Trust Management II

Anupam Datta
Fall 2007-08
Logistics

- Please plan to meet with me over this week to discuss project progress and issues
  - Office hours
  - Friday 2-3:30PM
Overview

- Last lecture
  - Overview of Trust Management [BFL96]
  - RT syntax and examples [LMW02]

- Today
  - Translating RT into datalog; algorithmic results [LMW02]
  - Distributed credential chain discovery [LWM01]
Translating RT₁ to Datalog

- **Type I:**
  - $A.R \leftarrow D$ to
  - isMember(D, A.R)

- **Type II:**
  - $A.R \leftarrow B.R₁$ to
  - isMember(?z, A.R) ← isMember(?z, B.R₁)

- **Type III:**
  - $A.R \leftarrow A.R₁.R₂$ to
  - isMember(?z, A.R) ← isMember(?x, A.R₁), isMember(?z,?x.R₂)

- **Type IV:**
  - $A.R \leftarrow B₁.R₁ \cap B₂.R₂ \cap ... \cap Bₙ.Rₙ$ to
  - isMember(?z, A.R) ← isMember(?z, B.R₁),..., isMember(?z,B.Rₙ)
Translation (2)

- `isMember(?z, A.r(h1,...,hn))` to
- `member(A, r, h1,..., hn, ?z)`

- `Trans(C): Datalog program resulting from translation of RT_1` credentials C

- Implications of C: Set of membership relations implied by C
Theorem: Given a set $C$ of $RT_1$ credentials, the implications of $C$ can be computed in time $O(MN^{v+2})$, where

- Each credential of $C$ has at most $v$ variables
- Each role name has at most $p$ arguments
- $N_0$ is the number of credentials in $C$
- $N = \max(N_0, pN_0)$
- $M$ is the size of $C$
  - $A.r \leftarrow f_1 \cap \ldots \cap f_k$ has size $k$, other credentials have size 1
Example

- RT Credentials
  - $A.R_i \leftarrow A.R_{i+1}$ for $i=1,...,K-1$
  - $A.R_K \leftarrow Alice$, $A.R_K \leftarrow Bob$

- Translation
  1. $\text{isMember(?z, A.R_i)} \leftarrow \text{isMember(?z, A.R_{i+1})}$ for $i=1,...,K-1$
  2. $\text{isMember(Alice, A.R_K), isMember(Bob, A.R_K)}$

- Start from ground facts; repeatedly apply rule 1 to infer implications in time $O((K+1)*2)$
  - $p = 0, v = 0, N_0 = K + 1, N = K + 1, M = K + 1$
  - $O(MN^{v+2}) = O((K+1)*(K+1)^2)$
  - Translation only introduces only one dummy variable because no linked roles, so $(v+1)$ instead of $(v+2)$
Summary

- Similar translations into Datalog and tractability results for other languages in the RT family

- This is an advantage of RT over other formalisms for distributed access control, e.g. Lampson et al logic (for which the implication problem is undecidable)
Overview

- Last lecture
  - Overview of Trust Management [BFL96]
  - RT syntax and examples [LMW02]
- Today
  - Translating RT into datalog; algorithmic results [LMW02]
  - Distributed credential chain discovery [LWM01]
Goal-directed Chain Discovery

- Three kinds of queries and algorithms for answering them in $RT_0$:
  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
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
Credential Graph $G_C$

- **Nodes:**
  - $A.r$ and $e$ for each credential $A.r \leftarrow e$ in $C$
  
- **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-theoretic semantics of $RT_0$
An Example Credential Graph

Key
- Credential
- Summary

EPub.university
- ABU.accredited
  - StateU

EPub.student
- EOrg.preferred

EPub.university.stuID

EPub.spdiscount

StateU.stuID

Alice

EOrg.preferred

ACM.member
The Backward Search Algorithm (Overview)

- Goal:
  - Given A.r, determines its members
- The backward algorithm:
  - each node stores outgoing edges
  - each node stores entities that can reach it
  - new edges are created in the reverse direction
  - solutions are propagated forward
Backward Search In Action
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

1. StateU.stuID ← Alice
2. ABU.accredited ← StateU
3. EPub.university ← ABU.accredited

0: Alice
StateU.stuID
EPub.student

2: StateU
ABU.accredited
EPub.university

1: StateU.stuID
StateU.stuID
EPub.student

3: ABU.accredited
ABU.accredited
EPub.university

4: ABU.accredited.stuID

5: ABU

6: EPub.university

7: Epub.university.stuID

8: EPub

9: EPub.student
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
Bidirectional Search

- The bi-direction search algorithm combines backward search and forward search
- Bidirectional(e, D)
  - Two queues: Forward processing and backward processing queue
    - Iteratively remove and process node until both queues are empty
  - Each node e stores
    - Backward solutions, backward monitors
    - Full and partial forward solutions, forward monitors
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 ← ABU.accredited
5. EPub.student ←
   EPub.university.stuID

1. COE.stuID ← Alice
2. StateU.stuID ← COE.stuID
3. ABU.accredited ← StateU

Who stores these statements?
Traversability of Edges and Paths

Idea: Confluent paths can be discovered by bidirectional search

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:** Use 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, 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 (e.g. A1.r ← A2.r ← A3.r)
  - issuer-traces-none

- **Subject side:**
  - subject-traces-all
  - subject-traces-none
Well-typedness of Role Names

Definition 5 (Well-typedness of Role Names)

