Toward Practical Analysis for Trust Management Policy

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

ABSTRACT

Trust management is a scalable and flexible form of access control that relies heavily on delegation techniques. While these techniques may be necessary in large or decentralized systems, stakeholders need an analysis methodology and automated tools for reasoning about who will have access to their resources today as well as in the future. When an access control policy fails to satisfy the policy author's security objectives, tools should provide information that demonstrate how and why the failure occurred. Such information is useful in that it may assist policy authors in constructing policies that satisfy security objectives, which support policy authoring and maintenance. This paper presents a collection of reduction, optimization, and verification techniques useful in determining whether security properties are satisfied by RT policies. We provide proofs of correctness as well as demonstrate the degree of effectiveness and efficiency the techniques provide through empirical evaluation. While the type of analysis problem we examine is generally intractable, we demonstrate that our reduction and optimization techniques may be able to reduce problem instances into a form that can be automatically verified.

Categories and Subject Descriptors

D.2 [**Software Engineering**]: Software/Program Verification

General Terms

Security, Verification

Keywords

Trust Management, Policy, Model Checking

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

ASIACCS'09, March 10–12, 2009, Sydney, NSW, Australia.

1. INTRODUCTION

Security analysis is a critical task in the design and maintenance of access control policies. The software components that comprise critical systems often endure extensive testing if not more rigorous verification measures such as formal methods. However, policy design that fails to satisfy security properties is an equally grave hazard. Correctly configuring security systems is often very difficult. In order to demonstrate and evaluate the correctness of access control properties, it is necessary to analyze the security policy. We use the term policy to refer to mechanism configuration, and use "security property" to refer to somewhat higher-level, though still formal, security objectives.

Policy design and maintenance can benefit from automated tools and techniques for evaluating policies. Every proposed change in the policy by trusted entities may require analysis prior to committing changes. It is critical to verify that the policy as stated meets one's intended policy objectives. Closely related is impact change analysis [4] where a policy author proposes a policy change, but must demonstrate that the change will not violate security objectives before committing it.

Without analysis techniques and tools, it may be difficult to claim that a particular access control policy exhibits desired characteristics. Policy authors and stakeholders seek effective, yet usable, techniques and tools in order to demonstrate such properties [9, 10, 12, 18] as availability, *e.g.*, will Alice always have access to resource R, and safety, *e.g.*, will access to resource R always be limited to some static set of users? One particularly useful, yet expensive, type of analysis is role containment [12]. Role containment asks whether every member of one role is contained within another role, which is useful in verifying safety properties. For example, is everyone with access to the database an employee? Development of techniques to reason about role containment properties is significant as it subsumes other types of security analysis.

Writing access control policies that reflect the author's intention is a challenging task for several reasons. First, the people who are charged with constructing or maintaining security in applications and systems are not always security experts. Second, even when expertise is not an issue, manually verifying policies can be a tedious and error-prone process, particularly for large or complex policies. Third, simply knowing that a security property fails to hold in a particular policy is not sufficient. Counterexamples are policy states in which the policy fails to satisfy one or more security properties, and are valuable because they provide the

policy author insight as to how the policy might fail so that it may be quickly corrected. Finally, there is a significant gap between theoretical analysis and practical techniques that needs to narrow before security analysis can be widely accepted. Part of this gap is due to a lack of tool support that leverages the theory but does not require the user to necessarily be an expert. The other part of this gap is due to a lack of evaluation of theoretical techniques. While a general technique may be inefficient, could there be usable techniques that address efficiency issues in the verification of specific policies? We provide the reader not only a proof of correctness of our techniques, but an assessment of efficiency over a set of test cases.

We present a collection of techniques for policy reduction and optimization to improve the ability of model-checking-based techniques to analyze containment queries. We believe our reduction and optimization techniques may be able to reduce such query instances into a form that can be verified. These techniques can be automated and associated with model checking to determine if a given policy satisfies a particular security property, and always provides an example of policy failure should it exist.

The structure of this paper is as follows. Section 2 describes the RT language and the complexity of role containment analysis. Section 3 describes reductions that provide support to our verification techniques. Section 4 describes our implementation of policy model checking. Section 5 evaluates our techniques over a collection of test cases. Section 6 compares our framework with related work, and we conclude in Section 7 with our contributions and future work.

2. THE RT POLICY LANGUAGE

The role-based trust management policy language RT was designed to support highly decentralized attribute-based access control [11]. It enables resource providers to make authorization decisions about resource requesters of whom they have no prior knowledge. This is achieved by delegating authority for characterizing principals in the system to other entities that are in a better position to provide the characterization. For instance, to grant discounted service to students, a resource provider might delegate to universities the authority to identify students and delegate to accrediting boards the authority to identify universities.

A significant problem that policy authors face in this context is that of determining the extent of their exposure through delegation to untrusted or semi-trusted principals. The security analysis problem [12] in this context consists of determining whether changes made by principals that are not fully trusted could cause certain policy objectives to become violated. One example of the problem would ask whether anyone outside the organization could, because of changes made by principals outside the inner circle, gain access to the organization's sensitive data. In this section, we summarize RT and the security analysis of it.

2.1 Overview of RT Syntax & Semantics

In RT, all principals are able to define their own roles and to assign other principals to them. A role owner can do this by issuing cryptographically verifiable, role-defining statements of a few different types. To her own roles, she can add a specific principal or she can add the members of another role. In the latter case, she is delegating authority to the owner of the other role. Delegating authority to another owner can occur in two ways. First she can identify a specific principal as a delegate. Secondly, she can identify a collection of principals as delegates such that these principals are grouped by a role. Set intersection and union are also both available for role definition.

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

The basic RT language consists of four types of statements as shown by Figure 1 [12]. Type I statements directly introduce individual principals to roles. For example, *Alice.friend* $\leftarrow$ *Bob* identifies Bob as a friend of Alice. A given principal must appear in a Type I statement if it is to be contained by any role. Type II statements express a form of delegation that describes the implication that if principals are in one role, then they are in another role as well. For example, the statement *Alice.friend* $\leftarrow$ *Bob.friend* describes the situation in which if a principal is a friend of Bob, then they are also a friend of Alice. Type III statements provide a mechanism to delegate to a set of principals identified by membership in a role, rather than a single principal. For example, the statement *Alice.friend* $\leftarrow$ *Bob.family.friend* says that any friend of Bob's family is also a friend of Alice. It does not imply that Alice's friends include Bob's family. Finally, Type IV statements introduce intersection such that a principal must be in the two given roles in order to be included. For example, *Alice.friend* $\leftarrow$ *Bob.friend* $\cap$ *Carl.friend* says that only those principals who are both Bob's friends and Carl's friends are introduced into the set of Alice's friends. Note that disjunction is provided through multiple statements defining the same role.

Policy classes [12] are defined based on the type of statements included in a policy. $RT[\,]$ consists of simple member and inclusion statements. $RT[\twoheadleftarrow]$ includes $RT[\,]$ and introduces linking inclusion statements. $RT[\cap]$ also includes $RT[\,]$ and introduces intersection inclusion statements. Finally, $RT[\twoheadleftarrow, \cap]$ is comprised of four types of policy statements.

For any given RT policy statement, we call the left hand side of the arrow the *defined role* and the right hand side of the arrow the *role expression*. In Type III statements, the right hand side of the arrow is called a *linked role expression*, the *base-linked role* is the role that contains the principals upon which the linking role is applied, and the *sub-linked role* is the role produced by the linking role. For example, in $A.r \leftarrow B.r_1.r_2$, $B.r_1.r_2$ is a linked role expression, $B.r_1$ is the base-linked role, and every role of the form $X.r_2$ in which X is a member of $B.r_1$ is a sub-linked role. A *policy state* $\mathcal{P}$ is a set of statements. The set of role names in $\mathcal{P}$ is given by $\mathsf{Names}(\mathcal{P})$; the set of principals in $\mathcal{P}$ is given by $\mathsf{Principals}(\mathcal{P})$; the set of roles of $\mathcal{P}$ $\mathsf{Roles}(\mathcal{P}) = \{A.r \mid A \in \mathsf{Principals}(\mathcal{P}) \wedge r \in \mathsf{Names}(\mathcal{P})\}$. The *restriction of a policy* $\mathcal{P}$ to a given set of roles τ is given by $\mathcal{P}|_\tau = \{A.r \leftarrow e \in \mathcal{P} \mid A.r \in \tau\}$.

Definition 1 (Semantics of RT). *The semantics of RT policy $\mathcal{P}$ is by the least fixpoint of the following function over functions $\pi : \mathsf{Roles}(\mathcal{P}) \rightarrow \wp(\mathsf{Principals}(\mathcal{P}))$, in which $\wp$*

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

Figure 1: RT Statements

denotes powerset:
For all $A.r \in \mathsf{Roles}(\mathcal{P})$, $T_{\mathcal{P}}(\pi)[\![A.r]\!] = \{D \mid$
$A.r \leftarrow D \in \mathcal{P} \vee$
$(A.r \leftarrow B.r_1 \in \mathcal{P} \wedge D \in \pi[\![B.r_1]\!]) \vee$
$(A.r \leftarrow B.r_1.r_2 \in \mathcal{P} \wedge \exists Z.Z \in \pi[\![B.r_1]\!] \wedge D \in \pi[\![Z.r_2]\!]) \vee$
$(A.r \leftarrow B.r_1 \cap C.r_2 \in \mathcal{P} \wedge D \in \pi[\![B.r_1]\!] \wedge D \in \pi[\![C.r_2]\!])\}$
(In the above, Quite quotes ([\![and]\!]) delimit syntactic objects denoted by metavariables, such as A and r.) The least fixpoint of $T_{\mathcal{P}}$ *can be computed as follows:*

$$T_{\mathcal{P}} \uparrow^0 = \pi_0, \text{ in which for all } A.r \in \mathsf{Roles}(\mathcal{P}),\ \pi_0[\![A.r]\!] = \emptyset$$

$$T_{\mathcal{P}} \uparrow^{i+1} = T_{\mathcal{P}}(T_{\mathcal{P}} \uparrow^i) \qquad T_{\mathcal{P}} \uparrow^{\omega} = \bigcup_{i < \omega} T_{\mathcal{P}} \uparrow^i$$

As the number of principals and role names in $\mathcal{P}$ *is finite, this increasing sequence converges at a finite stage. We now define the semantics as the function* $[\![\cdot]\!]_{\mathcal{P}}$ *that given any role* $B.r_1$*, is defined by* $[\![B.r_1]\!]_{\mathcal{P}} = T_{\mathcal{P}} \uparrow^{\omega} [\![B.r_1]\!]\}$.

