The Nature of Order: From Security Patterns to a Pattern Language

Munawar Hafiz
Auburn University
munawar@auburn.edu

Paul Adamczyk
paul.adamczyk@gmail.com

Categories and Subject Descriptors D.2.11 [*Software Engineering*]: Software Architectures—Patterns

General Terms Design, Security.

Keywords Security Patterns, Pattern Language.

1. Introduction

Christopher Alexander [1] was the first to introduce the concept of a pattern language. He has inspired many computer scientists to develop pattern languages for software, but so far we have not produced a result that is as impressive as his. This paper describes our experiences of developing a pattern language for security [6]. We describe the mechanism of growing this pattern language: how we cataloged the security patterns from books, papers and pattern catalogs, how we classified the patterns to help developers find appropriate patterns, and how we identified and described the relationships between patterns in the pattern language. Ours is the first pattern language that covers patterns of an entire problem domain; to our best knowledge, it is also the largest in software. But the most significant contribution of this work is the story behind how the pattern language is grown, as the mechanism can be adapted to other domains.

2. Growing a Pattern Language

Security patterns capture security knowledge. We have been maintaining a comprehensive catalog of all security patterns published in the last 15 years (the first security pattern paper [14] appeared in 1997). It is a union of all security patterns that appear in many books [7, 11, 12], catalogs [2, 8–10], and papers [3, 4, 14]. It accumulates the experience of the entire security pattern community and is a fair representation of the solution domain of security.

SPLASH'12, October 19–26, 2012, Tucson, Arizona, USA.
ACM 978-1-4503-1563-0/12/10.

The pattern sources describe 174 security patterns, but with many overlaps. After removing the overlaps, the current catalog contains 96 security patterns.

We started by evaluating various classification schemes that break up the large catalog into smaller clusters of related patterns. Second, we created small pattern languages, one for each cluster of patterns. Third, we combined the small diagrams into one large diagram, adding more inter-group relationships. We describe these three steps here.

2.1 Categorize Security Pattern Catalog

Our previous work [5] organized the security patterns. We tried several classification schemes; and found that a hierarchical scheme using threat models works the best. A tree-based scheme allows us to classify a pattern by placing it on a leaf node (low-level patterns) as well as internal nodes (high-level patterns), therefore creating the hierarchy. Patterns at the root of the tree are applicable to multiple contexts. The application contexts (core, perimeter, and exterior) are in the internal nodes. Each context is further classified using threat models; we use the STRIDE threat model [13].

To illustrate the classification, consider SADE DATA STRUCTURE [4] that suggests that strings should be represented with a separate data structure that keeps information about allocated and used memory. This is a *core* pattern that prevents *tampering*, and it is classified at that leaf node. On the other hand, SECURITY NEEDS IDENTIFICATION FOR ENTERPRISE ASSETS [11] describes a process for asset evaluation. This is clearly more general, and applies to all contexts and all threat models. It is classified at the root node of the tree.

2.2 Build Category-specific Pattern Languages

A pattern language offers the reader a guidance in selecting the next pattern to consider. It shows all the closely related patterns. But how does one determine that there exists a relationship between patterns? In Alexander [1], the pattern at the head of the connecting arrow is typically mentioned in the Context section of the pattern that appears at the end of the arrow. Alternatively, the pattern at the end of the arrow is mentioned in the Resulting Context of the pattern at the head of the arrow. Unfortunately, most of the security patterns

were not written in that format, so we had to re-create those relationships by carefully considering the typical order in which they would be applied. We listed them in that temporal order, and then described their relationships.

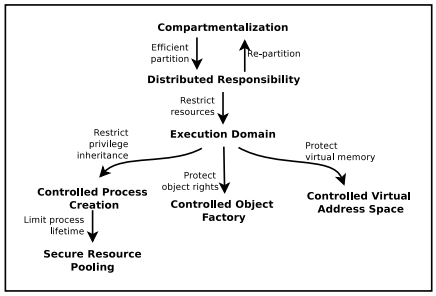

Figure 1. Core Patterns for Preventing Privilege Escalation

Figure 1 shows the relationship between 7 core patterns that prevent privilege escalation. The first step of preventing privilege escalation is partitioning an application, described by two patterns. We relate them by an arrow from COMPARTMENTALIZATION [4] to DISTRIBUTED RESPONSIBILITY [4], meaning that the latter follows the former. In practice, these two patterns typically go hand in hand. A user compartmentalizes a system, then checks whether the responsibilities are properly distributed, and may compartmentalize again and so on. So, we add an arrow in the reverse direction. EXECUTION DOMAIN [11] specializes DISTRIBUTED RESPONSIBILITY [4] by restricting access to resources available to a process.

