Pragmatic XML Access Control
Enforcement Mechanism
via Query Filtering and
its Applications

Dongwon Lee
Dec. 20, 2004
Penn State University

Credits

- Joint work with
  - Padmapriya Ayyagari
  - Wang-Chien Lee
  - Cathy Li
  - Peng Liu
  - Robert Luo
  - Prasenjit Mitra
- From
  - XSym03, SDM04, CIKM04, etc
XML Research (IMHO)

- **1st Generation**: XML Model, QL
  - Semi-structured/XML models and QLs
  - Eg, OEM, StrudQL, LOREL, …
- **2nd Generation**: XML in RDBMS vs. Native XML
  - XML-2-Relational Schema/Data Conversion
  - Relational-2-XML Schema/Data Publishing
  - XML encoding, indexing, storage, …
  - Eg, STORED, Inlining, XRel, XTRACT, TIMBER, …
- **3rd Generation**: Advanced Features of XML
  - Building advanced features of XML databases

In particular, in 3rd Generation,
- Building advanced features “XYZ” of XML using either Vanilla XDBMS or RDBMS
  - **Vanilla XDBMS**: Let middleware handle the feature XYZ so that underlying XDBMS can process the outputs
  - **RDBMS**: Convert XML-XYZ to Relational-XYZ
Outline

- Introduction
- Motivation
- Framework
- QFilter Details
- QFilter Validation
- QFilter Applications
- Conclusion

Introduction

- **Relational Access Control** ensures only authorized users can access only authorized portion of relational data
  - Role-based security model (user << role)
  - GRANT / REVOKE
  - Can control table-level or column-level access
- Eg,
  - GRANT SELECT ON Foo to dlee
  - GRANT INSERT ON Bar(A, B) to admin
Introduction

- XML Access Control ensures only authorized users can access only authorized portion of XML data
- Emerging techniques and standards for XML Access Control
  - Model and semantics
  - Enforcement ⇔ Our Focus

Introduction

- Often, XML Access Control is represented as 5-tuple ACR
  - \{subject, object, action, +/-, RC/LC\}
  - Object is expressed by XPath

- Eg, “Manager can read employee’s project-related information, but not their salaries”
  - (manager, //employee/proj, read, +, RC)
  - (manager, //employee/@salary, read, -, LC)

ACR (Access Control Rule)

ACT (Access Control Table)
Introduction

- Relational AC vs. XML AC Comparison
  - Relational Model has table and column
  - XML has element and attribute
- In conversion, a node in an XML tree can be mapped to (in RDBMS)
  - Table
  - Column
  - Tuple
  - Cell (column X, tuple Y)
- Table and column level security is supported in commercial RDBMS, but not the other two cases

```
A <B+, C>, B <D, E> ⇔ A'(C', FK), B'(D', E')
```

Control access to user “john”

- Table: /A/B ⇔ B'
- Column: /A/C ⇔ C'
- Cell: /A/B[D>5]/E ⇔ B'(E') with D'>5
Outline

- Introduction
- Motivation
- Framework
- QFilter Details
- QFilter Validation
- QFilter Applications
- Conclusion

Motivation

- Popular solution to enforce XML Access Control is to use “Materialized Views” (eg, [Dimiani, 2002; Bertino, 2002; Yu, 2002])
  - Construct a view per role/user
  - Once view is constructed, no more security check
  - Space cost / Maintenance issue
- Others rely on the support of security feature of XML database [Cho; 2002]
  - No XML databases have such features yet
Motivation

- Neither approach is fully satisfactory

- Our Approach is
  - Framework-based: devise and compare various approaches
  - Practical solution: can work with off-the-shelf XML database engine (ie, Vanilla XDBMS)

Outline

- Introduction
- Motivation
- Framework
- QFilter Details
- QFilter Validation
- QFilter Applications
- Conclusion
Framework

- XML data (document).
- Stored in XML database

- Describes the information that users want in XPath
- Query has the same security role as the user who issues it

- Each ACR describes the access control policy of a role.
- Objects are specified in XPath

Scenarios

Scenario 1
- Query
- ACR
- Data

Scenario 2
- Query
- ACR
- Data

Scenario 3
- Query
- ACR
- Data

Scenario 4
- Query
- Data
- ACR

Scenario 5
- Query
- Data
- ACR
Post-Processing

- Intermediate answers are computed as usual
- Then, ACR prunes out unsafe answers

- Suitable for role-based data delivery model, where the same data is delivered to different roles.
- Can be implemented by XML data filtering package (e.g., YFilter [Diao, 2003])
Pre-Processing

