Lecture 15 (March 8)
Distributed Credential Chain Discovery in Trust Management
Review: An Example in SDSI 2.0

- SDSI Certificates
  - \((K_C \text{ access} \iff K_C \text{ mit faculty secretary})\)
  - \((K_C \text{ mit} \iff K_M)\)
  - \((K_M \text{ faculty} \iff K_{\text{EECS faculty}})\)
  - \((K_{\text{EECS faculty}} \iff K_{\text{Rivest}})\)
  - \((K_{\text{Rivest secretary}} \iff K_{\text{Rivest alice}})\)
  - \((K_{\text{Rivest alice}} \iff K_{\text{Alice}})\)

- From the above certificates, \(K_C\) concludes that \(K_{\text{Alice}}\) has access
Recap of the SDSI Rewriting-based Semantics

- Defines answers to queries having the form “can $\omega_1$ rewrite into $\omega_2$?”
- Specialized algorithms (either developed for SDSI or for model checking pushdown systems) are needed
- Papers by Abadi and Halpern and van der Meyden try to come up with axiom systems for the rewriting semantics
Defining Set-based Semantics

(1)

A valuation \( V \) maps each local name to a set of principals.

A valuation \( V \) can be extended to map each name string to a set of principals:

- \( V(K) = \{ K \} \)
- \( V(K \ A) = V(K A) \)
- \( V(K \ B_1 \ldots B_m) = \bigcup_{j=1}^{n} V(K_j \ B_2 \ldots B_m) \)
- where \( m > 1 \) and \( V(K \ B_1) = \{K_1, K_2, \ldots, K_n\} \)
Defining Set-based Semantics (2)

- A 4-tuple \((K A \leftrightarrow \omega)\) is the following constraint
  - \(V (K A) \supseteq V (\omega)\)
- The semantics of a set \(P\) of 4-tuples is the least valuation \(V_P\) that satisfies all the constraints
- Queries
  - “can \(\omega\) rewrite into \(K\)?” answered by checking whether “\(K \in V_P (\omega)\)”. 
- Does not define answers to “can \(\omega_1\) rewrite into \(\omega_2\)”. 
  - asking whether \(V_P (\omega_1) \supseteq V_P (\omega_2)\) is incorrect
Relationship Between Rewriting and Set Semantics

- **Theorem**: Given $P$, $\omega_1$, and $\omega_2$, $\omega_1$ rewrites into $\omega_2$ using $P$ if and only if for any $P' \supseteq P$, $V_{P'}(\omega_1) \supseteq V_{P'}(\omega_2)$.

- **Corollary**: Given $P$, $\omega$, and $K$, $\omega$ rewrites into $K$ using $P$ if and only if $V_P(\omega) \supseteq \{K\}$.
What is RT?

- RT is a family of Role-based Trust-management languages

Publications on RT

- Li, Winsborough & Mitchell: “Distributed Credential Chain Discovery in Trust Management”, JCS’01, CCS’01
- Li, Mitchell & Winsborough: “Design of a Role-Based Trust Management Framework”, S&P’02
- Li & Mitchell: “Datalog with Constraints: A Foundation for Trust Management Languages”, PADL’03
- Li & Mitchell: “RT: A Role-based Trust-management Framework”, DISCEX’03
RT$_0$: An Example

1. StateU.stuID $\leftarrow$ Alice
2. ABU.accredited $\leftarrow$ StateU
3. EPub.university $\leftarrow$ ABU.accredited
4. EPub.student $\leftarrow$ EPub.university.stuID
5. EPub.spdiscount $\leftarrow$
   
   EPub.student $\cap$ EOrg.preferred

6. EOrg.preferred $\leftarrow$ ACM.member
7. ACM.member $\leftarrow$ Alice

Together, the seven credentials prove that Alice is entitled to EPub’s spdiscount
RT₀: Concepts and Credentials

- Concepts:
  - Entities (Principals): A, B, D
  - Role names: r, r₁, r₂, ...
  - Roles: A.r, B.r₁, ... e.g., StateU.stuID

