Modeling Architectural Access Control with UML 2.0

Azadeh Nematzadeh, Vahid Kazempour, Pooya Jaferian,
Pejman Salehi, Ahmad Abdollahzadeh
Amirkabir University of Technology, Tehran, Iran
{a-nematzade, kazempour, jaferian, psalehi, ahmad}@ce.aut.ac.ir

Abstract. In today's world, security is one of the most important quality attributes in software architecture. Previous modeling approaches provide insufficient support for an in-depth treatment of security. They lack the ability to model important security concepts. This paper presents a more comprehensive treatment of an important security aspect, access control, at the architecture level. Our approach models security user, role, permission, two many-to-many relations for assigning a role to user and permission to role, and policy of architectural constituents. We base our new modeling language on UML 2.0 and try to model security concepts by extending UML. We also provide mechanisms for checking UML models to detect architectural vulnerabilities and assure correct access control at an early design stage.

1. Introduction

Security means correct behavior in face of an intelligent adversary or adversaries [1]. Security is a property of a computer system that prevents unauthorized access to resources, modification of information and data, and use of resources. Application security refers to security controls and mechanisms that are included within a software system. Three important properties in secure systems include confidentiality, integrity, and availability. Each secure system can be called secure in three mentioned viewpoints.

In software engineering, security is considered a non-functional requirement [2],[3],[4]. Therefore, it should be considered in all phases of software development. Currently, security is an afterthought and developers usually attempt to remove the vulnerable points, after attacks against the system. This method is called “penetrate and patch” [5] and it usually imposes heavy costs and defects on software projects.

The first phase in software development in which non-functional requirements and specifically security should be addressed is building software architecture [6],[7]. The software architecture of a program or computing system is the structure or structures of the system, which comprise software elements, the externally visible properties of those elements, and the relationships among them [8]. Software Architecture consists of early design decisions and is a basis for design, implementation, and deployment of a system. Thus, in order to address and evaluate security from early stages of software development, security requirements should be modeled at architecture level. To achieve security, there exist various mechanisms that control interactions between
components of the system to prevent unauthorized access to system resources and services. At architecture level, these mechanisms are used as architectural tactics [8]. One of the most important tactics to be considered at architecture level is access control, which controls how protected computational resource can be accessed, is arguably the most dominant security assurance mechanism.

In this paper we model Role Based Access Control (RBAC) [9] at architecture level. Our approach brings deeper and more comprehensive modeling of architectural security, and helps software architects detect architectural vulnerabilities and support correct access control at the architecture level. While our approach cannot fully solve the general software security problem, it can complement and possibly guide other solutions that operate on the mathematical properties and low-level implementations to collectively provide the comprehensive solution that is necessary for a complex, componentized, and networked software system.

In this work we have chosen Unified Modeling Language (UML) as our base modeling language and extend it to model important security abstractions. We believe that UML is one of the most complete and flexible modeling languages. In contrast with formal languages, UML is easy to understand and use. It is also accepted as an industrial modeling language between software developers and lots of tools exist to build UML models.

The rest of this paper is organized as follows. Section 2 surveys related work. Section 3 outlines our approach, introducing security abstractions to be modeled by UML and the modeling extensions necessary for security development. Section 4 gives an example of applying the approach to a case study system. The paper is concluded with Section 5 which contains brief recapitulation of the main points and further works.

2. Related Work

In this section, we overview existing attempts and works in the field of secure software engineering, especially secure software architecture.

Jürjens, in [10] states “the relationship between nonfunctional requirements and software architectures is only very poorly understood”, therefore he suggests improvements to this situation by integrating security requirements analysis with a standard development process, and modeling of security related features such as confidentiality, access control, etc. through an extension to the UML called UMLsec.

Schneider [11] defines different architectures (e.g., security, system, and software architectures) that together form an overall architecture that satisfies the constraints of each of its constituent architectures. Schneider asserts that a system’s design is the product of these architectures. It is based on concepts from the Defense Goal Security Architecture (DGSA), which is an architectural framework in which system architects define security according to the requirements to protect information [11], [12]. In other words, the basis for such architecture is independent of everything but the information one must protect.
Bidan and Issarny in [13] demonstrate how the software architecture paradigm is beneficial for addressing security issues in distributed systems through system customization. The software architecture paradigm allows the application developer to abstractly specify security-related requirements. Then, this framework takes in charge the system customization to meet these requirements. The practical use of this approach is also addressed by discussing its integration in a configuration-based distributed programming environment.

