Verification of Secure Inter-operation Properties in Multi-domain RBAC Systems

Antonios Gouglidis, Ioannis Mavridis
Department of Applied Informatics
University of Macedonia
156 Egnatia Str., 54006, Thessaloniki, Greece
Email: {agougl, mavridis}@uom.gr

Vincent C. Hu
Computer Security Division
National Institute of Standards and Technology
Gaithersburg, MD 20899-8930, USA
Email: vincent.hu@nist.gov

***Abstract*—The increased complexity of modern access control (AC) systems stems partly from the need to support diverse and multiple administrative domains. Systems engineering is a key technology to manage this complexity since it is capable of assuring that an operational system will adhere to the initial conceptual design and defined requirements. Specifically, the verification stage of an AC system should be based on techniques that have a sound and mathematical underpinning. Working on this assumption, model checking techniques are applied for the verification of predefined system properties, and thus, conducting a security analysis of a system. In this paper, we propose the utilization of automated and error-free model checking techniques for the verification of security properties in multi-domain AC systems. Therefore, we propose a formal definition in temporal logic of four AC system properties regarding secure inter-operation with Role-Based Access Control (RBAC) policies in order to be verified by using model checking. For this purpose, we demonstrate the implementation of a tool chain for expressing RBAC security policies, reasoning on role hierarchies and properly feeding the model checking process. The proposed approach can be applied in any RBAC model to efficiently detect non-conformance between an AC system and its security specifications. As a proof of concept, we provide examples illustrating the verification of the defined secure inter-operation properties in multi-domain RBAC policies.**

***Index Terms*—Verification, model checking, multi-domain, RBAC, secure inter-operation, temporal logic.**

I. Introduction

Access control (AC) in modern distributed systems has become even more challenging since they are complicated and require the collaboration among domains. A domain can be defined as a protected computing environment, consisted of users and resources under a same AC policy. AC is an essential process in all systems. The role of an AC system is to control and limit the actions or operations in a system that are performed by a user on a set of resources. An AC system is considered of three abstractions of control, namely AC policies, AC models, and AC mechanisms. A policy can be defined as a high-level requirement that specifies how a user may access a specific resource and when. AC policies can be enforced in a system through a machine enforceable AC mechanism that is responsible for permitting or denying a user access upon a resource. An AC model can be defined as an abstract container of a collection of AC mechanism implementations, which are capable of preserving support for the reasoning of the system policies through a conceptual framework. Consequently, the AC model enables the bridging of the existing abstraction gap between the mechanism and the policy in a system [1], [2].

As stated in [2], a system can be argued to be secure only if the model is secure and the mechanism correctly implements the model. Therefore, systems engineering technologies should be applied to manage the complexity of AC systems. The development process of a system includes the stages of requirements engineering, system's design and implementation, and verification. However, despite the importance of the aforementioned system's life cycle, not all of the aforementioned stages are performed adequately during the development of an AC system. Usually, the stage of verification where an AC system is being verified against its initial security requirements is absent or poorly performed.

The principal methods for the verification of complex systems can be grouped under four types. These are testing, simulation, deductive verification, and model checking [3]. Testing is performed on the system itself. However, testing of distributed systems is not always a cost effective process since it can be performed when an implementation of the system is available. Furthermore, it can only prove the existence of bugs, but not their absence. Simulation-based approaches ensure that a finite number of user-defined system trajectories meet the desired specification. However, simulation suffers from completeness as it is impossible or impractical to test all system trajectories. Furthermore, simulation-based testing is semi-automatic since the user must provide a large number of test cases. Deductive verification is based on manual mathematical proof of correctness of a model of a system. It is a very highly cost process and, furthermore, requires highly skilled personnel. Model checking performs exhaustive testing of all behaviours of a model of the system; It is not vulnerable to the like-hood that an error is exposed. This contrasts with testing and simulation that are aimed at tracing the most probable defects. Additionally, it provides diagnostic information in case a property is invalidated, which is very useful for debugging purposes. In principle, model checking is an automated process and its use requires neither a high degree of user interaction nor complex test data. Furthermore, it does not require the development of custom tools for verifying

a system, which can be a time consuming and error-prone process. On the contrary, verification via model checking can be applied using existing model checkers. Therefore, model checking can serve as a technique to detect non-conformance between the AC system and its specifications (e.g. secure inter-operation) as efficiently as possible. It is also noteworthy that when using model checking it is feasible to perform a security analysis of a system. Security analysis generalizes safety analysis since with security analysis we can study not only safety, but also several other interesting properties (e.g. mutual-exclusion) [4].

Fundamentally, a set of security requirements can be transformed into security properties (e.g. using temporal logic) and be verified on the transition states of an AC system [5]. In this paper, we elaborate on the verification of security properties in RBAC systems using formal methods; Specifically, through requirements engineering, we provide a set of formally defined security requirements as **properties** to a model checking mechanism to verify the conformance to the formal RBAC model. Figure 1 illustrates a schematic view of the methodology. Verification is a critical process well separated from the previous stages of requirements engineering, systems design, and implementation. Verification is used in the comparison of the initial conceptual system based on defined requirements to the computer representation that implements that conception, and concerned with building the system right. Specifically, it must ensure that the system does what it should, only the way it should and does not do what is should not do [6].

To the best of our knowledge, in RBAC systems, the verification of security properties is limited to the verification of Separation of Duty (SoD) constraints as presented in [7] and [8]. Therefore, we worked on the definition in temporal logic of a set of AC system properties regarding secure inter-operation that can be verified in RBAC systems. Moreover, we conducted research on formal methods for the verification of security properties based on model checking techniques and on proper techniques and tools for its automated implementation. As a result, we augmented with RBAC reasoning the technique proposed by NIST in [9] and applied it to illustrate the verification of the defined secure inter-operation properties in multi-domain RBAC policies.

The remaining of the paper is organized as follows. Section II elaborates on related works. In section III, a formal definition of the ANSI INCITS 359-2004 is briefly provided, along with our proposed modifications, predicates and the definition of its transition system. Section IV provides the specification of properties related to secure inter-operation. Implementation aspects are discussed in section V and proof of concept examples are provided in section VI. Finally, we conclude this paper in section VII.

II. Related Works

In collaborative systems each domain implements a different AC policy in its environment. Hence, an essential security requirement is to preserve secure inter-operation. Specifically, secure inter-operation requires that the principles of autonomy and security should be guaranteed, as stated in [10]. The principle of autonomy states that if an access is permitted within an individual system, it must also be permitted under secure inter-operation. On the contrary, the principle of security states that if an access is not permitted within a system, it must also be denied under secure inter-operation. In the existing literature, there are several approaches that preserve the principles of secure inter-operation in RBAC models. In [11] it proposed an integer programming (IP)-based approach for optimal resolution of the examined conflicts. In [12] an inter-domain role-mapping approach based on the least privilege principle is suggested. Research in [13] presents a protocol for secure inter-operation that is based on the idea of access paths and access path's constraint. Yet, in [14], an approach based on graph theory assures a secure collaborative environment by checking gradually for violations, which can be caused by new inter-domain role assignments. However, to the best of our knowledge, the verification of secure inter-operation is not being examined in any of the existing literature. Therefore, stemmed from the absence of related work, we elaborate on the definition of secure inter-operation properties that should be verified in an AC system that implements a global RBAC policy in a multi-domain environment. A multi-domain environment consists of individual domains where each implements an intra-domain AC policy. During a collaboration, an inter-domain AC policy is being formed, which consists of the individual intra-domain and cross-domain policies. In particular, we assume a transition system (TS) for an RBAC model and we formally define the security properties of cyclic inheritance, privilege escalation, separation of duties (SoD) and autonomy in temporal logic. When all the aforementioned apply in an RBAC system then secure inter-operation is maintained as stated in [11] and [14]. Nevertheless, the defined security properties are not bounded to multi-domain environments since their verification in single-domain systems can also guarantee its security.

