



# Trust Management

## CS 6463 Lecture 8

Prof. William Winsborough  
February 13, 2006

# Business

- Next week we will continue discussing the paper posted for today:  
N. Li, W. Winsborough, and J. C. Mitchell. Distributed Credential Chain Discovery in Trust Management. *Journal of Computer Security*, 11(1):35-86. February 2003.
- Synopsis due 2/20 should discuss pages 19-36 (sections 4-7)
  - Focuses on finding credential chains when credential storage is distributed
- Only one extended synopsis will be due for this paper
  - Due March 1
  - Make it a really good extended synopsis

# Today's Outline

- KeyNote adequacy: discussion
- Discussion of KeyNote projects
- Start discussing  $RT_0$

# KeyNote's Adquacy

- Determining whether a principal has any authorizations according to a given policy seems to be undecidable
  - Encode Hilbert's Tenth Problem in conditions:
    - Hilbert's Tenth Problem: Determination of the solvability of a Diophantine equation. Given a Diophantine equation with any number of unknown quantities and with rational integral numerical coefficients. The problem is to devise a process according to which it can be determined by a finite number of operations whether the equation is solvable in rational integers.
    - <http://logic.pdmi.ras.ru/~yumat/H10Pbook/index.html>
    - [http://www.cs.cmu.edu/~lblum/courses/flac/FLAC\\_Projects/JoeRoth\\_flac\\_project\\_final.pdf](http://www.cs.cmu.edu/~lblum/courses/flac/FLAC_Projects/JoeRoth_flac_project_final.pdf)
- Other problems with this design?
  - How are credentials found?
  - Who stores them?

# KeyNote-based Project Ideas

- Certainly one could build a distributed application using KeyNote as the basis of a course project
- Such a project would need to be organized around testing some hypothesis about the suitability of KeyNote for solving some identifiable group of authorization problems

# Policy Language Wish List

- Decentralize authority to define attributes
  - Utilize policy and credentials from many sources
- Delegation of attribute authority
  - To specific principals
  - To principals with certain attributes
- Inference of attributes
  - E.g., derive access rights based on roles or other characteristics
- Intersection of attributes
- Parameterization
- Support for thresholds, separation of duty

# Role-based Trust Management (*RT*)

- A family of credential / policy languages
  - Simplest,  $RT_0$ , has no parameterization, thresholds, or separation of duty
- $RT_0$  example: student discount subscription
  - EPub.studentDiscount  $\leftarrow$  StateU.student
  - StateU.student  $\leftarrow$  URegistrar.fulltimeLoad
  - StateU.student  $\leftarrow$  URegistrar.parttimeLoad
  - URegistrar.parttimeLoad  $\leftarrow$  Alice

# Role-based Trust Management (*RT*)

- A family of credential / policy languages
  - Simplest,  $RT_0$ , has no parameterization, thresholds, or separation of duty
- $RT_0$  example: student discount subscription
  - $EPub.studentDiscount \leftarrow StateU.student$
  - $StateU.student \leftarrow URegistrar.fulltimeLoad$
  - $StateU.student \leftarrow URegistrar.parttimeLoad$
  - $URegistrar.parttimeLoad \leftarrow Alice$
- Credential chain proves authorization

# Example: Attribute-based Delegation

- Accepting student ID from **any** university
  - EPub.studentDiscount ← FAB.accredited.student
  - FAB.accredited ← StateU
  - StateU.student ← URegistrar.fulltimeLoad
  - StateU.student ← URegistrar.parttimeLoad
  - URegistrar.parttimeLoad ← Alice

# Example: Expressivity in Credentials

- Deferring a Guaranteed Student Loan
  - `BankWon.deferGSL` ← `FAB.accredited.fulltimeStudent`
  - `FAB.accredited` ← `StateU`
  - `StateU.fulltimeStudent` ← `URegistrar.fulltimeLoad`
  - `StateU.fulltimeStudent` ← `URegistrar.parttimeLoad`  $\cap$   
`StateU.gradOfficer.phdCandidate`
  - `URegistrar.parttimeLoad` ← `Bob`
  - `StateU.gradOfficer` ← `Carol`
  - `Carol.phdCandidate` ← `Bob`

