Distributed Access Control

- Logics
  - ABLP, PCA, GP, ...

- Trust Management
  - PolicyMaker, RT, ...

Last 3 lectures

Next 2 lectures
Discussion Question

What are the points of similarity and difference between access control logics and trust management languages?
Readings

- Blaze, Feigenbaum, Lacy, "Decentralized Trust Management", S&P'96
- Li, Mitchell & Winsborough: "Design of a Role-Based Trust Management Framework", S&P'02
What is “Trust Management”? 

“It is our thesis that a coherent intellectual framework is needed for the study of security policies, security credentials, and trust relationships. We refer collectively to these components of network services as the trust management problem.”

- Blaze, Feigenbaum, Lacy, 1996
Example

An electronic banking system must enable a bank to

- state that at least k bank officers are needed to approve loans of $1,000,000 or more (a policy);
- it must enable a bank employee to prove that he can be counted as 1 out of k employees (a credential)
- it must enable a bank to specify who may issue such a credential (a trust relationship)
Principles [BFL96]

◆ Unified mechanism
  • Policies, credentials and trust relationships are expressed as programs in a “safe” programming language

◆ Flexibility
  • Expressive enough to support complex trust relationships

◆ Locality of control
  • Each party can decide whether to accept the credentials of a second party or, alternatively, which third party it should rely on for the appropriate “certificate”

◆ Separation of mechanism from policy
  • The mechanism for verifying credentials does not depend on the credentials themselves or the semantics of the applications that use them
Standard PKI doesn’t work

- PGP and X.509 certificates answer the question:
  - Who is the holder of this public key?
- TM seeks to answer the more relevant question:
  - Can we trust this public key for this purpose?
  - Need credentials that associate attributes with principals, e.g. Alice is a bank employee
  - Policies can refer to attributes
Early TM languages

◆ **PolicyMaker**

◆ **KeyNote**

◆ **SPKI (Simple Public Key Infrastructure) / SDSI (Simple Distributed Security Framework)**
  - Clarke et al.: Certificate Chain Discovery in SPKI/SDSI, JCS’01.
Datalog-based TM languages

◆ Delegation Logic
  • Li, Grosof & Feigenbaum: “Delegation Logic: A Logic-based Approach to Distributed Authorization”, TISSEC’03. (Conference versions appeared in CSFW’99 and S&P’00)

◆ SD3 (Secure Dynamically Distributed Datalog)

◆ Binder

◆ RT: A Family of Role-based Trust-management Languages
  • Li, Mitchell & Winsborough: “Design of a Role-Based Trust Management Framework”, S&P’02
Concepts

- **Principal**
  - Individual or entity, identified by public key

- **Right**
  - Action and Resource

- **Policy**
  - Statements that associate rights with principals
  - May classify resources, other auxiliary concepts

- **Delegation**
  - Owner A gives a right to B
  - B delegates that right to C
Attribute-based Access Control

◆ **ABAC systems should be able to express the following:**

1. **Decentralized attributes**
   ◆ An entity asserts that another entity has an attribute

2. **Delegation of attribute authority**
   ◆ An entity delegates (trusts the judgment of) another entity on an attribute

3. **Inference of attributes**

4. **Attribute fields**
   ◆ Attribute may have field values e.g. age, credit limit

5. **Attribute-based delegation of attribute authority**
   ◆ Delegate to entities that are certified universities the authority to identify students
ABAC and TM languages

- Keynote, SPKI 1.0, X.509 cannot express 3 & 5
- SDSI 1.0, SPKI/SDSI 2.0 cannot express 4
- RT expresses all 5
Features of the RT family

- 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
An Example of RT\(_0\)

1. \text{StateU.stuID} \leftarrow \text{Alice}
2. \text{ABU.accredited} \leftarrow \text{StateU}
3. \text{EPub.university} \leftarrow \text{ABU.accredited}
4. \text{EPub.student} \leftarrow \text{EPub.university.stuID}
5. \text{EPub.access} \leftarrow \text{EPub.student}

Five statements prove Alice is entitled to access
Expressive Features (I)

I. Simple attribute assignment
   \[ \text{StateU.stuID} \leftarrow \text{Alice} \]

II. Delegation of attribute authority
    \[ \text{StateU.stuID} \leftarrow \text{COE.stuID} \]

III. Attribute inferencing
     \[ \text{EPub.access} \leftarrow \text{EPub.student} \]

IV. Attribute-based delegation of authority
    \[ \text{EPub.student} \leftarrow \text{EPub.university.stuID} \]

- Role: set of entities
- Credentials define role membership
Expressive Features (part two)