Several papers have examined the automated verification of AC models and generic policies, and a number of techniques have been proposed to verify them [8], [9], [15], [16], [17], [18], [19], [20]. The great number of different techniques is mostly the result of the need for more expressive power or better performance. Several of them use a verification tool as back end. Such tools are for instance, Alloy [21], a declarative language with support of first order logic and relational calculus, NuSMV [22], a symbolic model checker that verifies temporal logic properties in a finite state system, the SPIN model checker [23] and so on and so forth. However, there are cases where AC policies are defined as ordering relations, which are further translated to Boolean satisfiability problems and applied to SAT solvers [19]. A SAT solver is a program that takes a formula in conjunctive normal form (CNF) and returns an assignment, or says none exists. These techniques can serve as a foundation for the verification of specifications of a system. A specification of a system can be defined as *"what the system is supposed to do"* [24].

For the verification of security properties, we are interested

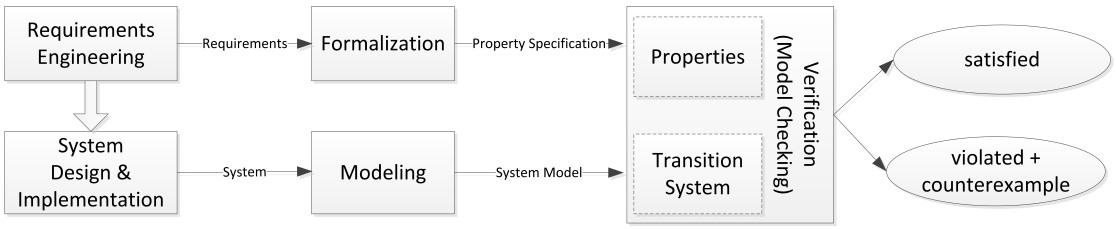

Fig. 1. System development and verification process.

in applying a technique that is able to support the verification of properties in RBAC policies. Additionally, exportation of the verified AC policies in the eXtensible Access Control Markup Language (XACML) is highly desired since it is becoming the de facto language for the description of policy rules in modern collaborative systems, as the Grid and Cloud computing paradigms [14]. Furthermore, it should be easy to express security properties regarding secure inter-operation and to successfully verify them against multiple RBAC policies that can be composed to a global security policy. Consequently, the initial security requirements (i.e. secure inter-operation) of the conceptual AC system will be verified in the implemented AC system.

We choose to apply the technique proposed in [9], which focuses on the verification of generic properties for AC models. The technique is able to cope with various types of AC models including static, dynamic, and historical. It also supports the generation of test cases to check the conformance between models and policy rules through combinatorial test array [25], and optionally generate the verified AC policies in eXtensible Access Control Markup Language (XACML) version 2.0 or 3.0. We adopt the finite state machine to describe the transitions of the authorization states, and the usage of static constraints so to adequately cover the verification of secure inter-operation properties in RBAC. The technique is to verify specified AC properties against AC models using a black-box model checking method [5]. An implementation – Access Control Policy Tool (ACPT) [26] is developed by NIST Computer Security Division in corporation of North Carolina State University.

ACPT provides graphical user interface (GUI) templates for composing AC policies and properties. Checking for conformance of AC properties and models is through the SMV (Symbolic Model Verification) model checker. In addition, ACPT provides a complete test suite generated by NIST's combinatorial testing tool ACTS [25] and an XACML policy output for the verified model. Through these four major functions, ACPT performs syntactic and semantic verifications as well as the interfacing for composing and combining AC policies. ACPT assures the efficiency of specified AC policies, and detects policy faults that leak or prohibit legitimate access privileges. Currently, ACPT provides model templates for three major AC models: static Attribute-Based AC, Multi-Leveled Security, and stated Work-Flow, and partially implements the methods described in [5]. Despite providing all the adequate functionality for the verification of AC policies, the function of RBAC reasoning regarding role hierarchies is absent. Nevertheless, we applied this model checking technique for its capabilities of defining and verifying basic RBAC rule statements and property propositions.

III. System Model

In this section, we provide a formal definition for the core and hierarchical RBAC and a set of RBAC predicates. Additionally, a formal definition of an RBAC transition system is formally defined.

A. Model Definitions

Each domain specifies its own policy in most collaborative systems to date. Hence, we separate the specification of single domain AC policies (i.e. intra-domain administration) from multiple domains collaborative policies (i.e. inter-domain administration). Both specifications follow the ANSI INCITS 359-2004 definition of RBAC [27]. We also define review functions for intra-domain and inter-domain administration. The main components [27] are defined below.

- USERS, ROLES, OPS, OBS, stands for users, roles, operations, and objects, respectively.
- UA $\subseteq$ USERS $\times$ ROLES, a many-to-many set of user-to-role assignment relations.
- PRMS = $2^{(OPS \times OBS)}$, the set of permissions.
- PA $\subseteq$ PRMS $\times$ ROLES, a many-to-many set of permission-to-role assignment relations.
- Op(p: PRMS) $\rightarrow \{op \subseteq OPS\}$, the permission to operation mapping, which gives the set of operations associated with permission p.
- Ob(p: PRMS) $\rightarrow \{ob \subseteq OBS\}$, the permission to object mapping, which gives the set of objects associated with permission p.

Henceforth, to differentiate roles, users and permissions among domains, we use the *DomainRole* format in [14] whenever is needed to, where *Domain* denotes a domain name and *Role* denotes a role name, thus, a role can be expressed as $d_{domain}r_{role}$. Such that if a role r_k belongs to a domain d_i, we write $d_i r_k$. The same applies for users and permissions.

In the presence of an intra-domain role inheritance relation, we redefine the following administrative review functions and hierarchical RBAC. A hierarchy is mathematically a partial order defining a seniority relation between roles, whereby senior roles acquire the permissions of their juniors and junior roles acquire users of their seniors [27].