2.2 RT Policy Analysis

Policy analysis [12] as we consider it here examines whether the specified relationships between roles hold in all reachable policy states. We explain reachable policy states below. The relationships, called *queries*, are set containments and take the form $\varrho \sqsupseteq \lambda$ in which ϱ and λ are each either roles or explicit (constant) sets of principals. For instance, $X.u \sqsupseteq A.r$ holds if every member of $A.r$ is a member of $X.u$ in every reachable policy state $\mathcal{P}'$, *i.e.*, $[\![A.r]\!]_{\mathcal{P}'} \supseteq [\![X.u]\!]_{\mathcal{P}'}$. Queries of this form can be used to express many important security properties such as availability, safety, liveness and mutual exclusion. For instance, a safety property might be that everyone in the role that has access to the secret database is in the employee role.

In general, any policy state can evolve into any other policy state by having principals issue new policy statements and revoke old ones. In security analysis we ask whether queries hold in all policy states that differ from a given current policy state *only* by changes to roles outside some trusted set. Intuitively, this corresponds to the fact that we expect certain principals to cooperate with us in our goal of preserving certain desired security properties. Specifically we assume that to this end these principals agree not to add or remove statements defining certain roles that they control. Other roles are not be assumed to be managed in cooperation with our goals. This intuition leads [12] to the defined two sets of roles that are used to determine the reachable policy states, the set of *growth-restricted* roles $\mathcal{G}_{\mathcal{R}}$ and the set of *shrink-restricted* roles $\mathcal{S}_{\mathcal{R}}$. Such a pair is called a *restriction rule* and is denoted by $\mathcal{R} = (\mathcal{G}_{\mathcal{R}}, \mathcal{S}_{\mathcal{R}})$.

Growth-restricted roles ($\mathcal{G}_{\mathcal{R}}$) are not allowed to have new statements defining them added to the state. Shrink-restricted roles ($\mathcal{S}_{\mathcal{R}}$) are not allowed to have statements defining them removed. We write $\mathcal{P} \overset{*}{\mapsto}_{\mathcal{R}} \mathcal{P}'$ to indicate that $\mathcal{P}' \restriction_{\mathcal{G}_{\mathcal{R}}} \subseteq \mathcal{P}$ and $\mathcal{P}' \supseteq \mathcal{P} \restriction_{\mathcal{S}_{\mathcal{R}}}$. It is important to note that these restrictions are not actually enforced. They are simply assumptions under which the analysis is performed. Their presence enables the analysis to provide us with assurances of things like, "So long as the people I trust do not make policy changes without first running the analysis, only company employees will be able to access the secret database."

Queries of certain restricted forms can be analyzed and verified in polynomial time. These include queries in which at least one of ϱ and λ is an explicit set. They also include situations in which only Type I and Type II statements are allowed in the policy state. However, when both ϱ and λ are roles and all forms of statements are allowed, the decision problem is EXPTIME complete [20]. This is unfortunate because such properties are extremely useful. For instance, suppose we want to determine whether only employees could ever get access to a company's secret database. This can be determined efficiently if the set of employees is enumerated explicitly in the query. However, this does not consider the effect of employee turnover. By identifying employees and those users with access to the database both as roles, we can determine whether the desired property will continue to hold as new employees are added. Thus we seek techniques that can solve queries of this general form as often as possible.

Definition 2 (RCPI). *An* instance of a role containment problem *(RCPI) is given by a triple* $\langle \mathcal{P}, \mathcal{R}, X.u \sqsupseteq A.r \rangle$. *An RCPI is said to be* satisfied *if and only if* $[\![X.u]\!]_{\mathcal{P}'} \supseteq [\![A.r]\!]_{\mathcal{P}'}$ *for each* $\mathcal{P}'$ *such that* $\mathcal{P} \overset{*}{\mapsto}_{\mathcal{R}} \mathcal{P}'$. *In this case we also say that* $\mathcal{P}$ satisfies $X.u \sqsupseteq A.r$ under $\mathcal{R}$.

3. REDUCTIONS

This section describes several reductions that transform one RCPI into another that is typically less expensive to evaluate. Our findings in Section 5 indicate that, when using our model checking technique and our platform configuation, these reductions often make the difference between being unable to evaluate an RCPI and being able to do so. We conjecture that they may also reduce the cost of applying other approaches to solve RCPI problems, such as one based one on the proof method of Sistla [20, 21].

3.1 Infinite State Space Reduction

Any analysis technique that operates by exhaustively examining reachable states must address the fact that in our analysis problem the size of reachable policy states is unbounded, and hence the state space is infinite. In this section we present a subspace of bounded size that was previously identified [12] and that has the property that, given any query, any restriction rule, and any initial state, there exists a reachable state in which the query is violated if and only if there exists a reachable state within the bounded state space in which the query is violated.

Given a policy state $\mathcal{P}$, a restriction rule $\mathcal{R}$, and a query $\mathcal{Q} = X.u \sqsupseteq A.r$, the state space that must be considered consists of all states that are reachable from $\mathcal{P}$ under $\mathcal{R}$ and that are composed of statements in $\mathcal{P}$ and in $\mathcal{N}$, in which $\mathcal{N}$ is constructed as follows: $\mathcal{N} = \{A.r \longleftarrow D \mid r \in \mathsf{Names}(\mathcal{P}) \wedge \mathsf{A}, \mathsf{D} \in \mathsf{Principals}(\mathcal{P}) \cup \mathsf{NewPrinc}(\mathcal{P}, \mathcal{Q})\}$, $\mathsf{NewPrinc}(\mathcal{P}, \mathcal{Q})$ is a set of new principals of size 2^K, $K = |\mathsf{SigRoles}(\mathcal{P}, \mathcal{Q})|$, and $\mathsf{SigRoles}(\mathcal{P}, \mathcal{Q})$ is the set $\{X.u\} \cup \{A.r_1 \mid A.r \longleftarrow A.r_1.r_2 \in \mathcal{P}\} \cup \{B_1.r_1, B_2.r_2 \mid A.r \longleftarrow B_1.r_1 \cap B_2.r_2 \in \mathcal{P}\}$. It is shown by Li *et al.* [12] that this set of reachable states has the desired property. We consider the number of new principals used here to be conservative in the sense that any

fewer number of principals does not guarantee this desired property.

3.2 Unrestricted Role Reduction

We now present a result that in many cases enables us to further reduce the size of the state space that must be explored. The idea is that given an initial state $\mathcal{P}$, we can often perform the analysis using a smaller initial state and obtain identical results. It is important to note that this is significantly different than the Lower Bound $LB(\mathcal{P})$ and Upper Bound $UB(\mathcal{P})$ programs described in [12]. The $LB(\mathcal{P})$ program removes all statements defining a role $C.r \notin \mathcal{S}_\mathcal{R}$ from $\mathcal{P}$ for the purpose of evaluating a query of the form $X.u \sqsupseteq \{B \mid B \in Principals\}$, whereas the $UB(\mathcal{P})$ program adds a special principal τ representing all principals to each growth unrestricted role for the purposes of evaluating a query of the form $\{B \mid B \in Principals\} \sqsupseteq A.r$. Our reduction program removes all statements defining a role $C.r \notin \mathcal{S}_\mathcal{R} \wedge C.r \notin \mathcal{G}_\mathcal{R}$ for the purpose of evaluating a query of the form $X.u \sqsupseteq A.r$.

We define $\equiv_\mathcal{R}$, a binary relation over policy states, by $\mathcal{P}_1 \equiv_\mathcal{R} \mathcal{P}_2$ if and only if $\mathcal{P}_1 \overset{*}{\mapsto}_\mathcal{R} \mathcal{P}_2$ and $\mathcal{P}_2 \overset{*}{\mapsto}_\mathcal{R} \mathcal{P}_1$. Note that $\equiv_\mathcal{R}$ is an equivalence relation: (1) it is reflexive, as $\mathcal{P} \overset{*}{\mapsto}_\mathcal{R} \mathcal{P}$ for all $\mathcal{P}$; (2) it is symmetric by construction; (3) it is transitive because $\overset{*}{\mapsto}_\mathcal{R}$ is transitive.

We also define the core of $\mathcal{P}$, $\mathsf{core}_\mathcal{R}(\mathcal{P})$, to be the subset of $\mathcal{P}$ consisting of those statements that define growth-restricted or shrink-restricted roles.

