Chapter 2
Using Security Patterns to Develop Secure Systems

Eduardo B. Fernandez
Florida Atlantic University, USA

Nobukazu Yoshioka
GRACE Center, National Institute of Informatics, Japan

Hironori Washizaki
Waseda University, Japan

Jan Jurjens
Technical University of Dortmund, Germany

Michael VanHilst
Florida Atlantic University, USA

Guenther Pernul
University of Regensburg, Germany

ABSTRACT

This chapter describes ongoing work on the use of patterns in the development of secure systems. The work reflects a collaboration among five research centers on three continents. Patterns are applied to all aspects of development, from domain analysis and attack modeling to basic design, and to all aspects of the systems under development, from the database and infrastructure to policies, monitoring, and forensics. The chapter, provides an overview of the method of development involving the full range of patterns, and describes many recent contributions from the many research threads being pursued within the collaboration. Finally, future directions of research in the use of patterns are described.

INTRODUCTION

We initiated an international collaboration between our security groups a few years ago, centered on methodologies to build secure systems using patterns. We describe here where we are now and where we are going. This chapter should be considered a survey of our work and not an attempt to present new work or to introduce in detail the

DOI: 10.4018/978-1-61520-837-1.ch002

models presented here, for that we refer the reader to our previous publications. We also provide a section comparing our work to others but again in each paper we relate our work to others in more detail. In particular, we have worked or we are working on:

- **Secure software development methodology:** We have worked on a general methodology to build secure systems and have produced until now some specific aspects of it, which are described below. Of course, these aspects have value independently of this methodology and can be applied to other methodologies or on their own.
- **Modeling and Classification of security patterns:** We have tried to provide a precise characterization of security patterns that can be used as a basis for classification. A good classification makes the application of the patterns much easier along the software lifecycle. It also helps understand the nature and value of the patterns. Another objective is to identify which patterns are missing.
- **Misuse patterns:** A misuse pattern describes, from the point of view of the attacker, how a type of attack is performed (what units it uses and how), analyzes the ways of stopping the attack by enumerating possible security patterns that can be applied for this purpose, and describes how to trace the attack once it has happened by appropriate collection and observation of forensics data. They can be used in the lifecycle to prevent the occurrence of known types of attacks and to evaluate a completed system.
- **Characterization and selection of access control models:** Access control is a fundamental aspect of security. There are many variations of the basic access control models and it is confusing for a software developer to select an appropriate model for her application. We have defined a way to clarify their relationships and a way to guide designers in selecting an appropriate model.
- **Databases in secure applications:** Most applications need to include databases to store the persistent information, which constitutes most of the information assets of the institution. We have studied the effect of databases on the security of a system under development.

The following sections describe these aspects in detail.

SECURE SOFTWARE DEVELOPMENT METHODOLOGY

A good methodology for design is fundamental to produce secure systems. In Fernandez, Yoshioka, Washizaki & Jürjens (2007) we defined some requirements for such a methodology. Principles to build secure systems have been defined in some classical papers (Saltzer & Schroeder, 1975) and textbooks (Viega & McGraw, 2001), patterns may apply them implicitly. Specific requirements include:

- At each stage, there is guidance on where to apply and how to select appropriate security patterns.
- There are guidelines for pattern selection to satisfy functional requirements or restrictions at each stage.
- There are guidelines to find vulnerabilities and threats in a system.
- There are guidelines to select patterns to mitigate the identified threats.
- The models of the patterns should be relatively detailed and precise, using languages such as UML and OCL to describe the solutions.

- There should be a clear way to apply formalizations at least to specific parts of the design.

Based on these requirements we chose object-oriented design as the most appropriate software methodology because of its ability for abstraction, well-defined life cycle, intuitive nature, and being known by many developers. While it has some limitations, the fact that it is a methodology known to many developers makes it of practical value.

We had proposed in the past separate methodologies (Fernandez, Larrondo-Petrie, Sorgente & VanHilst, 2006, Jürjens, 2004, Yoshioka, 2006). We found that they have many common and complementary aspects and we proposed a combination of them in Fernandez, Yoshioka, Washizaki & Jürjens, (2007). This methodology appears to satisfy all the requirements described above, although it is still not complete. A main idea in the proposed methodology is that security principles should be applied at every stage of the software lifecycle and that each stage can be tested for compliance with security principles. Another basic idea is the use of patterns at each stage. A security pattern describes a solution to a recurrent problem and providing a complete set of them, appropriately classified, can be very useful to developers with little experience on security. The methodology includes the following stages:

- **Domain analysis stage:** A business model is defined. Legacy systems are identified and their security implications analyzed. Domain, institutional, and regulatory constraints are identified. These constraints become policies that apply to the complete system and can be defined in the domain model in the form of patterns. From business goals or institutional points of view, assets in the domain are identified. The suitability of the development team is assessed, possibly leading to added training. This phase is performed only once for each new domain. The possible selections for specialized database architectures and other specific platform requirements should be determined at this point.
- **Requirements stage:** Use cases define the required interactions with the system. Applying the principle that security must start from the highest levels, it makes sense to relate attacks to use cases. Activity diagrams indicate access to existing and created objects and are a good way to determine which data should be protected. Threats in each activity define misuse activities, which might threaten assets which we need to protect. Since many possible threats may be identified we should apply risk analysis to prune them according to their impact and probability of occurrence. Any requirements for degree of security should be expressed as part of the use cases. We then determine which policies would stop these attacks. These include aspects such as mutual authentication to stop impostors, authorization based on roles, need for logging accesses, etc. From the use cases we can also determine the needed rights for each actor and thus apply a need-to-know policy. The security test cases for the complete system are also defined at this stage.
- **Analysis stage:** Analysis patterns can be used to build the conceptual model in a more reliable and efficient way. The policies defined in the requirements can now be expressed as abstract security models, e.g. access matrix, represented as patterns (Schumacher, Fernandez, Hybertson, Buschmann, & Sommerlad, 2006). The model selected must correspond to the type of application; for example, multilevel models have not been successful for medical applications. One can build a conceptual model where repeated applications of a security model pattern realize the rights determined from use cases.

