Automated Analysis of Access Control Policies

Alessandro Armando

joint work with Silvio Ranise

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

Security & Trust Research Unit
FBK-IRST
Trento

Access Control

- 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
  - mechanisms enforcing policies
- Separation implies
  - protection requirements are independent of their implementation
  - security policies can be analyzed abstractly

Role-based Access Control

User	Permission
Alice	GrantTenure
Alice	AssignGrades
Alice	ReceiveHBenefits
Alice	UseGym
Bob	GrantTenure
Bob	AssignGrades
Bob	UseGym
Charlie	GrantTenure
Charlie	AssignGrades
Charlie	UseGym
David	AssignHWScores
David	Register4Courses
David	UseGym
Eve	ReceiveHBenefits
Eve	UseGym
Fred	Register4Courses
Fred	UseGym
Greg	UseGym

Role-based Access Control

User Assignment (UA)

User	Role
Alice	PCMember
Bob	Faculty
Charlie	Faculty
David	TA
David	Student
Eve	UEmployee
Fred	Student
Greg	UMember

Permission Assignment (PA)

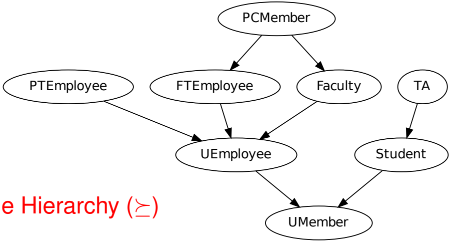
Role	Permission
PCMember	GrantTenure
PCMember	AssignGrades
PCMember	ReceiveHBenefits
PCMember	UseGym
Faculty	AssignGrades
Faculty	ReceiveHBenefits
Faculty	UseGym
TA	AssignHWScores
TA	Register4Courses
TA	UseGym
UEmployee	ReceiveHBenefits
UEmployee	UseGym
Student	Register4Courses
Student	UseGym
UMember	UseGym

Permission Assignment (PA)

Role	Permission
PCMember	GrantTenure
Faculty	AssignGrades
TA	AssignHWScores
UEmployee	ReceiveHBenefits
Student	Register4Courses
UMember	UseGym

User Assignment (UA)

User	Role
Alice	PCMember
Bob	Faculty
Charlie	Faculty
David	TA
David	Student
Eve	UEmployee
Fred	Student
Greg	UMember



Role Hierarchy ($\succeq$)

Administrative RBAC

- 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:
  $UEmployee \ : \ \{Student, \overline{TA}\} \Longrightarrow +PTEmployee$
- can_revoke:
  $UEmployee \ : \ \{Student\} \Longrightarrow -Student$
- Static Mutually Exclusive Roles (SMER):
  $SMER(TA, PTEmployee)$

User	Role
Alice	PCMember
Bob	Faculty
Charlie	Faculty
David	TA
David	Student
Eve	UEmployee
Fred	Student
Greg	UMember

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

- can_assign:
  $UEmployee : \{Student, \overline{TA}\} \Longrightarrow +PTEmployee$

- can_revoke:
  $UEmployee : \{Student\} \Longrightarrow -Student$

- Static Mutually Exclusive Roles (SMER):
  $SMER(TA, PTEmployee)$

User	**Role**
Alice	PCMember
Bob	Faculty
Charlie	Faculty
David	TA
David	Student
Eve	UEmployee
Fred	Student
Greg	UMember

ARBAC97: URA97 sub-model

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

- can_assign:
  $UEmployee \;:\; \{Student, \overline{TA}\} \Longrightarrow +PTEmployee$
- can_revoke:
  $UEmployee \;:\; \{Student\} \Longrightarrow -Student$
- Static Mutually Exclusive Roles (SMER):
  $SMER(TA, PTEmployee)$

User	Role
Alice	PCMember
Bob	Faculty
Charlie	Faculty
David	TA
David	Student
Eve	UEmployee
Fred	Student
Fred	PTEmployee
Greg	UMember

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

- can_assign:
  $UEmployee : \{Student, \overline{TA}\} \Longrightarrow +PTEmployee$
- can_revoke:
  $UEmployee : \{Student\} \Longrightarrow -Student$
- Static Mutually Exclusive Roles (SMER):
  $SMER(TA, PTEmployee)$

User	Role
Alice	PCMember
Bob	Faculty
Charlie	Faculty
David	TA
David	Student
Eve	UEmployee
Fred	Student
Fred	PTEmployee
Greg	UMember

ARBAC97: URA97 sub-model

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

- can_assign:
  $UEmployee : \{Student, \overline{TA}\} \Longrightarrow +PTEmployee$
- can_revoke:
  $UEmployee : \{Student\} \Longrightarrow -Student$
- Static Mutually Exclusive Roles (SMER):
  $SMER(TA, PTEmployee)$

