Automated Analysis of Access Control Policies

Alessandro Armando
joint work with Silvio Ranise

Artificial Intelligence Laboratory (Al-Lab)
DIST, University of Genova
Genova

Security & Trust Research Unit
FBK-IRST
Trento
The process of mediating requests to resources maintained by a system and determining whether a request should be granted or denied.

Crucial role in system security.

Usually separation between policies specified by a language with an underlying model and mechanisms enforcing policies.

Separation implies protection requirements are independent of their implementation and security policies can be analyzed abstractly.
### Role-based Access Control

<table>
<thead>
<tr>
<th>User</th>
<th>Permission</th>
</tr>
</thead>
<tbody>
<tr>
<td>Alice</td>
<td>GrantTenure</td>
</tr>
<tr>
<td>Alice</td>
<td>AssignGrades</td>
</tr>
<tr>
<td>Alice</td>
<td>ReceiveHBenefits</td>
</tr>
<tr>
<td>Alice</td>
<td>UseGym</td>
</tr>
<tr>
<td>Bob</td>
<td>GrantTenure</td>
</tr>
<tr>
<td>Bob</td>
<td>AssignGrades</td>
</tr>
<tr>
<td>Bob</td>
<td>UseGym</td>
</tr>
<tr>
<td>Charlie</td>
<td>GrantTenure</td>
</tr>
<tr>
<td>Charlie</td>
<td>AssignGrades</td>
</tr>
<tr>
<td>Charlie</td>
<td>UseGym</td>
</tr>
<tr>
<td>David</td>
<td>AssignHWScores</td>
</tr>
<tr>
<td>David</td>
<td>Register4Courses</td>
</tr>
<tr>
<td>David</td>
<td>UseGym</td>
</tr>
<tr>
<td>Eve</td>
<td>ReceiveHBenefits</td>
</tr>
<tr>
<td>Eve</td>
<td>UseGym</td>
</tr>
<tr>
<td>Fred</td>
<td>Register4Courses</td>
</tr>
<tr>
<td>Fred</td>
<td>UseGym</td>
</tr>
<tr>
<td>Greg</td>
<td>UseGym</td>
</tr>
</tbody>
</table>
Role-based Access Control

User Assignment (UA)

<table>
<thead>
<tr>
<th>User</th>
<th>Role</th>
</tr>
</thead>
<tbody>
<tr>
<td>Alice</td>
<td>PCMember</td>
</tr>
<tr>
<td>Bob</td>
<td>Faculty</td>
</tr>
<tr>
<td>Charlie</td>
<td>Faculty</td>
</tr>
<tr>
<td>David</td>
<td>TA</td>
</tr>
<tr>
<td>David</td>
<td>Student</td>
</tr>
<tr>
<td>Eve</td>
<td>UEmployee</td>
</tr>
<tr>
<td>Fred</td>
<td>Student</td>
</tr>
<tr>
<td>Greg</td>
<td>UMember</td>
</tr>
</tbody>
</table>

Permission Assignment (PA)

<table>
<thead>
<tr>
<th>Role</th>
<th>Permission</th>
</tr>
</thead>
<tbody>
<tr>
<td>PCMember</td>
<td>GrantTenure</td>
</tr>
<tr>
<td>PCMember</td>
<td>AssignGrades</td>
</tr>
<tr>
<td>PCMember</td>
<td>ReceiveHBenefits</td>
</tr>
<tr>
<td>PCMember</td>
<td>UseGym</td>
</tr>
<tr>
<td>Faculty</td>
<td>AssignGrades</td>
</tr>
<tr>
<td>Faculty</td>
<td>ReceiveHBenefits</td>
</tr>
<tr>
<td>Faculty</td>
<td>UseGym</td>
</tr>
<tr>
<td>TA</td>
<td>AssignHWScores</td>
</tr>
<tr>
<td>TA</td>
<td>Register4Courses</td>
</tr>
<tr>
<td>TA</td>
<td>UseGym</td>
</tr>
<tr>
<td>UEmployee</td>
<td>ReceiveHBenefits</td>
</tr>
<tr>
<td>UEmployee</td>
<td>UseGym</td>
</tr>
<tr>
<td>Student</td>
<td>Register4Courses</td>
</tr>
<tr>
<td>Student</td>
<td>UseGym</td>
</tr>
<tr>
<td>UMember</td>
<td>UseGym</td>
</tr>
</tbody>
</table>
Role-based Access Control

### User Assignment (UA)

<table>
<thead>
<tr>
<th>User</th>
<th>Role</th>
</tr>
</thead>
<tbody>
<tr>
<td>Alice</td>
<td>PCMember</td>
</tr>
<tr>
<td>Bob</td>
<td>Faculty</td>
</tr>
<tr>
<td>Charlie</td>
<td>Faculty</td>
</tr>
<tr>
<td>David</td>
<td>TA</td>
</tr>
<tr>
<td>David</td>
<td>Student</td>
</tr>
<tr>
<td>Eve</td>
<td>UEmployee</td>
</tr>
<tr>
<td>Fred</td>
<td>Student</td>
</tr>
<tr>
<td>Greg</td>
<td>UMember</td>
</tr>
</tbody>
</table>

### Permission Assignment (PA)

<table>
<thead>
<tr>
<th>Role</th>
<th>Permission</th>
</tr>
</thead>
<tbody>
<tr>
<td>PCMember</td>
<td>GrantTenure</td>
</tr>
<tr>
<td>Faculty</td>
<td>AssignGrades</td>
</tr>
<tr>
<td>TA</td>
<td>AssignHWScores</td>
</tr>
<tr>
<td>UEmployee</td>
<td>ReceiveHBenefits</td>
</tr>
<tr>
<td>Student</td>
<td>Register4Courses</td>
</tr>
<tr>
<td>UMember</td>
<td>UseGym</td>
</tr>
</tbody>
</table>

### Role Hierarchy (≿)

- PCMember
- Faculty
- TA
- PTEmployee
- FTEmployee
- UEmployee
- Student
- UMember
Changes to RBAC policies subject to administrative policy.

