WEB SECURITY
UNIT - III

Outline:

Database Security:

1. Recent Advances in Access Control
2. Access Control Models for XML
3. Database Issues in Trust Management and Trust Negotiation
4. Security in Data Warehouses and OLAP Systems

1. Recent Advances in Access Control

1.1 Introduction

- **Information security goals**:
  - **Secrecy**: protection against unauthorized disclosure
  - **Integrity**: protection against unauthorized or improper modifications
  - **Availability**: ensuring access to legitimate users (no denial-of-service)
- **Access control service**:
  - Mediates every request to resources and data
  - Determines whether access is **granted or denied**
- **Three abstractions of access control**:
  1. **Access control policy**
     - High-level rules deciding authorization
  2. **Access control model**
     - Formal representation of the policy
  3. **Access control mechanism**
     - Enforces the model
- **Advantages of separating policy, model, and mechanism**:
  - Policies discussed independently of implementation
  - Comparison of different policies and mechanisms
  - Mechanisms can enforce multiple policies
  - Policy changes do not require mechanism changes
  - Formal verification of security properties at the model level
- **Challenges in access control policy definition**:
  - Must be **simple** (ease of management)
  - Must be **expressive** (flexible protection requirements)
- **Key features required in modern access control systems**:
  - **Policy combination**: support multiple authorities and scalable composition
  - **Anonymity**: access decisions based on attributes, not real identity
  - **Data outsourcing**: selective access to remotely stored data
- **Chapter organization overview**:
  - Section 2: Classical access control models
  - Section 3: Access control in open environments
  - Section 4: Policy composition
  - Section 5: Selective access in outsourced scenarios

- Section 6: Conclusions

Fig. 1. An example of access matrix

- Illustrates subjects, objects, and actions
- Shows how authorizations are represented in matrix form
- Example: user *Ann* can read and write *Document1*

	Document1	Document2	Program1	Program2
Ann	read, write	read	execute	
Bob	read	read	read, execute	
Carol		read, write		read, execute
David			read, write, execute	read, write, execute

Fig. 1. An example of access matrix

1.2 Classical Access Control Models

- Three main classes:
  - **Discretionary Access Control (DAC)** – based on user identity
  - **Mandatory Access Control (MAC)** – based on centrally mandated rules
  - **Role-Based Access Control (RBAC)** – based on roles assigned to users

1.2.1 Discretionary Access Control

- **Authorization-based model**
  - Authorization = (s, o, a)
    - *s*: subject (user)
    - *o*: object
    - *a*: action
- **Access Matrix Model**:
  - Sets: Subjects (S), Objects (O), Actions (A)
  - Matrix A of size |S| × |O|
  - Entry A[s,o] lists allowed actions
- **Fig. 1 reference**:
  - Demonstrates access matrix with specific permissions (e.g., Ann → read/write Document1)

Access Matrix Implementation Mechanisms

- **Authorization table**
  - Stores triples (user, action, object)
- **Access Control List (ACL)**
  - Stored by column (per object)
  - Each object lists users and allowed actions
- **Capability**
  - Stored by row (per subject)
  - Each subject lists accessible objects and actions

Fig. 2. Access matrix implementation mechanisms

- Shows:
  - Authorization table
  - Access control lists

- Capability lists
- All derived from the same access matrix in Fig. 1

User	Action	Object
Ann	read	Document1
Ann	write	Document1
Ann	read	Document2
Ann	execute	Program1
Bob	read	Document1
Bob	read	Document2
Bob	read	Program1
Bob	execute	Program1
Carol	read	Document2
Carol	write	Document2
Carol	execute	Program2
David	read	Program1
David	write	Program1
David	execute	Program1
David	read	Program2
David	write	Program2
David	execute	Program2

(a)