User	Role
Alice	PCMember
Bob	Faculty
Charlie	Faculty
David	TA
David	Student
Eve	UEmployee
Fred	Student
Fred	PTEmployee
Greg	UMember

ARBAC97: URA97 sub-model

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

- can_assign:
  $UEmployee : \{Student, \overline{TA}\} \Longrightarrow +PTEmployee$
- can_revoke:
  $UEmployee : \{Student\} \Longrightarrow -Student$
- Static Mutually Exclusive Roles (SMER):
  $SMER(TA, PTEmployee)$

User	Role
Alice	PCMember
Bob	Faculty
Charlie	Faculty
David	TA
David	Student
Eve	UEmployee
Fred	PTEmployee
Greg	UMember

Administering Access Control Policies

- (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 fron 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!

URA97: security analysis problems

Let ψ 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 ψ (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 ψ?
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 ψ 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 ψ.

Symbolic Reachability Analysis of ARBAC Policies

1. A. Armando and S. Ranise. **Automated Symbolic Analysis of ARBAC Policies**. In Proc. of 6th Intl. Workshop on Security and Trust Management (STM'10), Athens, September 23-24, 2010.
2. F. Alberti, A. Armando, and S. Ranise. **Efficient Symbolic Automated Analysis of Administrative Attribute-based RBAC-Policies**. In Proc. of 6th ACM Symposium on Information, Computer and Communications Security (ASIACCS 2011), Hong Kong, March 22-24, 2011.
3. A. Armando and S. Ranise. **Automated Analysis of Infinite State Workflows with Access Control Policies**. In the Proceedings of the 7th International Workshop on Security and Trust Management (STM'11), Copenhagen (Denmark), July 27-28, 2011.
4. F. Alberti, A. Armando, and S. Ranise. **ASASP: Automated Symbolic Analysis of Administrative Policies**. In Proc. of 23rd Intl. Conf. on Automated Deduction (CADE-23), Wroclaw (Poland), Jul 31-Aug 5, 2011.
5. A. Armando and S. Ranise. **Automated Analysis of Infinite State Workflows with Access Control Policies.** In the Proceedings of the 7th International Workshop on Security and Trust Management, Copenhagen (Denmark), July 27-28, 2011.

Symbolic Representation of RBAC Policies

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

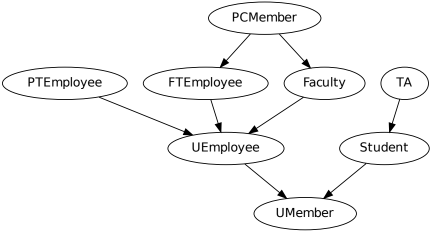
- Sorts: *User*, *Role*
- Predicate symbols: $ua : User \times Role$ (flexible)
  $\succeq : Role \times Role$ (rigid)
- Defining *ua*:

$$\forall u, r.(ua(u,r) \Leftrightarrow \left(\begin{array}{ll} (u = u_1 \wedge r = Role\,1) & \vee \\ (u = u_2 \wedge r = Role\,2) & \vee \\ (u = u_3 \wedge r = Role\,3) & \vee \\ \vdots & \end{array}\right))$$

- Defining $\succeq$:

$TA \succeq Student$,
$PTEmployee \succeq UEmployee$,
$UEmployee \succeq UMember$, ...,

$$\forall r.(r \succeq r)$$
$$\forall r_1, r_2, r_3.((r_1 \succeq r_2 \wedge r_2 \succeq r_3) \Rightarrow r_1 \succeq$$
$$\forall r_1, r_2.((r_1 \succeq r_2 \wedge r_2 \succeq r_1) \Rightarrow r_1 = r_2)$$



Symbolic Representation of RBAC Policies

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

$$\forall u. \neg(ua(u, TA) \wedge ua(u, PTEmployee))$$

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

$$\exists u, r.(ua(u, r) \wedge r \succeq Student)$$

- $UEmployee \ : \ \{Student, \overline{TA}\} \Longrightarrow +PTEmployee$

$$\exists u_a, r_a.(ua(u_a, r_a) \wedge r_a \succeq UEmployee) \wedge$$
$$\exists u. \left(\begin{array}{l} ua(u, Student) \wedge \forall r_2.(r_2 \succeq TA \Rightarrow \neg ua(u, r_2)) \wedge \\ \forall x, y.(ua'(x, y) \Leftrightarrow ((x = u \wedge y = PTEmployee) \vee ua(x, y))) \end{array} \right)$$

- $UEmployee \ : \ \{Student\} \Longrightarrow -Student$

$$\exists u_a, r_a.(ua(u_a, r_a) \wedge r_a \succeq UEmployee) \wedge$$
$$\exists u. \left(\begin{array}{l} \exists r_1.(ua(u, r_1) \wedge r_1 \succeq Student) \wedge \\ \forall x, y.(ua'(x, y) \Leftrightarrow (\neg(x = u \wedge y = Student) \wedge ua(x, y))) \end{array} \right)$$

