CS590U

Access Control: Theory and Practice

Lecture 18 (March 10)

SDSI Semantics & The RT Family of Role-based Trust-management Languages

Understanding SPKI/SDSI Using First-Order Logic

Ninghui Li and John C. Mitchell

International Journal of Information Security. Preliminary version in CSFW 2003.

What is a Semantics?

- Elements of a semantics
  - syntax for statements
  - syntax for queries
  - an entailment relation that determines whether a query Q is true given a set P of statements

Why a Formal Semantics?

- What can we gain by a formal semantics
  - understand what queries can be answered
  - defines the entailment relation in a way that is precise, easy to understand, and easy to compute
- How can one say a semantics is good
  - subjective metrics:
    - simple, natural, close to original intention
  - defines answers to a broad class of queries
  - can use existing work to provide efficient deduction procedures for answering those queries

Summary of SDSI Semantics

- Rewriting based
  - can answer queries such as can one string rewrites into another one
- Set based
  - can answer queries such as which principals are in the valuation of a string
- Logic programming based
- First-Order Logic based

A Logic-Programming-based Semantics

- Translate each 4-tuple into a LP clause
  - Using a ternary predicate m
    - m(K, A, K') is true if K' ∈ *V* (K A)
  - (K A ⇨ K') to m(K, A, K')
  - (K A ⇨ K_1 A_1) to m(K, A, ?x) :- m(K_1, A_1, ?x)
  - (K A ⇨ K_1 A_1 A_2)
    to m(K,A,?x) :- m(K_1,A_1,?y_1), m(?y_1,A_2,?x)
  - (K A ⇨ K_1 A_1 A_2 A_3)
    to m(K,A,?x) :- m(K_1,A_1,?y_1), m(?y_1,A_2,?y_2), m(?y_2,A_3,?x)
- The minimal Herbrand model determines the semantics

Example

From

$(k_C \text{ mit} \Rightarrow k_M)$

$(k_M \text{ faculty} \Rightarrow k_{EECS} \text{ faculty})$

$(k_C \text{ access} \Rightarrow k_C \text{ mit faculty secretary})$

To

$m(k_C, \text{mit}, k_M).$

$m(k_M, \text{faculty}, Z) \text{ :- } m(k_{EECS}, \text{faculty}, Z).$

$m(k_C, \text{access}, Z) \text{ :- } m(k_C, \text{mit}, Y_1),$
$\quad m(Y_1, \text{faculty}, Y_2), m(Y_2, \text{secretary}, Z).$

Set semantics is equivalent to LP semantics

- The least Herbrand model of SP[P] is equivalent to the least valuation, i.e.,
  - K’ $\in$ V_P (K A)   iff. m(K,A,K’) is in the least Herbrand model of SP[P]
- Same limitation as set-based semantics
  - does not define answers to containment between arbitrary name strings

An Alternative Way of Defining the LP-based Semantics (1)

- Define a macro **contains**
  - **contains**[ω][K'] means that K' Î *V* (ω)
    - **contains**[K][K'] ≡ (K= K')
    - **contains**[K A][K'] ≡ m(K, A, K')
    - **contains**[K A_1 A_2 … A_n][K'] ≡
      $*y* (m(K, A_1, *y*) Ù **contains**[*y* A_2 … A_n][K'])
      where n>1

An Alternative Way of Defining the LP-based Semantics (2)

- Translates a 4-tuple (K A ⇨ ω) into a FOL sentence
  - $\forall z$ (**contains**[K A][z] ⇐ **contains**[ω][z])
- This sentence is also a Datalog clause
- A set P of 4-tuples defines a Datalog program, denoted by SP[P]
  - The minimal Herbrand model of SP[P] defines the semantics

An Example of Translation

From (K_C access ⇨ K_C mit faculty secretary)

to $\forall z$ (**contains**[K_C access][z] $\Leftarrow$
  contains[K_C mit faculty secretary][z])

to $\forall z$ (**m(**K_C, access, z) $\Leftarrow$
  $\$y_1$ (m(K_C, mit, y_1) Ù **contains**[y_1 faculty secretary][z])

to $\forall z\ \forall y_1$ (**m(**K_C, access, z) $\Leftarrow$
  m(K_C, mit, y_1) Ù
  $\exists y_2$ (m(y_1, faculty, y_2) Ù **contains**[y_2 secretary] [z])

to $\forall z\ \forall y_1\ \forall y_2$ (**m(**K_C, access, z) $\Leftarrow$
  m(K_C, mit, y_1) Ù
  m(y_1, faculty, y_2) Ù
  m(y_2, secretary, z]))

