Towards an Organization of Security Patterns

Munawar Hafiz, Paul Adamczyk, Ralph E. Johnson
University of Illinois
{mhafiz, padamczy, johnson}@cs.uiuc.edu

Abstract
Every empire, after a period of rapid expansion, needs some time for consolidation or it risks disintegration. The expansion of software patterns has produced a large body of work that needs organization. This paper documents early efforts of consolidating and organizing a subset of software patterns in the domain of security. People trying to organize patterns for other domains can benefit from these lessons.

Keywords
D.2.11.e Patterns, K.6.5 Security and Protection

Early software systems communicated in predefined ways and therefore were easy to secure. The Internet has shown the inadequacy of the old ways of ensuring security and reliability of software systems. To this day, building secure systems is difficult, while retrofitting existing systems to introduce security is even harder. For example, sendmail, the most popular mail transfer agent (MTA), has been plagued with security vulnerabilities[1,2] ever since it was first exploited by the Morris worm in 1988[3]. Since then, sendmail developers have been addressing each newly discovered vulnerability by following the 'penetrate and patch' paradigm. However new patches often result in opening up new security holes. As new security vulnerabilities are constantly unraveled by attackers, security architects find it difficult to anticipate new attacks proactively.

Security patterns are one way of collecting and disseminating security knowledge. In a short period of time, many collections of security patterns appeared (see the sidebar, "How Many Security Patterns?"). But the sheer number of existing security patterns makes it difficult for pattern users to find the most appropriate solutions.

Proper organization of patterns is useful for both pattern writers and pattern users. One aspect of pattern organization is classification, i.e. grouping the patterns into small, correlated sets. The pattern writers can then place new patterns in the appropriate category. Good organization also makes cross-referencing between related patterns easier, because patterns that provide alternative solutions for similar problems are grouped together. Users can find a pattern that solves their particular problem more easily. Good organization includes guidelines for finding relationships and applying patterns in sequence. Helping a user navigate through the inter-related patterns is another objective of pattern organization.

There is no proper organization of security patterns. Several schemes for organizing the patterns have been used so far, but all of these approaches fall short of successfully organizing all the security patterns. In this article, we explore various classification schemes of security patterns. We analyze the current approaches and use their shortcomings to charter a roadmap towards an alternative pattern organization methodology that is also applicable to software patterns from other domains.

How Many Security Patterns?

Yoder and Barcalow wrote the first paper on security patterns[1] in 1997. Others followed it by making security pattern catalogs either individually[2,3,4], or by formal[5] and informal collaboration[6]. Recently, three books on security patterns have been published.

Steel et. al. wrote a book that covers architectural security patterns for J2EE-based applications, Web services and identity management[7]. The book present the fundamentals of Java application security and documents 23 security patterns. It was published in October, 2005.

Markus Schumacher led a working group that produced a book in December, 2005[8]. This book includes 46 security patterns from the domains of enterprise security and risk management, identification and authentication, access control, accounting, firewall architecture, and secure Internet applications.

Microsoft's Patterns and Practices group published a security patterns book[9] accompanying the release of Web Services Enhancement (WSE) 3.0 in March, 2006. The book is a tutorial for using WSE 3.0 in Web services development. It lists 18 patterns that address authentication, message protection, transport and message layer security, resource access, service boundary protection and service deployment. All these patterns are described from the perspective of Microsoft technology.

The patternshare website (www.patternshare.org) is a single repository to describe all kinds of software patterns. One of the goals of patternshare is to develop a uniform vocabulary for practitioners by combining all patterns that differ only by the name into one. The patternshare repository of security patterns summarizes the security patterns from various sources and eliminates overlaps. We are maintaining this repository while the patternshare as a whole is managed by the Hillside Group. In December 2006, the repository contained 90 unique security patterns.

New security vulnerabilities are exposed daily. The problems uncovered by these vulnerabilities and the solutions developed to combat them will add many more security patterns to the arsenal of developers of secure software systems in the years to come.

References

1. 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.
2. S. Romanosky. Security design patterns part 1. http://citeseer.ist.psu.edu/575199.html, November, 2001.
3. M. Hafiz. Security Architecture of Mail Transfer Agents. Master's Thesis. University of Illinois at Urbana-Champaign, July 2005.
4. M. Hafiz. A collection of privacy design patterns. In Proceedings of the 13th Conference on Patterns Languages of Programming (PLoP'06), 2006.
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. D. M. Kienzle, M. C. Elder, D. Tyree, and J. Edwards-Hewitt. Security patterns repository, 2002.
7. 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.
8. 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.

