Security Concerns in an Aspect-Oriented Modeling Approach

Dan Matheson

Colorado State University
Computer Science Department
211 University Services Center
601 South Howes Street
Fort Collins, Colorado, 80523, USA
+1 970 225 9147
+1 970 225 9082 fax
matheson [AT] cs [DOT] colostate [DOT] edu

Abstract

Security concerns are present in many software solutions and products. While the functional requirements most often drive the development of models in Model Driven Development (MDD), the modeling of non-functional concerns is equaling important for a high quality solution. Aspect Oriented Modeling (AOM) is an MDD approach that helps develop higher quality solutions by considering various requirements independently and composing the separate sub-solution models into a complete solution. It is possible to express the solutions for different security concerns as Aspects, at both generic and technology specific levels, for use in an AOM-based solution.

This paper explores some of the possible patterns for the common security concerns of authentication, authorization and data privacy. Generic aspect models will be developed along with the marks or mapping points for application to the primary functional model. The security concerns within a solution are not homogeneous, parts of the solution need higher levels of security than other parts. The security variations can significantly increase the solution complexity. Just as important as the Aspects are process guidelines that recommend the appropriate order of composition, so that undesirable emergent properties do not occur.

1 Introduction

Security concerns are present in all software solutions and products. While the functional requirements most often drive the development of models in Model Driven Development (MDD), the modeling of non-functional concerns is equally important for a high quality solution. In particular for solutions that involve the Internet, security is of high concern[Sch00]. The security concerns within a solution are not homogeneous, parts of the solution need different levels and

types of security than other parts. The security variations can significantly increase the solution complexity. This increase in complexity can be controlled to a large degree by using an aspect-oriented modeling approach.

Aspect Oriented Modeling (AOM) [FrR+04] is an MDD approach that helps software engineers develop better solutions by allowing them to consider various requirements independently and compose the separate sub-solution models into a complete solution. The solutions to non-functional requirements often involve additional functionality beyond the basic solution functionality.

Knowledge of security techniques and how to apply them effectively is uncommon in the average software engineer. The use of design patterns to raise the effectiveness of a software engineer is well documented [GaH+95]. The security patterns along with characteristics that describe the strengths and weaknesses can be stored in a repository. The software engineer can then choose a tested pattern in the form of an aspect model that meets the solution requirements. This should raise the quality level of the solution, as well as making the software engineer more effective in creating a solution.

Even more rare than knowledge of the various security techniques is a good understanding of the interactions and dependencies. This includes the dependencies between security approaches as well as interactions with different solution engineering viewpoints. For example given a solution that is distributed with strong fault tolerance requirements and multiple administrative domains, which authentication mechanism is the best choice?

The first step in this process is determining a way to express the different security concerns that are present in a solution. At different levels of abstraction different expressions of the concerns and related solution patterns are needed. The requirements are often supplied in a text form that uses domain specific terms. However, common words and phrases often appear in the requirements, these words will be used to categorize the various security concerns. The catalog listing of the generic solution provides an intermediate step that maps well to the requirement for the non-technical people.

Within each category solution designs with various trade-offs will be collected. At the design level the Unified Modeling Language (UML) [Uml04] will be used to specify the design patterns. The design will have both structural elements such as classes and behavioral elements such as interaction diagrams. The set of elements then forms a design pattern for that concern. It is possible to have several design patterns for a particular concern like confidentiality that represent alternative solutions. It is also possible to have the solution represented at lower levels of abstraction using for example *AspectJ* [Asj04] for the code level.

The capability to select a solution pattern from a catalog of predesigned and tested solutions for a particular security concern should raise the effectiveness of the software engineer. The addition of technology specific models to the catalog insures that the programmer starts from a well defined and requirements aligned position.

With the development of these design patterns and the translation into aspects, they can be stored in a repository for re-use. The addition of relationships among the patterns help the engineer understand the consequences of picking a particular solution.

The paper is organized into the following sections: First a brief introduction of Aspect-Oriented Modeling is given to set the context for the rest of the paper. The next section defines some of the patterns for the common security concerns of authentication, authorization, access control, data integrity and confidentiality. The section following the definitions deals with composing the aspects with the primary functional model. Consequences of the security aspects on a repository for storage of the models is examined. The relation of this work to other research is discussed. Finally the conclusions and future research are listed.

