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.

The Abstract Security Analysis Problem

- 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 $ reachable P’ s.t. Q is true in P’
  - Is Q necessary? (universal)
    - whether " 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 ⟨UA, PA, RH⟩
- State change rules: admin model, e.g. ARBAC97 [Sandhu et al., TISSEC'99]
- Queries:
  - Have the form "userSet_1 ? 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 = ⟨*can_assign*, T⟩
  - *can_assign* $\subseteq$ R x C x 2^R
    - ⟨manager, employee ? engineer, {projLead}⟩
  - T: a set of trusted users
- AAR = ⟨*can_assign*, *can_revoke*⟩
  - *can_revoke* $\subseteq$ R x 2^R
    - ⟨manager, {projLead}⟩

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 ⟨state, query, state-change rule⟩
  - Output: RT ⟨state, query, state-change rule⟩

Beyond Proof-of-Compliance: Security Analysis in Trust Management

N. Li, J.C. Mitchell & W.H. Winsborough.

To Appear in JACM.

Conference version in IEEE S&P 2003.

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[⇦, ∩]=RT_0

- Basic concepts in RT[⇦, ∩]:
  - 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[⇦, ∩]

- Type-1: $K.r \leftarrow K_1$
  - mem[K.r] Ê $\{K_1\}$
  - K_{HR}.manager ← K_{Alice}
- Type-2: $K.r \leftarrow K_1.r_1$
  - mem[K.r] Ê mem[$K_1.r_1$]
  - K_{SSO}.admin ← K_{HR}.manager

Statements in RT[⇦, ∩]

- Type-3: $K.r \leftarrow K.r_1.r_2$
  - Let mem[$K.r_1$] be {K_1, K_2, …, K_n}   mem[K.r] Ê mem[$K_1.r_2$] È mem[$K_2.r_2$]   È ¼ È mem[$K_n.r_2$]
  - K_{SSO}.delegAccess ← K_{SSO}.admin.access

- Type-4: $K.r \leftarrow K_1.r_1$ Ç $K_2.r_2$
  - mem[K.r] Ê mem[$K_1.r_2$] Ç mem[$K_2.r_2$]
  - K_{SSO}.access←K_{SSo}.delegAccessÇK_{HR}.employee

The Query Q

- Form-1: mem[K.r] Ê $\{K_1,\ldots,K_n\}$?
- Form-2: $\{K_1,\ldots,K_n\}$ Ê mem[K.r] ?
- Form-3: mem[$K_1.r_1$] Ê mem[K.r] ?

The Semantic Relation

- A statement ⇒ a Datalog rule
  - $K.r \leftarrow K_2$ ⇒ $m(K, r, K_2)$
  - $K.r \leftarrow K_1.r_1$ ⇒ $m(K, r, z)$:- $m(K_1, r_1, z)$
  - ...
- A state P ⇒ a Datalog program SP[P]
  - mem[K.r] º { K’ | m(K,r,K’) 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. $K_{HR}.employee \leftarrow K_{David}$

$mem[K_{SSO}.access] \supseteq \{K_{David}\}$?	No
$\{K_{Alice}, K_{David}\} \supseteq mem[K_{SSO}.employee]$?	Yes
$mem[K_{HR}.employee] \supseteq mem[K_{SSO}.access]$?	Yes

The State-Change Rule R

- R=(G,S)
  - G is a set of growth-restricted roles
    - if A.r ∈ G, then cannot add “A.r ← …”
  - S is a set of shrink-restricted roles
    - if A.r ∈ S, then cannot remove “A.r ← …”
- Motivation:
  - Definitions of roles that are not under one’s control may change

Sample Analysis Queries

- Simple safety (existential form-1):
  - Is mem[K.r] $\supseteq \{K_1\}$ possible?
- Simple availability (universal form-1):
  - Is mem[K.r] $\supseteq \{K_1\}$ necessary?
- Bounded safety (universal form-2):
  - Is $\{K_1,\ldots,K_n\} \supseteq$ mem[K.r] necessary?
- Containment (universal form-3):
  - Is mem[$K_1.r_1$] $\supseteq$ mem[K.r] necessary?