- Credentials: A.r ← e
  - Type-1: A.r ← D
  - Type-2: A.r ← B.r₁
  - Type-3: A.r ← A.r₁.r₂
    - e.g., EPub.student ← EPub.university.stuID
  - Type-4: A.r ← B₁.r₁ ∩ B₂.r₂ ∩ ... ∩ Bₖ.rₖ
RT₀ and SDSI 2.0

- SDSI 2.0 (The SDSI part of SPKI/SDSI 2.0)
  - has arbitrarily long linked names, e.g., A.r₁.r₂.....rₖ, which can be broken up by introducing new role names

- RT₀
  - has intersection (type-4 credentials)
  - is thus more expressive than SDSI 2.0
    - algorithms for RT₀ can be used for SDSI 2.0
Goal-directed Chain Discovery

Three kinds of queries and algorithms for answering them:

1. Given A.r, determines its members
   - The backward search algorithm
2. Given D, determines the set of roles that D is a member of
   - The forward search algorithm
3. Given A.r and D, determines whether D is a member of A.r
   - The Bi-direction search algorithm
Credential Graph $G_C$

- **Nodes:**
  - For each credential $A.r \leftarrow e$ in $C$
  - $A.r$ and $e$

- **Credential edges:**
  - $e \rightarrow A.r$ for each credential $A.r \leftarrow e$ in $C$

- **Summary edges:**
  - $B.r_2 \rightarrow A.r_1.r_2$ if there is a path from $B$ to $A.r_1$
  - $D \rightarrow A_1.r_1 \cap ... \cap A_k.r_k$ if there are paths from $D$ to each $A_j.r_j$

- Reachability in the credential graph is sound and complete wrt. the set semantics of $RT_0$
An Example Credential Graph

Key

Credential

Summary

StateU

EPub.university

ABU.accredited

EPub.university.stuID

StateU.stuID

EPub.student

EPub.student ∩ EOrg.preferred

EPub.spdiscount

EOrg.preferred

ACM.member

Alice
The Forward Search Algorithm (Overview)

- Starts with one entity node
- Constructs a proof graph
- Each node in the graph stores its solutions:
  - roles that this node can reach (is a member of )
- Maintains a work list of nodes need to be processed
- Algorithm Outline:
  - keep processing nodes in the work list until it is empty
### Forward Search In Action

<table>
<thead>
<tr>
<th>Step</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.</td>
<td>StateU.stuID ← Alice</td>
</tr>
<tr>
<td>2.</td>
<td>ABU.accredited ← StateU</td>
</tr>
<tr>
<td>3.</td>
<td>EPub.university ← ABU.accredited</td>
</tr>
</tbody>
</table>

#### Diagram:

- **0: Alice**
  - StateU.stuID
  - EPub.student
- **1: StateU.stuID**
  - StateU.stuID
  - EPub.student
- **2: StateU**
  - ABU.accredited
  - EPub.university
- **3: ABU.accredited**
  - ABU.accredited
  - EPub.university
- **4: ABU.accredited.stuID**
- **5: ABU**
- **6: EPub.university**
  - EPub.university
- **7: Epub.university.stuID**
- **8: EPub**
- **9: EPub.student**
The Backward and Bi-direction Search Algorithms (Overview)

- The backward algorithm differs from the forward algorithm in that:
  - each node stores outgoing edges, instead of incoming ones
  - each node stores entities that can reach it, instead of roles that it can reach
  - the processing of a node is different
    - traversing the other direction

- The bi-direction search algorithm combines backward search and forward search
Backward Search In Action

17: Alice

7: Alice

0: EPub.spdiscount

10: StateU.stuID

9: StateU

8: ABU.accredited

6: EPub.university

4: EPub.university.stuID

5: ACM.member

3: EOrg.preferred

2: EPub.student

1: EPub.student ∩ EOrg.preferred

Alice

Alice

Alice

Alice

Alice

Alice

Alice

Alice

Alice

Alice
Worst-Case Complexity

- Backward: time $O(N^3+NM)$, space $O(NM)$
  - $N$ is the number of rules
  - $M$ is the sum of the sizes of all rules,
    - $A.r \leftarrow f_1 \cap \ldots \cap f_k$ having size $k$, other credentials have size 1
- Forward: time $O(N^2M)$, space $O(NM)$
- However, this is goal oriented, making it much better in practice
Why Develop These Algorithms?