In [14] the authors present a structured and flexible way for describing security system architectures using the Software Architecture Model (SAM). They also introduce the concept of security constraint patterns, which provides a generic form to formally specify security policies that the security system must enforce. They present a technique to decompose system-wide constraint patterns onto individual components of the system based on the security architecture model and to verify the consistency between global and component constraints. These constraint patterns define what conditions or properties each component and their composition must satisfy under the system architecture. In concert with the architecture model and constraint propagation, they present a flexible and scalable technique to verify whether the security system architecture satisfies the required security constraints. They integrate the above aspects into a systematic and incremental process of security system architecture modeling and verification.

Ren and Tylor specify a model [15], [16] that centered on software connectors and provides a suitable vehicle to model, capture, and enforce access control. They addressed security problem from an architectural viewpoint that can be used during design and analysis of secure software systems. They also provide a secure software architecture description language for describing architectural access control. This language enables specifying security contracts of components and connectors.

3. Our Approach to Secure Architecture

This section details the elements of the security modeling approach we are taking. We first give an overview on extension mechanisms of UML and discuss suitability of them for modeling security abstractions. Then we propose a method for modeling architectural components and connectors with UML. After that we outline the new modeling capabilities we proposed to help assuring correct architectural access control.

UML Extension Mechanisms

UML does not support modeling of important security abstractions. As a result, we need to extend UML in well-defined way in order to capture required modeling concerns. UML provides four important extension mechanisms that allow designers to customize and extend the semantic of model elements. Constraints place added semantic restrictions on model elements. Tagged Values allow attributes to be associated with model elements. Stereotypes allow grouping of constraints and tagged values and assigning them a descriptive name to create a new form of meta-class for
models. Profiles are predefined sets of stereotypes, tagged values, constraints, and icons to support modeling in specific domains [17].

Modeling Architectural Security

We extend UML to model software security, focusing on architectural access control. Access control is one of the most common types of security model. Security is usually defined by security model and security policy. Security policy captures the security requirement of an enterprise or describes the steps that have to be taken to achieve security. It discriminates authorized and unauthorized as considered in a secure system. Security model is an abstraction of security policy. It identifies the relation among the subjects and objects of a system in a formal manner. This relationship specifies the access rights the subject has over system’s resources.

Access Control

Access control, which controls how a protected computational resource can be accessed, is arguably the most dominant security assurance mechanism. In the classic access control model [18],[19] a system contains a set of subjects that has permissions and a set of objects (also called resources) on which these permissions can be exercised. An access matrix specifies what permission a subject has on a particular object. The rows of the matrix correspond to the subjects, the columns correspond to the objects, and each cell lists the allowed permissions that the subject has over the object. The access matrix can be implemented directly, resulting in an authorization table. More commonly, it is implemented as an access control list (ACL), where the matrix is stored by column, and each object has one column that specifies permissions each subject possesses over the object.

Also, there are several models that control access according to information flow between different security levels of an organization. Such models usually address one of the security needs, for instance the BLP model [20] consider confidentiality and the Biba [21] consider integrity.

More recent models, such as the role based access control (RBAC) model [9] are often based on employee functions rather than data ownership. In these models, access control decisions are often determined by the roles that individual users take as part of organization. A recent study by NIST demonstrates that RBAC addresses many needs of the commercial and government sectors. A major purpose of RBAC is to facilitate security administration and review. RBAC introduces the concept of roles as an indirection to organize the permissions assignments to subjects. Instead of assigning permissions directly to subjects, the permissions are assigned to roles and users are made members of appropriate roles. This greatly simplifies management of permissions.
User, Roles, Permission and their relation

We introduce following core components that are necessary to model access control at architecture level: User, Role, Permission, and Security Policy. We need to introduce two many-to-many relations, first for assigning a role to user and second for assigning permission to role.

Users are assigned roles based on their responsibilities and qualifications. Users can be easily reassigned from one role to another. The users can not pass access permission on to other users at their discretion.

Roles are created for the various job functions in an organization. Roles can be granted new permissions as new applications and systems are incorporated, and permissions can be revoked from roles as needed. A role is properly viewed as a semantic construct around which access control policy is formulated. The particular collection of users and permissions brought together by a role is transitory. The role is more stable because an organization's activities or functions usually change less frequently.

A permission is an approval of a particular mode of access to one or more objects in the system. Permission confers the ability to the holder of the permission to perform some action(s) in the system. Permissions to modify the sets U, R, and P and relations PA and UA are called administrative permissions.

User to role assignment (UA relation) and permission to role assignment (PA relation) are both a many to many relation. The model expected that each role to be assigned to at least one permission and each user to be assigned to at least one role. The key to access control lies in these two relations. Ultimately, it is a user to exercises permissions. The placement of a role as an intermediary to enable a user to exercise a permission provides much greater control over access configuration and review than does directly relating user to permission.

