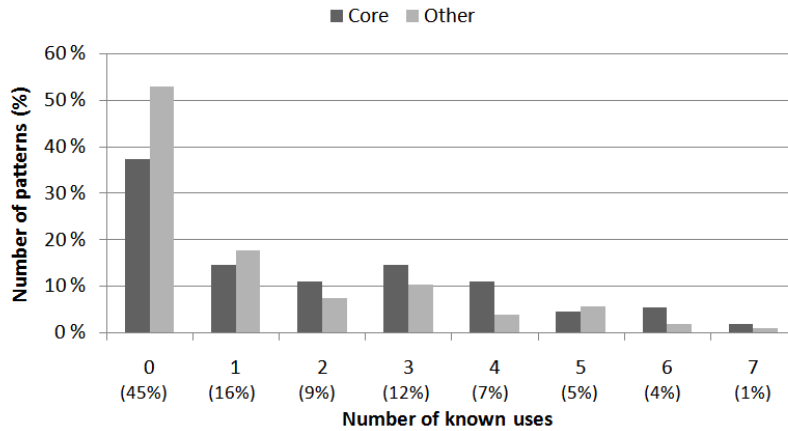


Figure 9: Number of patterns with the given number of known uses.

the Gang of Four book [GHJV94], a pattern description is supposed to mention some of its actual instantiations. Whether the security patterns actually live up to this expectation is analyzed in this section.

5.1 Method

The number of known uses mentioned in the pattern description is assessed, again, by means of three independent reviews. A known use is only counted if it:

1. contains a reference that is specific enough so that it can be studied if necessary;
2. is an instantiation in a real-world system (i.e., not in a research project); and
3. is not a system built by the author(s) of the pattern (for reasons of objectivity).

5.2 Results

The results of the analysis are shown in Figure 9. The percentages below the horizontal axis denote the total amount of patterns (both core and non-core) with the given number of known uses. The most apparent outcome is the lack of known uses for a large percentage of patterns. This is especially true for the non-core patterns, of which more than 50% do not contain any known uses at all. Surprisingly, a correlation between the core patterns and a higher number of known uses is not reflected by the results, contrary to what might be expected. The graph shows that, in general, there are relatively more core than non-core patterns on the right-hand side of the graph (i.e., with two or more known uses). However, this difference is too small to be significant.

Although the focus of the work presented here is on analyzing the body of security pattern publications, an interesting extension would be to measure the

actual occurrences of these patterns in real-life cases. This is by no means a simple feat, and is revisited in Section 8.

6 Improvements

Out of the observations that were presented in the previous sections, several improvements can be mentioned in order to better the usability and adoption of security patterns.

The security patterns landscape would greatly benefit from an overall rationalization and simplification. 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 in Section 3. To this aim, particular attention must be given to patterns published during the publication peak shown in Figure 4. Indeed, major discrepancies can be noted during this peak, meaning that high heterogeneity is present during that period.

Most importantly, the simplification process should also *remove significant overlaps in published patterns*. Often, patterns from different sources have different names, but similar (sometimes overlapping) content. Also, while a number of pattern inventories exist in which overlaps are removed (see, a.o., [BHmoTOGSF04, SFBH⁺06, KETEH01]), these inventories often overlap themselves. This observation comes from the authors' experience in working with the patterns.

Provided that the landscape is first simplified according to the above-mentioned suggestions, there are two remaining dimensions for improving the coverage of the security problems.

The first improvement can be made by *converting guidelines to core patterns*. Some of the more stable security objectives such as confidentiality are mainly covered by guidelines, i.e., patterns that successfully describe a significant problem but fail in providing a constructable solution. If proven methods to implement solutions for these guidelines are, or become, available, it is possible to incorporate them in the pattern to transform the guideline into a core pattern. Conversely, it is possible to *convert algorithms and protocols to core patterns* as well. Again, consider the OpenSSL-library. While this is clearly an implementation of a protocol, if the proper usage of this library is documented with a clear problem statement, conflicting forces in the problem domain and guidelines on how to instantiate the library to resolve these forces, the usage of OpenSSL can be abstracted into a security pattern.

Second, *additional security patterns for lesser covered security objectives* can be introduced. However, it is important to keep the lesson learned during the analysis of known uses (in Section 5) in mind. Indeed, according to [Cop], 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 such as Liberty [lib] and Shibboleth [shi]. Eventually, a time-tested implementation of this approach may be captured in a security pattern.

