

Chapter 2

Towards a Conceptual Framework for Security Patterns

Clive Blackwell

Abstract We introduce security patterns as the most mature domain within cyberpatterns, and outline a conceptual framework to help understand and develop good security patterns. Security patterns help us move from an improvised craft to engineering discipline because they transfer knowledge about proven solutions in an understandable and reusable format to experienced users and novices alike. Although security patterns are widely known, many questions remain unanswered regarding their conceptual foundation and practical use. We characterise the current pattern schemes using the Zachman Framework for enterprise architecture modelling, which allows us to structure and pose questions about both the problem domain and corresponding solutions provided by security patterns. We propose a parallel security plane overlaying the entire Zachman grid allowing the separate consideration of security within the security plane using the interrogative questions (who, what, where, when, why and how) to evaluate the six aspects. The integration between security and functional concerns is similarly aided by using the correspondence between aspects in the security and functional planes to decompose and examine the relationship between security patterns and problem context. We also briefly discuss security patterns as transformations, and related concepts such as tactics that may usefully be applied to security. We conclude with a set of unsolved challenges for security patterns. This discussion is relevant to other types of cyberpattern such as attack patterns, and may aid the eventual development of a comprehensive framework for cyberpatterns.

C. Blackwell (✉)

Department of Computing and Communication Technologies, Oxford Brookes University,
Wheatley Campus, Wheatley, Oxford OX33 1HX, UK
e-mail: CBlackwell@brookes.ac.uk

1 Introduction and Rationale

Designing a secure system is a hard endeavour requiring unique skills that we cannot expect from typical development teams. It would be helpful if there were a set of easily usable building blocks that provide a foundation for secure systems, without users needing to grasp the finer points of security engineering. Security patterns aim to provide these blocks by providing realisable solutions to help solve known recurrent security problems.

A pattern can be characterised as a solution to a problem that arises within a specific context. Each pattern expresses a relation between a problem context, a set of unwanted forces that occurs repeatedly in that context, and a system configuration that allows these forces to be resolved [1]. Similarly, a security pattern encapsulates security expertise in the form of vetted solutions to recurring problems [2], as in the ‘Gang of Four’ (GoF) Design Patterns book [3]. A security pattern describes a solution to the problem of controlling specific threats through the application of some security mechanisms in a given context [4].

Security patterns organise and preserve this knowledge in an understandable and reusable format, so that less proficient developers may benefit. This may help us move from a craft to engineering discipline, because patterns transfer knowledge and understanding about proven solutions to the community and thereby help to establish security on a sounder footing.

2 Pattern Classification

2.1 Zachman Framework

We propose the use of the Zachman Framework [5] to model security patterns and their various classification schemes. The Zachman Framework [6] is a widely used and influential framework for enterprise system architecture, and is displayed as a matrix or table providing different system scopes and levels of abstraction in the horizontal rows and different system aspects in the vertical columns, as shown in Table 1.

The framework comprehensively models all horizontal scopes and levels of abstraction from global in the top row down to the operational or data scope. When used to develop enterprise systems, the development lifecycle starts with high-level analysis of the system goals and context in the top row progressing down through the rows until very detailed concerns are addressed in the operational system. The columns allow comprehensive analysis of different system aspects using particular methods or tools to answer the question relevant to the particular concern. Each cell analyses the syl suitable for the concerns of the column aspect with an appropriate level of detail for the row.

Table 1 Zachman framework for security patterns (some row names altered)

Scope	Aspect Question	Data What?	Function How?	Network Where?	People Who?	Time When?	Motivation Why?
Global							
Enterprise							
Architecture							
Design							
Implementation							
Operation or data							

2.2 *Modelling Security Patterns*

We use the Zachman Framework somewhat differently for modelling security patterns and so the names of some of the rows have been changed to match their new use. A key point is that the implementation of an enterprise system employs all the framework rows with clear relationships between them, where analysis in a lower row breaks down the model for the next higher row into more detail.

A security pattern lives at one or possibly two rows only, but does relate to a system problem also represented in the same row of the Zachman Framework. The framework has been used to model attack patterns [7] by mapping various attack pattern sections to the six Zachman aspects. The same idea for security patterns helps integration of security patterns into the systems they protect by matching the six aspects between the pattern solution and system problem. The connection between patterns of different horizontal scopes is not examined further, but we suggest that one relationship is pattern aggregation where a broader pattern such as an architectural pattern encapsulates narrower patterns such as design patterns.

The crucial link is between security and system concerns in the third dimension. We are inspired by the SABSA (Sherwood Applied Business Security Architecture) model [8] of an entire plane devoted to security replicating all the elements of the Zachman Framework from a security perspective. The separate security plane is displayed in the third dimension parallel to the original functional Zachman matrix. The different aspects help to answer the questions who, what, why, where, when and how about both the system problem and security pattern solution. The Zachman aspects of the security pattern in the security plane must constrain the undesirable problem behaviour by resolving the security forces, and match the aspects of the problem context in the functional plane for compatibility.