2 Aspect-Oriented Modeling Overview

Aspect-Oriented Modeling is a type of Model Driven Development that composes several models together to create a more comprehensive solution model. The typical situation has several model each built for a specific purpose. There is a primary model that embodies a solution of the most important functional requirements. There can be one or more aspect models each embodying a solution for a non-functional requirement. The aspect models are composed with the primary model according to a set of composition rules to produce a more comprehensive solution model. Early work in separation of concerns is by S. Clark [CIH+99] [CHO+99]. Later work in AOM comes from R. B. France [FrR+04] using authorization as an example.

In this paper I focus on a modeling environment based on the Unified Modeling Language [Uml04]. The models can be any of the types: class, interaction, activity, etc. The same model types in the aspect and the primary model are composed together. One does not compose an interaction model with a class model.

There can be several aspects that are used to represent the complementary functionality that is needed to fulfill a non-functional requirement. The aspect model can consist of additions, deletions or changes to the primary model or to previously composed aspects. The primary power of aspects is that they can uniformly apply a change in specific places scattered across the whole model.

The order of composition is as important as the contents of the models, since the wrong composition order can yield an incorrect solution model. To discover unacceptable composition consequences it is important that an analysis check and testing follow any composition action. In many cases there are emergent properties of the model that are only visible after composition. When these are discovered changes are applied to any of the aspects, primary model or composition directives to correct the deficiencies.

In the case of the security category of aspects the composition order is critical. For example, improper composition of a confidentiality aspect with an auditing aspect could place private data into an audit log.

The security concerns often apply to specific sections of a model. Different security concerns will apply to different parts of the model. The AOM approach provides a good mechanism for identifying those places and applying the solution for those concerns.

Work on security patterns provides a reasonable starting point for developing security aspects. A general overview on the state of work in security patterns can be found in [Rod03].

3 Security Concerns

Phrases in requirements like “access is limited to second level managers and up”, “only the approved suppliers can see this information”, or from regulatory sources like “employee compensation data must be transmitted to the IRS intact” are sources of security concerns. The term “security” is too general to be useful. Several categories of security concerns can be identified and need to be identified if they are to be handled effectively. While specific industries or government situations might have additional categories the following list covers most design situations:

- Identity and authentication
- Authorization and access control
- Data integrity
- Confidentiality or data privacy
- Auditing
- Data Authenticity
- Availability
- Non-repudiation

Identifying users and authenticating them is the basis for handling all the other concerns. This typically takes the form of a unique login identifier or name and some unique information, a password, that the user enters at the start of a session with the solution. Sometimes there are shared identifiers for access to public information. There can also be multiple pieces of information that a user must supply to prove his identity. These variations will appear in the requirements and necessitate multiple design solution options. The patterns *Single Access Point* and *Session* [YoB97] are solutions for these concerns.

Authorization is a solution to the concern of limiting what a user can do. This can apply to initiating actions or accessing data. In looking at requirements

expressed in a natural language, authorization is often used to indicate limitations on a user's actions, while access control is most often used to express a user's limit on accessing data. The pattern *Check Point* [YoB97] is a possible solution for this concern.

Data integrity requirements mostly come from two sources. The first is the business valuable data that is needed to run the business. The second often comes from regulatory agencies, SEC, IRS and laws (Sarbanes-Oxley). They generally converge around limiting changes to the data and tracking, when the change was made and who made the change.

In addition to this simple definition data integrity can be expanded to include an understanding of the source of the data and if it is up-to-date [Rom01]. Does the data come from an authoritative or authentic source? This also increases the identity concern to servers and users. Depending on the environment this might lead to authenticating the server or even mutual authentication. Achieving this might require additional functionality to ensure non-repudiation of the data.

