Volume 2, No. 3, May-June 2011
ISSN No. 0976-5697
International Journal of Advanced Research in Computer Science
RESEARCH PAPER
Available Online at www.ijarcs.info

Formal Verification of Access Control Policies

Ramanuj Chouksey*
Asst.Professor
School of Information Technology and Engineering
Vellore Institute of Technology, Vellore
Tamilnadu - 632014, India
ramanuj@vit.ac.in

R Sivashankari
Asst.Professor
School of Information Technology and Engineering
Vellore Institute of Technology, Vellore
Tamilnadu - 632014, India
sivashankari.r@vit.ac.in

Abstract: The primary purpose of security mechanisms in a system is to control access to information. Access control is the process of limiting access to the resources of a system to authorized users, programs, processes, or other systems. In this paper we illustrate different access control techniques and models that have been proposed in the literature and model checking approach to verify the properties of models. Model checking approach first expresses access control models in the specification language of a model checker. It expresses generic access control properties in temporal logic formulas and then uses the model checker to verify these properties for the access control models and generate the counter example for those properties which is not true in the specified model. We use NuSMV model checker tool. We present a case study of a health care system. The goal of our paper is to give a general approach for verification of a health care system using model checking.

Keywords: Access Control; RBAC,Model Checking;NUSMV;LTL;CTL;

I. INTRODUCTION

An important requirement of any system is to protect data and resources against unauthorized disclosure (secrecy) and unauthorized or improper modifications (integrity), while at the same time ensuring their availability to authorized users. Enforcing protection therefore requires that every access to a system and its resources be controlled so that only authorized accesses can take place. Thus the primary purpose of security mechanisms in a system is to control access to information. Access control [1, 2] is the process of limiting access to the resources of a system to authorized users, programs, processes, or other systems. The importance of access control is growing rapidly in a world where computers are ever-more interconnected. When planning an access control system, three abstractions of controls should be considered:

• **Access control policies:** Access control policies are high-level requirements that specify how access is managed and who may access information under what circumstances.

• **Security models:** Security models are formal presentations of the security policies enforced by the system and are useful for proving theoretical limitations of a system.

• **Security mechanism:** At a high level, access control policies are enforced through a mechanism that translates a user's access request, often in terms of a structure that a system provides.

For an access control system to be effective, it is important to ensure that its access control policies are properly defined. It is common that a system's privacy and security are compromised due to the misconfiguration of access control policies instead of the failure of protocols. The access control policy should consist of two aspects. First, the policy should provide users enough permissions to carry out their actions and achieve their legitimate goals. Secondly, at the same time, the policy should prohibit malicious goals from being reached. The achievability of malicious goals may reveal certain security holes in the policy. To formally and precisely capture the security properties that access control should contains, access control models are usually built and model checking approach is used to verify them.

The paper is organized as follow. Section II introduces some of the concepts that are commonly used in the field of access control and are also used throughout this paper. In Section III we described the role based access control technique and it's related models. In section IV we described the model checking approach for model verification and NuSMV model checker. In section V we present the case study of a health care system.

II. CONCEPT

- **Object:** An entity that contains or receives information. Access to an object implies access to the information it contains. Examples of objects are records, fields (in a database record), blocks, pages, segments, files, directories, programs etc.
- **Subject:** Active entities that can access or manipulate objects. At a high level users are subjects, but within the system, a subject is usually considered to be a process, job, or task, operating on behalf of the user.
- **Operation:** An active process invoked by a subject.
- **Permission (privilege):** An authorization to perform some action on the system.

III. ACCESS CONTROL TECHNIQUE

Access control techniques are sometimes categorized as either discretionary or non-discretionary. The three most widely recognized techniques are :

1. Discretionary Access Control (DAC)
2. Mandatory Access Control (MAC)
3. Role Based Access Control (RBAC)

In DAC, each object has an owner who exercises primary control over the object and can set an access control mechanism to allow or deny access to an object. The access matrix model [3] provides a framework for describing discretionary access control. MAC [1] is a security mechanism that restricts the level of control that users (subjects) have over the objects that they create. Unlike in a DAC implementation, where users have full control over their own files, directories, etc., MAC adds additional labels, or categories, to all system objects. DAC allows user to pass rights they possess to other users without constraints, MAC restrict how users can pass rights to others users. In DAC there is no distinction between users and subjects while MAC make a distinction between users and subjects. Users are human beings who can access the system, while subjects are processes (i.e., programs in execution) operating on behalf of users. This distinction allows the policy to control the indirect accesses (modifications) caused by the execution of processes and controls the direct and indirect flows of information to the purpose of preventing leakages to unauthorized subjects. The best known security model for MAC is Bell and LaPadula model.

