Security Patterns and their Classification Schemes

Munawar Hafiz, Ralph E. Johnson
University of Illinois at Urbana-Champaign
mhafiz@uiuc.edu, johnson@cs.uiuc.edu

August 15, 2006

Abstract

Finding the appropriate pattern to solve a particular security problem is difficult because of the absence of a scientific classification scheme for security patterns. A suitable classification scheme helps efficient storage and retrieval of information, beneficial for both software pattern miners and pattern navigators. In this paper, we provide a survey of security patterns and evaluate various classification schemes for security patterns. Our proposed classification scheme uses security concepts to efficiently partition the problem space, therefore solving the pattern navigation problem.
Keywords: Security Patterns, Patterns Classification.

1 Introduction

Before the introduction of the Internet, software systems were like separate islands, secure from outside intrusion (except for a physical break-in). Security was discussed in terms of password management, access control of enterprise information, confidentiality protection using some cryptographic means, or code level security like buffer overflow. Software architectural qualities like efficiency, reliability and availability were considered to be the primary quality requirements.

The introduction and proliferation of the Internet exposed software systems to intrusion and tampering from globally distributed attackers. Software development had to undergo a paradigm shift. Many software systems that were previously considered adequate have become inadequate. For example, sendmail, the most popular mail transfer agent (MTA), has been plagued with security vulnerabilities, since it was first exploited by the Morris worm in 1988 [10]. sendmail was developed by Eric Allman in the early

1980s. The major quality requirement for sendmail was flexibility and it is a very good example of flexible architecture because it supported various types of protocols. The requirements for an MTA changed with the introduction of Internet and security and reliability became as important as flexibility. The problems of sendmail architecture were illustrated by the constant detection and fix of security vulnerabilities. These inspired Daniel Bernstein to design qmailas a secure alternative for sendmail. qmail has had no security defects detected since the release of qmail 1.0 in 1997. The security of qmail is the result of a set of architectural decisions. The same architecture that makes it secure also makes it more efficient than sendmail and easier to understand. Later these architectural decisions were adopted/adapted by other MTA developers to make their MTAs secure. These secure MTAs have taken away the monopoly that sendmail enjoyed in the MTA domain.

The sendmail example shows the changed impact of security on current software architecture. Current software engineering practices demand up-front security approaches, but security is often considered as an afterthought. The probable reason is that security is a non-functional requirement and software project managers often remain concerned about the functional requirements or the more visible non-functional requirements like performance. This practice is gradually changing and more software projects are now considering security up-front.

Security architects reuse proven solutions for security problems by using security patterns. Referring back to the sendmail example, security patterns influenced many of the design decisions taken by the designer of qmail.

Recently there has been an increasing interest about documenting security patterns, manifested by the release of a number of books [35, 32, 19]. Along with these books, researchers have been maintaining security catalogs. There is a significant overlap among these catalogs, *i.e.* similar patterns are listed under different names. One of the advantages of patterns is that they provide a uniform vocabulary for exchanging ideas among software practitioners. The unrestrained growth of security patterns keeps the vocabulary from being uniform.

Both pattern writers and pattern users need a classification scheme or a taxonomy. For the pattern writers, the presence of a classification scheme means that they will be able to define the patterns in appropriate scopes. This way the authors would be able to find overlaps more efficiently. This will also make cross-referencing easier, because patterns that provide alternative solutions for similar problems will be adjacent in the classification scheme. For the users, it will be easy to find a pattern as a solution for a particular problem. Also, alternative solutions can be found and compared.

This paper surveys the existing security patterns and their classification schemes. We list the problems of the current classification schemes. We then provide a roadmap towards an alternative pattern classification methodology.

2 Security Patterns and classification schemes

Christopher Alexander [3, 2] introduced patterns in the architecture of cities, roads and buildings. The use of patterns in software design has been popularized by "Design Patterns", written by the Gang of Four [12]. Architectural patterns were described by the POSA team in their book [8]. As patterns were proven valuable for software design, there were efforts to document domain specific patterns, *e.g.* concurrent and distributed systems [31], resource management [26], user interfaces [37] *etc.*

An important non-functional requirement of software is security. The security pattern documentation efforts started in 1997 by Yoder *et al.* In this section, we first describe the efforts of security patterns documentation till date. Then we describe the classification efforts of security patterns.

2.1 Security Patterns

Yoder *et al.* described 7 architectural patterns that contribute to security [41]. Three of these patterns, *Single Access Point*, *Checkpoint* and *Secure Access Layer* focus on securing the access interface of a system. The *Roles* pattern groups the users into roles to perform access control on system resources. The *Session* pattern performs secure storage and distribution of session information throughout parts of the system. Two patterns *Limited View* and *Full View with Errors* provided design options for a secure interface.

There were some papers on security patterns [6, 7] in the PLoP conferences in the next couple of years but the next big wave of work on security patterns came in 2001. Schumacher *et al.* described in their PLoP paper the methodology and benefits of applying security patterns in secure software design [33]. Sasha Romanosky compiled a collection of 8 design level security patterns [30]. These patterns cover the security threats in different application domains, *e.g.* enterprise management (*Risk Assessment and Management*), brokered communication (*3rd Party Communication*), testing (*White Hats, Hack Thyself*), fault tolerance (*Fail Securely*), software maintenance (*Low Hanging Fruit*) *etc.*

In EuroPLoP 2002, Sasha Romanosky presented another collection of enterprise level security patterns [29]. This catalog had four security patterns. The same year, Bob Blakley and Craig Heath compiled the first draft of their catalog of security patterns with the members of the Open Group forum. In 2004, they released a revised version of the catalog [5]. This version had 13 patterns described under two broad categories - available systems and protected systems. The 5 available systems patterns describe fault tolerance and error handling schemes. The 8 patterns for protected systems cover security aspects of system perimeter and network.

Kienzle *et al.* listed 26 patterns and 3 mini patterns in their security patterns catalog [25]. The patterns were described under two broad categories. There were 16 structural patterns (including the three mini-patterns) and 13 procedural patterns.