The security plane promotes both separation and integration of security and the other system concerns. The parallel plane allows the separate consideration of security by decomposing security analysis into the six column aspects, as in the SABSA model [8]. This allows comprehensive investigation of security aspects within the plane, and thereby helps security pattern specialisation and composition to address pure security concerns.

Table 2 Zachman framework for security patterns (development lifecycle perspective)

Level	Aspect Question	Data What?	Function How?	Network Where?	People Who?	Time When?	Motivation Why?
Analysis							
Requirements							
Architecture							
Design							
Implementation							
Deployment							

The matching between the functional and security planes on the other hand allows the integration of security with the system concerns. The relationships between the planes enable integration, because the security aspect in a cell in the security plane must be compatible with the functional aspect in the corresponding cell in the functional domain. In particular, the security patterns and the system context must be at the same level of abstraction and have compatible horizontal scope.

We can model security patterns at all six different scopes from the wider global scope right down to the very narrow operation or data scope, extending beyond their existing use spanning the enterprise down to implementation rows. Then, the six scopes of security pattern (with example security issues in brackets) are in descending order: Global (social engineering), Enterprise (insider threat), Architecture (man-in-the-middle), Design (SQL injection), Implementation (buffer overflow), and Operation or Data (malware operation detected by signature).

The Zachman grid does not distinguish the horizontal scope of analysis in each row from the stage in the system development lifecycle, as they usually match when building enterprise IT systems. The broader abstract system concerns are considered first before moving down to more and more detailed and narrow functionality eventually finishing with the deployed system in the bottom row.

However, the horizontal scope and lifecycle stages need to be considered separately when the framework is used for patterns, as the lifecycle stage does not necessarily bear a close connection to the pattern scope. For example, an early-lifecycle analysis pattern could apply narrowly to a small component, whereas a late-lifecycle deployment pattern could suggest a broad system solution involving architectural restructuring. Therefore, we can also consider the levels of abstraction in pattern development using a staged lifecycle process and map its various phases to the Zachman hierarchy as shown in Table 2. The relationship between horizontal scope and level of abstraction therefore needs more thought for security patterns.

Fernandez et al. proposed a methodology that incorporates security patterns in all the lifecycle stages [9], which we relate to the Zachman Framework levels. A development methodology for security patterns is required, as many security pattern catalogues do not provide clear information about when or where to apply them. This methodology proposes guidelines for incorporating security patterns within the requirements, analysis, design, implementation, testing and deployment phases of

development. Our methodology contains the analysis, requirements, architecture, design, implementation and deployment phases, as shown in the rows of Table 2.

This suggests more research is needed for security patterns outside the design and architectural stages where most work has occurred so far. For example, the answers to the aspect questions depend crucially on the lifecycle stage and level of abstraction. In addition, the processes involved in each stage are also important, and common techniques may be abstracted and modelled as behavioural patterns.

3 History and Classification of Security Patterns

3.1 Basic Pattern Catalogues

This is not intended to be a comprehensive survey of security patterns, so the cited sources should be consulted as several have extensive bibliographies. We describe the simpler pattern arrangements that are not full classification schemes in this section summarised in Table 3, before discussing multifaceted schemes in the next section. Yoder and Barcalow [10] wrote the first paper on security patterns in 1997, introducing seven patterns using the GoF (Gang of Four) template and structuring them as a pattern language.

The pattern community developed a large number of security patterns over a number of years for workshops at the Pattern Languages of Programs (PLOP) conferences. Subsequently, a working group was established under Markus Schumacher with the results published as a book [4] in 2005 that was the first to categorise most known security patterns. Their systematic collection of 46 security patterns covered multiple scopes and levels of abstraction, including several enterprise patterns for risk assessment and mitigation, as well as the usual architectural and design patterns. Graham [11] studied more general business patterns creating a pattern language containing 42 patterns with several that could be specialised as procedural security patterns.

Finally, Munawar Hafiz worked with Ward Cunningham and the Microsoft Patterns and Practices group on a comprehensive catalogue of all known security patterns following the path started at the PLOP workshops and continued by Schumacher's book. The current catalogue currently contains nearly 100 security patterns by culling duplicate, poor quality and out-of-scope patterns, which were then classified and organised into a pattern language [12].

There are several other security pattern repositories, collections and surveys. The Security Patterns Repository Version 1.0 (the only version) [2] is a book-length technical report published in 2002 consisting of a comprehensive description of 29 patterns. They considered the patterns under two broad classes of structural and procedural patterns, with 16 structural patterns for implementing secure applications and 13 procedural patterns for the development process.

