Jurnal Kejuruteraan 32(1) 2020: 1-7
https://doi.org/10.17576/jkukm-2020-32(1)-01

Modelling and Verifying Dynamic Access Control Policies in Workflow-Based Healthcare Systems

Rokan Uddin Faruqui [*,a]
[a]*Department of Computer Science and Engineering, Faculty of Engineering, University of Chittagong, Bangladesh.*
**Corresponding author: rufaruqui@cu.ac.bd*

Received 16 January 2018, Received in revised form 5 September 2019
Accepted 21 November 2019, Available online 28 February 2020

ABSTRACT

Access control system is an important component to protect patients' information from abuse in a health care system. It is a major concern in the management, design, and development of healthcare systems. Designing access control policies for healthcare systems is complicated due to the dynamic and inherent complexity of the tasks performed by the healthcare personnel. Permissions in access control systems are usually granted on the basis of static policies. However, static policies are not enough to cope with various situations such as emergencies. Most often, the Break-the-glass mechanism is used to bypass static policies to handle emergency situations. Since healthcare systems are critical systems, where errors can be very costly in terms of lives, quality of life, and/or dollars, it is crucial to identify discrepancies between policy specifications and their intended function to implement correctly a flexible access control system. Formal verifications are necessary for exhaustive verification and validation of policy specifications to ensure that the policy specifications truly encapsulate the desires of the policy authors. We present a verifiable framework to enact a dynamic access control model by integrating the ANSI/INCTIS RBAC Reference Model in a workflow and an approach for property verifications of the access control model. Access control policies are expressed by the formal semantics of a model checker and properties are verified by the DiVinE model checker.

Keywords: Model Checking; Formal Verifications; Access Control.

INTRODUCTION

An access control system regulates the operations that can be executed on the data and resources to be protected. The essence of access control is in determining what access privileges a given user can exercise in a given context. Access control decisions are driven by access control policy. Popular access control models include the Chinese Wall Access model, discretionary access control (DAC), mandatory access control (MAC), and role-based access control. Role-based access control (RBAC) is a preferred model implemented in healthcare systems. The American National Standard Institute (ANSI) and the National Institute of Standards and Technology (NIST) developed the standard RBAC Reference Model (ANSI 2004). Canada Health Infoway also developed a conceptual model for access control using RBAC (Infoway 2005).

The RBAC model is usually inflexible in making decisions and provides only two output decisions: grant or deny. The main drawback of the RBAC model is failing to provide the flexibility required to deal with emergency situations which are typical in healthcare applications. The Break-the-glass concept is an approach to granting emergency access to healthcare systems to overcome this problem (SPC 2004). Most existing implementations use break-the-glass as an exception-handling mechanism. The Rostard (2009) study observed the use of the break-the-glass mechanism in a system in which normal access control is enforced as a combination of role and workplace. They found that exception-handling requests exceed access requests and concluded that static approaches to define access control policies do not achieve reasonable outcomes in a dynamic hospital setting and will not reduce the misuse of exception-handling mechanisms. They also found that combining norms and practices, medical guidelines, observational data, and audit data to learn patterns of information used in healthcare and then applying these patterns to create access control rules will help to minimize the use of the break-the-glass access. These resources also contain information about workflow in healthcare. We present a verifiable framework for dynamic access control model integrating with standard RBAC that will work as an enactment service in healthcare workflow. We also present an approach for verifying the policies of access control.

Access control system is the critical component of an information systems. It is common that a system's privacy and security are compromised due to the wrong configuration of access control policies. This problem becomes increasingly severe in healthcare systems because healthcare systems are large scale systems involving complex and distributive processes which are deployed to manage a huge amount of sensitive information and resources. Moreover, the

information and resources are organized into sophisticated structures with a high degree of variability. Healthcare systems are also critical systems, where errors can be very costly in terms of lives, quality of life, and/or dollars. It is crucial to identify discrepancies between policy specifications and their intended function to implement correctly a flexible access control system. Formal verifications are necessary for exhaustive verification and validation of policy specifications to ensure that the policy specifications truly encapsulate the desires of the policy authors. We present a model checking based approach for automated verifications of access control policies. Access control policies are expressed by the formal semantics of the model checker and properties are verified by the model checker.