In fact, analysis patterns can be built with predefined authorizations according to the roles in their use cases. Instances of patterns for authentication, logging, and secure channels are also applied at this level (Fernandez & Yuan, 2007). Note that the model and the security patterns should define precisely the requirements of the problem, not its software solution. UML is a good semi-formal approach for defining policies, avoiding the need for ad-hoc policy languages. The addition of OCL (Object Constraint Language) can make the approach more formal. An alternative is the use of UMLSec (Jürjens, 2004), which adds stereotypes to UML to describe the security requirements of the application.

- **Design stage:** When we have found the needed policies and added their pattern representation to the conceptual model, we can select mechanisms that correspond to their concrete software realizations. A specific security model, e.g. Role-Based Access Control (RBAC), is now implemented in terms of software units. Misuse patterns at the design level are useful to analyze how the attacks operate and the security patterns related to the attacks are used to implement the policies. User interfaces should correspond to use cases and may be used to enforce the authorizations defined in the analysis stage when users interact with the system. Components can be secured by using authorization rules for Java or.NET components. Security restrictions can be applied in the distribution architecture; for example, access control for web services. Deployment diagrams can define secure configurations to be used by security administrators. System behavior fragments can be used to consider also performance aspects. A multilayer architecture is needed to enforce the security constraints defined at the application level. In each level we use patterns to represent appropriate security mechanisms. Security constraints must be mapped between levels. Iteration of the application of misuse patterns and security patterns may be useful to remove security holes. The persistent aspects of the conceptual model are typically mapped into relational databases. The design of the database architecture is performed according to the requirements from the uses cases for the level of security needed and the security model adopted in the analysis stage.
- **Implementation stage:** We now reflect in the code the security rules defined in the design stage. Because these rules are expressed as classes, associations, and constraints, they can be implemented as classes in object-oriented languages. In this stage one can also select specific security packages or COTS components, e.g., a firewall product or a cryptographic package. Some of the patterns identified earlier in the cycle can be replaced by COTS components (these can be tested to see if they include a similar pattern). Performance aspects become now important and may require iterations. Attack scenarios derived from attack patterns and test cases are useful to examine systems to find security holes.

An important aspect for the complete design is assurance. We could verify each pattern used but this does not verify that the system using them is secure. We can still say that since we used a careful and systematic methodology with verified and tested patterns, the design should provide a good level of security. The set of patterns can be shown to be able to stop or mitigate the identified threats (Fernandez, Yoshioka & Washizaki, 2009b).

MODELING AND CLASSIFICATION OF SECURITY PATTERNS

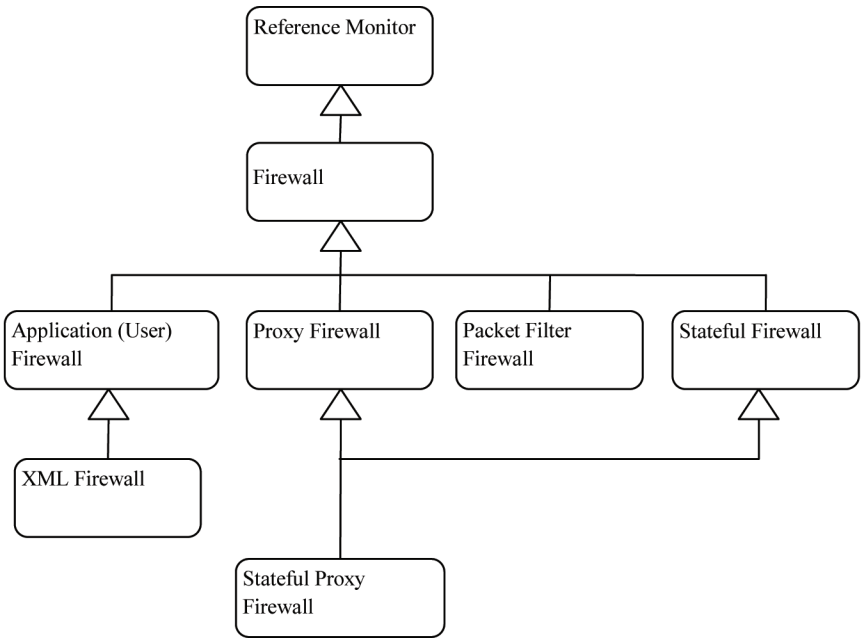
A fundamental tool for any methodology based on patterns is a good catalog. This catalog should be not only complete, covering every stage and architectural level, but also organized in such a way that the designer can find the right pattern at the right moment in the development cycle.

The solution section of a pattern must be given in a generic form and be an abstraction of best practices. It must also provide enough detail and guidance for developers to incorporate them in their applications. This implies that the solution should be expressed as precisely as possible and be complemented with textual descriptions and examples. As indicated earlier, a good way to present the solution is in the form of UML models, which are understood by most software developers and can be easily converted into executable code. UML models can be enhanced with OCL constraints for greater precision (Warmer & Kleppe, 2003). Our patterns in Fernandez & Pernul (2006) and Morrison & Fernandez (2006) are examples of our style. Purely formal definitions of the solution have the problem that most software developers cannot understand them. In addition, there are many formal notations, without any of them becoming an accepted standard. Other authors take the opposite view and prefer short (thumbnail) pattern descriptions, indicating mostly the general idea of the pattern. This approach could be useful to provide a perspective of what is available for a particular domain and a catalog of this type of patterns could complement a more detailed catalog to serve as a roadmap. A requirement for a pattern is that the solution it describes has been used in at least three real systems (Buschmann, Meunier, Rohnert, Sommerlad & Stal, 1996, Gamma, Helm, Johnson & Vlissides, 1994). This is consistent with the idea of patterns as best practices. However, a pattern can also describe solutions that have not been used (or have been used only once) but appear general and useful for several situations. Because of this, we have included sometimes both types: good practices patterns and useful solutions patterns.