The procedural patterns are not the same as the behavioural patterns in the GoF book, as they have a broader scope involving people performing activities such as

Table 3 Simple security pattern schemes

Authors	Purpose	Classification criteria	Scope
Kienzle et al. [2]	General pattern repository attempting to document known patterns and improve pattern quality	Two classes of structural and procedural (manual security processes)	Enterprise (procedural patterns), architectural and design (mainly structural patterns)
Open Group [14]	Aiding security and availability (against unintentional causes not just deliberate denial of service)	Two categories of security and availability (availability is distinguished from the other security objectives)	Architectural and design. Availability patterns have the wider goal of aiding dependability
Steel et al. [17]	To aid implementation of Web services on Java platforms	By architectural tier (web, business, web service and identity tiers) in the system domain	Architectural, design and implementation, but restricted to Web services only
Microsoft's Patterns and Practices group [18]	To aid implementation of Web services using Microsoft technologies	By security purposes (eg. authentication) and indirectly by protection target (eg. resource in resource access)	Architectural, design and implementation, but restricted to Web services only
Yoshioka et al. [15]	A survey paper establishing the current state of the art and performing gap analysis to identify future research directions	By system development phases	Spanning all scopes from enterprise to implementation as befits a survey
Dougherty et al. [16]	To protect existing design patterns by decoration with security aspects	Secure design patterns are specialised from existing architectural, design and implementation patterns	Architectural, design and implementation

documentation, testing and patching, similar to Graham's business patterns. Hafiz [13] discovered very few procedural patterns in his comprehensive classification, so this might be an area for future pattern discovery, both in security and possibly in the emerging domain of digital forensics.

The Open Group Guide to Security Patterns book [14] in 2004 contains architectural and design patterns following the GoF template [3]. The catalogue classifies 13 patterns into two broad groups based on the protection objective with eight patterns for security and five for availability. The security patterns aim to protect valuable assets against unauthorised use, disclosure, or modification, whereas the five availability patterns aim to provide predictable and uninterrupted access to services

and resources against both deliberate and unintentional causes, rather than just for the narrow security objective of defeating intentional denial of service attacks.

Yoshioka et al. [15] produced a comprehensive survey that investigated the various approaches to security patterns and proposed future research directions. They characterised security patterns according to the software development phases and analysed the adequacy of patterns by their conformance to established security concepts. They identified the need to ground analysis in actual risks by keeping attacks in mind when developing security patterns. However, it is difficult to find specific security threats before implementation, and therefore, we need to establish a way of discussing attacks abstractly at the design stage to help us develop corresponding security patterns.

The Software Engineering Institute guide to Secure Design Patterns [16] in 2009 describes various secure design patterns addressing security issues in the architectural, design and implementation phases of the development lifecycle. Secure design patterns are distinguished from security patterns, as they do not describe specific security mechanisms (eg. access control, authentication, and authorisation), but simply adapt and extend existing functional patterns.

Finally, two books take a more practical technological viewpoint. Steel et al. [17] developed an extensive collection of security patterns in 2005, specifically for developing Web services on Java platforms at quite a detailed level of granularity ready for implementation. They classified 23 security patterns according to logical architectural tiers; namely, web, business, web service and identity tiers. These are logical functions in the system decomposed into separate components within the Web services domain.

Microsoft's Patterns and Practices group published a security patterns book [18] for Web services in 2006 with 18 patterns focussing on implementation with Microsoft technologies. The patterns are classified mainly by security purpose with classes for Authentication, Message protection, Transport and message layer security, Resource access, Service boundary protection and Service deployment. These categories mention both security objectives and protection targets, unlike the Steel book that classifies patterns according to system functionality alone (Table 4).

3.2 Multi-Dimensional Classification Schemes

Pattern classification schemes are needed for organisation, classification, comparison and selection of patterns. We proceed to discuss the more complex classification schemes using multiple criteria and indicate their place within the Zachman framework.

MITRE [19] introduced the use of the Zachman Framework in 2002 to help characterise and classify security patterns. They invented a seventh column for the framework to represent the Information Assurance perspective (which we consider synonymous with security here). The Information Assurance column enables the separate self-contained consideration of security concerns in every row of the framework.

Table 4 Security pattern classification schemes using the Zachman framework

Authors	Purpose	Classification criteria	Scope
MITRE [19]	Captures security best practices as reusable knowledge at all scopes of the enterprise from broad concerns down to implementation	Firstly by row in the Zachman table and secondly by security objective	Spanning all scopes from enterprise to implementation
Microsoft [20]	Aiding the development of products using Microsoft technologies	Classification by rows and then by subdivision into roles within each row, and finally by their Zachman aspects in the vertical dimension	General functional patterns at the bottom five rows of the Zachman framework
Schumacher [4]	Catalogued and documented known good quality patterns to support a firmer theoretical foundation for security patterns and aid their use by developers	Mainly by security objective, but other aspects such as resource being protected intrude into the classification. They place these groups of patterns within a diagram of horizontal sections resembling the Zachman rows, and different vertical columns that partition each row according to specific criteria appropriate for that level or scope	Enterprise, architectural and design, but not implementation
Hafiz et al. [13]	To consolidate and organise security patterns using a variety of classification criteria	By application context referring to where the pattern is applied relative to the system (core, perimeter, exterior), by Zachman aspect, by STRIDE category of security threat, and by fundamental security objective (confidentiality, integrity and availability)	Architectural, design and implementation
Hafiz [12]	Catalogued and documented all known good quality patterns and formed them into a pattern language	Final classification is by application context (core, perimeter, exterior) and then by STRIDE threat category	Architectural, design and implementation
Fernandez ^a [24]	Developed a methodology for building secure systems using patterns, and catalogued and documented good quality patterns including several new ones	Main criteria for patterns are firstly by security concern (security objective or application domain) and secondly by computer layer where the pattern is deployed. Several other classification criteria were also discussed	Architectural and design