A First-Order Logic (FOL) Semantics

- A set P of 4-tuples defines a FOL theory, denoted by Th[P]
- A query is a FOL formula
  - "ω_1 rewrites into ω_2" is translated into $\forall z\ (\mathbf{contains}[\omega_1][z] \Leftarrow \mathbf{contains}[\omega_2][z])$
  - Other FOL formulas can also be used as queries
- Logical implication determines semantics

FOL Semantics is Extension of LP Semantics

- LP semantics is FOL semantics with queries limited to LP queries
  - m(K,A,K') is in the least Herbrand model of SP[P] iff. Th[P] |= m(K,A,K')

Equivalence of Rewriting Semantics and FOL Semantics

- Theorem: for string rewriting queries, the string rewriting semantics is equivalent to the FOL semantics
  - Given a set P of 4-tuples, it is possible to rewrite ω_1 into ω_2 using the 4-tuples in P if and only if

$$\text{Th}[\text{P}] \;\; {}^{2} \;\; {}''z \; (\mathbf{contains}[\omega_1][z] \Leftarrow \mathbf{contains}[\omega_2][z])$$

Advantages of FOL semantics: Computation efficiency

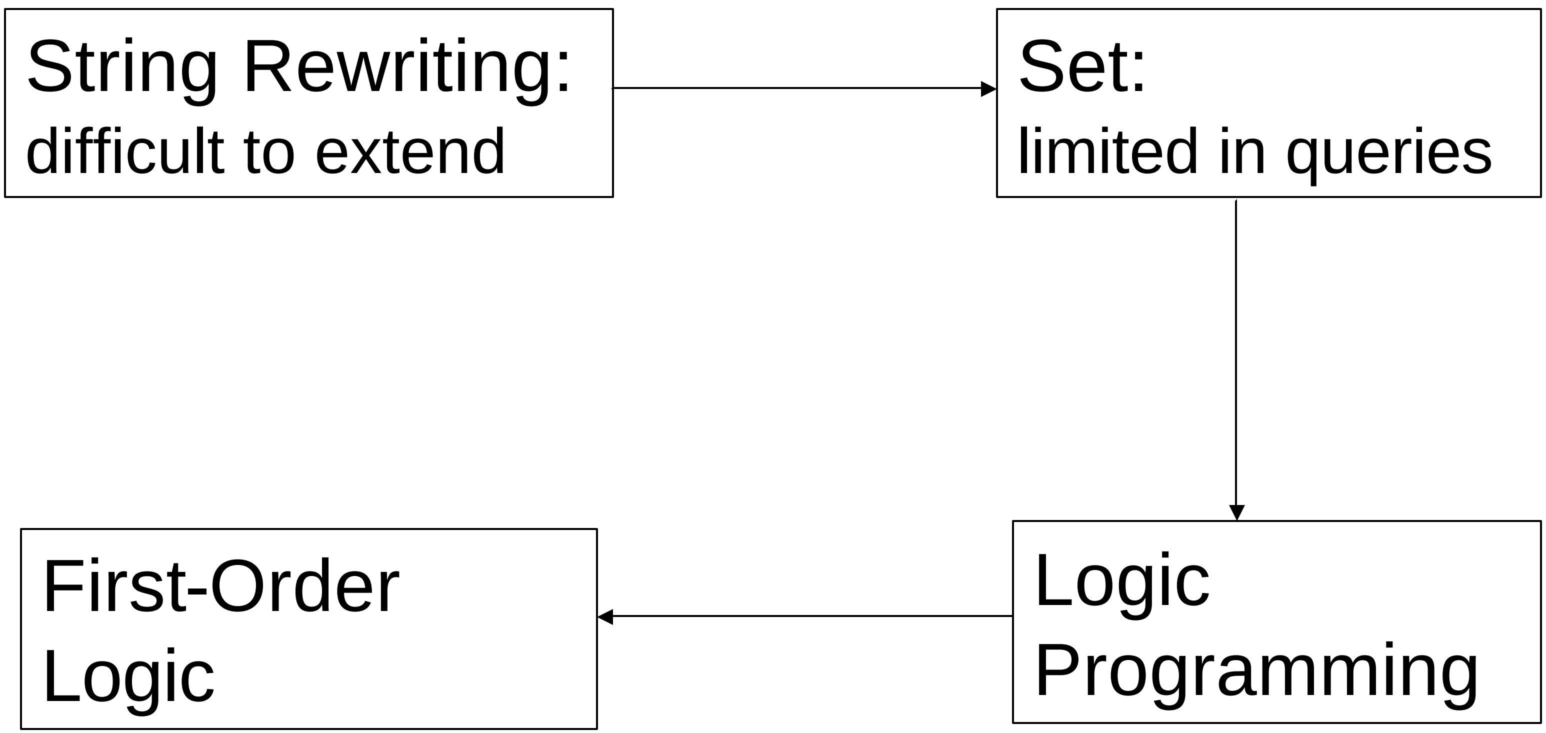
- A large class of queries can be answered efficiently using logic programs
  - including rewriting queries
  - e.g., whether ω rewrites into K B_1 B_2 under P can be answered by determining whether $SP[P \cup (K'\ A' \Rightarrow \omega) \cup (K\ B_1 \Rightarrow K'_1) \cup (K'_1\ B_2 \Rightarrow K'_2)]$ ² $m(K', A', K'_2)$
    - where K', K'_1, and K'_2 are new principals
    - this proof procedure is sound and complete
      - this result also follows from results in proof theory regarding Harrop Hereditary formulas

