Engineering Trust Management into Software Models

Mark Reith,[*] Jianwei Niu, William H. Winsborough
University of Texas at San Antonio
One UTSA Circle
San Antonio, Texas, USA 78249
{*mreith, niu*}*@cs.utsa.edu, wwinsborough@acm.org*

Abstract

Security in software is often considered a nonfunctional requirement because it is often interpreted as an emergent feature of the system. Too often it is introduced as a last-minute requirement over an otherwise completed product rather than properly integrated during the early stages of software design and development. One significant aspect of security involves access control. This paper proposes a multi-layer model detailing the integration of trust management access control with an application's model behavior. Our previous work focused on modeling the dynamic changes of a trust management policy for the purpose of verifying security properties using model checking. We are working toward integrating both the trust management policy and the mechanisms that enforce that policy for the purpose of verifying security properties. We focus on the Role-based Trust Management (RT) language and suggest concerns specific to it.

1. Introduction

Software plays essential roles in our lives and is prevalent in financial, transportation and telecommunication systems. Software security is a critical aspect of these systems because it prevents a subset of software failures due to both accidental and malicious incorrect operation, allowing these systems to function correctly and deliver reliable service. Common security technologies such as cryptographical protocols, firewalls, and antivirus software attempt to shape the environment in which software operates, but have not always been successful at preventing the exploitation of software. While these technologies do provide some measure of protection, they have not always proven effective because they are primarily used to protect software at the communication and system level rather than at the application software level [20].

Application security focuses on preventing the exploitation of software by constraining application behavior rather than the environment in which it executes. One key area of application security involves controlling access of software features. Various access control models have been studied including access control matrices and lists [8], role-based [16, 17], and logic-based [21]. We are primarily interested in such models that are expressed through policies. Policies provide a means of configuring access control without recompiling the executable. Instead of implementing the access control details such as users and permissions directly into the software, policies provide the customizable and modifiable logic that determines access and then passes the decision results to mechanisms that are implemented in the software. However, problems arise when policy and mechanisms are not matched properly. We suggest state machine based modeling as a means of addressing this issue.

A significant concern for software developers is how to incorporate access control into software such that it is configurable through policies, and then verify that security properties in the software/policy system still hold. We outline a framework for reasoning about configurable policies, security enforcement mechanisms, and access control security requirements in a multi-layer model. The multi-layer approach attempts to organize a complex security design into simpler layers that can be evaluated against security requirements. It seems reasonable that some requirements might be satisfied within a layer, while other requirements may be satisfied only through a combination of layers. We are interested in verifying properties both within a layer and across multiple layers. The layered approach is inspired from the observation that policy usage may satisfy some, but not necessarily all, security requirements. Consequently, those unsatisfied requirements must be addressed by some other means, ideally in a separate layer. The goal of the proposed approach is to scope these layers to a degree of complexity that is appropriate for automated tools such

[*] The views expressed in this article are those of the author and do not reflect the official policy or position of the United States Air Force, Department of Defense, or the U.S. Government.

International Workshop on Modeling in Software Engineering (MISE'07)
0-7695-2953-4/07 $20.00 © 2007 IEEE

as model checkers.

In previous work [15], we demonstrated the feasibility of using automated analysis tools such as model checkers to verify security properties in RT policies. Two main contributions came from this work. First, it provided a relatively quick means to find counterexamples to certain expensive types of analysis when properties fail to hold. Second, it provided software engineers a means of analyzing policies with well understood and automated tools. Such a technique is only the beginning of a larger effort to model and analyze security properties of not only the policy, but the associated application as well. Doing so provides more confidence in the security attributes of the application from a holistic point of view.

The structure of this paper is as follows. Section 2 briefly describes trust management, a specific trust management language called RT, and our previous work on analyzing security properties of this language. Section 3 outlines the proposed layered framework for modeling policy and enforcement points, and how the two models shall relate. Section 4 presents related work on engineering secure software systems. Finally, Section 5 provides a discussion of future work.