Our contributions are twofold: first, we present a workflow-based access control model based on standard RBAC for healthcare systems with an exception handling mechanism. Second, we present formal verifications for security analysis of the access control policies. We used the distributed and high performance model checker DiVinE (Verstoep et al., 2009; Barnat et al., 2009) to verify the access control policies of our proposed model for a healthcare workflow.

The remainder of the paper is organized as follow: firstly, we present background information; then we introduce the core RBAC reference model and our proposed workflow-based access control model. Thirdly, we present a brief discussion of how the proposed model. Finally, properties are specified and verified by the model checker and related works are presented.

PRELIMINERIES

CORE RBAC

Role Base Access Control (RBAC) is the state-of-the-art access control model proposed in (Sandhu 1996). In this section, we present the ANSI/INCITS core RBAC model.

The Core RBAC Reference Model defines the set of basic elements: users, roles, operations, sessions, permissions, and objects. It consists of the following relations:

1. User-Assignment: UA $\subseteq$ USERS $\times$ ROLES, a many to many mapping user-to-role assignment.
2. Permission-Assignment: PA $\subseteq$ PRMS $\times$ ROLES, a many to many mapping permission-to-role assignment. PRMS = $2^{(OPS \times OBS)}$, the set of permissions.
3. The set of operations associated with permission p is Op(p $\in$ PRMS) $\rightarrow$ {op $\in$ OPS}.
4. The set of objects associated with permission p is Ob (p $\in$ PRMS) $\rightarrow$ {ob $\in$ OBS}.
5. SESSIONS is the set of sessions. User-Session: US (s $\in$ SESSIONS) $\rightarrow$ USERS, the mapping of user u onto a set of sessions. Session-Roles: SR(s $\in$ SESSIONS) $\rightarrow$ 2^{ROLES}, the mapping of session s onto set of roles.

The core RBAC model makes authorization the decisions grant/deny by taking user information, intended operations, and the target object as inputs. It can be expressed as follows:

$$coreRBAC: USERS \times OPS \times OBS \rightarrow \{Grant, Deny\}$$

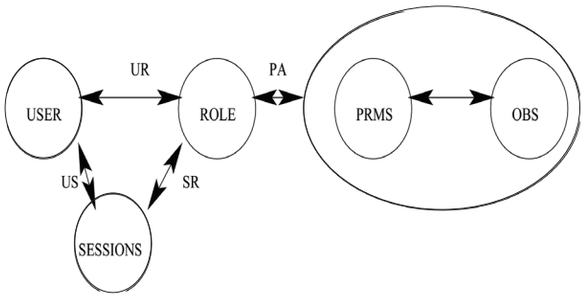

FIGURE 1. The core RBAC model

The core RBAC module identifies the role **r** for the user **u** then check the permissions for role **r** to perform operations **Ops** on object **obj**.

MODEL CHECKING

Model checking is an automated technique that, given a finite-state model of a system and a logical property, systematically checks that this property holds for a given initial state in that model (Clarke 1986). Model checking requires an exhaustive search through all possible configurations, or states, of a system. We use liner temporal logic (LTL) to specify the properties of our access control model. Temporal logic is a formalism for describing sequences of transitions between states in a reactive system like workflow. In temporal logic, time is not specified explicitly; instead, temporal operators are used to specify properties such as: eventually some designated states are reached, or an error state is never reached. LTL is a 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. LTL formulas are built from predicates, logical connectives, universal and existential quantifiers, and modalities **G** (Globally in the future), **F** (eventually in the future), **X** (in the next step) and **U** (until).