Advantages of FOL semantics: Extensibility

- Additional kinds of queries can be formulated and answered, e.g.,
  - $\forall z\ (m(K_1, A_1, z) \Leftarrow m(K_1, A_2, z))$
    $\Leftarrow \exists z\ (m(K_2, A_1, z) \wedge m(K_2, A_2, z))$

- Additional forms of statements can be easily handled, e.g.,
  - (K A ⇨ K_1 A_1 $\cap$ K_2 A_2) maps to
    $\forall z\ (m(K,A,z) \Leftarrow m(K_1,A_1,z)$ Ù $m(K_2,A_2,z))$

Summary: 4 Semantics for SDSI



Advantages of FOL Semantics: Summary

- Simple
  - captures the set-based intuition
  - defined using standard FOL
- Extensible
  - additional policy language features can be handled easily
  - allow more meaningful queries
- Computation efficiency

Design of A Role-based Trust-management Framework

Ninghui Li, John C. Mitchell & William H. Winsborough

IEEE S&P 2002

Features of the RT family of TM languages

- Expressive delegation constructs
- Permissions for structured resources
- A tractable logical semantics based on Constraint Datalog
- Strongly-typed credentials and vocabulary agreement
- Efficient deduction with large number of distributed policy statements
- Security analysis

Expressive Features (part one)

I. Simple attribute assignment

StateU.stuID ¬ Alice

II. Delegation of attribute authority

StateU.stuID ¬ COE.stuID

III. Attribute inferencing

EPub.access ¬ EPub.student

IV. Attribute-based delegation of authority

EPub.student ¬ EPub.university.stuID

Expressive Features (part two)