- assigned users: $SU_{d_i}(d_ir_k : ROLES) \rightarrow 2^{USERS}$, the mapping of role d_ir_k onto a set of users enrolled in domain d_i. Formal definition: $SU_{d_i}(d_ir_k) = \{d_iu_t \in USERS | (d_iu_t, d_ir_k) \in UA\}$.
- assigned permissions: $SP_{d_i}(d_ir_k : ROLES) \rightarrow 2^{PRMS}$, the mapping of role d_ir_k onto a set of permissions defined in domain d_i. Formal definition: $SP_{d_i}(d_ir_k) = \{d_ip_w \in PRMS | (d_ip_w, d_ir_k) \in PA\}$.
- $RH_{d_i} \subseteq ROLES \times ROLES$ is a partial order set on $ROLES$ called the inheritance relation in domain d_i, written as $\geq$, where $d_ir_k \geq d_ir_m$ only if all permissions of d_ir_m are also permissions of d_ir_k, and all users of d_ir_k are also users of d_ir_m. Formal definition: $d_ir_k \geq d_ir_m \Rightarrow UP_{d_i}(d_ir_m) \subseteq UP_{d_i}(d_ir_k) \wedge\ UU_{d_i}(d_ir_k) \subseteq UU_{d_i}(d_ir_m)$.
- authorized users: $UU_{d_i}(d_ir_k : ROLES) \rightarrow 2^{USERS}$, the mapping of role d_ir_k onto a set of users enrolled in domain d_i in the presence of a role hierarchy defined in domain d_i. Formal definition: $UU_{d_i}(d_ir_k) = \{d_iu_t \in USERS | d_ir_m \geq d_ir_k, (d_iu_t, d_ir_m) \in UA\}$.
- authorized permissions: $UP_{d_i}(d_ir_k : ROLES) \rightarrow 2^{PRMS}$, the mapping of role d_ir_k onto a set of permissions defined in domain d_i in the presence of a role hierarchy define in domain d_i. Formal definition: $UP_{d_i}(d_ir_k) = \{d_ip_w \in PRMS | d_ir_k \geq d_ir_m, (d_ip_w, d_ir_m) \in PA\}$.

For inter-domain, we extend the aforementioned hierarchy relations and administrative review functions below:

- $RH \subseteq ROLES \times ROLES$ is a partial order set on $ROLES$ called the inheritance relation, written as $\geq$, where $d_ir_k \geq d_jr_m$ only if all permissions of d_jr_m are also permissions of d_ir_k, and all users of d_ir_k are also users of d_jr_m. Formal definition: $d_ir_k \geq d_jr_m \Rightarrow UP(d_jr_m) \subseteq UP(d_ir_k) \wedge\ UU(d_ir_k) \subseteq UU(d_jr_m)$.
- authorized users: $UU(d_ir_k : ROLES) \rightarrow 2^{USERS}$, the mapping of role d_ir_k onto a set of users enrolled in any domain in the presence of a inter-domain role hierarchy. Formal definition: $UU(d_ir_k) = UU_{d_i}(d_ir_k) \cup \{d_ju_t \in USERS | d_jr_m \geq d_ir_k, (d_ju_t, d_jr_m) \in UA\}$.
- authorized permissions: $UP(d_ir_k : ROLES) \rightarrow 2^{PRMS}$, the mapping of role d_ir_k onto a set of permissions defined in any domain in the presence of a inter-domain role hierarchy. Formal definition: $UP(d_ir_k) = UP_{d_i}(d_ir_k) \cup \{d_jp_w \in PRMS | d_ir_k \geq d_jr_m, (d_jp_w, d_jr_m) \in PA\}$.

The absence of relational operators in temporal logic, i.e. $\gg$ and $\geq$, led us to the definition of a series of appropriate predicates below.

- $IR(r_k, r_m)$ denotes the existence of an immediate (either inter or intra domain) inheritance relationship between the two roles. Formal definition: $IR(r_k, r_m) = true \Leftrightarrow r_k \gg r_m$. The operator $\gg$ means immediate inheritance relation as defined in [27].
- $MR_{d_i}(d_ir_k, d_ir_m)$ denotes that there is an (immediate or not) inheritance relationship between the two roles in the role hierarchy defined in domain d_i. Formal definition: $MR_{d_i}(d_ir_k, d_ir_m) = true \Leftrightarrow d_ir_k \geq d_ir_m$.
- $RP(r_k, r_m)$ denotes that for two roles with an immediate inheritance relation ($r_k, r_m : r_k \gg r_m$) the set of role's r_k assigned permissions is a subset of role's r_m authorized permissions. Formal definition: $RP(r_k, r_m) = true \Leftrightarrow IR(r_k, r_m) \wedge SP_{d_i}(r_k) \subseteq UP(r_m)$.
- $IB_{d_i}(d_ir_k, d_ir_m, r_n)$ denotes that for two roles d_ir_k and d_ir_m in domain d_i the set of role's r_n authorized permissions, regardless of the domain to which it belongs, includes the assigned permissions of both roles d_ir_k and d_ir_m, where r_n is a role senior to roles d_ir_k and d_ir_m. Formal definition: $IB_{d_i}(d_ir_k, d_ir_m, r_n) = true \Leftrightarrow SP_{d_i}(d_ir_k) \cup SP_{d_i}(d_ir_m) \subseteq UP(r_n) \wedge r_n \geq d_ir_k \wedge r_n \geq d_ir_m$.
- $BA(d_ir_k)$ denotes that the mapping of role d_ir_k onto the set of all its assigned and authorized permissions in domain d_i is a subset of all its permissions under the presence of an inter-domain hierarchy. Formal definition: $BA(d_ir_k) = true \Leftrightarrow UP_{d_i}(d_ir_k) \subseteq UP(d_ir_k)$.

B. Transition System

In this section, we provide the definitions of AC rule, property and transition system for an RBAC model. The definitions are based on that in [5] and redefined accordingly. Henceforth, we make use of Computation Tree Logic (CTL) for the specification of system properties. However, Linear-time Temporal Logic (LTL) can be used to express the examined properties as well. In CTL, prefixed path quantifiers assert arbitrary combinations of linear-time operators. For our purpose, we use universal path quantifier $\forall$ means *"for all paths"* and the linear temporal operators $\Box$ and $\Diamond$ means *"always"* and *"eventually"*, respectively. Furthermore, we use the temporal modalities $\forall\Box\Phi$ representing *invariantly* Φ, and $\forall\Diamond\Phi$ representing inevitably Φ, where Φ is a state formula.

Definition 1. An RBAC rule is a proposition of type "if c then d", where constraint c is a predicate expression on $(r, UP(r))$ for the permission decision d. Thus, an RBAC model can be characterized as a sequence of rules, each of which is of the form $(r, UP(r))$, where $r \in ROLES$.

Definition 2. An RBAC AC property p is a formula of type "$b \rightarrow d$", where the result of the access permission d depends on quantified predicate b on $(r, UP(r))$ mapping. In this case the $\rightarrow$ symbol expresses an implication relation.

Definition 3. A transition system TS is a tuple (S, Act, δ, i_0) where

- S is a set of states, $S = \{Permit, Deny\}$,
- Act is a set of actions, where $Act = \{(r_1, UP(r_1)), \ldots, (r_n, UP(r_n))\}$,
- δ is a transition relation where $\delta : S \times Act \rightarrow S$, and
- $i_0 \in S$ is the initial state.

An RBAC property p in Definition 2 is expressed by the proposition $p : S \times Act^2 \rightarrow S$ of TS, which can be collectively translated in terms of logical formula such that $p = (s_i * (r_1, UP(r_1)) * \ldots * (r_n, UP(r_n))) \rightarrow d$, where $p \in P$ is a set

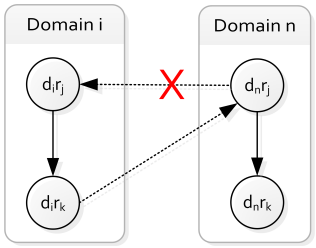

Fig. 2. Cyclic inheritance.

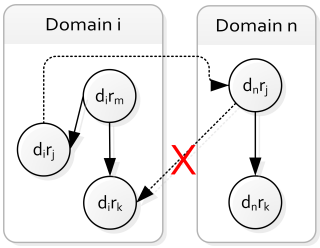

Fig. 3. Privilege escalation.