We use automata-based model checking (Kupferman 2000) to verify access control policies. In automata-based model checking, the modeled system is converted into a corresponding Buchi automaton and the negation of property specification to be checked is converted into another automaton which is known as never-claim automaton. Then, the emptiness of the intersection of the system and the negated specification automaton is checked. If the system holds the property then the intersection is empty. If the intersection is not empty, a counter example is reported.

DIVINE MODEL CHECKER

DiVinE is a distributed and high performance explicit-state model checker that follows the automata-based approach. It has several platform dependent versions such as the

distributed-memory DiVinE Cluster (Barnat et al. 2005), and the shared-memory DiVinE Multi core (Barnat et al. 2008). DiVinE offers model checking of finite state systems against specification formulated in LTL.

State-space explosion is the main concern of the model checking. DiVinE utilizes high performance cluster computing to overcome this problem. Due to the looping and inherent complexity of workflow, we use DiVinE to verify WfAC module integrated with workflow. The finite state machine (FSM) specifications of Workflow-based Access Control Model (WfACM) are translated into DVE , the formal semantics of DiVinE that use the standard notation of the Communicating Sequential Processes (CSP) (Hoare, 1983). Therefore, it can be translated into the formal semantics of any classical finite-state model checker such as SPIN.

PROPOSED MODEL

WORKFLOW-BASED ACCESS CONTROL MODULE

We propose the Workflow-based Access Control Model (WfACM) as a part of a workflow engine to enact access control policy. The WfACM enforces the dynamic access policies which vary some process to process. WfACM is interfaced with standard ANSI/INCITS RBAC with modifications for exception-handling and reducing misuse. WfACM is shown in Figure 1.

The Workflow-based Access Control model first checks the mode of operations. There are two possible modes: normal mode and emergency/exception-handling mode. If normal mode is selected, it invokes core RBAC module which is part of the main model in which core RBAC is implemented. In case of emergency/exception-handling modes, it performs some additional tasks that include: re-validating user credentials, triggering for audit logging, asking user to provide reasons for using emergency mode, and notifying the resource owner; and finally, changing user role. An exception-handling role is assigned to the current user. This role is not a flat exception-handling role; it varies from process to process and grants the least privileges i.e., necessary permissions to complete the assigned task. The standard RBAC procedure is then invoked with following arguments: user, temporarily assigned role, operations and objects.

The workflow-based access control model makes authorization decisions by taking all the information necessary for core RBAC module with additional mode information as input. It provides grant decision if the user is adequate to perform the actions on the targeted object in the current mode, or deny if access is not granted. It can be expressed as follows:

$$WfAC: MODE \times USERS \times OPS \times OBS \rightarrow \{Grant, Deny\}$$

The WfAC module identifies the role **r** for the user **u** then checks the permissions for role **r** to perform operations **Ops** on object **obj** for the specified mode.

FINITE STATE MACHINES (FSMS) SPECIFICATION OF WFACM

Dynamic constraints that regulate the access permissions in the workflow can be specified with asynchronous specification expressions of FSM. Hu et al. (2008) developed a FSM specification to model the generic access control policies for verifications. They specified an authorization process for a user with four states: idle, entering, critical, and exiting for a particular dynamic constraint.