- A role name is strongly well typed if it is issuer-traces-all or subject-traces-all.
- A role name is weakly well typed if it is both issuer-traces-def and subject-traces-none.
- A role name is well typed if it is strongly well typed or weakly well typed.
- A role name is ill-typed if it is not well typed. In other words, a role name is ill-typed if it is both issuer-traces-none and subject-traces-none.
Typing role expressions

Definition 6 (Well-typedness of role expressions)

- An entity $A$ is both issuer-traces-all and subject-traces-all.
- A role $A.r$ has the same type as $r$.
- A linked role $A.r_1.r_2$ is
  \[
  \begin{cases}
  \text{issuer-traces-all} & \text{if both } r_1 \text{ and } r_2 \text{ are issuer-traces-all} \\
  \text{subject-traces-all} & \text{if both } r_1 \text{ and } r_2 \text{ are subject-traces-all} \\
  \text{weakly well typed} & \text{otherwise, if either } r_1 \text{ is issuer-traces-all and } r_2 \text{ is well typed,} \\
  & \text{or } r_1 \text{ is well typed and } r_2 \text{ is subject-traces-all} \\
  \text{ill-typed} & \text{otherwise}
  \end{cases}
  \]
- An intersection $f_1 \cap \cdots \cap f_k$ is
  \[
  \begin{cases}
  \text{issuer-traces-all} & \text{if there exists an } f_\ell \text{ that is issuer-traces-all, and all } f_j \text{'s are well typed} \\
  \text{subject-traces-all} & \text{if there exists an } f_\ell \text{ that is subject-traces-all, and all } f_j \text{'s are well typed} \\
  \text{weakly well typed} & \text{if all } f_j \text{'s are weakly well typed} \\
  \text{ill-typed} & \text{otherwise}
  \end{cases}
  \]
Definition 7 (Well Typed Credentials)

A credential \( A.r \leftarrow e \) is *structurally well typed* if the following three conditions are satisfied:

1. Both \( A.r \) and \( e \) are well typed.
2. If \( A.r \) is issuer-traces-all, \( e \) must also be issuer-traces-all.
3. If \( A.r \) is subject-traces-all, \( e \) must also be subject-traces-all.

A credential \( A.r \leftarrow e \) is *well typed* if it is structurally well typed and satisfies the following two storage requirements:

- If \( A.r \) is issuer-traces-def or issuer-traces-all, \( A \) stores this credential.
- If \( A.r \) is subject-traces-all, every subject of this credential stores this credential.
Example

EPub.spdiscount ← EOrg.preferred ∩ ACM.member \hspace{1cm} (1)
EOrg.preferred ← EOrg.university.student \hspace{1cm} (2)
EOrg.university ← ABU.accredited \hspace{1cm} (3)
StateU.student ← RegistrarB.student \hspace{1cm} (5)
ACM.member ← Alice \hspace{1cm} (7)
ABU.accredited ← StateU \hspace{1cm} (4)
RegistrarB.student ← Alice \hspace{1cm} (6)

One appropriate type assignment is as follows:

\[
\begin{cases}
\text{spdiscount, preferred, university} & \text{are \: issuer-traces-def and subject-traces-none} \\
\text{accredited, student, member} & \text{are \: issuer-traces-none and subject-traces-all}
\end{cases}
\]

Under this type assignment, all the credentials are structurally well typed. To satisfy the storage requirements for well typed credentials, credential (1) is stored with EPub, (2) and (3) are stored with EOrg, (4) is stored with StateU, (5) is stored with RegistrarB, and both (6) and (7) are stored with Alice. One can verify that the chain \(\langle\text{EPub}.\text{discount} \leftarrow \text{Alice}\rangle\) in Figure 4 can be discovered by doing bidirectional search.
Figure 4: The chain \( \langle E\text{Org}.\text{discount} \leftarrow \text{Alice} \rangle \) discovered by \texttt{DBidirectional}(\text{E\text{Org}.\text{discount}, Alice}), with the set of credentials in Example 3, typed as in Example 5. Edges with numbers are credential edges; the numbers above (or to the right of) edges correspond to the credential numbers in Example 3. Double arrows originate at a role expression whose base stores the corresponding credential. Thus, if the double arrow has the same direction as the edge, the credential can be discovered by forward search. Otherwise, the credential can be discovered by backward search.
Properties

Theorem 9 (Traversability of Well Typed Chains) Given a set $C$ of well typed credentials, and any chain $\langle e \leftarrow e_1 \rangle$ in a credential graph $G_{C,Q}$ of $C$, the following propositions hold:

1. If $e$ is well typed, $\langle e \leftarrow e_1 \rangle$ is confluent.
2. If $e$ is issuer-traces-all, $\langle e \leftarrow e_1 \rangle$ is backward traversable.
3. If $e$ is subject-traces-all, $\langle e \leftarrow e_1 \rangle$ is forward traversable.

Theorem 10 Given a set $C$ of well typed credentials, the following three propositions hold:

1. Given an entity $D$ and a role expression $e$ that is well typed, one can use $\text{DBidirectional}(D, e)$ to determine whether $D \in \text{expr}[S_C](e)$.
2. Given a role expression $e$ that is issuer-traces-all, one can use $\text{DBackward}(e)$ to determine $\text{expr}[S_C](e)$.
3. Given an entity $D$, one can use $\text{DForward}(D)$ to determine all the roles $A.r$ such that $A.r$ is subject-traces-all and $D \in S_C(A.r)$.
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
Acknowledgement

- Slides for the second part of the lecture were based, in part, on slides from Ninghui Li and John Mitchell.