Policy is a powerful mechanism for laying out higher-level organizational Constraints. Policy can apply to the UA and PA relations. Policy are predicates which, applied to these relations, return a value of “acceptable” or “not acceptable.” Access control policy is embodied in various components of RBAC such as role-permission, user-role relation. These components collectively determine whether a particular user will be allowed to access a particular piece of data in the system. There are several ways to describe and model policies; one way is using formal languages. We use OCL as a basis architectural security policy modeling.

Modeling Secure Architectural Components and Connectors with UML

There are different approaches available to model software architecture views such as components and connectors [8]. In this paper we model components and connectors view with UML 2.0.

Some works has been done to model component and connector view with UML. In [22], two strategies for supporting architectural concerns within UML are presented. One strategy involves using UML “as is,” while the other incorporates useful features of existing ADLs as UML extensions. But the strategies are customized for a specific ADL (C2) architectural style. In [23], some strategies are presented to model
components and connectors with UML 2.0 and strengths and weaknesses of each approach is discussed.

In our work, we model architectural components with UML component, and architectural connectors with UML classes. Also to document connector’s properties we extend UML meta-model to allow new semantics (properties) to be incorporated into UML classes.

In order to model our architecture based on security issues we use component and connector view which show the system as a set of cooperating units of runtime behavior. The widespread use of UML has lead practitioners to use it for documenting software architecture. There are some reasons which made UML appropriate for software architecture which first mentioned by Garlan and associates [24]. These reasons are as follows:

- Semantic match: The UML constructs should map intuitively to the architectural features being documented.
- Visual clarity: The UML description should bring conceptual clarity to a system design, avoid visual clutter, and highlight key design details.
- Completeness: All relevant architectural features for the design should be represented in the UML model. The ability to capture this information is also referred to as expressiveness.

Also there is another reason for using UML as a basis for documenting software architecture which motioned in [23]:

- Tool support: Not all uses of UML are supported equally by all tools (particularly when specializing UML), which can limit documentation options.

Unfortunately UML 2.0 has not a built-in model for documenting software architecture despite it’s improvements in compare with UML 1.4. Therefore, we must use UML extension mechanisms to extend it for architectural documentations especially component and connector view with respect to security considerations. In [23] they introduce some new ways to document building blocks of C&C view and they mention alternatives for component, connector, and port.

In this paper we model access control at architectural level with respect to their guidelines and especially focus on architectural connector which is the main part to enforce security strategies. In the following part of this section we demonstrate our viewpoints on documenting building blocks of C&C view.

**Documenting Component**

The UML 2.0 changes improve the suitability of a UML class for representing a C&C component also apply to UML components because UML components were changed to be a subtype of classes in the meta-model. UML components have an expressive power and can be used to represent C&C components.
Documenting Ports

Documenting C&C ports is one area in which the changes in UML 2.0 really shine. The newly introduced port concept in UML 2.0 is so well suited for documenting C&C ports that we no longer consider any other strategies.

Documenting Connector

While the changes in UML 2.0 improve its suitability for documenting components and ports, similar improvements supporting C&C connectors are missing. The UML 2.0 connector concept, which is new, is too lacking in expressiveness to be a good solution for documenting C&C connectors. In particular, it lacks any ability to associate semantic information with a connector (e.g., a behavioral description) or to clearly document C&C connector roles. There are several approaches to model connector with UML 2.0: using UML associations, using UML association classes, and using UML classes we use classes in order to model our connectors due to importance of connectors in security. We apply our security strategies mainly in connector’s . We extend UML 2.0 meta-model and construct a meta-class named connector.

![Fig. 1. Connector meta-class](image-url)

In our approach, each component is running on behalf of a User. It sends its message to the connected connector without considering the destination to which the message is routed. Connectors play an important role in our approach. It will perform all access control activities and decide on whether the message should be routed to the destination or not. To achieve this goal we should model the mentioned security abstractions such as User, Role, Permission, and Policy at connector level.

