

QFilter: Fine-Grained Run-Time Access Control via NFA-based XML Query Rewriting

Bo Luo

Dongwon Lee

Wang-Chien Lee

Peng Liu



Penn State University

Outline

- Introduction
- Motivation & Framework
- QFilter Details
- Evaluation
- Conclusion

Introduction

- XML Access Control ensures only authorized users can access only authorized portion of XML data
- Rich research community in RDBMS
- Emerging techniques and standards for XML Access Control
 - Model
 - 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)

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

- We aim at
 - Non-view based method
 - No need for XML engine with security feature
- Our Approach is
 - **Framework-based**: devise and compare various approaches that are not view-based
 - **Practical solution**: can work with off-the-shelf XML database engine

Framework

Data

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

Query

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

ACR

- 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

Scenarios

Conventional

Query

ACR

Data

View-based

Query

ACR

Data

Pre-processing

Query

ACR

Data

Post-processing

Query

Data

ACR

Miscellaneous

Query

Data

ACR

Post-Processing

Query

Data

ACR

- 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 (eg, YFilter [Diao, 2003])

Pre-Processing

Query

ACR

Data

- 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: $Q \cap \text{ACR} \Leftrightarrow Q'$
 - QFilter: $\text{QFilter}(Q, \text{ACR}) \Leftrightarrow Q'$

Pre-Processing: Primitive

- View Query and ACR as two **constraints** to satisfy
 - Q and + ACR: $Q \cap \text{ACR} \Leftrightarrow Q'$
 - Q and – ACR: $Q - \text{ACR} \Leftrightarrow Q'$
- 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

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

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** ($Q \cap \text{ACR} \Leftrightarrow Q$)
 - `/dept[year<2004]//budget` \cap `/dept/**`
 - \Leftrightarrow `/dept[year<2004]//budget`
- **Disjoint Case** ($Q \cap \text{ACR} \Leftrightarrow \{\}$)
 - `/dept[year<2004]//budget` \cap `/dept[year>2005]//budget` $\Leftrightarrow \{\}$
- **Overlapping Case** ($Q \cap \text{ACR} \Leftrightarrow Q'$)
 - `/dept[year<2004]//budget` \cap `//south/budget`
 - \Leftrightarrow `/dept/[year<2004]/south/budget`

Pre-Processing: QFilter

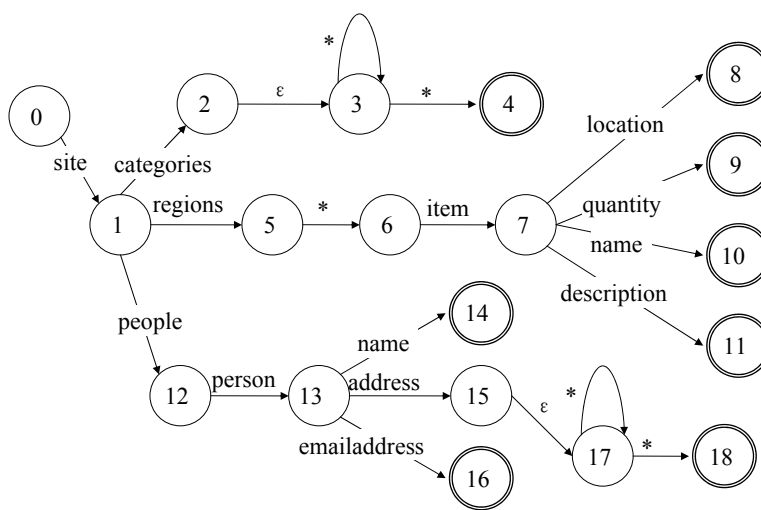
- QFilter captures ACR as NFA (Non-deterministic Finite Automata)
- Given Q, quickly determine if it is contained, disjoint or overlapping by traversing NFA
- When it's overlapping case, Q' is generated