Hafiz *et al.*approached the security patterns from a different direction. Their work studied the architecture of qmail and described the security patterns that contributed to its secure architecture [17]. They listed 5 security patterns, 4 reliability patterns and 1 performance pattern including 3 new security patterns, 1 new reliability pattern and 1 new performance pattern. These patterns are not MTA specific; they can be used by the designers of other secure systems. Hafiz later extended his work by documenting 6 more security patterns (including 5 new patterns) in the architecture sendmail, qmail, Postfix and sendmail X [15]. Two of these patterns were workshopped in PLoP 2004 and 2005 [13, 14].

Privacy has become an important concern with the growth in computing power that has enabled the storage and analysis of large volumes of data. A collection of privacy design patterns have been listed by Hafiz [16] in 2006.

Markus Schumacher led the working group on a security patterns group, and the group launched the `www.securitypatterns.org` website and a mailing list. Wiley and Sons published the book [32] in December, 2005. This book had 46 security patterns from the domain of enterprise security and risk management, identification and authentication, access control, accounting, firewall architecture, and secure internet application.

A group at Sun Microsystems was working on a book covering architectural security patterns for J2EE based applications, Web services and identity management. Prentice Hall published the book [35] in October 2005. The book provides a comprehensive view of Java application security fundamentals and lists 23 security patterns.

Microsoft's Patterns and Practices group published a security patterns book [19] with their release of Web Services Enhancement (WSE) 3.0. The book acts as an implementation guidance for using WSE 3.0 in Web services

development. This book lists 5 patterns for authentication, 2 patterns for message protection, 5 patterns for transport and message layer security, 2 patterns for resource access, 3 patterns for service boundary protection and 1 pattern for service deployment. These patterns are all described from Microsoft technology perspective.

There is a significant amount of overlap in the different security pattern catalogs. For example, patterns for authentication have been listed in a number of pattern catalogs. The *Authenticator* pattern, listed in conjunction with other patterns of operating systems security and access control [11], was later included in the Wiley security patterns book [32]. *Authenticator* is essentially a *Policy Enforcement Point* [41] that intercepts the interactions of a subject with the system and applies a protocol to verify the identity of the subject. The protocol that is used might depend on some user input (*i.e.* something that the user knows *e.g.* password, or possesses *e.g.* a smart card, or has *e.g.* a biometric characteristic), or the input provided by a trusted third party (*i.e.* brokered authentication). Sasha Romanosky described the same pattern under the name *Security Provider* [30]. This pattern is also listed in the security patterns repository by Kienzle *et al.* as *Authenticated Session* [25]. The Core Security Patterns book described the *Authentication Enforcer* pattern [35] from Java tool oriented perspective. This pattern creates a centralized authentication enforcement that performs authentication of users and encapsulates the details of the authentication mechanism. The same pattern appeared in the Microsoft security patterns book. Microsoft described authentication in terms of various strategies supported by the WSE 3.0 framework. They divided authentication in two parts and described two patterns. The *Direct Authentication* pattern describes the scenario when web services act as the authentication service to validate the subject credentials. The *Brokered Authentication* pattern, alternatively, provides authentication without the direct communication between the subject and the service. Three different brokered authentication strategies were described as separate patterns, namely, *Brokered Authentication: Kerberos*, *Brokered Authentication: X509 PKI* and *Brokered Authentication: Security Token Service (STS)*.

The patternshare website (`www.patternshare.org`) was launched as a single repository to describe the patterns. One of the goals of patternshare is to limit the overlaps and create a uniform vocabulary for practitioners. The patternshare repository for security patterns [34] provides an overview of the security patterns work from various sources removing the overlaps. In March 2006, the repository had 59 patterns. The repository would be continuously updated with future work from different sources.

2.2 Security Patterns Classification

The *Authenticator* pattern example necessitates a solution to resolve conflicts. Biologists solved this problem with the use of taxonomy. Taxonomy refers to either a classification of things, or the principles underlying the classification [40]. Mathematically, classification is performed as a tree based mechanism where the root node provides a more general concept that is refined by each descendant. Another classification method is to partition of the domain in discrete groups based on some criteria. The security patterns classification efforts are based on partitioning the systems space. Here we list a survey of the classification schemes.

Classification based on applicability. The Open Group security catalog classifies the patterns into two broad groups based on applicability; patterns for protected systems and patterns for available systems. The protected system patterns protect valuable resources against unauthorized use, disclosure, or modification. The available system patterns provide predictable and uninterrupted access to the services and resources. The advantage of this scheme is that it classifies the patterns according to the software architectural qualities that they address. However, this partitioning is too broad to be useful.

Classification based on product and process. Kienzle *et al.* described their patterns under two broad classes, structural patterns and procedural patterns. The structural patterns can be implemented in the final product. An example of a structural patterns is *Account Lockout*. This pattern is used to thwart the brute force attack on passwords by locking the account after a number of authentication tryouts. The solution can be implemented and integrated in a software architecture. The procedural patterns, on the other hand, improve the process for development of security-critical software. These patterns influence the organization or the management of a development process. An example is the *Red Team the Design* pattern from this catalog. The pattern promotes the development of a red team (*a.k.a.* a team of gray hat hackers) to try to break the system by finding the vulnerabilities of the system.

Classification based on logical tiers. The security patterns in layered systems are classified according to the system tiers. In this scheme, the patterns are classified as presentation or web tier patterns, business tier patterns and integration tier patterns. The web tier patterns intercept external requests and perform authentication and authorization. *e.g. Single Access Point* [41]. The business tier patterns support the security services in the business tier. *Role* [41] is a business tier pattern that implements

access control by grouping users under roles and perform Role Based Access Control (RBAC). The integration tier patterns facilitate secure integration with external data sources.

The Core Security Patterns book classify their patterns according to logical architecture tiers, namely web tier, business tier, web service tier, and identity tier. The identity tier is an additional logical tier that has patterns for signing-on the authenticated identity with identity infrastructure providers. An example is the *Single Sign-on Delegator* pattern that describes how to construct a delegator agent for handling a legacy system for single sign-on.

This classification schemes has the advantage that the partitioning is aligned with the system tiers. Hence the classification does not introduce new vocabulary for system architects and developers. However, the advantage of a classification scheme comes out of using the domain specific vocabulary. Here the concepts of security are not used for classifying.