Patterns can be defined at several levels of abstraction. The highest level is typically a principle or a very fundamental concept, e.g. the concept of Reference Monitor, which indicates that every access must be intercepted and checked. Another example shows that firewalls, database authorization systems, and operating system access control systems are special cases of access control systems. Figure 1 shows a generalization hierarchy showing that a Firewall pattern is a concrete version of a Reference Monitor. There are four basic types of firewalls, which filter at different architectural levels: the Application (User level) Firewall, the Proxy Firewall (system application), the Stateful Firewall, and the Packet Filter Firewall. An XML Firewall is a specialized type of Application Firewall. One can combine Stateful firewalls with Proxy or Packet Filter firewalls to produce even more specialized types of firewalls such as Stateful Proxy firewall, which combines aspects of both Proxy and Stateful firewalls (Schumacher, et al., 2006). Descriptions of all these patterns can be found in the website of the F.A.U.'s Secure Systems Research Group (2009).

We started with an initial classification (Fernandez, Washizaki, Yoshioka, Kubo & Fukazawa, 2008), where we used three aspects or dimensions to classify patterns: the architectural layers where the patterns belong, the security concerns considered by the patterns, and relationships between the patterns' textual descriptions. Figure 2 shows some patterns classified according to these criteria. Each column describes some concern; in particular we show three concerns: Filtering (using firewalls), Enforcement of access control (using variations of the Reference Monitor), and Authentication. Each concern may appear in multiple levels; for example, we may authenticate at the operating system level or the web service level, or the distribution level.

Figure 1. Firewall patterns generalization hierarchy



We then developed a multidimensional approach, which provides a finer classification (VanHilst, Fernandez & Braz, 2009a, VanHilst, Fernandez & Braz, 2009b). We address pattern classification and problem coverage through the use of a multi-dimensional matrix of concerns. Each dimension of the matrix presents a range of concerns along a single continuum, with a simple concept defined by two polar opposites, like internal and external or begin and end. The categories along each dimension should be easily understood and represent widely used and accepted classifications with respect to that concept. In addition to the dimensions mentioned earlier, another dimension would be a list of lifecycle activities, covering domain analysis, requirements, problem analysis, design, implementation, integration, deployment (including configuration), operation, maintenance and disposal. A pattern applies to a lifecycle stage if a developer could use knowledge from the pattern in performing tasks at that stage. Another dimension is component source, ranging for completely internal new code, to completely external, like a web service. In-between categories include legacy code, library components, outsourced, and COTS. Response to stages of attack is a third dimension, from avoidance of an initial intent, through deterrence, prevention, detection, mitigation, recovery, and investigation (or forensics).

As an example of the use of multiple dimensions, the classification of the XACML Access Control Evaluator pattern is shown in Figure 3. This pattern defines the reference monitor for XACML rules. The classification includes the dimensions mentioned early and also dimensions for domains and constraints (mechanism, human (operator or developer), organizational, and regulatory). Distinctions on the Component Source dimension were considered not significant – hence all are valid. The XACML Access Control Evaluator pattern is part of a pattern language and is related to an XACML protocol pattern for the domain analysis stage, and a more general Access Control abstract pattern for the analysis stage. Early experience indicates that this clas-

Figure 2. Types of patterns based on levels, concerns, and relationships

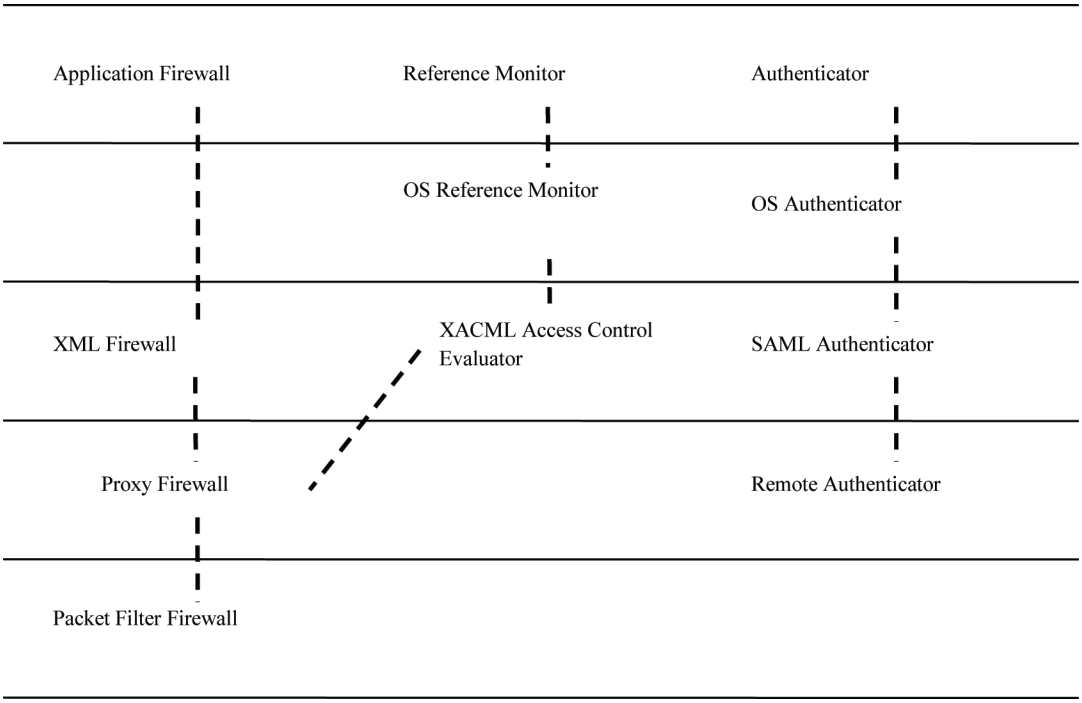
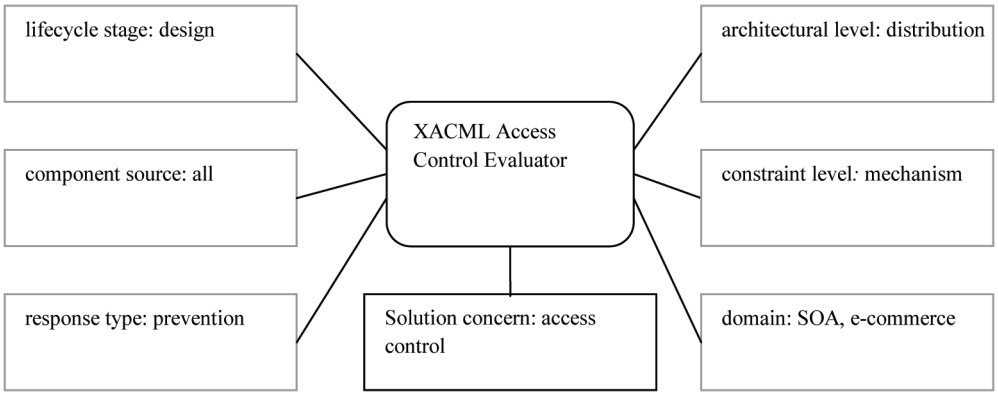