2. Trust Management

Trust management [4, 12] can be described as a type of access control where access decisions are determined by a set of entities represented as a set of credentials. Each entity may delegate a subset of permissions they posses using digitally signed credentials as documentation. Such trust management systems are useful in decentralized environments where scalability of access control is desirable. Due to their distributed nature, each relevant credential is first consolidated into a collection that is then used to prove an access request. Over time, resource owners and intermediaries may add, modify or revoke credentials and thus dynamically change access control. Although there exists several TM languages such as PolicyMaker [4], KeyNote [3] and Cassandra [2], we focus our attention on Role-based Trust Management (RT) since it already provides a strong theoretical foundation for security property analysis.

2.1. The RT Language

The role-based trust management policy language RT was designed to support highly decentralized attribute-based access control [12]. It enables resource providers to make authorization decisions about resource requesters of whom they have no prior knowledge. This is achieved by delegating authority for characterizing principals in the system to other entities that are in a better position to provide the characterization. For instance, to grant discounted service to students, a resource provider might delegate to universities the authority to identify students and delegate to accrediting boards the authority to identify universities. A full treatment of the RT language can be found in [13].

Type	Syntax	Description
Type I	$A.r \leftarrow D$	Simple Member
Type II	$A.r \leftarrow B.r_1$	Simple Inclusion
Type III	$A.r \leftarrow B.r_1.r_2$	Linking Inclusion
Type IV	$A.r \leftarrow B.r_1 \cap C.r_2$	Intersection Inclusion

Figure 1. RT Statements

The RT language consists of two primary objects called roles and principals. A principal is an entity such as a person or software agent. Each role can be described as a set of principals and is of the form "principal.role_name". One interpretation of this role is that the principal considers the members (also principals) of this role to have an attribute denoted by the role name. For example, *Alice.friend* may be a role that contains the principals whom Alice considers friends.

The basic RT language consists of four types of statements as shown by Figure 1 [13]. Type I statements directly introduce individual principals to roles. For example, *Alice.friend ← Bob* identifies Bob as a friend of Alice. A given principal must appear in a Type I statement if it is to be contained by any role. Type II statements express a form of delegation that describes the implication that if principals are in one role, then they are in another role as well. For example, the statement *Alice.friend ← Bob.friend* describes the situation in which if a principal is a friend of Bob, then they are also a friend of Alice. Type III statements provide a mechanism to delegate to all members of a role. For example, the statement *Alice.friend ← Bob.friend.friend* says that any friend of Bob's friends is also a friend of Alice. It does not imply that Alice's friends include Bob's friends. Finally, Type IV statements introduce intersection such that a principal must be in two roles in order to be included. For example, *Alice.friend ← Bob.friend ∩ Carl.friend* says that only those principals who are both Bob's friends and Carl's friends are introduced into the set of Alice's friends. Note that disjunction is provided through multiple statements defining the same role. Each delegation statement may also include an optional conditional statement that determines if the delegation occurs. For example, the expression *Alice.friend ← Bob.friend [x]* could be interpreted as every friend of Bob is also a friend of Alice if and only if condition *x* is satisfied. Conditional statements may utilize local variables from a trust management application.

2.2. Model Checking RT Security Properties

Of particular interest to policy designers is the ability to reason about access control despite the changes that are

0-7695-2953-4/07 $20.00 © 2007 IEEE

made to the policy. If we define the state of the policy as a specific collection of credentials, then this state changes as credentials are added and removed. Given a set of policy authors who are identified as competent and trustworthy, we can assume for the purpose of analysis that the addition and removal of certain credentials is restricted and thus we may evaluate the policy state for certain properties. For example, is there a reachable state where Alice can be denied rightful access due to policy changes made by an untrusted principal? This question of availability is closely related to the safety question. Does there exist a reachable state where an untrusted principal gains access? In our previous work [15], we developed an automated approach to reason about such properties. We translated RT policies into the input language of model checker SMV to perform such security analysis. Having established a foothold on techniques for policy analysis, we now turn our attention to a larger framework that includes the application associated with the policy.