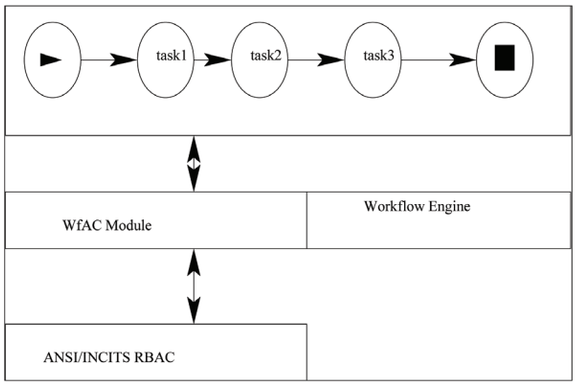

FIGURE 2. Workflow-based Access Control Model

A user is normally in the *idle* state. He will move to the *entering* state when he wants to access the critical object. If it is in the access limit, he will move to the critical state, and the number of the current access is increased by 1. When the user finishes accessing the critical object, he will return back to the *exiting* state, and the number of the current access is decreased by 1. The user is then move from the *exiting* state to the *idle* state again. We use the similar specifications with the extensions for exceptions handling.

```
VARIABLE
1 count, access_limit :
INTEGER;
2 req1 : WfAC(access_limit);
3 req2 : WfAC(access_limit);
4 req3 : WfAC(access_limit);
end
ASSIGN
5 access_limit : = MAX_LIMIT;
6 count : = 0;
7 act : = {rd, wrt};
8 object : = {obj};
end
```

ALGORITHM 1. FSM Specifications

We introduce a mode variable to enforce different access policies. In normal mode, it follows the similar RBAC reference model and in emergency mode, it performs the tasks recommended for exceptions handling. In our framework,

we convert this FSM specification into a Kripke Structure and translate the Kripke Structure into the formal semantics of model checker. The FSM specifications for WfACM are given in Algorithm 1 and its procedure is shown in Procedure 1.

MODELING WFACM WITH DICINE

The workflow access control properties to be verified are specified in LTL. Then we translate the Kripke structure and LTL property into DVE language. DVE specification and defined properties are similar to finite state automata. Each automaton is translated as a process in DVE and processes interact through channels. Process can read from (designated by "?") or write to (designated by "!") channels. We use the same mapping relations defined in RBAC reference model. Numerical values are assigned to each entity of USERS, PRMS, OBS, OPS for unique identifications. Roles: Manager = 0, CareNavigator = 1, CaseManager = 2, Pharmacist = 3, SocialWorker = 4, SystemNavigator = 5, Users: MacIsac = 0, Kmiller = 1 etc. Ops: Read = 1,Write = 2. Obs: PatientInfo = 0. PrescribedDrug/Therapy = 1. ExtraSensitiveInfo = 2. Prms: Grant = 1, Deny = 0.

Two-dimensional mapping relations are represented by one dimensional variable because DiVinE does not support multidimensional array variable. A vector variable UA′ is used to represent many to many mapping user-to-role assignment UA $\subseteq$ USERS $\times$ ROLES. The first index of UA′ indicates the user information then followed by the Boolean information of all roles. If a user is assigned to a role then its value is 1 otherwise 0. In similar way, all users' information is stored linearly. For example, if USERS = 2, ROLE = 3, then UA [8] = $\{u_1, r_1, r_2, r_3, u_2, r_1, r_2, r_3\}$.

Another vector variable PA′ is used to represent many to many mapping permission-to-role assignment PA $\subseteq$ PRMS $\times$ ROLES. The first index of PA′ indicates the object followed by the role information followed by the access rights information. For example, if OBJECTS = 3, ROLE = 3, then PA[30] = $\{obj_1, r_1, R, W, r_2, R, W, r_3, R, W, obj_2, r_1, R, W, r_2, R, W, r_3, R, W, obj_3, r_1, R, W, r_2, R, W, r_3, R, W\}$. R/W stands for Boolean read/write permissions.

There are five processes for WfACM: *checkaccess, normal Mode Operation, emergency Mode Operation, check-Permission,* and decisions. The CheckAccess process is invoked by any other processes in workflow that needs to use access control service. It accepts user information, objects information and actions intended to perform. Then it also considers the mode of operations whether it is normal mode or emergency / exception handling mode. The DVE code for the *Check Access* process is given in Figure 3.

AC POLICIES VERIFICATIONS

We outlined the workflow access control model in DVE language. In this section, we will show how access control policies for healthcare systems can be checked by DiVinE. Li and Tripunitara (2006) introduced a family of security analysis problems in RBAC. We only verified the *restrictions* property. In a healthcare system, a restriction property can be formulate as: *Does a pharmacist have the access to the patient personal information?* In our workflow, a user whose role is pharmacist doses not have the access to the patient information but in emergency, the pharmacist may need to know a patient's allergy information to execute prescribed drug or therapy. If we check the property for pharmacist p, and it returns accepting cycle then we can check the property which will return no accepting cycle. Verifications result is shown in Table 1.