V. Conjunction

EPub.access ← EPub.student ∩ 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
Languages in the RT Framework

- **RT\(^0\):** Decentralized Roles
- **RT\(^1\):** Parameterized Roles
- **RT\(^T\):** for Separation of Duties
- **RT\(^D\):** for Selective Use of Role memberships
- **RT\(^2\):** Logical Objects
- **RT\(^1C\):** structured resources
- **RT\(^2C\):** structured resources

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\(^1C\), and RT\(^2C\).
\[ RT_1 = RT_0 + \text{Parameterized Roles} \]

\begin{itemize}
  \item \textbf{Motivations: to represent}
    \begin{itemize}
      \item attributes that have fields, e.g., digital ids, diplomas
      \item relationships between principals, e.g., physicianOf, advisorOf
      \item role templates, e.g., project leaders
    \end{itemize}
  \end{itemize}

\begin{itemize}
  \item \textbf{Approach:}
    \begin{itemize}
      \item a role term \( R \) has a role name and a list of fields
    \end{itemize}
\end{itemize}
Example 1: Alpha allows manager of an employee to evaluate the employee:

\[
\text{Alpha.evaluatorOf}(\text{employee}=y) \leftarrow \\
\text{Alpha.managerOf}(\text{employee}=y)
\]

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

\[
\text{EPub.access}(\text{action}='\text{read}', \text{resource}='\text{file1}') \leftarrow \\
\text{EPub.university.stuID}(\text{dept}='\text{CS}')
\]
Syntax of RT₁ Credentials

- **Type I:**
  - $A.r(h₁,…,hₙ) \leftarrow D$

- **Type II:**
  - $A.r(h₁,…,hₙ) \leftarrow B.r₁(s₁,…,sₙ)$

- **Type III:**
  - $A.r(h₁,…,hₙ) \leftarrow A.r₁(t₁,…,tₗ).r₂(s₁,…,sₘ)$

- **Type IV:**
  - $A.R \leftarrow B₁.R₁ \cap B₂.R₂ \cap … \cap B_k.R_k$
RT^T: Supporting Threshold and Separation-of-Duty

- Threshold: require agreement among \( k \) different principals drawn from a given list
  - Intersection of roles not sufficient

- 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 non-monotonic

- 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.
Syntax of $RT^\top$ Credentials

◆ Type I-IV as before

- **Type-5:** $A.R \leftarrow B_1.R_1 \odot \cdots \odot B_k.R_k$
  
  In which $R$ and the $R_i$’s are (single-element or manifold) role names. This credential means:
  
  $$members(A.R) \supseteq members(B_1.R_1 \odot \cdots \odot B_k.R_k) = \{ s_1 \cup \cdots \cup s_k \mid s_i \in members(B_i.R_i) \text{ for } 1 \leq i \leq k \}.$$  

  Here, when $s_i$ is an individual entity, say, $D$, it is implicitly converted to the singleton $\{D\}$.

- **Type-6:** $A.R \leftarrow B_1.R_1 \otimes \cdots \otimes B_k.R_k$
  
  This credential means:
  
  $$members(A.R) \supseteq members(B_1.R_1 \otimes \cdots \otimes B_k.R_k) = \{ s_1 \cup \cdots \cup s_k \mid (s_i \in members(B_i.R_i) \& s_i \cap s_j = \emptyset) \text{ for } 1 \leq i \neq j \leq k \}.$$
RT\top (Examples)

◆ Example 1: require a manager and an accountant
  • $K.\text{approval} \leftarrow K.\text{manager} \odot K.\text{accountant}$
  • $\text{members}(K.\text{approval}) \supseteq \{\{x,y\} \mid x \in K.\text{manager}, y \in K.\text{accountant}\}$

◆ Example 2: require a manager and a different accountant
  • $K.\text{approval} \leftarrow K.\text{manager} \otimes K.\text{accountant}$
  • $\text{members}(K.\text{approval}) \supseteq \{\{x,y\} \mid x \neq y, x \in K.\text{manager}, y \in K.\text{accountant}\}$
Example 3: require three different managers

- $\text{K.approval} \leftarrow \text{K.manager} \otimes \text{K.manager} \otimes \text{K.manager}$
- $\text{members(K.approval)} \supseteq \{x, y, z \mid x \neq y \neq z \in \text{K.manager}\}$
Delegation of Role Activations