9. 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.

Steps in organizing patterns

Any organization effort must begin by collecting the existing patterns that need to be organized. The sidebar, "How Many Security Patterns?" describes the sources of security patterns. They are independent collections that require a single classification.

The second step of pattern organization is eliminating the overlaps between pattern catalogs. Since catalogs are developed independently, many authors describe the same or similar concepts, but give them different names. The overlaps between patterns are evident only when the patterns are put together. For example, patterns for authentication have been described in a number of catalogs. The Authenticator[4] pattern is listed in the Wiley security patterns book. Authenticator is essentially a Policy Enforcement Point[5]. Sasha Romanosky describes the same pattern under the name Security Provider[6]. This pattern is also listed in the security patterns repository by Kienzle et al. as Authenticated Session[7]. The Core Security Patterns book[8] describes the Authentication Enforcer pattern from the perspective of Java tools. The Microsoft security patterns book[9] describes authentication in terms of various strategies supported by the WSE 3.0 framework. It describes two patterns Direct Authentication and Brokered Authentication. The latter is specialized even more into different authentication strategies, described in separate patterns – Brokered Authentication: Kerberos, Brokered Authentication: X509 PKI and Brokered Authentication: Security Token Service (STS). Since they all represent one type of solution to a problem in a shared context, they should be documented as *one* pattern.

The third step is to define (or use an existing) categorization that splits patterns into disjoint sets. Initial attempts to organize security patterns were meant to organize patterns within a single catalog. Most of the schemes choose partitions that are too general to be useful. For example, the Open Group catalog classifies the patterns into two groups based on applicability; patterns for protected systems and patterns for available systems. Having only two broad groups does not help in organization. Kienzle et. al. classify their security patterns into structural patterns and procedural patterns. The structural patterns describe how to implement the software. The procedural patterns describe how to improve the development process. For example, Grey Hats is a procedural pattern, because it explains how to set up testing environment that most closely resemble the conditions of the system under security attack. However, almost all known security patterns are structural. 84 out of 90 patterns on patternshare are. This disparity makes Kienzle's classification scheme infeasible for large collections of patterns. While these schemes worked for their respective catalogs, they are not rich enough to provide organization of all security patterns.

Ideally, the organization scheme should be as comprehensive as the biological taxonomy. The efficacy of the biological taxonomy comes from the metrics that are used to create the partitions. But what are the appropriate metrics for partitioning the patterns? Some ideas come from the definition of pattern – "a solution to a problem in a context." More

elaborate patterns organization schemes use these key elements – domain concepts, the context, or the problem of patterns – as the metric of classification. We describe the following classification schemes based on the metrics.

- Classification scheme based on domain concepts – Confidentiality, integrity and availability, the fundamental concepts[10] of security.
- Classification scheme based on context
  - Application context – The part of the system where a pattern is applied.
  - Stakeholders and their viewpoints – Zachman's tabular classification framework[11] and its extensions.
- Classification scheme based on problem domain – Threat model.

Safe Data Buffer pattern

Context
You are designing a system in a programming language that does not have built-in array bounds checking.

Problem
Buffer overflow occurs when a process attempts to store data beyond the boundary of a fixed-length buffer. The problem is caused by bad programming practice. If every buffer handling routine checked allocated memory and operated within that memory bounds, buffer overflow would not occur. In practice, the buffer handling routines do not handle these tasks. An attacker exploiting a buffer overflow can execute arbitrary code and take complete control of the operating system. How can you design buffers that do not have this vulnerability?

Forces
1. Programming languages that do not check for buffer bounds are used in practice.
2. Programmers can prevent buffer overflow by checking buffer bounds every time they are writing a buffer handling routine; but they instead call the library functions without any pre-conditions check.
3. Buffer handling library functions do not check for buffer bounds.

Solution
Represent the buffer with a data structure that includes the length information and allocated memory information. In all buffer handling routines, check for length and available memory before updating the data buffer.