A final remark concerns the improvement of the documentation quality. The average documentation quality of security patterns is already 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. The length of the description of a pattern is not necessarily a quality indicator—indeed, sometimes shorter is better. However, care must be taken that the description of the pattern provides sufficient information to the practitioner. In general, quality and length appear to be correlated, even though high variance exists. Nevertheless, there is a need for substantial reworking in order to extend the way the patterns are articulated, as they often lack a sufficiently detailed description of the solution. This clearly limits their usability. 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 [KETEHO2, SNL05, SFBH⁺06, YHSJ06].

7 Related work

It is hard to cover all the sources that have been analyzed for this study in depth, due to space constraints. Only the major ones are mentioned here; a complete list of references can be found in Appendix A. After the seminal paper by Yoder and Barcalow [YB97], 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. Concerning pattern catalogues, Blakley and Heath collect an inventory of about fifteen security patterns [BHmoTOGSF04]. 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 [SNL05]. A third systematic collection of about fifty security patterns is presented by Schumacher et al. [SFBH⁺06]. 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. [KETEHO1].

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 [HCS04]. The evaluation criterion they use is the support provided by each pattern to the ten security principles by Viega and McGraw [VM02]. In a previous study, Konrad et al. apply the same evaluation criteria to the Yoder and Barcalow inventory [KCC03]. Furthermore, Konrad also proposes different classification means to organize patterns, such as the nature of the patterns (creational, structural or behavioral, as defined in the Gang of Four [GHJV94]) and their abstraction level (network, host, application). Similarly, Rosado et al. map out the relationships

between security requirements and security patterns [RGFMP06]. 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.

8 Conclusions

This paper presents an analysis of existing literature on security patterns. The study draws conclusions that can be generalized as follows:

- The pattern landscape is extremely heterogeneous and this possibly leads to misconceptions about the real efficacy of security patterns.
- In particular, the number of patterns is overwhelming and better means of classification are needed to increase usability.
- The quality of documentation for existing patterns needs to be improved, as well as the real world examples presented and the known uses. Among others, a pattern should be more specific on how it should be implemented and on its applicability in a given context.

In order to enable the adoption security patterns, the usability of security patterns has to be improved. Indeed, a catalog of patterns in which it is hard to find the right item hampers the adoption of any methodology using those patterns. To counter this, this work contributes by rationalizing the existing patterns set. The set is made *homogeneous* by grouping the security patterns of the same ‘nature’, i.e., guidelines, objectives, protocols, and so on. For the group of patterns of greater relevance to the software engineer (i.e., the ‘core patterns’), further classification is necessary. In [SYHJ08], the core patterns are therefore connected to the requirements space by linking each pattern to one or more *security objectives*, since requirements are usually formulated in terms of these objectives. Core patterns are also positioned in the right phase of the development process, like architecture, design, and deployment. The end result is a classification framework that is imposed over the the patterns catalog, in order to facilitate the selection process for the software engineer. That is, it becomes easier to answer the question, “Which pattern helps me to reach a certain security objective?”, because (a) it is clearer in which direction to look (i.e., following the objectives decomposition) and (b) the set of candidate solutions to be scrutinized is smaller, more homogeneous and more suitable for the intended stakeholders.

This work provides a solid base for improving the usability of the security patterns landscape, which can be extended by addressing the following research challenges.

Identifying real-world usages. Analyzing software systems in order to identify which security patterns (if any) were used. Next to identifying recurring solutions that might be considered for creating new patterns, identifying the cases in which a security pattern is intentionally implemented by the developers might give additional insights in the usability and quality of these security patterns. Furthermore, in the case of open-source projects, the source code of the project provides a valuable illustration of how the pattern can be instantiated.

Reducing the complexity. The core patterns still contain redundancy, as there are partially overlapping patterns that cover the same solution. Initial work in this direction has produced promising results.

Supporting the selection of a compatible set 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. The possible effects of the selection of a pattern should be made explicit to facilitate trade-off decisions during design on the basis of feature interaction. Furthermore, the methodology should be integrated with risk management frameworks, which represent a cornerstone of secure development.

Some of the above challenges have already been tackled by the authors. [YHSJ06] describes an initial proposition for a structured inventory of patterns, which supports the selection of a compatible set of patterns. [SYHJ08] proposes a methodology for pattern-based secure software design. Others are subject to current and future work.

Acknowledgment

This work is partially funded by the SoBeNet project – Software Security for Network Applications, an SBO-project of the Flemish government (see <http://sobenet.cs.kuleuven.be>).

A Patterns description

For each pattern mentioned in the text, a short description, as well as a reference to the original source of the pattern, is provided.

