Security Design Patterns: Survey and Evaluation

M-A. Laverdière
Computer Security Laboratory, CIISE, Concordia University, Montreal, Canada
ma_laver@ciise.concordia.ca

A. Mourad
Computer Security Laboratory, CIISE, Concordia University, Montreal, Canada
mourad@ciise.concordia.ca

A. Hanna
Computer Science and Software Engineering, Concordia University, Montreal, Canada
ahanna@cse.concordia.ca

M. Debbabi
Computer Security Laboratory, CIISE, Concordia University, Montreal, Canada
debbabi@ciise.concordia.ca

Abstract

Security design patterns have been proposed recently as a tool for the improvement of software security during the architecture and design phases. Since the apperance of this research topic in 1997, several catalogs have emerged, and the security pattern community has produced significant contributions, with many related to design. In this paper, we survey major contributions in the state of the art in the field of security design patterns and assess their quality in the context of an established classification. From our results, we determined a classification of inappropriate pattern qualities. Using a Six Sigma approach, we propose a set of desirable properties that would prevent flaws in new design patterns, as well as a template for expressing them.

1. Motivations

Computer security professionals have been promoting, for many years, tools and best practices guidelines to be used by the software development industry, with little adoption so far. Developers, often pressed by a dominating time-to-market priority, must deal with a large set of technical and non-technical issues, in which case security concerns are not thoroughly addressed. The practical help for developpers are typically centered around frameworks, standard and design guidelines, which can be of limited use for both implementers and maintainers.

Security design patterns approach the problem from a different perspective, by encapsulating expert knowledge in the form of proven solutions to common problems. The idea of patterns was introduced by Christopher Alexander et al. [4] in the field of building architecture, and was later reused in the object-oriented world. Security patterns are such patterns, but applied for information security. These patterns will fit at different levels of abstraction and areas of concerns, resulting in many patterns that are not "design patterns" in the common sense of the expression.

The current research on the topic is characterized by various publications. The most comprehensive catalog so far was published by the Open Group in 2004 [5], which attempts to summarize into pattern a very wide variety of sources, such as security framework standards. However, the field is lacking a core reference similar to the Gang of Four patterns [8] in typical software design. Moreover, the patterns published so far typically fit at the "concept" and "example" levels of Kienzle et al. classification of pattern abstraction [11], with various templates and no established criteria for evaluation. This situation makes the security design patterns hard to use for software designers and maintainers alike, which limits their adoption in the industry, and thus lowers their positive impact on security.

The twofold contribution of this paper is to evaluate the current state of research on the topic of security design patterns, and draw out their undesirable properties. In section 2, we present the related work. Section 3 consists of the evaluation of the existing security design patterns. Section 4 presents the desired properties, the pattern template and the state of our work in progress. Section 5 gives some concluding remarks on this work as well as a few statements on future work.

2. Related Work

Yoder et al., in [15] introduced a 7-pattern catalog. Although some of these patterns are related to each others, it is not clear whether or not the proposed patterns should be considered as a core set of patterns for building all kinds of security-enabled systems. In fact, the proposed patterns by Yoder et al. were not meant to be a complete set of security patterns; rather just as starting point towards a collection of patterns that can help developers address security issues when developing applications.

Kienzle et al. [11], [12] have created a 29-pattern security pattern repository, which categorized security patterns as either structural or procedural patterns. Structural patterns are implementable patterns in an application whereas procedural patterns are patterns that were aimed to improve the development process of security-critical software. The presented patterns were implementations of specific web application security policies.

Romanosky [14] introduced another set of design patterns. The discussion however has focused on architectural and procedural guidelines more than security patterns. For example, the "White Hats, Hack Thyself" pattern indicated that implementation must be verified through testing.

Brown et al. [10] introduced a single security pattern, the authenticator, which described a general mechanism for providing identification and authentication to a server from a client. Although authentication is a very important feature of secure systems, the pattern, as was described, was limited to distributed object systems.

Braga et al. [6] also investigated security-related patterns specialized for cryptographic operations. They showed how cryptographic transformations over messages could be structured as a composite of instantiations of the cryptographic meta-pattern. This meta-pattern would effectively allow designers to apply cryptography in their applications, permitting to transparently change the underlying cryptographic operations.