RELATED WORK

Research on the rise is to design dynamic access control model with exception handling features for healthcare system. RBAC model is a widely-accepted access control model, but due to rigidness in making decisions, it is not suitable for healthcare systems. To overcome this problem, numerous methodologies are proposed. Povey introduced the idea of optimistic security in (Povey, 1999) which is a rollback concept in case of misuse and information from the audit logs are traces to identify the misuse. They claim that this idea is suitable for healthcare system to compensate misuse. Optimistic security exists in many healthcare systems as a break-the-glass mechanism which is intended to be used in emergency situations.

Russello et al. argues that such property does not always hold. In many situations, all information is not recoverable such as the disclosed confidentiality of the patient (Russello et al. 2008). Etalle and Winsborough (2007) presented Posteriori compliance control which delays the access control after gaining access decisions by providing an infrastructure which allows securely auditing and rolling back in case of a denied access, focusing on risks of not granting privileges. We used the notion of workflow to specify and implement the fine-grained access control policies.

In workflow, specific tasks are assigned to the designated entities and this feature will help to reduce the misuse of the exception handling mechanisms. A framework is proposed in (Russello et al. 2008) where access rights are provided to entities on the basis of the actual task so that the assigned tasks must be fulfilled by the entities as a part of their duties. For capturing the requirements of entities' duties, the notion of workflow is used. They argued that RBAC is not flexible enough for coping with the dynamic working environment. They used YAWL for specifying workflow which is similar to our approach. However, they proposed a new policy based access control module using Ponder2 policy language and interpreter developed at Imperial College London (Twidle et al. 2009). But we are motivated to use standard RBAC model due to its broadly recognition. Ferreira et al. (2009) proposed a framework that integrates break-the-glass mechanism within the NIST/ANSI RBAC model to adopt generically in any domain

```
WfAC(access_limit)
VARIAABLE
    permission:{start,grant,deny};
    mode:{normal,emergency};
state:{idle,entering,critical,exiting,};
    state_em:{em_idle,em_entering,
em_critical,em_exiting,em};
end
INITIAL(permission):=start;
INITIAL(state):=idle;
INITIAL(state_em):=em_idle;
NEXT(state) :=CASE
    state == idle and mode == normal
:{idle,entering};
    state == idle and mode == emergency and
state_em == em_exiting :{idle,entering};
    state == entering and !(count >
access_limit):{critical};
    state == critical :{critical,exiting};
    state == exiting : idle;
    DEFAULT: state ;
ESAC
NEXT(state_em) :=CASE
    state_em == em_idle and mode == emergency and
state == idle :{em_idle,em_entering};
    state_em == em_entering :{em_critical};
    state_em == em_critical
:{em_critical,em_exiting};
    state_em == em_exiting : em_idle;
    DEFAULT: state_em ;
ESAC
INITIAL(mode):= normal;
NEXT(mode) :=CASE
    mode == emergency and state_em =   em_exiting
and state != idle : normal ;
DEFAULT: mode ;
ESAC
NEXT(permission) :=CASE
(u,r) in U-R and (p,r) in P-R and op in act and
obj in OBJECTS : grant ;
OTHERWISE: deny ;
ESAC
```

PROCEDURE 1. WfAC(access-limit)

where unanticipated or emergency situations may occur. The model provides a third decision option break the glass along with two decisions by RBAC. However, they proposed a new model apart from the standard model for exception handling.