Example
qmail is a mail server written in C. It has a custom-written string library. Strings are not null-terminated. Instead, they are encapsulated in a structure named *stralloc*. This data structure keeps the length information of the string buffer. All string-manipulation functions check if the input data fits the buffer before attempting to update the buffer.

The data structure is as follows.

```
typedef struct stralloc {
        char *s;            // pointer to the string or 0 if unallocated
        unsigned int len;   // length of the buffer, in bytes
        unsigned int a;     // count of allocated bytes in the string
}
```

Source
This pattern appeared in the masters' thesis of Munawar Hafiz titled "Security Architecture of Mail Transfer Agents."[11]

Candidate Security Patterns
To illustrate different organization schemes, we will use one set of patterns and apply it to all the schemes as we discuss them. We have randomly chosen 14 security patterns from the security pattern repository at patternshare. Table 1 lists these patterns along with information about their source and a short description of their intent. To give the reader a sneak peek at the actual patterns, the sidebar "Safe Data Buffer pattern" includes the complete pattern as available at patternshare.

Security Pattern	Intent
Authenticator[4]	Verify the identity of the subject.
Authorization[4]	Define the access policy for resources.
Checkpointed System[12]	Design the system as a state machine that retains the state information all the time.
Defense in Depth[13]	Add security checks in multiple layers of the application.
Exception Shielding[9]	Sanitize unsafe exceptions with exceptions that are safe by design (exceptions that do not reveal sensitive information like detailed error code and stack trace).
Minefield[7]	Introduce variations in the implementation of well-known security aspects of standard components.
Password Synchronizer[8]	Centralize management of synchronizing user credentials across different application systems via programmatic interfaces.
Policy Enforcement Point[5]	Concentrate identification, authentication and authorization mechanisms at one point.
Replicated System[12]	Replicate components to distribute workload.
Safe Data Buffer[14]	Include the length information of the memory buffer and check it before allocation.
Secure Pre-forking[14]	Run pre-forked processes for a limited time.
Single Access Point[4]	Define a single entry point for the system.
Subject Descriptor[12]	Describe attributes of the subject in a data structure.
Grey Hats[6]	Have testers plan and execute attacks on the system to test its security.

Table 1: Candidate security patterns and their summaries

Using the domain concepts: CIA Model

The simplest approach for classification is to identify the fundamental concepts in a domain and to use them to create the partition. The CIA model[10] describes the 3 key aspects of security – Confidentiality, Integrity and Availability.

When security patterns are partitioned using confidentiality, integrity and availability as classification parameters, many patterns fall into the overlapping region, because these security concepts have overlapping concerns.

Consider the Replicated System pattern as an example. This pattern increases availability of a system. 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 classification based on the CIA model is shown in table 2. This classification scheme is not concrete enough – 5 patterns belong to more than one category and 3 patterns are applicable to all of them. Moreover, most of the patterns fall into one category, Confidentiality.

Confidentiality. (*Number of Patterns in this Category: 7*) Authenticator, Authorization, Exception Shielding, Policy Enforcement Point, Secure Pre-forking, Single Access Point, Subject Descriptor.
Integrity. (*1*) Safe Data Buffer.
Availability. (*1*) Checkpointed System.
Integrity and Availability. (*2*) Password Synchronizer, Replicated System.
Confidentiality, Integrity and Availability. (*3*) Defense In Depth, Minefield, Grey Hats

Table 2: Classification based on CIA model

But this classification scheme has one important advantage – it uses widely known fundamental security concepts to create the partition. Classification schemes should use existing terminology whenever available.

Using the context: Simple Application Context

The context, when and where to apply a pattern, provides a good metric for pattern classification. Pattern writers often begin by describing the context. Pattern users are familiar with the context where they are trying to apply a pattern. So, anyone interested in patterns typically understands a classification scheme based on the pattern context.

We can compare the architecture of a system with the structure of a secure military base. In a military base, security mechanisms are installed at the entry point and around the perimeter. Another level of security mechanisms is installed deep inside the base where the critical infrastructure exists. The communication of logistics between military bases is also made secure. Similarly, a simple classification based on the application context classifies the security patterns considering the part of the system they are trying to secure – application core, application perimeter and application exterior.

Core security patterns consider the security mechanisms inside of a system. Perimeter security patterns consider the authentication, authorization and security filtering related issues at system entry points. Exterior security patterns deal with secure data transmission and communication protocols.