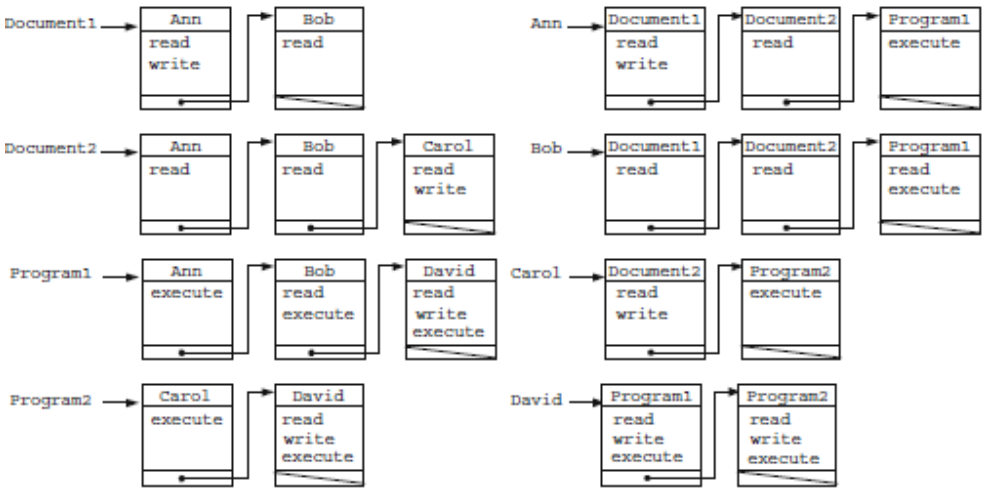


Fig. 2. Access matrix implementation mechanisms

Extensions of Discretionary Access Control

Conditions

- Authorization validity depends on constraints:
  - Content-dependent
  - System-dependent
  - History-dependent

Abstractions

- Support for:

    - User groups
    - Object classes
    - Hierarchical organization
- Authorizations propagate according to policies

Fig. 3. An example of user-group hierarchy

- Illustrates hierarchical groups
- Example: authorization for *Nurse* group applies to *Bob* and *Carol*

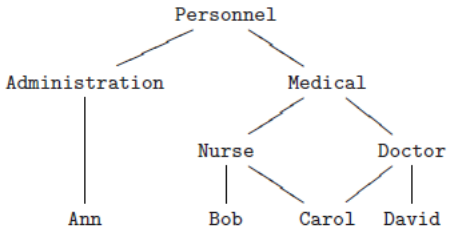


Fig. 3. An example of user-group hierarchy

Exceptions

- Use of **positive and negative authorizations**
- Needed to override group-level permissions for specific users
- **Problems introduced**:
    - **Inconsistency**: conflicting authorizations
    - **Incompleteness**: no authorization applies
- **Default policies**:
    - Open policy → allow access
    - Closed policy → deny access (more common)

Conflict Resolution Policies

- No conflict
- Denials take precedence
- Permissions take precedence
- Nothing takes precedence
- Most specific takes precedence
- Most specific along a path takes precedence

Fig. 4. An example of security (a) and integrity (b) lattices

- Shows lattice structures for:
    - Security levels (secrecy)
    - Integrity levels
- Used in mandatory access control
- **Limitations of DAC**:
    - Vulnerable to Trojan horses
    - No distinction between users and subjects
    - Cannot control information flow

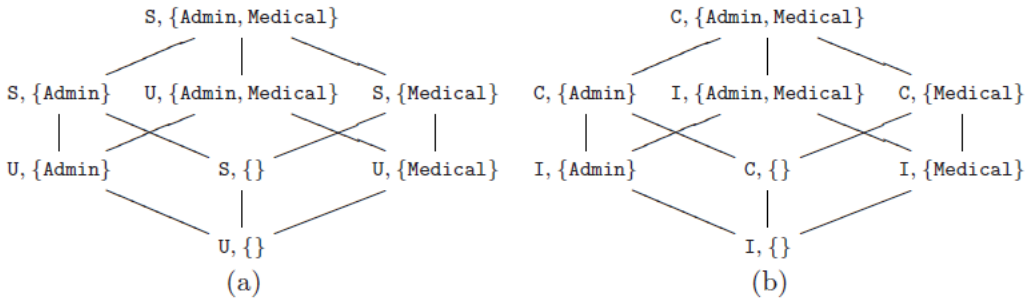


Fig. 4. An example of security (a) and integrity (b) lattices

1.2.2 Mandatory Access Control

- Enforced by a **central authority**
- Based on **access classes**:
  - Security level
  - Categories
- Dominance relation (≥) forms a **lattice**

Secrecy-Based Mandatory Policy

- Goal: protect confidentiality
- Principles:
  - **No-Read-Up**
  - **No-Write-Down**
- **Fig. 4(a)**:
  - Security lattice with levels *Secret* and *Unclassified*
  - Categories: Admin, Medical
- Prevents information flow from high to low levels
- Trusted processes may allow exceptions (sanitization)

Integrity-Based Mandatory Policy

- Goal: protect data integrity
- Principles:
  - **No-Read-Down**
  - **No-Write-Up**
- **Fig. 4(b)**:
  - Integrity lattice with levels *Crucial* and *Important*
- Prevents low-integrity data influencing high-integrity objects
- Limited to information-flow integrity
- **MAC limitations**:
  - Cannot prevent covert channels