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

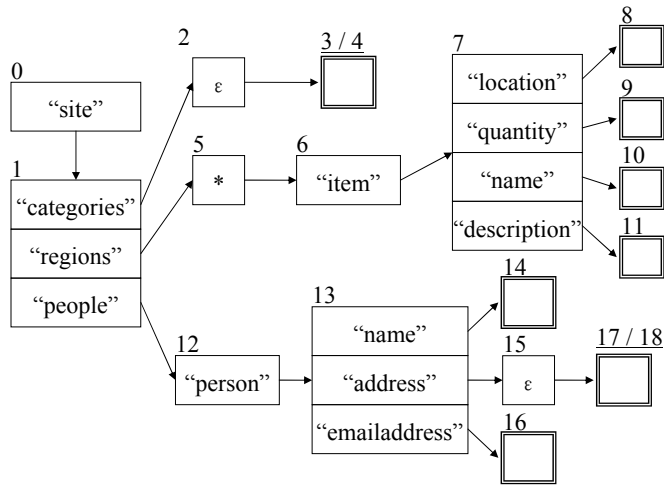
QFilter Example

- 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

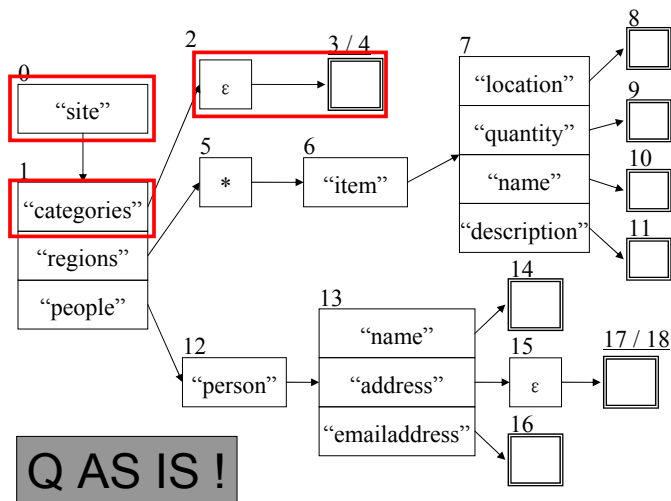
State Transition Map



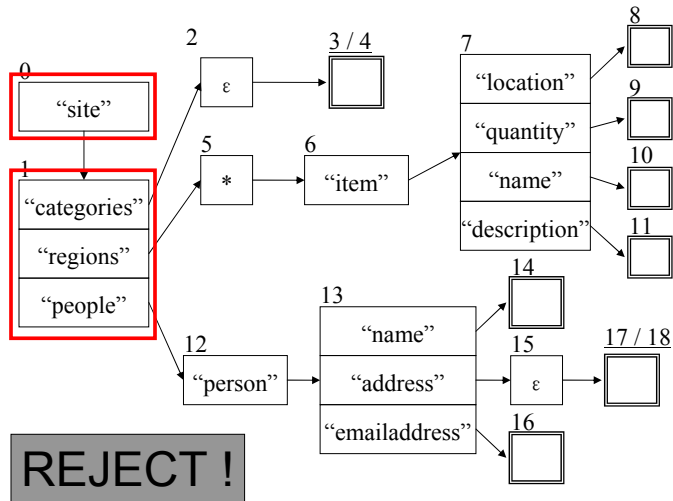
NFA



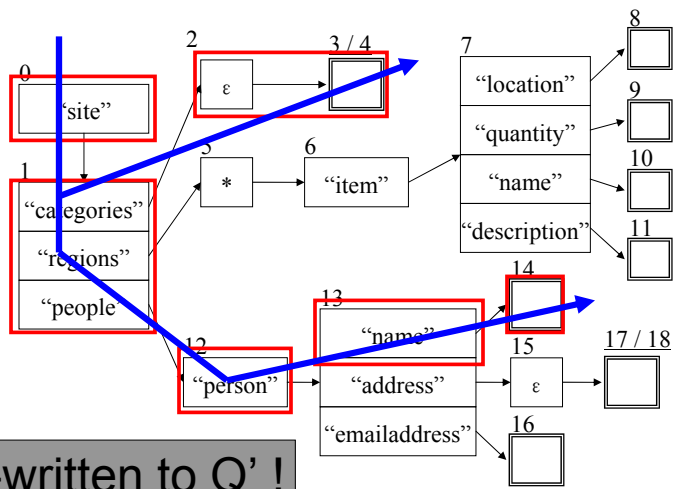
Q: /site/categories/NW/item



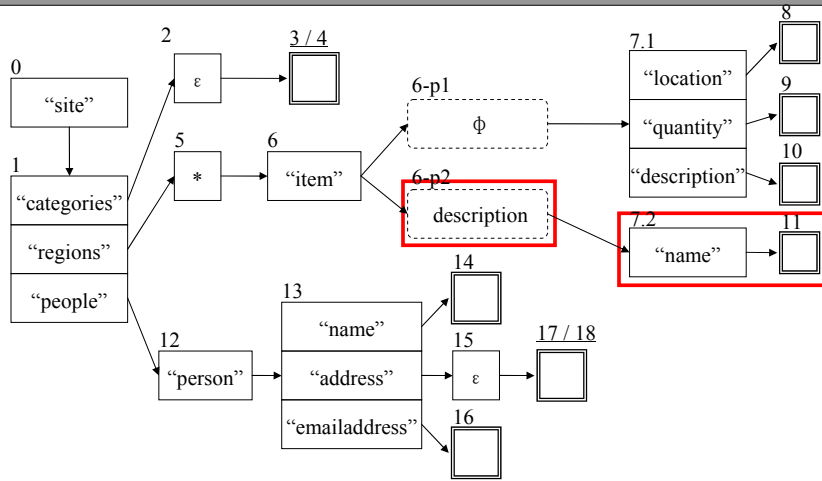
Q: /site/top//item



Q: /site/*/person/name