Table 3 summarizes the security patterns classification based on the application context.

Core Security. (6) Checkpointed System, Exception Shielding, Minefield, Safe Data Buffer, Secure Pre-forking, Subject Descriptor.
Perimeter Security. (4) Authenticator, Authorization, Policy Enforcement Point, Single Access Point.
Exterior Security. (2) Password Synchronizer, Replicated System.
All categories. (2) Defense in Depth, Grey Hats.

Table 3: Security patterns classification based on application context

The advantage of this classification scheme is that the concrete patterns are easily assigned to a single category. However, more general patterns, Defense in Depth and Grey Hats cannot be classified using this scheme, because they impact the core, the perimeter and the exterior security. Another problem is that most patterns fall into the core security and this scheme does not provide a way to separate them into smaller, more cohesive subgroups.

Using the context: Zachman Framework

The security pattern classification efforts we discussed so far rely on a flat partitioning scheme.

More complex classification schemes are based on the Zachman framework[11]. Inspired by the different documents produced during different phases of building development, Zachman identified five types of specifications produced during the software development process that are used by different stakeholders, i.e. the customer, the system architect, the designer, the developer and the module developer. These five viewpoints comprise five rows in the table. The columns of the table represent the concerns in terms of six interrogatives – data (what?), function (how?), network (where?), people (who?), time (when?) and motivation (why?).

Microsoft developed a tabular classification scheme based on the Zachman framework for all patterns[15] (see table 4). This scheme retains the two-dimensional tabular space, but divides each row into smaller groups based on the architectural standard description from IEEE 1471[16]. This multi-dimensional classification scheme has more categories allowing more specific classification. An additional column (Test column) was added to include the software debugging concern.

The benefit of the Microsoft scheme is that it clearly identifies the context of each pattern making pattern navigation easier. Users treat specific cells in the table similarly to the context of their problem; and look for solutions there. This classification scheme also helps to identify 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 a similar pattern could be documented for that context and that these two patterns are related.

The security patterns tend to fit many roles of system development (many rows in the table) and also many interrogatives (many columns). This tabular approach is more successful in classifying functional patterns, which correspond better to the perspectives and interrogatives of system development; hence their scope can be identified by one cell in the table. Non-functional quality attributes (such as security or performance) are orthogonal to the aspects covered in the table and they cannot be contained by one cell in the table. Table 4 shows 11 of the 14 patterns classified according to the Microsoft's scheme.

		Function	**Data**	**Test**
Application Architecture	**Architect**	Checkpointed System, Policy Enforcement Point, Replicated System, Secure Pre-forking, Single Access Point.		
	Designer	Authenticator, Password Synchronizer	Exception Shielding, Subject Descriptor	
	Developer		Safe Data Buffer	Grey Hats

Table 4: Security patterns classification based on Microsoft classification scheme (partial view of the table)

Patterns like Minefield and Defense in Depth cannot be assigned to a single cell. In fact, the broad context of Defense in Depth places it in all the rows and columns of the table. Authorization is another pattern that should be placed in multiple cells, because it is applicable to the roles of architect and designer (and perhaps even developer).

Another symptom that the tabular scheme is unsuccessful in pattern classification is the skewness, i.e. patterns are concentrated in particular cells of the table. The pattern classification table at patternshare included 90 security patterns. Of the patterns, 63 were in the 'function' column. Among the 14 candidate patterns in our running example, 10 are classified in the 'function' column (including Minefield, Authorization, and Defense in Depth—the three patterns that are not shown in Table 4). This is because most of the software patterns provide a mechanism to achieve some functionality. A pattern classification scheme that considers the 'function' aspect is skewed.

However, skewness does not rule out a classification scheme. 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 fine granularity of classification. The organizing table does not illustrate this hierarchy.

The tabular classification scheme applied to classify security patterns shows that the context of the pattern is not an effective metric of classification. It shows also that multidimensional schemes are not necessarily better than flat ones.

Using the problem domain: Considering the Threat Model

The tabular classification scheme based on the Zachman framework, just like the classification based on the domain concepts, fails to account for general patterns that span multiple categories. In search of a more appropriate scheme, we tried to classify patterns based on the problems they solve. The problem space of security patterns can be partitioned using the threat model.