Security analysis: bounded user-role reachability

Given an integer $k \geq 0$ and symbolic representation of

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

Check the satisfiability of

$$T_{\text{RBAC}} \wedge I(ua_0) \wedge \tau(ua_0, ua_1) \wedge \cdots \wedge \tau(ua_{k-1}, ua_k) \wedge G(ua_k)$$

Can be reduced to the satisfiability of **B**ernays-**S**hönfinkel-**R**amsey formulae
$\Longrightarrow$ Decidable*!*

Security analysis: unbounded user-role reachability (I)

Given symbolic representation of

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

Run a symbolic backward reachability procedure

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

Three requirements

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

Security analysis: unbounded user-role reachability (II)

- Effective computation of pre-images

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

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

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

- Termination of backward reachability

  by model-theoretic methods in combination with results on well-quasi-order

Security analysis: theoretical results, overview

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)

Security analysis: practical results, overview (I)

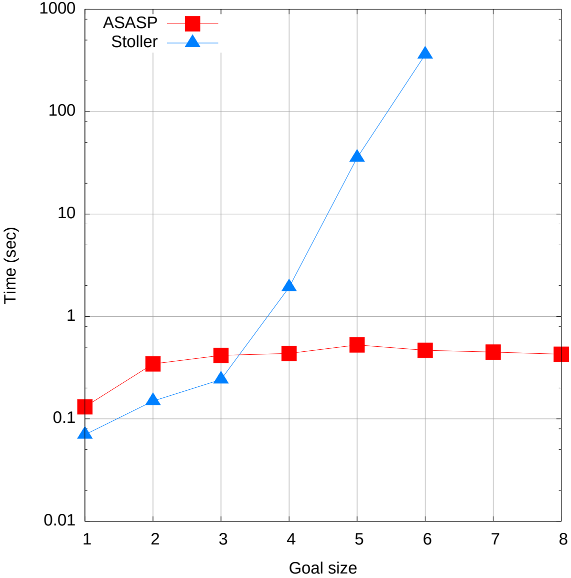
joint work with Francesco Alberti

Tool `ASASP`: Automated Symbolic Analysis of Administrative Policies

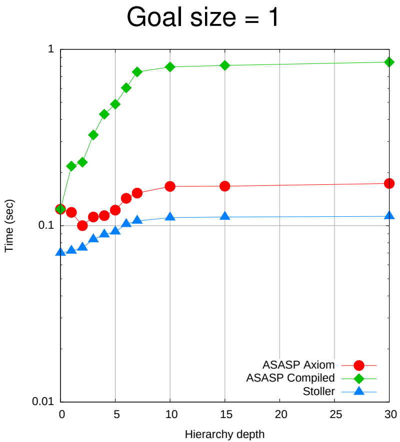
- architecture: client-server
- client: pre-image computation + generation of logical problems
- server: state-of-the-art SMT solvers and theorem provers on satisfiability problems.
  - Z3, incomplete over BSR but incremental
  - SPASS (refutation) complete but not incremental
  - hierarchical combination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

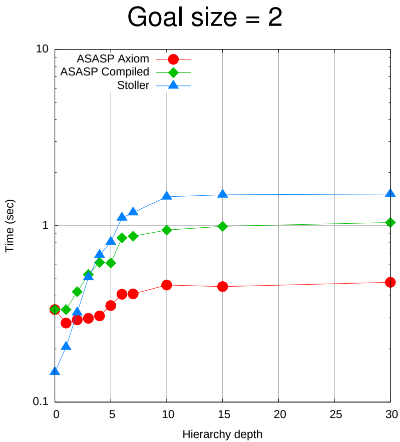
Security analysis: practical results, overview (II)

No role hierarchy

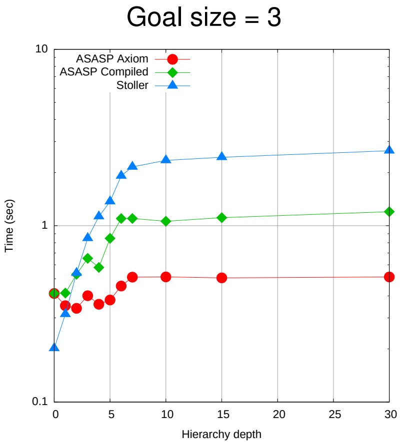


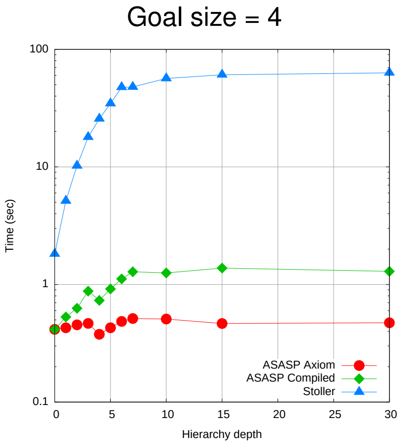
With role hierarchy





With role hierarchy





Symbolic Reachability Analysis of Personal Health Record Policies

1. A. Armando, R. Carbone, and S. Ranise. **Automated Analysis of Semantic-Aware Access Control Policies: a Logic-Based Approach.** In the Proceedings of the IEEE Workshop on Semantic Computing for Security and Privacy, San Francisco (USA), September 21, 2011.