V. Conjunction

EPub.access ¬ EPub.student Ç ACM.member

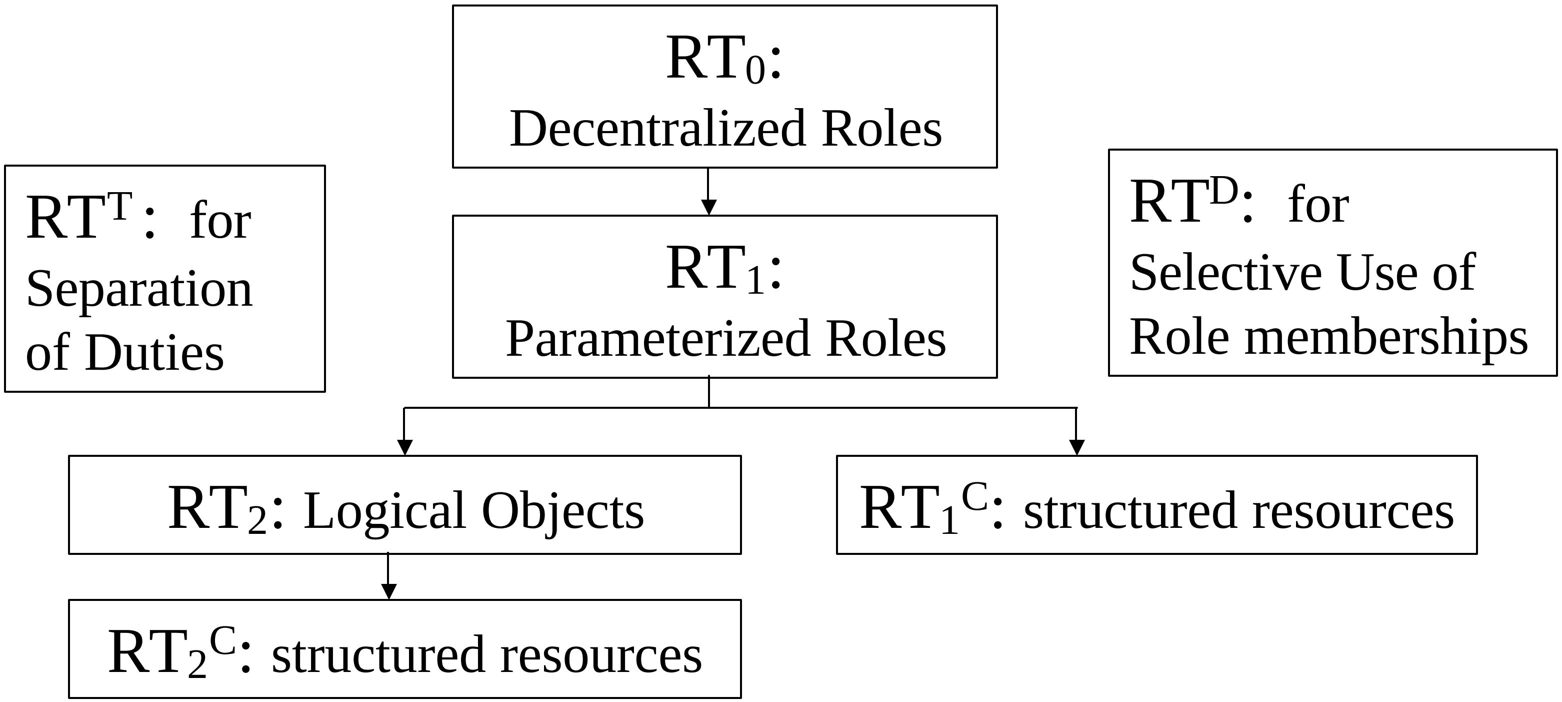
VI. Attributes with fields

- **StateU.stuID (name=.., program=.., ...) ¬ Alice**
- **EPub.access ¬ StateU.stuID(program="graduate")**

VII. Permissions for structured resources

- e.g., allow connection to any host in a domain and at any port in a range

The Languages in the RT Framework



RT^T and RT^D can be used (either together or separately) with any of the five base languages: RT_0, RT_1, RT_2, RT_1^C, and RT_2^C

$RT_1 = RT_0$ + Parameterized Roles

- Motivations: to represent
  - attributes that have fields, e.g., digital ids, diplomas
  - relationships between principals, e.g., physicianOf, advisorOf
  - role templates, e.g., project leaders

- Approach:
  - a role term *R* has a role name and a list of fields

RT_1 (Examples)

- Example 1: Alpha allows manager of an employee to evaluate the employee:

  Alpha.evaluatorOf(employee=*y*) ¬
  Alpha.managerOf(employee=*y*)

- Example 2: EPub allows CS students to access certain resources:

  EPub.access(action=‘read’, resource=‘file1’) ¬
  EPub.university.stuID(dept=‘CS’)

RT_1 (Technical Details)

- A credential takes one of the following form:
  1. $K.r(h_1, \ldots, h_n) \leftarrow K_2$
  2. $K.r(h_1, \ldots, h_n) \leftarrow K_1.r_1(s_1, \ldots, s_m)$
  3. $K.r(h_1, \ldots, h_n) \leftarrow K.r_1(t_1, \ldots, t_L).r_2(s_1, \ldots, s_m)$
  4. $K.R \leftarrow K_1.R_1 \cap K_2.R_2 \cap \ldots \cap K_k.R_k$
- Each variable
  - must have a consistent data type across multiple occurrences
  - can have zero or more static constraints
  - must be safe, i.e., must appear in the body