3. A Layered Framework for Modeling Software & Security Policies

We propose a framework for describing software that utilizes trust management technology such as RT. This software comprises an application and an associated security policy such that the application bases access control decisions on this policy. We define security policy as a given set of RT statements. As such, we expect it to be dynamic in the sense that it is allowed to change. By contrast, we define security requirements as invariant access control properties that must hold in all access control decisions under all policies. Such requirements may be addressed by the policy or by constraints hard-coded into the application.

The purpose of this framework is twofold. First, we suggest one approach for incorporating trust management technology into applications. We suspect that a significant percentage of security defects in software are a result of improper incorporation of security technologies into software. Thus it is relevant to examine how policies are incorporated into applications. Second, we wish to reason about security properties related to access control. We illustrate our framework in Figure 2 as three layers: the application behavior layer, the enforcement layer, and policy layer. Each layer has its own form of state information and thus each may be modeled. Software can then be described in terms of these layers. By separating the application, enforcement, and policy concerns into different layers, we hope to limit the scope of the model we need to consider during the verification of security properties. We also consider the need to verify security properties across layers, in which case a well defined mapping between layers is necessary. Verifying properties across layers provides a holistic perspective of security.

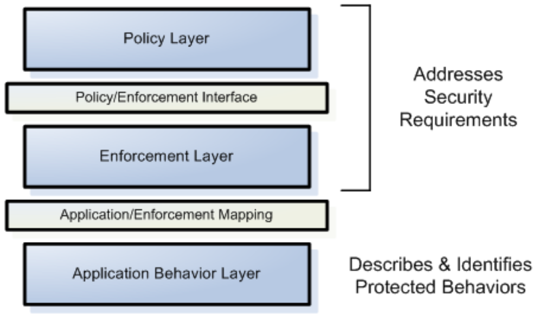
Policy Layer
Policy/Enforcement Interface
Enforcement Layer
Application/Enforcement Mapping
Application Behavior Layer
Addresses Security Requirements
Describes & Identifies Protected Behaviors

Figure 2. Application, Enforcement, and Policy Layers

3.1. Modeling the Application Behavior Layer

The application layer describes the behavior of the base software without access control concerns. Here a behavior is defined as a sequence of transitions on a state transition diagram. We make no assumptions about the level of abstraction used to define the diagram, however we do recognize that behaviors can often be expressed as a composition of other behaviors. Our interest at this layer is to identify and label those behaviors, such as W, V, and X, that require protection from unauthorized access, as in the example in Figure 3.

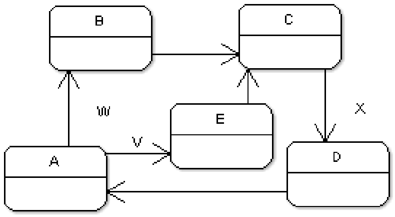

Figure 3. Example of Application Behavior

3.2. Modeling the Policy Layer

The policy layer describes structure and security properties of policies. Structure refers to how policies are constructed as well as how they change. Section 2 provided a description of both the structure and security properties of the RT language. We suggest role dependency graphs as a model for our policy because it shows the nature of the delegation chain and how users/principals are introduced into the policy. A role dependency graph (RDG) [15] is a tool for visually depicting and analyzing role-to-role and role-to-principal relationships. It is a directed graph where each node represents a role, a linked role, the conjunction of two roles, or a principal. Each edge represents a specific policy statement and is labeled by its statement index. An edge is understood to mean that the source node is dependent on the destination node. We add to this RDG two additional notations, as illustrated in Figure 4. First, the expression $role : behavior$ in a node represents a binding of a role to the ability to access a behavior. Second, we label edges with conditions that must hold for the delegation to occur. Such condition statements may rely on application state information. In this example of a policy model, the behaviors W and V represent transitions in Figure 3, and as such Eve has no access to any behaviors, while $Alice$ has access to behavior W if both conditions hold.