Access Control in Online Distributed Environments

- 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.

ProjectHealth Design and TreC

Project HealthDesign
Rethinking the Power and Potential
of Personal Health Records

TreC

Regulating Access to PHRs

- 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

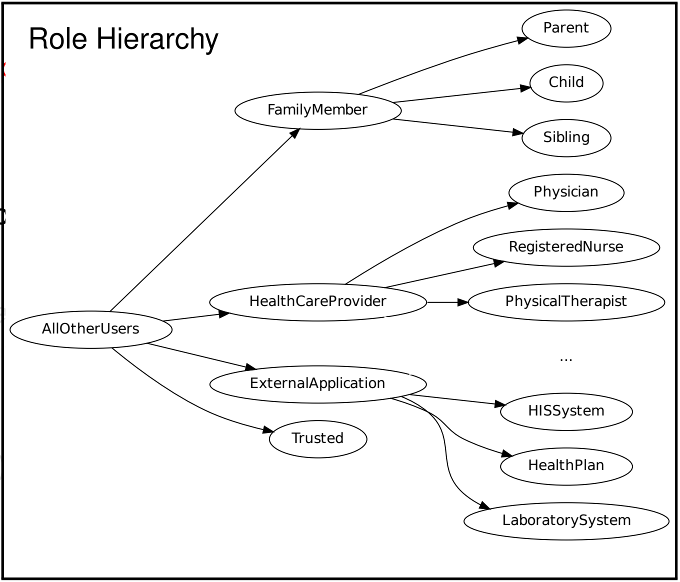
- U is the set of the user accounts and $u_o \in U$;
- R is a set of roles endowed with the hierarchy relation $\sqsupseteq_R$;
- $UA \subseteq (U \times R)$ is the user-role assignment relation;
- $P = (Act \times Res)$ is the set of permissions, where
  - Act is a set of actions endowed with the hierarchy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

- U is the set of
- R is a set of ro
- $UA \subseteq (U \times R)$
- $P = (Act \times Re$
  - Act is a se
  - Res is the
  - $\sqsupseteq_{Res}$.
- $PA = ((U \cup R)$



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 $\sqsupseteq_R$;
- $UA \subseteq (U \times R)$ 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_o is a tuple $\pi = (u_o, U, R, P, UA, PA)$, where

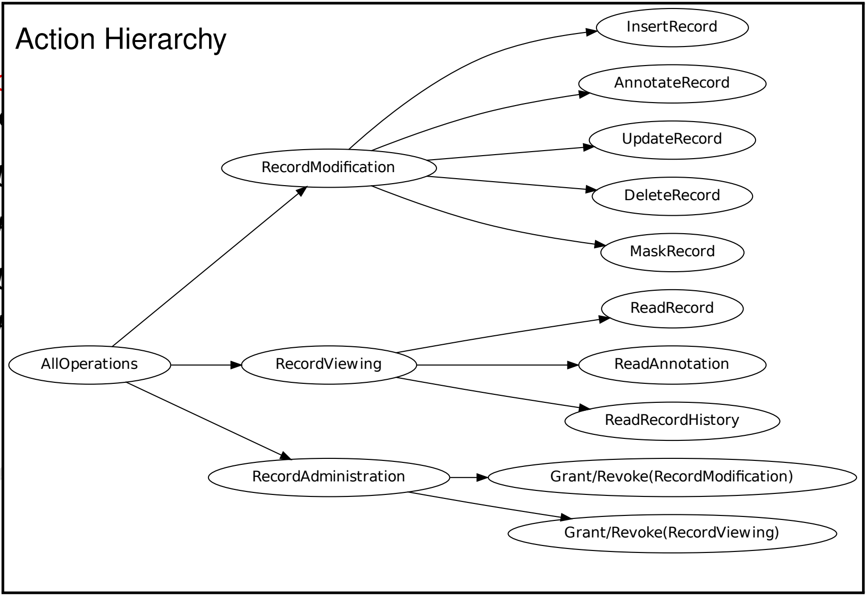
- U is the set of the user accounts and $u_o \in U$;
- R is a set of roles endowed with the hierarchy relation $\sqsupseteq_R$;
- $UA \subseteq (U \times R)$ is the user-role assignment relation;
- $P = (Act \times Res)$ is the set of permissions, where
  - Act is a set of actions endowed with the hierarchy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