The security concerns are not independent from each other. The relationships between them are shown in Table 1. The table is read *row label cell text column label*, for example *Identity used by Authorization.*

	Identity	***Author.***	***Integrity***	***Privacy***	***Audit***	***Authen.***	***Avail.***	***Non-repudiation***
Identity		Used by	Used by	Used by	Used by	Used by		Used by
Authorization	Requires		Used by	Used by	Used by		Used by	
Integrity	Uses	Uses						
Privacy	Requires	Requires			Conflicts			
Audit	Requires	Uses	Requires	Conflicts				Used by
Authenticity	Requires	Requires	Has					
Availability		Requires						
Non-repudiation	Requires				Requires			

Table 1 Security Pattern Dependencies

As the relationships in Table 1 indicate there are structural and behavioral dependencies. If there is a concern about authorization, then identification and authentication of the user or principle is needed before the authorization calculation can be performed. The identification and authentication actions add functional actions and data storage requirements to the solution.

In this paper I will only deal with the first level of interactions. Secondary levels of interaction can be envisioned. For example, the common system login requires integrity on the identities and confidentiality on the passwords. This administration data can be composed with aspects that solve these concerns.

Without a careful design process a complex spiral can occur.

There are well accepted security principles that are often expressed in a security policy and therefore appear in the requirements. One of the most important is the principle of *least privilege*, which says that a user or principal should operate with the smallest number and least powerful privileges possible. This would apply to the authorization concerns.

The realization of the security concerns results in other concepts emerging. For example the identity and authentication realization results in the concept of a session. The session data needs to be passed around the software artifacts and the integrity of session data needs to be maintained as well.

A well designed solution with strong security concerns often takes a layered approach [Rom01]. As the primary model and the aspects are developed the realization of a particular concern can result in several different mechanisms fulfilling requirements and guided by the layering principle. For example tests on network addresses can be combined with user identification to restrict access. This can involve solution components outside the immediate software application such as a network router's ACLs.

4 Security Aspect Definitions

This section contains definitions of various security aspects as UML class diagrams.

In illustration 1 a class model for a simple access control list is presented. Each *subject* has an *access_list* object. Each *access_list* object contains a reference to one or more *principal* objects which could be an individual or a group. Each *principal* has a list of allowed *permissions*.

The *access()* method in subject represents any access to that object. The internal logic of *access()* will make a call to the private method *check_access()* to determine if this access type is allowed for this principal. The check_access() method then calls the *access_list:check()* method which finds the appropriate principal or principals when groups are involved. It then checks all combinations until success occurs or no success occurs.