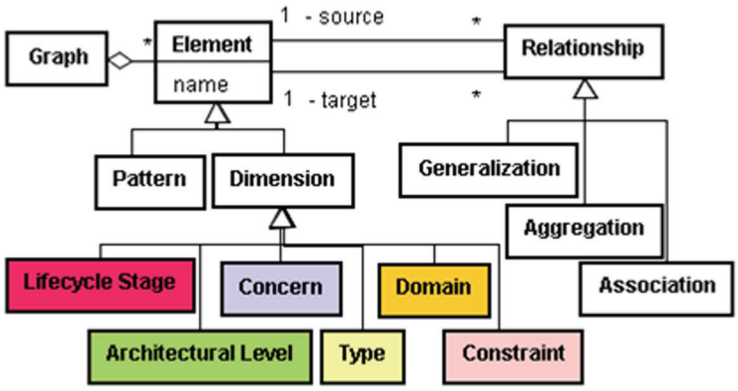
Figure 3. Classifications of the XACML access control evaluator



sification approach is feasible and offers many desirable properties.

We introduced the concept of Dimension Graph (DG), to formalize this classification (Washizaki, Fernandez, Maruyama, Kubo, & Yoshioka, 2009). DGs describe patterns in a multidimensional space relating each pattern to a set of dimensions of interest, i.e. they formalize the multidimensional classification defined earlier. Therefore, DGs are useful to understand properties of each pattern, and to classify precisely the pattern. DGs can be modeled by using details of the target pattern documents and expertise in the patterns.

Figure 4. Metamodel for pattern representation



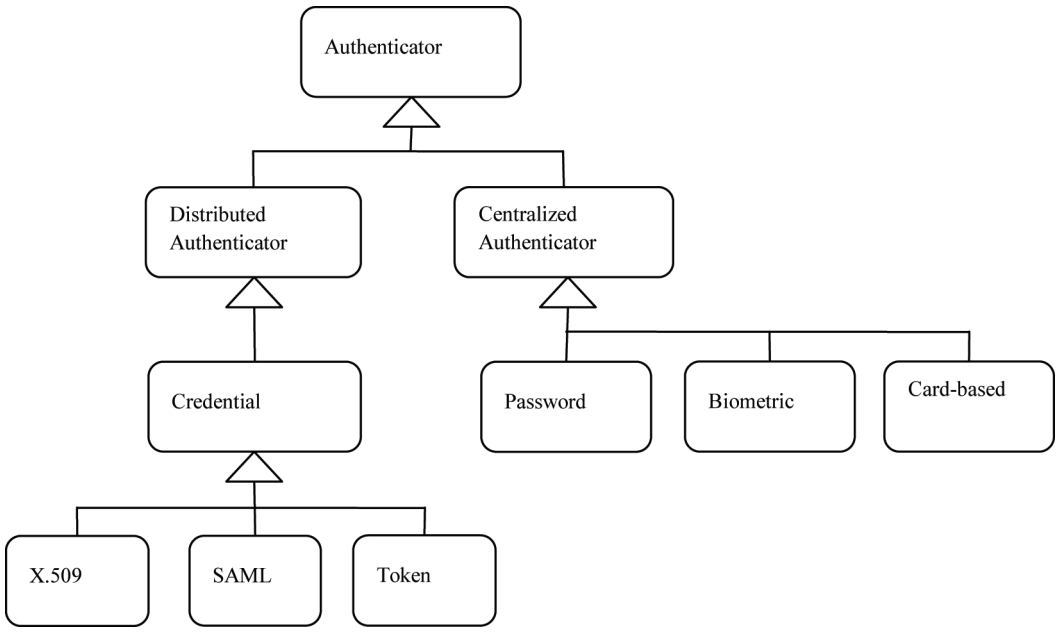
A property of a pattern can be considered as a relationship between the pattern and one of all possible classification dimensions, such as the lifecycle stage (i.e. when to use a pattern) and the concern (i.e. what kind of concerns a pattern addresses). We defined a metamodel to describe the relationships of pattern-to-dimension and pattern-to-pattern uniformly. Figure 4 shows a metamodel to represent patterns as a UML class diagram. The relations of pattern-to-dimension and pattern-to-pattern can be modeled uniformly at the instance level, by using two associations between Element and Relationship, and a generalization hierarchy whose root is Element. In the hierarchy, Element has two children: Pattern and Dimension; therefore it is possible to model both relationships between patterns and between patterns and dimensions. Moreover, it is also possible to describe relationships between dimensions

In the analysis stage of software development we are trying to make the problem precise, we are not concerned with software aspects. From a security point of view we only want to indicate which specific security mechanisms are needed, not their implementation. For this purpose, we introduced the idea of abstract security patterns which define abstract security mechanisms incorporating only the fundamental functions of the specific mechanism. Abstract patterns can also help for classification and systematization of patterns. Figure 5 shows an abstract Authentication pattern which defines the basic functions of any authentication mechanism. More concrete or specialized patterns are defined for specific environments, e.g. credentials for any types of distributed environments (Morrison & Fernandez, 2006). X.509 certificates for computer distributed environments, and SAML assertions for web services are special types of credentials. The specialized patterns all have the basic properties of the abstract authentication pattern but they perform this authentication process in specific ways. This idea can also be used to relate different types of patterns as shown in Figure 2 (Figure 1 was another example, where Firewall ia an abstract pattern).