Several administrative models for RBAC: ARBAC97, SARBAC, Oracle DBMS, UARBAC, ...

Key issue: definition of administrative domains, e.g.

- ARBAC: admin. domain = role-based
- UARBAC: admin. domain = attribute-based
In URA97, administrative actions can only modify the User Assignment (UA) relation.

- **can_assign:**
  
  $\text{UEmployee} : \{ \text{Student, TA} \} \rightarrow +\text{PTEmployee}$

- **can_revoke:**
  
  $\text{UEmployee} : \{ \text{Student} \} \rightarrow -\text{Student}$

- Static Mutually Exclusive Roles (SMER): $\text{SMER(TA, PTEmployee)}$
ARBAC97: URA97 sub-model

In URA97, administrative actions can only modify the User Assignment (UA) relation.

- **can_assign:**
  \[
  UEmployee : \{ Student, TA \} \implies +PTEmployee
  \]

- **can_revoke:**
  \[
  UEmployee : \{ Student \} \implies -Student
  \]

- Static Mutually Exclusive Roles (SMER):
  \[
  SMER(TA, PTEmployee)
  \]
In URA97, administrative actions can only modify the User Assignment (UA) relation.

- **can_assign:**
  - $\text{UEmployee} : \{ \text{Student, TA} \} \implies +\text{PTEmployee}$

- **can_revoke:**
  - $\text{UEmployee} : \{ \text{Student} \} \implies -\text{Student}$

- Static Mutually Exclusive Roles (SMER): $\text{SMER}(\text{TA, PTEmployee})$
In URA97, administrative actions can only modify the User Assignment (UA) relation.

- **can_assign:**
  \[ \text{UEmployee} : \{ \text{Student, TA} \} \implies + \text{PTEmployee} \]

- **can_revoke:**
  \[ \text{UEmployee} : \{ \text{Student} \} \implies - \text{Student} \]

- Static Mutually Exclusive Roles (SMER): \[ \text{SMER}(\text{TA, PTEmployee}) \]

<table>
<thead>
<tr>
<th>User</th>
<th>Role</th>
</tr>
</thead>
<tbody>
<tr>
<td>Alice</td>
<td>PCMember</td>
</tr>
<tr>
<td>Bob</td>
<td>Faculty</td>
</tr>
<tr>
<td>Charlie</td>
<td>Faculty</td>
</tr>
<tr>
<td>David</td>
<td>TA</td>
</tr>
<tr>
<td>David</td>
<td>Student</td>
</tr>
<tr>
<td>Eve</td>
<td>UEmployee</td>
</tr>
<tr>
<td>Fred</td>
<td>Student</td>
</tr>
<tr>
<td>Fred</td>
<td>PTEmployee</td>
</tr>
<tr>
<td>Greg</td>
<td>UMember</td>
</tr>
</tbody>
</table>
In URA97, administrative actions can only modify the User Assignment (UA) relation.

- **can_assign:**
  \[ \text{UEmployee} : \{ \text{Student, TA} \} \rightarrow +\text{PTEmployee} \]

- **can_revoke:**
  \[ \text{UEmployee} : \{ \text{Student} \} \rightarrow -\text{Student} \]

- Static Mutually Exclusive Roles (SMER): \[ \text{SMER}(\text{TA, PTEmployee}) \]
In URA97, administrative actions can only modify the User Assignment (UA) relation.

- **can_assign:**
  \[\text{UEmployee} : \{ \text{Student, TA} \} \implies +\text{PTEmployee} \]

- **can_revoke:**
  \[\text{UEmployee} : \{ \text{Student} \} \implies -\text{Student} \]

- Static Mutually Exclusive Roles (SMER):
  \[\text{SMER}( \text{TA, PTEmployee} ) \]
(A)RBAC model simplifies specification and administration of access control policies.

Yet, in large systems (e.g., Dresdner bank: 40,000 users and 1,400 permissions), administration of RBAC policies can be very difficult.

**Question:** Starting from an initial RBAC policy and using the administrative actions in the ARBAC policy, is there a way to grant Alice access to salaries.xls?

To predict the effects of changes on policies of real-world complexity by manual inspection is unfeasible: automated support needed!
Let $\psi$ be an administrative policy.

1. **(Bounded) user-role reachability problem:** Given (an integer $k \geq 0$, resp.) an initial RBAC policy, and a role $r$, does there exist a sequence of administrative actions in $\psi$ (of length $k$, resp) assigning a user $u$ to role $r$?

2. **Role containment:** Given an initial RBAC policy and two roles $r_1$ and $r_2$, does every member of role $r_1$ also belong to role $r_2$ in all reachable policies by applying finite sequences of administrative actions in $\psi$?

3. **Weakest precondition:** Given a user $u$ and a role $r$, compute the minimal set of RBAC policies from which a sequence of administrative actions in $\psi$ can make $u$ a member of role $r$.

4. **Inductive policy invariant:** Check if a property remain unaffected under any (finite) sequence of administrative actions in $\psi$. 

A. Armando (U. of Genova & FBK-IRST)
Symbolic Reachability Analysis of ARBAC Policies

Symbolic Representation of RBAC Policies

Symbolic representation of RBAC policies and properties, using a decidable fragment of (many-sorted) first-order logic.