Classification based on security concepts. ISO 13335 [24] and ISO 7498 [23] provides a definition of the four key concepts of security - confidentiality, integrity, availability and accountability. A pattern classification scheme based on these domain level concepts, will facilitate pattern mining and pattern navigation.

Classification based on system viewpoints and interrogatives. The security patterns book from Wiley publications introduced a classification scheme for security patterns that is based on the Zachman framework (figure 1). Zachman framework [42] was introduced in 1987 as a table with the rows describing the levels of information model and the columns describing the architectural views.

The levels of information model are based on three fundamental architectural representations one for each stakeholder, *i.e.* the customer or the owner, the designer and the builder. The owner has his own concept of the end product. The architect translates these perceptions into the designer's perspective. The builder then adds the constraints of the laws of nature and available technology to make a refinement of the architect's plan. Preceding these three views is a gross representation of the end system that establishes a ballpark. Succeeding the three views are the detailed, out-of-scope representations of parts of the system that are important during detailed development of the system. These five levels are the five rows in the Zachman framework. The Zachman views are represented in the six columns in the table: data (what?), function (how?), network (where?), people (who?), time (when?) and motivation (why?). These represent the different aspects of the object being described. Each of the views are orthogonal to each

other, but they describe the same object and are therefore inextricably associated. Zachman framework is also used as the basis of Microsoft patterns classification scheme [38] described in the next section.

	What (Data)	How (Function)	Where (Locations)	Who (People)	When (Time)	Why (Motivation)
Scope {contextual} Planner	List of things important to the business	List of processes that the business performs	List of locations in which the business operatses	List of organizations important to the business	List of events/ cycles important to the business	List of business goals/strategies
Enterprise Model {conceptual} Business Owner	e.g. Semantic Model	e.g. Business Process Model	e.g. Business Logistics System	e.g. Workflow Model	e.g. Master Schedule	e.g. Business Plan
System Model {logical} Designer	e.g. Logical Data Model	e.g. Application Architecture	e.g. Distributed System Architecture	e.g. Human Interface Architecture	e.g. Process Structure	e.g. Business Rule Model
Technology Model {physical} Implementer	e.g. Physical Data Model	e.g. System Design	e.g. Technology Architecture	e.g. Presentation Architecture	e.g. Control Structure	e.g. Rule Design
Detailed Representation {out-of-context} Subcontractor	e.g. Data Definition	e.g. Program	e.g. Network Architecture	e.g. Security Architecture	e.g. Timing Definition	e.g. Rule Definition
Functioning System	e.g. Data	e.g. Function	e.g. Network	e.g. Organization	e.g. Schedule	e.g. Strategy

Figure 1: The Zachman framework (c) 1982-2006 John A. Zachman. www.zachmaninternational.com

To classify security patterns, the Zachman framework has been modified by adding a column representing the security view [21, 18]. The security view addresses all model levels, from the enterprise scope to the detailed representations. However, such a mechanism would mean that the security column would include the security patterns that occur in the context of the 6 other views of the table. The other views of the Zachman framework primarily deal with various functional properties of the system whereas security is a non-functional property that should be orthogonal to the functional aspects of the system. Hence adding security as a different view does not solve the problem of classification. Basically this is just classifying the security patterns according to the five viewpoints of the Zachman framework. This results in the same classification as partitioning based on system tiers.

3 Classification Scheme proposed by Microsoft

In 2004, Microsoft Patterns and Practices group introduced a tabular classification scheme for patterns [38], primarily based on the Zachman framework. The classification scheme encapsulates the enterprise architectural space, and illustrates the relationship among artifacts in the enterprise space.

The classification scheme is based on four key pieces of work. The first one is the Zachman framework. The columns of the table are based on interrogatives and the rows are based on roles. The table has seven columns and six of them are the Zachman framework views.

The second key piece of work is the architectural standards description from IEEE 1471 [22]. In the Microsoft scheme, The rows are specified in finer granularity than the Zachman framework by adopting the architectural standard description from IEEE 1471. The discrete viewpoints adopted in the table provide the classification scheme more depth because of the specificity.

The third key piece of work is the Enterprise Architecture Framework [27]. The framework influences the grouping of rows according to the different levels of architecture.

Finally, the scheme is influenced by the principles of test-driven development [4]. The *Test* column in the table is influenced by the principles of test-driven development. This column would include the patterns that validate the artifacts contained in the *Purpose* column of the table.

The scheme described in the next section is followed by a critique.

3.1 Enterprise Architectural Space Organizing Table

The *Enterprise Architectural Space Organizing Table* is a two-dimensional table that encapsulates the different aspects of software architecture. Each cell in the table is defined by an interrogative and an architectural viewpoint. The classification of patterns is based on the definition of the cells.

Architectural viewpoints. The table has five grouped rows based on the five broad architectural viewpoints. The viewpoints are,

- Business Architecture
- Integration Architecture
- Application Architecture
- Operational Architecture
- Development Architecture

The `Business Architecture` viewpoint encapsulates the business and management perspective of software development. The `Integration Architecture` viewpoint is concerned with the integration between internal and external systems in an enterprise. The `Application Architecture` viewpoint covers the system and software elements of an executable application. The `Operational Architecture` viewpoint is concerned with the operation of the production system. Finally, the `Development Architecture` viewpoint covers the systematic implementation concerns of application and integration architecture.

Interrogatives. Although the viewpoints provide a clear categorization of the perspectives, finer granularity can be achieved based on the interrogatives in the Zachman framework. The interrogatives from the Zachman framework and test driven development are illustrated by the seven columns.

- **Purpose (Why).** The reason behind an architectural decision.
- **Data (What).** Input and output of a decision making process.
- **Function (How).** The mechanism of architectural decision making.
- **Timing (When).** Timing related issues of a decision or the decision making process.
- **Network (Where).** Communication related issues of architecture.
- **People (Who).** Issues concerning the stake holders and users of a system.
- **Scorecard (Test).** Checking for compliance with the requirements.