MISUSE PATTERNS

A misuse pattern describes, from the point of view of an attacker, a generic way of performing an attack that takes advantage of the specific vulnerabilities of some environment or context. It also presents a way to counteract its development as well as a way to trace back the information needed at each stage of the attack. We introduced this concept in Fernandez, Pelaez, & Larrondo-Petrie

Figure 5. The authentication hierarchy



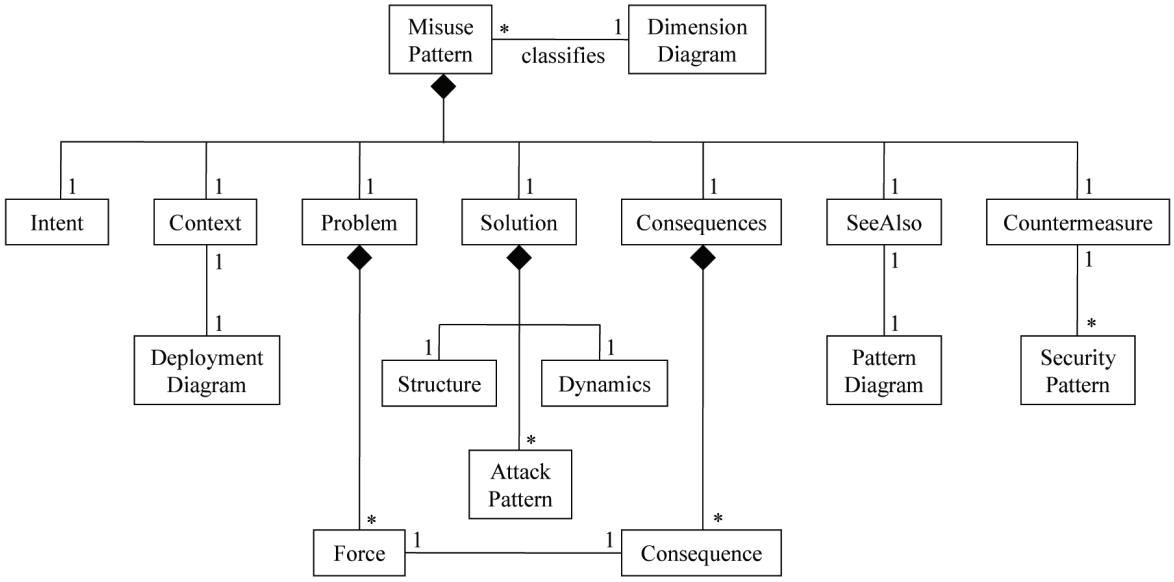
(2007) under the name of attack pattern. Independently, the NII group had developed a similar concept (Yoshioka, Honiden & Finkelstein, 2004) and we merged the two approaches in Fernandez, Yoshioka & Washizaki (2009a). We adopted the name "misuse" pattern because the name attack pattern had already been used for a slightly different concept. A misuse is an unauthorized use of information and our emphasis is in how the misuse is performed, i.e., the steps of the attack and the system units used to perform the misuse.

Figure 6 presents a UML model that describes the sections of a misuse pattern. We describe below the components of the Misuse Pattern class, which correspond to sections of the template used for its desciption.

Intent or thumbnail description: A short description of the intended purpose of the pattern (which problem it solves for an attacker). The context describes the generic environment including the conditions under which the misuse may occur. This may include minimal defenses present in the system as well as typical vulnerabilities of the system. The context is specified using a deployment diagram of the relevant portions of the system as well as sequence or collaboration diagrams that show the normal use of the system.

Problem: From a hacker's perspective, the problem is how to find a way to attack the system The forces indicate what factors may be required in order to accomplish the attack and in what way. The solution describes how the misuse can be accomplished and the expected results of the attack. UML class diagrams describe the relevant portions of the system under attack. Sequence or collaboration diagrams show the exchange of messages needed to accomplish the attack. State or activity diagrams may add further detail. Attack patterns which are necessary for the misuse are also listed here (an attack pattern is a specific action, e.g. a buffer overflow). **Known uses:** This section describes specific incidents where this attack has been used. **Consequences:** Discusses the benefits and drawbacks of a misuse pattern from the attacker's viewpoint. The section on **Countermeasures and Forensics** describes the security measures necessary in order to stop, mitigate, or trace this type of attack. This im-