It is important to check process creation (CONTROLLED PROCESS CREATION [11]), object creation (CONTROLLED OBJECT FACTORY [11]), and memory usage (CONTROLLED VIRTUAL ADDRESS SPACE [11])—all three follow EXECUTION DOMAIN [11]. SECURE RESOURCE POOLING [4] eases process creation by pre-forking a process pool; it also adds control on process lifetime to prevent privilege escalation vulnerabilities in long-running processes.

2.3 Unify Pattern Languages

The small pattern languages created are relatively self-contained; each contains patterns for a type of security problem. But if we consider the pattern language for high level security patterns, they will explain how to approach solving any security problem, but stop short of solving any. They will leave off where all the other figures for small pattern languages start. Figure 2 presents a relationship between the small pattern languages. It sketches the unification step.

3. Conclusion

A pattern language communicates *the nature of order* by describing the way patterns relate to one another. Our current pattern language is a work in progress. It has not yet been used on a large project, been reviewed by more than a few security experts, or been used to teach students how to make

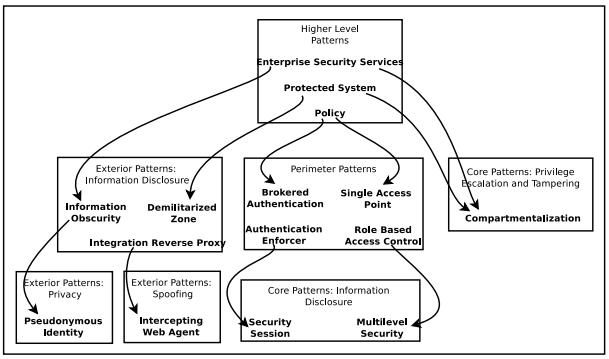

Figure 2. Unifying the Pattern Languages

secure systems. Doing these things will undoubtedly point out ways to improve it. However, its current version is already an improvement on the state of the art.

A. Location of Security Pattern Catalog

`http://munawarhafiz.com/securitypatterncatalog/index.php`.

References

[1] C. Alexander, S. Ishakawa, and M. Silverstein. *A Pattern Language: Towns, Building and Construction*. Oxford University Press, New York, 1977.

[2] B. Blakley and C. Heath. Security design patterns technical guide–Version 1. Technical report, Open Group(OG), 2004.

[3] M. Hafiz. A Pattern Language for Developing Privacy Enhancing Technologies. *To be published in Software—Practice and Experience*, 2012.

[4] M. Hafiz and R. Johnson. Evolution of the MTA architecture: The impact of security. *Software—Practice and Experience*, 38(15):1569–1599, Dec 2008.

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

[6] M. Hafiz, P. Adamczyk, and R. Johnson. Growing a pattern language (for security). In *OOPSLA*, 2012.

[7] J. Hogg, D. Smith, F. Chong, D. Taylor, L. Wall, and P. Slater. *Web Service Security: Scenarios, Patterns, and Implementation Guidance for Web Services Enhancements (WSE) 3.0*. Microsoft Press, March 2006.

[8] D. Kienzle, M. Elder, D. Tyree, and J. Edwards-Hewitt. Security patterns repository version 1.0, 2002.

[9] S. Romanosky. Security design patterns part 1, Nov 2001.

[10] S. Romanosky. Enterprise security patterns, 2002.

[11] M. Schumacher, E. Fernandez-Buglioni, D. Hybertson, F. Buschmann, and P. Sommerlad. *Security Patterns: Integrating Security and Systems Engineering*. John Wiley and Sons, December 2005. ISBN 0-470-85884-2.

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

[13] F. Swiderski and W. Snyder. *Threat Modeling*. Microsoft Press, 2004.

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