THEOREM 1. *Given any policy states $\mathcal{P}_1$ and $\mathcal{P}_2$ and any restriction rule $\mathcal{R}$, $\mathcal{P}_1 \equiv_\mathcal{R} \mathcal{P}_2$ if and only if $\mathsf{core}_\mathcal{R}(\mathcal{P}_1) = \mathsf{core}_\mathcal{R}(\mathcal{P}_2)$.*

PROOF. The "if" direction is straightforward: starting from any state $\mathcal{P}$, it is clear that any other state with the same core is reachable from $\mathcal{P}$. For the "only if" direction, suppose $\mathsf{core}_\mathcal{R}(\mathcal{P}_1) \neq \mathsf{core}_\mathcal{R}(\mathcal{P}_2)$. There are two cases, depending on whether the cores differ in statements that define roles that are growth-restricted or roles that are shrink-restricted. If $\mathsf{core}_\mathcal{R}(\mathcal{P}_1) \backslash \mathsf{core}_\mathcal{R}(\mathcal{P}_2)$ contains a statement defining a growth restricted role, then $\mathcal{P}_2 \overset{*}{\mapsto}_\mathcal{R} \mathcal{P}_1$ does not hold. If $\mathsf{core}_\mathcal{R}(\mathcal{P}_1) \backslash \mathsf{core}_\mathcal{R}(\mathcal{P}_2)$ contains a statement defining a shrink restricted role, then $\mathcal{P}_1 \overset{*}{\mapsto}_\mathcal{R} \mathcal{P}_2$ does not hold. □

Given this result, it is clear that the least state (under the subset ordering) that is equivalent to a given $\mathcal{P}$ is $\mathsf{core}_\mathcal{R}(\mathcal{P})$. It follows from the definition of $\equiv_\mathcal{R}$ and the transitivity of $\overset{*}{\mapsto}_\mathcal{R}$ that for all $\mathcal{P}'$, $\mathcal{P} \overset{*}{\mapsto}_\mathcal{R} \mathcal{P}'$ if and only if $\mathsf{core}_\mathcal{R}(\mathcal{P}) \overset{*}{\mapsto}_\mathcal{R} \mathcal{P}'$. Moreover, $\mathsf{core}_\mathcal{R}(\mathcal{P})$ is the least set for which this is true. Thus, given an initial state $\mathcal{P}$, it is both correct and more efficient to perform our analysis using $\mathsf{core}_\mathcal{R}\ (\mathcal{P})$ as the initial state instead.

DEFINITION 3. *Given a policy $\mathcal{P}$ and restriction rule $\mathcal{R}$, URR ($\mathcal{P}$, $\mathcal{R}$) = $\mathsf{core}_\mathcal{R}\ (\mathcal{P})$*

3.3 Cone of Influence Reduction

A given security policy $\mathcal{P}$ may contain statements that do not affect the membership of queried roles. Such extraneous statements can safely be filtered from the policy model in order to reduce the size of the problem. This reduction removes statements that are said to be outside of a role's cone of influence. This reduction is particularly significant because it has the potential to remove linked or intersection inclusion type statements that contribute to a larger number of principals in the model.

Given a set of roles, Λ, the following definition constructs a set of roles in $\mathcal{P}$. A role, ρ, is in the constructed set if the membership of some role in Λ depends in some way on the membership of ρ. This dependency is reflective of relationship established between roles via RT statements. We call this set of roles *DefRoles*.

DEFINITION 4. *Let Λ and $\mathcal{M}$ be sets of roles, and $\mathcal{P}$ be a policy. We define DefRoles($\mathcal{P}, \Lambda, \mathcal{M}$) to be the least set of roles $\mathcal{O}$ satisfying the following conditions:*

- $\Lambda \subseteq \mathcal{O}$
- $(\lambda \in \mathcal{O} \ \wedge \ \lambda \leftarrow B.r_1 \in \mathcal{P} \wedge B.r_1 \notin \mathcal{M}) \Rightarrow B.r_1 \in \mathcal{O}$
- $(\lambda \in \mathcal{O} \ \wedge \ \lambda \leftarrow B.r_1.r_2 \in \mathcal{P} \wedge D \in Principals) \Rightarrow ((B.r_1 \notin \mathcal{M} \Rightarrow B.r_1 \in \mathcal{O}) \wedge (D.r_2 \notin \mathcal{M} \Rightarrow D.r_2 \in \mathcal{O}))$
- $(\lambda \in \mathcal{O} \ \wedge \ \lambda \leftarrow B.r_1 \ \cap \ C.r_2 \in \mathcal{P}) \Rightarrow ((B.r_1 \notin \mathcal{M} \Rightarrow B.r_1 \in \mathcal{O}) \wedge (C.r_2 \notin \mathcal{M} \Rightarrow C.r_2 \in \mathcal{O}))$

Using *DefRoles*, we define the *Cone of Influence* (COI) as a policy constructed from those statements that define roles in *DefRoles*. In other words, we retain only those statements that influence the answer to an RCPI.

DEFINITION 5. *Given an RCPI $\langle \mathcal{P}, \mathcal{R}, X.u \sqsupseteq A.r \rangle$, we define*

$$COI(\langle \mathcal{P}, \mathcal{R}, X.u \sqsupseteq A.r \rangle) = \mathcal{P}\restriction_{DefRoles(\mathcal{P}, \{X.u\}, \mathcal{S}_\mathcal{R}) \cup DefRoles(\mathcal{P}, \{A.r\}, \mathcal{G}_\mathcal{R})}$$

THEOREM 2. *Given any RCPI $\langle \mathcal{P}, \mathcal{R}, X.u \sqsupseteq A.r \rangle$, let $\mathcal{P}' = COI(\langle \mathcal{P}, \mathcal{R}, X.u \sqsupseteq A.r \rangle)$. Then $\langle \mathcal{P}, \mathcal{R}, X.u \sqsupseteq A.r \rangle$ is satisfied if and only if $\langle \mathcal{P}', \mathcal{R}, X.u \sqsupseteq A.r \rangle$ is satisfied.*

Proof for this and subsequent reductions are given in [17].

3.4 Decomposition

It is sometimes useful to decompose an RCPI into subproblems that can be solved separately. The membership of $A.r$ is the union of $[\![e_i]\!]_\mathcal{P}$ for $i = 1 \ldots n$ in which $e_1 \ldots e_n$ enumerates $\{e \mid A.r \leftarrow e \in \mathcal{P}\}$. Thus if $A.r$ is growth and shrink restricted, then for each reachable state $\mathcal{P}'$ and each principal $E \in [\![A.r]\!]_{\mathcal{P}'}$, there is some $i \in \{1 \ldots n\}$ such that $E \in [\![e_i]\!]_{\mathcal{P}'}$. By isolationg e_i through the use of a new role $A'.r'$, we construct from a given RCPI a collection of new RCPIs such that the orginal is satisfied just in case each RCPI in the collection is satisfied. We demonstrate in Section 5.2 that the decomposed RCPIs can sometimes be successfully solved by our analysis tool when the original cannot.

Decompose constructs the collection of new RCPIs.

DEFINITION 6. *Given an RCPI $\langle \mathcal{P}, \mathcal{R}, X.u \sqsupseteq A.r \rangle$, let A' be a new principal not in $\mathcal{P}$ and r' be a new role name not in $\mathcal{P}$. We define Decompose($\langle \mathcal{P}, \mathcal{R}, X.u \sqsupseteq A.r \rangle$) = $\{\langle \mathcal{P}, \mathcal{R}, X.u \sqsupseteq \rho \rangle \mid A.r \leftarrow \rho \in \mathcal{P} \wedge \rho \in \mathsf{Roles}\} \cup \{\langle \mathcal{P} \cup \{A'.r' \leftarrow e\}, \langle \mathcal{G}_\mathcal{R} \cup \{A'.r'\}, \mathcal{S}_\mathcal{R} \cup \{A'.r'\} \rangle, X.u \sqsupseteq A'.r' \rangle \mid A.r \leftarrow e \in \mathcal{P} \wedge e \notin \mathsf{Roles}\}$.*

Note that each of the new policies contains at most one occurrence of $A'.r'$, and that there is no occurrence of either A' or r' in any statement body. Furthermore, when

e is a role, we do not make use of the new role $A'.r'$. We now formally describe the relationship between the solution to the original RCP instance and the solutions of the new problems.

THEOREM 3. *Given an RCPI $\langle \mathcal{P}, \mathcal{R}, X.u \sqsupseteq A.r \rangle$, if $A.r \in \mathcal{G}_\mathcal{R} \cap \mathcal{S}_\mathcal{R}$, then $\mathcal{P}$ satisfies $X.u \sqsupseteq A.r$ under $\mathcal{R}$ if and only if $\mathcal{P}'$ satisfies $X.u \sqsupseteq \rho$ under $\mathcal{R}'$ for each $\langle \mathcal{P}', \mathcal{R}', X.u \sqsupseteq \rho \rangle \in Decompose(\mathcal{P}, \mathcal{R}, X.u \sqsupseteq A.r)$.*

3.5 Chain Reduction

One type of optimization that is often useful involves pruning the part of the policy that does not impact the membership of the queried roles. This effectively collapses a set of states into a single state such that role containment evaluation on that single state is representative of all the collapsed states. The efficiency of this is evident from not having to evaluate all reachable subsets of pruned statements. It is trivial to see the benefit of such a reduction on the initial state, but more practical technique would be one that can be applied to each reachable state. The most effective pruning would require the evaluation of the queried roles to determine which statements could be removed, however little is gained by evaluating queried role membership in order to prune the policy and then re-evaluating the queried role membership. We propose a technique for pruning a policy without evaluating the queried role membership that is guaranteed to be conservative but not necessarily the smallest pruned policy possible. This is possible by recognizing that given a reachable state, a role that is not defined by any statement must be empty in that state. Since empty roles cannot contribute any principal to the queried roles per the monotonicity of the language, other statements that depend on these empty roles may also be removed. Such an approach can be efficiently implemented as a set of constraints on the reachable statespace, thus reducing the size of the reachable statespace without incurring the cost of evaluating role membership.

DEFINITION 7. *Given any policy $\mathcal{P}$ and any restriction rule $\mathcal{R}$, we define $Chain(\mathcal{P}, \mathcal{R})$ to be the greatest set of policy statements $\mathcal{P}'$ that satisfies the following constraints:*

1. $\mathcal{P}' \subseteq \mathcal{P}$
2. $B.r \in \mathcal{G}_\mathcal{R} \wedge \neg\exists e.\ B.r \leftarrow e \in \mathcal{P}' \Rightarrow$
   $\neg\exists \lambda.\ \lambda \leftarrow B.r \in \mathcal{P}' \wedge$
   $\neg\exists \lambda\ \exists r_2.\ \lambda \leftarrow B.r.r_2 \in \mathcal{P}' \wedge$
   $\neg\exists \lambda\ \exists C\ \exists r_2.\ \lambda \leftarrow B.r \cap C.r_2 \in \mathcal{P}'$
   $\neg\exists \lambda\ \exists C\ \exists r_2.\ \lambda \leftarrow C.r_2 \cap B.r \in \mathcal{P}'$

where $\Rightarrow$ represents implication, $\neg$, negation, $\exists$, an existential quantification, and e is a role expression.