- **Sorts:** User, Role
- **Predicate symbols:**
  - \( ua : \text{User} \times \text{Role} \) (flexible)
  - \( \succeq : \text{Role} \times \text{Role} \) (rigid)

**Defining \( ua \):**

\[
\forall u, r . (ua(u, r) \iff (u = u_1 \land r = \text{Role 1}) \lor (u = u_2 \land r = \text{Role 2}) \lor (u = u_3 \land r = \text{Role 3}) \lor \ldots)
\]

**Defining \( \succeq \):**

- \( \text{TA} \succeq \text{Student} \),
- \( \text{PTEmployee} \succeq \text{UEmployee} \),
- \( \text{UEmployee} \succeq \text{UMember} \), ...

\[
\forall r . (r \succeq r) \\
\forall r_1, r_2, r_3 . ((r_1 \succeq r_2 \land r_2 \succeq r_3) \Rightarrow r_1 \succeq r_3) \\
\forall r_1, r_2 . ((r_1 \succeq r_2 \land r_2 \succeq r_1) \Rightarrow r_1 = r_2)
\]
Symbolic Representation of RBAC Policies

- **SMER Constraints:** No user can be TA and PTEmployee at the same time:

  \[ \forall u. \neg (ua(u, \text{TA}) \land ua(u, \text{PTEmployee})) \]

- **Queries:** There exists a user who is member of a certain role:

  \[ \exists u, r. (ua(u, r) \land r \geq \text{Student}) \]
Symbolic Representation of ARBAC Policies

\[ U\text{Employee} \ : \ \{ \text{Student}, \overline{\text{TA}} \} \implies +\text{PTEmployee} \]

\[ \exists u_a, r_a. (\text{ua}(u_a, r_a) \land r_a \succeq U\text{Employee}) \land \]

\[ \exists u. \left( \text{ua}(u, \text{Student}) \land \forall r_2. (r_2 \succeq \text{TA} \implies \neg \text{ua}(u, r_2)) \land \right. \]

\[ \forall x, y. (\text{ua}'(x, y) \iff ((x = u \land y = \text{PTEmployee}) \lor \text{ua}(x, y))) \right) \]

\[ U\text{Employee} \ : \ \{ \text{Student} \} \implies -\text{Student} \]

\[ \exists u_a, r_a. (\text{ua}(u_a, r_a) \land r_a \succeq U\text{Employee}) \land \]

\[ \exists u. \left( \exists r_1. (\text{ua}(u, r_1) \land r_1 \succeq \text{Student}) \land \right. \]

\[ \forall x, y. (\text{ua}'(x, y) \iff (\neg (x = u \land y = \text{Student}) \land \text{ua}(x, y))) \right) \]
Given an integer $k \geq 0$ and symbolic representation of

- $T_{RBAC} = \text{theory constraining RBAC policies (⪰, SMER constraints)}$
- $I(ua) = \text{initial RBAC policy}$
- $G(ua) = \text{user } u \text{ is a member of role } r$
- $\tau(ua, ua') = \text{administrative actions in } \psi$

Check the satisfiability of

$$T_{RBAC} \land l(ua_0) \land \tau(ua_0, ua_1) \land \cdots \land \tau(ua_{k-1}, ua_k) \land G(ua_k)$$

Can be reduced to the satisfiability of

Bernays-Shönfinkel-Ramsey formulae

$\implies$ Decidable!
Security analysis: unbounded user-role reachability (I)

Given symbolic representation of

- $T_{RBAC}$ = theory constraining RBAC policies
- $l(ua)$ = initial RBAC policy
- $G(ua)$ = user $u$ is a member of role $r$
- $\tau(ua, ua')$ = administrative actions in $\psi$

Run a **symbolic backward reachability** procedure

- $R_0(ua) := G(ua)$ (goal)
- $R_{i+1}(ua) := \exists ua'. (R_i(ua') \land \tau(ua, ua'))$ (pre-image) for $i \geq 0$

Three requirements

1. **Effective computation** of BSR formulae for pre-images
2. **Decidability** of satisfiability of $(R_i \land l)$ (safety) and validity of $(R_{i+1} \Rightarrow R_i)$ (fix-point), both modulo $T_{RBAC}$
3. **Termination** of backward reachability
Effective computation of pre-images

if pre-processing of negation in pre-conditions of administrative actions to eliminate $\forall$

Satisfiability of $(R_i \land I)$ and validity of $(R_{i+1} \Rightarrow R_i)$ modulo $T_{RBAC}$

can be reduced to satisfiability of BSR formulae $\implies$ Decidable!

Termination of backward reachability

by model-theoretic methods in combination with results on well-quasi-order
Decidability of parameterized user-role reachability with respect to the number of users

- Role containment and weakest precondition can be reduced to unbounded user-role reachability
- Inductive policy invariant can be reduced to bounded user-role reachability

Extensions
- Parametric roles (limited use of negation in pre-conditions of administrative actions)
- Attributes (crucial for distributed and open environments)
Tool **ASASP**: Automated Symbolic Analysis of Administrative Policies

- **architecture**: client-server
- **client**: pre-image computation + generation of logical problems
- **server**: state-of-the-art SMT solvers and theorem provers on satisfiability problems.
  - **Z3**, incomplete over BSR but incremental
  - **SPASS** (refutation) complete but not incremental
  - hierarchical combination

- Benchmarks for unbounded user-role reachability by Stoller *et al*
  - Parameter: **goal size**
  - **Better scalability** wrt. tool by Stoller *et al*

- Tool and benchmarks publicly available at [http://st.fbk.eu](http://st.fbk.eu)
No role hierarchy
With role hierarchy

Goal size = 1

Goal size = 2
With role hierarchy

Goal size = 3

Goal size = 4
Symbolic Reachability Analysis of Personal Health Record Policies

Increasingly large number of security-sensitive applications for e-business, e-health, and e-government are available and routinely used by the general public.

Regulate the access to sensitive data (e.g., health records) handled by these applications is a growing concern.

Traditional access control models are unsatisfactory:

- **Policy Administration**: Separation between policy and policy administration is usually assumed (c.f. ARBAC).

- **Policy Integration**: With the advent of the SaaS paradigm, users may give third-party applications access to their own data. The policy may thus span several applications.
Understanding the implications of the PHRs policies goes beyond the ability of a security administrator, let alone an average user.