Each of the rows are further partitioned according to the important role-players in that perspective. For example, the `Business Architecture` viewpoint is partitioned using the four primary role-players. They are,

- Chief Executive Officer (CEO)
- General Manager
- Process Owner
- Process Worker

As an example of classification, consider the *Safe Data Structure* [17] pattern. This pattern is applied to remove the array bounds checking vulnerability in a programming language with no garbage collection. A system written in C is vulnerable to buffer overflow attacks because of unsafe array operations, *e.g.* unsafe string handling. The *Safe Data Structure* pattern advocates the inclusion of length and allocated memory information with a data structure. All the array processing libraries are re-written to consider the length and allocated memory operation before processing the array. This

pattern is considered in the development phase of an application when the safe string processing libraries are written or re-used. Hence this pattern fits into the cell defined by the `Developer` row of `Application Architecture` perspective and the `Function` column.

3.2 Security Patterns Classification using the Enterprise Architectural Space

The security pattern repository in the patternshare website provides a comprehensive listing of security patterns from different sources. Currently it has 59 patterns. The patterns are classified using the classification scheme. Table 1 summarizes the current classification.

Perspective	**Viewpoint**	**Interrogative**	**Patt. count**	**Example Pattern**
Business Architecture	CEO	Function	5	*Security Needs Identification.* [32] Create an association between enterprise assets and security needs.
Integration Architecture	Enterprise Architect	Function	2	*Single Sign On.* [25] Allow a user to access multiple services in a distributed network environment without having to re-authenticate on every request.
Application Architecture	Architect	Data	2	*Error Detection and Correction.* [5] Redundancy added to data for error detection and correction.
		Function	27	*Single Access Point.* [41] Single Entry Point for each process.
	Design	Network	4	*Stateful Firewall.* [32] Filter traffic based on state information.
		Data	4	*Encrypted Storage.* [25] Server data is is protected by encryption.
		Function	12	*Server Sandbox.* [25] Servers run with least privilege to limit client activities.
	Developer	Function	1	*Safe Data Structure.* [17] Memory buffers contain length information that is checked before allocation.
		Test	1	*White Hats hack Thyself.* [30] Test the system's security by attacking it.
Operational Architecture	System Architect	Function	1	*Low Hanging Fruit.* [30] Get quick fixes rather than trying to re-design the system every time a vulnerability is found.

Table 1: Classification of Security Patterns

The benefit of such a classification scheme is that it clearly identifies the context of each patterns. This way pattern navigation becomes easier. Users can look into specific cells in the table similar to the context of their problem, and look for solutions there. This can also be used for identifying missing patterns. For example, if a pattern is defined in one context but the

user is looking for the pattern in a different context, it is likely that the same pattern exists in that context as well and these two patterns are related.

However, listing the same pattern in different contexts with varying granularity will create a huge number of patterns that exists in more than one cell in the table. This will be easier for pattern navigation, but will make it harder to manage the patterns. Our experience with classification scheme was that many security patterns cannot be defined by one cell. For example, the `Test` column covers all the patterns pertinent to testing. A candidate pattern that falls into the `Test` column is *White Hats, Hack Thyself* [30], which advocates putting the system's security under test by gray hat hackers. The issue is that it cannot be identified by one single viewpoint, so it straddles the entire column. In our classification, we put it in the cell defined by the `Developer` row of `Application Architecture`, but clearly it can be in any other row in the test column.

Another example is the *Enterprise Security Services* [32] pattern, which guides an enterprise in selecting security services for its assets, based on a prevention, detection or response strategy. This pattern is used during the period when the long term business goals are set and it is therefore used by the `CEO` and the `General Manager` in the `Business Architecture` group. However, a more important issue is that the use of this pattern straddles all the columns because all the interrogatives are covered while setting up the long term business goals and choosing security services for them.

An even more intriguing example is the *Defense in Depth* [17] pattern. It advocates the use of security checks in multiple layers of the application. If we are bound with the constraint that every pattern has to be classified into one cell only we can provide an argument for classifying this pattern in the cell defined by the `Architect` row of `Application Architecture` group and the `Function` column, but the scope of this pattern is broader. Defense in depth is more of a security principle and therefore its context is defined by many rows and columns of the table.

Patterns that deal with non-functional quality attributes (*e.g.* security, performance, reliability *etc.*) tend to cover many levels of system development (many rows in the table) and also many interrogatives (many columns). The table is more successful in classifying functional patterns because the functional patterns are more in line with the perspectives and interrogatives of system development. Non-functional quality attributes are orthogonal to the aspects covered in the table.

The organizing table provides finer granularity than the Zachman framework but patterns are often classified based on other characteristics not mentioned in the table. It is impossible to distinguish between a security

pattern *Compartmentalization* [17] and a reliability pattern *Standby* [5] because both of them reside in the same cell, identified by the `Architect` row of the `Application Architecture` group and the `Function` column.

One symptom of an unsuccessful classification scheme is skewness, *i.e.* when too many patterns are classified to be in the same partition. The April 2006 snapshot of the security pattern classification table at patternshare reveals this skewness. At that point, the classification of security patterns was not integrated into the table. The table had 252 patterns from the Gang of Four book, the POSA book, the works of Gregor Hohpe and Bobby Woolf, Eric Evans and Martin Fowler and the Microsoft Patterns and Practices publications. The table had 30 patterns listed under the `Data` column, 10 patterns under the `Network` column, 1 pattern under the `People` column and 211 patterns under the `Function` column. This skewness is also evident in the security patterns classification summarized in table 1. 48 among the 59 security patterns were under the `Function` column. This is because most of the software patterns provide a mechanism to achieve some functionality. By having such a classification scheme, the population under the `Function` column is predictable.

Having a skewness in a classification system is not a sign of its inappropriateness. After all, the classification scheme for animals is heavily skewed towards the phylum Arthropod, which includes over three-fourths of all currently known living and fossil organisms. However, the taxonomy of the animal kingdom is hierarchical and from phylum to species, it provides the finer granularity that is required for the taxonomy to be successful. The organizing table does not support this hierarchy.