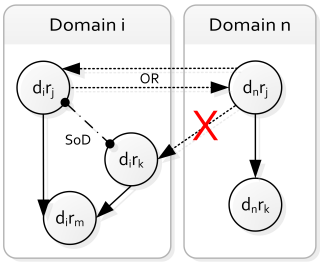

Fig. 4. Separation of duty.

of properties, and $*$ is a Boolean operator in terms of logical formulas of temporal logic such as CTL and LTL [28].

The behaviour of the system is defined by RBAC rules that function as the transition relation δ in the TS. Thus, having the RBAC AC property to be represented by a temporal logic formula p, we can represent the assertion that model TS satisfies p by $TS \vDash \forall\Box(b \rightarrow \forall\Diamond d)$. Property $\forall\Box(b \rightarrow \forall\Diamond d)$ is a response property pattern, which means that d responds to b globally (b is the cause and d is the effect) [29].

IV. Property Specification

In this section, we provide the definition of secure inter-operation properties in temporal logic that requires to be verified to ensure a secure policy for a consistent and conflict-free inter-operation. Specifically, we define the security properties of cyclic inheritance, privilege escalation, SoD, and autonomy.

During a collaboration, a violation of secure inter-operation can be caused by new immediate inter-domain role inheritance relations. As stated in [10], [11] and [14] these types of violations can be identified in RBAC approaches by searching for cyclic inheritance, privilege escalation, and violation of SoD relations in a domain. Additionally, autonomy should be preserved.

Following, we illustrate how the aforementioned properties can be verified in an RBAC policy.

A. Cyclic Inheritance Property.

In multi-domain RBAC systems, the cyclic inheritance refers to the problem that a user d_iu_t assigned to the role d_ir_k in domain d_i, is authorized for the permissions of another local role d_ir_j such as $d_ir_j \gg d_ir_k$ (see Subsection III-A for definition of $\gg$), even though d_iu_t is not directly assigned to d_ir_j in the role hierarchy of domain d_i as shown in Figure 2. Henceforth, in figures, a solid line arrow refers to an intra-domain role inheritance relation and a dashed line arrow refers to an inter-domain role inheritance relation. To detect a cyclic inheritance for a role d_ir_k, we check if the proposition $RP(d_ir_j, d_ir_k) \rightarrow \forall\Diamond Deny$ is satisfied invariantly in the TS, formally:

$$TS_{RBAC} \vDash \forall\Box(RP(d_ir_j, d_ir_k) \rightarrow \forall\Diamond Deny). \quad (1)$$

B. Privilege Escalation Property.

Privilege escalation refers to the problem that a user d_iu_t assigned to a role d_ir_j in domain d_i, is authorized for the permissions of another local role d_ir_k such as $\neg(d_ir_j \geq d_ir_k)$ (see Subsection III-A for definition of $\geq$), even though d_iu_t is not directly assigned to role d_ir_k in the role hierarchy of domain d_i (Figure 3).

To detect privilege escalation for a role d_ir_k against a role d_ir_j, we check if the proposition $(\neg MR_{d_i}(d_ir_j, d_ir_k) \wedge RP(d_ir_j, d_ir_k)) \rightarrow \forall\Diamond Deny$ is satisfied invariantly by the TS_{RBAC}, formally:

$$TS_{RBAC} \vDash \forall\Box((\neg MR_{d_i}(d_ir_j, d_ir_k) \wedge RP(d_ir_j, d_ir_k)) \rightarrow \forall\Diamond Deny). \quad (2)$$

C. Separation of Duty Property.

Separation of duty (SoD) is a fundamental security principle supported by RBAC. SoD requires two or more division between users, so that no single user can compromise security. SoD methods can be further categorized into Static SoD (SSD) and Dynamic SoD (DSD). SSD are constraints that are placed on roles at the time roles are assigned to users. When implementing SSD in role hierarchy, both inherited and directly assigned roles need to be considered. In the same manner, DSD needs to check the role hierarchy when users activate already assigned roles [30].

Verification of the SSD property is based on the following properties [30]:

Property 1. Roles r_k and r_m are mutually exclusive if neither one inherit the other directly or indirectly.

Property 2. If roles r_k and r_m are mutually exclusive then there is no other role inherits both of them.

In general, we verify SoD by role pairs [30]. Therefore, since we rely on mutual exclusion of roles specified by role

pairs, for n roles, the number of pairs that have to be verified is equal to the binomial coefficient ${}_nC_2 \equiv \begin{pmatrix} n \\ 2 \end{pmatrix} \equiv \frac{n!}{2!(n-2)!}$.

SSD is a collection of pairs (d_irs, n) in static SoD, where each d_irs is a role set and n is a number ≥ 2 such that no user is assigned to or authorized for (in the presence of a hierarchy) n or more roles from the set d_irs in each $(d_irs, n) \in SSD$. Thus, formally an SSD constraint can be verified as follows.

$$TS_{RBAC} \vDash \forall\Box((d_ir_j \in d_irs \wedge d_ir_k \in d_irs \wedge (RP(d_ir_j, d_ir_k) \vee RP(d_ir_k, d_ir_j) \vee IB_{d_i}(d_ir_j, d_ir_k, r_m))) \rightarrow \forall\Diamond Deny). \quad (3)$$

Similar to SSD, DSD has the property [30]:

Property 3. If SSD holds, then DSD is maintained.

Thus, properties 1 and 2 must be guaranteed.

D. Autonomy Property.

In addition to the security principle, autonomy should also be preserved for secure inter-operation. Maintaining the autonomy of all collaborative domains is a key requirement of the policy for inter-operation. However, access of inter-operation may be significantly reduced or even not authorized at all if the autonomy of individual domains is over addressed. Therefore, balancing autonomy and interoperability might be considered [11]. In almost any collaborative environment, it is not permissible to violate any domain's security policy. However, some domains may be willing to compromise their autonomy for the sake of establishing more interoperability, provided that autonomy loss remains within acceptable limits. Specifically, when using an RBAC policy integration framework, a violation in the autonomy of a domain may occur because of induced SoD constraints, as described in [11]. An induced SoD constraint is a SoD constraint between two intra-domain roles (e.g. d_1r_1 and d_1r_2) which do not conflict with each other in their original domain's RBAC policy. In a multi-domain system such a SoD constraint will deny concurrent access on roles d_1r_1 and d_1r_2, thus, reducing the autonomy in the original domain. Nevertheless, the autonomy principle can be verified by checking if all the assigned and authorized permissions of a role d_ir_k in a domain d_i are preserved for inter-operation.

The autonomy principle can be verified by checking if all the assigned and authorized permissions of a role d_ir_k in a domain d_i are preserved during inter-operation, formally:

$$TS_{RBAC} \vDash \forall\Box(BA(d_ir_k) \rightarrow \forall\Diamond Permit). \quad (4)$$

V. Implementation

This section discusses aspects of the implemented technique. The technique described in [9] is unable to specify role hierarchies for RBAC policies because it is not geared for RBAC models. To specify role hierarchies, we propose a role-to-role mapping algorithm derived from the graph theory in terms of Definition 1. When defining a role hierarchy $r_k \geq r_m$, r_k and r_m are assigned to permissions $PRMS_k$ and $PRMS_m$, respectively. In turn, we generate additional rules according to Definition 1, in addition to the initial rules that map roles and their assigned permissions. For example, regarding the previously mentioned roles r_k and r_m, an additional rule is generated automatically to record the r_k's inheritance of permission $PRMS_m$. In this way, a reasoning for RBAC hierarchies is introduced, which depicts the role hierarchy $r_k \geq r_m$ (i.e. role r_k is a senior of role r_m). As role hierarchies are represented by sparse graphs, we use linked lists instead of matrices for implementing role hierarchies since, in the examined case, the former are more efficient regarding performance. Formal implementation aspects are listed below [14].