Automatic analysis techniques and tools for policies are therefore key.
An access control policy of user $u_o$ is a tuple $\pi = (u_o, U, R, P, UA, PA)$, where

- $U$ is the set of the user accounts and $u_o \in U$;
- $R$ is a set of roles endowed with the hierarchy relation $\sqsubseteq_R$;
- $UA \subseteq (U \times R)$ is the user-role assignment relation;
- $P = (Act \times Res)$ is the set of permissions, where
  - $Act$ is a set of actions endowed with the hierarchy relation $\sqsubseteq_{Act}$;
  - $Res$ is the set of resources endowed with the hierarchy relation $\sqsubseteq_{Res}$;
- $PA = ((U \cup R) \times P)$ is the permission assignment relation.
An access control policy of user $u_0$ is a tuple $\pi = (u_0, U, R, P, UA, PA)$, where

- $U$ is the set of the user accounts and $u_0 \in U$;
- $R$ is a set of roles endowed with the hierarchy relation $\sqsubseteq_R$;
- $UA \subseteq (U \times R)$ is the user-role assignment relation;
- $P = (Act \times Res)$ is the set of permissions, where
  - $Act$ is a set of actions endowed with the hierarchy relation $\sqsubseteq_{Act}$;
  - $Res$ is the set of resources endowed with the hierarchy relation $\sqsubseteq_{Res}$;
- $PA = ((U \cup R) \times P)$ is the permission assignment relation.
An access control policy of user $u_o$ is a tuple $\pi = (u_o, U, R, P, UA, PA)$, where

- $U$ is the set of user accounts and $u_o \in U$;
- $R$ is a set of roles endowed with the hierarchy relation $\sqsubseteq R$;
- $UA \subseteq (U \times R)$ is the user-role assignment relation;
- $P = (Act \times Res)$ is the set of permissions, where $Act$ is a set of actions endowed with the hierarchy relation $\sqsubseteq Act$ and $Res$ is the set of resources endowed with the hierarchy relation $\sqsubseteq Res$;
- $PA = ((U \cup R) \times P)$ is the permission assignment relation.

Family Member
- Parent
- Child
- Sibling

Health Care Provider
- Physician
- Registered Nurse
- Physical Therapist

External Application

Trusted

Role Hierarchy

...
An access control policy of user $u_o$ is a tuple $\pi = (u_o, U, R, P, UA, PA)$, where

- $U$ is the set of the user accounts and $u_o \in U$;
- $R$ is a set of roles endowed with the hierarchy relation $\sqsupseteq_R$;
- $UA \subseteq (U \times R)$ is the user-role assignment relation;
- $P = (Act \times Res)$ is the set of permissions, where
  - $Act$ is a set of actions endowed with the hierarchy relation $\sqsupseteq_{Act}$;
  - $Res$ is the set of resources endowed with the hierarchy relation $\sqsupseteq_{Res}$;
- $PA = ((U \cup R) \times P)$ is the permission assignment relation.
An access control policy of user $u_0$ is a tuple $\pi = (u_0, U, R, P, UA, PA)$, where

- $U$ is the set of the user accounts and $u_0 \in U$;
- $R$ is a set of roles endowed with the hierarchy relation $\sqsubseteq_R$;
- $UA \subseteq (U \times R)$ is the user-role assignment relation;
- $P = (Act \times Res)$ is the set of permissions, where
  - $Act$ is a set of actions endowed with the hierarchy relation $\sqsubseteq_{Act}$;
  - $Res$ is the set of resources endowed with the hierarchy relation $\sqsubseteq_{Res}$;
- $PA = ((U \cup R) \times P)$ is the permission assignment relation.
An access control policy of user $u_o$ is a tuple $\pi = (u_o, U, R, P, UA, PA)$, where

- $U$ is the set of the user accounts and $u_o \in U$;
- $R$ is a set of roles endowed with the hierarchy relation $\sqsubseteq_R$;
- $UA \subseteq (U \times R)$ is the user-role assignment relation;
- $P = (Act \times Res)$ is the set of permissions, where
  - $Act$ is a set of actions endowed with the hierarchy relation $\sqsubseteq_{Act}$;
  - $Res$ is the set of resources endowed with the hierarchy relation $\sqsubseteq_{Res}$;
- $PA = ((U \cup R) \times P)$ is the permission assignment relation.
An access control policy of user $u_o$ is a tuple $\pi = (u_o, U, R, P, UA, PA)$, where $U$ is the set of user accounts and $u_o \in U$; $R$ is a set of roles endowed with the hierarchy relation $\sqsubseteq R$; $UA \subseteq (U \times R)$ is the user-role assignment relation; $P = (Act \times Res)$ is the set of permissions, where $Act$ is a set of actions endowed with the hierarchy relation $\sqsubseteq Act$; $Res$ is the set of resources endowed with the hierarchy relation $\sqsubseteq Res$. $PA = ((U \cup R) \times P)$ is the permission assignment relation.
An access control policy of user $u_o$ is a tuple $\pi = (u_o, U, R, P, UA, PA)$, where

- $U$ is the set of the user accounts and $u_o \in U$;
- $R$ is a set of roles endowed with the hierarchy relation $\sqsubseteq_R$;
- $UA \subseteq (U \times R)$ is the user-role assignment relation;
- $P = (Act \times Res)$ is the set of permissions, where
  - $Act$ is a set of actions endowed with the hierarchy relation $\sqsubseteq_{Act}$;
  - $Res$ is the set of resources endowed with the hierarchy relation $\sqsubseteq_{Res}$;
- $PA = ((U \cup R) \times P)$ is the permission assignment relation.
An Access Control Model for Personal Health Records

An access control policy of user $u_o$ is a tuple $\pi = (u_o, U, R, P, UA, PA)$, where $U$ is the set of the user accounts and $u_o \in U$; $R$ is a set of roles endowed with the hierarchy relation $\sqsubseteq_R$; $UA \subseteq (U \times R)$ is the user-role assignment relation; $P = (Act \times Res)$ is the set of permissions, where $Act$ is a set of actions endowed with the hierarchy relation $\sqsubseteq_{Act}$; $Res$ is the set of resources endowed with the hierarchy relation $\sqsubseteq_{Res}$; $PA = ((U \cup R) \times P)$ is the permission assignment relation.
An access control policy of user $u_o$ is a tuple $\pi = (u_o, U, R, P, UA, PA)$, where