Threat modeling is used to identify and prioritize security vulnerabilities of a system. This allows the prioritization of mitigation effort. Several frameworks for threat modeling have been proposed, e.g. the STRIDE[17] model. Classification process based on threat modeling is more intuitive, because it uses the security problems that the patterns solve.

The STRIDE model is used to categorize different types of threats that systems face. STRIDE is an acronym containing the following concepts.

- *Spoofing* – Attempt to gain access to a system using a forged identity. A compromised system would provide an authorized user access to sensitive data.
- *Tampering* – Corruption of data during communication through the network. Integrity of the data is under threat.
- *Repudiation* – User's refusal to acknowledge participation in a transaction.
- *Information disclosure* – Unwanted exposure and loss of confidentiality of private data.
- *Denial of service* – Attack on system availability.
- *Elevation of privilege* – Attempt to raise the privilege level by exploiting some vulnerability. Confidentiality, integrity and availability of a resource are under threat.

Table 5 lists the classification of the candidate security patterns based on the STRIDE model. Not surprisingly, this scheme also fails to include all the patterns.

Spoofing. (*3*) Authenticator, Password Synchronizer, Subject Descriptor.
Tampering. (*3*) Checkpointed System, Safe Data Buffer, Single Access Point.
Information disclosure. (*2*) Authorization, Exception Shielding.
Denial of service. (*1*) Replicated System.
Elevation of privilege. (*1*) Secure Pre-forking.
All categories. (*4*) Defense in Depth, Minefield, Policy Enforcement Point, Grey Hats.

Table 5: Security patterns classification based on the STRIDE model

Four patterns, Defense in Depth, Minefield, Policy Enforcement Point and Grey Hats cannot be classified into one category. All these patterns are applicable to all the security concepts of the STRIDE model.

By now, it should be obvious, that regardless of the classification scheme, some patterns will always belong to multiple categories. This is the distinguishing characteristic of the more abstract patterns that present a guiding principle rather than a precise solution. Applications of such principled patterns are often refined further by applying smaller, more concrete patterns. Other patterns fit in some partitioning schemes well, but not other schemes.

The STRIDE-based scheme, just like all the previous ones, does not solve the key issue of classification – how to classify patterns that straddle multiple categories. The classification based on the STRIDE model is single-dimensional. Of all the schemes discussed so far, the tabular scheme had the most unique categories, because it was multi-dimensional. It would seem that adding more dimensions to STRIDE should solve this problem. We combined the STRIDE-based classification with another single-dimensional scheme, the simple application context (from table 3). Classifying the patterns from both the problem and the context perspective creates a better partition. The resulting partition puts almost every pattern in its own cell, but still fails to properly separate others. (We do not show the resulting table. It will be refined further in the next section.) While patterns that apply to a single context, Minefield (core) and Policy Enforcement Point (perimeter) can be separated, the most general patterns – in our example, Defense in Depth and Grey Hats – do not fit this scheme. These general patterns cannot fit any scheme that treats all patterns equally. These patterns have a different level of complexity that suggests using a hierarchy.

Adding hierarchy to classification

Biologists solved the organization problem with taxonomies. Taxonomy is a classification of things and the principles underlying the classification. The biological classification organizes its concepts in a hierarchy. The hierarchy is a tree where the root node provides a general concept that is refined by each descendant.

Adding a hierarchical taxonomy system to a flat partitioning scheme means that some patterns can be placed higher in the hierarchy. A hierarchical scheme allows partitioning patterns along several dimensions, based on context, and problem, and generality of patterns.

Figure 1 illustrates the proposed classification of the 14 patterns. The nodes of the classification tree are ascertained based on domain-specific vocabulary. This makes pattern navigation easier and more meaningful. The classification scheme based on the STRIDE model alone is limited, because patterns that address multiple concepts could not be classified using a flat scheme. With a hierarchical scheme, the patterns need not be at the leaf of the tree only. Patterns in the internal nodes are at a higher level than those that are in the leaf; they address multiple threats of the STRIDE model. For example, Policy Enforcement Point pattern addresses multiple threats of the STRIDE model (spoofing, tampering, information disclosure and DoS). Patterns at the root of the tree are applicable to multiple contexts (core, perimeter and exterior). General patterns, e.g. Defense in Depth, reside at the highest level of the classification hierarchy because they are applicable to all the contexts.