1) $G = (V, E)$ is a directed graph of inter-domain role hierarchy. The graph consists of a finite, non-empty set of role vertices $V \subseteq ROLES$ and a set of edges E. Each edge is an ordered pair (d_ir_m, d_jr_n), of role vertices that indicates the relation: $d_ir_m \geq d_jr_n$.
2) A path in a G graph is a sequence of edges (d_ir_1, d_ir_2), (d_ir_2, d_ir_3), ..., (d_ir_{n-1}, d_ir_n) with length n-1 connecting from role vertex d_ir_1 to role vertex d_ir_n. A path represents indirect inheritance relations between role vertex d_ir_1 and d_ir_n.
3) An adjacency list representation for graph $G = (V, E)$ is an array L of $|V|$ lists, one for each role vertex in V. For each role vertex d_ir_m there is a pointer $L_{d_ir_m}$ to a linked list containing all the role vertices that are adjacent to d_ir_m. A linked list is terminated by a nil pointer. We refer to the adjacency list of a graph G as A_G.
4) $G^* = (V, E^*)$ is the transitive closure of the graph $G = (V, E)$ where E^* contains an edge (u, v) if and only if G contains a path from u to v. The applied transitive closure computation algorithm is based on [31] and [32] (i.e. strong components) with a time complexity of $O(|V||E|)$. We refer T_G as the transitive closure list of a directed graph $G = (V, E)$ with adjacency list A_G.

	Algorithm Rule and property creation algorithm
1:	**procedure** ITERATOR_SKELETON (T_G)
2:	**for all** *vertex* $dr_i \in T_G$
3:	**for all** *adjacent vertex* dr_j
4:	*//Generate the required rule or property*
5:	**end procedure**

Fig. 5. Iterator skeleton procedure.

To specify role hierarchies according to Definition 1, we compute the transitive closure list T_G from the adjacency list A_G of graph $G = (V, E)$ built from RBAC rules. The result is generated by an iteration on the vertices of the transitive closure list T_G. And using iteration for the vertices to generate AC properties for verification as defined in Definition 2. The pseudo code in Figure 5 procedure illustrates the iteration procedure.

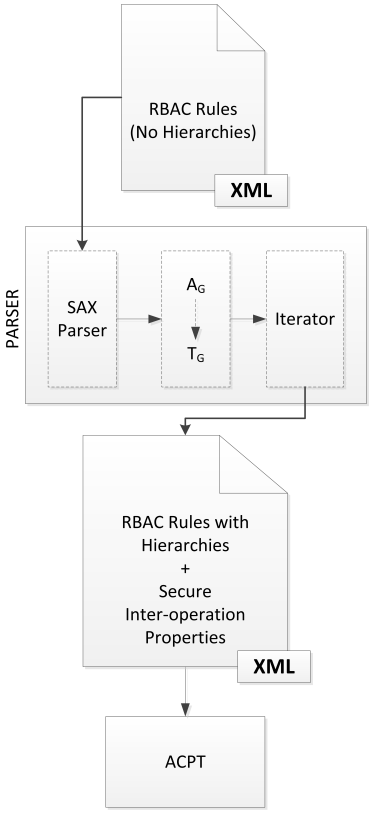

Fig. 6. The proposed tool chain.

Specifically, SAX stream API is employed as the parser to load an RBAC policy in XML (e.g. created by ACPT [26]) and produce the adjacency list A_G. In turn, the T_G is computed from the A_G. Next, a new XML file is created by iterating the vertices of T_G using the iterator in Figure 5. The XML includes both the RBAC rules with hierarchies, and properties to be verified. Figure 6 illustrates the described tool chain for the aforementioned process.

We further elaborate on the creation and computation of the A_G and T_G, respectively, through an example. Let's assume the multi-domain AC policy depicted in Figure 7 that allows collaboration between domain d_1 and domain d_2. Henceforth, in figures, a solid line refers to a permission-role assignment. Domain d_1 has the following roles: d_1r_a, d_1r_b, d_1r_c, d_1r_d and d_1r_e. Role d_1r_a inherits all permissions of d_1r_b which further inherits d_1r_e. Role d_1r_c inherits all permissions of d_1r_d which further inherits d_1r_e. A SSD relation is specified for d_1r_b and d_1r_c meaning that these roles cannot be assigned to the same user simultaneously. Domain d_2 has the following roles: d_2r_f and d_2r_g. Role d_2r_f inherits all permissions of d_2r_g. The multi-domain AC policy defines the following inter-domain inheritance relationships between domains d_1 and d_2: *i)* Role d_1r_b inherits role d_2r_g, *ii)* Role d_2r_g inherits role d_1r_c.

The A_G and T_G list representations are created, as listed below, and can used as an input to our proposed tool chain for further manipulation from the iterator procedure. In our implementation, we utilise functionalities provided by the BOOST C++ libraries [33]. Specifically, we use the *adjacency_list* class, which implements a generalized adjacency list graph structure for A_G and the *transitive_closure()* function, which transforms the input graph *G* into the transitive closure graph. In the following examples, A_G and T_G represents the adjacency and transitive closure lists of the multi-domain AC policy depicted in Figure 7. $A_{G_{d_1}}$ and $T_{G_{d_1}}$ represents the adjacency and transitive closure lists of the AC policy in domain 1. The latter are utilised as a requirement for verifying the principle of autonomy between the original domain and collaborative environment.

$$A_G = \begin{cases} Vertex & Linked\,list \\ d_1r_a: & d_1r_b \rightarrow nil \\ d_1r_b: & d_1r_e \rightarrow d_2r_g \rightarrow nil \\ d_1r_c: & d_1r_d \rightarrow nil \\ d_1r_d: & d_1r_e \rightarrow nil \\ d_1r_e: & nil \\ d_2r_f: & d_2r_g \rightarrow nil \\ d_2r_g: & d_1r_c \rightarrow nil \end{cases}$$

$$A_{G_{d_1}} = \begin{cases} Vertex & Linked\,list \\ d_1r_a: & d_1r_b \rightarrow nil \\ d_1r_b: & d_1r_e \rightarrow nil \\ d_1r_c: & d_1r_d \rightarrow nil \\ d_1r_d: & d_1r_e \rightarrow nil \\ d_1r_e: & nil \end{cases}$$

$$T_G = \begin{cases} Vertex & Linked\,list \\ d_1r_a: & d_1r_b \rightarrow d_1r_c \rightarrow d_1r_d \rightarrow d_1r_e \rightarrow d_2r_g \rightarrow nil \\ d_1r_b: & d_1r_c \rightarrow d_1r_d \rightarrow d_1r_e \rightarrow d_2r_g \rightarrow nil \\ d_1r_c: & d_1r_d \rightarrow d_1r_e \rightarrow nil \\ d_1r_d: & d_1r_e \rightarrow nil \\ d_1r_e: & nil \\ d_2r_f: & d_2r_g \rightarrow d_1r_c \rightarrow d_1r_d \rightarrow d_1r_e \rightarrow nil \\ d_2r_g: & d_1r_c \rightarrow d_1r_d \rightarrow d_1r_e \rightarrow nil \end{cases}$$

$$T_{G_{d_1}} = \begin{cases} Vertex & Linked\,list \\ d_1r_a: & d_1r_b \rightarrow d_1r_e \rightarrow nil \\ d_1r_b: & d_1r_e \rightarrow nil \\ d_1r_c: & d_1r_d \rightarrow d_1r_e \rightarrow nil \\ d_1r_d: & d_1r_e \rightarrow nil \\ d_1r_e: & nil \end{cases}$$