- Query Q is modified to a safe one Q’ by ACR
- Then, Q’ is processed by regular XML engine
- “Modified” can be implemented by many ways
  - Primitive: \( Q \cap ACR \Leftrightarrow Q' \)
  - QFilter: \( \text{QFilter}(Q, ACR) \Leftrightarrow Q' \)

Pre-Processing: Primitive

- View Query and ACR as two constraints to satisfy
  - \( Q \) and + ACR: \( Q \cap ACR \Leftrightarrow Q' \)
  - \( Q \) and – ACR: \( Q - ACR \Leftrightarrow Q' \)
- Then, Q’ is passed to regular XML engine that can handle XPath with set operator
  - Easy to implement
  - Performance is highly dependent on the capability of underlying XML engine (how it handles set operators, etc)
Pre-Processing: Primitive

- $Q:/dept[\text{name}='HR']/\text{budget}$
  - Manager John wants to access HR dept’s budget
- ACR (for all manager/read)
  - R1: $/dept/\text{salary}$, +, LC
  - R2: $/dept/\text{south}$, +, RC
  - R3: $/dept[\text{year}=2003]/\text{budget}$, -, LC
- $Q' \Leftrightarrow Q \cap (+ \text{ rules}) - (- \text{ rules})$
  - $\Leftrightarrow Q \cap (R1 \cup R2) - R3$
  - $\Leftrightarrow /dept[\text{name}='HR']/\text{budget} \cap (/dept/\text{salary} \cup /dept/\text{south}) - /dept[\text{year}=2003]/\text{budget}$

Outline

- Introduction
- Motivation
- Framework
- QFilter Details
- QFilter Validation
- QFilter Applications
- Conclusion
Pre-Processing: QFilter

- Primitive Pre-Processing satisfies our two goals
  - Non-view based
  - Independent on underlying XML engine

- But, the re-written query Q’ is not the most efficient form

Pre-Processing: QFilter

- Idea of QFilter: Improve Q’ further for better performance
- **Contained Case** (Q ∩ ACR ⇔ Q)
  - /dept[year<2004]/budget ∩ /dept/*
  - ⇔ /dept[year<2004]/budget

- **Disjoint Case** (Q ∩ ACR ⇔ {})  
  - /dept[year<2004]/budget ∩ /dept[year>2005]/budget ⇔ {}  

- **Overlapping Case** (Q ∩ ACR ⇔ Q’)
  - /dept[year<2004]/budget ∩ //south/budget
  - ⇔ /dept/[year<2004]/south/budget
Pre-Processing: QFilter

- QFilter captures ACR as NFA (Non-deterministic Finite Automata)
- Given Q, quickly determine if it is contained, disjoint or overlapping by traversing NFA
- When it’s overlapping case, Q’ is generated

- Cannot handle general case of XPath
- XPath containment is:
  - /, //, [], *: P [Wood, 2001] ⇔ QFilter supports this
  - =, NOT, <: undecidable [Neven, 2003]

QFilter Example

- R1: /site/categories/*
- R2: /site/regions/*/item/location
- R3: /site/regions/*/item/quantity
- R4: /site/regions/*/item/name
- R5: /site/regions/*/item/description
- R6: /site/people/person/name
- R7: /site/people/person/address/*
- R8: /site/people/person/emailaddress
Q: /site/categories/NW/item

Q AS IS!

Q: /site/top//item

REJECT!
Q: /site/*/person/name

Re-written to Q’!

/site/categories/person/name ∪ /site/people/person/name

QFilter with Predicate Handling

R9:/site/regions/*/item[description]/name
Q: /site/regions/item[quantity]/name

Q'=/site/regions/item[quantity][description]/name

QFilter Discussion

Theorem: Q' generated by QFilter never returns unauthorized answers to unauthorized users for XPath with /, //, *, []

- QFilter construction: $O(|ACR|)$
- QFilter execution
  - No *: $O(|Q|)$
  - *: $O(|NFA|)$
  - //: $O(|Q| \times \Pi \text{ child for i-th } //)$
  - Worst case only occurs for a query “/*/*///*.../*”
Deep-Set Operator

- QFilter often uses “set operators”
- However, standard semantics of XQuery/XPath set operators are too limited
- Eg,
  - /dept[year<2004]/budget \(\cap\) //south/budget \(\iff\)
    /dept/[year<2004]/south/budget
  - /dept[year<2004]/budget \(\cap\) //south \(\iff\) \{