1.2.3 Role-Based Access Control

- Access based on **roles**, not identity
- Roles = sets of privileges
- Users activate roles at login
- **Key properties**:
  - Role hierarchies
  - Multiple roles per user

    - Multiple users per role
- **Difference between roles and groups**:
    - Groups = users
    - Roles = privileges
- **Advantages**:
    - Suitable for commercial environments
    - Reflects organizational structure

1.3 Credential-Based Access Control

- Designed for **open and dynamic environments**
- Clients and servers may be unknown
- Trust management replaces traditional authentication + authorization
- Early systems:
    - PolicyMaker
    - KeyNote
- Limitations:
    - Authorizations bound to public keys
    - Identity leakage
- **Digital certificates**:
    - Certified attributes or properties
    - Used to prove access eligibility

Key Issues in Credential-Based Access Control

- Ontologies
- Client-side and server-side restrictions
- Credential-based access control rules
- Access control evaluation outcomes
- Trust negotiation strategies

1.3.1 Overview of Trust Negotiation Strategies

- Iterative exchange of:
    - Policies
    - Credentials
- **Basic negotiation steps**:
    1. Client requests resource
    2. Server checks credentials
    3. Server requests missing credentials
    4. Client responds
    5. Access decision
- **Privacy issue**: excessive disclosure
- **Gradual trust establishment**
- **PRUNES strategy**
    - Uses negotiation search tree
    - Prunes unnecessary credential evaluations
- **Disclosure Tree Strategy (DTS)**

- **UniPro framework**
- **ATNAC approach**
    - Uses suspicion levels

1.3.2 Overview of a Credential-Based Access Control Framework

- Proposed by Bonatti and Samarati
- Components:
    - Access control model
    - Language
    - Policy filtering mechanism
- **Client–server negotiation**
- Portfolios:
    - Credentials
    - Declarations
- Rules:
    - Service accessibility rules
    - Portfolio disclosure rules

Fig. 5. Client-server negotiation

- Shows interaction:
    - Prerequisites checked first
    - Requisites disclosed later
- Minimizes credential and policy disclosure

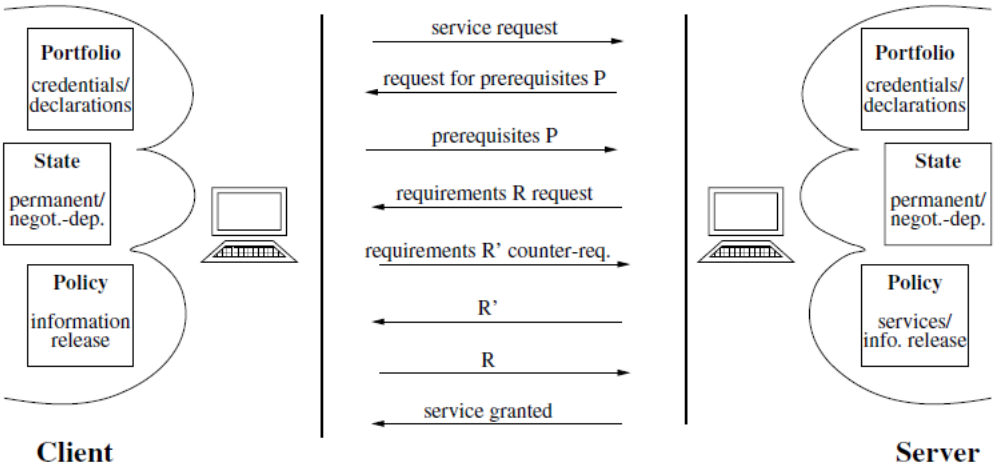


Fig. 5. Client-server negotiation

1.4 Policy Composition

- Combines policies from multiple authorities
- Important in decentralized administration

Desiderata for Policy Composition Frameworks

- Heterogeneous policy support
- Unknown policy support
- Controlled interference
- Expressiveness

- Multiple abstraction levels
- Formal semantics

1.4.1 Overview of Policy Composition Solutions

- Approaches include:
  - Meta-policies
  - Policy algebra
  - Policy combiners
- **Policy algebra [36]**:
  - Policies as sets of (s,o,a) triples
  - Operators: +, &, −, ∗, ∧, o, τ
- Supports:
  - Exceptions
  - Propagation
  - Black-box policies
- Formal semantics allow security proofs

1.5 Access Control Through Encryption

- Motivated by **data outsourcing**
- Service provider not fully trusted
- Access control enforced by **selective encryption**