The properties we might consider verifying include availability and safety concerns as described in Section 2.2. In addition, we may wish to explore the following questions:

1. Can we verify that invariant properties hold regardless of how the dynamic policy changes?

2. How is the policy allowed to change?

3. Does the policy require application state information to determine access?

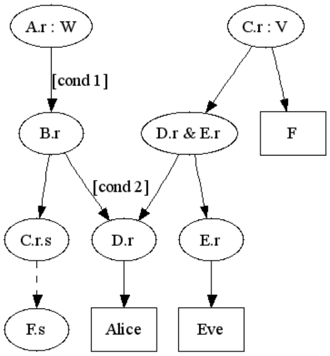

Figure 4. Example Policy Model

3.3. Modeling the Enforcement Layer

The enforcement layer describes the means in which access control decisions are made in the application layer. A security enforcement point (SEP) is a mechanism associated with a particular behavior, and makes a decision whether a principal may access this behavior. It is defined as a structure that consists of a guard component and a state component. The guard component controls a given transition in the application layer, and thus permits or denies application behavior based upon a set of conditions. These conditions may depend on three pieces of information. First, a condition may depend on whether a principal requesting access to a behavior can prove that they are authorized according to the policy. Second, a condition may rely on application state information, such as whether a resource is available. Finally, a condition may depend on state component information from one or more SEP's. The state component can be described as a state machine that captures access history. An example where a SEP might require its own state component information is in the situation where accesses must alternate between two sets of principals. Consider Figure 5 where the principals in role A.r and B.r must take turns accessing a single behavior. In another example, collections of SEP's may rely on each other's state information to make decisions such as in the situation where a SEP permits one behavior only in the case where a different SEP denies another behavior.

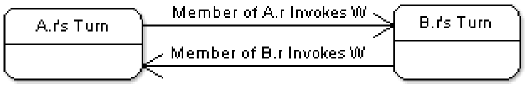

Figure 5. Enforcement Layer: State Component of SEP Associated with Behavior W

This layer is also concerned with placement of enforcement points in such a way as to provide coverage over the protected behaviors. We seek to apply formal methods such as model checking to verify the design of not only the placement of enforcement points, but also the conditions of these points. Our approach is to begin with a notation for describing placement of SEP's. Although a specific notation for modeling SEP's is the subject of future work, we initially suggest the following. Consider a set of behaviors labeled V, W, and X are exhibited by the application in Figure 3. Behaviors V, W, and X are not decomposable into simpler behaviors, however we may define behavior Y as a composition of W and X, and behavior Z as a composition of V and X. In this notation, we use a comma separated sequence to describe behavior composition. For example, $Y : W, X$ expresses behavior Y as an ordered composition of W and X. Such a notation may assist us to decide whether to place a SEP as a guard on behavior Y explicitly, or as a set of SEP's on behaviors W and X. We denote the guard component of the SEP associated with behavior Y as $guard(Y)$. Alternatively we may desire a finer degree of control in the form of $Y : guard(W), guard(X)$. A composite behavior is said to be accessible if all of its sub-behaviors are accessible. With respect to the application layer model, we may verify such properties as:

1. From state A, behavior Y is accessible if and only if the conditions of W and X are satisfied.

2. From state B, behavior X is accessible if and only if the conditions of X are satisfied.

Analysis at the enforcement layer might include verification of invariant properties, and the behavior of individual and collective SEP's. As the middle layer of this framework, it may be necessary to simulate the dynamic policy layer and interpret the effect on the application layer. Such an approach focuses on modeling security from the point of view of SEP's and invariant properties in hopes to answer such questions as:

1. Can we verify invariant properties hold regardless of how the state of the SEP's change?
2. Is there some reachable state of SEP's that may lead to a prohibited application behavior? Can this SEP state be created through dynamic policy?
3. If access to a software behavior is denied by a SEP, will this cause some undesirable effect (inadvertent access, availability violation)?