VI. Model Checking

In this section, we provide two proof of concept examples for the verification of the properties defined in section IV. The examples were implemented in the NuSMV model checker [22]. The relevant code was based on the NuSMV source code generated by ACPT [26], and recoded accordingly in order to depict only the portions of code required by an RBAC model. The NuSMV code of all examples is provided in the Appendix section.

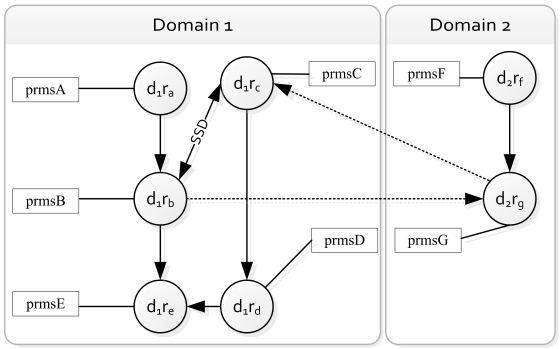

Fig. 7. Case #1 of a multi-domain AC policy between domains d_1 and d_2.

A. Verification of Privilege Escalation and SSD Properties

Let us assume the multi-domain AC policy depicted in Figure 7. In the global security policy, a privilege escalation violation can be identified since roles d_1r_a and d_1r_b are able to be authorized with the permissions of roles d_1r_c and d_1r_d, which is not permissible in domain d_1. As a proof of concept, all the following satisfaction relations are evaluated as *false* in NuSMV.

- $TS_{RBAC} \vDash \forall\Box((\neg MR_{d_1}(d_1r_a, d_1r_c) \wedge RP(d_1r_a, d_1r_c)) \rightarrow \forall\Diamond Deny)$
- $TS_{RBAC} \vDash \forall\Box((\neg MR_{d_1}(d_1r_a, d_1r_d) \wedge RP(d_1r_a, d_1r_d)) \rightarrow \forall\Diamond Deny)$
- $TS_{RBAC} \vDash \forall\Box((\neg MR_{d_1}(d_1r_b, d_1r_c) \wedge RP(d_1r_b, d_1r_c)) \rightarrow \forall\Diamond Deny)$
- $TS_{RBAC} \vDash \forall\Box((\neg MR_{d_1}(d_1r_b, d_1r_d) \wedge RP(d_1r_b, d_1r_d)) \rightarrow \forall\Diamond Deny)$

Likewise an SSD violation occurs since any user u assigned to role d_1r_b is also authorized for role d_1r_c, which is not permissible in domain d_1 due to the SSD constraint. Thus, the satisfaction relation $TS_{RBAC} \vDash \forall\Box((d_1r_b \in (d_1r_b, d_1r_c) \wedge d_1r_c \in (d_1r_b, d_1r_c) \wedge (RP(d_1r_b, d_1r_c) \vee RP(d_1r_c, d_1r_b))) \rightarrow \forall\Diamond Deny)$ is evaluated in NuSMV as *false*. In the latter satisfaction relation, BI is omitted since there is not any role senior to both roles d_1r_b and d_1r_c.

B. Verification of Cyclic Inheritance and Autonomy Properties

In Figure 8, we assume a multi-domain AC policy that allows the collaboration between domains d_1 and d_2. Domain d_1 has the following roles: d_1r_a and d_1r_b where d_1r_a inherits all permissions of d_1r_b. In turn, domain d_2 has the following roles: d_2r_c and d_2r_d where d_2r_c inherits all permissions of d_2r_d. Furthermore, a collaboration is depicted between domains $d_1$and d_2 where it is instantiated by the addition of two inter-domain role assignments: d_1r_b inherits d_2r_c and d_2r_c inherits d_1r_a.

It is straightforward that the aforementioned inter-domain policy leads to a cyclic inheritance violation since it is permitted to role d_1r_b to access the permissions of its senior role d_1r_a, in domain d_1, through role d_2r_c in domain d_2. Therefore, also the verification of the satisfaction relation: $TS_{RBAC} \vDash \forall\Box(RP(d_1r_a, d_1r_b) \rightarrow \forall\Diamond Deny)$ in NuSMV is evaluated as *false*.

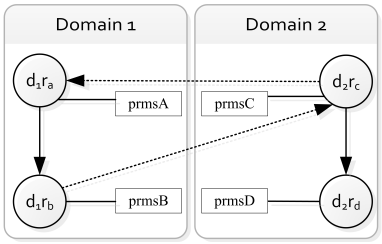

Fig. 8. Case #2 of a multi-domain AC policy between domains d_1 and d_2.

Furthermore, we proceed with the verification of the autonomy property in the aforementioned RBAC policy, as this is defined in section IV. At first, we appoint the authorized permissions for each role in each individual domain without taking into account any inter-operation between the two domains. Thus, in domain d_1, role d_1r_a is authorized for permissions $prmsA$ and $prmsB$, and role d_1r_b is authorized for permission $prmsB$. Additionally, in domain d_2, role d_2r_c is authorized for permissions $prmsC$ and $prmsD$, and role d_2r_d is authorized for permission $prmsD$. Therefore, in order to verify the autonomy property under secure inter-operation, we check the following satisfaction relations in NuSMV, which are evaluated as $true$, thus, meaning that the autonomy principle is maintained in both domains.

- $TS_{RBAC} \vDash \forall\Box(BA(d_1r_a) \rightarrow \forall\Diamond Permit)$
- $TS_{RBAC} \vDash \forall\Box(BA(d_1r_b) \rightarrow \forall\Diamond Permit)$
- $TS_{RBAC} \vDash \forall\Box(BA(d_2r_c) \rightarrow \forall\Diamond Permit)$
- $TS_{RBAC} \vDash \forall\Box(BA(d_2r_d) \rightarrow \forall\Diamond Permit)$

VII. Conclusion

In this paper, we proposed the verification of secure inter-operation properties for RBAC systems using formal verification methods (i.e. model checking). We have pinpointed that requirements engineering can be used for the definition of security related properties and that their verification can be performed using model checking processes upon a formally defined model of the designed or implemented system. Specifically, we proposed an efficient verification technique to detect non-conformance between an RBAC system and its initial security specifications. We have partially redefined the ANSI INCITS 359-2004 in the context of intra-domain and inter-domain administration. A TS to model RBAC and four security properties regarding secure inter-operation were defined, viz. cyclic inheritance, privilege escalation, SoD constraints and autonomy. These properties should be verified in RBAC policies when a global security policy is enforced to assure secure inter-operation. Nevertheless, the defined security properties can also be verified in single-domain environments to assure a single domain's security. The definitions of the properties were provided in temporal logic. To the best of our knowledge, there are not any equivalent formal specifications and verification of RBAC security properties. A proposed parser managed to tackle the absence of RBAC reasoning

regarding role hierarchies in the applied technique. Through a series of examples that depict global security policies between domains using RBAC policies, the a posteriori enforcement of the properties was demonstrated. It is noteworthy that the proposed technique is not limited to the aforementioned security properties. Therefore, any security related requirement properly defined in temporal logic can be verified in an RBAC system using the proposed technique, thus, leading to an efficient and flexible security analysis approach for RBAC systems.

