CS590U

Access Control: Theory and Practice

Lecture 20 (March 24)
Security Analysis in Trust Management
What is Security Analysis?

- Inspired by safety analysis, which was initially formalized by Harrison et al.
- An access control policy verification technique
- Studies properties of access control systems whose state may change
- Precisely evaluates which principals/users are trusted for what properties.
Given a start state $P$, a query $Q$, and a rule $R$ that determines how states can change (defines reachability among states);

Ask

- Is $Q$ possible? (existential) whether $\exists$ reachable $P'$ s.t. $Q$ is true in $P'$
- Is $Q$ necessary? (universal) whether $\forall$ reachable $P'$, $Q$ is true in $P'$
How to Use Security Analysis

- Guarantee safety and availability properties of an AC system:
  - Properties one wants to guarantee are encoded in a set of queries
  - $R$ identifies trusted principals
    - Assumes that parts under these principals’ control do not change
  - Trusted principals perform security analysis before making changes
Security Analysis in RBAC

N. Li & M. Tripunitara
SACMAT 2004
Security Analysis in RBAC

- RBAC state is \langle UA, PA, RH \rangle
- State change rules: admin model, e.g. ARBAC97 [Sandhu et al., TISSEC’99]
- Queries:
  - Have the form “userSet_1 \neq userSet_2 ?”
    - e.g. “is r_1 \cap r_2 ? \{u1, u2\}?”
  - Called semi-static if either userSet_1 or userSet_2 can be evaluated independent of the state
Admin Models: AATU and AAR

- AATU = \( \langle \text{can\_assign}, T \rangle \)
  - can\_assign \( \subseteq \mathbb{R} \times \mathbb{C} \times 2^\mathbb{R} \)
    - \( \langle \text{manager, employee ? engineer, \{projLead\} \rangle} \)
  - T: a set of trusted users
- AAR = \( \langle \text{can\_assign, can\_revoke} \rangle \)
  - can\_revoke \( \subseteq \mathbb{R} \times 2^\mathbb{R} \)
    - \( \langle \text{manager, \{projLead\} \rangle} \)
Results - AATU

- For semi-static queries, security analysis is efficient (polynomial time)

- For other types of queries, security analysis is decidable, but intractable (coNP-hard)
Results - AAR

- For semi-static queries, security analysis is efficient.

- For other queries, security analysis is decidable, but intractable (coNP-complete)
How We Showed This

- We present a reduction from our security analysis instances to instances in RT

  Mapping:
  - Input: RBAC \(\langle\text{state, query, state-change rule}\rangle\)
  - Output: RT \(\langle\text{state, query, state-change rule}\rangle\)
Beyond Proof-of-Compliance: Security Analysis in Trust Management

N. Li, J.C. Mitchell & W.H. Winsborough. To Appear in JACM.
Motivation for Security Analysis in TM?

- Delegation is used extensively in TM
- Control may be delegated to partially trusted principals
- What if one delegates to the wrong principal?
- How to ensure that desirable security properties are maintained with delegation?
The TM Language

$RT[\leftrightarrow, \cap] = RT_0$

- Basic concepts in $RT[\leftrightarrow, \cap]$:
  - Principals: $K, K_1, K_2$
  - Role names: $r, r_1, r_2$
  - Roles: $K.r$ (K’s r role)
    - each role has a member set
Statements in $RT[\leftrightarrow, \cap]$

- **Type-1:** $K_r \leftarrow K_1$
  - $\text{mem}[K.r] \supseteq \{K_1\}$
  - $K_{HR}\text{.manager} \leftarrow K_{Alice}$

- **Type-2:** $K_r \leftarrow K_1.r_1$
  - $\text{mem}[K.r] \supseteq \text{mem}[K_1.r_1]$
  - $K_{SSO}\text{.admin} \leftarrow K_{HR}\text{.manager}$
Statements in $RT[\leftrightarrow, \cap]$

- **Type-3:** $K.r \leftarrow K.r_1.r_2$
  - Let $\text{mem}[K.r_1]$ be $\{K_1, K_2, ..., K_n\}$   $\text{mem}[K.r] \supseteq \text{mem}[K.r_1] \cup \text{mem}[K.r_2]$
  - $\text{mem}[K_1.r_2] \cup \text{mem}[K_2.r_2]$
  - $\cup \text{mem}[K_n.r_2]$
  - $K_{SSO}.\text{delegAccess} \leftarrow K_{SSO}.\text{admin.access}$