- $U$ is the set of the user accounts and $u_o \in U$;
- $R$ is a set of roles endowed with the hierarchy relation $\sqsubseteq_R$;
- $UA \subseteq (U \times R)$ is the user-role assignment relation;
- $P = (Act \times Res)$ is the set of permissions, where
  - $Act$ is a set of actions endowed with the hierarchy relation $\sqsubseteq_{Act}$;
  - $Res$ is the set of resources endowed with the hierarchy relation $\sqsubseteq_{Res}$;
- $PA = ((U \cup R) \times P)$ is the permission assignment relation.
A user \( u \) can execute \( act \) on \( res \) in \( \pi \) iff

1. \((u, p) \in PA\) for some permission \( p \) such that \( p \sqsupseteq_P (act, res) \), or
2. there exist roles \( r, r' \in R \) such that \((u, r) \in UA\), \( r \sqsupseteq_R r' \), and \((r', p) \in PA\) for some permission \( p \) such that \( p \sqsupseteq_P (act, res) \).

**Example:**

If Bob is the owner and \((Alice, p) \in PA\) with \( p = (RecordViewing, MedicalEvents) \), then Alice can view Bob’s MedicalEvents (including his MedicalAppointments and MedicationAdministrations because of the resource hierarchy).
The policy also allows for the specification of administration privileges.

For instance, if Bob is the owner and Alice is assigned the permission (Grant(RecordViewing), MedicalEvents), then Alice can grant the privilege to viewing Bob’s MedicalEvents to any other user.

In other words, Alice can change $PA$ into $PA' = PA \cup \{(u, (RecordViewing, MedicalEvents))\}$ for some arbitrary user $u \in U$.

This is useful, but too liberal.

For instance, Bob might be willing to delegate Alice the permission to grant privileges to viewing his MedicalEvents only to those Physicians that are not relatives of him.
Administering the Policy: Delegation

The policy also allows for the specification of administration privileges. For instance, if Bob is the owner and Alice is assigned the permission \( \text{Grant}(\text{RecordViewing}, \text{MedicalEvents}) \), then Alice can grant the privilege to viewing Bob’s MedicalEvents to any other user. In other words, Alice can change \( PA \) into \( PA' = PA \cup \{ (u, \text{RecordViewing}, \text{MedicalEvents}) \} \) for some arbitrary user \( u \in U \).

This is useful, but too liberal. For instance, Bob might be willing to delegate Alice the permission to grant privileges to viewing his MedicalEvents only to those Physicians that are not relatives of him.

### Action Hierarchy

- AllOperations
  - RecordModification
    - InsertRecord
    - AnnotateRecord
    - UpdateRecord
    - DeleteRecord
    - MaskRecord
    - ReadRecord
    - ReadAnnotation
    - ReadRecordHistory
    - Grant/Revoke(RecordModification)
  - RecordViewing
    - ReadRecord
    - ReadAnnotation
    - ReadRecordHistory
    - Grant/Revoke(RecordViewing)
  - RecordAdministration
    - Grant/Revoke(RecordModification)
Administering the Policy: Delegation

The policy also allows for the specification of administration privileges. For instance, if Bob is the owner and Alice is assigned the permission \((\text{Grant}(\text{RecordViewing}), \text{MedicalEvents})\), then Alice can grant the privilege to viewing Bob’s MedicalEvents to any other user.

In other words, Alice can change \(PA\) into \(PA' = PA \cup \{(u, (\text{RecordViewing}, \text{MedicalEvents}))\}\) for some arbitrary user \(u \in U\).

This is useful, but too liberal. For instance, Bob might be willing to delegate Alice the permission to grant privileges to viewing his MedicalEvents only to those Physicians that are not relatives of him.

AllOperations
RecordModification
RecordViewing
RecordAdministration
InsertRecord
AnnotateRecord
UpdateRecord
DeleteRecord
MaskRecord
ReadRecord
ReadAnnotation
ReadRecordHistory
Grant/Revoke(RecordModification)
Grant/Revoke(RecordViewing)

A. Armando (U. of Genova & FBK-IRST)
Automated Analysis of Access Control
VTSA11, Sept. 23, 2011 26 / 52
The policy also allows for the specification of administration privileges.

For instance, if Bob is the owner and Alice is assigned the permission \((\text{Grant} (\text{RecordViewing}), \text{MedicalEvents})\), then Alice can grant the privilege to viewing Bob’s MedicalEvents to any other user.

In other words, Alice can change \(PA\) into \(PA' = PA \cup \{(u, (\text{RecordViewing}, \text{MedicalEvents}))\}\) for some arbitrary user \(u \in U\).

This is useful, but too liberal.

For instance, Bob might be willing to delegate Alice the permission to grant privileges to viewing his MedicalEvents only to those Physicians that are not relatives of him.
The policy also allows for the specification of administration privileges.

For instance, if Bob is the owner and Alice is assigned the permission (Grant(RecordViewing), MedicalEvents), then Alice can grant the privilege to viewing Bob’s MedicalEvents to any other user.

In other words, Alice can change $PA$ into $PA' = PA \cup \{(u, (RecordViewing, MedicalEvents))\}$ for some arbitrary user $u \in U$.

This is useful, but too liberal.

For instance, Bob might be willing to delegate Alice the permission to grant privileges to viewing his MedicalEvents only to those Physicians that are not relatives of him.
The policy also allows for the specification of administration privileges.

For instance, if Bob is the owner and Alice is assigned the permission \(\text{Grant(RecordViewing), MedicalEvents}\), then Alice can grant the privilege to viewing Bob’s MedicalEvents to any other user.

In other words, Alice can change \(PA\) into:

\[ PA' = PA \cup \{(u, (\text{RecordViewing}, \text{MedicalEvents}))\} \]

for some arbitrary user \(u \in U\).

This is useful, but too liberal.