The Open Group [5] has possibly introduced the most mature design patterns up-to-date. Their catalog proposes 13 patterns, and is based on architectural framework standards such as the ISO/IEC 10181 family [1], [2], [3]. The catalog is supplemented by an example of the patterns' use for solving a secure mail problem.

3. Evaluation of Security Design Patterns

In the scope of this evaluation, we examined contributions on security design patterns that were introduced by Yoder et al. [15], the Open Group [5], Kienzle et al. [12], [11], Brown et al. [10], and Priebe et al. [13]. Although not all were described as security design patterns, per se, those references all included patterns that could be used in design activities and were thus considered as such. We identified recurring issues in the patterns studied, which motivated this work. The patterns were manifesting one or many of the following undesirable characteristics: they were over-specified, under-specified, lacking generality, lacking consensus and misrepresented. We will first describe the undesirable characteristics, accompanied by one or more examples, before summarizing our evaluation.

3.1. Over-Specified Patterns

An over-specified pattern is one for which the specification provides more details and properties than needed. Patterns with variants are considered to be over-specified, as each variant would lead to a different pattern. This situation can lead to confusion, especially with variants specified. In all cases, the understandability of the pattern is impacted, as too many details will inevitably create confusion. The protected system [5] illustrates this issue, by defining two variants to the pattern. The encrypted pattern [12] illustrates the added complexity to the pattern when too much is specified. As the specification discusses at length about key management, it becomes harder to visualize a simple "add encryption" step in the processing, as the solution implies.

3.2. Under-specified Patterns

An under-specified pattern is a pattern that is incomplete, or that sees its specification not having enough properties. This will cause understanding or implementation problems, especially if no structure is provided in the specification. The layered security pattern [14] is a good example of this situation, whereas the solution was described in a very abstract manner.

3.3. Patterns Lacking Generality

Kienzle et al. [11] classified patterns in four levels of abstraction:

- Concepts
- Classes of patterns
- Patterns
- Examples

A pattern lacking generality is at a level 4 of abstraction, but it is claiming to be at level 3. This means that it is not general enough to allow it to be used in multiple circumstances with multiple targets [11]. Most of the patterns we studied were at level 4, as they were very practically-oriented, and often proposed with a single context in mind. For example, we can look at the authenticator [10] to see an interesting pattern that was proposed for a system built of remotely-accessible distributed objects. The author mentioned that the pattern is applicable to other uses, but the specification remains mostly focused on this context.

3.4. Patterns Lacking Consensus

A pattern lacking consensus can be recognized when its solution and/or intent is not uniform in the literature. Patterns are also lacking consensus when named using terminology common in the literature, but for which a different concept is associated. This makes the pattern of little use, as developers and maintainers are likely not to share a common understanding about the system's design.

A good example of this lack of consensus is the subject descriptor. In [5], the subject descriptor is used as a holder for subject-related attributes. In this context, the "subject" refers to any entity interacting with a program. However, in [13], the subject descriptor is used to aggregate for attribute qualifiers within the metadata-based access control pattern. The session pattern also fits in this category, as the notion of a session refers to a variety of concepts, such as OS logins (e.g. a Unix "session") and some usages of web application (e.g. a shopping cart for an online store).

3.5. Misrepresented Patterns

A misrepresented pattern is a pattern that does not offer what it claims to, especially if its naming and/or intent has little to do with the rest of the pattern's description. The most illustrative example would be the protected system pattern [5]. This pattern describes security policy enforcement specific to resource access, which obviously does not "protect" a system on all its aspects.

3.6. Summary of Evaluation

In order to illustrate the current state of design patterns, we summarized our analysis of patterns from various sources in Table I. Every mark indicates that this undesirable property was found in the described pattern.

4. Proposed Directions

Various problems related to the level of abstraction and security properties requirements exist in the current security design patterns. To avoid having such flaws in our proposed patterns, we will first introduce in this paper a set of desirable security properties, in addition to a template for developing and presenting security design patterns.