Example

1. K_{SSO}.access ← K_{SSO}.admin
2. K_{SSO}.access ← K_{SSO}.delegAccess Ç K_{HR}.employee
3. K_{SSO}.admin ← K_{HR}.manager
4. K_{SSO}.delegAccess ← K_{SSO}.admin.access
5. K_{HR}.employee ← K_{HR}.manager
6. K_{HR}.employee ← K_{HR}.engineer
7. K_{HR}.manager ← K_{Alice}
8. Alice.access ← K_{Bob}

Legend: fixed

can grow, can shrink

A Simple Availability Query

1. K_{SSO}.access ← K_{SSO}.admin
2. K_{SSO}.access ← K_{SSO}.delegAccess Ç K_{HR}.employee
3. K_{SSO}.admin ← K_{HR}.manager
4. K_{SSO}.delegAccess ← K_{SSO}.admin.access
5. K_{HR}.employee ← K_{HR}.manager
6. K_{HR}.employee ← K_{HR}.engineer
7. K_{HR}.manager ← K_{Alice}
8. Alice.access ← K_{Bob}

Query: Is mem[K_{SSO} .access] Ê {K_{Alice}} necessary?
Answer: Yes. (Available)
Why: Statments 1, 3, and 7 cannot be removed

A Simple Safety Query

1. K_{SSO}.access ← K_{SSO}.admin
2. K_{SSO}.access ← K_{SSO}.delegAccess Ç K_{HR}.employee
3. K_{SSO}.admin ← K_{HR}.manager
4. K_{SSO}.delegAccess ← K_{SSO}.admin.access
5. K_{HR}.employee ← K_{HR}.manager
6. K_{HR}.manager ← K_{Alice}
7. K_{HR}.employee ← K_{HR}.engineer
8. K_{Alice}.access ← K_{Bob}

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

A Containment Analysis Query about Safety

1. K_{SSO}.access ← K_{SSO}.admin
2. K_{SSO}.access ← K_{SSO}.delegAccess Ç K_{HR}.employee
3. K_{SSO}.admin ← K_{HR}.manager
4. K_{SSO}.delegAccess ← K_{SSO}.admin.access
5. K_{HR}.employee ← K_{HR}.manager
6. K_{HR}.employee ← K_{HR}.engineer
7. K_{HR}.manager ← K_{Alice}
8. K_{Alice}.access ← K_{Bob}

Query: Is mem[K_{HR}.employee] ⊒ 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}.access ← K_{SSO}.admin
2. K_{SSO}.access ← K_{SSO}.delegAccess Ç K_{HR}.employee
3. K_{SSO}.admin ← K_{HR}.manager
4. K_{SSO}.delegAccess ← K_{SSO}.admin.access
5. K_{HR}.employee ← K_{HR}.manager
6. K_{HR}.employee ← K_{HR}.engineer
7. K_{HR}.manager ← K_{Alice}
8. Alice.access ← K_{Bob}

Query: Is mem[K_{SSO}.access] ⊇ mem[K_{HR}.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: mem[K.r] Ê $\{K_1,\ldots,K_n\}$
Form-2 query: $\{K_1,\ldots,K_n\}$ Ê mem[K.r]

Universal Form-3 ≡ 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: $K.r \leftarrow K_1$