A. Role Based Access Control (RBAC)

Role-based access control (RBAC) [4] is an approach to restricting system access to authorized users. In role-based access control, access decisions are based on the roles that individual users have as part of an organization. Role can be defined as a set of actions and responsibilities associated with a particular working activity such as doctor, nurse, manager. In RBAC instead of specifying all the accesses each users is allowed to execute, access authorizations are specified for roles. Users are then given authorizations to adopt roles. Thus in RBAC permissions are associated with roles, and users are made members of appropriate roles thereby acquiring the role's permissions. This greatly simplifies management of permissions. Users can be easily reassigned from one role to another. Roles can be granted new permissions and permissions can be revoked from roles as needed.

Four conceptual models has been defined to understand the RBAC. The relationship between these four models is shown in Figure 1. $RBAC_0$, the base model. $RBAC_1$ and $RBAC_2$ both include $RBAC_0$,but add independent features to it. $RBAC_1$ adds the concept of role hierarchies (situations where roles can inherit permissions from other roles). $RBAC_2$ adds constraints (which impose restrictions on different components of RBAC). $RBAC_1$ and $RBAC_2$ are incomparable to one another. $RBAC_3$, includes $RBAC_1$ and $RBAC_2$ and, by transitivity, $RBAC_0$. A general family of RBAC models called RBAC96 was defined by Sandhu et al. The RBAC96 model [5] is a comprised of four models: $RBAC_0$, $RBAC_1$, $RBAC_2$, $RBAC_3$ ..

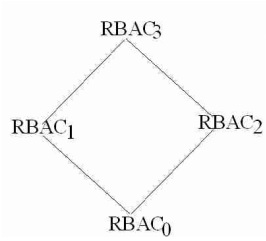

Figure 1: Relationship among RBAC models

B. RBAC 96

Figure 2 illustrates the RBAC96 model. In Figure 2 a single headed arrow indicates a one to one relationship and a double headed arrow indicates a many to many relationship. The top half of the figure shows roles and permissions in the system and the bottom half shows administrative roles and administrative permissions.

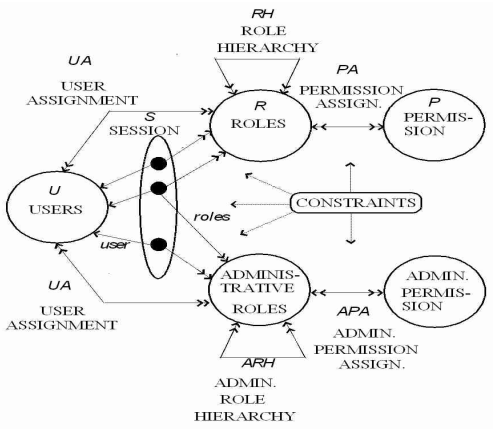

Figure 2: RBAC96 model

The RBAC96 model has the following components:

- Users : A user is a human being or an autonomous agent.
- Roles : A role is a job function or job title within the organization with some associated semantics regarding the authority and responsibility. A role may be a administrative role (AR) or regular role (R) (non-administrative). It is required that AR must be disjoint from R .
- Permission: Permission may be regular permission (P) or administrative permission (AP) . Administrative permissions control operations which modify the components of RBAC, such as adding new users and roles and modifying the user assignment and permission assignment relations. Regular permissions on the other hand control operations on the data and resources and do not permit administrative operations. P must be disjoint from AP .
- The user assignment (UA) : UA is many-to-many relation. A user can be a member of many roles, and a role can have many users.