-
-
-
-

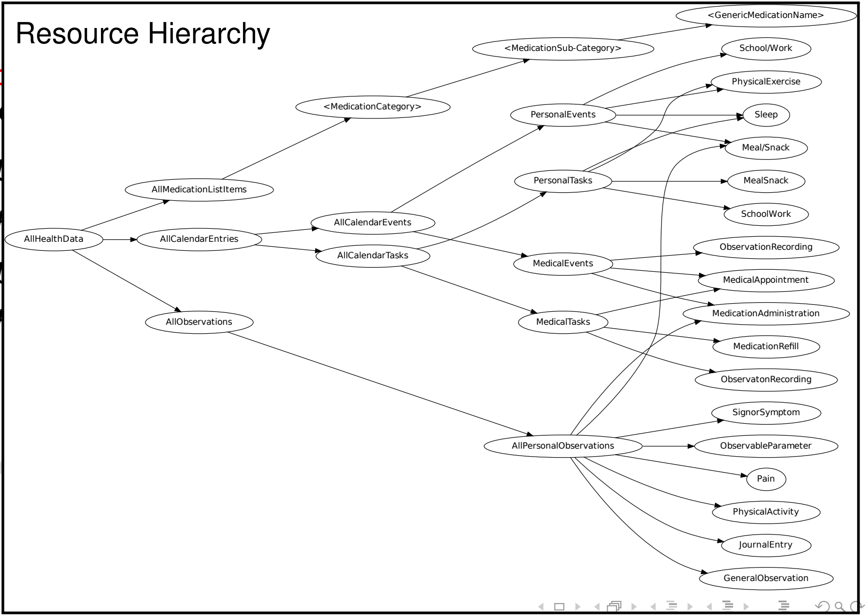

Action Hierarchy
InsertRecord
AnnotateRecord
UpdateRecord
RecordModification
DeleteRecord
MaskRecord
ReadRecord
AllOperations
RecordViewing
ReadAnnotation
ReadRecordHistory
RecordAdministration
Grant/Revoke(RecordModification)
Grant/Revoke(RecordViewing)

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 Model for Personal Health Records

An ac
wher



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.

Who can access to which resource?

A user u can execute act on res in π 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 $p \sqsupseteq_P (act, res)$.

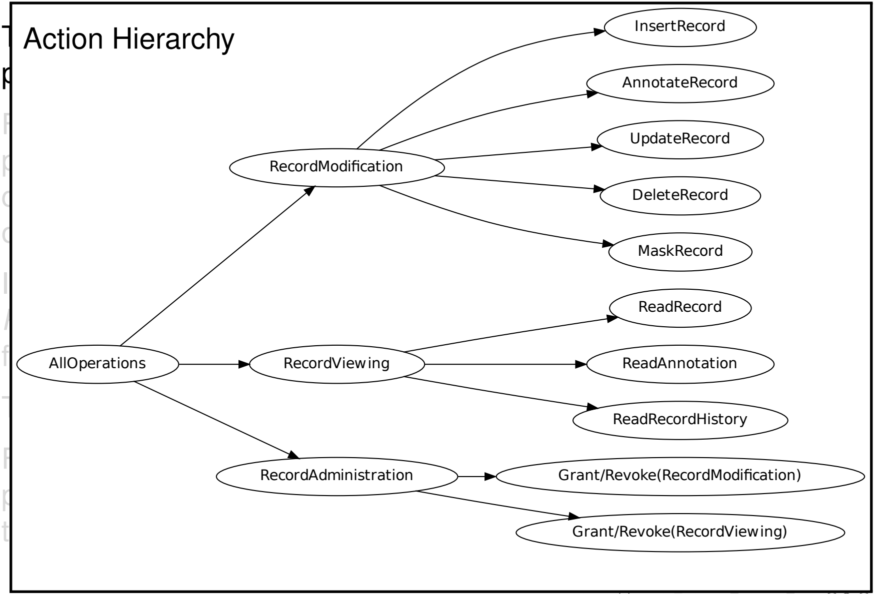
Example:

If Bob is the owner and $(\text{Alice}, p) \in PA$ with $p = (\text{RecordViewing}, \text{MedicalEvents})$, then Alice can view Bob's MedicalEvents (including his MedicalAppointments and MedicationAdministrations because of the resource hierarchy).