Our classification scheme uses the STRIDE model as classification criteria in all the branches. This is different from biological taxonomy where subgroups are divided differently. For example, mammals and insects are classified using different characteristics. Mammals are divided into the canine family, the feline family, the primates, the rodents, etc. Insects, on the other hand, have neither teeth nor bone. Divisions into families are based on wings, mouth parts, antennae, etc. Our scheme is sufficient for current classification, but future research might introduce variation in the hierarchy.

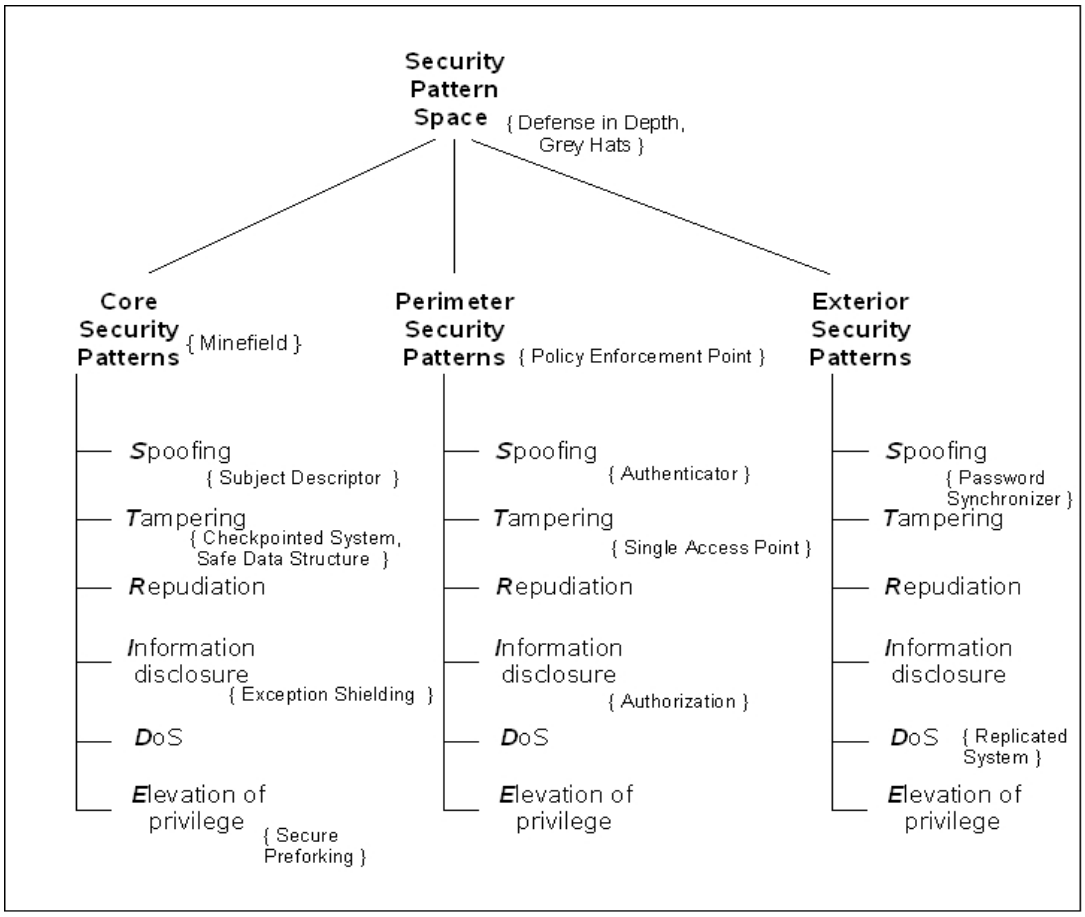

Figure 1: Classification of security patterns using the tree

Classification or Navigation?

Classification is one aspect of the organization effort. It divides patterns into small groups. Within each group, patterns are not necessarily related, but share a common problem domain and application context. Another aspect of organization is navigation – the ability to guide the reader between the related patterns. Well-organized collections of patterns describe inter-pattern relationships in more details.

No pattern is an island. Patterns form a chain in which each pattern is derived from some more general patterns and depends on the more specific patterns it contains. One pattern creates problems that another solves. One pattern creates the context for another pattern.

One pattern solves the high-level problem and the other solves lower level problems. An organization that illustrates such relationships between patterns is a pattern language[18]. More than just a classification, a pattern language helps pattern users apply the patterns.