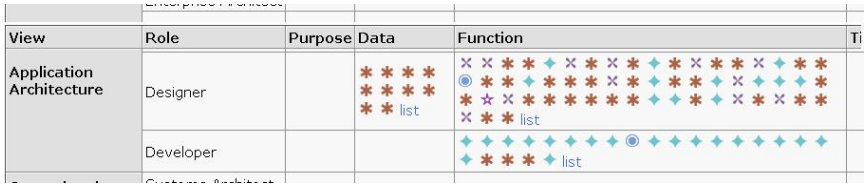

Figure 2: Using icon to denote meta-characteristics

From the pattern navigation viewpoint, having a lot of patterns in one cell makes it difficult to browse patterns. The organizing table uses different icons for the authors of patterns. For example, the 'x' icon in figure 2 is used to denote the patterns from the Gang of Four book, the '*' icon denotes the patterns from Martin Fowler's Patterns of Enterprise Application Architecture and the diamond shaped icon denotes the patterns coming

from Microsoft publications. Using different icons and colors do not add any value for pattern navigation, but the idea can be extended to create separate icons for security patterns, reliability patterns *etc.* Even finer granularity can be supported by introducing only as many icons.

4 Security Pattern Classification Proposals

Based on our experience in using the organizing table for classification, in this section, we shall make some concrete proposals for making the classification scheme more effective.

4.1 Better Classification with Additional Information

The major shortcoming of the organizing table is that it does not provide enough options for classification other than the flat partitioning scheme. The success of a classification scheme comes from the application of various parameters to get different views of the project. A classification mechanism incorporating multiple views is better for pattern mining and pattern navigation.

Using characteristic information. One way to augment the classification scheme is to introduce finer partitioning in the cells of the organizing table. An example would be to create a broad level classification inside each cell based on non-functional qualities like reliability, security, performance etc. The functional patterns create another broad level category. Then the patterns are classified further by defining finer metrics, *e.g.* the functional patterns at design level can be classified as structural, creational and behavioral following the Gang of Four classification scheme. The partitioning schemes are context sensitive and are not the same for all levels.

The main advantage of this is twofold. First it introduces finer granularity, so that patterns can be classified with more specificity. Second it uses domain specific vocabulary. One of the key problems of security patterns classification was that it did not depend on the security terminology. Using domain specific partitioning scheme would also make pattern navigation easier.

Using hierarchical classification. A hierarchical classification scheme has several advantages. The classification notation does not need to be mentioned; so an inexperienced user can have the advantage of using a hierarchical scheme without the distraction of the notation itself. Users can also broaden or narrow a search when required.

The advantages of hierarchical classification have been exploited in plant and animal taxonomy. We can leverage this by creating a parallel classification for each pattern. Each pattern will be classified using a hierarchical scheme in addition to the partitioning scheme. The partitioning scheme would then be used for broad level pattern navigation. Each partition will have a set of related patterns. The user can then use the hierarchical classification to identify the relevant pattern for his task.

The key issue in both the schemes is the choice of vocabulary used for creating the classification. The parameters chosen would have to be domain specific, *e.g.* the parameters for classifying reliability patterns would be different than the parameters for partitioning security patterns. In the next section we analyze the concepts and frameworks that can be adopted as the basis of security patterns classification.

Security Pattern	**Intent**
1. *Authenticator* [32]	Identity verifier of the subject.
2. *Authorization* [32]	Access policy definition for resources.
3. *Checkpointed System* [5]	System is a state machine and retains state information all the time.
4. *Compartmentalization* [17]	System architecture is composed of several compartments.
5. *Defense in Depth* [39]	Add security checks in multiple layers of the application.
6. *Full Access with Errors* [32]	Users can see all the options but can only access the options they are authorized to perform.
7. *Minefield* [25]	Introduces variations in the implementation of well-known security aspects of standard components.
8. *Policy Enforcement Point* [32]	Concentrates Identification and Authorization mechanisms at a point.
9. *Replicated System* [5]	Components are replicated to distribute workload.
10. *Secure Pre-forking* [14]	Run pre-forked processes for a limited time.
11. *Single Access Point* [32]	Single Entry Point for each process.
12. *Single Threaded Facade* [15]	Processes that communicate with the outside world are single-threaded.
13. *Subject Descriptor* [5]	Subject attributes are described in a data structure.
14. *Trust Partitioning* [15]	Validate messages between components and insulate them trustwise.

Table 2: A subset of security patterns from one cell of the Organizing Table (Full List in [34])

4.2 Classification Framework for Security Patterns

In this section, we shall describe several frameworks that can be applied as the basis to create an efficient classification framework for security patterns. As a running example we shall use the security patterns in the cell defined by the `Architect` row of the `Application Architecture` group and the

`Function` column. Table 2 summarizes 14 of the 27 patterns that are in this cell.

We have chosen patterns in the same cell because this will make the pattern navigation problem more obvious. In the next sections, we shall describe several schemes to use as the security domain vocabulary for the classification.

4.2.1 CIA Model

The CIA model [9] describes the 3 key issues of security, *i.e. C*onfidentiality, *I*ntegrity and *A*vailability. The interpretation of these three issues varies based on the application context. There has been a lot of debate over whether these three aspects complete the security viewpoint, or whether there is need for accountability, a fourth concept.

We shall consider confidentiality, integrity and availability as the parameters that create the classification. However, these issues have overlapping concerns.For example, consider the *Replicated System* pattern. This is a pattern for available systems. However, creating redundancy has the additional benefit of improving the integrity of data. Hence the *Replicated System* pattern lies in the overlapped region of integrity and availability. The *Defense In Depth* pattern applies to all three issues, and it is classified to be in the common region of all three aspects.

The classification based on the CIA model is shown in table 3.

Confidentiality. (*Number of Patterns in this Category: 10*) Authenticator, Authorization, Compartmentalization, Full Access with Errors, Policy Enforcement Point, Secure Preforking, Single Access Point, Single Threaded Facade, Subject Descriptor, Trust Partitioning.
Availability. (*1*) Checkpointed System.
Integrity and Availability. (*1*) Replicated System.
Confidentiality, Integrity and Availability. (*2*) Defense In Depth, Minefield.

Table 3: Classification based on CIA model

The advantage of this classification scheme is that it uses standard terminology from security literature to create the partition. The problem is that the partitions are not disjoint from each other and most of the patterns would fall in a gray area if their contribution is considered in detail.

4.2.2 Application Context

Another classification scheme considers the structure of the system and partitions the patterns based on which part of the system they are trying to secure. We can analyze the security of a system by considering core security, perimeter security and exterior security. The core security considers internal security mechanisms of a system. The perimeter security considers the authentication, authorization and security filtering related issues at system entry points. Exterior security considers data transmission security and secure communication protocols.