Patterns / Undesirable Properties	Single Access Point	Protected System	Policy	Subject Descriptor	Authenticator	Security Context	Session		Secure Communication	Security Association	Encrypted Storage	Layered Security	Minefield
Over-Specified		⊗							⊗				
Under-Specified	⊗							⊗				⊗	
Lacking Generality	⊗	⊗	⊗		⊗	⊗				⊗	⊗		⊗
Lacking Consensus				⊗		⊗	⊗	⊗				⊗	
Misrepresented		⊗										⊗	
Author	[15]	[5]	[5]	[5], [13]	[10]	[5]	[13]	[15]	[5]	[5]	[12]	[14]	[12]

TABLE I
Summary of Pattern Evaluation

4.1. Desirable Properties

Consequently to our initial results, we studied the desirable properties that should be exhibited by security design patterns. Considering that CASE tools such as Rational XDE offer to integrate known design patterns into UML diagrams, it is essential to examine patterns as deliverable software products.

This need can be met using the particularly fitting approach of the Six-Sigma house of quality [9]. Using this tool, customer requirements and technical parameters are put together, in order to determine interrelations between the customer and technical requirements, as well as between technical requirements. The house of quality we determined for security design patterns is presented in Figure 1.

We identified the following customer requirements:

- *Proper level of abstraction*: The pattern is at level 3 of the classification introduced by Kienzle et al. [11], and at the Meta-layer level [7]. In short, the pattern can be used in different contexts without redefinition.
- *Completeness*: The pattern is complete, and is specified correctly.
- *Ease of use*: The pattern should be applicable with minimal possible impact on the software utilizing it.
- Implementability: The pattern should be easily implementable, taking target platform constraints, especially for embedded devices.
- *Reusability*: The pattern should be easily applied and used in different contexts.
- *Pattern composability*: The pattern should be easily used with other patterns.
- *Security compositionality*: The pattern can be used with other patterns without loosing any of the security it provides, nor affecting the security provided by the other patterns.
- *Validability*: The patterns' security properties, as well as their implementation and usage, can be easily validated. This validation would preferably be automated.

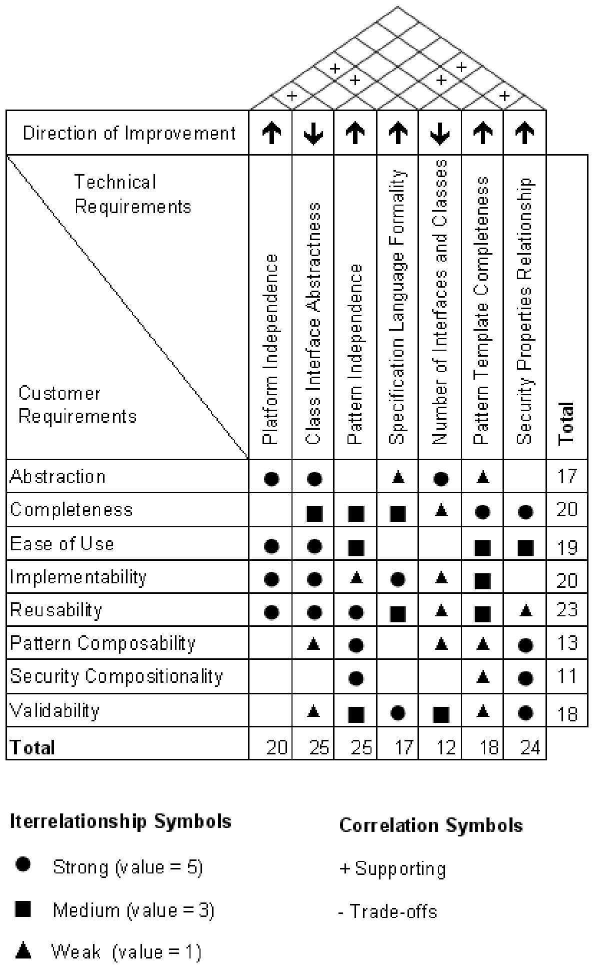
We identified the following technical requirements:

- *Platform independence*: The pattern is specified without a specific programming language, operating system, framework, etc., in mind.
- *Class interface abstractness*: The pattern's interface level of definition. The higher the definition level of the pattern is, the more conceptually clear the pattern becomes, at the risk of becoming too specific.
- *Pattern independence*: The pattern is independent when it can be used in a standalone manner.
- *Specification language formality*: The specification language is the language used to specify and describe the pattern. Its formality and abstractness is ranked in this category.
- *Number of interfaces and classes*: The amount of classes and interfaces specified.
- *Pattern template completeness*: The structure of the template used to describe the patterns includes all the required information.
- *Security properties relationship*: How related the pattern is to security properties.