# $RT_0$ Syntax

- Basic structure is a role (i.e., an attribute):  $A.r$ 
  - $A$  is a principal (authority for  $A.r$ ),  $r$  is a role name
- Four types of policy statement
  - $A.r \leftarrow D$   
Role  $A.r$  contains principal  $D$  as a member
  - $A.r \leftarrow B.r_1$   
 $A.r$  contains role  $B.r_1$  as a subset
  - $A.r \leftarrow A.r_1.r_2$   
 $A.r$  contains  $B.r_2$  as a subset, for each  $B$  in  $A.r_1$
  - $A.r \leftarrow A_1.r_1 \cap A_2.r_2$   
 $A.r$  contains the intersection
- A credential is a statement signed by  $A$ , the credential issuer and the authority over  $A.r$
- The first 3 statement types give a language equivalent to pure SDSI

# Set-based Semantics for $RT_0$

- Given as set of credentials  $C$ , the semantics of  $C$  is  $S_C : \text{Role} \rightarrow \wp(\text{Entity})$  given by the least function  $\text{rmem} : \text{Role} \rightarrow \wp(\text{Entity})$  that satisfies the following system of set inequalities:

$$\{ \text{rmem}(A.r) \supseteq \text{expr}[\text{rmem}](e) \mid A.r \leftarrow e \in C \}$$

- Where

- $\text{expr}[\text{rmem}](B) = \{B\}$
- $\text{expr}[\text{rmem}](A.r) = \text{rmem}(A.r)$
- $\text{expr}[\text{rmem}](A.r_1.r_2) = \bigcup_{B \in \text{rmem}(A.r_1)} \text{rmem}(B.r_2)$
- $\text{expr}[\text{rmem}](f_1 \wedge f_2 \wedge \dots \wedge f_k) = \bigcap_{1 \leq j \leq k} \text{expr}[\text{rmem}](f_j)$

# Logic-Programming Semantics

## ■ Four credential types

- $A.r \leftarrow D$  (type 1)
  - $m(A, r, D).$
- $A.r \leftarrow B.r_1$  (type 2)
  - $m(A, r, ?X) :- m(B, r_1, ?X).$
- $A.r \leftarrow A.r_1.r_2$  (type 3)
  - $m(A, r, ?X) :- m(A, r_1, ?Y), m(?Y, r_2, ?X).$
- $A.r \leftarrow A_1.r_1 \cap A_2.r_2 \cap \dots \cap A_k.r_k$  (type 4)
  - $m(A, r, ?X) :- m(A_1, r_1, ?X), \dots, m(A_k, r_k, ?X).$

# Credential Chain Discovery

## ■ Outline

- Sound and complete evaluation model
  - Based on graphical model (“credential graph”)
- Efficient search for proof of authorization
- Provides a basis for discovering chains when credentials are stored in a distributed manner
  - Evaluation steps can be interleaved with credential retrieval steps

# Example: Student ACM Discount

- $\text{EPub.studentACM} \leftarrow \text{EOrg.student} \cap \text{ACM.member}$
- $\text{EOrg.student} \leftarrow \text{EOrg.university.student}$
- $\text{EOrg.university} \leftarrow \text{FAB.accredited}$

*Credential Discovered  
in Backward Direction*

- $\text{FAB.accredited} \leftarrow \text{StateU}$
- $\text{StateU.student} \leftarrow \text{URegistrar.parttimeLoad}$
- $\text{URegistrar.parttimeLoad} \leftarrow \text{Alice}$
- $\text{ACM.member} \leftarrow \text{Alice}$

*Credential Discovered  
in Forward Direction*

# Credential Graph

EPub gives a double subscription discount:

EPub.studentacm

EOrg.student  $\cap$  ACM.member

EOrg.student

EOrg.university

EOrg.university.student

FAB.accredited

StateU.student

StateU

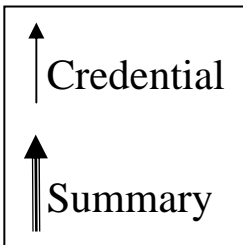
URegistrar.parttimeLoad

ACM.member

Alice

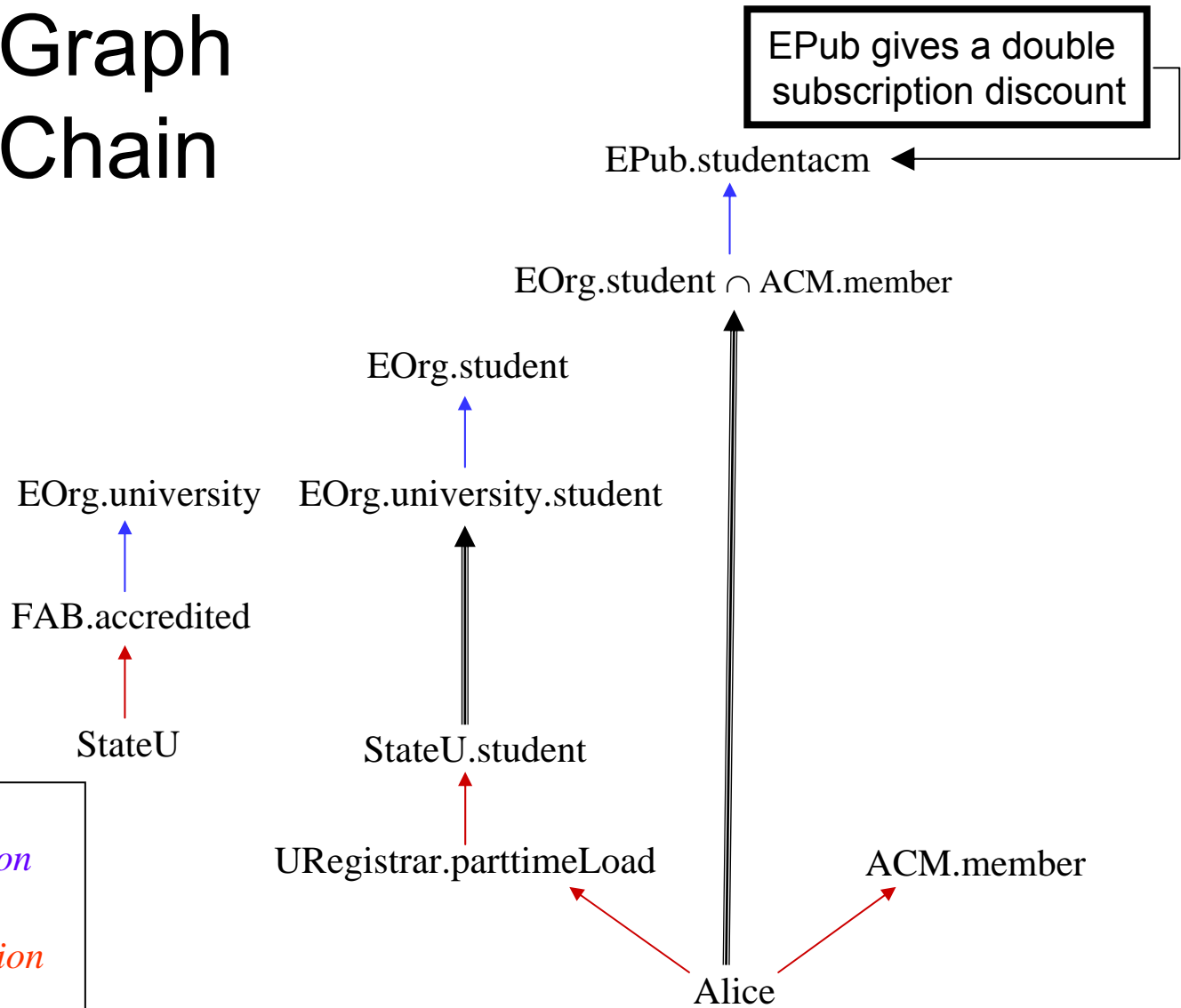
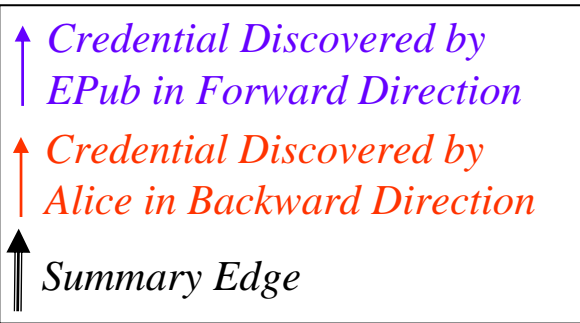
- Type 1: FAB.accredited  $\leftarrow$  StateU
- Type 2: EOrg.university  $\leftarrow$  FAB.accredited
- Type 3: EOrg.student  $\leftarrow$  EOrg.university.student
- Type 4: EPub.studentacm  $\leftarrow$  EOrg.student  $\cap$  ACM.member

## Key



# Credential Graph Organizes Chain Discovery

## Key



# Algorithmic Contributions

- Search algorithms:
  - Worst case efficiency as good as any existing algorithm
    - Backward.  $O(N^3)$  time,  $N$  = number of credentials
    - Forward.  $O(N^2M)$  time,  $M$  = sum of credential sizes
    - Both directions.  $O(N^2M)$  time
  - Well suited to the application
    - Efficient when there are lots of unrelated credentials
    - Changes to credential pool do not degrade performance
    - Graph search can drive credential discovery