- Delegation of the capacity to exercise one’s membership in a role
- Examples
  - Administrator logs in as ordinary user to read email
  - User delegated certain access rights by his manager when manager is out of town
- Different concept from delegation of authority to define a role
  - EPub lets CMU define student role
New Delegation Credential

B1 delegates to B2 the ability to act on behalf of D in D’s capacity as a member of A.R.
B1 delegates all role activations it has to B2
Example 8 In a small organization S0rg, any purchasing order has to be submitted and approved before it is placed. Any employee can submit a purchasing order. A manager can approve an order. A manager is also an employee; however, a manager cannot approve his own order. This can be represented as follows:

\[
\text{S0rg.place} \leftarrow \text{S0rg.submit} \times \text{S0rg.approve} \\
\text{S0rg.submit} \leftarrow \text{S0rg.employee} \\
\text{S0rg.approve} \leftarrow \text{S0rg.manager} \\
\text{S0rg.employee} \leftarrow \text{S0rg.manager}
\]

Suppose that both Alice and Bob and managers:

\[
\text{S0rg.manager} \leftarrow \text{Alice} \\
\text{S0rg.manager} \leftarrow \text{Bob}
\]

Alice can submit an order by issuing:

\[
\text{Alice as S0rg.employee} \rightarrow \text{order(orderid)}
\]

And Bob can approve it by issuing:

\[
\text{Bob as S0rg.approve} \rightarrow \text{order(orderid)}
\]

Then one can prove that

\[
\text{forRole(ReqID, \{Alice, Bob\}, S0rg.place)},
\]

where ReqID is the dummy principal representing order(orderID).

If Bob does not issue the above approval and Alice approves the order by also issuing:

\[
\text{Alice as S0rg.approve} \rightarrow \text{order(orderid)}
\]

One still cannot prove that

\[
\text{forRole(ReqID, \{Alice, Bob\}, S0rg.place)}.
\]
Example 9 A server $S$ authorizes `fileA` to be deleted if it is requested from a good workstation on behalf of a user. $S$ knows that `alice` is a user and trusts CA in certifying public keys for users. $S$ knows that `ws1` is a good workstation and trusts CA in certifying public keys for workstations. These are expressed in the following credentials:

$$S\text{.del}(\text{fileA}) \leftarrow S\text{.user} \otimes S\text{.goodWS}$$
$$S\text{.user} \leftarrow \text{CA}\text{.userCert}(\text{alice})$$
$$S\text{.goodWS} \leftarrow \text{CA}\text{.machineCert}(\text{ws1})$$

The following are credentials issued by CA:

$$\text{CA}\text{.userCert}(\text{alice}) \leftarrow K\text{.alice}$$
$$\text{CA}\text{.machineCert}(\text{ws1}) \leftarrow K\text{.ws1}$$
In the setting studied in [1], a workstation stores its private key in tamper-resistant firmware. When it boots, it generates a key pair for the operating system and issues a credential to delegate the activation of $S\.goodWS$ to the new key. When the user alice logs into a workstation $ws1$, a new process $p1$ is set up and a new key pair is generated. Through $p1$, alice then makes a request to the server $S$ to delete fileA. The process $p1$ sets up a secure channel $Ch$ to the server, and then sends the request through the channel. The following are delegation credentials that are needed:

\[
\begin{align*}
K_{\text{ws1}} & \xrightarrow{K_{\text{ws1 as S.goodWS}}} K_{\text{os1}} \\
K_{\text{os1}} & \xrightarrow{K_{\text{ws1 as S.goodWS}}} K_{p1} \\
K_{\text{alice}} & \xrightarrow{K_{\text{alice as S.user}}} K_{p1} \\
K_{p1} & \xrightarrow{K_{\text{ws1 as S.goodWS}}, K_{\text{alice as S.user}}} K_{\text{Ch}}
\end{align*}
\]

The request sent by $K_{\text{Ch}}$ to delete fileA on behalf of user alice working on a good workstation is represented as:

\[
K_{\text{ws1 as S.goodWS}}, K_{\text{alice as S.user}} \xrightarrow{K_{\text{Ch}}} \text{del(fileA)}.
\]

And this request should be authorized.
Discussion Question

What are the points of similarity and difference between access control logics and trust management languages?

- TM languages closer to implementation of distributed authorization systems?
- Underlying logic of TM languages is simpler (and hence tractable)?
- TM languages are less expressive than some logics, but is something essential missing from RT?
- Can RT express policies not expressible in some of the logics
Acknowledgement

A number of the RT slides are based on slides from Ninghui Li and John Mitchell
Thanks!

Questions?