Table 4 summarizes the security patterns classification based on application context. The table lists 13 of the 14 security patterns.

Core Security. (*5*) Checkpointed System, Compartmentalization, Minefield, Secure Preforking, Subject Descriptor, Trust Partitioning.
Perimeter Security. (*6*) Authenticator, Authorization, Full Access with Errors, Policy Enforcement Point, Single Access Point, Single Threaded Facade.
Exterior Security (*1*) Replicated System.

Table 4: Security Patterns Classification based on Application Context

The advantage of this classification scheme is that it is disjoint, so there is a clear separation between the patterns. More general patterns like *Defense in Depth* cannot be classified using this scheme, because it impacts the core, the perimeter and the exterior security. Another problem is the lack of specificity because a lot of patterns would be classified as the core patterns without clear separation.

4.2.3 The Security Wheel

A security wheel represents the security features as the spokes in a wheel [35]. At the core of the hub of the wheel is the service or application that is under consideration. The spokes represent 12 core security services applicable to the service. These are authentication, authorization, confidentiality, integrity, policy, auditing, management, availability, compliance, logging, PKI and labeling. The edge of the wheel represent perimeter security.

The classification using the security wheel provides finer granularity than application context based classification. However, the metrics chosen as the security services have overlaps. For example the *Policy Enforcement Point* patterns can be classified under authentication, authorization and policy.

4.2.4 The McCumber Cube

The McCumber Cube [28] mechanism examines security in the context of information state. An analyst using the McCumber Cube identifies the information flow of a system, parses the information flow for security-relevant environment, and then maps the findings (security vulnerabilities and safeguards) on a rubik's cube like structure. The McCumber cube is shown in figure 3.

The X-axis represents the three primary categories of safeguards, *i.e.* technology, policy and procedure, and human factor. The Y-axis of the model represents information states of transmission, storage and processing. The vertical axis comprises the three security perspectives of confidentiality, integrity and availability. The cube is used for assessment and management of security risks in information technology systems.

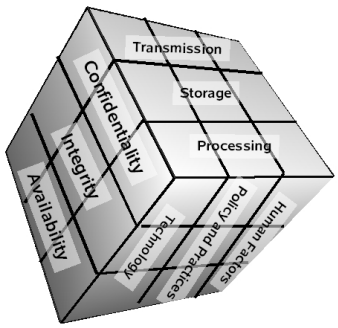

Figure 3: The McCumber Cube

The classification space of the McCumber cube is identified by an information state, a security perspective and a safeguard. The classification scheme based on the three security perspectives (the CIA model) has already been discussed. The information state concept weakly covers the classification scheme based on application context. The perimeter security and exterior security part of the system is encapsulated by the concept of transmission. The core security part of the system is covered by two concepts, storage and processing. The partitioning is done in this way because the scheme is based on an information-centric view of the system, whereas the other partitioning scheme is based on the architecture-centric view of the system. The third dimension in the cube defines a new concept that has not been covered by other classification schemes.

The classification of security patterns based on the McCumber cube is

listed in table 5. The patterns and then provide the classification as a three tuple of (safeguard, information state, security perspective). We use the '|' symbol to describe multiple factors.

Authenticator. (Technology, Transmission, Confidentiality).
Authorization. (Technology, Transmission, Confidentiality).
Checkpointed System. (Technology, Processing, Availability).
Compartmentalization. (Technology, Processing, Confidentiality).
Defense In Depth. (Technology, Storage | Processing | Transmission, Confidentiality | Integrity | Availability).
Full Access With Errors. (Technology, Processing, Confidentiality).
Minefield. (Technology, Storage | Processing | Transmission, Confidentiality).
Policy Enforcement Point. (Technology, Transmission, Confidentiality).
Replicated System. (Technology, Storage, Integrity | Availability).
Secure Pre-forking. (Technology, Processing, Confidentiality).
Single Access Point. (Technology, Transmission, Confidentiality).
Single Threaded Facade. (Technology, Processing, Confidentiality).
Subject Descriptor. (Technology, Processing, Confidentiality).
Trust Partitioning. (Technology, Processing, Confidentiality).

Table 5: Security Patterns Classification based on the McCumber Cube

The advantage of this classification scheme is that it integrates three separate viewpoints. The main disadvantage is that the dimensions of the cube does not provide a good partition. We have already discussed the shortcoming of a classification scheme based on the CIA model. Again, the perspectives covered by the McCumber cube's information state is only a subset of the interrogatives in the Microsoft enterprise architectural space organizing table organizing table (what, how and where). The safeguards viewpoint of the McCumber cube is effective if the model is used for assessment and management of risks, but it does not provide an effective partition for the classification of security patterns. This is why, all the patterns in table 5 are classified under the technology perspective.

4.2.5 Threat Modeling

Threat modeling is used for identification and prioritization of security vulnerabilities of a system. This allows the prioritization of mitigation effort. Frameworks for threat modeling have been proposed, *e.g.* the OCTAVE model from CERT [1], and the STRIDE/DREAD methodology from Microsoft [20, 36]. Classification process based on threat modeling is advantageous because it uses the concepts of security.

Microsoft's STRIDE model is used to categorize different threat types. STRIDE is an acronym of the following concepts.

- **Spoofing.** Attempt to gain access to a system using spoofed identity. A compromised system would have an access control vulnerability.
- **Tampering.** Manipulation of data during communication through the network. Integrity of the data is under threat.
- **Repudiation.** Denial of a user about their participation in a transaction. Availability of a resource is under threat.
- **Information disclosure.** Unwanted exposure and loss of confidentiality of private data.
- **Denial of service.** Attack on system availability.
- **Elevation of privilege.** A user with limited privilege forges the identity of a privileged user to gain privileged access to an application. Confidentiality, integrity and availability of a resource is under threat.

DREAD is a risk calculating mechanism. Each letter of the acronym stands for a threat attribute. Each of the attributes are ranked using one of 10 criticality ratings with 1 being the lowest rating and 10 being the highest rating. The attributes are,

- **Damage potential.** The damage that will be done if the vulnerability is exploited by the attacker.
- **Reproducibility.** The ease of exploiting a vulnerability.
- **Exploitability.** The required skill level to exploit a vulnerability.
- **Affected users.** The affected parties of an exploit.
- **Discoverability.** The ease of exploration and discovery of a system vulnerability.