For instance, Bob might be willing to delegate Alice the permission to grant privileges to viewing his MedicalEvents only to those Physicians that are not relatives of him.
The policy also allows for the specification of administration privileges.

For instance, if Bob is the owner and Alice is assigned the permission \( \text{Grant(RecordViewing), MedicalEvents} \), then Alice can grant the privilege to viewing Bob’s MedicalEvents to any other user.

In other words, Alice can change \( PA \) into \( PA' = PA \cup \{(u, (\text{RecordViewing, MedicalEvents})) \} \) for some arbitrary user \( u \in U \).

This is useful, but too liberal.

For instance, Bob might be willing to delegate Alice the permission to grant privileges to viewing his MedicalEvents only to those Physicians that are not relatives of him.
**Idea:** Add conditions to permissions.

For instance, if Alice is assigned the permission

\[(\text{Grant}(\text{RecordViewing}) \text{ to } \{+\text{Physician}, -\text{FamilyMember}\}, \text{MedicalEvents})\]

then Alice can add \((u, (\text{RecordViewing}, \text{MedicalEvents}))\) to \(PA\) for any \(u \in U\) such that

- \((u, r) \in UA\) for some \(r \in R\) such that \(\text{Physician} \sqsupseteq_R r\) and
- \((u, r) \notin UA\) for all \(r \in R\) such that \(\text{FamilyMember} \sqsupseteq_R r\).
Administering the Policy: Problem

- Consider the situation in which Bob wants to delegate Alice the right to grant viewing privileges only to physicians he trusts.
- By assigning Alice the permission
  \[(\text{Grant(RecordViewing)} \to \{+\text{Physician}, +\text{ Trusted}\}, \text{MedicalEvents})\]

Bob could conclude that only physicians he trusts could access his MedicalEvents. But this is not necessarily the case.

- If Charlie was trusted by Bob, then Alice might have granted Charlie the right to modify Bob’s MedicalEvents, but Charlie can keep this privilege even if he is no longer trusted by Bob.
- Morale: it can be difficult to predict the effects of delegations and this may lead the user to draw wrong conclusions.
  \[\Rightarrow \text{Automated support needed!}\]
Consider the situation in which Bob wants to delegate Alice the right to grant viewing privileges only to physicians he trusts. By assigning Alice the permission

\[
\text{Grant(RecordViewing) to } \{+\text{Physician}, +\text{Trusted}\}, \text{MedicalEvents}
\]

Bob could conclude that only physicians he trusts could access his MedicalEvents. But this is not necessarily the case.

If Charlie was trusted by Bob, then Alice might have granted Charlie the right to modify Bob’s MedicalEvents, but Charlie can keep this privilege even if he is no longer trusted by Bob.

Morale: it can be difficult to predict the effects of delegations and this may lead the user to draw wrong conclusions. ⇒ Automated support needed!
Consider the situation in which Bob wants to delegate Alice the right to grant viewing privileges only to physicians he trusts. By assigning Alice the permission

\[
(\text{Grant}(\text{RecordViewing}) \text{ to } \{+\text{Physician}, +\text{Trusted}\}, \text{MedicalEvents})
\]

Bob could conclude that only physicians he trusts could access his MedicalEvents. **But this is not necessarily the case.**

If Charlie was trusted by Bob, then Alice might have granted Charlie the right to modify Bob’s MedicalEvents, but Charlie can keep this privilege even if he is no longer trusted by Bob.

Morale: it can be difficult to predict the effects of delegations and this may lead the user to draw wrong conclusions. ⇒ Automated support needed!
Consider the situation in which Bob wants to delegate Alice the right to grant viewing privileges only to physicians he trusts.

By assigning Alice the permission

\[
(\text{Grant(RecordViewing)} \to \{ +\text{Physician}, +\text{Trusted}\}, \text{MedicalEvents})
\]

Bob could conclude that only physicians he trusts could access his MedicalEvents. But this is not necessarily the case.

If Charlie was trusted by Bob, then Alice might have granted Charlie the right to modify Bob’s MedicalEvents, but Charlie can keep this privilege even if he is no longer trusted by Bob.

Morale: it can be difficult to predict the effects of delegations and this may lead the user to draw wrong conclusions. ⇒ Automated support needed!
If $\pi'$ can be reached from $\pi$, then we write $\pi \rightarrow \pi'$.

$\pi_n$ is reachable from $\pi_0$ (in symbols, $\pi \rightarrow^* \pi'$) iff there exist $\pi_0, \pi_1, \ldots, \pi_{n-1}, \pi_n$ s.t. $\pi_i \rightarrow \pi_{i+1}$ for $i = 0, \ldots, n-1$.

A query is triple of the form $(u, \text{act}, \text{res})$ where $u \in U$, $\text{act} \in \text{Act}$ and $\text{res} \in \text{Res}$.

**PHR reachability problem:** Given a query $(u, \text{act}, \text{res})$ and a policy $\pi$, determine whether there exists $\pi'$ such that $\pi \rightarrow^* \pi'$ and $u$ can execute $\text{act}$ on $\text{res}$ in $\pi'$.
Use first-order formulae to symbolically represent:
- sets of policies, $R_k^i(pa)$
- transitions between them, $\tau(pa, pa')$.