^aDoes not explicitly mention the Zachman Framework

Secondly, the elements of the Information Assurance perspective are considered to form a plane overlaying the entire Zachman framework. This allows the integration of security and functional concerns as well, as the security concerns can be linked to the functional concerns in each of the other six columns. Our framework extends this with an explicit security plane overlaying the entire Zachman table cell-for-cell to model each of the six security aspects within their own cells, thus comprehensively decomposing and clarifying the relationships between the functional and security concerns.

They further subdivided the Information Assurance concerns into the classes of Identification and Authentication (Authentication), Access Control, Integrity, Denial of Service (Availability) and IA Accounting (Accountability) to aid detailed analysis. Confidentiality is a crucial omission, and Denial of Service is a threat not a security service (with the corresponding Availability service and the more common names for the other services given in brackets).

The Microsoft Patterns and Practices team [20] in 2004 adapted and extended the Zachman Framework to describe 110 patterns classified as Enterprise (64 patterns), Data (16 patterns) or Integration Patterns (30 patterns) for use in some of their products. The patterns are for modelling functional rather than security concerns, but the classification is similar to several of the security pattern schemes presented in this section.

They also have five levels that are roughly comparable to the bottom five rows of the Zachman Framework, which are (from the top): Business, Integration, Application, Operational and Development Architectures (the last two levels are ordered the opposite way in the Zachman Framework). The top row of the Zachman Framework is purely conceptual covering organisational objectives and the external environment and so has no architecture.

Their classification system goes a step further by dividing the rows according to specific roles giving particular use cases for each role, as originally described in the IEEE Standard 1471 for software architecture [21]. For example, the four roles given at the Business Level are CEO, General Manager, Process Owner and Process Worker who all clearly need different use cases to model the specific activities required to achieve their differing objectives, which can be analysed in detail by answering the questions associated with the various aspects.

It also includes a seventh column for testing called Scorecard that indicates test methods and success criteria appropriate to the purpose of the use cases and their level of abstraction. This is similar to MITRE [19], but Microsoft's conception has a narrower focus as the patterns are developed for implementation rather than conceptual analysis. They explicitly trace the tests to the purpose column for validation against desired objectives, rather than having relationships between the seventh column and all the other cells in the same row. Furthermore, they used the seven columns along with the roles in each row to classify each security pattern within a particular cell. However, this does not appear to be very useful, as most of the patterns were classified by function (how) with some by data (what) and a few by network (where) with no patterns classified within the remaining four columns.

Schumacher et al. [4] in 2005 structured their security pattern classification by overlaying collections of related patterns onto Zachman's levels of information models from the enterprise to design levels. They faithfully reproduced MITRE's idea of a seventh column, but changed its name to Security and gave it the question 'what security' or 'what protection'. Their security model has four explicit levels mapping to the upper five rows in the Zachman hierarchy (shown in brackets): Security strategy and policy (rows 1 and 2), Services (row 3), Mechanisms and Implementations (row 4), and an unnamed row for products and tools not containing any patterns (row 5).

They categorised the patterns mostly by security objective, although some other aspects such as the resource being protected (eg. a firewall) are sometimes incorporated as well. The classes are Enterprise security and risk management, Identification and authentication, Access control, Accounting, Firewall architecture, and secure Internet applications. The enterprise security and risk management patterns are at the enterprise scope, whereas most of the others are at both of the Services level (mapping to Zachman's architectural row) and Mechanisms and Implementations level (mapping to Zachman's design row).

They also partitioned each level by different criteria according to the theme for the level, and then placed the pattern classes (mentioned above) within these divisions. For example, the Mechanisms and Implementation level (design row) was partitioned according to the mechanisms supported, with the patterns classified into management support, automated, physical and procedural mechanisms. However, they discovered that all their patterns at this level were automated mechanisms, so there is ample scope for pattern discovery within the other categories.

Hafiz et al. [13] proposed several ways to classify and organise security patterns by:

- The application context referring to where the pattern is applied relative to the system
- Zachman aspect like Microsoft's general pattern scheme [21]
- The STRIDE threat model [22] devised by Microsoft
- Security objectives of confidentiality, integrity and availability.

The application context classifies the security patterns by their location within the system core, on the system perimeter or exterior in the system environment. It is the horizontal location where the pattern is applied, and is not to be confused with the vertical application level at layer 7 in the OSI network model. One issue is that most patterns fall into the system core category and the scheme does not provide a way to separate them into smaller unified groups.

The Zachman aspects were proposed for classifying security patterns based on the ideas put forward by Microsoft discussed earlier [20]. Nearly all security patterns are classified by the functional and data aspects with a few by the network aspect, which is not very useful for classification as we already mentioned. In addition, the Zachman table could not uniquely classify many patterns whose classification occupied multiple cells vertically or horizontally. Others employing the Zachman Framework used the column aspects to characterise different facets of patterns rather than for classification.