The concepts from the STRIDE model can be used to classify the patterns. This classification is shown in table 6.

Spoofing. (*6*) Authenticator, Authorization, Full Access with Errors, Policy Enforcement Point, Single Access Point, Single Threaded Facade.
Information disclosure. (*3*) Compartmentalization, Subject Descriptor, Trust Partitioning
Denial of service. (*2*) Checkpointed System, Replicated System.
Elevation of privilege. (*1*) Secure Pre-forking.

Table 6: Security Patterns Classification based on STRIDE model

The table lists 12 patterns. Two patterns, *Minefield* and *Defense in*

Depth cannot be classified into a single group. Both of these patterns are applicable to all the security concepts of the STRIDE model. This means that these patterns are at a higher degree of granularity than other patterns.

4.2.6 Hierarchical Classification

Adding the partitioning schemes with a hierarchical taxonomy system would take advantage of both hierarchical and flat partition. Using a hierarchical scheme, the classification can integrate multiple parameters to create a tree based classification. We present a scheme that adopts the application context and STRIDE approach to create a classification tree. The tree is shown in figure 4.

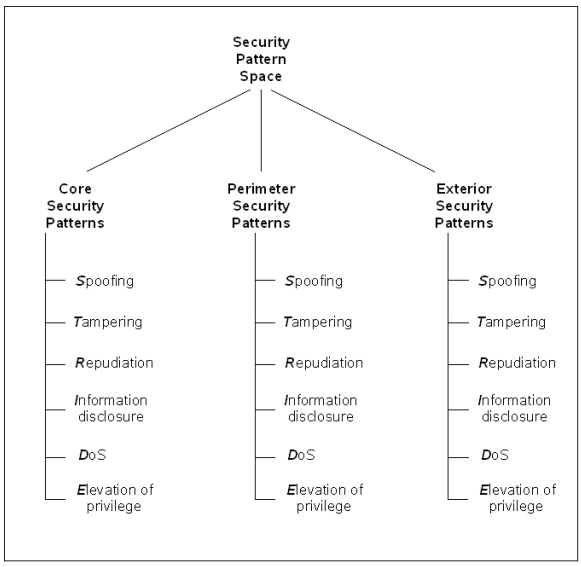

Figure 4: Hierarchical Classification

One thing that has not been considered in the classification schemes mentioned in this section is the security patterns that are relevant to the software development process. Kienzle *et al.* classified them as procedural patterns in their security patterns repository [25]. We are considering only structural patterns, patterns that are implemented in some product artifact. The procedural patterns can already be classified by the organizing table because the table encapsulates the procedural aspects.

These classification trees can be drawn for patterns for other non-functional qualities. The nodes of the classification tree would be ascertained

based on domain specific vocabulary. This would make pattern navigation easier and more meaningful. The classification scheme based on the STRIDE model was limited because patterns that cover multiple concepts could not be classified using a stratified scheme. With a hierarchical scheme, the patterns need not be at the leaf of the tree only. Patterns in the internal nodes are higher level patterns than those that are in the leaf. Security principles, *e.g.Defense in Depth*, reside at the highest level of the classification hierarchy. Figure 5 summarizes the classification.

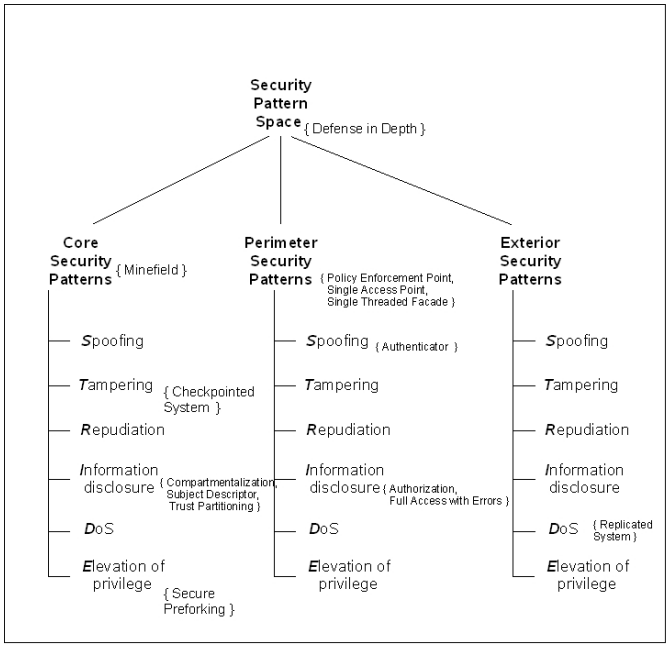

Figure 5: Classification of Security Patterns using the tree

The organizing table lists the patterns using separate icons (or some color scheme) for showing the broad level classification of patterns. The detailed classification cannot be shown in a two-dimensional table. The parent nodes of the tree encapsulate the more specific child nodes. We can assume that each cell in the table is partitioned into smaller subspaces, and as we traverse these subspaces with more detail, the corresponding partitions with the finer granularity becomes apparent. The high level view of the organizing table hides these complexities, and only shows the broad partitions using icons.

5 Conclusion

The security pattern classification scheme proposed in this paper integrates hierarchical classification mechanism with the classical table based approach. The parameters used for hierarchical classification may change with the introduction of new security patterns. This is not unusual, because classification schemes are supposed to change in order to better encapsulate the changed reality. The biological taxonomy is still undergoing changes at different levels as new plant and animal species are discovered.

A useful extension of the proposed classification scheme would be the incorporation of meta-information specifying the relationship between patterns. This can be augmented into a pattern language that would help the users navigate through patterns in a logical way. A pattern language would not only help pattern classification, but also would be useful for documenting design process guided by related patterns.

References

[1] C. Alberts and A. Dorofee. *Managing information security risks: The OCTAVE approach*. Addison-Wesley Professional, 9 Jul 2002.

[2] C. Alexander. *The timeless way of building*. Oxford University Press, 1979.

[3] C. Alexander, S. Ishakawa, and M. Silverstein. *A pattern language*. Oxford University Press, New York, 1977.