To illustrate how a pattern language works, we selected a portion of the 14 randomly chosen candidate patterns. Collectively, they do not represent a true pattern language. However, 6 out of these 14 patterns are associated with access control. The relationships between them are shown in figure 2. Patterns are shown as boxes, and the relationships as arrows.

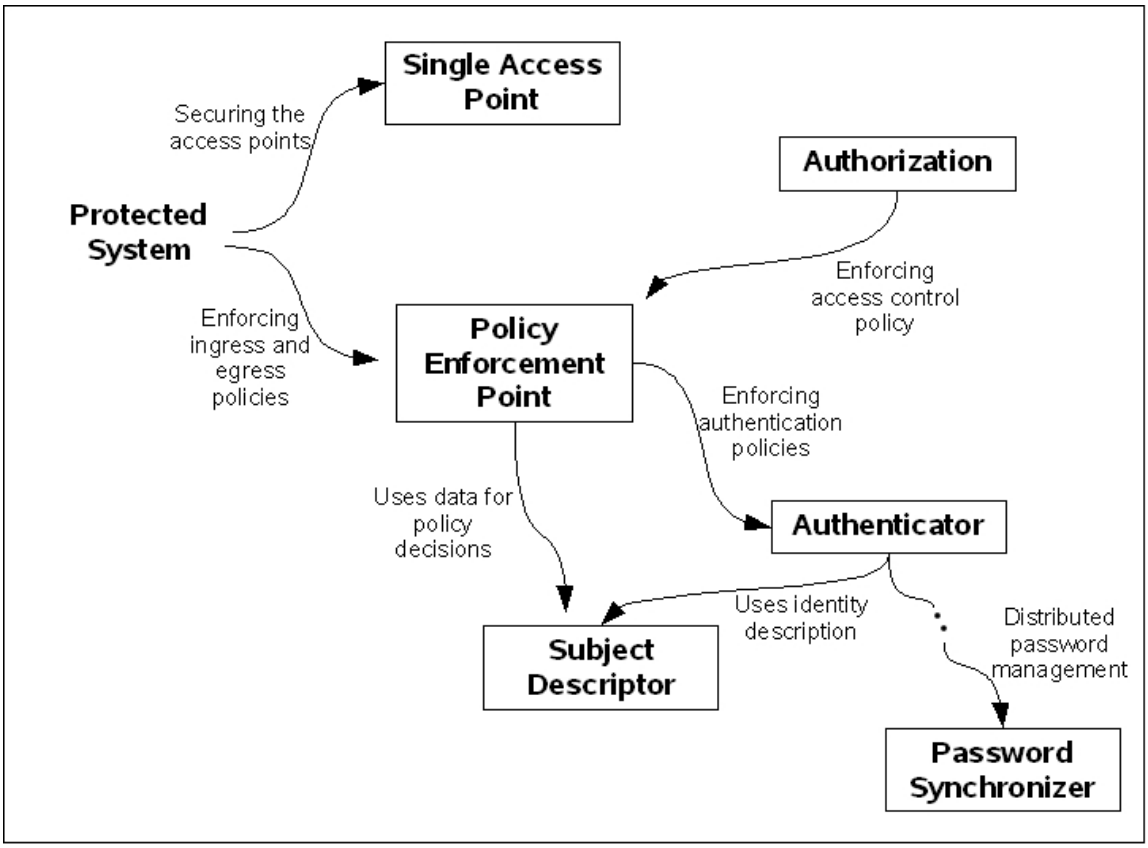

Figure 2: Pattern language describing the relationships of 6 patterns

Let us give some examples of how this pattern language guides the users. A user might be interested in applying patterns to secure the system perimeter. She would need to minimize the access points (using Single Access Point pattern) and then define guards at these points (using Policy Enforcement Point). The guards check the incoming and outgoing data traffic based on the ingress and egress policies of the system. Another user, interested in authorization, would find that authorization is performed by the definition and enforcement of access control policies. These policies are enforced at the Policy Enforcement Point. Yet another user might be interested about authentication only. She would first create minimal access points where authentication policies will be enforced by the Authenticator pattern. The data used to grant authentication decision are defined by the Subject Descriptor. Distributed password management is done by the Password Synchronizer pattern. Authenticator and Password Synchronizer are not

directly related; other patterns would need to be applied after Authenticator before Password Synchronizer can be used. Many other patterns, related with these patterns, are not shown in the figure (indicated by dots).