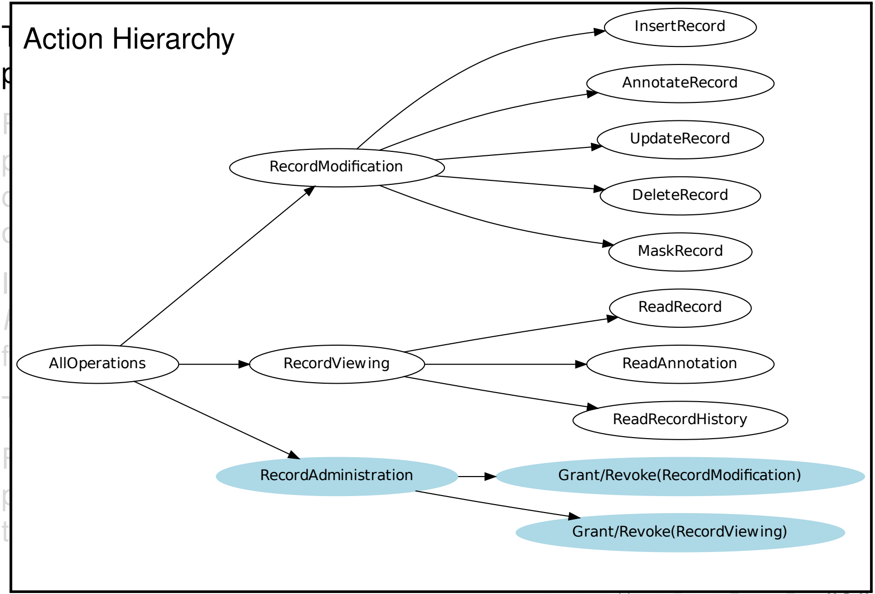
Administering the Policy: Delegation

- The policy also allows for the specification of administration privileges.
- For instance, if Bob is the owner and Alice is assigned the permission (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 $PA' = PA \cup \{(u, (\mathrm{RecordViewing}, \mathrm{MedicalEvents})\}$ 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



Administering the Policy: Delegation



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.

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 $(\mathrm{Grant}(\mathrm{RecordViewing}), \mathrm{MedicalEvents})$, then Alice can grant the privilege to viewing Bob's MedicalEvents to any other user.
- In other words, Alice can change PA into $PA' = PA \cup \{(u, (\mathrm{RecordViewing}, \mathrm{MedicalEvents})\}$ 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 (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 $PA' = PA \cup \{(u, (\mathrm{RecordViewing}, \mathrm{MedicalEvents})\}$ 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 $(\mathrm{Grant}(\mathrm{RecordViewing}), \mathrm{MedicalEvents})$, then Alice can grant the privilege to viewing Bob's MedicalEvents to any other user.
- In other words, Alice can change *PA* into $PA' = PA \cup \{(u, (\mathrm{RecordViewing}, \mathrm{MedicalEvents})\}$ 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: Conditional Delegation

- **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 Physician $\sqsupseteq_R r$ and
- $(u, r) \notin UA$ for all $r \in R$ such that 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

(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.
  $\Rightarrow$ Automated support needed!

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

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

  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.
  $\Rightarrow$ Automated support needed!

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

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

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.

$\Rightarrow$ Automated support needed!

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

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

  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.
  $\Rightarrow$ Automated support needed!

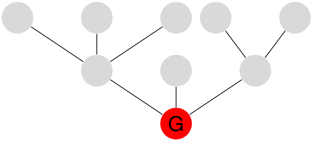
The PHR Reachability Problem

- If π' can be reached from π, then we write $\pi \rightarrow \pi'$.
- π_n is reachable from π_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, act, res) where $u \in U$, $act \in Act$ and $res \in Res$.
- **PHR reachability problem:** Given a query (u, act, res) and a policy π, determine whether there exists π' such that $\pi \rightarrow^* \pi'$ and u can execute *act* on *res* in π'.

Symbolic Reachability Analysis of PHR Policies

- Use first-order formulae to symbolically represent:
  - sets of policies, $R^i_k(pa)$
  - transitions between them, $\tau(pa, pa')$.
- Backward search: compute nodes as follows:
  - $R^0(pa) := G(pa)$ (goal)
  - $R^{i+1}(pa) := \exists pa'.(R^i(pa') \wedge \tau(pa, pa'))$ (pre-image) for $i \geq 0$

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



Symbolic representation of PHR policies

PHR policies can be specified by formulae of the Bernays-Schönfinkel-Ramsey (BSR) fragment, i.e. FOL formulae of the form

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

where φ is a quantifier-free formula containing only individual constants and predicate symbols (no function symbols allowed).