Figure 6. UML class model for misuse patterns



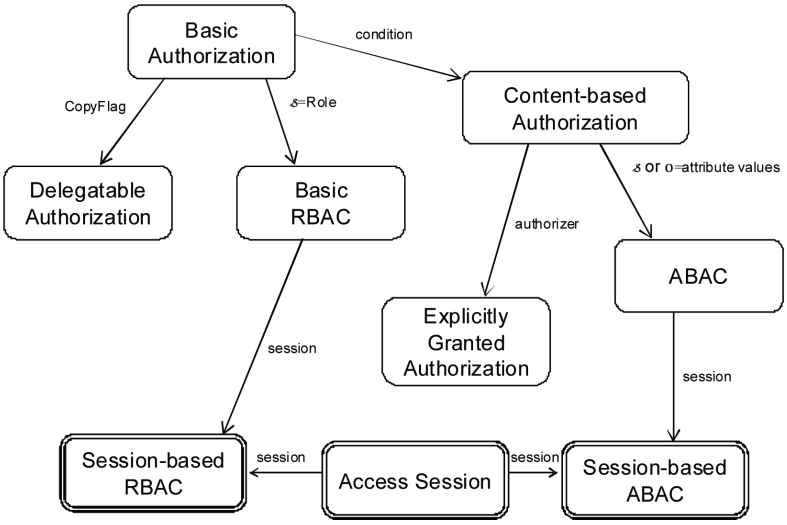
plies an enumeration of which security patterns are effective against this attack. From a forensic viewpoint, it describes what information can be obtained at each stage tracing back the attack and what can be deduced from this data in order to identify this specific attack. Finally, it may indicate what additional information should be collected at the involved units to improve forensic analysis. Each pattern may also carry information about the time it takes to apply its solution (Yoshioka, Honiden & Finkelstein, 2004). **Related Patterns (See also):** Discusses other misuse patterns with different objectives but performed in a similar way or with similar objectives but performed in a different way. It also considers patterns of complementary misuses o patterns of attacks that support the misuse. These patterns can be related using a misuse pattern diagram.

We have applied this approach to the construction of a catalog of the most typical attack patterns in VoIP (Pelaez, Fernandez, & Larrondo-Petrie, 2009). We need to expand this catalog to make it of more general use. Note that as usual, patterns provide only guidelines, not plug-in solutions; that is, for each new application the patterns provide guidelines about what to expect, where to look, and how to start, their solutions must be tailored to the specific environment.

CHARACTERIZATION AND SELECTION OF ACCESS CONTROL MODELS

Access control is a fundamental aspect of security. Because of its importance there are many variations of the basic access control models, emphasizing different aspects. It is confusing for a software developer to select an appropriate model for her application (Fernandez, Pernul & Larrondo-Petrie, 2008). In practice, this confusion results in designers adopting only simple models and missing the richness of other models. We have tried to clarify this panorama through the use of patterns. A pattern diagram shows relationships between patterns (represented by rectangles with rounded corners). In particular, we use pattern diagrams to navigate the pattern space. Figure 7

Figure 7. Relationships between access control patterns



shows some access control models; for example, a Basic Authorization pattern has components *(s,o,t)*; making *s* a role we get the Basic Role-Based Access Control (RBAC) pattern. Adding sessions we obtain Session-based RBAC, and so on. A subproduct of our work is the analysis of which patterns are available for use and which need to be written. The goal here is to provide the designer of a secure system with a navigation tool that she can use to select an appropriate pattern from a catalog of security patterns. Our examples show how to compose new access control models by adding features to an existing pattern and how to define patterns by analogy. The patterns with a double border are the ones discussed in (Fernandez & Pernul, 2006).

Using our approach of mapping relationships, we can study complex models, e.g. Attribute-Based Access Control (ABAC), where the attributes of the subject and object determine access, and relate them to simpler patterns (Priebe, Fernandez, Mehlau, & Pernul, 2004). We can combine access control with auxiliary functions such as sessions (Fernandez & Pernul, 2006). Pattern maps are also useful to perform semi-automatic model transformations as required for Model-Driven Development (MDD). For MDD, they can serve as metamodels of possible solutions being added at each transformation.

A pattern in the pattern diagram shows how it is related to other patterns. We believe that this perspective can help developers to align their needs with the selection of appropriate access control models. The selected access control pattern not only guides the conceptual security of the application but later it also guides the actual implementation of the model. We can navigate the pattern diagram because patterns are composable with respect to features, i.e. adding a feature (perhaps embodied by another pattern) produces a new pattern with extra features. This aspect of their composition can be understood in terms of object-oriented models, where a new feature implies the addition of a new class or of a class attribute.

Our focus on the relationships among access control models and patterns has led us also to discover the need for new security patterns: Subject, Object, Labeled Security, DAC, MAC. These are concepts with models for which there are no patterns or which are needed to precisely represent some aspects of existing patterns. For

example, Labeled Security is necessary to implement mandatory multilevel models. Access control patterns give us also the possibility of evaluating commercial products: we can see if the product contains the corresponding pattern in its design.

ADDING DATABASES TO THE SECURE METHODOLOGY

A design aspect which is interesting and not much studied is the incorporation of databases as part of the secure architecture (Fernandez, Jürjens, Yoshioka & Washizaki, 2008). The database system is a fundamental aspect for security because it stores the persistent information, which constitutes most of the information assets of the institution. We presented some ideas on how to make sure that the database system has the same level of security as the rest of the secure application.

An interesting problem in when incorporating database management systems is the mapping from the conceptual security model (that may apply to a collection of DBMSs) to the authorization system of a specific database. For example, security constraints defined in a conceptual UML model defining authorizations in terms of classes (Fernandez & Yuan, 2007) must be mapped to an SQL-based authorization system which defines authorizations in terms of relations. Clearly, whatever is defined in the common conceptual model must be reflected in the DBMS.

RELATED WORK

A general methodology for developing security-critical software which has common aspects with this methodology has been proposed in Jürjens (2004). It makes use of an extension of the Unified Modeling Language (UML) to include security-relevant information, which is called UMLsec. The approach is supported by extensive automated tool-support for performing a security analysis of the UMLsec models against the security requirements and has been used in a variety of industrial projects (Best, Jürjens, & Nuseibeh, 2007). As mentioned earlier, we are incorporating some parts of this model in our methodology.

Mouratidis and his group use the Secure Tropos methodology to model security. Their work includes modeling requirements (Mouratidis, Jürjens & Fox, 2006, Mouratidis & Giorgini, 2004). They have also considered other stages; for example how to test security along the lifecycle (Mouratidis & Giorgini, 2004). They use patterns. However, instead of UML, they use special diagrams expressed in a special notation, which has its advantages and disadvantages. By connecting security requirements to misuse patterns, our approach provides more context on the specific threats to a protected resource. However, this approach is complementary to ours in several respects.