Creating a pattern language for all the security patterns is a non-trivial task. The concepts defining the relationships between security patterns are selected ad hoc. Therefore, this definition might be useful for a group of users, but completely useless for others. Finding the standard relationship between patterns to enable navigation for everyone is an even more difficult problem than classification of security patterns.

Summary

We analyzed various classification schemes for security patterns and proposed a new one. Our classification scheme uses the threat model and application context to partition patterns. It further organizes the patterns in a hierarchy. We evaluated these schemes by applying them to 14 randomly selected security patterns. We also described how a pattern language helps in organizing patterns. Rather than classifying them, a pattern language describes how to navigate between the patterns and how to apply multiple patterns in sequence.

Many issues arise from organizing security patterns. The ways in which they are solved provide lessons for people trying to organize patterns for other domains.

- Patterns should be classified in multiple dimensions – e.g. context, problem, domain-specific partitions. Classifications based on a single metric cannot partition the pattern space sufficiently.
- Arriving at a good scheme requires iterative refinement of the classification. Different metrics need to be combined to find the most appropriate combinations.
- Related patterns often form well-structured relationships, e.g. hierarchies. The organization scheme should preserve such relationships between patterns.
- In addition to classification, navigation based on relationships between patterns helps to better understand the patterns space.

References

1. D. J. Bernstein. sendmail disasters. http://cr.yp.to/maildisasters/sendmail.html, 1997.
2. Security Focus. sendmail vulnerabilities. http://securityfocus.com/bid
3. 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, pp. 326-343, 1989
4. 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.
5. J. Yoder and J. Barcalow. Architectural patterns for enabling application security. In Proceedings of the 4th Conference on Patterns Languages of Programming (PLoP'97), 1997.
6. S. Romanosky. Security design patterns part 1. http://citeseer.ist.psu.edu/575199.html, November, 2001
7. 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.
8. 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.

9. 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.
10. Commission of European Communities. Information technology security evaluation criteria, version 1.2, 1991.
11. J. A. Zachman. A framework for information systems architecture. IBM Systems Journal, 26(3), 1987.
12. 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.
13. J. Viega and G. McGraw. Building secure software - How to avoid security problems the right way. Addison-Wesley, 2001.
14. M. Hafiz. Security Architecture of Mail Transfer Agents. Master's Thesis. University of Illinois at Urbana-Champaign, July 2005.
15. D. Trowbridge, W. Cunningham, M. Evans, L. Brader, and P. Slater. Describing the enterprise architectural space. MSDN, June 2004.
16. IEEE Std 1471-2000. IEEE recommended practice for architectural description of software intensive systems. http://standards.ieee.org/reading/ieee/std_public/description/se/1471-2000desc.html, 2000.
17. F. Swiderski and W. Snyder. Threat modeling. Microsoft Press, 14 Jul 2004.
18. C. Alexander, S. Ishakawa, and M. Silverstein. A pattern language. Oxford University Press, New York, 1977.

Author Biography

Munawar Hafiz is a graduate student at the University of Illinois at Urbana-Champaign. His research interests are software architecture and design, software patterns and security. Contact him at Siebel Center, 201 N Goodwin Avenue, University of Illinois at Urbana-Champaign, Urbana, IL 61801, USA; mhafiz@uiuc.edu.

Paul Adamczyk is a graduate student at the University of Illinois at Urbana-Champaign. His research interests are software architecture and design, Web technologies, and software patterns. Contact him at Siebel Center, 201 N Goodwin Avenue, University of Illinois at Urbana- Champaign, Urbana, IL 61801, USA; padamczy@uiuc.edu.

Ralph Johnson is a research associate professor at the University of Illinois. His research interests include object-oriented design, especially design patterns and the design of object-oriented frameworks. He is a coauthor with Erich Gamma, Richard Helm, and John Vlissides of "Design Patterns: Elements of Reusable Object-Oriented Programming," and was one of the organizers of the original conference on Pattern Languages of Programs.
Johnson received a BA from Knox College and a PhD from Cornell. He is a member of the IEEE Computer Society and ACM. johnson@cs.uiuc.edu.