1.5.1 Overview of Database Outsourcing Solutions

- Access matrix used to represent policy

Fig. 6. An example of binary access matrix

- Users: A, B, C, D
- Resources: r1–r4
- Binary values indicate read permission
- **Selective encryption**
- Key derivation hierarchies:
  - Chain
  - Tree
  - DAG

	r_1	r_2	r_3	r_4
A	1	1	0	1
B	1	1	0	0
C	1	0	1	0
D	0	1	1	0

Fig. 6. An example of binary access matrix

Fig. 7. An example of UH (a) and RH (b)

- UH: user-based hierarchy
- RH: resource-based hierarchy
- Enforces access control through key derivation

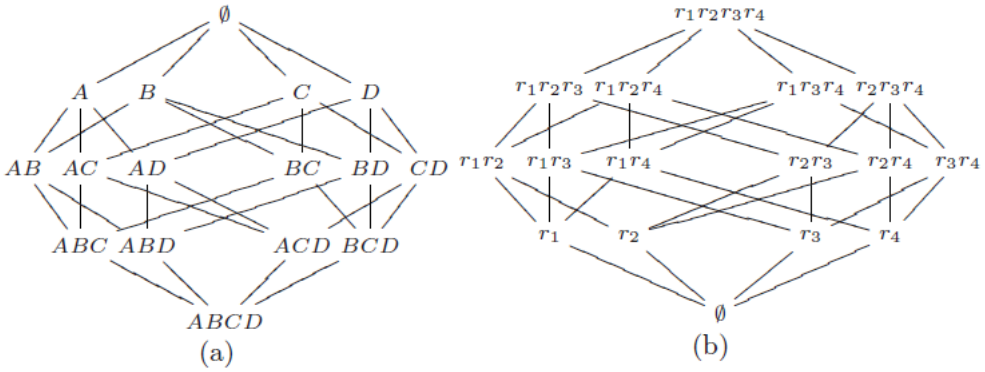


Fig. 7. An example of UH (a) and RH (b)

Fig. 8. An example of transformed user hierarchy

- Reduced hierarchy
- Removes unnecessary vertices
- Improves efficiency

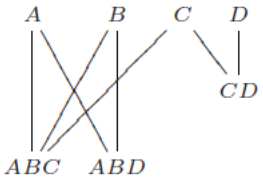


Fig. 8. An example of transformed user hierarchy.

1.5.2 Overview of XML Document Outsourcing Solutions

- XML structure exploited for access control
- Selective encryption applied to XML nodes
- Metadata and association constraints used

1.6 Conclusions

- Reviewed recent advances in access control
- Addressed:
  - Open environments
  - Policy composition
  - Selective encryption
- Highlighted emerging models and solutions for modern systems

2. Access Control Models for XML

2.1 Introduction

- Web-based information exchange is **rapidly increasing**, especially sensitive data such as:
  - E-commerce (EC) transactions
  - Financial and personal information
- **Security problem with HTML**:
  - No separation between **structure and layout**
  - Markup focuses on presentation, not semantics

    - Browser-dependent design
    - Poor support for fine-grained security
- **XML (eXtensible Markup Language)**:
    - Standardized by **W3C**
    - Semantics-aware markup language
    - Separates **content, structure, and presentation**
    - Extensible: users define custom tags and attributes
    - Subset of **SGML**, richer than HTML
- **XSL (XML Stylesheet Language)**:
    - Separate W3C standard
    - Handles presentation and rendering of XML data
- **Motivation for XML access control**:
    - XML documents may replace relational databases
    - XML documents may contain **both public and sensitive data**
    - File-level protection is insufficient
    - **Fine-grained access control** is required (element/attribute level)
- **Chapter organization**:
    - Section 2: Preliminary XML concepts
    - Section 3: XML access control requirements
    - Section 4: XML access control models
    - Section 5: Conclusions

2.2 Preliminary Concepts

- XML is used for **semi-structured information**
- Understanding XML structure is essential for defining access control

2.2.1 Well-Formed and Valid XML Documents

XML Structure

- XML document consists of:
    - **Elements** (start/end tags or empty tags)
    - **Attributes** (properties inside start tags)
- Example tags:
    - <request> ... </request>
    - <request/>
    - <request number="10">

Well-Formed XML Document

An XML document is **well-formed** if it satisfies W3C syntax rules:

- Starts with XML prologue: <?xml version="1.0"?>
- Has exactly one **root element**
- All tags are properly opened and closed
- Elements are **properly nested**
- Tags are **case-sensitive**
- Attribute values are **quoted**

XML Languages and Schemas

- An **XML language**:

    - Defined by syntax + semantics
- A **schema**:
    - Formal definition of XML structure
    - Defines allowed elements, attributes, and relationships
- Common schema languages:
    - **DTD (Document Type Definition)**
    - **XML Schema**