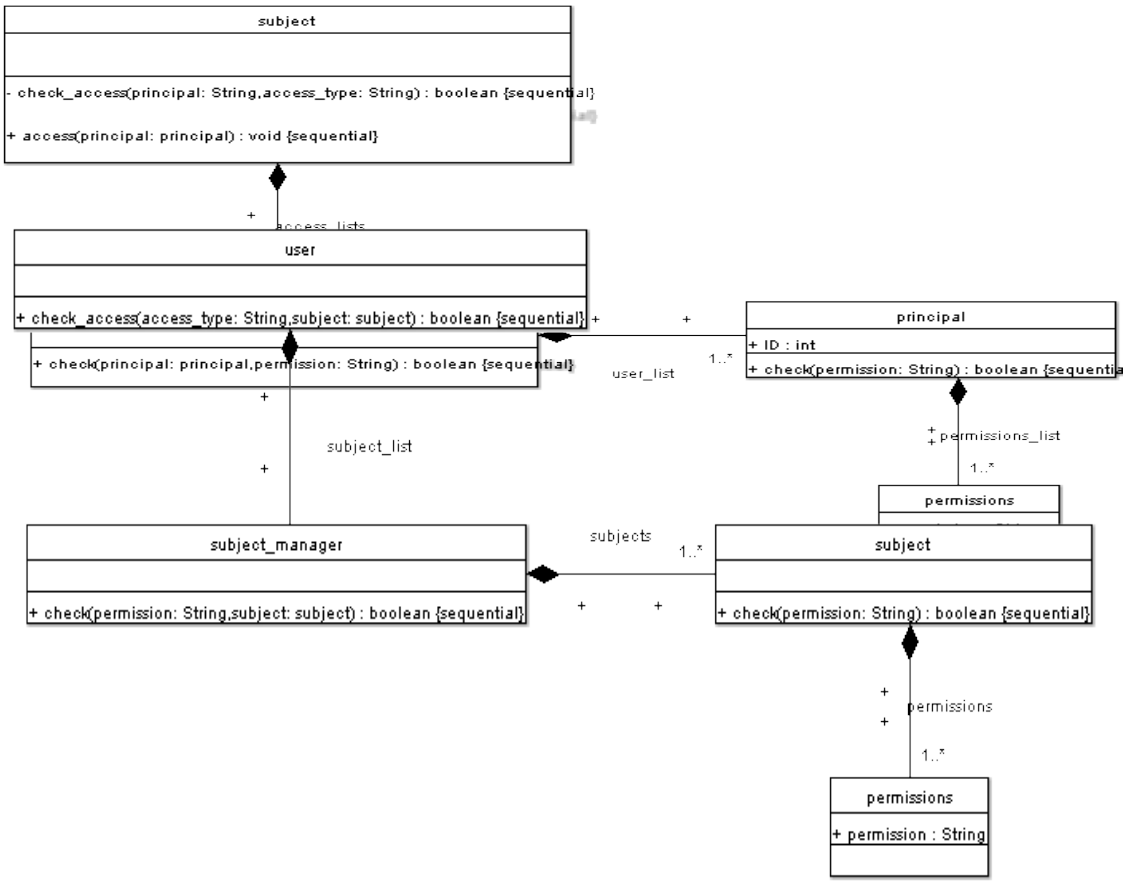

Illustration 2 Simple Capabilities Aspect Model

A simple capabilities access control class model is shown in illustration 2. A *user* object has a *subject_manager* which manages a list of *subjects* that the user has access to. Each subject has the set of *permissions* that are granted to that user.

A common access control mechanism is the Role-Based Access Control (RBAC). In illustration 3 is a class model for RBAC from AOM research by I. Ray [RFL03]. It shows the classes that reflect the basic concepts of RBAC: a user, a role, a session and permissions.

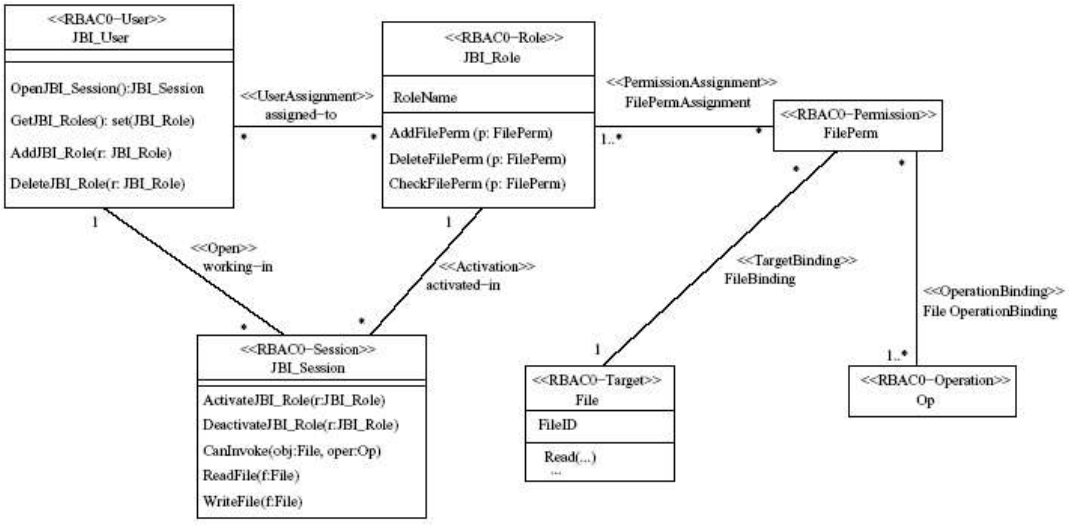

Illustration 3 RBAC Model Example [RFL03]

Each of these high level designs show how a high level concern encapsulates a number of data entities and behavior. This emphasizes the realization of many non-functional requirements via functionality.

5 Security Aspect Relationships

A repository for models that supports AOM will maintain the consistency of the models and the relationships between them. The primary relationship from an AOM perspective is the mapping of the aspect model onto the primary model. The usefulness of the repository is greatly enhanced if other types of relationships between models are supported. From the perspective of this research maintaining **dependency** and **refines** kinds of relationships will help produce higher quality solutions.