THEOREM 4. *Given any RCPI $\langle \mathcal{P}, \mathcal{R}, X.u \sqsupseteq A.r \rangle$, and any policy state $\widehat{\mathcal{P}}$ such that $\mathcal{P} \overset{*}{\mapsto}_\mathcal{R} \widehat{\mathcal{P}}$, let $\mathcal{P}' = Chain(\widehat{\mathcal{P}}, \mathcal{R})$. Then $[\![X.u]\!]_{\widehat{\mathcal{P}}} = [\![X.u]\!]_{\mathcal{P}'}$ and $[\![A.r]\!]_{\widehat{\mathcal{P}}} = [\![A.r]\!]_{\mathcal{P}'}$.*

Consider the example in Figure 2 where $\mathcal{P}$ is the initial policy and $\widehat{\mathcal{P}}$ is a reachable state according to $\mathcal{R}$. In removing the statement $B.r \leftarrow C.r$, it can easily be determined that $[\![B.r]\!]_{\widehat{\mathcal{P}}} = \emptyset$ since no statement defines $B.r$. Since $B.r$ is empty, then clearly the statements $X.u \leftarrow B.r$ and $A.r \leftarrow B.r$ do nothing to contribute to the membership of the queried roles, and thus can be safely removed,

Index	Policy Statement of $\mathcal{P}$		
0	$X.u \leftarrow B.r$	6	$C.r \leftarrow E.r$
1	$X.u \leftarrow J$	7	$D.r \leftarrow F$
2	$A.r \leftarrow B.r$	8	$D.r \leftarrow G$
3	$A.r \leftarrow K$	9	$E.r \leftarrow H$
4	$B.r \leftarrow C.r$	10	$E.r \leftarrow I$
5	$C.r \leftarrow D.r$		

$\mathcal{G}_\mathcal{R} = \{X.u, A.r, B.r, C.r, D.r, E.r, F.r\}$, $\mathcal{S}_\mathcal{R} = \{E.r\}$
RCPI: $\langle \mathcal{P}, \mathcal{R}, X.u \sqsupseteq A.r \rangle$

Policy State	Statement Indices	Comment
$\mathcal{P}$	0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10	Initial policy
$\widehat{\mathcal{P}}$	0, 1, 2, 3, 5, 6, 7, 8, 9, 10	Reachable state
$\mathcal{P}'$	1, 3, 5, 6, 7, 8, 9, 10	Apply $Chain(\widehat{\mathcal{P}}, \mathcal{R})$
$\mathcal{P}''$	1, 3, 9, 10	Apply $COI(\mathcal{P}', \mathcal{R})$

Figure 2: Chain Reduction Example

yielding $\mathcal{P}'$. We can then further reduce this policy state by applying the Cone of Influence reduction and removing statements that define roles not in *DefRoles*, yielding $\mathcal{P}''$. In this example, $\mathcal{P}$ would have required us to examine at least 2^9 states (9 statements defining non-shrink restricted roles), but since $\mathcal{P}'$ is the smallest representative of the set $\{\mathcal{P}' \cup \mathcal{P}''' \mid \mathcal{P}''' \subseteq \{0, 2, 5, 6, 7, 8\}\}$, we save ourselves from examining 2^6 states through this technique.

4. POLICY MODEL CHECKING

Automated techniques for verifying RT policies are necessary for two important reasons. First, manual techniques may be tedious and error prone even in relatively simple RT policies. Second, it is not difficult to succumb to the complexity of even moderately sized policies, particularly since policies devised in a decentralized environment may exhibit characteristics that interrelate several roles in complicated ways. For these reasons, automated analysis seems especially useful to policy authors and stakeholders. We specifically explore the use of model checking as an automated tool to verify if a given policy satisfies a given security specification.

We present a novel technique for model checking any given role containment problem in RT. We address the issue of cyclic dependencies by incorporating the fixpoint evaluation into the model with the intent to let the model checking tool find a counterexample as a result of this process. Such a dependency occurs when at least one role depends, directly or indirectly, on its own membership to reach the least fixpoint in membership evaluation. We also describe a collection of reductions and optimizations that assist the model checking effort by reducing the policy model into a form that may be tractable. Section 3 formally described these techniques and here we describe how they fit into the overall analysis.

We begin by providing an overview to model checking with a specific emphasis on symbolic model checking. Next, we describe our policy model and translation to the model checking tool. Finally, we discuss how reductions and optimizations are applied in our technique.

4.1 Model Checking Overview

Model checking [1, 2] is an automated verification technique that constructs a finite model of a system and exhaustively explores the state space of this model to determine if

desired properties hold in all reachable states. When they do not, a counterexample will be produced to show an error trace, which can be used to fix the model or the property specification. The model is composed of state and transition relations. The state of the system is determined by the current values of all state variables, whereas the state space is the set of all possible states. The transition relation is defined by the set of next assignments which execute concurrently in a step to determine the next state of the model. We are only interested in those states that are reachable from a given start state.

The properties we want to check are called specifications, which are expressed in temporal logic [15] formulas. Temporal logic is a language for expressing properties related to a sequence of states in terms of temporal logic operators and logic connectives (e.g., $\wedge$ and $\vee$). Temporal operators X, F, and G represent next state, some future state, and all future states, respectively. For example, Gp means that property p is always true in all possible states.

SMV [14] is a mature, symbolic model checking tool. Although conceptually similar to the aforementioned description, it operates on a set of states at a time rather than each explicit state. This is accomplished through the use of specialized data structures encoding the set of initial states, the transition relation, and the set of states representing the negated specification. The transition relation is repeatedly applied to the negated specification until a fixpoint is reached. For each application of the transition relation, if the produced set of states intersects with any of the initial states, the model fails to satisfy the specification. This approach often permits substantially larger models to be verified than the explicit approach.

4.2 Model Checking RT Policies

We now formally describe the construction of our policy model, associated finite state machine, and translation to SMV.

4.2.1 Maximal Relevant Policy Statements

We begin by constructing a set of policy statements that serve as the foundation for evaluating the input policy. The set of maximal relevant policy statements (MRPS) represents a policy state reachable from the initial policy $\mathcal{P}$ and constructed by adding a set of simple member statements that introduce an appropriate number of new principals into each non-growth restricted role. From the Statespace Reduction in Section 3.1, it is easy to see that if $\mathcal{P}$ fails to satisfy the query, then there exists some $\mathcal{P}'$ and principal E such that $E \in [\![A.r]\!]_{\mathcal{P}'}$, $E \notin [\![X.u]\!]_{\mathcal{P}'}$, and $\mathcal{P}' \subseteq MRPS(\mathcal{P}, \mathcal{R}, X.u \sqsupseteq A.r)$. Thus $MRPS(\mathcal{P}, \mathcal{R}, X.u \sqsupseteq A.r)$ is considered a maximal policy state since the set of reachable states to be evaluated are subsets of this set. Figure 3 formally describes the construction of $MRPS(\mathcal{P}, \mathcal{R}, X.u \sqsupseteq A.r)$.

Two important observations regarding the placement and composition of principals in the $MRPS(\mathcal{P}, \mathcal{R}, X.u \sqsupseteq A.r)$ should be made from Figure 3. First, while the set $ModelPrinc(\mathcal{P})$ consists of both existing and new principals, we can limit the set of existing principals to those that are directly introduced via simple member statements in $\mathcal{P}$ and those that, when combined with a linked role name in $\mathcal{P}$, produce a role found in $\mathcal{P}$. It is easy to see that all other existing principals in $\mathcal{P}$ produce roles that are representable

$$
\begin{aligned}
S &= \alpha \times \beta \times \{\mathit{fix}, \mathit{eval}\} \\
\alpha &= \{\mathcal{P}' \mid \widehat{\mathcal{P}} \upharpoonright_{\mathcal{S}_\mathcal{R}} \subseteq \mathcal{P}' \wedge \mathcal{P}' \subseteq \widehat{\mathcal{P}}\} \\
\beta &= \mathit{Roles}(\widehat{\mathcal{P}}) \rightarrow 2^{\mathit{ModelPrinc}(\mathcal{P})} \\
&\quad (\textit{i.e.}, \beta \text{ is the set of functions from roles to sets of} \\
&\quad \text{principals.}) \\
\widehat{\mathcal{P}} &= \mathit{MRPS}(\mathcal{P}, \mathcal{R}, X.u \sqsupseteq A.r) \\
&\mathit{Roles}(\mathcal{P}) \\
&\quad = \{B.r \mid B.r \leftarrow e \in \mathcal{P} \vee \lambda \leftarrow B.r.r_1 \in \mathcal{P} \vee \\
&\quad \lambda \leftarrow B.r \cap C.r_1 \in \mathcal{P} \vee \lambda \leftarrow C.r_1 \cap B.r \in \mathcal{P}\} \cup \\
&\quad \mathit{SubLinkRoles}(\mathcal{P}) \\
&\mathit{SubLinkRoles}(\mathcal{P}) \\
&\quad = \{B.r \mid B \in \mathit{ModelPrinc}(\mathcal{P}) \\
&\quad \wedge\, r \in \mathit{LinkedRoleNames}(\mathcal{P})\} \\
\sigma_0 &= \langle \widehat{\mathcal{P}}, \pi_0 \rangle \\
\pi_0(\lambda) &= \emptyset, \text{ forall } \lambda \in \mathit{Roles}(\widehat{\mathcal{P}}) \\
\delta &= \{\langle\langle \mathcal{P}_1, \pi, \mathit{eval}\rangle, \langle \mathcal{P}_1, \pi', \mathit{eval}\rangle\rangle \mid \pi' = T_{\mathcal{P}_1}(\pi)\} \cup \\
&\quad \{\langle\langle \mathcal{P}_1, \pi, \mathit{eval}\rangle, \langle \mathcal{P}_1, \pi, \mathit{fix}\rangle\rangle \mid \pi = T_{\mathcal{P}_1}(\pi)\} \cup \\
&\quad \{\langle\langle \mathcal{P}_1, \pi, \mathit{fix}\rangle, \langle \mathcal{P}_2, \pi_0, \mathit{eval}\rangle\rangle \mid \widehat{\mathcal{P}} \upharpoonright_{\mathcal{S}_\mathcal{R}} \subseteq \mathcal{P}_2 \wedge \mathcal{P}_2 \subseteq \widehat{\mathcal{P}}\}
\end{aligned}
$$

Figure 4: Construction of AFSM for $\langle \mathcal{P}, \mathcal{R}, X.u \sqsupseteq A.r \rangle$

by new principals. Let $B \notin \mathit{EffectPrinc}(\mathcal{P})$ and $B.r_1$ be a role that is not found in $\mathcal{P}$. If $B.r_1 \in \mathcal{G}_\mathcal{R}$, then $B.r_1$ must be empty since there are no statements defining it in $\mathcal{P}$. If $B.r_1 \in \mathcal{S}_\mathcal{R}$, then no principal is forced into $B.r_1$ since there are no statements defining it in $\mathcal{P}$. Thus $B.r_1$ is effectively unrestricted and B may serve as a new principal.