Document Type Definition (DTD)

- Can be **internal or external**
- Not written in XML syntax

DTD Components

- **Element declaration**:
  <!ELEMENT element_name content>
- Possible content models:
    - #PCDATA (text)
    - Empty
    - Any
    - Sub-elements
    - Mixed content
- **Cardinality operators**:
    - * → zero or more
    - + → one or more
    - ? → zero or one
    - No operator → exactly one
- **Attribute declaration**:
  <!ATTLIST element_name attribute_def>
- Attribute constraints:
    - #REQUIRED
    - #IMPLIED
    - #FIXED
- **Valid XML document**:
    - Well-formed
    - Conforms to its DTD

XML Schema

- XML Schema is itself an **XML document**
- Advantages over DTD:
    - Supports **data types**
    - Supports **namespaces**
    - More expressive and extensible
- Element types:
    - **Simple types** (string, decimal, float, etc.)
    - **Complex types** (elements + attributes + order + cardinality)
- Attribute properties:
    - Required / Optional

- Default or Fixed values

Example 1 (DTD and XML Document)

- **Figure 1(a)**:
  - Shows a DTD for bank account operations
  - Root element: account operation
  - Mandatory attributes: bankAccN, id
  - Sub-elements: request, operation
- **Figure 1(b)**:
  - XML document valid with respect to the DTD in Fig. 1(a)

```
<!DOCTYPE record[
  <!ELEMENT account_operation
        (request, operation+)>
  <!ATTLIST account_operation
        bankAccN CDATA #REQUIRED
        id CDATA #REQUIRED>
  <!ELEMENT request
        (date,means?,notes?)>
  <!ATTLIST request number CDATA #REQUIRED>
  <!ELEMENT operation
        (type, amount, recipient, (notes|value)?)>
  <!ELEMENT date (#PCDATA)>
  <!ELEMENT means (#PCDATA)>
  <!ELEMENT notes (#PCDATA)>
  <!ELEMENT type (#PCDATA)>
  <!ELEMENT amount (#PCDATA)>
  <!ELEMENT recipient (#PCDATA)>
  <!ELEMENT value (#PCDATA)>
]>
```

(a)

```
<?xml version="1.0" ?>
<!DOCTYPE record SYSTEM "record.dtd">
  <account_operation
     bankAccN="0012" id="00025">
     <request number="10">
        <date> 04-20-2007 </date>
        <means> Internet </means>
        <notes> urgent </notes>
     </request>
     <operation>
        <type> bank transfer </type>
        <amount> $ 1,500 </amount>
        <recipient> 0023 </recipient>
        <notes> Invoice 315 of 03-31-2007
        </notes>
     </operation>
  </account_operation>
```

(b)

Fig. 1. An example of DTD (a) and a corresponding valid XML document (b)

Tree Representation of DTD and XML Documents

- **DTD tree**:
  - Elements → ovals
  - Attributes → rectangles
  - Cardinalities shown on arcs
  - Choice (|) represented by branching
  - Sequence order preserved vertically
- **XML document tree**:
  - Nodes for elements, attributes, and values
  - Arcs represent containment
  - No labels on arcs
- **Figure 2**:
  - Graphical representation of both DTD and XML document from Fig. 1