- **Type-4:** $K.r \leftarrow K_1.r_1 \cap K_2.r_2$
  - $\text{mem}[K.r] \supseteq \text{mem}[K_1.r_2] \cap \text{mem}[K_2.r_2]$
  - $K_{SSO}.\text{access} \leftarrow K_{SSO}.\text{delegAccess} \cap K_{HR}.\text{employee}$
The Query Q

- Form-1: $\text{mem}[K.r] \supseteq \{K_1, \ldots, K_n\}$ ?
- Form-2: $\{K_1, \ldots, K_n\} \supseteq \text{mem}[K.r]$ ?
- Form-3: $\text{mem}[K_1.r_1] \supseteq \text{mem}[K.r]$ ?
The Semantic Relation

- A statement $\Rightarrow$ a Datalog rule
  - $K.r \leftarrow K_2 \quad \Rightarrow \quad m(K, r, K_2)$
  - $K.r \leftarrow K_1.r_1 \quad \Rightarrow \quad m(K, r, z) :- m(K_1, r_1, z)$
  - ...

- A state $P \Rightarrow$ a Datalog program $SP[P]$
  - $\text{mem}[K.r] \equiv \{ K' \mid m(K,r,K') \text{ is in the minimal Herbrand model of } SP[P] \}$
Example Queries & Answers

1. $K_{SSO}.access \leftarrow K_{SSO}.admin$
2. $K_{SSO}.admin \leftarrow K_{HR}.manager$
3. $K_{HR}.employee \leftarrow K_{HR}.manager$
4. $K_{HR}.manager \leftarrow K_{Alice}$
5. $K_{HR}.employee \leftarrow K_{David}$

$\text{mem}[K_{SSO}.access] \supseteq \{K_{David}\}$? No

$\{K_{Alice}, K_{David}\} \supseteq \text{mem}[K_{SSO}.employee]$? Yes

$\text{mem}[K_{HR}.employee] \supseteq \text{mem}[K_{SSO}.access]$? Yes
The State-Change Rule $R$

- $R = (G, S)$
  - $G$ is a set of growth-restricted roles
    - if $A.r \in G$, then cannot add “$A.r \leftarrow \ldots$”
  - $S$ is a set of shrink-restricted roles
    - if $A.r \in S$, then cannot remove “$A.r \leftarrow \ldots$”

- Motivation:
  - Definitions of roles that are not under one’s control may change
Sample Analysis Queries

- Simple safety (existential form-1):
  - Is $\text{mem}[K.r] \supseteq \{K_1\}$ possible?

- Simple availability (universal form-1):
  - Is $\text{mem}[K.r] \supseteq \{K_1\}$ necessary?

- Bounded safety (universal form-2):
  - Is $\{K_1, \ldots, K_n\} \supseteq \text{mem}[K.r]$ necessary?

- Containment (universal form-3):
  - Is $\text{mem}[K_1.r_1] \supseteq \text{mem}[K.r]$ necessary?
Example

1. $K_{SSO}.access \leftarrow K_{SSO}.admin$
2. $K_{SSO}.access \leftarrow K_{SSO}.delegAccess \cap K_{HR}.employee$
3. $K_{SSO}.admin \leftarrow K_{HR}.manager$
4. $K_{SSO}.delegAccess \leftarrow K_{SSO}.admin.access$
5. $K_{HR}.employee \leftarrow K_{HR}.manager$
6. $K_{HR}.employee \leftarrow K_{HR}.engineer$
7. $K_{HR}.manager \leftarrow K_{Alice}$
8. $Alice.access \leftarrow K_{Bob}$

Legend:
- fixed
- can grow, can shrink
A Simple Availability Query

1. \( K_{SSO}.access \leftarrow K_{SSO}.admin \)
2. \( K_{SSO}.access \leftarrow K_{SSO}.delegAccess \cap K_{HR}.employee \)
3. \( K_{SSO}.admin \leftarrow K_{HR}.manager \)
4. \( K_{SSO}.delegAccess \leftarrow K_{SSO}.admin.access \)
5. \( K_{HR}.employee \leftarrow K_{HR}.manager \)
6. \( K_{HR}.employee \leftarrow K_{HR}.engineer \)
7. \( K_{HR}.manager \leftarrow K_{Alice} \)
8. \( Alice.access \leftarrow K_{Bob} \)

Query: Is \( mem[K_{SSO}.access] \supseteq \{K_{Alice}\} \) necessary?