- The queries can be answered using logic programs
  - however, this requires collection of all credentials in the system
- The backward algorithm is a goal-directed top-down algorithm
- The forward algorithm is a goal-directed bottom-up algorithm
- Distributed discovery requires combination of both
Distributed Storage of Credentials

- Example:
  1. `EOrg.preferred ← ACM.member`
  2. `ACM.member ← Alice`

- Who should store a credential?
  - either issuer or subject

- It is not reasonable to require that
  - all credentials are stored by issuers, or,
  - all are stored by subjects.
Who stores these statements?

4. EPub.university $\leftarrow$ ABU.accredited
5. EPub.student $\leftarrow$
   EPub.university.stuID

1. COE.stuID $\leftarrow$ Alice
2. StateU.stuID $\leftarrow$ COE.stuID
3. ABU.accredited $\leftarrow$ StateU
Traversability of Edges and Paths

- A credential edge is
  - forward traversable, if stored by subject
  - backward traversable, if stored by issuer
  - confluent, if either forward traversable or backward traversable

- A path $e_1 \xrightarrow{} e_2$ is
  - forward traversable, if all edges on it are, or $e_1=e_2$
  - backward traversable, if all edges on it are, or $e_1=e_2$
  - confluent, if it can be broken into $e_1 \xrightarrow{} e' \rightarrow e'' \xrightarrow{} e_2$,
    - With $e_1\xrightarrow{}e'$ forward, $e' \rightarrow e''$ confluent, and $e'' \xrightarrow{} e_2$ backward
Traversability of Edges and Paths (con’d)

An edge $B.r_2 \rightarrow A.r_1.r_2$ has the same traversability as $B \Rightarrow A.r_1$
How to Ensure that Every Path is Confluent?

- **Goal:** using constraints local to each credential to ensure that every path is confluent

- **Approach:**
  - give each role name a traceability type
  - introduce a notion of well-typed credentials

- **Main idea:**
  - by requiring consistent storage strategy at role name level, we guarantee chains using well-typed credentials are confluent
Types of Role Names

- A role name has two types:
  - Issuer side:
    - issuer-traces-all
    - issuer-traces-def
    - issuer-traces-none
  - Subject side:
    - subject-traces-all
    - subject-traces-none
A Typing Scheme

**EPub**

4. `EPub.university ← ABU.accredited`

**ABU**

3. `ABU.accredited ← StateU`

**StateU**

2. `StateU.stuID ← COE.stuID`

**COE**

1. `COE.stuID ← Alice`
Well-typed Credentials

- A credential $A.r \leftarrow e$ is well-typed if:
  - Both $A.r$ and $e$ are well-typed
    - A role $A.r$ has the same type as $r$
    - A role expression is well-typed if it is not both issuer-none and subject-none
  - If $A.r$ is issuer-def or issuer-all, then $A$ must store the credential
  - If $A.r$ is subject-all, then every subject of the credential must store it
  - If $A.r$ is issuer-all, then $e$ must be issuer-all
  - If $A.r$ is subject-all, then $e$ must be subject-all
Agreement on Types and Meaning of Role Names

- An approach inspired by XML namespaces
  - Use an Application Domain Specification Document (ADSD) to define a vocabulary
    - Each role has a storage type
  - Credentials have a preamble
    - Which defines vocabulary identifier to correspond to an ADSD
  - When using a role name, add a vocabulary identifier as prefix
Main Result about Type System

- Given a set of well-typed credentials \( C \), if \( D \rightarrow e \)
  - \( D \rightarrow e \) is confluent
  - if \( e \) is issuer-traces-all, \( D \rightarrow e \) is backward traversable
  - if \( e \) is subject-traces-all, \( D \rightarrow e \) is forward traversable
Benefits of the Storage Type System

- Guarantees that chains of well-typed credentials can be discovered
- Enables efficient chain discovery by telling the algorithm whether forward or backward search should be used for an intermediate query
- Communicates the application domain knowledge to the algorithm
Next Lecture

- More on SDSI Semantics and the RT Languages