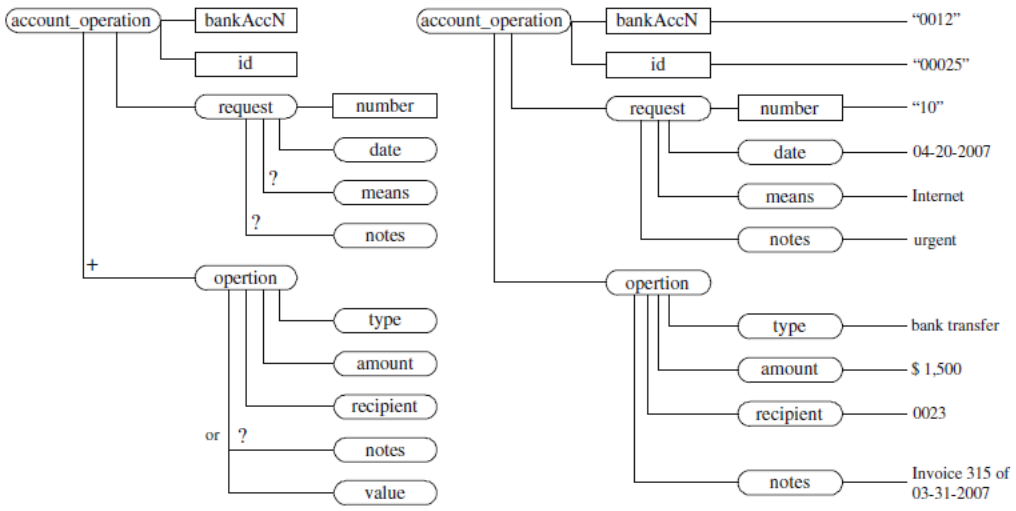


Fig. 2. An example of graphical representation of DTD and XML document

2.2.2 Elements and Attributes Identification

- XML access control models identify protected objects using **XPath**

XPath Language

- **Path expressions**:
  l1/l2/.../ln
- Attribute access:
  - Prefixed with @
- Types of paths:
  - **Absolute** (/)
  - **Relative**
- Operators:
  - . current node
  - .. parent node
  - // arbitrary descendant
- Conditions:
  - Written in [ ]
  - Combined using and, or
  - Operate on:
    - Element text
    - Attribute names/values
- Functions:
  - last()
  - position()

XQuery

- Built on XPath
- Uses **FLWOR** expressions:
  - FOR
  - LET

- WHERE
- ORDER BY
- RETURN

Example 2 (XPath and XQuery)

- Demonstrates:
  - Absolute paths
  - Attribute access
  - // operator
  - Conditional predicates
- `/account_operation/operation`: returns the content of the `operation` element, child of `account_operation`;
- `/account_operation/@bankAccN`: returns attribute `bankAccN` of element `account_operation`;
- `/account_operation//notes`: returns the content of the `notes` elements, anywhere in the subtree rooted at `account_operation`; in this case, it returns both `/account_operation/request/notes` and `/account_operation/operation/notes`;
- `/account_operation/operation[./type=`"bank transfer"`]`: returns the content of the `operation` element, child of `account_operation`, only if the `type` element, child of `operation`, has value equal to "bank transfer".

- XQuery extracts:
  - Bank transfer operations
  - Amount, recipient, and notes

```
<BankTransf>
{ FOR    $r in document("update_account")/account_operation
  WHERE  $r/operation/type="bank transfer"
  RETURN $r/operation/amount, $r/operation/recipient, $r//notes
}
</BankTransf>
```

- Uses document from **Fig. 1(b)**

2.3 XML Access Control Requirements

- Traditional access control models are insufficient for XML
- XML models differ based on:
  - Subjects
  - Objects
  - Actions

Subject

- Identified by:
  - User identity
  - Network location (IP or symbolic name)
- Extended subject definitions:
  - User groups
  - Location patterns (*)
  - Roles
- Subjects organized in **hierarchies**

- **Figure 3**:
  - Example of a user-group hierarchy
  - Represented as a directed acyclic graph (DAG)
- Recent models support **credential-based access control**

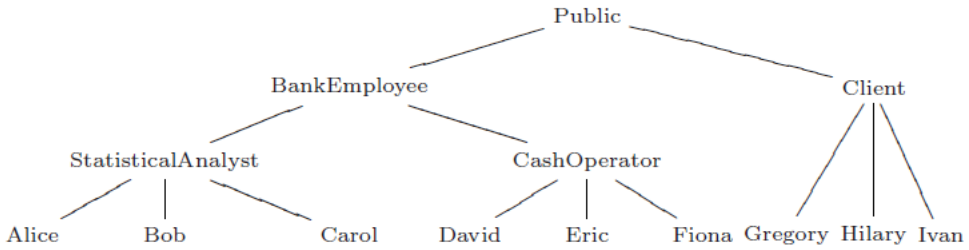


Fig. 3. An example of user-group hierarchy

Object Granularity

- Objects identified via **XPath expressions**
- Fine-grained protection:
  - Document
  - Element
  - Attribute
- XPath restrictions in some models:
  - Limited // usage
  - No predicates after //

Action

- Most models support **read** only
- Write operations are complex due to:
  - DTD constraints
  - Hidden elements/attributes
- Write actions:
  - Insert
  - Update
  - Delete
- Additional actions:
  - **read** (content)
  - **position** (existence without content)

Support for Fine and Coarse Authorizations

- Instance-level authorizations
- Schema-level (DTD-based) authorizations
- Organization-wide vs domain-specific policies

Propagation Policy

- Local authorizations:
  - Apply to attributes only
- Recursive authorizations:
  - Apply to entire subtree

- Propagation strategies:
  - No propagation
  - No overriding
  - Most specific overrides
  - Path overrides
- Applies to object and subject hierarchies