Answer: Yes. (Available)

Why: Statements 1, 3, and 7 cannot be removed
A Simple Safety Query

1. $K_{SSO} \cdot access \leftarrow K_{SSO} \cdot admin$
2. $K_{SSO} \cdot access \leftarrow K_{SSO} \cdot delegAccess \cap K_{HR} \cdot employee$
3. $K_{SSO} \cdot admin \leftarrow K_{HR} \cdot manager$
4. $K_{SSO} \cdot delegAccess \leftarrow K_{SSO} \cdot admin \cdot access$
5. $K_{HR} \cdot employee \leftarrow K_{HR} \cdot manager$
6. $K_{HR} \cdot manager \leftarrow K_{Alice}$
7. $K_{HR} \cdot employee \leftarrow K_{HR} \cdot engineer$
8. $K_{Alice} \cdot access \leftarrow K_{Bob}$

Query: Is $\text{mem}[K_{SSO} \cdot access] \supseteq \{K_{Eve}\}$ possible?
Answer: Yes. (Unsafe)
Why: Both $K_{HR} \cdot engineer$ and $K_{Alice} \cdot access$ may grow.
A Containment Analysis Query about Safety

<table>
<thead>
<tr>
<th>Statement</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.</td>
<td>$K_{SSO}.access \leftarrow K_{SSO}.admin$</td>
</tr>
<tr>
<td>2.</td>
<td>$K_{SSO}.access \leftarrow K_{SSO}.delegAccess \cap K_{HR}.employee$</td>
</tr>
<tr>
<td>3.</td>
<td>$K_{SSO}.admin \leftarrow K_{HR}.manager$</td>
</tr>
<tr>
<td>4.</td>
<td>$K_{SSO}.delegAccess \leftarrow K_{SSO}.admin.access$</td>
</tr>
<tr>
<td>5.</td>
<td>$K_{HR}.employee \leftarrow K_{HR}.manager$</td>
</tr>
<tr>
<td>6.</td>
<td>$K_{HR}.employee \leftarrow K_{HR}.engineer$</td>
</tr>
<tr>
<td>7.</td>
<td>$K_{HR}.manager \leftarrow K_{Alice}$</td>
</tr>
<tr>
<td>8.</td>
<td>$K_{Alice}.access \leftarrow K_{Bob}$</td>
</tr>
</tbody>
</table>

**Query:** Is $\text{mem}[K_{HR}.employee] \supseteq \text{mem}[K_{SSO}.access]$ necessary?

**Answer:** Yes. (Safe)

**Why:** $K_{SSO}.access$ and $K_{SSO}.admin$ cannot grow and Statement 5 cannot be removed.
An Containment Analysis Query about Availability

1. $K_{SSO}\text{.access} \leftarrow K_{SSO}\text{.admin}$
2. $K_{SSO}\text{.access} \leftarrow K_{SSO}\text{.delegAccess} \cap K_{HR}\text{.employee}$
3. $K_{SSO}\text{.admin} \leftarrow K_{HR}\text{.manager}$
4. $K_{SSO}\text{.delegAccess} \leftarrow K_{SSO}\text{.admin}\text{.access}$
5. $K_{HR}\text{.employee} \leftarrow K_{HR}\text{.manager}$
6. $K_{HR}\text{.employee} \leftarrow K_{HR}\text{.engineer}$
7. $K_{HR}\text{.manager} \leftarrow K_{Alice}$
8. $Alice\text{.access} \leftarrow K_{Bob}$

Query: Is $\text{mem}[K_{SSO}\text{.access}] \supseteq \text{mem}[K_{HR}\text{.manager}]$ necessary?
Answer: Yes. (Available)
Why: Statements 1 and 3 cannot be removed
Form-1 and Form-2 Queries

- **PTIME**
  - Form-1 queries are monotonic in P
  - Form-2 queries are anti-monotonic in P
  - Use the minimal reachable state to answer universal form-1 and existential form-2
  - The maximal reachable state answers existential form-1 and universal form-2
    - the state is simulated by a logic program

Reminder:  
Form-1 query: \( \text{mem}[K,r] \supseteq \{K_1,\ldots,K_n\} \)  
Form-2 query: \( \{K_1,\ldots,K_n\} \supseteq \text{mem}[K,r] \)
Universal Form-3 $\equiv$ Containment Analysis

- With just type 1 and 2 statements
  - containment analysis is in PTIME
  - using logic programs with stratified negation
- With type 1, 2, and 4 statements
  - containment analysis is coNP-complete
  - equivalent to determining validity of propositional-logic formulas

Reminder:

Queries: Form-3: $\text{mem}[K_1.r_1] \supseteq \text{mem}[K.r]$

Statements: Type-1: $K.r \leftarrow K_1$
            Type-2: $K.r \leftarrow K_1.r_1$
            Type-4: $K.r \leftarrow K_1.r_1 \cap K_2.r_2$
Universal Form-3
(Containment Analysis)