Second, model principals are only added to roles that are both non-growth restricted and in the set of $\mathit{DefRoles}(\mathcal{P}, A.r)$. This is easy understood from the examination of $\mathcal{P}'$, where $E \in [\![A.r]\!]_{\mathcal{P}'}$, $E \notin [\![X.u]\!]_{\mathcal{P}'}$. Clearly the membership of $X.u$ is irrelevant so long as $E \notin [\![X.u]\!]_{\mathcal{P}'}$. Thus minimizing the introduction of principals into $\mathit{DefRoles}(\mathcal{P}, X.u) - \mathit{DefRoles}(\mathcal{P}, A.r)$ serves to reduce the number of policy states to be examined while preserving $\mathcal{P}'$ if one should exist.

4.2.2 Analysis Finite State Machines

The following definition shows how we use $MRPS(\mathcal{P}, \mathcal{R}, X.u \sqsupseteq A.r)$ to define the finite state machine that we model check.

Definition 8 (AFSM). *Given an RCPI, $\langle \mathcal{P}, \mathcal{R}, X.u \sqsupseteq A.r \rangle$, the corresponding* analysis finite state machine *(AFSM) is given by the 3-tuple, $\langle S, \sigma_0, \delta \rangle$, in which S is a finite, non-empty set of states, $\sigma_0 \in S$ is the initial state, $\delta \subseteq S \times S$ is a transition relation, and these structures are constructed as shown in Figure 4.*

The AFSM state includes not only a policy state, but also a mapping that takes a role to a set of principals in that role. Because the RT language allows cyclic dependencies among roles, we found it necessary to use a sequence of state transitions to calculate the fixpoint that determines the membership of each role. This is accomplished by differentiating between what we call *fixpoint mode* (*fix*) and

$MRPS(\mathcal{P}, \mathcal{R}, X.u \sqsupseteq A.r) = \mathcal{P} \cup \{\lambda \leftarrow e \mid \lambda \notin \mathcal{G}_\mathcal{R} \wedge \lambda \in DefRoles(\mathcal{P}, \{A.r\}) \wedge e \in ModelPrinc(\mathcal{P})\}$, where

1. $ModelPrinc(\mathcal{P}) = NewPrinc(\mathcal{P}) \cup EffectPrinc(\mathcal{P}) \cup DirectPrinc(\mathcal{P})$
2. $NewPrinc\ (\mathcal{P})$ is any set of principals of size $2^{|SigRoles(\mathcal{P})|}$ such that $NewPrinc(\mathcal{P}) \cap (Principals(\mathcal{P}) \cup Principals(\mathcal{R})) = \emptyset$
3. $SigRoles(\mathcal{P}) = \{X.u\} \cup \{B.r_1 \mid C.r \leftarrow B.r_1.r_2 \in \mathcal{P}\} \cup \{B_1.r_1, B_2.r_2 \mid C.r \leftarrow B_1.r_1 \cap B_2.r_2 \in \mathcal{P}\}$
4. $EffectPrinc(\mathcal{P}) = \{B \mid B \in Principals(\mathcal{P}) \wedge B.r \leftarrow e \in \mathcal{P} \wedge r \in LinkedRoleNames(\mathcal{P})\}$
5. $DirectPrinc(\mathcal{P}) = \{B \mid \lambda \leftarrow B \in \mathcal{P}\}$
6. $LinkedRoleNames(\mathcal{P}) = \{r_1 \mid \lambda \leftarrow B.r.r_1 \in \mathcal{P}\}$
7. $Principals(\mathcal{P}) = \{B \mid B.r \leftarrow e \in \mathcal{P} \vee \lambda \leftarrow B \in \mathcal{P} \vee \lambda \leftarrow B.r.r_1 \in \mathcal{P} \vee \lambda \leftarrow B.r \cap C.r_1 \in \mathcal{P} \vee \lambda \leftarrow C.r \cap B.r_1 \in \mathcal{P}\}$
8. $Principals(\mathcal{R}) = \{B \mid B.r \in \mathcal{G}_\mathcal{R} \vee B.r_1 \in \mathcal{S}_\mathcal{R}\}$

Figure 3: Construction of $MRPS(\mathcal{P}, \mathcal{R}, X.u \sqsupseteq A.r)$

evaluation mode (*eval*). Evaluation mode signifies that the calculation of role memberships has not stabilized, so the query should not be evaluated yet; from such a state, the fixpoint evaluation should proceed and the policy state should not change. Fixpoint mode signifies that role membership under the given policy is now known and the query should be evaluated; provided the query is satisfied, from such a state, a new policy state should be chosen.

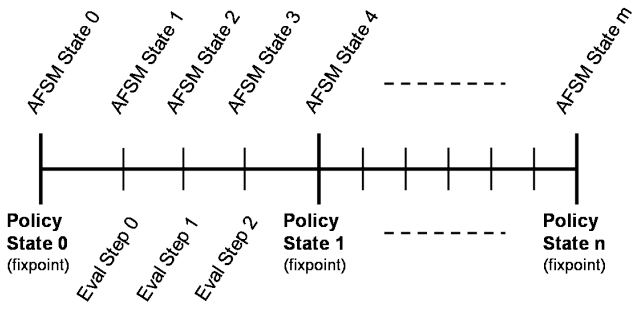

Figure 5: Policy to AFSM Mapping of States

4.2.3 Translation to SMV

Now that we have established the AFSM for a particular policy, restriction rule, and query, we can now translate the AFSM into SMV's input language. Translation consists of constructing four core components: data structure declaration, initialization, next state relations, and specification. Together these components define the set of states that will be evaluated against the specification. These states define not only a particular policy state, but also the membership of roles in the policy state as it is evaluated against the specification. Recall that the finite state machine represents not only policy states, but intermediate role membership evaluations of each policy state. In constructing it this way, the finite state machine simulates the fixpoint calculation to determine the membership of the queried roles. Only policy states that have reached the simulated fixpoint are evaluated against the specification.

Each finite state machine declares a set of data structures consisting of a bit vector representing the set of statements in the model, a bit vector per role representing the membership of that role, and a boolean variable that distinguishes two evaluation modes. The statement bit vector defines the current policy state by indicating which policy statements are included in that state. Each index into the bit vector corresponds to a particular policy statement in the maximal policy state. Statements that define shrink restricted roles and are included in the initial policy are present in all reachable states. In addition, each role is represented by a distinct bit vector where each index represents whether a principal is a member of that role. Finally, the evaluation mode boolean provides a means to evaluate policies with circular dependencies. The boolean signifies whether the current policy state is still performing a fixpoint calculation or not. Simulating fixpoint calculations is necessary for correctly handling policies with cyclic dependencies. The policy state is only allowed to change if the fix point simulation has finished for a particular state.

Initialization of these variables is a straightforward process. Each statement bit variable that defines a role that is shrink restricted is asserted, while every other statement variable is non-deterministically set as either asserted or not asserted. This constructs a set of initial policy states that are reachable from the initial input policy state. By using a set of initial policy model states rather than a single initial state, we may trade memory (larger BDD size) for speed (fewer fixpoint calculations). Role membership bits are initialized to represent empty roles, and the mode bit is always initialized to assert a fixpoint mode.

The next state relation component defines how the AFSM may change. This component is converted by SMV into a transition relation. The manner in which the model policy state is changed is determined by the fixpoint mode. If the model is in a fixpoint mode, the policy state must remain consistent while the membership of the roles is updated. This occurs until a fixpoint (none of the roles can introduce any more principals) is reached, at which point the statement bit vector is non-deterministically set to a new reachable policy state, the role membership bit vectors are re-initialized, and the mode is reset to fixpoint mode.

The specification component expresses the role containment property to be verified. Given a policy property such as $X.u \sqsupseteq A.r$, the specification verifies the model property that all model states in which the fixpoint calculation has

completed implies that there exists no witness principal in the membership of $A.r$ that is not also found it the membership of $X.u$. Figure 6 describe such SMV specifications. For comparison, we include other types of RT queries represented as SMV specifications as well.

4.2.4 Translation of Policy without Cycles

Our previous work [16] remains useful in cases where the input policy does not contain cycles, and the preferred approach in such cases since the overhead of simulating the fixpoint role membership calculation can be removed. Specifically the state space we need to consider is limited to the number of subsets of statements from the *MRPS*. This can be easily understood from the fact that in an acyclic policy, the membership of each role needs to be evaluated at most once, and thus can be implemented as a set of constraints. This permits larger models to be verified than the approach described above. We compare the general translation against the non-cyclic translation in Section 5.2.

4.3 Optimizations

In Section 3, we demonstrated the correctness of several techniques that may assist in reducing the size of the input policy and possibly reduce the complexity of the analysis. We describe these techniques below as they are applied to our analysis technique.