ACKNOWLEDGMENT

This work has been (partially) funded by the Research Committee of the University of Macedonia, Greece.

REFERENCES

[1] R. S. Sandhu and P. Samarati, "Access control: Principles and practice," *IEEE Communications Magazine*, vol. 32, pp. 40–48, 1994.

[2] S. Capitani di Vimercati, S. Foresti, and P. Samarati, "Authorization and access control," in *Security, Privacy, and Trust in Modern Data Management*, ser. Data-Centric Systems and Applications, M. Petkovic and W. Jonker, Eds. Springer Berlin Heidelberg, 2007, pp. 39–53.

[3] K. Heljanko, "Model checking based software verification," 2006. [Online]. Available: http://iplu.vtt.fi/digitalo/modelchecking.pdf

[4] N. Li and M. Tripunitara, "Security analysis in role-based access control," *ACM Transactions on Information and System Security (TISSEC)*, vol. 9, no. 4, pp. 391–420, 2006.

[5] V. C. Hu, D. R. Kuhn, T. Xie, and J. Hwang, "Model checking for verification of mandatory access control models and properties," *International Journal of Software Engineering and Knowledge Engineering*, vol. 21, no. 1, pp. 103–127, 2011.

[6] R. Stevens, P. Brook, K. Jackson, and S. Arnold, *Systems engineering: Coping with complexity*. Pearson Education, 1998.

[7] A. Schaad and J. D. Moffett, "A lightweight approach to specification and analysis of role-based access control extensions," in *Proceedings of the seventh ACM symposium on Access control models and technologies*, ser. SACMAT '02. New York, NY, USA: ACM, 2002, pp. 13–22.

[8] F. Hansen and V. Oleshchuk, "Conformance checking of RBAC policy and its implementation," in *Information Security Practice and Experience*, ser. Lecture Notes in Computer Science, R. Deng, F. Bao, H. Pang, and J. Zhou, Eds. Springer Berlin / Heidelberg, 2005, vol. 3439, pp. 144–155.

[9] V. C. Hu, D. R. Kuhn, and T. Xie, "Property verification for generic access control models," in *Proceedings of the 2008 IEEE/IFIP International Conference on Embedded and Ubiquitous Computing - Volume 02*, ser. EUC '08. Washington, DC, USA: IEEE Computer Society, 2008, pp. 243–250.

[10] L. Gong and X. Qian, "Computational issues in secure interoperation," 1996.

[11] B. Shafiq, J. B. D. Joshi, E. Bertino, and A. Ghafoor, "Secure interoperation in a multidomain environment employing RBAC policies," *IEEE Trans. on Knowl. and Data Eng.*, vol. 17, no. 11, pp. 1557–1577, 2005.

[12] L. Chen and J. Crampton, "Inter-domain role mapping and least privilege," in *SACMAT '07: Proceedings of the 12th ACM symposium on Access control models and technologies*. New York, NY, USA: ACM, 2007, pp. 157–162.

[13] M. Shehab, E. Bertino, and A. Ghafoor, "SERAT: Secure role mapping technique for decentralized secure interoperability," in *SACMAT '05: Proceedings of the tenth ACM symposium on Access control models and technologies*. New York, NY, USA: ACM, 2005, pp. 159–167.

[14] A. Gouglidis and I. Mavridis, "domRBAC: An access control model for modern collaborative systems," *Computers & Security*, vol. 31, no. 4, pp. 540 – 556, 2012. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0167404812000144

[15] W. Sun, R. France, and I. Ray, "Rigorous analysis of uml access control policy models," in *Policies for Distributed Systems and Networks (POLICY), 2011 IEEE International Symposium on*. IEEE, 2011, pp. 9–16.

[16] K. Jayaraman, V. Ganesh, M. Tripunitara, M. Rinard, and S. Chapin, "Automatic error finding in access-control policies," in *Proceedings of the 18th ACM conference on Computer and communications security*, ser. CCS '11. New York, NY, USA: ACM, 2011, pp. 163–174.

[17] K. Fisler, S. Krishnamurthi, L. A. Meyerovich, and M. C. Tschantz, "Verification and change-impact analysis of access-control policies," in *Proceedings of the 27th international conference on Software engineering*, ser. ICSE '05. New York, NY, USA: ACM, 2005, pp. 196–205.

[18] H. Hu and G. Ahn, "Enabling verification and conformance testing for access control model," in *Proceedings of the 13th ACM symposium on Access control models and technologies*, ser. SACMAT '08. New York, NY, USA: ACM, 2008, pp. 195–204.

[19] G. Hughes and T. Bultan, "Automated verification of access control policies using a SAT solver," *Int. J. Softw. Tools Technol. Transf.*, vol. 10, no. 6, pp. 503–520, Oct. 2008.

[20] S. Jha, N. Li, M. Tripunitara, Q. Wang, and W. Winsborough, "Towards formal verification of role-based access control policies," *IEEE Transactions on Dependable and Secure Computing*, vol. 5, pp. 242–255, 2008.

[21] Alloy, "A language and tool for relational models, http://alloy.mit.edu/alloy/." [Online]. Available: http://alloy.mit.edu/alloy/

[22] NuSMV, "A new symbolic model checker, http://nusmv.fbk.eu/." [Online]. Available: http://nusmv.fbk.eu/

[23] SPIN, "The SPIN model checker, http://spinroot.com/spin/." [Online]. Available: http://spinroot.com/spin/

[24] L. Lamport, *Specifying Systems: The TLA+ Language and Tools for Hardware and Software Engineers*, 1st ed. Addison-Wesley Professional, 2002.

[25] NIST, "Combinatorial and pairwise testing, http://csrc.nist.gov/groups/sns/acts/," 2012. [Online]. Available: http://csrc.nist.gov/groups/SNS/acts/

[26] J. Hwang, T. Xie, V. Hu, and M. Altunay, "ACPT: A tool for modeling and verifying access control policies," in *Proceedings of the 2010 IEEE International Symposium on Policies for Distributed Systems and Networks*, ser. POLICY '10. Washington, DC, USA: IEEE Computer Society, 2010, pp. 40–43.

[27] ANSI, *ANSI INCITS 359-2004, Role Based Access Control*, ANSI Std., 2004.

[28] C. Baier and J.-P. Katoen, *Principles of Model Checking*. The MIT Press, 2008.

[29] SAnToS Laboraroty, "Spec patterns, responce property pattern, http://patterns.projects.cis.ksu.edu/," 2012. [Online]. Available: http://patterns.projects.cis.ksu.edu/documentation/patterns/response.shtml

[30] D. F. Ferraiolo, D. R. Kuhn, and R. Chandramouli, *Role-Based Access Control*. Artech House, Inc., 2003.