Symbolic representation of PHR policies

$$
\begin{aligned}
&\forall r. R(r) \Leftrightarrow (r = \text{RecordSubject} \vee r = \text{RecordCustodian} \\
&\qquad\qquad \vee\, r = \text{AllOtherUsers} \vee \cdots) \\
&\text{RecordSubject} \neq \text{RecordCustodian}, \\
&\text{RecordSubject} \neq \text{AllOtherUsers}, \\
&\text{RecordCustodian} \neq \text{AllOtherUsers}, \ldots,
\end{aligned}
$$

$$
\begin{aligned}
\forall r_1, r_2. r_1 \sqsupseteq_R r_2 \Rightarrow (R(r_1) \wedge R(r_2)) \\
\forall r. r \sqsupseteq_R r \\
\forall r_1, r_2. (r_1 \sqsupseteq_R r_2 \wedge r_2 \sqsupseteq_R r_1) \Rightarrow r_1 = r_2 \\
\forall r_1, r_2, r_3. (r_1 \sqsupseteq_R r_2 \wedge r_2 \sqsupseteq_R r_3) \Rightarrow r_1 \sqsupseteq_R r_3,
\end{aligned}
$$

$$\forall a, s, p. ARP(a, s, p) \Rightarrow (Act(a) \land Res(s) \land P(p))$$
$$\forall a, s, p, p'. (ARP(a, s, p) \land ARP(a, s, p')) \Rightarrow p = p'$$

$$\forall a, s, p, a', s', p'. (ARP(a, s, p) \land ARP(a', s', p')) \Rightarrow$$
$$(p \sqsupseteq_P p' \Leftrightarrow (a \sqsupseteq_{Act} a' \land s \sqsupseteq_{Res} s'))$$

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

$$\exists p. ARP(a, s, p) \Rightarrow$$
$$\left(\begin{array}{l} \exists p'. (PA(u, p') \land p' \sqsupseteq_P p) \;\lor \\ \exists r, r', p'. (R(r) \land R(r') \land UA(u, r) \land r \sqsupseteq_R r' \land PA(r', p') \land p' \sqsupseteq_P p) \end{array} \right)$$

The administrative action related to the pair $(\text{Alice}, p) \in \mathit{PA}$ with

$p = (\text{Grant(RecordViewing) to } \{+\text{Physician}, -\text{FamilyMember}\}, \text{MedicalEvents})$

is represented by the formula:

$$\exists p.(P(p) \wedge \mathit{ARP}(\text{RecordViewing}, \text{MedicalEvents}, p) \wedge \mathit{PA}(\text{Alice}, p) \Rightarrow \\ \exists u. \left(\begin{array}{l} \exists r_1.(\mathit{UA}(u, r_1) \wedge \text{Physician} \sqsupseteq_R r_1)\wedge \\ \forall r_2.(\text{FamilyMember} \sqsupseteq_R r_2 \Rightarrow \neg \mathit{UA}(u, r_2))\wedge \\ \forall x, y.(\mathit{PA}'(x, y) \Leftrightarrow (\mathit{PA}(x, y) \vee (x = u \wedge y = p))) \end{array} \right))$$

Symbolic Reachability Analysis of PHR Policies

- For the procedure to be effective
  - $\exists pa'.(R^i(pa') \wedge \tau(pa, pa'))$ must be turned into an equivalent BSR formula
  - the satisfiability of $R^i_k \wedge I$ 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

1. A. Armando and S. Ranise. **Automated Analysis of Infinite State Workflows with Access Control Policies.** In the Proceedings of the 7th International Workshop on Security and Trust Management, Copenhagen (Denmark), July 27-28, 2011.

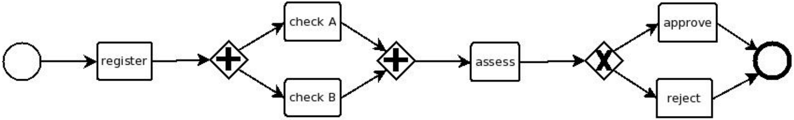
Context

- 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?

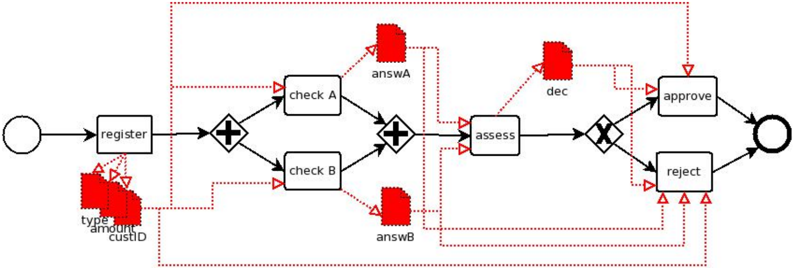
- register insurance claim
- check A of insurance policy
- check B of damage reported
- assess the results of checks A and B
- approve the payment of damage
- reject the payment of damage

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?