ASSET VALUATION [SFBH⁺06]

Asset valuation helps you to determine the overall importance an enterprise places on the assets it owns and controls. Loss or compromise of such assets may result in anything from hard costs, such as fines and fees, to soft costs due to loss of market share and consumer confidence.

KEEP SESSION DATA IN THE CLIENT [Sor02]

Session specific data has to be stored in between requests, and made available to the code handling a request. Therefore, keep the session specific data in the client. Transfer all or the necessary subset of it to the server along with each request.

KNOWN PARTNERS [SFBH⁺06]

Restrict access to known partners that must use secure authentication measures.

NATURE OF SAFEGUARDS [EH03]

In everyday life there are always alternative ways to achieve the goal of security. These alternatives might not only differ in complexity or effect, but also in nature. This pattern provides a comprehensive and complete set

of these natures that can be used in a structured approach to information security.

REPLICATED SYSTEM [BHmoTOGSF04]

Structure a system which allows provision of service from multiple points of presence, and recovery in case of failure of one or more components or links.

ROLE BASED ACCESS [KBZ01]

Direct access to information objects is technically possible but may lead to misuse of organizational resources. You want to control user access to information objects so users are permitted to invoke only operations explicitly corresponding to their responsibilities.

ROLES [YB97]

Create one or more role objects that define the permissions and access rights that groups of users have.

SECURE PIPE [SNL05]

Use a secure pipe to guarantee the integrity and confidentiality of data sent over a wire.

SESSION KEY EXCHANGE WITH PUBLIC KEYS [LP02]

Use public key cryptography to securely deliver session keys between two parties.

STANDBY [BHmoTOGSF04]

Structure a system so that the service provided by one component can be resumed from a different component.

Additional sources for security patterns that were considered for this work, besides the ones already referenced in this appendix, are [DGFRLP04], [KETE01], [MF06a], [Sch03], [Sar03], [Sch02], [Rom01], [Som03], [MGS03], [FW03], [FLBDdVF99], [MF06b], [Wei06], [FSLP06], [FLPS⁺03], [DL03], [LP01], [FS03], [HLF00], [Fer02], [SLPF05], [Haf06], [RHCF06], [EH02], [Rom02], [RCB⁺02], [SR01], [FP01], [BRD98], and [DF05].

References

- [ALRL04] Algirdas Avizienis, Jean-Claude Laprie, Brian Randell, and Carl Landwehr. Basic concepts and taxonomy of dependable and secure computing. *IEEE Transactions on Dependable and Secure Computing*, 01(1):11–33, 2004.
- [BHmoTOGSF04] Bob Blakley, Craig Heath, and members of The Open Group Security Forum. Security design patterns. <http://www.opengroup.org/security/gsp.htm>, 2004. *pattern source*.
- [BRD98] Alexandre M. Braga, Cecilia M. F. Rubira, and Ricardo Dahab. Tropic: A pattern language for cryptographic software. In *The 5th Pattern Languages of Programming Conference (PLoP 1998)*, 1998. *pattern source*.

- [Cop] James O. Coplien. A pattern definition. <http://www.hillside.net/patterns/definition.html>.
- [DF05] Nelly Delessy and Eduardo B. Fernandez. Patterns for the extensible access control markup language. In *Pattern Languages of Programs Conference (PLoP 2005)*, 2005. *pattern source*.
- [DGFRLP04] Nelly Delessy-Gassant, Eduardo B. Fernandez, Sajeed Rajput, and Maria M. Larrondo-Petrie. Patterns for application firewalls. In *Pattern Languages of Programs Conference (PLoP 2004)*, 2004. *pattern source*.
- [DL03] Paul Dyson and Andy Longshaw. Patterns for managing internet-technology systems. In *The 8th European Conference on Pattern Languages of Programs (EuroPLoP 2003)*, Germany, 2003. *pattern source*.
- [EH02] Ben Elsinga and Aaldert Hofman. Control the actor-based access rights. In *The 8th European Conference on Pattern Languages of Programs (EuroPLoP 2003)*, Germany, 2002. *pattern source*.
- [EH03] Ben Elsinga and Aaldert Hofman. Security taxonomy pattern language. In *The 8th European Conference on Pattern Languages of Programs (EuroPLoP 2003)*, Germany, 2003. *pattern source*.
- [EK03] Amnon H. Eden and Rick Kazman. Architecture, design, implementation. In *ICSE '03: Proceedings of the 25th International Conference on Software Engineering*, pages 149–159, Washington, DC, USA, 2003. IEEE Computer Society.
- [Fer02] Eduardo B. Fernandez. Patterns for operating system access control. In *Pattern Languages of Programs Conference (PLoP 2002)*, 2002. *pattern source*.
- [FLBDdVF99] Jr. F. Lee Brown, James DiVietri, Graziella Diaz de Villegas, and Eduardo B. Fernandez. The authenticator pattern. In *Pattern Language of Programs Conference (PLoP 1999)*, August 1999. *pattern source*.
- [FLPS⁺03] Eduardo B. Fernandez, Maria M. Larrondo-Petrie, Naeem Seliya, Nelly Delessy, and Angela Herzberg. A pattern language for firewalls. In *The 10th Conference on Pattern Languages of Programs (PLoP 2003)*, 2003. *pattern source*.
- [FP01] Eduardo B. Fernandez and Rouyi Pan. A pattern language for security models. In *The 8th Conference on Pattern Languages of Programs (PLoP 2001)*, 2001. *pattern source*.
- [FS03] Eduardo B. Fernandez and John C. Sinibaldi. More patterns for operating system access control. In *The 8th European Conference on Pattern Languages of Programs (EuroPLoP 2003)*, Germany, 2003. *pattern source*.

