Logic and Logic Programming in Distributed Access Control (Part Two)

Ninghui Li
Department of Computer Science
and CERIAS
Purdue University

Security Analysis in Trust Management

- Publications:
  - Li, Mitchell & Winsborough: “Beyond Proof-of-Compliance: Security Analysis in Trust Management”, JACM 2005. Conference version in SSP 2003.

The Abstract Security Analysis Problem

- Given an initial state P,
  - a query Q,
  - and a rule R that restricts how states can change (defines reachability among states);
- Ask
  - Is Q possible?   (existential)
    - whether $ reachable P’ s.t. P’ ▶ Q
  - Is Q necessary?   (universal)
    - whether " reachable P’ , P’ ▶ Q

Statements in RT_0 = RT[⇦, ∩]

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

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,…,K_n} ?
- Form-2: {K_1,…,K_n} Ê mem[K.r] ?
- Form-3: mem[K_1.r_1] Ê mem[K.r] ?

The Semantic Relation ▶

- A statement $\Rightarrow$ a Datalog rule
  - $K.r \leftarrow K_2 \quad \Rightarrow \quad m(K, r, K_2)$
  - $K.r \leftarrow K_1.r_1 \quad \Rightarrow \quad m(K, r, z) \text{ :- } m(K_1, r_1, z)$
  - …
- A state P $\Rightarrow$ 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 ← K_{SSO}.admin
2. K_{SSO}.admin ← K_{HR}.manager
3. K_{HR}.employee ← K_{HR}.manager
4. K_{HR}.manager ← K_{Alice}
5. K_{HR}.employee ← K_{David}

mem[K_{SSO}.access] Ê {K_{David}}? No

{K_{Alice}, K_{David}} Ê mem[K_{SSO}.employee]? Yes

mem[K_{HR}.employee] Ê mem[K_{SSO}.access]? Yes

The Restriction Rule R

- R=(G,S)
  - G is a set of growth-restricted roles
    - if K.r $\in$ G, then cannot add "K.r $\leftarrow$ …"
  - S is a set of shrink-restricted roles
    - if K.r $\in$ S, then cannot remove "K.r $\leftarrow$ …"
- 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?

Security Analysis: Usage Cases

- Guarantee safety and availability properties of an access control system:
  - Properties one wants to guarantee are encoded in a set of queries & desirable answers
  - R represents how much control one has
    - parts not under one's control may change in R
    - parts under one's control are considered fixed in R
  - Before making changes, one can use analysis to guarantee properties are not violated

An Example

1. K_{SSO}.access $\leftarrow$ K_{SSO}.admin
2. K_{SSO}.access $\leftarrow$ K_{SSO}.delegAccess Ç 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. K_{Alice}.access $\leftarrow$ 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. K_{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] ⊇ {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. K_{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

Answering Form-1 and Form-2 Queries: Intuitions (1)

- RT[⇦, ∩] is monotonic
  - more statements derive more role memberships
- Form-1 queries are monotonic
  - mem[K.r] Ê {K1,…,Kn}
  - universal form-1 queries can be answered by considering a lower-bound (minimum) state
  - existential form-1 queries can be answered by considering an upper-bound (maximal) state

Answering Form-1 and Form-2 Queries: Intuitions (2)

- Form-2 queries are anti-monotonic
  - {K1,…,Kn} Ê mem[K.r]
  - universal form-2 queries can be answered by considering the upper-bound state
  - existential form-1 queries can be answered by considering the lower-bound state
- Given P and R, the lower-bound state uniquely exists, we denote it $P|_R$
  - it can be reached by removing all removable statements

The Lower-Bound Program LB(P,R)

- For each $K.r \leftarrow K_1$ in $P|_R$, add
  lb(K, r, K_1)
- For each $K.r \leftarrow K_1.r_1$ in $P|_R$, add
  lb(K, r, ?Z) :- lb(K_1, r_1, ?Z)
- For each $K.r \leftarrow K.r_1.r_2$ in $P|_R$, add
  lb(K, r, ?Z) :- lb(K, r_1, ?Y), lb(?Y, r_2, ?Z)
- For each $K.r \leftarrow K_1.r_1$ Ç $K_2.r_2$ in $P|_R$, add
  lb(K, r, ?Z) :- lb(K_1, r_1, ?Z), lb(K_2, r_2, ?Z)

Using the Lower-Bound Program

- To answer whether a form-1 query
  mem[K.r] Ê $\{K_1,\ldots,K_n\}$ is necessary,
  - check whether
    $LB(P,R) \models lb(K,r,K_1) \wedge \ldots \wedge lb(K,r,K_n)$
- To answer whether a form-2 query
  $\{K_1,\ldots,K_n\}$ Ê mem[K.r] is possible
  - check whether
    $\{K_1,\ldots,K_n\}$ Ê $\{ Z \mid LB(P,R) \models lb(K,r,Z) \}$