They included a seventh column for testing correctness or verification, not validation as proposed by Microsoft [20] or the wider sense of considering the different security concerns for all of the aspects as proposed by MITRE [19]. The testing column also houses separate testing patterns consistent with their classification scheme by aspect, whereas most others only used the seventh column for posing questions about the adequacy of patterns described in the other columns.

Furthermore, they classified patterns based on the problems they try to overcome in the security domain using Microsoft's threat modelling scheme. STRIDE [22] is an acronym that categorises the different types of threats to systems: Spoofing, Tampering, Repudiation, Information disclosure, Denial of service and Elevation of privilege. Once again, many patterns fall into several classes.

The final classification method is by the fundamental security objectives of confidentiality, integrity and availability. This is not a good classification scheme either, as there are many other security requirements such as authentication, authorisation and accountability, and several patterns have overlapping objectives even within even this restricted set, so it does not classify security patterns either completely or uniquely.

They finally settled on a hierarchical classification criteria using the two concepts of location first and then STRIDE security issue, discarding the Zachman aspects and security objective classifiers from their earlier work because they were not discriminating enough. The two dimensional schema overcomes to an extent the inadequate and incomplete classification provided by the single classifiers.

They then used the classification criteria to build a large security pattern language cataloguing all known patterns [23], and succeeded in classifying around 80 % of the around 100 unique patterns. The rest were generic patterns that they called higher-level patterns that could not be classified uniquely according to the two criteria. They weeded out poor quality and duplicate patterns, grouped the patterns using the criteria to help classification and selection, and identified and documented the relationships between the patterns in a pattern language displayed visually in pattern diagrams.

Fernandez [24] extends the ideas developed in the previous book with Schumacher et al. [4] adding many new patterns, and developing a methodology for utilising security patterns in building secure systems. Fernandez's schema does not mention the Zachman Framework, but is still able to be classified within its organisational structure*.

He discussed patterns at different levels (not scopes), thinking of a computer system as a hierarchy of layers (as in the 7-layer OSI network model), including the metalayer (people), application layer, system layer (operating system and database), distribution layer and hardware configuration [25]. These are the levels where patterns are deployed in the system domain, not the levels of abstraction in problem detail described in the rows of the Zachman Framework.

They then refined the basic classification using a multi-dimensional matrix of categories [26]. One dimension corresponds to lifecycle stages, including domain analysis, requirements, problem analysis, design, implementation, integration, deployment, operation, maintenance and disposal. They gave a mapping of patterns to the different computer levels along with the different lifecycle phases as the two main classification criteria. This separates out the Zachman horizontal row into two

separate concepts divorcing level from lifecycle stage, as the levels considered by Fernandez are in the system domain where patterns are deployed, not levels of abstraction in problem detail as in the Zachman Framework that is correlated with the lifecycle stage.

Fernandez also indicates a dimension of security concern [24] that typically refers to security objective (eg. access control) in the security domain or application area in the system domain (eg. cloud computing). The security concerns are Identity Management, Authentication, Access control, Process Management, Execution and File Management, Operating System Architecture and Administration, Networks, Web Services, Cryptography, Middleware, and Cloud Computing that are the topics of several of the chapters of his book. Fernandez finally settles on a mapping of patterns that are firstly grouped by security concern, and within that, different computer levels as the two main classification criteria.

4 More Pattern Types and Related Concepts

4.1 *Security Tactics and Specialisation*

The less familiar idea of tactics [27] playing the role of design primitives may complement the broader pattern construct. A tactic is a design decision that influences the achievement of a quality attribute (aka non-functional requirement). The focus is on the attribute with little consideration of the wider concerns seen in patterns, such as problem context or trade-off analysis. Tactics are simpler and therefore, it is argued, easier to devise and apply.

The tactics idea has been applied to security, which we consider a quality for the purposes of this paper. There are four classes of security tactics: Detect, Resist, React and Recover [28, Chap.9]. For example, within the Detect Attack class, we have Detect Intrusion, Detect Service Denial, Verify Message Integrity and Detect Message Delay. Security tactics are quite restricted and therefore ought to be more easily characterised and analysed than security patterns. We suggest that security tactics can adapt or compose with existing functional or security patterns to address particular security concerns.

Secure design patterns [16] were created by adding security services to several of the design patterns in the Gang of Four (GoF) book [3]. Secure design patterns are distinguished from security patterns, as they do not describe separate security mechanisms, but simply adapt and extend existing patterns (possibly using security tactics). This is a specialisation of an existing pattern to create a secure design pattern decorated with security concerns, whereas a standalone security pattern has to be composed with a design pattern to provide protection.

4.2 Transformational Patterns

Refactoring is the process of altering the internal structure of existing code to improve its quality without changing its externally observable behaviour [28]. Refactoring is generally simpler than a pattern as it makes more tactical rather than systemic improvements to code. For example, each refactoring in Fowler's catalogue [28] has a name, motivation, mechanics and examples rather than a dozen or more sections typical in a pattern template.