- [FSLP06] Eduardo B. Fernandez, Tami Sorgente, and Maria M. Larrondo-Petrie. Even more patterns for secure operating systems. In *The Conference on Pattern Languages of Programs (PLoP 2006)*, 2006. *pattern source*.
- [FW03] Eduardo B. Fernandez and Reghu Warriar. Remote authenticator/authorizer. In *Pattern Languages of Programs Conference (PLoP 2003)*, 2003. *pattern source*.
- [Gam06] Erich Gamma. Design patterns – 15 years later. In *European Conference on Object-Oriented Programming (ECOOP)*, Nantes, France, July 2006.
- [GHJV94] Erich Gamma, Richard Helm, Ralph Johnson, and John M. Vlissides. *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison Wesley Professional, 1994.
- [Haf06] Munawar Hafiz. A collection of privacy design patterns. In *The 13th Conference on Patterns Language of Programming (PLoP 2006)*, 2006. *pattern source*.
- [HCS04] Spyros T. Halkidis, Alexander Chatzigeorgiou, and George Stephanides. A qualitative evaluation of security patterns. In *International Conference on Information and Communications Security (ICICS)*, Malaga, Spain, October 2004.
- [HLF00] Viviane Hays, Marc Loutrel, and Eduardo B. Fernandez. The object filter and access control framework. In *Pattern Languages of Programs Conference (PLoP 2000)*, 2000. *pattern source*.
- [HYSJ07] Thomas Heyman, Koen Yskout, Riccardo Scandariato, and Wouter Joosen. An analysis of the security patterns landscape. In *The 3rd International Workshop on Software Engineering for Secure Systems (SESS07)*, 2007.
- [KBZ01] Saluka R. Kodituwakku, Peter Bertok, and Liping Zhao. Aprac: A pattern language for designing and implementing role-based access control. In *European Conference on Pattern Languages of Programs (EuroPLoP 2001)*, 2001. *pattern source*.
- [KCC03] Sascha Konrad, Betty H.C. Cheng, and Laura A. Campbell. Using security patterns to model and analyze security requirements. In *IEEE International Conference on Requirements Engineering (RE)*, Monterey Bay, CA, USA, 2003.
- [KETE01] Darrell M. Kienzle, Matthew C. Elder, David Tyree, and James Edwards-Hewitt. Security patterns repository, version 1.0. http://www.modsecurity.org/archive/securitypatterns/dmdj_repository.pdf, *pattern source*, 2001.