The **dependency** type relationship is used to construct a more complete security solution pattern, that matches to a higher level expression of a security concern. For example, our security concern is confidentiality. We construct an abstract model of the concern that captures the primary concepts of maintaining confidentiality. We know that supporting concepts of user identity and authentication are key to achieving that goal. The more complete confidentiality model will have the dependency relationship to the identity model created. Illustration 4 shows the top level dependency relationships.

A composition tool can use the extended solution model as directives for ordering the aspect composition with the primary model. An analysis tool can refer to the confidentiality concern solution model as an extended specification to be tested against.

The **refines** relationship is used to capture refinements of a concern as well as

refinements of relationships between the aspect models. For example, the confidentiality concern can be refined by access control, or encryption or any other information hiding technology. If encryption is picked as the general engineering solution approach, then a number of supporting technologies are needed. If a particular encryption technology is picked, such as AES, there will be a number of appropriate supporting technologies for key management, identification, etc. needed.

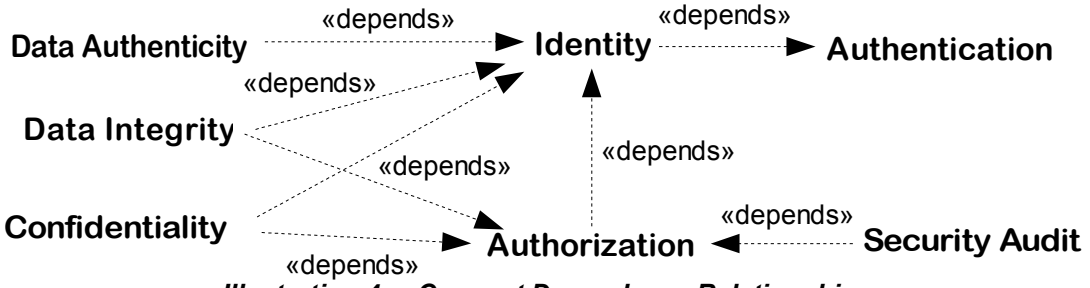

Illustration 4 Concept Dependency Relationships

Illustrations 5, 6, 7 show the first level refinements of the basic concepts. These concepts can now be cast in the UML. The UML *package* could be used for the

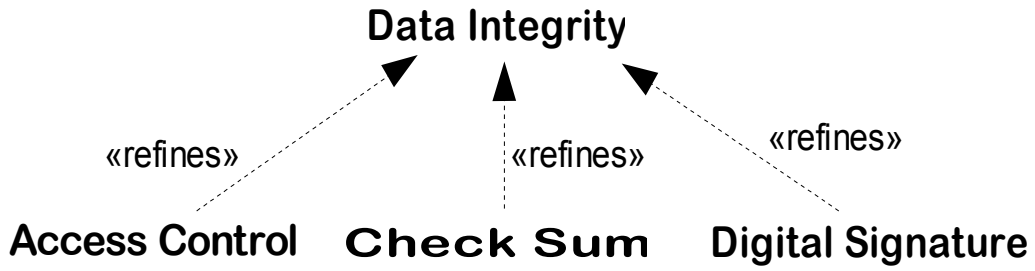

Illustration 5 Data Integrity Refinement, First Level

first level refinement to specify dependencies and provide structuring for refinements.

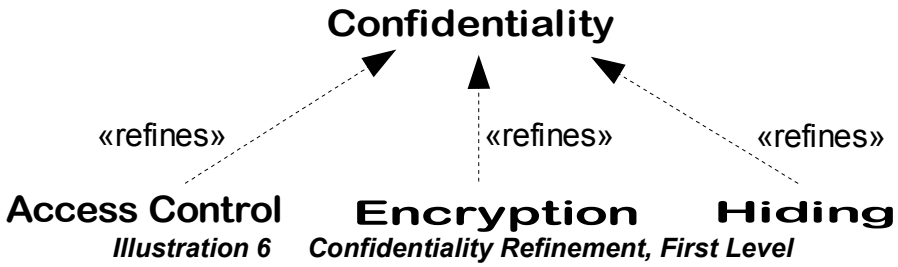

Illustration 6 Confidentiality Refinement, First Level

The refinement example in Illustration 7 shows how the «refines» relationship leads to a set of «depends» between models. At the next level of refinement there are multiple depends relationships between the entities in each model as

well as newly un-encapsulated entities.