Backward search: compute nodes as follows:
- $R^0(pa) := G(pa)$ (goal)
- $R^{i+1}(pa) := \exists pa'. (R^i(pa') \land \tau(pa, pa'))$ (pre-image) for $i \geq 0$

until
- we reach a formula $R_k^i$ whose denotation contains an initial state
  this is done by checking the satisfiability of $R_k^i \land I$, or
- we reach a fix-point
  this is done by checking the validity of $\bigvee_k R_{k+1}^i \Rightarrow \bigvee_k R_k^i$. 

\[ G \]

\[ \text{Diagram:} \]

\[ \text{Nodes connected by transitions, with initial state} \]

\[ \text{Goal node} \]
PHR policies can be specified by formulae of the Bernays-Schönfinkel-Ramsey (BSR) fragment, i.e. FOL formulae of the form

$$\exists x_1, \ldots, x_n. \forall y_1, \ldots, y_m. \varphi(x_1, \ldots, x_n, y_1, \ldots, y_m)$$

where $\varphi$ is a quantifier-free formula containing only individual constants and predicate symbols (no function symbols allowed).
Symbolic representation of PHR policies

\[ \forall r. R(r) \iff (r = \text{RecordSubject} \lor r = \text{RecordCustodian} \lor r = \text{AllOtherUsers} \lor \cdots) \]

\[ \text{RecordSubject} \neq \text{RecordCustodian}, \]
\[ \text{RecordSubject} \neq \text{AllOtherUsers}, \]
\[ \text{RecordCustodian} \neq \text{AllOtherUsers}, \ldots, \]

\[ \forall r_1, r_2. r_1 \sqsupseteq_R r_2 \Rightarrow (R(r_1) \land R(r_2)) \]
\[ \forall r. r \sqsupseteq_R r \]
\[ \forall r_1, r_2. (r_1 \sqsupseteq_R r_2 \land r_2 \sqsupseteq_R r_1) \Rightarrow r_1 = r_2 \]
\[ \forall r_1, r_2, r_3. (r_1 \sqsupseteq_R r_2 \land r_2 \sqsupseteq_R r_3) \Rightarrow r_1 \sqsupseteq_R r_3, \]
Symbolic representation of PHR policies

\[ \forall a, s, p. \text{ARP}(a, s, p) \Rightarrow (\text{Act}(a) \land \text{Res}(s) \land P(p)) \]
\[ \forall a, s, p, p'. (\text{ARP}(a, s, p) \land \text{ARP}(a, s, p')) \Rightarrow p = p' \]

\[ \forall a, s, p, a', s', p'. (\text{ARP}(a, s, p) \land \text{ARP}(a', s', p')) \Rightarrow (p \sqsupseteq_P p' \iff (a \sqsupseteq_{\text{Act}} a' \land s \sqsupseteq_{\text{Res}} s')) \]

A query \((u, a, s)\) is represented by the formula:

\[ \exists p. \text{ARP}(a, s, p) \Rightarrow \left( \exists p'. (\text{PA}(u, p') \land p' \sqsupseteq_P p) \lor \left( \exists r, r', p'. (\text{R}(r) \land \text{R}(r') \land \text{UA}(u, r) \land r \sqsupseteq_R r' \land \text{PA}(r', p') \land p' \sqsupseteq_P p) \right) \right) \]
The administrative action related to the pair \((\text{Alice}, p) \in \text{PA}\) with 
\[ p = (\text{Grant(RecordViewing)} \text{ to } \{+\text{Physician}, -\text{FamilyMember}\}, \text{MedicalEvents}) \]
is represented by the formula:

\[
\exists p. (P(p) \land ARP(\text{RecordViewing}, \text{MedicalEvents}, p) \land PA(\text{Alice}, p) \Rightarrow
\begin{align*}
&\exists r_1. (UA(u, r_1) \land \text{Physician} \sqsupseteq_R r_1) \land \\
&\exists u. \\
&\forall r_2. (\text{FamilyMember} \sqsupseteq_R r_2 \Rightarrow \neg UA(u, r_2)) \land \\
&\forall x, y. (PA'(x, y) \iff (PA(x, y) \lor (x = u \land y = p)))
\end{align*}
\]

A. Armando (U. of Genova & FBK-IRST)
For the procedure to be effective
- $\exists pa'. (R^i(pa') \land \tau(pa, pa'))$ must be turned into an equivalent BSR formula
- the satisfiability of $R^i_k \land I$ and the validity of $\bigvee_k R^{i+1}_k \Rightarrow \bigvee_k R^i_k$ must be decidable

We have shown that the above conditions are satisfied and that the backward reachability procedure terminates.
Automated Analysis of Infinite State Workflows with Access Control Policies

Workflow management used in several applications
- E-business
- E-health
- E-government
- Scientific computing
- ...

Workflow management specification
- What are the tasks?
- What is the order of execution of the tasks?
- Which data are manipulated by each task?
- Who performs the tasks?
What are the tasks?

- register insurance claim
- check A of insurance policy
- check B of damage reported
- assess the results of checks A and B
- approve the payment of damage
- reject the payment of damage

What is the order of execution of the tasks?
Simple example: insurance claim (data-flow)

Which data are manipulated by each task?

- custID: unique identifier for customer
- type: enumerated data-type for identifying type of damages
- amount: money requested for damage
- answA, answB: either “ok” or “nok”
- decision: either “grant” or “refuse”
**Insurance claim: Who performs the tasks?**

**Role-based Access Control (RBAC)**

<table>
<thead>
<tr>
<th>User</th>
<th>Role</th>
</tr>
</thead>
<tbody>
<tr>
<td>Anna</td>
<td>Customer Service</td>
</tr>
<tr>
<td>Adam</td>
<td>Customer Service</td>
</tr>
<tr>
<td>Benn</td>
<td>Specialist A</td>
</tr>
<tr>
<td>Beate</td>
<td>Specialist A</td>
</tr>
<tr>
<td>Beate</td>
<td>Specialist B</td>
</tr>
<tr>
<td>Carol</td>
<td>Specialist B</td>
</tr>
<tr>
<td>Chris</td>
<td>Specialist B</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Role</th>
<th>Task</th>
</tr>
</thead>
<tbody>
<tr>
<td>Customer Service</td>
<td>register</td>
</tr>
<tr>
<td>Customer Service</td>
<td>assess</td>
</tr>
<tr>
<td>Customer Service</td>
<td>approve</td>
</tr>
<tr>
<td>Customer Service</td>
<td>reject</td>
</tr>
<tr>
<td>Specialist A</td>
<td>check A</td>
</tr>
<tr>
<td>Specialist B</td>
<td>check B</td>
</tr>
</tbody>
</table>