To embed security abstractions in connectors, we introduce them as properties. Therefore, we extend UML meta-model by inheriting a meta-class named <<secure-connector>> from <<connector>> meta-class. We also define four UML tags with the type of set for Users, Roles-Users, and Permissions-Roles, and Valid-Permissions. The Users set defines the list of users that may interact using the connector. The Roles-Users set contains ordered pair of Users and their assigned role. The Permissions-Roles set contains ordered pair of Roles and their assigned Permissions. The Valid-Permissions set contains those permissions for which the connector will route the message. When modeling software architecture with secure-connectors, we can assign value to the mentioned tags as tagged-values. We should also define policy for connector to route a message. We specify secure-connector’s policy using OCL. The default policy is that the message will be routed if it came from a user with a valid permission. The default policy can be override by extending <<secure-connector>> meta-class and build a specific secure-connector. We show the extended meta-class in figure 2.
A Case Study

Architectural modeling is instrumental for architects to design architecture and evaluate different alternatives for possibly competing goals. With the modeling capability introduced by our approach, architects are better equipped for such design and analysis on security. In this section, we illustrate the use of the secure software architecture modeling technique with a message routing system.

The message routing system allows two parties to share data with each other. However, these two parties do not necessarily fully trust each other, thus the data shared should be subjective to the control of each party. The two parties participating in this application are company A and company B.

The Original Architecture

Figure 3 illustrates the original message routing system’s architecture, using components and connectors. In this architecture, Company A and B each has its own process. A is on the left side, and B is on the right. The squares are components. The regular rectangles are connectors. The A Receiver Filter Connector sends all messages downward. The A to A Filter Component forwards all such notifications to the A Filter and Command & Control Connector. However, A does not want B to
receive all the notifications. Thus it employs an A to B Filter Component to filter out sensitive messages, and send those safe messages through A Distributed Fred Connector, which connects to the B’s Local Fred Connector to deliver those safe messages. (A Fred connector broadcast messages to all Fred connectors in the same connectors group.) The B side essentially has the same architecture, using a B to A Filter Component to filter out sensitive messages and send out safe messages. The advantage of this architecture is that it maintains a clear trust boundary between A and B. Since only the A to B Filter and the B to A Filter come across trust boundaries, they should be the focus of further security inspection. This architecture does have several shortcomings. First, it is rather complex, this architecture uses 4 Fred connectors (A’s Local, A’s Distributed, B’s Local, and B’s Distributed) and 2 components (A to B Filter, B to A Filter) to implement secure data routing such that sensitive data only goes to appropriate receivers. Second, it lacks conceptual integrity. It essentially uses filter components to perform data routing, which is a job more suitable for connectors. Third, it lacks reusability, since each filter component has its own internal logic, and they must be implemented separately.

An Improved Architecture using our new approach

As depicted in figure 4 an alternative architecture uses two secure connectors, a A to B Connector and a B to A Connector. Both are based on the same class. The A to B Secure Connector connects to both the A Filter and Command & Control Connector
and the B Filter and Command & Control Connector. When it receives data from the A Radar Filter Connector, it always route it to the A Filter and Command & Control Connector. And if it detects that it is also connected to the B Filter and Command & Control Connector, and the data is releasable to the B side, then it also routes messages to the B Filter and Command & Control Connector. The B to A Secure Connector adopts the same logic. This architecture simplifies the complexity and promotes understanding and reuse. Only two secure connectors are used. These connectors perform a single task of secure message routing, and they can be used in other cases by adopting a different policy. A shortcoming of this architecture is that the secure connectors can see all traffic, thus they are obvious targets for penetration, and their breach leads to secret leak. An architect should balance all such tradeoffs.

Fig. 4. Improved architecture of message routing system using our new approach

**Conclusion**

Traditional security research has not fully addressed new challenges presented in component-based software operating in a modern networked environment. Recent advancement on software architecture shed light on high-level structure and communication issues, but has paid insufficient attention to security aspect.

In this paper, we argue that architectural access control is necessary to advance existing knowledge and meet the new challenges. We extend UML 2.0 to model
software architecture and also specify core security concepts in RBAC access control: users, roles, permissions, and policy. Component compositions are handled by connectors, which regulate the desired access control property. Our extended modeling language can describe the security properties of software architecture, specify intended access control policy, and facilitate security design and analysis at the architecture level. We illustrate our approach through an application sharing data among two companies, demonstrating how architectural access control can be described and enforced.

The contributions of this research lie in that 1) we address the security problem from an architectural viewpoint. Our use of an architecture model can guide the design and analysis of secure software systems and help security assurance from an early development stage; 2) we provide an extension on UML 2.0 for describing architectural access control, arguably the most important aspect of security; 3) the extended UML enables specifying security aspects of components and connectors, laying the foundations for secure composition and operation.

This research is still on-going work. Our future work includes: 1) developing an algorithm to check whether the architecture meets the access control policies specified in various architectural constituents or not; 2) Developing some add-ons for well-known UML modeling tools to support modeling and checking our architectural models.

References