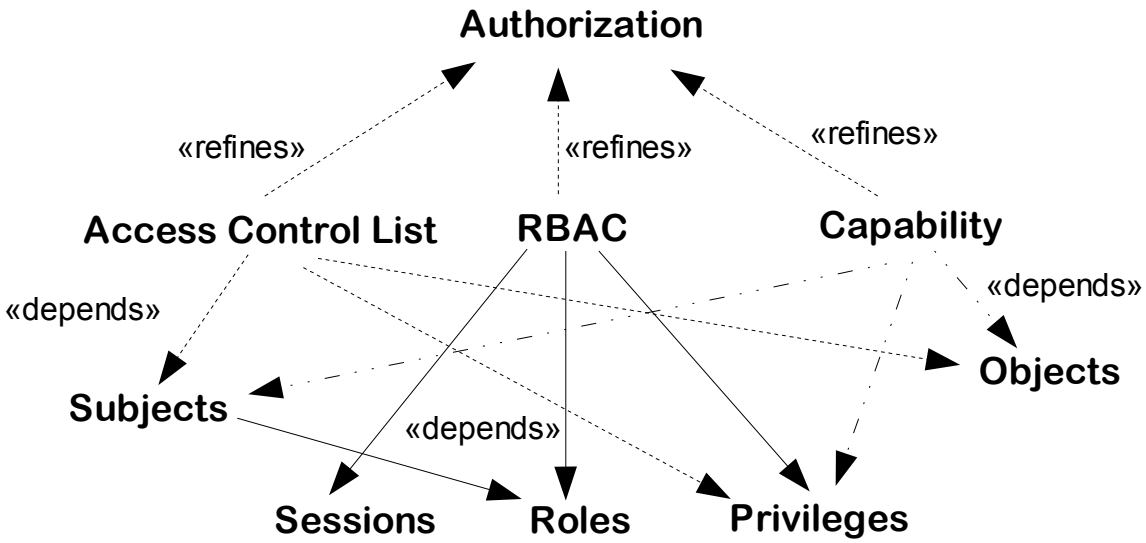

Illustration 7 Authorization Refinement with Dependencies

6 Applying Security Aspects

Selecting and applying the security aspects properly requires a good understanding of the advantages as well as the risks involved. The dependencies discussed in the previous section show supporting aspects which must be included in the composition.

Just as important as the individual aspect patterns are process guidelines that recommend the appropriate order of composition of the patterns, so that undesirable emergent properties do not occur. There are design principles that can be followed to achieve a good result.

The concept of a security pattern system is discussed by Schumacher [ScR01]. The security pattern system considers the set of security patterns that work together. Currently this focuses on structural dependencies, but can be extended to cover behavior. The behavioral dependencies can be used to guide the composition directives.

Design philosophy of the pattern *Full View with Errors* [YoB97] take the philosophy of showing the user everything, then checking each action. The opposite approach is *Limited View* [YoB97] which only shows the user what he can do. These are two different philosophies on how to choose what to make an aspect and how to combine them.

The *Full View with Errors* approach is often best when there are few restrictions on most of the users. The consequences are simpler a UI, because the same user interface implementation can be used for everyone. However, in the controller section of a Model-View-Controller pattern there would be an access

check on every action. This pattern requires a supporting error reporting mechanism.

The *Limited View* creates a custom interface for each user at start-up. The access checks are done once. The user is faced with a specific task-oriented interface to the application, which can limit errors. This can be confusing to users if they use the application in different roles.

Session object proliferation can be controlled by aspects in an easy manner. The session object itself is probably an instance of the singleton pattern. One needs to be careful about which values can be set and the integrity of the session object.

A principle that should be followed as the development progresses is that of secure failure. If the system fails, then it should fail in a manner that does not violate any of the security requirements or the security policy.