[4] K. Beck. *Extreme programming explained: Embrace change*. Addison-Wesley, 1999.

[5] B. Blakley and C. Heath. Security design patterns technical guide - version 1. *Open Group (OG)*, led by Bob Blakley and Craig Heath. 2004. `http://www.opengroup.org/security/gsp.htm`.

[6] A. Braga, C. Rubira, and R. Dahab. Tropyc: A pattern language for cryptographic software. In PLoP 1998 Proceedings, 1998.

[7] F. L. Brown Jr., J. DiVietri, G. D. Villegas, and E. B. Fernandez. The authenticator pattern. 1999.

[8] F. Buschmann, R. Meunier, H. Rohnert, P. Sommerlad, and M. Stal. *A system of patterns - Pattern oriented software architecture*. Wiley, 1996.

[9] Commission of European Communities. Information technology security evaluation criteria, version 1.2, 1991.

[10] M. W. Eichin and J. A. Rochlis. With microscope and tweezers: An analysis of the Internet virus of November 1988. In *IEEE Security and Privacy*, pages 326–343, 1989.

[11] E. B. Fernandez and J. C. Sinibaldi. More patterns for operating systems access control. In Proceedings of the European Conference on Patterns Language of Programming (EuroPLoP'03), 2003.

[12] E. Gamma, R. Helm, R. Johnson, and J. Vlissides. *Design patterns: Elements of reusable object-oriented software*. Addison-Wesley, Reading/MA, 1995.

[13] M. Hafiz. Unique atomic chunks: A pattern for security and reliability. In Proceedings of the 11th Conference on Patterns Language of Programming (PLoP'04). `http://hillside.net/plop/2004/papers/mhafiz0/PLoP2004_mhafiz0_0.doc`, 2004.

[14] M. Hafiz. Secure pre-forking: A pattern for security and performance. In Proceedings of the 12th Conference on Patterns Language of Programming (PLoP'05). `http://hillside.net/plop/2005/proceedings/PLoP2005_mhafiz0_2.pdf`, 2005.

[15] M. Hafiz. Security architecture of mail transfer agents. Master's thesis, University of Illinois at Urbana-Champaign, 2005.

[16] M. Hafiz. A collection of privacy design patterns. In Proceedings of the 13th Conference on Patterns Language of Programming (PLoP'06), 2006.

[17] M. Hafiz, R. Johnson, and R. Afandi. The security architecture of *qmail*. In Proceedings of the 11th Conference on Patterns Language of Programming (PLoP'04). `http://hillside.net/plop/2004/papers/mhafiz1/PLoP2004_mhafiz1_0.pdf`, 2004.

[18] J. Heaney, D. Hybertson, A. Reedy, S.Chapin, T. Bollinger, D. Williams, and M. Kirwan Jr. Information assurance for enterprise engineering. In Proceedings of the 9th Conference on Patterns Language of Programming (PLoP'02), 2002.

[19] 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.

[20] M. Howard and D. C. LeBlanc. *Writing secure code*. Microsoft Press, second edition, 4 Dec 2002.

[21] D. Hybertson, J. Heaney, and A. Reedy. Conceptual aspects of security patterns. 2002.

[22] IEEE Std 1471-2000. IEEE recommended practice for architectural description of software-intensive systems. `http://standards.ieee.org/reading/ieee/std_public/description/se/1471-2000_desc.html`, 2000.

[23] ISO 7498-2. Information processing system - Open systems interconnections - Basic reference model - Part 2: Security architecture. Technical report, ISO, 1989.

[24] ISO/IEC 13335-1. Information technology - Guidelines for the management of IT security - Part 1: Concepts and models for IT security. Technical report, ISO, 1996.

[25] D. M. Kienzle, M. C. Elder, D. Tyree, and J. Edwards-Hewitt. Security patterns repository version 1.0. `http://www.scrypt.net/~celer/securitypatterns/`, 2002.

[26] M. Kircher and P. Jain. *Pattern-oriented software architecture, Patterns for resource management*. John Wiley and Sons, June 8, 2004.

[27] Mark Goodyear, Andersen Consulting, editor. *Enterprise System Architectures: Building Client Server and Web Based Systems*. CRC Press, Sep 28 1999.

[28] J. R. McCumber. Information systems security: A comprehensive model. In *Proc. 14th NIST-NCSC National Computer Security Conference, Washington D.C.*, pages 328–337, October 1991.

[29] S. Romanosky. Enterprise security patterns. http://citeseer.ist.psu.edu/romanosky02enterprise.html, 2002.

[30] S. Romanosky. Security design patterns part 1. http://citeseer.ist.psu.edu/575199.html, November, 2001.

[31] D. C. Schmidt, M. Stal, H. Rohnert, and F. Buschmann. *Pattern-oriented software architecture volume 2 – Networked and concurrent objects*. John Wiley and Sons, 2000.

[32] 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.

[33] M. Schumacher and U. Roedig. Security engineering with patterns. In Proceedings of the 8th Conference on Patterns Language of Programming (PLoP'01). http://citeseer.ist.psu.edu/schumacher02security.html, 2001.

[34] Security patterns catalog. Patternshare Wiki page on security patterns. Maintained by Munawar Hafiz. http://patternshare.org/default.aspx/Home.UIUC.HomePage.

[35] 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, October 2005.

[36] F. Swiderski and W. Snyder. *Threat modeling*. Microsoft Press, 14 Jul 2004.

[37] J. Tidwell. *Designing interfaces : Patterns for effective interaction design*. O'Reilly Media, Inc., November 2005.

[38] D. Trowbridge, W. Cunningham, M. Evans, L. Brader, and P. Slater. Describing the enterprise architectural space. *MSDN*, June 2004.

[39] J. Viega and G. McGraw. *Building secure software - How to avoid security problems the right way*. Addison-Wesley, 2001.

[40] Wikipedia. Taxonomy — Wikipedia, The free encyclopedia, 2006.

[41] J. Yoder and J. Barcalow. Architectural patterns for enabling application security. In Proceedings of the 4th Conference on Patterns Language of Programming (PLoP'97). http://citeseer.ist.psu.edu/yoder98architectural.html, 1997.

[42] J. A. Zachman. A framework for information systems architecture. *IBM Systems Journal*, 26(3), 1987.