- [KETEHE02] D. Kienzle, M. Elder, D. Tyree, and J. Edwards-Hewitt. Security patterns template and tutorial, February 2002.
- [Lea] Doug Lea. Checklist for writing great patterns. www.hillside.net/patterns/writing/writingpatterns.htm.
- [lib] The liberty alliance project. <http://www.projectliberty.org/>.
- [LP01] Sami Lehtonen and Juha Pärssinen. A pattern language for key management. In *The Conference on Pattern Languages of Programs (PLoP 2001)*, 2001.
- [LP02] Sami Lehtonen and Juha Pärssinen. A pattern language for cryptographic key management. In *The 8th European Conference on Pattern Languages of Programs (EuroPLoP 2003)*, Germany, 2002. *pattern source*.
- [MF06a] Patrick Morrison and Eduardo B. Fernandez. The credential pattern. In *The Conference on Pattern Languages of Programs (PLoP 2006)*, Portland, 2006. *pattern source*.
- [MF06b] Patrick Morrison and Eduardo B. Fernandez. Securing the broker pattern. In *The 11th European Conference on Pattern Languages of Programs (EuroPLoP 2006)*, July 2006. *pattern source*.
- [MGS03] Haralambos Mouratidis, Paolo Giorgini, and Markus Schumacher. Security patterns for agent systems. In *The 8th European Conference on Pattern Languages of Programs (EuroPLoP 2003)*, Germany, pages 25–29, June 2003. *pattern source*.
- [RCB⁺02] Dirk Riehle, Ward Cunningham, Joe Bergin, Norm Kerth, and Steve Metsker. Password patterns. In *The 8th European Conference on Pattern Languages of Programs (EuroPLoP 2003)*, Germany, 2002. *pattern source*.
- [RGFMP06] David G. Rosado, Carlos Gutiérrez, Eduardo Fernández-Medina, and Mario Piattini. Security patterns related to security requirements. In *4th International Workshop on Security in Information Systems (WOSIS)*, 2006.
- [RHCF06] Sasha Romanosky, Alessandro Acquisti, Jason Hong, Lorie Faith Cranor, and Batya Friedman. Privacy patterns for online interactions. In *The 13th Conference on Patterns Language of Programming (PLoP 2006)*, 2006. *pattern source*.
- [Rom01] Sasha Romanosky. Security design patterns, part 1 v1.4. <http://www.cgisecurity.com/lib/securityDesignPatterns.pdf>, *pattern source*, 2001.
- [Rom02] Sasha Romanosky. Enterprise security patterns. In *The 8th European Conference on Pattern Languages of Programs (EuroPLoP 2003)*, Germany, 2002. *pattern source*.

- [Sar03] Titos Saridakis. Design patterns for fault containment. In *The 8th European Conference on Pattern Languages of Programs (EuroPLoP 2003)*, Germany, 2003. *pattern source*.
- [Sch02] Markus Schumacher. Security patterns and security standards – selected security patterns for anonymity and privacy. In *The 8th European Conference on Pattern Languages of Programs (EuroPLoP 2003)*, Germany, 2002. *pattern source*.
- [Sch03] Markus Schumacher. Firewall patterns. In *The 8th European Conference on Pattern Languages of Programs (EuroPLoP 2003)*, Germany, 2003. *pattern source*.
- [SFBH⁺06] Markus Schumacher, Eduardo Fernandez-Buglioni, Duane Hybertson, Frank Buschmann, and Peter Sommerlad. *Security Patterns*. Wiley & Sons, 2006. *pattern source*.
- [shi] Shibboleth project. <http://shibboleth.internet2.edu/>.
- [SLPF05] Mauricio Sadicoff, Maria M. Larrondo-Petrie, and Eduardo B. Fernandez. Privacy-aware network client pattern. In *Pattern Languages of Programs Conference (PLoP 2005)*, 2005. *pattern source*.
- [SNL05] Christopher Steel, Ramesh Nagappan, and Ray Lai. *Core Security Patterns: Best Practices and Strategies for J2EE, Web Services, and Identity Management*. Prentice Hall PTR, 2005. *pattern source*.
- [Som03] Peter Sommerlad. Reverse proxy patterns. In *The 8th European Conference on Pattern Languages of Programs (EuroPLoP 2003)*, Germany, 2003. *pattern source*.
- [Sor02] Kristian Elob Sorensen. Session patterns. In *The European Conference on Pattern Languages of Programs (EuroPLoP 2002)*, 2002. *pattern source*.
- [SR01] Markus Schumacher and Utz Roedig. Security engineering with patterns. In *The 8th Conference on Pattern Languages of Programs (PLoP 2001)*, 2001. *pattern source*.
- [SYHJ08] Riccardo Scandariato, Koen Yskout, Thomas Heyman, and Wouter Joosen. Architecting software with security patterns. Technical Report CW-515, Katholieke Universiteit Leuven, Department of Computer Science, 2008.
- [VM02] John Viega and Gary McGraw. *Building Secure Software: How to Avoid Security Problems the Right Way*. Addison-Wesley, 2002.
- [Wei06] Michael Weiss. Credential delegation: Towards grid security patterns. In *The Nordic Pattern Languages of Programs Conference (VikingPLoP 2006)*, 2006. *pattern source*.

- [YB97] Joseph Yoder and Jeffrey Barcalow. Architectural patterns for enabling application security. In *The 4th Conference on Patterns Language of Programming (PLoP 1997)*, 1997. *pattern source*.
- [YHSJ06] Koen Yskout, Thomas Heyman, Riccardo Scandariato, and Wouter Joosen. A system of security patterns. Technical Report CW-469, Katholieke Universiteit Leuven, Department of Computer Science, 2006.