3.4. Analysis Across Layers

In addition to analysis of each layer, we may also want to explore security properties across layers. Some questions of interest might include:

1. Are changes in the policy accurately reflected in the enforcement layer?
2. Is it ever possible for a policy state to override a constraint in the enforcement layer?
3. Is the policy enforceable, and if so, what security enforcement points are necessary?

A well defined interface may be necessary to relate the state of one layer to the state of another layer. We suspect the mapping between the application layer and the enforcement layer will consist of a many-to-one relationship between application behaviors and enforcement points, since redundant enforcement points may unnecessarily increase the complexity and analysis of the model, not to mention increase the potential for implementation flaws. Note that this mapping is not intended to address security requirements, but rather simply serve as definition of the relationship between the application layer and the enforcement layer.

The interface between the enforcement layer and the policy layer comprises a collection of controllable behaviors and a collection of observable states. Controllable behaviors must exist in the application behavior layer and have a SEP associated with it. They may be bound to one or more roles in a policy. Behavior W is an example that would be included in this collection. The collection of observable states provides application state information used in conditions attached to policy statements in the policy layer.

4. Related Work

The interconnection of software engineering and security engineering gives rise to fascinating research challenges and opportunities. Currently, the software engineering community is working toward incorporating security into software development process [5, 6, 14]. Security requirements are elicited via anticipating threats by examining system-related assets and attacker's goals, so that countermeasures can be derived and documented accordingly [7, 11, 19]. Software modeling notations and tools have been extended to incorporate security concerns so that security properties can be described in the design models [1, 9].

The model driven security approach SecureUML [1] introduces a means of merging a role-based access control (RBAC) security model, which restricts system access based on the role of the user, into a software model for the purpose of automatic security mechanism implementation. In this approach, the security model is described as a meta-model extension of UML and a system architecture along with its access control infrastructure can be automatically generated from the SecureUML models. Our work pursues a related goal of constructing a framework from the software model and trust management (TM) policy models, again for the purpose of verifying security properties. While the specific TM policy language we examine in this paper is role-based, it differs from [1] in two ways. First, TM is inherently described by a security policy, which is a collection of rules describing how the software system may be used. Although closely related, SecureUML does not provide a means of reasoning about how policies change, the impact a policy change has on software, or how incorrectly configured policies affect software behavior. Second, TM involves the extensive use of delegation, which is not addressed in SecureUML or standard RBAC. Our previous work [15] investigated the use of model checking to verify security properties of just the TM security policies. A framework that represents both security mechanisms and dynamic delegation policy may provide a means of demonstrating the robustness of an application's access control.

UMLsec [10, 9] introduces specific stereotypes and tags that can be used to model security concerns using UML to primarily describe security protocols. They have demonstrated the feasibility of model checking a cryptographic protocol described in UMLsec, however, it has not been extended to include access control concerns, which is our interest.

Finally, [18] describes "enforceable security policies" as those policies enforceable by mechanisms that monitor system execution. This is significantly related to our framework as we are interested in determining whether policies are enforceable. This paper suggests that, individually and collectively, access control mechanisms can be used to maintain security properties. However our work is interested in analyzing whether enough software behaviors are controlled by security mechanisms to ensure that security properties are upheld.

5. Discussion & Future Work

As in other engineering disciplines, models built in the early development stages (i.e., requirements and design stages) describe the important functions of software systems by abstracting away nonessential aspects. Models improve the quality of the resulting systems by providing a foundation for early analysis and fault detection [1]. One of the key benefit of modeling software is the ability to disclose in the model subtle errors that would be difficult and expensive to find in an implementation. We contend that modeling and analysis of both security and application behaviors is a necessity in designing and implementing secure and reliable software.

We argue that our layered framework provides significant benefits such as:

1. Multi-level analysis of access control from policy to behavior provides a holistic view of software.

2. Separation of concerns allows independent analysis of each layer.

3. Given a specific policy, we can test the effect on the application behavior.

We suggest several areas for future work. First, a concise notation for describing security enforcement points needs to be developed. While we have suggested using predicates to describe guard components in Section 3.3, we suspect there may be a cleaner notation that includes the collection of conditions that must hold for the guard to permit its associated behavior. Second, the mapping between the enforcement points and application behaviors, as well as the policy/enforcement interface needs to be formalized in order to perform automated analysis across layers. Finally, we need to evaluate this approach in terms of the ability of these models to capture various access control specifications as well as verify access control properties.