7 Requirements for an Aspect Repository

A repository for models that supports AOM will maintain the consistency of the models and the relationships between them. The primary relationship from an AOM perspective is the mapping of the aspect model onto the primary model. The usefulness of the repository is greatly enhanced if other types of relationships between models are supported. From the perspective of this research maintaining **dependency** and **refines** kinds of relationships will help produce higher quality solutions.

Since the repository also maintains the composition directives between models, it also needs to represent and maintain the critical sections. A critical section of a model is a place where compositions instructions are prevented or severely limited. The definition of critical sections might be distributed across models and composition directives. For example, auditing actions from an auditing aspect should be disallowed in some places in the encryption aspect as that would defeat the goal of confidentiality.

The critical sections in a model imply that a simplistic name based AOM composition approach is inadequate in the general case.

8 Related Work

Some of the earliest work on security patterns was done by Joseph Yoder and Jeffery Barcalow [YoB97]. This work predates the AOM approach and are patterns that are close to the code level. Several of the patterns that they developed can be converted to aspects, which makes them applicable at higher levels of abstraction.

Security patterns as well as the concept of a security pattern system is discussed

by Schumacher [ScR01]. The security pattern system considers the set of security patterns that work together. He also describes a template for expressing a security pattern, but does not go as far as using UML or AOM. The work here can be extended to models in UML and expressed as aspects.

Most of the work in security patterns to date is often expressed in natural language. There are a few results of recent research that use UML, both class and interaction models, such as the Open Group [Ope04] and Priebe [PrF+04]. When UML is used it is often in a descriptive context of a stand alone design, rather than as an aspect.

Related work often takes the approach of creating a security framework with pluggable components [YoB97]. At first glance this might seem like a reasonable approach, but it is static in nature. Experience has shown that the algorithm for putting the components together is nearly always custom. Aspects through the composition directives keep the assembly algorithm more explicit and avoid the unnecessary framework code.

The AOM community often uses security examples for explaining the concepts behind the AOM approach. However, the security examples are a means to an education end.

9 Conclusion

This paper has shown how a set of security concerns can be represented as aspects. This representation in a pattern creates a basis for collecting and re-using the security knowledge by a less experienced software engineer. The first order dependencies between the security aspects were also developed.

9.1 Future Work

How are exceptions handled in aspects? Especially in the case where additional logging is needed for security audit. There is also the case where too many exceptions, like failed log-in attempts result in other actions like locking an account. These cases are part of the security policy. How much of the security policy should be represented in the model? The security policy will require functionality to implement the constraints and consequences.

The full consequences of all the dependencies between the various security concerns have not been fully explored. The behavior interactions need deeper study as flaws in behavior are a significant source of security holes.

The resolution of conflicting requirements such as auditing versus privacy need further research. Improper auditing can defeat the requirement of privacy. The understanding of when each has priority as well as poor application of the aspects is not well defined.

The aspects in this paper have been studied at a high level of abstraction.

Further refinement and mapping onto technologies is needed. The important questions in this exercise are related to discovering emergent properties of a real implementation. The development of code related to the design should not introduce weaknesses such as buffer overflow.

Each of the security aspects can be related to an attack tree and a threat tree. The interactions of the trees of various aspects are of interest. The composition of those trees and the use of the composition results by analysis tools to evaluate a solution is a future validation approach.

The development of these informal specifications could lead to a more formal specification and evaluation of the individual solutions and the composed solutions.

References

[Amo03] Alfred Amos, “Designing Security into Software with Patterns”, *SANS Institute*, 2003.

[App00] Brad Appleton, *Patterns and Software: Essential Concepts and Terminology*, http://www.cmcrossroads.com/bradapp/docs/patterns-intro.html, Feb. 2000.

[Asj04] AspectJ definition, http://eclipse.org/aspectj/.

[ClH+99] S. Clarke, W. Harrison, H. Ossher and P. Tarr, “Subject-Oriented Design: Towards Improved Alignment of Requirements, Design and Code”, *Proceedings of Object-Oriented Programming Systems, Languages and Applications (OOPSLA)*, 1999.