Support of Exceptions

- Use of:
  - Positive authorizations (permissions)
  - Negative authorizations (denials)
- Conflict resolution:
  - Typically **denials take precedence**
- Incompleteness handled by:
  - Default open policy
  - Default closed policy

2.4 XML Access Control Models

2.4.1 Fine Grained XML Access Control System

- Proposed by **Damiani et al. [9]**
- Extends earlier XML access control proposals

Authorizations Specification

Definition 1 (Access Authorization)

Authorization = ⟨subject, object, action, sign, type⟩

- **Subject**:
  - User-id
  - IP address
  - Symbolic address
  - Groups and location patterns
  - Combined into subject hierarchy (AS)
- **Object**:
  - URI
  - URI + XPath expression
- **Action**:
  - read (write extensions possible)
- **Sign**:
  - + permission
  - − denial
- **Type**:
  - Controls propagation and overriding
  - Eight types: LDH, RDH, L, R, LD, RD, LS, RS
  - Priority shown in **Table 1**

Table 1. Authorization types

Level/Strength	Propagation	
	Local	Recursive
Instance	L	R
Instance (soft statement)	LS	RS
DTD	LD	RD
DTD (hard statement)	LDH	RDH

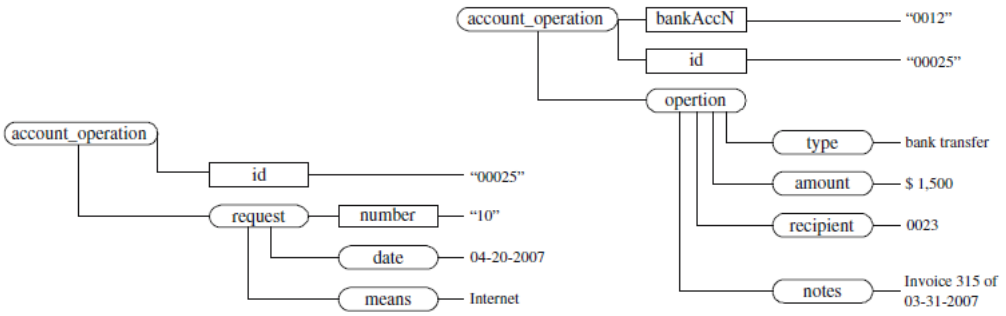
Access Control Enforcement

- Enforcement via **document view generation**

Steps:

1. Authorization retrieval
2. Initial labeling of XML tree nodes
3. Label propagation
4. View computation

- **Uses**:
  - Conflict resolution
  - Closed policy
  - DTD loosening to preserve validity
- **Figure 4**:
  - Shows different document views returned to users



(a) Alice's view (b) David's view

Fig. 4. Examples of views

2.4.2 Kudo et al. Static Analysis

- Avoids expensive runtime evaluation
- Combines **static analysis + runtime analysis**

Authorization Specification

- Authorization = (subject, action, object)
- Subjects:
  - User-id
  - Group
  - Role
- No subject hierarchies
- XPath without functions

- Read-only access
- Denials take precedence

Static Analysis Using Automata

- Uses **Non-deterministic Finite Automata (NFA)**

Definition 2 (NFA):

- Alphabet Ω
- States Q
- Initial states Qinit
- Final states Qfin
- Transition function δ

Static Analysis Steps

1. Build schema automaton (MG)
2. Build access control automata (MΓ, MΓ)
3. Translate XQuery paths into regular expressions
4. Compare automata to classify access as:
    - Always granted
    - Always denied
    - Statically indeterminate
- Improves efficiency by:
    - Query rewriting
    - Early pruning

Example 4

- **Uses**:
    - DTD and XML from Fig. 1
    - Groups: BankEmployee, StatisticalAnalyst, Client
- **Demonstrates**:
    - Schema automaton construction
    - Access control automata
    - Query classification

Consider a set of authorizations stating that the members of the BankEmployee group can access the whole content of the account operation element, members of the statisticalAnalyst group can access the content of the account operation element but the notes elements, and each client can access the account operation elements about their bank account. Formally, these authorizations can be expressed as follows.

- group: `BankEmployee`, `/account_operation`, + `read`, recursive
- group: `StatisticalAnalyst`, `/account_operation`, + `read`, recursive
- group: `StatisticalAnalyst`, `//notes`, - `read`, recursive
- group: `Client`, `/account_operation[./@bankAccN=$userAcc]`, + `read`, recursive

We first define the schema automaton corresponding to the considered DTD. It is first necessary to define two sets of symbols, representing elements and attributes, respectively.