- Permission assignment (PA) : PA is many-to-many relation. A role can have many permissions, and the same permission can be assigned to many roles. There is similarly a administrative permission assignment (APA) relation.
- Role hierarchy (RH) : RH is a partially ordered relation also written as $\pm$, where $x \pm y$ signifies that role x inherits the permissions assigned to role y. Equivalently $x \pm y$ signifies a user who is a member of x is also implicitly a member of y. There is similarly a partially ordered administrative role hierarchy (ARH).
- Session : Session relates one user to possibly many roles. The double-headed arrows from a session to R and AR indicate that multiple roles and administrative roles can be activated simultaneously. Each session is associated with a single user, as indicated by the single-headed arrow from the session to U. This association remains constant for the life of a session. A user may have multiple sessions open at the same time.
- Constraints : Constraints can apply to any components of RBAC. An example of constraints is mutually disjoint roles, such as purchasing manager and accounts payable manager, where the same user is not permitted to be a member of both roles.
- RBAC96 model components can be formalize as follow:
  - U is a set of users.
  - R and AR are disjoint sets of roles and administrative roles, respectively.
  - $UA \subseteq U \times (R \cup AR)$, is a many-to-many user to role, and administrative role assignment relation.
  - $PA \subseteq (P \times R)$ and $APA \subseteq (AP \times AR)$, are respectively, many-to-many permission to role assignment and administrative permission to administrative role assignment relations.
  - $RH \subseteq (R \times R)$ and $ARH \subseteq (AR \times AR)$, are respectively, partially ordered role and administrative role hierarchies.
  - S is a set of sessions.
  - $user : S \to U$, is a function mapping each session s_i to the single user $user(s_i)$ and is constant for the session's lifetime.
  - $roles : S \to 2^{R \cup AR}$ is a function mapping each session s_i to a set of roles and administrative roles.
  - $roles(s_i) \subseteq \{r \mid (\exists r' \geq r)[(user(s_i), r') \in UA]\}$ so that session s_i has the permission

$$\bigcup_{r \in roles(s_i)} \{p \mid (\exists r' \leq r)[(p, r') \in PA \cup APA]\}$$

There are many components in RBAC96 model. The issue of assigning users to role, assigning permission to roles, and assigning role to role are separated through an administrative model called ARBAC97 by R.Sandhu et al. ARBAC97 model [6, 7] has three components: URA97 (user--role assignment) model. PRA97 (permission--role assignment) model. RRA97 (role--role assignment) model. We will mainly concerned upon URA97 model and PRA97 model.

C. URA 97 Model

URA97 model is used for managing user role assignment defined in two steps : granting a user membership in a role and revoking a user's membership. URA97 uses the prerequisite condition to impose the restriction on which users can be added to a role.

- A prerequisite condition is a boolean expression on roles using the usual $\wedge$ and $\vee$ of the form x and $\neg x$, where x is a regular role $(i.e, x \in R)$.
- A prerequisite condition is evaluated for a user u by interpreting x to be true if $(\exists x' \pm x)\,(u, x') \in UA$, and $\neg x$ to be true if $(\forall x' \pm x)\,(u, x') \notin UA$.

URA97 controls user-role assignment and revocation can be defined by following relations

$$can\_assign \subseteq AR \times CR \times 2^R.$$

$$can\_revoke \subseteq AR \times 2^R.$$

$can\_assign(x, y, \{a, b, c\})$ means that a member of the administrative role x (or a member of an administrative role that is senior to x) can assign a user that satisfies the prerequisite condition y to be a member of regular roles a, b or c. $can\_revoke(x, Y)$ means that a member of the administrative role x (or a member of an administrative role that is senior to x) can revoke membership of a user from any regular role $y \in Y$.

D. PRA 97 Model

PRA97 is concerned with role-permission assignment and revocation. PRA97 is dual of URA97 model. The prerequisite condition is identical to that in PRA97, except the boolean expression is now evaluated for membership and non membership of a permission in specified role. Permission-role assignment and revocation can be defined by the following relations

$$can\_assignp \subseteq AR \times CR \times 2^R.$$

$$can\_revokep \subseteq AR \times 2^R.$$

$can\_assignp(x, y, \{a, b, c\})$ means that a member of the administrative role x (or a member of an administrative role that is senior to x) can assign a permission that satisfies the prerequisite condition y to regular roles a, b or c. $can\_revokep(x, Y)$ means that a member of the administrative role x (or a member of an administrative role that is senior to x) can revoke membership of a permission from any regular role $y \in Y$.

IV MODEL CHECKING

Model checking [8] is an automatic technique for verifying finite state systems. Specifications about the

system are expressed as temporal logic formulas, and the system is modeled as a state transition graph. The proof of specification is entirely carried out by machine. In the case specification does not hold, the model checker will construct a counterexample suitable for failure diagnosis.

Definition 1 *A finite state system can be described as a tuple* $M = \langle S, R, s_0, L \rangle$ *where*

- S =is a finite set of states.
- $R \subseteq (S \times S)$ transition relation satisfying $\forall S \in S \,.\, \exists s' \in S.(s, s') \in R$.
- $s_0 \subseteq S$ set of initial states.
- L is a function that labels states with atomic proposition from a given language.