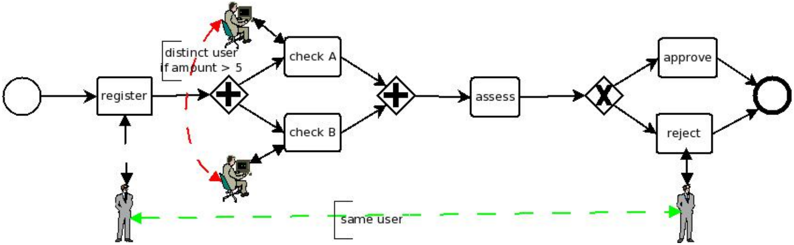
Role-based Access Control (RBAC)

User	Role
Anna	Customer Service
Adam	Customer Service
Benn	Specialist A
Beate	Specialist A
Beate	Specialist B
Carol	Specialist B
Chris	Specialist B

Role	Task
Customer Service	register
Customer Service	assess
Customer Service	approve
Customer Service	reject
Specialist A	check A
Specialist B	check B

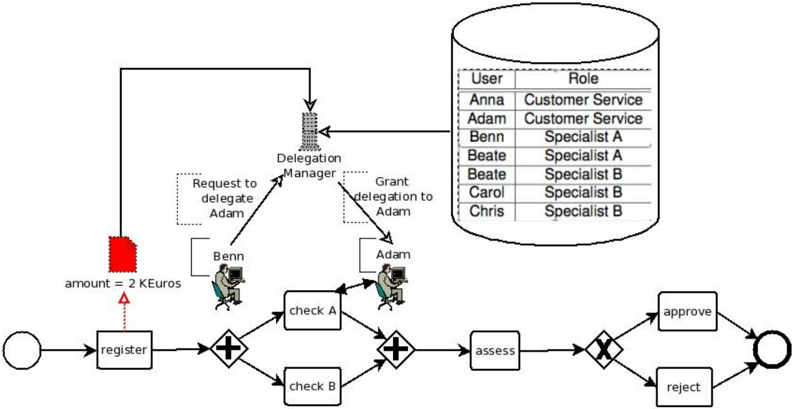
- Can Beate perform task check A? Yes!
- Can Benn perform task check B? No!
- Can Beate perform task check B? Yes!

- 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 KEuros, then user with role Specialist A can delegate the right to execute task check A to user with role Customer Service



- 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)

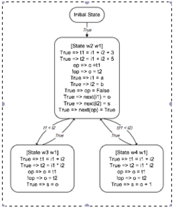
Safety and workflows

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

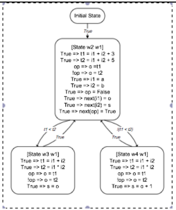
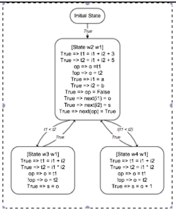
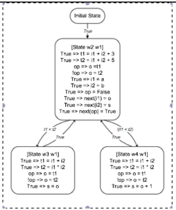


$$q \xrightarrow{\left[\begin{array}{l}\text{answA=ok}\wedge\\ \text{answB=ok}\end{array}\right] \Rightarrow \left[\begin{array}{l}\text{decision:=grant;}\\ \text{ex:=ex} \cup \{\text{assess}\}\end{array}\right]} q'$$

- Access control = RBAC + Delegation + Authorization constraints

$$\begin{array}{rcl}\text{Policy} & : & \langle \text{Users, Roles, Tasks, User-Role, Role-Task, Role hierarchy} \rangle \\ \text{Rule} & : & \text{amount} < 5 \;:\; \text{Specialist A} \xrightarrow{\text{check A}} \text{Customer Service} \\ & & \ldots \end{array}$$

Our framework: the safety problem

<table>
<tr><th>RBAC session</th><th>RBAC session</th><th>· · ·</th><th>RBAC session</th></tr>
<tr><td>
</td><td>
</td><td>· · ·</td><td>
</td></tr>
<tr><td>id_1</td><td>id_2</td><td>· · · · · · · · · · · · ·</td><td>id_n</td></tr>
<tr><td colspan="4">RBAC policy with delegation rules and authorization constraints</td></tr>
</table>

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

$\Longrightarrow$ 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: $\mathsf{var} := \mathsf{var}'$ or $\mathsf{var} := \mathsf{constant}$
  - 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

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

Transition system:	$\langle V, In(V), \{\tau_h(V, V')\}_h\rangle$
Error condition:	$E(V)$

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

2. Use "standard" backward reachability and prove mechanization and termination

Backward reachability: overview

- Let $\mathcal{T} := \bigvee_h \tau_h$. Iteratively compute

$$R_0(V) := E(V) \text{ and } R_{j+1}(V) := R_j(V) \vee \underbrace{\exists V'.(R_j(V') \wedge \mathcal{T}(V, V'))}_{\text{pre-image}} \text{ for } j \geq 0$$

$$\widehat{R}_j(V) := R_j(V) \wedge \underbrace{\mathcal{AC}(V)}_{\text{Authorization constraints}}$$

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

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

and safety

$$\exists V.(\widehat{R}_j(V) \wedge In(V)) \text{ is unsatisfiable?}$$

Mechanization of backward reachability

Theorem

If

- *formulae in* $\{In\} \cup \mathcal{AC}$ *are universal BSR*
- *each* τ_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:

1. Easy: simple logical manipulations
2. 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?