4.3.1 Pre-Processing

Many of the reductions from Section 3 take a policy $\mathcal{P}$, restriction rule $\mathcal{R}$, and role containment query $X.u \sqsupseteq A.r$ (an instance of RCP such as $\langle \mathcal{P}, \mathcal{R}, X.u \sqsupseteq A.r \rangle$) and produce either an equivalent problem or a set of problems that when combined produce the original problem. Such new problems are often easier to verify than the original, and for this reason such an effort is significant. It is important to note that these new problems can be solved using any other RT analysis technique, such as those of Sistla *et al.* [21], since the reduction does not assume anything about the manner of analysis used.

Our experience with these reductions allows us to suggest a partial ordering when applying these reductions. We have found that applying *COI* first, followed by *Decompose*, followed by *COI* to be a highly effective reduction strategy. We apply *COI* first to remove as many obvious statements as possible. This sometimes removes statements that impede the use of *Decompose*. We finish with the *COI* reduction to remove statements that no longer influence the membership of the queried roles due to the removal of other statements using another reduction. As such, applying *Chain* followed by *COI* is another effective and straightforward strategy. These partial orderings of reductions are always applied before constructing the *MRPS*.

4.3.2 Restriction Relaxation

Another interesting strategy for simplifying verification involves relaxing the restriction rule. Observe that given an instance of the RCP such as $\langle \mathcal{P}, \mathcal{R}, X.u \sqsupseteq A.r \rangle$, if one or more roles in $\mathcal{R}$ are relaxed (removed completely from $\mathcal{R}$) producing $\mathcal{R}'$ and $\langle \mathcal{P}, \mathcal{R}', X.u \sqsupseteq A.r \rangle$ is satisfied, then $\langle \mathcal{P}, \mathcal{R}, X.u \sqsupseteq A.r \rangle$ is satisfied. Clearly any statements defining a role that is not restricted can be removed by URR. This strategy, called *Restriction Relaxation*, effectively under-specifies the policy and should be applied after the pre-processing reductions but prior to construction of the *MRPS*. Based on our understanding and experience with this problem, it is our conjecture that this technique is sound but not complete.

Initial Policy $\mathcal{P}$	
$A.r \leftarrow B.r_1 \cap C.r_2$	
$B.r_1 \leftarrow D.r_3.r_4$	
$C.r_2 \leftarrow E.r_5.r_4$	
$F.r_6 \leftarrow D.r_3 \cap E.r_5$	$\mathcal{G}_\mathcal{R} = \{A.r, B.r_1, C.r_2, F.r_6, X.u\}$
$X.u \leftarrow F.r_6.r_4$	$\mathcal{S}_\mathcal{R} = \{A.r, B.r_1, C.r_2, F.r_6, X.u\}$
$X.u \leftarrow D.r_3$	
$X.u \leftarrow E.r_5$	RCPI: $\langle \mathcal{P}, \mathcal{R}, X.u \sqsupseteq A.r \rangle$

New Principals: P_1, P_2

New Model Statements	
$D.r_3 \leftarrow P_1$	$P_1.r_4 \leftarrow P_1$
$D.r_3 \leftarrow P_2$	$P_1.r_4 \leftarrow P_2$
$E.r_5 \leftarrow P_1$	$P_2.r_4 \leftarrow P_1$
$E.r_5 \leftarrow P_2$	$P_2.r_4 \leftarrow P_2$

a. Principal Abstraction 2

New Principals: P_1, P_2, P_3

New Statements
$D.r_3 \leftarrow P_1$
$E.r_5 \leftarrow P_2$
$P_1.r_4 \leftarrow P_3$
$P_2.r_4 \leftarrow P_3$

b. Counterexample

Figure 7: Incompleteness of Principal Abstraction

4.3.3 Principal Abstraction

In many cases the number of new principals added to the *MRPS* makes the model difficult to analyze since not only are there significantly more policy statements to consider, but the maximum membership size of each role increases. This is problematic since the state space of the AFSM depends on both the number of subsets of policy statements and the number of subsets of principals in each role. One technique for analyzing such policies is to under-approximate the *MRPS* by adding fewer than the conservative number of new principals. Recall that the conservative number of new principals was described in Section 3.1. Any violation of the query detected by the model checker is clearly valid since a counterexample is produced. However upon verification, it is indeterminant whether the result is a false positive or not. A false positive could occur in a situation where additional new principals may be necessary to expose a counterexample. We call this technique *Principal Abstraction*, and it is applied during *MRPS* construction.

Figure 7 demonstrates how the Principal Abstraction can interfere with finding all counterexamples. In this example, it is necessary to add three new principals in order to expose a counterexample. When we apply our technique and limit the number of new principals to two, we see that no subset of new model statements will produce a counterexample. In this case, the problem has to do with an insufficient number of unique sub-linked roles being created. In our experience of constructing and evaluating policies for role containment by hand, it is often the case that we need one new principal for each unique linked role expression and one new principal to serve as the witness. However, in many cases where linked role names are not shared among linked role expressions, two new principals are sufficient to expose counterexamples.

4.3.4 Chain Reduction

Chain Reduction is applied as a set of constraints on the transition relation of the AFSM and implemented as a set of

Property	RT Query	SMV LTL Specification	Notes
Availability	$A.r \sqsupseteq \{C, D\}$ Always	`assert G (fixpoint -> (Ar[0] & Ar[1]))`	C and D in A.r
Safety	$\{C, D\} \sqsupseteq A.r$ Always	`assert G (fixpoint -> (~ Ar[2]))`	E not in A.r
Containment	$A.r \sqsupseteq B.r$ Always	`assert G (fixpoint -> (Ar | Br = Ar))`	Nothing new in B.r

Figure 6: RT Queries to SMV Specifications

`TRANS` statements in SMV. Specifically, each constraint takes the form of a condition that forces certain policy statements to be absent if other policy statements also happen to be absent, thus forcing sets of states to collapse to a single *representative state*. The relationship between such policy statements is identified by examining how each statement defines or depends on a given role. For each role, we identify the statements that define the role (*defining statements*) as well as the statements that depend on the role (*dependency statements*). If all defining statements are absent from a given role, then we may also remove all dependency statements for that role as well. In SMV, `TRANS` statements are explicit transition relation statements that overlay the existing transition relation. By incorporating these constraints into the transition relation, only representative states are examined.

5. EMPIRICAL FINDINGS & ANALYSIS

We now show the results of our empirical evaluation of our reductions and techniques. We demonstrate that model checking can be a viable means of evaluating role containment. More significantly, reduction techniques reduce the problem size in a sound and complete manner, such that some RCPIs can be reduced to forms that can then be automatically verified, and may increase the performance of our model checking technique. In the cases that the reduction techniques cannot effectively reduce a problem, other optimization techniques such as principal abstraction and restriction relaxation can provide sound solutions. Finally, we discuss some characteristics that we have observed that seemed to be difficult for our approach to effectively handle.

We begin by discussing the test cases used and how they were generated. Next, we summarize the results of the evaluation, and lastly we conclude with an analysis of those results.

5.1 Test Case Suite

To the best of our knowledge, RT has not been implemented in any real-world systems so it is difficult to gather policies used in practice. We construct a collection of test cases that exhibit various characteristics that might be found in real-world systems.

We designed six policies to evaluate the effectiveness and efficiency of our reductions and model checking technique. The tests were executed on a 3.4 GHz Pentium 4 operating on Windows XP Service Pack 2. This machine was equipped with 4 GB of memory, although only 3.24 GB was recognized by Windows. Attributes of the input policies include the number of statements, principals, roles and significant roles. We describe whether or not cycles are present in the policy in the accompanied notes. The observed results include the sizes of the state space of the AFSM and *MRPS*, the number of reachable states as given by SMV, translation time, verification time, and whether the policy satisfied the query. In the cases where SMV crashed, we include the time elapsed and memory usage at the time of the error. By comparison, the baseline memory usage prior to experimentation was approximately 750 MB.

The results of our experiments are reported in Figure 8. For each policy, we first evaluated the original RCPI without any reductions or optimizations. We then applied various techniques and reevaluated so that we could observe the impact that our reductions and optimizations have on analysis efficiency and effectiveness. Efficiency was measured by the speedup of time required to perform the analysis, either between a reduced policy and the original policy, or between reduced policies. Effectiveness was measured by the correctness of the query result.

Various test cases were designed to demonstrate the following characteristics. Test Case 1 is a simple example that fails to satisfy the given query, and could not be evaluated using our model checking technique despite the resources of the underlying machine. However, when our reductions and optimizations were applied, we could correctly determine that the policy fails to satisfy the given query. Test Case 2 is an example from [12] that satisfies the given query, but again could not be evaluated using our modeling checking technique alone. Test Case 3 demonstrates a policy where decomposition is the only effective means of transforming the RCPI into a form that can be evaluated. Furthermore, it was a case where the policy model was so large that the translation process reached a point where the application became unresponsive. Test Case 4 is an example that fails to satisfy the given query, and the relation of the roles is such that a principal introduced into a role could propagate to the queried roles through three distinct sequences of roles as opposed to a single sequence. Test Case 5 is an example that satisfies the given query, and exhibits cycles such that our non-cycle translation [16] could not reliably verify. Finally, Test Case 6 is an example that satisfies the given query, and the relation of the roles is such that all of our techniques failed to provide any traction with this problem unless Restriction Relaxation was used.