We use NuSMV model checker [9, 10]. In the following section, we discuss the NuSMV model checker which will be later employed for the verification of a health care system.

E. Temopral Logic

Temporal logic is [11] an extension of classical logic. It uses atomic propositions, boolean connectives and some temporal operators. Temporal logic has been proposed as applying both to the specification and verification of program behavior, and to the specification of system behavior. Two useful temporal logics are Linear Temporal Logic (called LTL) and Computation Tree Logic (called CTL).

F. Linear Temporal Logic

Linear--Time temporal logic [11], or LTL for short, is temporal logic, with connectives that allow us to refer to the future. It models time as a sequence of states, extending infinitely into the future. This sequence of states is sometimes called a computation path, or simply a path. When a set of paths is considered, the LTL formula has to be true on all paths.

LTL uses atomic propositions, the usual boolean connectives $\neg, \wedge, \vee, \rightarrow$ and the following temporal operators:

- **X** (neXt) requires that the property holds at the next state of the path.
- **G** (Globally) requires that the property holds at every state on the path.
- **F** (eventually or in the future) holds when a property is true at some state of the path.
- **U** (Until) is a binary operator. Formula P U Q holds when P is true until Q becomes true. Also, the second argument must become true at some point.
- **W** (Weak-until) is a binary operator. Weak-until just like an Until operator except that formula P W Q does not require that Q is eventually satisfied along the path, which is required by P U Q.
- **R** (Release) is a binary operator. Formula P R Q requires that either Q is always true or it is true until P becomes true. For example

LTL formula $G(a\,U\,(b \wedge c))$ would mean that on every path either b or c is true at the current position or a holds until either b or c becomes true.

- The prefix “non” is not a word; it should be joined to the word it modifies, usually without a hyphen.
- There is no period after the “et” in the Latin abbreviation “et al.”.
- The abbreviation “i.e.” means “that is”, and the abbreviation “e.g.” means “for example”.

G. The NuSMV Model Checker

NuSMV stands for New Symbolic Model Verifier. NuSMV is a symbolic model checker that can be used to analyze temporal logic (LTL or CTL) specifications of various systems. The system can be expressed in the NuSMV modeling language. The system specifications are expressed in temporal logic. NuSMV takes as input a text consisting of a program describing a model and some specifications (temporal logic formulas). It produces as a output either the word true if the specification hold, or generate a counter example for the model represented by program. The analysis of the counter example is usually impossible to do automatically and thus involves human assistance. The counter example can help the designer to find the errors in the design or in the model. Specifications can be added in any module of the program. Each Specification is verified separately.

The NuSMV modeling language allows the representation of synchronous and asynchronous finite state systems. The models consist of one or more modules. Every model contains the main module, which can have references to the other modules. The states of a finite state system are described by the state variable values in NuSMV. Modules can declare variable and assign to them. With the help of init function, we can also set the initial values for state variables. It is also possible to define variables which do not change over time and variables which are completely unrestricted. The transition relation of a finite state system is represented as next commands in the NuSMV model. If no next is specified for a variable, then the variable can evolve nondeterministically.

In order to specify asynchronous systems, a process statement can be used. For a asynchronous systems (having more than one component), it is not required that each component be eventually executed, this is ensured through fairness conditions. NuSMV allows to specify fairness constraints. Thus, in order to ensure that an asynchronous process is eventually executed add the following condition to the module: FAIRNESS running.

IV. CASE STUDY HEALTH CARE SYSTEM

The policy for our health care system is based on the policy for a small aged-care facility [12] and with some aspects of the electronic health records policy [13]. We use RBAC96 style role based access control approach for our health care system and model checking approach to verify the properties. We write a smv program to describe the model and use LTL and CTL logic to specify the properties. Our health care system has Patient, Doctor, Nurse, and Manager roles. Manager role is an administrative role, decision for assigning this role to the user is taken by central

authorities. Role hierarchy relationship between these roles are as follow:

We use USER() module to define the initial role membership of the user. For example VAR Ram : array 1..4 of Patient, Employee, Manager, Nurse, Doctor, NoRole; ASSIGN init(Ram[1]):= Employee; init(Ram[2]):= NoRole; init(Ram[3]):= NoRole; init(Ram[4]):= NoRole; next(Ram[1]):= Ram[1]; next(Ram[2]):= Ram[2]; next(Ram[3]):= Ram[3]; next(Ram[4]):= Ram[4]; where entry in Ram[1] implies that whether Ram is an Employee or Patient or no role is assigned to him, entry in Ram[2] implies that whether he is Manager or not, entry in Ram[3] implies that whether he is Doctor or not, entry in Ram[4] implies that whether he is Nurse or not. The above code defines that initially Ram is only Employee.