- RT[↔] (Type 1, 2, and 3 statements)
  - containment analysis is PSPACE-complete
    - RT[↔] ↔ string-rewriting systems
    - equivalent to determining containment of languages accepted by NFA’s
  - remains PSPACE-complete without shrinking
  - coNP-complete without growing

Reminder:

Type-1: K.r ← K₁
Type-2: K.r ← K₁.r₁
Type-3: K.r ← K₁.r₁.r₂
Universal Form-3
(Containment Analysis)

- RT[\(\leftrightarrow, \cap\)] (all four types of statements)
  - in coNEXP
    - although infinitely many new principals and statements may be added, if the containment does not hold, there exists a counter example whose size is at most exponential
  - PSPACE-hard
  - exact complexity still open!
  - coNP-complete without growing
Summary of Complexities for Containment Analysis

- Type-1 and 2: PTIME
- Type-1, 2, and 3: PSPACE-complete
- Type-1, 2, and 4: coNP-complete
- Type-1, 2, 3, and 4: PSPACE-hard, coNEXP
The analysis problem: Given P, Q, and R, is Q possible, is Q necessary?

Certain classes of security analysis in RBAC reduce to that in RT[\(\leftrightarrow, \cap\)]

Security analysis problems for RT[\(\leftrightarrow, \cap\)]
- decidable
- efficiently decidable for most queries
- for containment analysis, complexity depends on delegation features of the policy language
Mapping the HRU model to the Abstract Analysis Problem

- **P:** an access matrix
- **R:** the protection system state can change by executing commands
  - e.g., \( c(x,y,z) \{ \text{if} \ ‘own’ \in \text{cell}(x,z) \land ‘controls’ \in \text{cell}(x,y) \text{ then add ‘read’ to cell}(y,z) \} \)
- **Q:** is \( r \in \text{cell}(s,o) \) possible?
  - simple safety queries only
- **Main result in the HRU model**
  - simple safety is undecidable
Relating RT[↔, ∩] with HRU

- Role memberships determined by a RT[↔, ∩] state is an access matrix
  - principals correspond to both subjects and objects
  - $K_1 \in \text{mem}[K.r]$
    subject $K_1$ has right $r$ over object $K$ $\iff r \in \text{cell}(K_1, K)$
- Adding a type-1 statement $K.r \leftarrow K_1$
  - adding $r$ into $\text{cell}(K_1, K)$
Relating RT[⇌,∩] with HRU

- Adding a type-2 statement $K.r \leftarrow K_1.r_1$
  - for every $K'$ such that $K' \in \text{mem}[K_1.r_1]$ add $r$ into $\text{cell}(K',K)$
  - need to run an HRU command for every principal
  - this propagation needs to happen every time the matrix is changed
<table>
<thead>
<tr>
<th>Access Matrix:</th>
<th>K₁</th>
<th>K₂</th>
<th>K₃</th>
<th>K₄</th>
</tr>
</thead>
<tbody>
<tr>
<td>K₁</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>K₂</td>
<td>r'</td>
<td>r</td>
<td></td>
<td></td>
</tr>
<tr>
<td>K₃</td>
<td>r'</td>
<td>r</td>
<td></td>
<td></td>
</tr>
<tr>
<td>K₄</td>
<td>r'</td>
<td>r</td>
<td>r'</td>
<td></td>
</tr>
</tbody>
</table>

2. ∀ K', execute \(rr'(K₁,K₂,K')\)

4. ∀ K',K'' execute \(r'rr(K₂,K',K'')\)

\[rr'(x,y,z) \{ \text{if } r \in \text{cell}(z,y) \text{ then add } r' \text{ to cell}(z,x) \}\]

\[r'rr(x,y,z) \{ \text{if } r' \in \text{cell}(z,y) \wedge r \in \text{cell}(y,z) \text{ then add } r' \text{ to cell}(z,x) \}\]
Can HRU simulate RT? (Probably not!)

- It seems that HRU cannot simulate RT
  - Adding one statement corresponds to executing multiple HRU commands
  - Seems unable to simulate the effect of propagation
  - Unclear how to simulate removal of statements
Why Our Problem is Decidable?

- Note that we consider queries that are more complicated than simple safety
  - e.g., containment analysis

- Some parameters in our analysis problem are simpler
  - no need to consider arbitrary commands
    - only four types of statements
  - restriction rules are static
Next Lecture

- Automated Trust Negotiation