Lano [29] proposed a novel representation of design patterns as transformation rules that refactor a system to improve its quality. A refactoring is a transformation pattern if it describes a recurrent problem and its solution along with consequences [29], so it is more generic and has wider concerns than basic refactoring. When applied, the pattern transforms a given situation in which a particular problem is present into another situation where it is resolved or attenuated. To clarify, the transformation is the pattern and the source and target of the transformation may not be patterns.

The concept of transformational patterns seems to apply equally well to security as other quality or non-functional concerns. For example, Fowler's catalogue [28] contains several refactorings that can remove the security flaws from existing object-oriented programs. Hafiz considered his catalogue of patterns as security-oriented program transformations [30]. These transformations may ease the task of retrofitting security to existing systems, which is a crucial benefit because many deployed systems cannot be easily replaced.

5 Pattern Organisation

5.1 Pattern Templates

Patterns are usually described using a template with predefined sections to document the different aspects. The template design is important, because it defines the type and characteristics of the information collected about the pattern and the problems it attempts to resolve. The level of detail in patterns templates differs considerably; although some templates provide very detailed explanations, others are quite concise. There is a strong correlation between poorly documented and inferior quality patterns. Short informal textual patterns may be useful for experienced designers to remind them of the different issues and possibilities, but they are not detailed or precise enough to aid implementation or acquire understanding by inexperienced users.

The GoF pattern template [3] is the de facto standard template for describing design patterns, and the POSA (Pattern-Oriented Software Architecture) format [31] is common for architectural patterns. However, there is no agreed standard format for describing security patterns and little consistency in the level of detail prescribed

within the various templates. Patterns using ad hoc templates make the job of selecting and applying patterns much harder.

Some security patterns are structured using the GoF [3] or POSA [32] formats without significant adaptation or extension to incorporate security aspects. This is a sensible starting point because many security patterns aim to solve particular design and architectural problems. However, these general formats may not be specific enough to capture particular security concerns making it difficult to design, select and implement suitable security patterns that resolve the particular security issues satisfactorily. We may need to adapt or extend the sections in a standard template for each level of abstraction to address the distinctive security concerns in each stage of the development lifecycle.

5.2 *Pattern Spaces and Diagrams*

Patterns resolve problems by shaping their environments by configuring elements that exhibit suitable behaviour to resolve the forces displayed by the problem. However, implementing a pattern in a specific way creates a concrete situation whose context narrows the potential solution space for resolving problems remaining within the system context or brought about by the design solution.

Security patterns are usually secondary to objective concerns, which is awkward as the system environment or problem context may make it difficult to apply adequate security because of prior system commitments. Therefore, a structured way of thinking about the functional and security problems together using patterns is helpful to connect the security requirements to the system objectives they serve.

The patterns community structures and organises pattern spaces with pattern catalogues [3], pattern systems [31] and pattern languages [32]. One objective is to elaborate how security patterns can be combined into meaningful structures with other types of pattern.

We need to extend pattern combination to multiple spaces with security patterns composing with or specialising functional patterns. This requires understanding how the languages in the two domains can interoperate, with the functional pattern describing purposeful activities and the security pattern constraining potential undesirable behaviour allowed by the system context or functional pattern.

We can describe the relationships between patterns using a pattern diagram in which rounded rectangles represent patterns and arrows indicate the contribution of a pattern to another [31]. The basic pattern diagram can be extended with the generalisation notation from UML class diagrams [24] to indicate patterns that are subpatterns or specialisations of other patterns. As already mentioned, a Carnegie Mellon group invented secure design patterns [16] by simply specialising existing design patterns into secure design patterns by adapting and extending the basic pattern with security features. We can extend pattern diagrams in the UML notation to show the specialisation of design patterns into secure design patterns across pattern spaces.

The typical view of a security pattern is as a separate concept from design patterns, so patterns from both classes may need to be composed together to resolve problems. The security forces present in the problem domain are resolved by the security pattern in the security domain. We invent some new relationships between patterns with ‘protects’ between security and design patterns, and ‘defeats’ for the oppositional relationship between security and attack patterns. Hafiz [12] used pattern diagrams to express the associations between all known security patterns in his classification system, which we can extend to display these new relationships across pattern spaces as indicated by Zhu [33].

6 Conclusions and Further Work

The Zachman Framework is a metamodel for developing enterprise architectures as it incorporates the concepts needed for representing system models at all levels of investigation. We have demonstrated that it can similarly be seen as a metamodel for understanding and developing security patterns as it contains the concepts needed to classify all existing schemes and indicate their limitations. Our framework extends the basic Zachman framework with a parallel security plane overlaying the entire Zachman grid cell-for-cell to model each security aspect within its own column, allowing analysis of its compatibility with the problem context in the corresponding columns in the functional plane.