5.2 Results & Analysis

Our experimentation demonstrated that both our reduction/optimization techniques as well as our underlying model checking strategy worked correctly for the six test cases, including the test case involving a cycle. We evaluated cases where the policy satisfied the role containment query as well as cases where the policy failed to satisfy the query, and in both types of cases the result was correct as compared with our manual evaluation. In examples such as Test Cases 3 and 4 where decomposition was applied, the results were also correct, even when one part of a decomposed problem produced a false positive (*i.e.* a subproblem is satisfied), another part reported a counterexample. Recall that a role containment problem is satisfied if and only if all of its decomposed subproblems are satisfied. In Test Case 1, when the initial policy required resources beyond those of our test machine, the cone of influence reduction reduced it to a form that SMV was able to handle. In Test

Test Case	Policy	#	Analysis Techniques	# Stms	# Princ	# Roles	# Sig Roles	# MRPS States	# AFSM States	# Reachable States	Translation Time (sec)	Verificaiton Time (sec)	Satis-fies?	Time to Crash (sec)	Mem at Crash (GB)
1	A.r <-- C.r	1.1	Original Policy	8	9	9	4	2^292	2^692	7.96E+87	0.484			102.203	1.71
	B.r <-- D.r	1.2	COI	7	7	7	2	2^27	2^71	6.70E+08	0.016	22.5	no		
	B.r <-- E.r	1.3	PA2	8	9	9	4	2^12	2^34	2.00E+04	0	0.11	no		
	C.r <-- D.r	1.4	URR	7	7	7	2	2^27	2^71	6.70E+08	0.016	22.593	no		
	C.r <-- F.r.r1	1.5	Decomp1	5	7	7	3	2^17	2^73	7.73E+05	0.016	14.5313	yes		
	D.r <-- F.r	1.6	Decomp2	7	8	8	4	2^291	2^675	3.98E+87	0.453			~86	1.65
	E.r <-- G.r & H.r	1.7	COI + Decomp1	4	5	5	1	2^4	2^14	8.40E+01	0	0.0156	yes		
	X.u <-- B.r	1.8	COI + Decomp2	6	6	6	2	2^26	2^66	3.35E+08	0.015	14.0156	no		
		1.9	Chain	8	9	9	4	2^292	2^692	7.96E+87	0.516			160.469	2.59
		1.10	COI + Chain	7	7	7	2	2^27	2^71	3.94E+08	0	17.313	no		
		1.11	NonCycle + Chain	8	9	9	4	2^372			0.203			~311	2.11
		1.12	NonCycle+Chain+PA8	8	9	9	4	2^122		1.85E+37	0.078	0.8906	no		
		1.13	NonCycle+Chain+PA12	8	9	9	4	2^230		6.04E+69	0.125			~128	2.71
	Notes														
	1. X.u contains A.r (false)														
	2. Compare baseline #1.1 with #1.9. Chain allowed verification to run much longer and to a greater memory usage than baseline.										G_R = {A.r, C.r}				
	3. Compare #1.2 and #1.10. Chain saved time and lowered reachable statespace.										S_R = {B.r, D.r, X.u}				
	4. Experiment #1.11 could not finish building transition relation and thus no reachable state size determined.														
2	Alice.access <-- Bob	2.1	Original Policy	10	5	7	3	2^155	2^359	1.37E+47	0.141			113.453	1.66
	HR.employee <-- HR.manager	2.2	COI	7	3	6	3	2^109	2^259	1.95E+33	0.094			117.61	1.97
	HR.employee <-- HR.programmer	2.3	PA2	10	5	7	3	2^41	2^107	1.27E+13	0.031	23.609	yes		
	HR.manager <-- Alice	2.4	URR	7	3	6	3	2^109	2^259	1.95E+33	0.078			114.281	1.97
	HR.programmer <-- Bob	2.5	Decomp1	5	4	3	1	2^6	2^21	2.56E+02	0	0.0468	yes		
	HR.programmer <-- Carl	2.6	Decomp2	6	5	4	1	2^6	2^26	2.56E+02	0	0.0625	yes		
	SA.access <-- SA.delegatedAccess & HR.employee	2.7	Decomp3	11	6	8	3	2^155	2^371	1.37E+47	0.141			~117	1.84
	SA.access <-- SA.manager	2.8	Chain	10	5	7	3	2^155	2^359	1.37E+47	0.14			108.516	1.63
	SA.delegatedAccess <-- SA.manager.access	2.9	COI + Chain	10	5	7	3	2^109	2^259	1.95E+33	0.125			116.32	1.97
	SA.manager <-- HR.manager	2.10	NonCycle + Chain	10	5	7	3	2^142		1.36E+39	0.11	0.1562	yes		
	Notes														
	1. HR.employee contains SA.access (true)										G_R = {HR.employee, SA.access, SA.delegatedAccess, SA.manager}				
	2. Compare baseline #2.1 and #2.7. Decomposition added a statement to the original policy.										S_R = {HR.employee,HR.manager,SA.accessm,SA.delegatedAccess,SA.manager}				
3	A.r <-- B.r	3.1	Original Policy	13	13	14	7								
	A.r <-- C.r	3.2	COI	12	13	14	7								
	A.r <-- D.r & E.r	3.3	PA2	13	13	14	7	2^48	2^105	5.63E+14	0			406.141	2.69
	B.r <-- H.r	3.4	URR	12	13	14	7								
	C.r <-- F.r	3.5	Decomp1	4	6	6	4	2^366	2^688	1.40E+101	0.391			~198	2.38
	C.r <-- G.r	3.6	Decomp2	4	6	6	4	2^16	2^384	6.55E+04	0.172	52.5156	no		
	D.r <-- K.r	3.7	Decomp3	4	6	6	4	2^16	2^384	1.31E+05	0.204	51.75	no		
	F.r <-- K.r	3.8	Decomp4	6	10	10	6	2^129	2^4865		29.829				
	F.r <-- Z.r.s	3.9	Decomp5	6	8	9	5	2^1157	2^3599		12.25			~245	2.72
	H.r <-- F.s.t	3.10	Decomp6	8	10	11	5	2^1192	2^3700		13.094			~242	2.72
	H.r <-- J.r & Y.r	3.11	Decomp1 + PA2	4	6	6	4	2^14	2^30	8.00E+04	0	0.0625	yes		
	K.r <-- Y.r	3.12	Decomp2 + PA2	4	6	6	4	2^2	2^20	4.00E+00	0	0.0313	no		
	X.u <-- B.r	3.13	Decomp3 + PA2	4	6	6	4	2^2	2^20	4.00E+00	0	0.0313	no		
		3.14	Decomp4 + PA2	6	10	10	6	2^5	2^29	3.20E+01	0	0.0469	no		
		3.15	Decomp5 + PA2	6	8	9	5	2^17	2^59	1.31E+05	0.016	0.4844	no		
		3.16	Decomp6 + PA2	8	10	11	5	2^22	2^70	1.60E+07	0	1.4844	no		
	Notes														
	1. X.u contains A.r (false)														
	1. Baseline #3.1 and #3.2 translation too big to finish										G_R = {A.r, C.r, D.r, F.r}				
	2. Experiment #3.8 - SMV error: "Too many BDD variables"										S_R = {A.r, B.r, H.r, X.u}				
	3. Experiment #3.9, #3.10 - Failed to finish constructing transition relation														
4	A.r <-- B.r	4.1	Original Policy	11	11	12	6								
	A.r <-- C.r	4.2	Decomp1	4	6	6	4	2^336	2^688		0.438				
	A.r <-- D.r & E.r	4.3	Decomp2	4	6	6	4	2^16	2^368	6.55E+04	0.172	44.5156	no		
	B.r <-- H.r	4.4	Decomp3	5	7	7	4	2^17	2^385	2.62E+05	0.203				
	C.r <-- F.r	4.5	Decomp4	6	7	8	4	2^18	2^402		0.187				
	D.r <-- K.r	4.6	Decomp5	7	10	10	6	2^130	2^4866		30.109				
	F.r <-- K.r	4.7	Decomp6	7	8	9	4	2^19	2^419		0.203				
	H.r <-- F.s.t														
	H.r <-- J.r & Y.r														
	K.r <-- Y.r														
	X.u <-- B.r														
	Notes										G_R = {A.r, C.r, D.r, F.r, K.r}				
	1. X.u contains A.r (false)										S_R = {A.r, B.r, H.r, X.u}				
5	Alice.friend <-- Bob.friend	5.1	Original Policy	6	4	4	1	2^7	2^43	760	0	0.094	yes		
	Alice.friend <-- Carol	5.2	Decomp	4	3	2	1	2^5	2^30	188	0	0.094	yes		
	Bob.friend <-- Alice.friend	5.3	COI	5	4	4	1	2^7	2^43	760	0	0.109	yes		
	Bob.friend <-- Bob.friend.friend														
	Carol.friend <-- Dave														
	Dave.friend <-- Carol.friend														
	Notes										G_R = {Alice.friend, Bob.friend, Dave.friend}				
	1. Bob.friend contains Dave.friend (true)										S_R = {Alice.friend, Bob.friend}				
	2. Exhibits cycles														
6	A.r <-- C.r2 C.r2 <-- P2 E.r4 <-- F.r5	6.1	Original Policy	30	19	12	4	2^621	2^1426		1.844			264.765	2.48
	A.r <-- D.r3 C.r2 <-- P3 E.r4 <-- G.r6	6.2	COI	30	19	12	4	2^621	2^1426		1.844				
	A.r <-- F.r5 C.r2 <-- P4 E.r4 <-- H.r7	6.3	URR	30	19	12	4	2^621	2^1426		1.844				
	A.r <-- G.r6 C.r2 <-- P6 E.r4 <-- J.r8	6.4	PA2	30	19	12	4	2^117	2^306		0.125				
	A.r <-- H.r7 D.r3 <-- P4 E.r4 <-- K.r9	6.5	RR	30	19	12	4	2^645	2^1450						
	A.r <-- J.r8 D.r3 <-- P5 F.r5 <-- L.r1.r0	6.6	RR + COI	28	18	11	1	2^45	2^144		0.031	0.094	yes		
	A.r <-- K.r9 D.r3 <-- P6 L.r1 <-- C.r2 & D.r3	6.7	RR + URR	29	19	12	3	2^75	2^255		0.094				
	B.r1 <-- P1 D.r3 <-- P7 X.u <-- E.r4	**Notes**													
	B.r1 <-- P2 E.r4 <-- B.r1	1. X.u contains A.r (true)											G_R = {A.r, B.r1, C.r2, D.r3, E.r4, F.r5, L.r1, X.u}				
	B.r1 <-- P4 E.r4 <-- C.r2	2. Several techniques not effective enough to make tractable.											S_R = {A.r, B.r1, C.r2, D.r3, E.r4, F.r5, L.r1, X.u}				
	B.r1 <-- P5 E.r4 <-- D.r3	3. Experiments #6.5-#6.7 use restriction relaxation to reduce statespace.															