There have been several attempts to classify security patterns. Hafiz, Adamczyk & Johnson (2007) proposed several classification dimensions to organize security patterns. One of these is based on security objectives, such as confidentiality, integrity, and availability. Their dimensions are subsumed in our set of dimensions (concerns). Rosado, Gutierrez, Fernandez-Medina, & Piattini (2006) related security requirements to security patterns, and classified security patterns into just two categories: architectural patterns and design patterns. Weiss and Mouratidis (2009) have looked at the issue of selecting security patterns for fulfilling security requirements.

Unfortunately, much of the work by others on classifying security patterns has been motivated by a desire to create taxonomies with which to group or distinguish patterns. As a result, the proposed classifications have dealt with only a few dimensions of classification and arranged them hierarchically, rather than treating them uniformly as independent properties of facets. It is difficult with those approaches for users to select and/or find appropriate patterns from a number of pat-

terns with any precision, or from the viewpoints of both pattern relations and properties. Our multidimensional classification is an attempt to improve on the other approaches.

CONCLUSIONS AND FUTURE WORK

Although this collaboration has been unstructured and between people far apart, it has resulted in a significant amount of work. This is due in part to the similarities of our interests and objectives. On his part, the first author and his group have produced a variety of security patterns (Secure Systems Research Group, 2009), fundamental to the application of the methods proposed here.

Future work includes:

- **Formalization and model checking.** As shown by the template of Figure 6 and by the templates normally used to describe patterns, patterns include textual descriptions as well as precise models. The textual descriptions are important for usability, so a pattern should not be fully formalized. However, the solution section and perhaps the related patterns sections can be formalized. How then do we incorporate formal methods into our secure methodology? We are investigating to augment patterns in ways that aid the process of producing formally verifiable systems, or at least formalizing aspects of systems.
- **Verification of security.** We cannot prove formally that a system produced in our approach is secure, due to the complexity of the applications we consider. We can however, verify that all identified threats are covered. Preliminary ideas are shown in Fernandez, Yoshioka & Washizaki (2009b).
- **Tool support.** Threat enumeration and the application of patterns following an MDA approach are highly desirable (Delessy & Fernandez, 2008). We need to develop tools to make development systematic.
- **Once we have enumerated the threats of a system, we define the policies needed to stop or mitigate them.** However, it is not simple to go from these policies to security patterns that describe the actual configuration of the system (Braz, Fernandez & VanHilst, 2008), this aspect needs more work.
- **Run-time monitoring.** How do we measure if a design is actually keeping a system secure? To date, there is little work on this topic (Nagaratnam, Nadalin, Hondo, McIntosh & Austel, 2005).

REFERENCES

Best, B., Jürjens, J., & Nuseibeh, B. (2007). Model-Based Security Engineering of Distributed Information Systems Using UMLsec. In *Proceedings of the 29th International Conference on Software Engineering* (pp. 581-590). New York: ACM.

Braz, F., Fernandez, E. B., & VanHilst, M. (2008). Eliciting security requirements through misuse activities. In *Proceedings of the 19th International Workshop on Database and Expert Systems Applications* (pp. 328-333). Los Alamitos, CA: IEEE Computer Society.

Buschmann, F., Meunier, R., Rohnert, H., Sommerlad, P., & Stal, M. (1996). Pattern-Oriented Software Architecture: *Vol. 1. A System of Patterns*. West Sussex, England: John Wiley & Sons.

Delessy, N., & Fernandez, E. B. (2008). A pattern-driven security process for SOA applications. In *Proceedings of the 3rd International Conference on Availability, Reliability, and Security* (pp. 416-421). Washington DC: IEEE Computer Society.

Fernandez, E. B., Jürjens, J., Yoshioka, N., & Washizaki, H. (2008). Incorporating database systems into a secure software development methodology. In *Proceedings of the 2008 19th International Conference on Database and Expert Systems Application* (pp. 310-314). Washington DC: IEEE Computer Society.

Fernandez, E. B., Larrondo-Petrie, M. M., Sorgente, T., & VanHilst, M. (2006). A Methodology to Develop Secure Systems Using Patterns. In Mouratidis, H., & Giorgini, P. (Eds.), *Integrating Security and Software Engineering: Advances and Future Vision* (pp. 107–126). Hershey, PA: IDEA Group.

Fernandez, E. B., Pelaez, J. C., & Larrondo-Petrie, M. M. (2007). Attack patterns: A new forensic and design tool. In P. Craiger & S. Shenoi (Eds.) *Advances in Digital Forensics III: Proceedings of the Third Annual IFIP WG 11.9 International Conference on Digital Forensics* (pp. 345-357). Berlin, Germany: Springer.

Fernandez, E. B., & Pernul, G. (2006). Patterns for session-based access control. In *Proceedings of the Conference on Pattern Languages of Programs*. Hillside Group. Retrieved November 25, 2009, from http://hillside.net/plop/2006/.

Fernandez, E. B., Pernul, G., & Larrondo-Petrie, M. M. (2008). Patterns and pattern diagrams for access control. In S. Furnell; S.K. Katsikas, & A. Lioy (Eds.) *LNCS 5185: Trust, Privacy and Security in Digital Business: 5th International Conference on Trust and Privacy in Digital Business* (pp. 38-47). Heidelberg, Germany: Springer.

Fernandez, E. B., Washizaki, H., Yoshioka, N., Kubo, A., & Fukazawa, Y. (2008). Classifying security patterns., In Y. Zhang, G. Yu, & E. Bertino (Eds.) *LNCS 4976 Progress in WWW Research and Development: Proceedings of the 10th Asia-Pacific Web Conference* (pp. 342-347). Heidelberg, Germany: Springer.

Fernandez, E. B., Yoshioka, N., & Washizaki, H. (2009a). Modeling misuse patterns. In *Proceedings of the International Conference on Availability, Reliability and Security* (pp. 566-571). Los Alamitos, CA: IEEE Computer Society.