References

[1] D. Basin, J. Doser, and T. Lodderstedt. Model driven security: From UML models to access control infrastructures. *ACM TOSEM*, 15(1):39–91, 2006.

[2] M. Y. Becker and P. Sewell. Cassandra: Flexible trust management, applied to electronic health records. In *CSFW*, Washington, DC, USA, 2004. IEEE Computer Society.

[3] M. Blaze, J. Feigenbaum, and A. D. Keromytis. Keynote: Trust management for public-key infrastructures. In *LNCS*. Springer Berlin/Heidelberg, 1998.

[4] M. Blaze, J. Feigenbaum, and J. Lacy. Decentralized trust management. In *SSP*, Washington, DC, USA, 1996. IEEE Computer Society.

[5] P. T. Devanbu and S. G. Stubblebine. Software engineering for security: a roadmap. In *ICSE*, pages 227–239, 2000.

[6] P. Giorgini, F. Massacci, J. Mylopoulos, and N. Zannone. Modeling security requirements through ownership, permission and delegation. In *RE*, pages 167–176. IEEE Computer Society, 2005.

[7] C. B. Haley, R. C. Laney, J. D. Moffett, and B. Nuseibeh. The effect of trust assumptions on the elaboration of security requirements. In *RE*. IEEE Computer Society, 2004.

[8] M. A. Harrison, W. L. Ruzzo, and J. D. Ullman. Protection in operating systems. *Commun. ACM*, 19(8):461–471, 1976.

[9] J. Jurjens. Sound methods and effective tools for model-based security engineering with UML. In *ICSE*, pages 322–331, 2005.

[10] J. Jurjens and P. Shabalin. Automated verification of UMLsec models for security requirements. In *LNCS*, pages 365–379, 2004.

[11] R. D. Landtsheer and A. van Lamsweerde. Reasoning about confidentiality at requirements engineering time. In *ESEC/FSE*, pages 41–49. ACM Press, 2005.

[12] N. Li, J. C. Mitchell, and W. H. Winsborough. Design of a role-based trust management framework. In *SSP*, pages 114–130. IEEE Computer Society Press, May 2002.

[13] N. Li, J. C. Mitchell, and W. H. Winsborough. Beyond proof-of-compliance: Security analysis in trust management. *JACM*, 52(3):474–514, 2005.

[14] G. McGraw. Software security. pages 80–83, 2004.

[15] M. Reith, J. Niu, and W. Winsborough. Apply model checking to security analysis in trust management. 2007. To appear in SECOBAP.

[16] R. Sandhu, V. Bhamidipati, and Q. Munawer. The ARBAC97 model for role-based administration of roles. *ACM Trans. Inf. Syst. Secur.*, 2(1):105–135, 1999.

[17] R. S. Sandhu, E. J. Coyne, H. L. Feinstern, and C. E. Youman. Role-based access control models. *IEEE Computer*, 29(2):38–47, 1996.

[18] F. B. Schneider. Enforceable security policies. *ACM Trans. Inf. Syst. Secur.*, 3(1):30–50, 2000.

[19] A. van Lamsweerde. Elaborating security requirements by construction of intentional antimodels. In *ICSE*, pages 148–157. ACM Press, 2004.

[20] J. M. Wing. A symbiotic relationship between formal methods and security. In *CSDA*, Washington, DC, USA, 1998. IEEE Computer Society.

[21] T. Y. C. Woo and S. S. Lam. Authorizations in distributed systems: A new approach. *Journal of Computer Security*, 2(2-3):107–136, 1993.

International Workshop on Modeling in Software Engineering (MISE'07)
0-7695-2953-4/07 $20.00 © 2007 IEEE