We highlight some of the key benefits of the Zachman Framework for modelling security patterns before elucidating some of the challenges. The advantages include:

- Separation of security from functional concerns using the separate security plane allowing for their independent analysis, as in the SABSA model [8]
- Integration of security and functional concerns by analysing the relationships between the corresponding aspects in the security and functional planes
- Extension to different horizontal scopes using the top row to model global patterns on the Internet, and the bottom row for modelling data patterns
- A meta-classification structure using the three dimensions of the framework (rows, aspects, planes) supporting examination of existing multidimensional schemes
- Aiding pattern selection as the solution pattern aspects in the security plane must be compatible with the corresponding functional aspects of the problem context.

The challenges regarding both the conceptual foundation and practical use of security patterns include:

- Clarifying their description to help both experienced and novice developers
- Determining the underlying concepts for their proper expression in a suitable ontology, possibly aided by the existing ontology for the Zachman Framework [34]
- Characterising their interrelationships in an adequate and complete pattern language

- Producing agreed classification criteria, unlike current pattern catalogues that use a variety of different attributes without giving a persuasive rationale
- Better coverage of the problem space identified from categorising existing patterns within the various rows, aspects and planes of the framework
- Providing a precise formalism for verifying the protection offered by security patterns against various attacks
- Developing tool support to enable pattern selection and formal reasoning
- Exploring the inconsistency between the horizontal scope of the problem compared with the lifecycle stage when the pattern is developed and its level of abstraction
- Establishing a systematic methodology with guidelines for incorporating security patterns within the various development stages to aid their differing objectives
- Developing patterns within fresh domains and applications such as data patterns
- Abstracting and modelling the processes and techniques involved in each lifecycle stage to extract common procedures into behavioural patterns
- Needing to distinguish pattern development from use in operational systems
- Needing to understand the relationships to descriptive patterns like attack patterns that are observed and inferred from their operational activities
- Analysing the relationships to other pattern types such as attack patterns by comparing their corresponding aspects using additional planes for each type
- Determining the utility of security patterns as transformations of insecure system configurations or observed attack activities into safe situations
- Possible development of related techniques such as security tactics for situations where patterns are unsatisfactory.

Relationships between patterns in different conceptual spaces can be advantageously considered [33] in addition to the existing relationships between security patterns such as generalisation, aggregation and conflict employed in security pattern languages [12]. The current use of specialisation in decorating design patterns with security concerns [16] and composition of patterns with tactics [27] can both be extended with a ‘protects’ relationship between complete security and design patterns.

The patterns in the various spaces must be compatible for the relationships between them to have practical utility. The integration and separation of security and functional concerns by using separate planes can be extended to other types of cyberpattern. We can again use aspects for analysis of the ‘protects’ relationship to match the characteristics of the design pattern in the functional plane and security pattern in the security plane, analogous to their use to overcome system problems with compatible security pattern solutions discussed earlier.

In addition, attack and security patterns can be modelled in oppositional planes connected by the ‘defeats’ relationship. The Zachman Framework has already been used for the decomposition of attack patterns [7] by their various aspects, and it is possible to use aspects to establish the likely success of attack instances in the prevailing system context. The idea now is to make the attack pattern incompatible with the system context using a security pattern that ‘defeats’ the attack pattern by preventing or overcoming the attack behaviour (how question) or another

crucial aspect such as its motivation (why). This allows consideration of a broader set of protection mechanisms such as deterrence rather than focussing exclusively on malicious actions.

In addition, the patterns in the various spaces must also be compatible as pattern constructs for the relationships between them to have meaning. For example, pattern types match when they have the same operations, which may occur for base class operations inherited from a cyberpattern superclass. There has been some formal work to show compatibility between design and security patterns [35] and an indication that the main classes of cyberpattern including attack patterns may be compatible with each other [36]. A uniform framework for cyberpatterns is a significant and credible objective integrating formalisation with a strong conceptual basis along the lines discussed here.

References

1. Gabriel RP. Patterns of software: tales from the software community. New York: Oxford University Press; 1996. <http://www.dreamsongs.com/Files/PatternsOfSoftware.pdf>. Accessed 10 Nov 2013.
2. Kienzle DM, Elder MC, Tyree D, Edwards-Hewitt J. Security patterns repository version 1.0. Washington DC: DARPA; 2002. www.scrypt.net/~celer/securitypatterns/repository.pdf. Accessed 7 Nov 2013.
3. Gamma E, Helm R, Johnson R, Vlissides J. Design patterns: elements of reusable object-oriented software. Reading: Addison-Wesley; 1995.
4. Schumacher M, Fernandez-Buglioni E, Hybertson D, Buschmann F, Sommerlad P. Security patterns: integrating security and systems engineering. Chichester: John Wiley; 2005.
5. Zachman JA. A framework for information systems architecture. IBM Syst J. 1987;26(3): 276–92.
6. Zachman JA. John Zachman's concise definition of The Zachman frameworkTM. Zachman International, Inc. 2008. <http://www.zachman.com/about-the-zachman-framework>. Accessed 28 Oct 2013.
7. Blackwell C. A strategy for formalising attack patterns. 1st cyberpatterns workshop. 2012. In cyberpatterns: unifying design patterns with security and attack patterns. Springer; 2014.
8. Sherwood J, Clark A, Lynas D. Enterprise security architecture, a business driven approach. San Francisco: CMP Books; 2005.
9. Fernandez EB, Larrondo-Petrie MM, Sorgente T, VanHilst M. A methodology to develop secure systems using patterns. Chapter V: Integrating security and software engineering: advances and future vision. IGI; 2006. p. 107–26.
10. Yoder J, Barcalow J. Architectural patterns for enabling application security. In: Proceedings of the 4th annual conference on pattern languages of programs (PLoP 1997). 1997.
11. Graham I. Business rules management and service oriented architecture: a pattern language. Chichester (WSX): John Wiley; 2007.
12. Hafiz M. Security pattern catalog. www.munawarhafiz.com/securitypatternscatalog. Accessed 30 Oct 2013 (15 March 2013).
13. Hafiz M, Adamczyk P, Johnson RE. Organising security patterns. IEEE Softw. 2007;24(4): 52–60.
14. Blakely B, Heath C. Security design patterns. Berkshire, UK: The Open Group; 2004.
15. Yoshioka N, Washizaki H, Maruyama K. A survey on security patterns. Prog Inf. 2008;5(5): 35–47.