Fernandez, E. B., Yoshioka, N., & Washizaki, H. (2009b). Security patterns and quality. In H. Washizaki, N. Yoshioka, E.B.Fernandez, & J. Jürjens (Eds.) *Proceedings of the Third International Workshop on Software Patterns and Quality* (pp. 46-47). *),* in conjuction with OOPSLA 2009. Retrieved November 25, 2009 from http://grace-center.jp/downloads/GRACE-TR-2009-07.pdf.

Fernandez, E. B., Yoshioka, N., Washizaki, H., & Jürjens, J. (2007). Using security patterns to build secure systems. *Proceedings of the 1st International Workshop on Software Patterns and Quality*, Retrieved November 25, 2009, from http://apsec2007.fuka.info.waseda.ac.jp/parts/W3SPAQu.pdf.

Fernandez, E. B., & Yuan, X. Y. (2007). Securing analysis patterns. In D. John and S.N. Kerr (Eds.) *Proceedings. of the 45th ACM Southeast Conference* (pp. 288-293), New York: ACM.

Gamma, E., Helm, R., Johnson, R., & Vlissides, J. M. (1994). *Design Patterns: Elements of Reusable Object-Oriented Software*. Reading, MA: Addison-Wesley Professional.

Hafiz, M., Adamczyk, P., & Johnson, R. E. (2007). Organizing security patterns. *IEEE Software*, *24*(4), 52–60. doi:10.1109/MS.2007.114

Jürjens, J. (2004). *Secure Systems Development with UML*. Heidelberg, Germany: Springer.

Morrison, P., & Fernandez, E. B. (2006). The credential pattern. In *Proceedings of the Conference on Pattern Languages of Programs*. Hillside Group. Retrieved November 25, 2009, from http://hillside.net/plop/2006/.

Mouratidis, H., & Giorgini, P. (2004). Analysing security in information systems. Presented at the *Second International Workshop on Security in Information Systems*, Porto Portugal. Retrieved November 25, 2009, from http://www.dit.unitn.it/~pgiorgio/papers/ICEISWorkshop04.pdf

Mouratidis, H., Jürjens, J., & Fox, J. (2006). Towards a Comprehensive Framework for Secure Systems Development. In *LNCS 4001: Proceedings of the 18th Conference on Advanced Information Systems,* (pp. 48-62). Heidelberg, Germany: Springer.

Nagaratnam, N., Nadalin, A., Hondo, M., McIntosh, M., & Austel, P. (2005). Business-driven application security: From modeling to managing secure applications. *IBM Systems Journal*, *44*(4), 847–867. doi:10.1147/sj.444.0847

Pelaez, J., Fernandez, E. B., & Larrondo-Petrie, M. M. (2009). Misuse patterns in VoIP. *Security and Communication Networks*. Wiley InterScience. Retrieved November 25, 2009 from http://www3.interscience.wiley.com/journal/122324463/abstract.

Priebe, T., Fernandez, E. B., Mehlau, J. I., & Pernul, G. (2004). A pattern system for access control. In C. Farkas and P. Samarati (Eds.) *Research Directions in Data and Applications Security XVIII: Proceedings of the 18th. Annual IFIP WG 11.3 Working Conference on Data and Applications Security* (pp. 25-28). Amsterdam, Netherlands: Kluwer Academic Publishers.

Rosado, D. G., Gutierrez, C., Fernandez-Medina, E., & Piattini, M. (2006). Security patterns related to security requirements. In E. Fernandez-Medina and M. Inmaculada (Eds.) *Security in Informaiton Systems: Proceedings of the 4th International Workshop on Security in Information Systems*. Setúbal, Portugal: INSTICC Press.

Saltzer, J. H., & Schroeder, M. D. (1975). The protection of information in computer systems. *Proceedings of the IEEE*, 63(9), 1278-1308. Retrieved November 25, 2009 from http://web.mit.edu/Saltzer/www/publications/protection/index.html

Schumacher, M., Fernandez, E. B., Hybertson, D., Buschmann, F., & Sommerlad, P. (2006). *Security patterns: Integrating security and systems engineering*. Hoboken, NJ: John Wiley & Sons.

Secure Systems Research Group. (2009). Florida Atlantic University. Retrieved November 25, 2009 from http://security.ceecs.fau.edu/

VanHilst, M., Fernandez, E. B., & Braz, F. (2009a). A multidimensional classification for users of security patterns. *Journal of Research and Practice in Information Technology*, *41*(2), 87–97.

VanHilst, M., Fernandez, E. B., & Braz, F. (2009b). Building a concept grid to classify security patterns. In H. Washizaki, N. Yoshioka, E.B.Fernandez, & J. Jürjens (Eds.) *Proceedings of the Third International Workshop on Software Patterns and Quality* (pp. 34-39). Tokyo:NII. Retrieved November 25, 2009 from http://grace-center.jp/downloads/GRACE-TR-2009-07.pdf.

Viega, J., & McGraw, G. (2001). *Building secure software: How to avoid security problems the right way*. Boston: Addison-Wesley.

Warmer, J., & Kleppe, A. (2003). *The object constraint language* (2nd ed.). Boston: Addison-Wesley.

Washizaki, H., Fernandez, E. B., Maruyama, K., Kubo, A., & Yoshioka, N. (2009). Improving the classification of security patterns. In *Proceedings of the International Workshop on Database and Expert Systems Applications* (pp. 165-170). Los Alamitos, CA: IEEE Computer Society.

Weiss, M., & Mouratidis, H. (2008) Selecting security patterns that fulfill security requirements, *Proceedings of the 16th IEEE International Conference on Requirements Engineering (RE'08*), IEEE Computer Society, pp. 169-172

Yoshioka, N. (2006, March 29). A development method based on security patterns. Presented at National Institute of Informatics. Tokyo, Japan.

Yoshioka, N., Honiden, S., & Finkelstein, A. (2004) Security patterns: A method for constructing secure and efficient inter-company coordination systems. In *Proceedings of the Eighth IEEE International Enterprise Distributed Object Computing Conference* (pp. 84-97). Los Alamitos, CA: IEEE Computer Society.