Semantics and Complexity for RT_1

- LP semantics makes each role name a predicate
  - E.g., $K.r(h_1, ..., h_n) \leftarrow K_1.r_1(s_1, ..., s_m)$ translates to $r(K, h_1, ..., h_n, ?X)$:- $r_1 (K_1, s_1, ..., s_m, ?X)$
- Apply known complexity results: The atomic implications of *SP*(*P* **)** can be computed in $O(N^{v+3})$
  - *v* is the max number of variables per statement
  - Each role name has a most *p* arguments
  - $N = \max(N_0, pN_0)$, N_0 is the number of statements in *P*

$RT_2 = RT_1$ + Logical Objects

- Motivations:
  - to group logically related objects together and assign permissions about them together

- Approach: introducing o-sets, which are
  - similar to roles, but have values that are sets of things other than entities
  - defined through o-set definition credentials, which are similar to role-definition credentials in RT_1

RT_2 (Examples)

- Example 1: Alpha allows members of a project team to read documents of this project

  Alpha.documents(projectB) ← "design_Doc_for_projectB"

  Alpha.team(projectB) ← Bob

  Alpha.fileAccess(read, ?F Î Alpha.documents(?proj))
  ← Alpha.team(?proj)

- Example 2: Alpha allows manager of the owner of a file to access the file

  Alpha.read(?F) ← Alpha.manager(?E Î Alpha.owner(?F))

RT^T: Supporting Threshold and Separation-of-Duty

- Threshold: require agreement among *k* principals drawn from a given list
- SoD: requires **two or more different** persons be responsible for the completion of a sensitive task
  - want to achieve SoD without mutual exclusion, which is nonmonotonic
- Though related, neither subsumes the other
- RT^T introduces a primitive that supports both: manifold roles

Manifold Roles

- While a standard role is a set of principals, a manifold role is a set of sets of principals
- A set of principals that together occupy a manifold role can collectively exercise privileges of that role
- Two operators: ? , ?
  - $K_1.R_1$? $K_2.R_2$ contains sets of two distinct principals, one a member of $K_1.R_1$, the other of $K_2.R_2$
  - $K_1.R_1$? $K_2.R_2$ does not require them to be distinct

RT^T (Examples)

- Example 1: require a manager and an accountant
  - **K.approval ← K.manager ⊙ K.accountant**
  - members(K.approval) ⊇ {{x,y} | x ∈ K.manager, y ∈ K.accountant}
- Example 2: require a manager and a ***different*** accountant
  - **K.approval ← K.manager** Ä **K.accountant**
  - members(K.approval) ⊇ {{x,y} | x ≠ y, x ∈ K.manager, y ∈ K.accountant}

RT^T (Examples)

- Example 3: require three different managers
  - **K.approval ← K.manager** Ä **K.manager** Ä **K.manager**
  - members(K.approval) $\supseteq$ {{x,y,z} | x ≠ y ≠ z ∈ K.manager}

RT^T Syntax

- Manifold roles can be used in basic *RT* statements
- Also add two new types of policy statement
  - $K.R \leftarrow K_1.R_1$? $K_2.R_2$? ... ? $K_k.R_k$
    - members(K.R) ? $\{s_1$? ... ? s_k | s_i ?members($K_i.R_i$) for $1 = i = k\}$
  - $K.R \leftarrow K_1.R_1$? $K_2.R_2$? ... ? $K_k.R_k$
    - members(K.R) ? $\{s_1$? ... ? s_k | (s_i ?members($K_i.R_i$) & s_i n s_i ? Ø) for $1 = i$? $j = k\}$

RT^T Complexity

- ADSD must declare a *size* for each manifold role

- Given a set P of RT^T statements, let t be the maximal size of all roles in P. The atomic implications of P can be computed in time
  $O(MN^{v+2t})$.

Implementation and Application Status of RT

- Java Implementation of inference engine for RT_0
- Preliminary version of RTML
  - an XML-based Encoding of RT statements
  - XML Schemas and parser exist
  - Used in an ATN demo
- Applications
  - U-STOR-IT: Web-based file storage and sharing
  - August: A Distributed Calendar Program
  - Automated Trust Negotiation Demo by NAI

Next Lecture

- Security analysis in Trust Management