[31] E. Nuutila, "Efficient transitive closure computation in large digraphs," Ph.D. dissertation, Acta Polytechnica Scandinavica, Helsinki University of Technology, 1995.

[32] P. Purdom, "A transitive closure algorithm," *BIT Numerical Mathematics*, vol. 10, pp. 76–94, 1970, 10.1007/BF01940892.

[33] Boost, "Boost c++ libraries." [Online]. Available: http://www.boost.org/

APPENDIX: NUSMV CODE

Example of privilege escalation and SSD properties as depicted in subsection VI-A.

```
MODULE main
VAR
 ROLES : {dummy, D1Ra, D1Rb, D1Rc, D1Rd, D1Re,
  D2Rf, D2Rg};
 OBJECT : {dummy, ObjA, ObjB, ObjC, ObjD,
  ObjE, ObjF, ObjG};
 OPERATION : {dummy, read};
 RBAC_InterDomain : RBAC_InterDomain(ROLES,
 OBJECT, OPERATION);
ASSIGN
 next (ROLES):=ROLES; next (OBJECT):=OBJECT;
MODULE RBAC_InterDomain(ROLES,OBJECT,OPERATION)
```

```
VAR decision : {Permit, Deny};
ASSIGN
 init (decision):=Deny; next (decision):=case
ROLES = D1Ra & OBJECT = ObjA &
OPERATION = read : Permit ;
ROLES = D1Ra & OBJECT = ObjB &
OPERATION = read : Permit ;
ROLES=D1Ra&OBJECT=ObjC&OPERATION=read:Permit;
ROLES=D1Ra&OBJECT=ObjD&OPERATION=read:Permit;
ROLES=D1Ra&OBJECT=ObjE&OPERATION=read:Permit;
ROLES = D1Ra & OBJECT = ObjG &
OPERATION = read : Permit ;
ROLES = D1Rb & OBJECT = ObjB &
OPERATION = read  : Permit ;
ROLES = D1Rb & OBJECT = ObjE &
OPERATION = read  : Permit ;
ROLES = D1Rb & OBJECT = ObjD &
OPERATION = read  : Permit ;
ROLES = D1Rb & OBJECT = ObjC &
OPERATION = read  : Permit ;
ROLES = D1Rb & OBJECT = ObjG &
OPERATION = read  : Permit ;
ROLES = D1Re & OBJECT = ObjE &
OPERATION = read : Permit ;
ROLES = D1Rd & OBJECT = ObjD &
OPERATION = read : Permit ;
ROLES = D1Rd & OBJECT = ObjE &
OPERATION = read : Permit ;
ROLES = D1Rc & OBJECT = ObjC &
OPERATION = read : Permit ;
ROLES = D1Rc & OBJECT = ObjD &
OPERATION = read : Permit ;
ROLES = D1Rc & OBJECT = ObjE &
OPERATION = read : Permit ;
ROLES = D2Rf & OBJECT = ObjF &
OPERATION = read : Permit ;
ROLES = D2Rf & OBJECT = ObjG &
OPERATION = read : Permit ;
ROLES = D2Rf & OBJECT = ObjC &
OPERATION = read : Permit ;
ROLES = D2Rf & OBJECT = ObjD &
OPERATION = read : Permit ;
ROLES = D2Rf & OBJECT = ObjE &
OPERATION = read : Permit ;
ROLES = D2Rg & OBJECT = ObjG &
OPERATION = read : Permit ;
ROLES = D2Rg & OBJECT = ObjC &
OPERATION = read : Permit ;
ROLES = D2Rg & OBJECT = ObjD &
OPERATION = read : Permit ;
ROLES = D2Rg & OBJECT = ObjE &
OPERATION = read : Permit ;
1 : Deny; esac;
-- SSD verification for any user
-- and SSD(D1Rb, D1Rc)
SPEC AG ( ((ROLES = D1Rb) & (OBJECT = ObjC) &
  (OPERATION = read)) | ((ROLES = D1Rc) &
  (OBJECT = ObjB) & (OPERATION = read)) ->
  AF  decision =  Deny)
-- Privilege escalation between D1Ra and
SPEC AG ( (ROLES = D1Ra) & (OBJECT = ObjC) &
(OPERATION = read)  -> AF  decision =  Deny)
```

Example of cyclic inheritance and autonomy properties as depicted in subsection VI-B

```
MODULE main
VAR
 ROLES : {dummy, D1Ra,D1Rb, D2Rc, D2Rd};
 OBJECTS : {dummy, objA, objB, objC, objD};
 OPERATIONS : {dummy, read};
 RBAC_InterDomain : RBAC_InterDomain ( ROLES,
  OBJECTS, OPERATIONS);
ASSIGN
 next (ROLES):=ROLES; next (OBJECTS):=OBJECTS;
 next (OPERATIONS) := OPERATIONS ;
MODULE RBAC_InterDomain(ROLES, OBJECTS,
OPERATIONS)
VAR decision : {Permit, Deny};
ASSIGN
init (decision):=Deny; next (decision):=case
ROLES=D1Ra&OBJECTS=objA&OPERATIONS=read:Permit;
ROLES=D1Ra&OBJECTS=objB&OPERATIONS=read:Permit;
ROLES=D1Ra&OBJECTS=objC&OPERATIONS=read:Permit;
ROLES=D1Rb&OBJECTS=objB&OPERATIONS=read:Permit;
ROLES=D1Rb&OBJECTS=objC&OPERATIONS=read:Permit;
ROLES=D1Rb&OBJECTS=objA&OPERATIONS=read:Permit;
ROLES=D2Rc&OBJECTS=objC&OPERATIONS=read:Permit;
ROLES=D2Rc&OBJECTS=objD&OPERATIONS=read:Permit;
ROLES=D2Rc&OBJECTS=objA&OPERATIONS=read:Permit;
ROLES=D2Rc&OBJECTS=objB&OPERATIONS=read:Permit;
ROLES=D2Rd&OBJECTS=objD&OPERATIONS=read:Permit;
ROLES=D1Ra&OBJECTS=objD&OPERATIONS=read:Permit;
ROLES=D1Rb&OBJECTS=objD&OPERATIONS=read:Permit;
  1 : Deny; esac;
-- Cyclic inheritance property
SPEC AG ((ROLES = D1Rb) & (OBJECTS = objA) &
 (OPERATIONS = read) -> AF  decision =  Deny)
-- Autonomy properties
SPEC AG ((ROLES = D1Ra) & (OBJECTS = objA) &
(OPERATIONS = read) -> AF  decision =  Permit)
SPEC AG ((ROLES = D1Ra) & (OBJECTS = objB) &
(OPERATIONS = read) -> AF  decision =  Permit)
SPEC AG ((ROLES = D1Rb) & (OBJECTS = objB) &
(OPERATIONS = read) -> AF  decision =  Permit)
SPEC AG ((ROLES = D2Rc) & (OBJECTS = objC) &
(OPERATIONS = read) -> AF  decision =  Permit)
SPEC AG ((ROLES = D2Rc) & (OBJECTS = objD) &
(OPERATIONS = read) -> AF  decision =  Permit)
SPEC AG ((ROLES = D2Rd) & (OBJECTS = objD) &
(OPERATIONS = read) -> AF  decision =  Permit)
```