[CHO+99] S. Clarke, W. Harrison, H. Ossher and P. Tarr, “Separating Objectives Throughout the Development Lifecycle”, *Proceedings of the 3rd ECOOP Aspect-Oriented Programming Workshop*, Springer, 1999.

[DeS00] Premkumar T. Devanbu and Stuart Stubblebine, “Software Engineering for Security: a Roadmap”, *International Conference on Software Engineering (ICSE 200)*, 2000.

[FeP01] Eduardo B. Fernandez and Rouyi Pan, “A pattern language for security models”, *PLoP Conference*, 2001.

[Fer01] Eduardo B. Fernandez, *Patterns for Secure System Design*, 2001 http://www.cse.fau.edu/~ed

[FrR+04] R. B. France, I. Ray, G. Georg and S. Ghosh, “An Aspect-Oriented Approach to Design Modeling”, to be published in *IEE Proceedings - Software, Special Issue on Early Aspects: Aspect-Oriented Requirements Engineering and Architecture Design*, 2004.

[GaH+95] E. Gamma, R. Helm, R. Johnson and J. Vlissides, *Design Patterns: Elements of Reusable Object-Oriented Software*, Prentice Hall, 1995.

[GeR+02] Geri Georg, Indrakshi Ray and Robert France, “Using Aspects to Design a Secure System”, *Proceedings of the International Conference on Engineering Complex Computing Systems (ICECCS 2002)*, Greenbelt, MD, ACM Press. December 2002.

[Kar04] Maxim V. Karpov, “Code Access Security and Design Patterns”, *Microsoft .NET Best Practices*, May 2004.

[Ope04] Bob Blackley, Craig Heath and members of the Open Group Security Forum, *Security Design Patterns*, 2004 http://www.opengroup.org/security/gsp.htm

[Pet01] Razvan Peteanu, “Design Patterns for Security”, *SecurityPortal*, June 2001.

[PrF+04] Torsten Priebe, Eduardo B. Fernandez, Jens I Mehlau and Günther Permul, “A Pattern System for Access Control”, early draft.

[RFL03] I. Ray, R.B. France, N. Li, G. Georg, “An Aspect-Based Approach to Modeling Access Control Concerns”, 2003.

[RaM+03] Awais Rashid, Ana Moreira and João Araújo, “Modularisation and composition of aspectual requirements”, *Proceedings of the 2nd International Conference on Aspect-Oriented Software Development (AOSD 2003)*, 2003.

[Rod03] Ed Rodriguez, “Security Design Patterns”, *Annual Computer Security Application Conference (ACSAC '03)*, 2003.

[Rom01] Sasha Romanosky, *Security Design Patterns: Part 1*, version 1.4, 2001.

[Rom02] Sasha Romanosky, *Enterprise Security Patterns*, June2001.

[SaB97] Ravi Sandhu and Venkata Bhamidipati, “The URA97 Model for Role-Based User-Role Assignment”, *Proceedings of IFIP WG 11.3 Workshop on database Security*, Lake Tahoe, California, Aug 11-13 1997

[Sch00] Bruce Schneier, *Secrets and Lies: Digital Security in a Networked World*, Wiley 2000, 0-471-25311-1.

[Sch03] Bruce Schneier, *Beyond Fear: Thinking sensibly about security in an uncertain world*, Copernicus Books 2003, 0-387-02620-7.

[Sec04] Security patterns web site for interested parties http://www.securitypatterns.org/

[Ses03] Roger Sessions, *Software Fortresses: Modeling Enterprise Architectures*, Addison-Wesley 2003, 0-321-16608-6.

[Smi00] Rick Smith, *Authentication, Patterns of Trust*, Information Security,

Aug 2000.

[ScR01] Markus Schumacher and Utz Roedig, “Security Engineering with Patterns”, *PLoP Conference*, 2001.

[Shr02] Doshi Shreyas, *Software Engineering for Security: Towards Architecting Secure Software*, Information and Computer Science Department, University of California, Irvine, 2002.

[Uml04] Unified Modeling Language 2.0, Object Management Group, http://www.uml.org/#UML2.0.

[YoB97] Joseph Yoder and Jeffery Barcalow, “Architectural Patterns for Enabling Application Security”, *PLoP Conference*, 1997.