Due to the complexity and importance of access control models, formal methods have been applied to verify the required properties. Vincent et al. [13] provides a method for specifying generic access control policies by

```
process checkAccess
{
  int mode, U, R, OB, act;
  state idle, entering, critical, exiting
  init idle;

  trans
  idle -> entering
  {
  sync ch_checkAccess ? {mode, U, OB, act}; },
  entering -> critical
  {guard mode == 0; sync ch_normal !{U, OB, act};},
  entering -> critical
  {guard mode == 1; sync ch_emergency !{U, OB, act};},
  critical -> idle{};
}
```

FIGURE 3. checkAccess Process

finite state specifications. They used the well-known SMV model checker to verify access control policies specified in computation tree logic (CTL). There are other model checking based verifications approach for verifying access control policy such as (Zhang et al. 2008). They modeled rule-based policies in their Read/Write (RW) language based on propositional logic and conduct model checking on the policies. However, our approach is specifically in the context of workflows. The RW language is not primarily designed towards such needs.

Schaad et al. (2006) used model checking techniques for automated analysis of revocation and delegation functionalities in organizational workflow. They used the symbolic model checker NuSMV (Cimatti et al. 2002) to verify the properties specified in liner temporal logic (LTL). We use an automata-based distributed and parallel model checker to verify the access control policies integrated with workflow.

```
  process identifyRole
  {

    int userIndex = 0, U, k = 0, N = 0;
    int act, OB, flag = 0;
    state idle, for_k, while  roleIdentify, entering;
    init idle;

    trans
    idle -> entering
    { sync ch_getUser ? {U, OB, act}; },
    entering -> for_k
      {effect userIndex    U*(TOTAL_ROLE + 1),
      k = userIndex + 1,   N = userIndex + TOTAL_ROLE;
  },
    for_k -> while, { guard k < N; },
    while -> roleIdentify  {guard UA[k] == 1;     } ,
    for_k -> idle
      { guard k == N ; sync ch_decisions3 ! 2 ;},
    while -> for_k
    { guard UA[k] != 1 &&  k < N; effect k = k + 1;},
    roleIdentify -> idle
    {sync ch_role !  {k - userIndex , OB, act}; };
  }
```

FIGURE 4. identifyRole Process

CONCLUSION AND FUTURE WORK

In this paper, we have presented a Workflow-based Access Control Model that incorporates the standard RBAC with exception handling mechanism which is especially suitable for healthcare system. Our model suggests the dynamic

TABLE 1. Verifications Result

Property	Mode	Accepting Cycle	States (MB)	Memory (MB)	Time (Seconds)
Restrictions	Normal	No	62133	1371.98	5.644
Restrictions	Emergency	Yes	99256	1577.79	22.231

access control policies will be extracted from the sources of information necessary to develop the workflow and therefore, the medical guidelines, best practices and observational information will be used to design and verify the access control policies. In this paper, we have not considered the role hierarchy.

```
process checkPermission
{

  int Role, index, obj, act;

  state idle, entering, working,grant,
     deny,   checking;
  init idle;
  commit checking;
  trans
  idle -> entering  { sync ch_role ?
          {Role, obj, act}; },
  entering -> checking
  { effect index  = obj*
      (3*TOTAL_ROLE + 1) + 3*Role + 1  + act; },
  checking -> grant  { guard  PA[index] == 1; },
  checking -> deny  { guard  PA[index] != 1; },
  grant ->idle {sync ch_decisions1 !1 effect
  gvmode = 0, gvuser = 0, gvact = 0,
                        gvobj = 0, index = 0;},
  deny -> idle { sync ch_decisions2 !2;
        effect gvmode = 0, gvuser = 0,
         gvact = 0, gvobj = 0, index = 0; };
}
```

FIGURE 5. checkPermission Process

REFERENCES

American National Standards Institute, Inc. 2004. American National Standard for Information Technology – Role Based Access Control (ANSI/INCITS 359-2004).

Barnat, J., Brim, L., Černá, I. 2005. Cluster-based LTL model checking of large systems. *Formal Methods for Components and Object (FMCO) Lecture Notes in Computer Science* 4111: 259-279.

Barnat, J., Brim, L., Ročkai, P. 2008. Divine multi-core – a parallel ltl model-checker. *Automated Technology for Verification and Analysis Lecture Notes in Computer Science* 5311: 234-239.