- Can **Beate** perform task check A? Yes!
- Can **Benn** perform task check B? No!
- Can **Beate** perform task check B? Yes!
Insurance claim: more flexibility in access control

**Authorization constraints:** Separation/Bound of Duty (SoD/BoD)

- **SoD:** If amount is larger than 5 KEuros, then the same user cannot execute both tasks check A and check B.
- **BoD:** Task reject have to be performed by the same user who performed the task register.
Insurance claim: more flexibility in access control

- **Delegation of task execution**

- **Rule**: if amount is less than 5 K€uros, then user with role Specialist A can delegate the right to execute task check A to user with role Customer Service.

```
<table>
<thead>
<tr>
<th>User</th>
<th>Role</th>
</tr>
</thead>
<tbody>
<tr>
<td>Anna</td>
<td>Customer Service</td>
</tr>
<tr>
<td>Adam</td>
<td>Customer Service</td>
</tr>
<tr>
<td>Benn</td>
<td>Specialist A</td>
</tr>
<tr>
<td>Beate</td>
<td>Specialist A</td>
</tr>
<tr>
<td>Beate</td>
<td>Specialist B</td>
</tr>
<tr>
<td>Carol</td>
<td>Specialist B</td>
</tr>
<tr>
<td>Chris</td>
<td>Specialist B</td>
</tr>
</tbody>
</table>
```
safety = assurance that an access control configuration will not result in the leakage of a right to an unauthorized principal

Shown **undecidable** for general access control models by Harrison, Ruzzo, and Ulmann in a CACM paper (1978)
Safety and workflows

Safety problem = *given a workflow management specification, are all authorization constraints satisfied?*

- How many workflow instances?

- What about the situation where two or more workflow instances may communicate/synchronize?

- What kind of data-flows should we model?
  - Enumerated data-types?
  - Integers with ordering? (no operations)
  - Integers with all operations?
Our framework

Workflow schema = Extended Finite State Automata

\[
q \xrightarrow{\text{answA=ok} \land \text{answB=ok}} q' \quad \Rightarrow \quad \text{decision:=grant}; \quad \text{ex:=ex} \cup \{\text{assess}\}
\]

Access control = RBAC + Delegation + Authorization constraints

Policy : \langle \text{Users, Roles, Tasks, User-Role, Role-Task, Role hierarchy} \rangle

Rule : \text{amount < 5} : \text{Specialist A} \xrightarrow{\text{check A}} \text{Customer Service}

\ldots
Our framework: the safety problem

<table>
<thead>
<tr>
<th>RBAC session</th>
<th>RBAC session</th>
<th>⋯</th>
<th>RBAC session</th>
</tr>
</thead>
<tbody>
<tr>
<td>[ \text{id}_1 ]</td>
<td>[ \text{id}_2 ]</td>
<td>⋯</td>
<td>[ \text{id}_n ]</td>
</tr>
</tbody>
</table>

RBAC policy with delegation rules and authorization constraints

Can an untrusted user get the right to execute a certain sensitive task regardless of the number of workflow instances?

⇒ Undecidable in general, but...
Reasonable restrictions for decidability

- Manipulated data have “simple” algebraic structure
  - Technically in FOL: class of structures axiomatized by universal formulae over a relational signature

- Updates are of the forms: $\text{var} := \text{var}'$ or $\text{var} := \text{constant}$
  - I.e. they reflect the “simplicity of the data”

- Only finitely many (known) workflow instances can be involved in one transition
  - This implies broadcast actions (where finitely many but an unknown number of instances participate in a transition) cannot be modelled
Key Ideas in the proof of decidability

Symbolic representation by Bernays-Shönfinkel-Ramsey (BSR) formulae

Transition system: \( \langle V, \text{In}(V), \{\tau_h(V, V')\}_h \rangle \)

Error condition: \( E(V) \)

- \( V \) = state variables (automata location, user-role assignment per instance, user-role delegated assignment per instance ...)
- \( \tau_h \) = either a transition of the extended finite automata or a delegation rule
- Error condition = \( \exists u, t. \text{Untrusted}(u) \land \text{Sensitive}(t) \land \text{exec}(u, t) \)

Use “standard” backward reachability and prove mechanization and termination
Backward reachability: overview

Let $T := \bigvee_h \tau_h$. Iteratively compute

$$R_0(V) := \text{E}(V) \quad \text{and} \quad R_{j+1}(V) := R_j(V) \lor \exists V'. (R_j(V') \land T(V, V')) \quad \text{for} \ j \geq 0$$

$$\hat{R}_j(V) := R_j(V) \land AC(V)$$

Authorization constraints

At each iteration $j \geq 1$, check for fix-point

$$\forall V. (\hat{R}_j(V) \Rightarrow R_{j-1}(V)) \text{ is valid?}$$

and safety

$$\exists V. (\hat{R}_j(V) \land \text{ln}(V)) \text{ is unsatisfiable?}$$
Theorem

If

- formulae in \( \{ \text{In} \} \cup \text{AC} \) are universal BSR
- each \( \tau_h \) is an existential BSR with a “functional” update
- \( E \) is an existential BSR

then

1. existential BSR formulae are closed under pre-image computation
2. fix-point and safety checks are decidable

Proof:

- Easy: simple logical manipulations
- Easy: reduction to satisfiability of BSR formulae
Termination of backward reachability

Theorem

Under the same hypotheses for mechanization on \langle V, In(V), \{\tau_h(V, V')\}_h \rangle and E(V), the backward reachability procedure terminates.

Proof:

1. Translate \langle V, In(V), \{\tau_h(V, V')\}_h \rangle into an array-based system and E(V) into an existential formula for which it is known that backward reachability terminates

2. Show that the translation preserves satisfiability

3. Show that the translation and pre-image computation commute
Conclusions

- Formal semantics of access control models
- Uniform and declarative specification/verification framework
- Automatic symbolic analysis guaranteed to terminate
- Nice scalability results for ARBAC.
  Can they be brought to more sophisticated models?