Each role has some permissions, whenever some role is assigned to user, permissions associated with role applies to user. Permissions associated with role can be defined using permission assignment (PA) relation. $PA(p,r)$ means that role r has permission P. In our health care system permission associated with roles are as follow:

PA (Doctor, View_OldMedicalRecords)
PA (Doctor, View_RecentMedicalRecords)
PA (Doctor, Add_RecentMedicalRecords)
PA (Doctor, View_Prescriptions)
PA (Doctor, Add_Prescriptions)
PA (Doctor, View_PrivateNotes)
PA (Doctor, Add_PrivateNotes)
PA (Manager, View_OldMedicalRecords)
PA (Manager, View_RecentMedicalRecords)
PA (Manager, Add_RecentMedicalRecords)
PA (Manager, Access_PatientPersonalInfo)
PA (Nurse, View_OldMedicalRecords)
PA (Nurse, View_RecentMedicalRecords)
PA (Nurse, Add_ProgressNotes)
PA (Nurse, View_CarePlan)
PA (Patient, (View_OldMedicalRecords)
PA (Patient, (View_RecentMedicalRecords)
PA (Patient, View_Prescriptions)
PA (Patient, View_Bills)

A. Modeling Role Assignment and Revocation in SMV

Our Health care system has following ARBAC policy related to role assignment and revocation:

can_assign (Manager, true, Employee)
can_assign (Manager, Employee, Nurse)
can_assign (Manager, Employee, Doctor)
can_revoke (manager, Employee, Nurse, Doctor)

The above policies say that Manager can assign a Employee role to any user and can assign the role of Doctor or Nurse to Employee. Manager can also revoke the Employee, Doctor, Nurse role.

ROLE_ASSIGNMENT(USER) module is used for user role assignment in our smv program which takes the instance of USER() module as an argument. For example following code express that Manager can assign Nurse role to Ram if Ram is an Employee. next(USER.Ram[4]) := case (USER.Ram[1] = Employee) (USER.Ram[2] = Manager | USER.John[2] = Manager | USER.Tom[2] = Manager) : Nurse; 1 : USER.Ram[4]; esac; ROLE_REVOKE(USER) module is used for user role revocation in our smv program which takes the instance of USER() module as an argument. For example following code express that Manager can revoke the Nurse role from Ram.

next(USER.Ram[4]) := case (USER.Ram[2] = Manager | USER.John[2] = Manager | USER.Tom[2] = Manager) : NoRole; 1 : USER.Ram[4]; esac;

B. Modeling Permission Assignment and Revocation from Users in SMV

C. PERMISSION_ASSIGNMENT module is defined to assign permission to users in our smv program. For example:

VAR View_OldMedicalRecords : array 1..3 of
Assign, NoAssign; ASSIGN Patient | USER.Ram[2] = Manager | USER.Ram[3] = Doctor | USER.Ram[4] = Nurse) : Assign; 1 : NoAssign; esac;

where View_OldMedicalRecords[1] implies that whether Ram has permission to view old medical records or not. Above code describes that if Ram is Patient or Manager or Doctor or Nurse then he is eligible to view the Old medical records. Whenever these roles are revoked from Ram then permission are revoked from Ram in our smv program as follow:

next(View_OldMedicalRecords[1]) := case
(USER.Ram[1] = Patient | USER.Ram[2] = Manager |
USER.Ram[3] = Doctor| USER.Ram[4]) = Nurse : Assign;
1 : NoAssign; esac;

D. Specifying Properties in LTL/CTL

We specify the property of our health care system using LTL and CTL temporal logic in our smv program. The properties of our health care system are as follow:

Property1.
Is every Doctor an Employee? For our health care system this property is true since this follows directly from the role-hierarchy. We can express this in LTL as follow: G (USER.Ram[3] = Doctor -> USER.Ram[1] = Employee);

Property2.
Can a user may be a member of Doctor and Nurse role simultaneously? For our health care system this property is true since the policy does not enforce disjointness of these roles. We express this in CTL as follow: EF (USER.Ram[3] = Doctor USER.Ram[4] = Nurse);