16. Dougherty CR, Sayre K, Seacord R, Svoboda D, Togashi K. Secure design patterns. Software engineering institute. Paper 47. 2009. <http://repository.cmu.edu/sei/47>. Accessed 8 Nov 2013.
17. Steel C, Nagappan R, Lai R. Core security patterns: best practices and strategies for J2EE, web services, and identity management. Englewood Cliffs: Prentice Hall; 2005.
18. Hogg J, Smith D, Chong F, Taylor D, Wall L, Slater P. Web service security: scenarios, patterns, and implementation guidance for web services enhancements (WSE) 3.0. Redmond (WA): Microsoft Press; 2006.
19. Heaney J, Hybertson D, Reedy A, Chapin S, Bollinger T, Williams D, Kirwan Jr. M. Information assurance for enterprise engineering. In: Proceedings of the 8th annual conference on pattern languages of programs (PLoP'02). 2002.
20. Trowbridge D, Cunningham W, Evans M, Brader L, Slater P. Describing the enterprise architectural space. MSDN. June 2004. msdn.microsoft.com/en-us/library/ff648192.aspx. Accessed 8 Nov 2013.
21. Maier MW, Emery D, Hilliard R. Software architecture: introducing IEEE standard 1471. IEEE Comput. 2001;34(4):107–9.
22. Swiderski F, Snyder W. Threat modelling. Redmond (WA): Microsoft Press; 2004.
23. Hafiz M, Adamczyk P, Johnson RE. Growing a pattern language (for security). In: Proceedings of the 27th object-oriented programming, systems, languages and applications (OOPSLA 2012). 2012.
24. Fernandez-Buglioni E. Security patterns in practice: designing secure architectures using software patterns (Wiley software patterns series). New York: John Wiley; 2013.
25. Fernandez EB, Pan R. A pattern language for security models. In: Proceedings of the 8th annual conference on pattern languages of programs (PLoP 2001). 2001.
26. VanHilst M, Fernandez EB, Braz F. A multidimensional classification for users of security patterns. J Res Pract Inf Tech. 2009;41(2):87–97.
27. Bass L, Clements P, Kazman R. Software architecture in practice. Boston: Addison-Wesley; 2012.
28. Fowler M. Refactoring: improving the design of existing code. Boston: Addison-Wesley; 1999.
29. Lano K. Design patterns: applications and open issues. 1st cyberpatterns workshop. 2012. In cyberpatterns: unifying design patterns with security and attack patterns. Springer; 2014.
30. Hafiz M. Security on demand. PhD dissertation. University of Illinois. 2011. <http://munawarhafiz.com/research/phdthesis/Munawar-Dissertation.pdf>. Accessed 6 Nov 2013.
31. Buschmann F, Meunier R, Rohnert H, Sommerlad P, Stal M. Pattern-oriented software architecture volume 1: a system of patterns. Chichester: John Wiley; 1996.
32. Buschmann F, Henney K, Schmidt DC. Pattern-oriented software architecture volume 4: a pattern language for distributed computing. Chichester: John Wiley; 2007.
33. Zhu H. Cyberpatterns: a pattern oriented research methodology for studying cyberspace. In cyberpatterns: unifying design patterns with security and attack patterns. Springer; 2014.
34. Sowa J, Zachman JA. Extending and formalizing the framework for information systems architecture. IBM Syst J. 1992;31(3):590–616.
35. Zhu H. Design space-based pattern representation. 1st cyberpatterns workshop. 2012. In unifying design patterns with security and attack patterns. Springer; 2014.
36. Bayley I. Challenges for a formal framework for patterns. 1st cyberpatterns workshop. 2012. In unifying design patterns with security and attack patterns. Springer; 2014.

Cyberpatterns

Unifying Design Patterns with Security and Attack
Patterns

Blackwell, C.; Zhu, H. (Eds.)

2014, XI, 264 p. 48 illus., Hardcover

ISBN: 978-3-319-04446-0