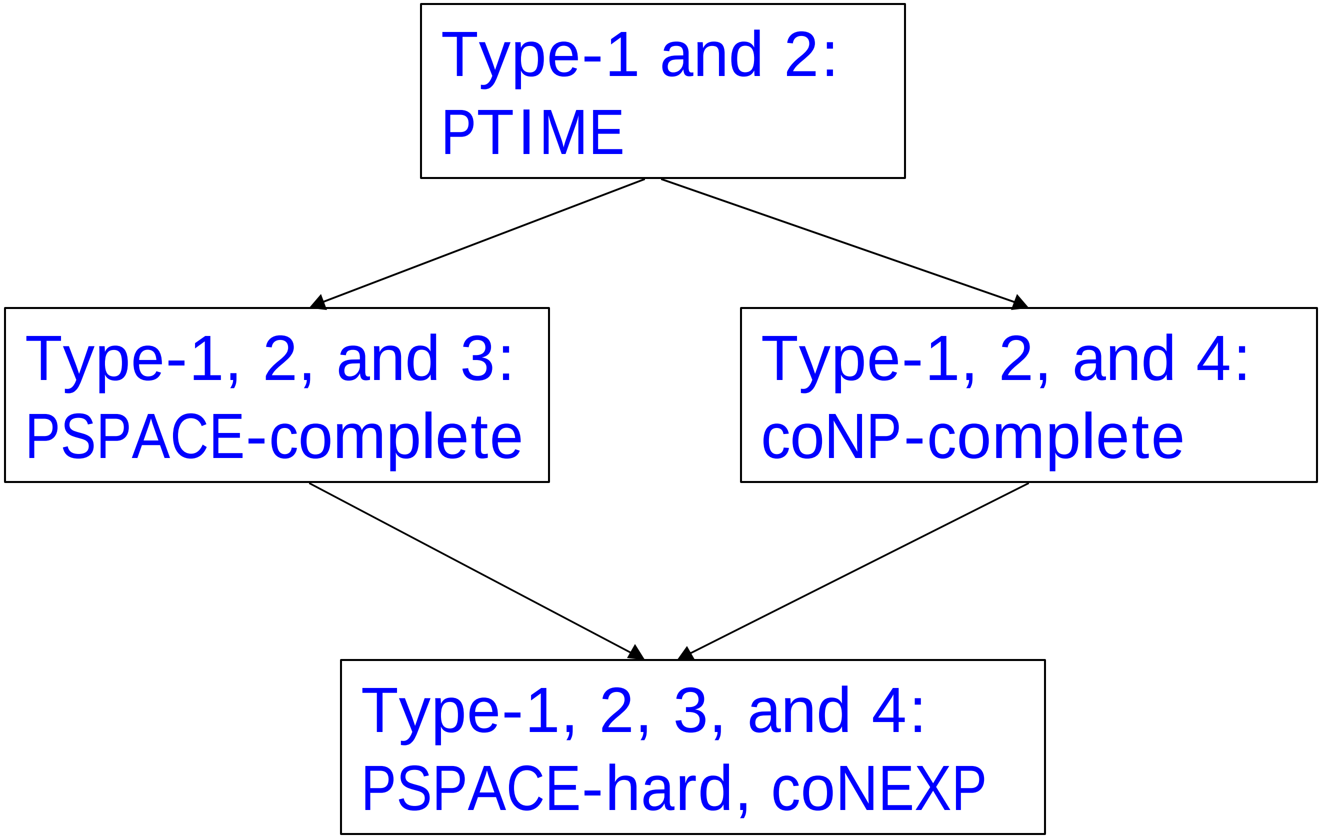
Type-2: $K.r \leftarrow K_1.r_1$

Type-3: $K.r \leftarrow K_1.r_1.r_2$

Universal Form-3 (Containment Analysis)

- RT[⇦,∩] (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

Summary

- 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[⇦,∩]
- Security analysis problems for RT[⇦,∩]
  - 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) { if 'own'$\in$cell(x,z) $\wedge$ 'controls' $\in$cell(x,y) then add 'read' to cell(y,z)}
- Q: is r$\in$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 Î mem[K.r] ⇔ subject K_1 has right r over object K ⇔ r Î cell(K_1,K)
- Adding a type-1 statement K.r ← K_1
  - adding r into 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’ Î mem[$K_1.r_1$] add r into cell(K’,K)
  - need to run an HRU command for every principal
  - this propagation needs to happen every time the matrix is changed

Access Matrix:

	K_1	K_2	K_3	K_4
K_1				
K_2	r’	r		
K_3	r’	r		
K_4	r’	r	r’	

1. Add K_2.r ← K_2
2. Add K_1.r’ ← K_2.r
3. Add K_2.r ← K_3
4. Add K_2.r ← K_2.r.r’
5. Add K_3.r’ ← K_4

Triggers:

2. " K’, execute rr’(K_1,K_2,K’)
4. " K’,K” execute r’rr(K_2,K’,K”)

rr’(x,y,z) { if r Î cell(z,y) then add r’ to cell(z,x) }

r’rr(x,y,z) { if r’ Î cell(z,y) Ù r Î cell(y,x) then add r’ 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