Σ^E= {`account_operation`, `request`, `operation`, `date`, `means`, `notes`, `type`, `amount`, `recipient`, `value`}
Σ^A= {`@bankAccN`, `@Id`, `@number`}

Given Σ^E and Σ^A, it is now possible to define the schema automaton M^G as follows.

- Ω=$\Sigma^E \cup \Sigma^A$
- Q={Account_Operation, Request, Operation, Date, Means, Notes, Type, Amount, Recipient, Value}$\cup\{q^{init}\}\cup\{q^{fin}\}$
- Q^{init}=$\{q^{init}\}$
- Q^{fin}={Date, Means, Notes, Type, Amount, Recipient, Value}$\cup\{q^{fin}\}$
- $\delta(q^{init}$,`account_operation`)=Account_Operation
  δ(Account_Operation,`request`)=Request
  δ(Account_Operation,`operation`)=Operation
  δ(Request,`date`)=Date
  δ(Request,`means`)=Means
  δ(Request,`notes`)=Notes
  δ(Operation,`type`)=Type
  δ(Operation,`amount`)=Amount;

δ(Operation,`recipient`)=Recipient
δ(Operation,`notes`)=Notes;
δ(Operation,`value`)=Value
δ(Account_Operation,`@bankAccN`)=q^{fin}
δ(Account_Operation,`@Id`)=q^{fin}
δ(Request,`@number`)=q^{fin}

The schema automaton defined accepts the same paths allowed by the considered DTD. Specifically, $L(M^G)$ is equal to: `/account_operation`, `/account_operation/@Id`, `/account_operation/@bankAccN`, `/account_operation/request`, `/account_operation/request/@number`, `/account_operation/request/date`, `/account_operation/request/means`, `/account_operation/request/notes`, `/account_operation/operation`, `/account_operation/operation/type`, `/account_operation/operation/amount`, `/account_operation/operation/recipient`, `/account_operation/operation/notes`, `/account_operation/operation/value`.

The second step of the static analysis method consists in building the access control automata $\underline{M^\Gamma}$ and $\overline{M^\Gamma}$, for each of the three groups of users considered. For the sake of simplicity, we represent only the language of the automaton.

BankEmployee $L(M^\Gamma)$={`account_operation`}$\cdot(\Sigma^E)^* \cdot (\Sigma^A \cup \{\epsilon\})$
StatisticalAnalyst $L(M^\Gamma)$={`account_operation`}$\cdot(\Sigma^E)^* \cdot (\Sigma^A \cup \{\epsilon\})\setminus$ {`notes`}$\cdot(\Sigma^E)^* \cdot (\Sigma^A \cup \{\epsilon\})$
Client $L(\underline{M^\Gamma})=\emptyset \cdot (\Sigma^E)^* \cdot (\Sigma^A \cup \{\epsilon\})$; $L(\overline{M^\Gamma})$={account_operation}$\cdot(\Sigma^E)^* \cdot (\Sigma^A \cup \{\epsilon\})$

Here, $\cdot$ is the concatenation operator, $\setminus$ is the set difference operator, ϵ is the nil character, and $(\Sigma^E)^*$ represents any string in Σ^E.

Consider now the XQuery expression introduced in Example 2. The corresponding XPath expressions, classified on the basis of the clause they are represented in, are:

`FOR, LET, ORDER BY ,WHERE`: `/account_operation`;
`/account_operation/operation/type`
`RETURN`: `account_operation/operation/amount`;
`account_operation/operation/recipient`;
`account_operation//notes`

Here `record//notes` implies both `record/request/notes` and `record/operation/notes`.

On the basis of the static analysis, it is possible to classify the requests submitted by users. As an example, consider the following requests.

- `BankEmployee` requests `/account_operation/operation/type`: the request is *always granted*;
- `StatisticalAnalyst` requests `/account_operation//notes`: the request is *always denied*;
- `Client` requests `/account_operation/operation/amount`: the request is *statically indeterminate*.

2.4.3 Other Approaches

- Provisional authorizations
- Access-Condition Tables (ACT)
- Policy Matching Trees (PMT)
- Function-based access control
- Compressed Accessibility Maps (CAM)
- Query rewriting (QFilter)
- Security views
- XSLT-based view generation
- Relational storage approaches
- Credential-based access control
- SQL-like and XQuery-based authorization
- Relationship-based authorizations
- RDF-based policies
- RBAC for XML
- Client-side access control
- Selective encryption for outsourced XML data

2.5 Conclusions

- XML is central to modern information systems
- Fine-grained XML access control is essential
- Research has produced:
    - Runtime models
    - Static analysis models
    - View-based models
    - Encryption-based models
- These models form a strong basis for **standardization**
- XML access control is expected to become a **core security mechanism** in future systems