Deep-Set Example

```
/x/a[e]  1
  2
  3  b
    8  e
  4  c
  5  d  //d[f]
```

```
1  x
  a  9
    2  a
      3  b
        8  e
      4  c
    5  d  //d[f]
      6  f
        7  g
    10  b
5  d
    11  d
      12  g
```
\( /x/a[e] \ \text{UNION} \ //d[f] \)

**UNION**

- Tree 1
  - Node 2
    - Node 5
      - Node d
        - Node e
        - Node f
      - Node f
    - Node f
  - Node f
- Tree 2
  - Node 3
    - Node b
      - Node 8
    - Node e
  - Node f
  - Node f

**DEEP-UNION**

- Tree 1
  - Node 2
    - Node 5
      - Node d
        - Node e
        - Node f
      - Node f
    - Node f
  - Node f
- Tree 2
  - Node 3
    - Node b
      - Node 8
    - Node e
  - Node f
  - Node f

\( /x/a[e] \ \text{INTERSECT} \ //d[f] \)

**INTERSECT**

- Node null

**DEEP-INTERSECT**

- Node 5
  - Node d
    - Node f
    - Node 6
    - Node 7
/x/a[e] EXCEPT //d[f]

Eg, Deep-EXCEPT Implementation

// using XQuery's UDF
declare function local:DEEP-EXCEPT($P,$q) {
    for $pn in $p return
    if (empty($pn//* intersect $q) and empty($pn intersect $q) and not(empty($pn))) then $pn
    else if (not(empty($pn//* intersect $q))) then
        element{name($pn)}{
            $pn/@*, $pn/text(), for $pnn in $pn/*
            return local:DEEP-EXCEPT($pnn,$q)
        }
    } else()
Outline

- Introduction
- Motivation
- Framework
- QFilter Details
- QFilter Validation
- QFilter Applications
- Conclusion

Evaluation Plan

- No Access Control
  - Q \rightarrow \text{Galax} \rightarrow \text{UD}
  - (a) No security support

- Pre-Processing Scenario
  - Primitive
  - QFilter
  - Static Analysis [Murata; 2003]
  - Q \cap \text{ACR} \rightarrow \text{Galax} \rightarrow \text{SD}
  - (b) Primitive

- Post-Processing Scenario
  - Static Analysis
  - Q \rightarrow \text{QFilter} \rightarrow Q' \rightarrow \text{Galax} \rightarrow \text{SD}
  - (c) Pre-processing
  - Q \rightarrow \text{Galax} \rightarrow \text{UD} \rightarrow \text{AFilter} \rightarrow \text{SD}
  - (d) Post-processing

Figure 1: Ways to support XML access control without using security features of DBMS
Set-Up

- XMark / Galax / YFilter
- XML data (1.5 MB)
- ACR
  - 550 Synthetic rules
  - 10 User-defined rules
- Q
  - 7 categories based on /, //, *, []
  - 100 Synthetic queries

QFilter Performance

![Chart showing QFilter Execution time (ms) for different query categories. The categories range from 1 to 7. The chart includes bars for Query Accepted, Query Denied, and Query Rewritten. The execution times range from 0.01 to 0.07 milliseconds.]

Dongwon Lee, 2004
QFilter Performance

Comparison among Scenarios
Two Pre-Processing Methods

- QFilter Construction
- Static Analysis Initialization

Number of Access Control Rules vs. Initializing Time (ms).

Number of Rules per Role vs. Security check time (ms).
Outline

- Introduction
- Motivation
- Framework
- QFilter Details
- QFilter Validation
- QFilter Applications
- Conclusion

QFilter Applications

- QFilter is in essence a black box that
  - Consists of Constraints $C$
  - Takes input $Q$
  - Outputs filtered result $Q'$

- Applications
  - ACR View Maintenance
  - Secure P2P
### View-based XML Access Controls

- High maintenance cost when update occurs
- For an organization, frequent updates to ACR are possible
  - Rules are being added/removed
  - For each rule change, new view must be materialized
- Metric
  - Amount of data from DB to view
  - Time/Storage
View-based XML Access Controls

- T1: +R1, +R2, +R3
- T2: +R3 is removed

Secure P2P Overlay Network

- XML documents are stored on peer nodes (with possible duplication)
- Different peer nodes have different access controls
- When a node $N$ gets a query $Q$, if $N$ cannot handle $Q$, it has to forward $Q$ to proper peers who have legitimate access controls.
Secure P2P Overlay Network

Conclusion

- Research on “XML-Something” in RDBMS or Vanilla XDBMS looks promising
- A framework and solution, QFilter, are presented in the context of XML Access Control
- Applications of QFilter

Thanks!