Property3.
Can a user have Add_ProgressNotes and Add_PrivateNotes permission simultaneously? For our health care system this property is true because a user can be a member of Doctor and Nurse role and these permission is assigned to Doctorand Nurse. We express this in CTL as follow:
EF (PERMISSION_ASSIGNMENT.Add_ProgressNotes[1] = Assign & PERMISSION_ASSIGNMENT .Add_PrivateNotes[1] = Assign

Property4.
Is every user with permission View_RecentMedicalRecords a member of some role in $\{Doctor, Patient\}$? This property is false for our health care system since nurse has also this permission. We express this in LTL as follow: G ((PERMISSION_ASSIGNMENT.View_RecentMedicalRecords[1] = Assign) ->(USER.Ram[4] = Doctor | USER.Ram[1] = Patient));

E. VI. RESULTS

In the previous section, we described the smv program and properties of our health care system. We verified the properties using NuSMV model checker. NuSMV model takes smv program as a input, verifies each property separately and returns true if property hold otherwise it gives the counter example. In our health care system model the no. of reachable states are 1989 out of 2^{64} states. In Table 1 we summarize the result obtained for different properties and the time taken to verify the property. The computer used is a laptop running widows vista on a intel dual core 2 GHz and 3GB RAM.

Table 1: Results of Verification

Property	Time taken (in seconds)	Result
Property1	0.1	True
Property2	0.1	True
Property3	0.1	True
Property4	0.8	False

VII CONCLUSION

In this paper we have illustrated different access control techniques and models that have been proposed in the literature. The concept of role is associated with the notion of functional roles in an organization, hence RBAC models provide support for expressing organizational access control. RBAC model are suitable for handling access control requirements of various organizations and service-based applications such as e-commerce, Loan origination process etc. ARBAC97 model controls changes to the user-role assignment, the permission-role assignment and the role hierarchy relation. We have taken health care system as a case study. We applied the RBAC technique to design the model of our health care system. We applied model checking technique to verify the specification of our health care system. We used NuSMV as model checker. Using a case study of a health care system, we have shown how an application can be specified and verified with the models in NuSMV via model checking. In our future work we would like to verify more complicated system where role hierarchy can be changed and two users for the same role having different permissions.

REFERENCES

[1] P. Samarati and S. de Capitani di Vimercati, "Access control: Policies, models, and mechanisms," Lecture Notes in Computer Science, vol. 2171, pp. 137–196, 2001.

[2] V. C. Hu, D. Ferraiolo, and D. R. Kuhn, "Computer security - access control," NIST Interagency Report 7316, Tech. Rep., 2006.

[3] M. Harrison, W. Ruzzo, and J.Ullman, "Protection in operating systems," Communication of the ACM, vol. 19, no. 8, pp. 461–471, 1976.

[4] R. S. Sandhu, E. J. Coyne, H. L. Feinstein, and C. harles E. Youman, "Role-based access control models," IEEE Computer, vol. 20, no. 2, pp. 38–47, Feb. 1996.

[5] R. S. Sandhu, "Rationale for the RBAC96 family of access control models," in ACM Workshop on Role-Based Access Control, 1995.

[6] R. S. Sandhu, V. Bhamidipati, and Q. Munawer, "The ARBAC97 model for role-based administration of roles," ACM Trans. Inf. Syst. Secur, vol. 2, no. 1, pp. 105–135, 1999.

[7] R. S. Sandhu and V. Bhamidipati, "Role-based administration of user-role assignment: The URA97 model and its oracle implementation," Journal of Computer Security, vol. 7, no. 4, 1999.

[8] E. M. Clarke, O. Grumberg, and D. A. Peled, Model Checking. Cam-bridge, Massachusetts: The MIT Press, 1999.

[9] R. Cavada, R. Cimatti, E. Olivetti, and M. Pistore, "NuSMV 2.1 user manual," jun 2004.

[10] A. Cimatti, E. M. Clarke, F. Giunchiglia, and M. Roveri, "NUSMV: A new symbolic model checker," STTT, vol. 2, no. 4, pp. 410–425, 2000.

[11] M. Huth and M. Ryan, Logic in Computer Science, 2nd ed. Cambridge University Press, 2004.

[12] M. Y. Becker, "A formal security policy for an NHS electronic health record service," Tech. Rep., Mar. 2005.

[13] R. S. Sandhu and V. Bhamidipati, "Role-based administration of user-role assignment: The URA97 model and its oracle implementation," Journal of Computer Security, vol. 7, no. 4, 1999.