The Upper-Bound Program UB(P,R)

- Add $ub(T, ?r, ?Z)$
- For each K.r that can grow, add $ub(K, r, ?Z)$
- For each $K.r \leftarrow K_1$ in P, add $ub(K, r, K_1)$
- For each $K.r \leftarrow K_1.r_1$ in P, add
  $ub(K, r, ?Z) \text{ :- } ub(K_1, r_1, ?Z)$
- For each $K.r \leftarrow K.r_1.r_2$ in P, add
  $ub(K, r, ?Z) \text{ :- } ub(K, r_1, ?Y),\ ub(?Y, r_2, ?Z)$
- For each $K.r \leftarrow K_1.r_1$ Ç $K_2.r_2$ in P, add
  $ub(K, r, ?Z) \text{ :- } ub(K_1, r_1, ?Z), ub(K_2, r_2, ?Z)$

Using the Upper-Bound Program

- A form-1 query mem[K.r] Ê {K_1,…,K_n} is possible iff. any of the following is true,
  - K.r is not growth restricted
  - up(K,r,T) is true
  - UB(P,R) |= ub(K,r,K_1) $\wedge \ldots \wedge$ ub(K,r,K_n)
- A form-2 query {K_1,…,K_n} Ê mem[K.r] is necessary iff.
  - {K_1,…,K_n} Ê { Z | UB(P,R) |= ub(K,r,Z) }

What about Form-3 Queries?

- Form-3: mem[$K_1.r_1$] $\supseteq$ mem[K.r]
- Neither monotonic nor anti-monotonic
  - cannot use the minimal state or the maximal state
- Difficulty: adding new members to K.r may affect $K_1.r_1$
- We only consider analysis asking whether mem[$K_1.r_1$] $\supseteq$ mem[K.r] is necessary
  - we call this containment analysis

Complexity Results for Containment Analysis

- RT[]: just type 1 and 2 statements
  - containment analysis is in PTIME
- RT[∩]: type 1, 2, and 4 statements
  - containment analysis is coNP-complete
- RT[⇦]: type 1, 2, and 3 statements
  - containment analysis is PSPACE-complete
  - remains PSPACE-complete without shrinking
  - coNP-complete without growing
- RT[⇦,∩]: decidable in coNEXP

Containment Analysis in RT[]

- Two cases that X.u contains K.r
  1. the containment is forced by statements in P and cannot be removed
  2. the containment is caused by nonexistence of statements
     - e.g., when no statement defines K.r and K.r cannot grow, K.r is always empty, and thus is contained in every role
     - direct translation of this intuition into a positive logic program does not work
       - e.g., P={"$K.r \leftarrow K_1.r_1$", "$K_1.r_1 \leftarrow K.r$", "$K.r \leftarrow K_2$", "$X.u \leftarrow K_2$"}, both K.r and $K_1.r_1$ are fixed, does X.u contain K.r?

The Containment Program for RT[]: BCP(P,R)

- Starts from LB(P,R)
- Add fc(?X,?u,?X,?u)
- For each $K.r \leftarrow K_1.r_1$ in $P|_R$, add
  fc(K,r,?Z,?w) :- fc(K_1,r_1,?Z,?w)
- For each K.r that can grow, add
  nc(?X,?u,K,r) :- ~ fc(?X,?u,K,r)
- For each $K.r \leftarrow K_1$ in P s.t. K.r can't grow, add
  nc(?X,?u,K,r) :- ~ fc(?X,?u,K,r), ~ lb(?X,?u,K_1)
- For each $K.r \leftarrow K_1.r_1$ in P s.t. K.r can't grow, add
  nc(?X,?u,K,r) :- ~ fc(?X,?u,K,r), nc(?X,?u,K_1,r_1)

Solving Containment Analysis in RT[] Using Negation

- BCP(P,R) is stratified
  - we use the perfect model semantics

- Theorem: BCP(P,R) |= nc(X,u, K,r) is true iff. X.u does *not* contain K.r

Containment Analysis in RT[∩] is coNP-complete

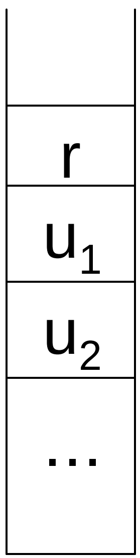
- It is in coNP, because a counter example can be found by considering just one new principal
- That it is coNP-hard is shown by reducing the monotone 3-SAT problem to it
  - intersection is conjunction,
  - a role may be defined by multiple statements (implicit disjunction)
  - containment equivalent to determining validity of formulas like $\varphi 1 \Leftarrow \varphi 2$
    - where $\varphi 1$ are $\varphi 2$ positive propositional formulas