Barnat, J., Brim, L., Ročkai. 2009. Divine 2.0: High-performance model checking. *Proceedings of the 2009 International Workshop on High Performance Computational Systems Biology*, 31-32.

Canada Health Infoway Inc. 2005. Electronic Health Record Infostructure (EHRi) Privacy and Security Conceptual Architecture (Version 1.1).

Cimatti, A., Clarke, E., Giunchiglia, E., Giunchiglia, F., Pistore, M., Roveri, M., Sebastiani, R., Tacchella, A. 2002. NuSMV 2: An OpenSource Tool for Symbolic Model Checking. *Proceedings of the 14th International Conference on Computer-Aided Verification,* LNCS 2404: 359-364.

Clarke, E.M., Emerson, E.A., Sistla, A.P, M. 1986. Automatic verification of finite-state concurrent systems using temporal logic specification. *Association for Computing Machinery Transactions on Programming Languages and Systems (TOPLAS)* 8(2): 244-263.

Etalle, S., Winsborough, W.H. 2007. A workflow-based access control framework for e-health applications. *Proceedings of the 12th Association for Computing Machinery symposium on Access control models and technologies, New York, NY, USA, ACM,* 11-20.

Ferreira, A., Chadwick, D.W., Farinha, P., Correia, R.J.C., Zhao, G., Chilro, R., Antunes, L. 2009. How to securely break into rbac: The btg-rbac model. *Proceedings of the 2009 Annual Computer Security Applications Conference, IEEE Computer Society,* 23-31.

Hoare, C.A.R. 1983. Role-based access control models. *Association for Computing Machinery Communications*, 26(1): 100-106.

Hu, V.C., Kuhn, D.R., Xie, T. 2008. Property verification for generic access control models. *Proceedings of the 2008 IEEE/IFIP International Conference on Embedded and Ubiquitous Computing, Washington, DC,* USA, IEEE *Computer Society,* 243-250.

Joint NEMA/COCIR/JIRA Security and Privacy Committee (SPC). 2004. Break-glass: An approach to granting emergency access to Healthcare Systems. Rosslyn, VA, USA

Kupferman, O., Vardi, M.Y. 2000. An automata-theoretic approach to modular model checking. *Association for Computing Machinery Transactions on Programming Languages and Systems (TOPLAS)* 22(1): 87-128.

Li, N., Tripunitara, M.V. 2006. Security analysis in role-based access control. *Association for Computing Machinery Transactions on Information and System Security (TISSEC)* 9(4): 391-420.

Povey, D. 1999. Optimistic security: a new access control paradigm. *Proceedings of the 1999 workshop on New security paradigms, New York, NY, USA, ACM,* 44-45.

Russello, G., Dong, C., Dulay, N. 2008. A workflow-based access control framework for e-health applications. *Proceedings of the AINA Workshops,* 111-120.

Rostard, L. 2009. Access Control in Healthcare Information Systems. PhD thesis, Norwegian University of Science and Technology, Department of Computer and Information.

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

Schaad, A., Lotz, V., Sohr, K. 2006. A model-checking approach to analysing organisational controls in a loan origination process. *Proceedings of the ACM Symposium on Access Control Models and Technologies (SACMAT),* 139-149.

Twidle, K., Dulay, N., Lupu, E. and Sloman, M. 2009. Ponder2: A Policy System for Autonomous Pervasive Environments. *Proceedings of the Fifth International Conference on Autonomic and Autonomous Systems, Valencia*, 330-335.

Verstoep, K., Bal, H.E., Barnat, J., Brim, L. 2009. Efficient large-scale model checking. *Proceedings of the 2009 IEEE International Symposium on Parallel & Distributed Processing*,1-12.

Zhang, N., Ryan, M., Guelev, D.P. 2008. Synthesising verified access control systems through model checking. *Journal of Computer Security* 16(1): 1-61.