4.2. Patterns' Template

Our template is based on existing design pattern templates [5], [11] with some additional sections, which we believe necessary for presenting the security patterns in an efficient way.

- *Name*: The name of the pattern is the common-usage short expression that encapsulates the pattern's meaning.
- *Abstract*: The abstract is a short summary of what the pattern is and does, that is short and independent of context.
- *Problem*: The problem is a short description of the design problem that this pattern aims at solving.
- *Solution*: The solution is a textual description of the pattern that solves the problem.
- *Applicability*: The applicability is a list of circumstances where this pattern is useful.
- *Consequences*: The consequences aim at quickly listing the advantages and disadvantages of using the pattern.



Direction of Improvement	↑	↓	↑	↑	↓	↑	↑	
Technical Requirements / Customer Requirements	Platform Independence	Class Interface Abstractness	Pattern Independence	Specification Language Formality	Number of Interfaces and Classes	Pattern Template Completeness	Security Properties Relationship	**Total**
Abstraction	●	●		▲	●	▲		17
Completeness		■	■	■	▲	●	●	20
Ease of Use	●	●	■			■	■	19
Implementability	●	●	▲	●	▲	■		20
Reusability	●	●	●	■	▲	■	▲	23
Pattern Composability		▲	●		▲	▲	●	13
Security Compositionality			●			▲	●	11
Validability		▲	■	●	■	▲	●	18
Total	20	25	25	17	12	18	24	

Figure 1: House of Quality for Security Patterns

• *Trade-Offs*: The trade-offs section aim at explaining in more details the nature of advantages and disadvantages of this pattern over quality attributes.
• *Structure*: The pattern's structure is illustrated using a modeling or formal language.
• *Related Patterns*: Patterns related to this pattern, or patterns that inspired this pattern, are listed in this section.

5. Conclusion and Future Work

In this paper, we reported a survey on the state of the art of security design patterns and found important undesirable properties in them. Basing ourselves on those results, and using a Six-Sigma approach, we were able to determine the desirable properties of security design patterns as an engineering product. Furthermore, we proposed a new template for pattern description that would allow to better specify patterns in the future.

In future research, we aim at defining new patterns for representing, managing and implementing security policies, as well as providing their usage-specific instances.

References

[1] ISO/IEC 10181-1:1996, *Security frameworks for open systems: Overview*, 1996.
[2] ISO/IEC 10181-2:1996, *Security frameworks for open systems: Authentication framework*, 1996.
[3] ISO/IEC 10181-3:1996, *Security frameworks for open systems: Access control framework*, 1996.
[4] C. Alexander, S. Ishikawa, and M. Silverstein, *A pattern language*, Oxford University Press, 1977.
[5] B. Blakley and C. Heath, *Security design patterns*, Tech. Report G031, Open Group, 2004.
[6] A. Braga, C. Rubira, and R. Dahab, *Tropyc: A pattern language for cryptographic software*, 1998.
[7] E. B. Fernandez and R. Pan, *A pattern language for security models*, Proceedings of the PLoP 2001, 2001.
[8] E. Gamma, R. Helm, R.Johnson, and J. Vlissides, *Design patterns: Elements of reusable object-oriented software*, Addison-Wesley, 1994.
[9] J. R. Hauser and D. Clausing, *The house of quality*, IEEE Engineering Management Review **24,1** (1996), pp. 24–32.
[10] F.L. Brown Jr. and E. B. Fernandez, *The authenticator pattern*, Proceedings of the PLoP 99, 1999.
[11] D. M. Kienzle and M. C. Edler, *Final technical report: Security patterns for web application development*, Tech. Report DARPA Contract # F30602-01-C-0164, 2002.
[12] D. M. Kienzle, M. C. Elder, D. Tyree, and J. Edwards-Hewitt, *Security patterns repository*, 2002.
[13] T. Priebe, E.B. Fernandez, J.I. Mehlau, and G. Pernul1, *A pattern system for access control*, Proceedings 18th Annual IFIP WG 11.3 Working Conference on Data and Applications Security, 2004.
[14] S. Romanosky, *Security design patterns part 1*, 2001.
[15] J. Yoder and J. Barcalow, *Architectural patterns for enabling application security*, Proceedings of the PLoP 97, 1997.