Containment Analysis in RT[⇦]

- First consider the case that no shrinking is allowed in R
- View statements as rewriting rules
  - $K.r \leftarrow K_1$ K r to K_1
  - $K.r \leftarrow K_1.r_1$ K r to $K_1\ r_1$
  - $K.r \leftarrow K.r_1.r_2$ K r to $K\ r_1\ r_2$
- A string has the form $K\ r_1\ r_2\ r_3\ r_4$
- Lemma 0: SP[P] proves $m(K,r, K_1)$ iff. the string K r rewrites into K_1 using P

RT[⇦] and Pushdown Systems

Stack:

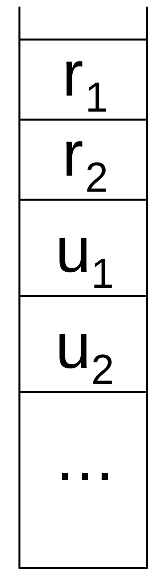


State: K

Apply the rewriting rule:

K r to K r_1 r_2

Stack:



State: K

A string corresponds to a configuration

“rewrites into” equivalent to “reaches”

Characteristic Set of a Role

- Given P and R (shrinking forbidden), define:
  - $strs_P$[K.r] = sets of strings K r rewrites to
  - χ_R = the set consisting of
    - all principals in P
    - all strings K_1 r_1 r_2 r_3 r_4 where K_1 appears in P and K_1 r_1 is g-unrestricted
  - $\chi_{P,R}$[K.r] = $strs_P$[K.r] Ç χ_R
    - each string K_1 r_1 r_2 r_3 r_4 in $\chi_{P,R}$[K.r] is a distinct way of adding a member to K.r
- Lemma 1: Given P, R, X.u, K.r, mem[X.u] Ê mem[K.r] is necessary iff. $\chi_{P,R}$[X.u] Ê $\chi_{P,R}$[K.r]

Lemma 2:

- Lemma 2: Given P, R (shrinking forbidden), and K.r, $\chi_{P,R}$[K.r] is recognized by an NFA that has size poly in |P|+|R|
- Proof: $\chi_{P,R}$[K.r] = $strs_P$[K.r] Ç χ_R
  - $strs_P$[K.r] is recognized by a poly-size NFA
    - Bouajjani, Esparza & Maler: “Reachability Analysis of Pushdown Automata: Application to Model-Checking”, CONCUR’97
  - χ_R is recognized by a poly-size NFA
  - $\chi_{P,R}$[K.r] is recognized by a poly-size NFA

Containment Analysis in RT[⇦] is in PSPACE

- Theorem: Given P, R (shrinking forbidden), X.u, K.r, determining whether mem[X.u] Ê mem[K.r] is necessary is in PSPACE
  - follows from Lemma 1 and 2 and the fact that determining containment of languages accepted by NFA’s is in PSPACE

Containment Analysis in RT[⇦] is PSPACE-hard

- Theorem: Given P, R (shrinking forbidden), X.u, K.r, determining whether mem[X.u] Ê mem[K.r] is necessary is PSPACE-hard
  - Reducing determining containment of languages over the alphabet {0,1} that are defined by right-linear grammars to the problem.

Proof of PSPACE-hardness

- From grammar to P:
  - N_1 ::= N_2 1 $\qquad$ $K.N_1 = K.N_2.r_1$
  - N_2 ::= 0 $\qquad$ $K.N_2 = K_1.r_0$
- The restriction rule R:
  - all $K.N_i$'s, $K.r_i$'s, and $K_1.N_i$'s are g-restricted
  - other roles, i.e., $K_1.r_0$ and $K_1.r_1$, are growth unrestricted
- Language[N_1] maps to $\chi_{P,R}[K.N_1]$
  - N_1 generates 1010 iff. $K_1.r_1.r_0.r_1.r_0$ $\hat{I}$ $\chi_{P,R}$ $[K.N_1]$

Theorem (shrinking allowed)

- Given P, R (shrinking allowed), X.u, K.r, determining whether mem[X.u] Ê mem[K.r] is necessary is in PSPACE
  - For every subset of P that can be obtained by legally removing statements in P, run the algorithm that does not allow shrinking

Containment Analysis in RT[⇦ ∩]

- Theorem: Given P (in RT[⇦ ∩]), R, X.u, K.r, determining whether mem[X.u] Ê mem[K.r] is necessary is in coNEXP
  - although infinitely many new principals and statements may be added, if a counter example exists, then a counter example of size exponential in P exists
  - if two new principals have the same memberships in all roles appearing in P, then the two principals can be collapsed into one

Summary of Complexities for Containment Analysis