Notation:
COI - Cone of Influence, URR - Unrestricted Role Reduction, RR - Restriction Relaxation
PAn - Principal Abstraction with n principals, Decompn - Decomposed Subproblem #n
Chain - Chain Reduction, Noncycle - non-cycle version of translation

Figure 8: Evaluation Results

Case 2, when the initial policy was not responsive to any reduction technique, the non-cycle translation coupled with chain reduction seemed to work well.

Based on the theoretical foundation and empirical results, we expect that the effectiveness of the model checking strategy is sound, as well as those reductions from Section 3. Furthermore, the principal abstraction technique demonstrated usefulness to detect at least some subset of counterexamples when the policy failed to satisfy the query. In each of the cases where the initial policy failed to satisfy the query, the use of principal abstraction with just two principals was sufficient to detect a counterexample.

Another significant observation involves the Restriction Relaxation technique. Test Case 6 demonstrates two role containment problems, identical in terms of statements and query, but with slightly different restriction rules. In particular, test cases 6.5-6.7 relax (remove) the role "F.r5" from the restriction rule of the original policy, allowing our other reduction techniques to significantly reduce the size of the problem. Since the reduced policy satisfies the query, then the original must also satisfy the same query due to the relationship between the problems.

In terms of efficiency, the reduction/optimization techniques were in most cases able to reduce the size of the state space, and in some cases to such a degree that the problem became tractable. In five of the six test cases, the initial policy required computing resources beyond those available, but through the application of our techniques the problem was reduced to a size that could be evaluated. For example, the first test case produced a *MRPS* of size 2^{292} and an AFSM of size 2^{692}. Through the use of COI and decomposition, we obtained two subproblems. The larger of these produced a *MRPS* of size 2^{26} and an AFSM of size 2^{66}, allowing the role containment problem to be proven unsatisfiable.

Clearly we cannot expect that all role containment problems will become tractable through the use of these techniques, however examination of such techniques is a significant contribution since it helps identify the characteristics of the more difficult problems. In particular, policies in which the queried roles are in a cycle and each role in the cycle is restricted in some way can be very difficult to reduce and evaluate. It is not difficult to understand that COI, URR, and Decomposition most likely will not be effective in such a case due to a significant overlap of restricted *DefRoles* of the queried roles. A second characteristic of difficult role containment problems has to do with a large number of principals introduced into the policy. This can occur in the *MRPS* through the introduction of new principals, but also in policies that happen to define restricted roles with many simple member statements. In the first case, we may be able to use principal abstraction to address part of the problem, but in the second case we will need a new abstraction technique. We suspect that such an abstraction may be possible as a result of future work. Finally, we reiterate a policy characteristic from Section 4.3.3 that suggests we may need many new principals in cases where the policy contains linked role expressions that share linked role names. For instance, if the policy were to contain the statements $A.r \leftarrow B.r_1.r_2$ and $X.u \leftarrow C.r_3.r_2$, then we may need sufficient principals to construct unique sub-linked roles using the linked role name r_2 for each linked role expression just in case this is necessary to expose a counterexample.

6. RELATED WORK

Security analysts of access control systems and policies have increasingly leveraged automated tool support to verify properties in support of security objectives. Jha et al. [10] verify such properties as authorization, availability, and shared access of the SPKI/SDSI policy language through the use of a language containment type of model checking. SPKI/SDSI is related to RT in that it can be reduced to a subset of the language, specifically the policy class $RT[\leftarrow]$, which does not support intersection inclusion statements. It is similar to our approach in that it is able to reason about policies that are unbounded in size, as well as describing specifications in a temporal logic.

Sistla et. al. [20, 21] provides a framework for reasoning about security analysis of dynamic RT policies. Of significant value is their proof of a tight EXPTIME complexity for role containment queries. Their approach focuses on using language containment to determine whether the membership of one role is contained in another role across policy states. It is unclear whether counterexamples can be constructed from this approach since no new principals are added to the evaluated policy.

Fisler et al. [4] analyzes the impact of policy changes on role-based access control (RBAC) systems using their Margrave tool. Such policies are represented as multi-terminal BDD's for efficient storage and manipulation. Unlike dynamic policy analysis which reasons about whether properties will hold despite future policy changes from unknown entities, their work focuses on specific policy changes by security administrators. They verify separation of duty properties, and they are able to produce counterexamples when a property is not satisfied.

Schaad et al. [19] also verifies separation of duty properties in RBAC systems, but uses a mature model checking tool called NuSMV. NuSMV is a model checker of the SMV family and supports additional features such as bounded model checking and step-wise exploration of the reachable state space. Their version of RBAC supports limited delegation and revocation of user permissions, though it does not support delegation of authorization as seen in modern trust management languages.

Other work [3, 5, 6, 7, 8, 13, 22, 23] also supports the use of formal tools to verify properties of security policies, however none develop a framework applicable to RT role containment problems.

7. CONCLUSIONS

Crafting and validating access control policies that reflect the author's intention is a difficult task for several reasons. First, stakeholders are generally neither security nor formal tool experts. They require an intuitive means of reasoning about the protection of their resources. Additionally, they require insightful counterexamples to describe why their policies fail to meet their expectations. Second, the complexity of large policies and intractability of certain cases makes manual security analysis an unreasonable approach. Thus tools that automate verification are clearly necessary.

Our contribution addresses these concerns through a collection of policy reduction and optimization techniques. Such techniques can be associated with automated model checking techniques, which always produce a counterexam-

ple when the policy fails to satisfy a property. This provides stakeholders the ability to verify properties without necessarily being experts in formal tools. Our contribution also includes proofs demonstrating correctness of our techniques, an empirical evaluation of our techniques showing the effectiveness and degree of efficiency of our approach, and analysis of the policy characteristics that remain difficult to verify. We propose a more thorough examination of principal abstraction and restriction relaxation techniques and proofs on correctness as future work.

Acknowledgements

William H. Winsborough is supported in part by NSF awards CCR-0325951, CCF-0524010, and CNS-0716750, and Texas Advanced Research Program award ARP 010115-0037-2007. Jianwei Niu is supported in part by Texas Advanced Research Program award ARP 010115-0037-2007, and University of Texas at San Antonio research award TRAC-2008.

8. REFERENCES

[1] E. M. Clarke, E. A. Emerson, and A. P. Sistla. Automatic verification of finite-state concurrent systems using temporal logic specifications. *ACM TOPLAS*, 8(2):244–263, 1986.

[2] E. M. Clarke, O. Grumberg, and D. A. Peled. *Model Checking*. The MIT Press, 1999.

[3] M. Drouineaud, M. Bortin, P. Torrini, and K. Sohr. A first step towards formal verification of security policy properties for RBAC. In *Proceedings of Fourth International Conference on Quality Software*, pages 60–67, 2004.

[4] K. Fisler, S. Krishnamurthi, L. A. Meyerovich, and M. C. Tshchantz. Verification and change-impact analysis of access-control policies. In *ICSE*, pages 196–205. ACM Press, 2005.

[5] D. Gilliam, J. Powell, and M. Bishop. Application of lightweight formal methods to software security. In *WETICE*, 2005.

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

[7] D. P. Guelev, M. Ryan, and P. Y. Schobbens. Model checking access control policies. In *Proceedings of the 7th Information Security Conference*, volume 3225 of *LNCS*. Springer-Verlag, 2004.

[8] F. Hansen and V. Oleshchuk. Conformance checking of RBAC policy and its implementation. In *Information Security Practice and Experience*, volume 3439 of *LNCS*, pages 144–155. Springer Berlin/Heidelberg, 2005.

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

[10] S. Jha and T. Reps. Model checking SPKI/SDSI. *Journal of Computer Security*, 12:317–353, 2004.

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

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

[13] M. J. May, C. A. Gunter, and I. Lee. Privacy APIs: Access control techniques to analyze and verify legal privacy policies. In *CSFW*, 2006.

[14] K. McMillan. *Symbolic Model Checking: An Approach to the State Explosion Problem*. Kluwer Academic, 1993.

[15] A. Pnueli. The temporal logic of programs. In *Proceedings of the 18th IEEE Symposium on Foundations of Computer Science*, volume 526, pages 46–67, 1977.

[16] M. Reith, J. Niu, and W. H. Winsborough. Apply model checking to security analysis in trust management. In *Proceedings of the First International Workshop on Security Technologies for Next Generation Collaborative Business Applications*, 2007.

[17] M. Reith, J. Niu, and W. H. Winsborough. Reductions and optimizations for the analysis of trust management policy. Technical Report CS-TR-2008-017, UTSA, 2008.

[18] R. Sandhu. The typed access matrix model. In *Symposium on Research in Security and Privacy*, pages 122–136. IEEE Computer Society, 1992.

[19] A. Schaad, V. Lotz, and K. Sohr. A model-checking approach to analysing organisational controls in a loan origination process. In *SACMAT '06: Proceedings of the eleventh ACM symposium on Access control models and technologies*, pages 139–149, New York, NY, USA, 2006. ACM Press.

[20] A. P. Sistla and M. Zhou. Analysis of dynamic policies. In *Proceedings of Foundations of Computer Security and Automated Reasoning for Security Protocol Analysis*, pages 233–262, 2006.

[21] A. P. Sistla and M. Zhou. Analysis of dynamic policies. *Information and Computation*, 206(2-4):185–212, 2008.

[22] N. Zhang, M. Ryan, and D. P. Guelev. Synthesising verified access control systems in XACML. In *FMSE*, pages 56–65. ACM Press, 2004.

[23] N. Zhang, M. Ryan, and D. P. Guelev. Evaluating access control policies through model checking. In *Proceedings of the 8th Information Security Conference*, volume 3650 of *LNCS*, pages 446–460. Springer-Verlag, 2005.