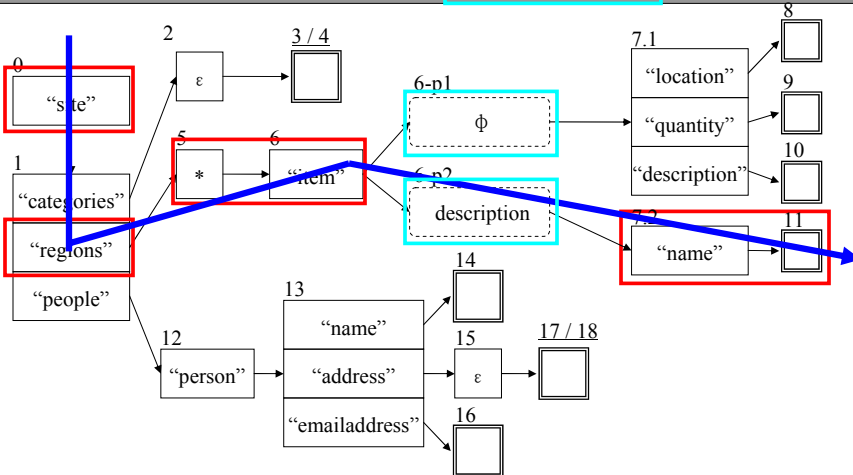


QFilter with Predicate Handling



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 $*$: $O(|Q|)$
 - $*$: $O(|NFA|)$
 - $//$: $O(|Q| * \Pi \text{ child for } i\text{-th } //)$
 - Worst case only occurs for a query $"/...//**.../**"$

Evaluation Plan

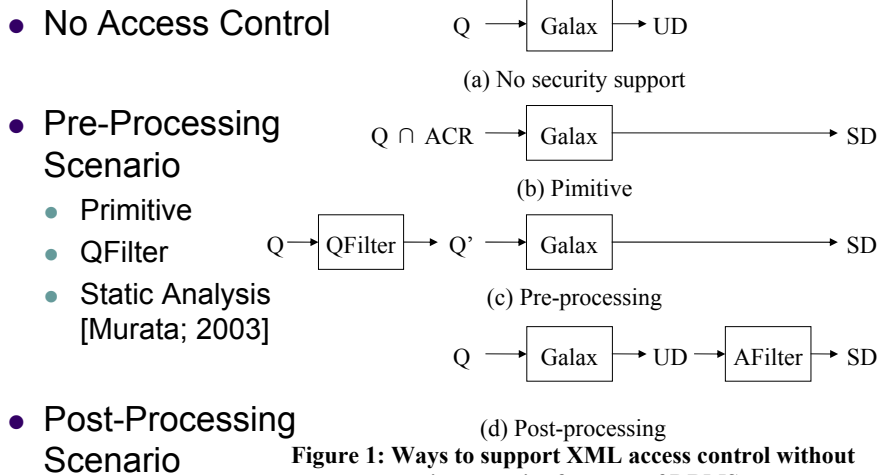
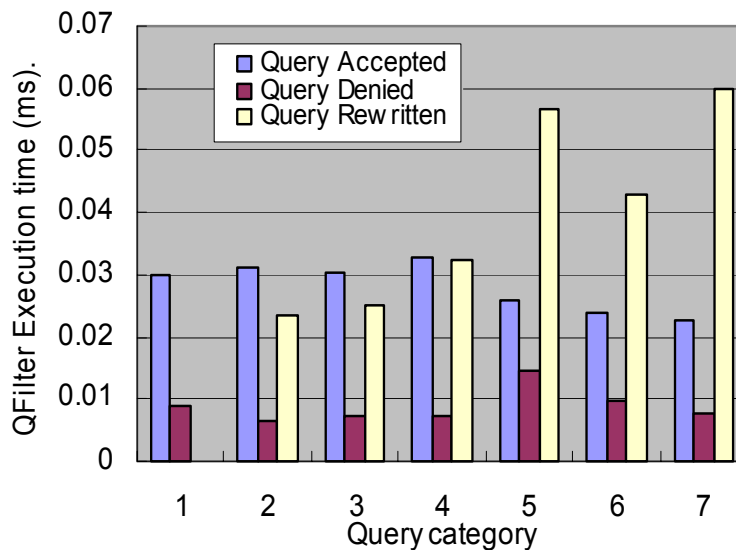


Figure 1: Ways to support XML access control without using security features of DBMS

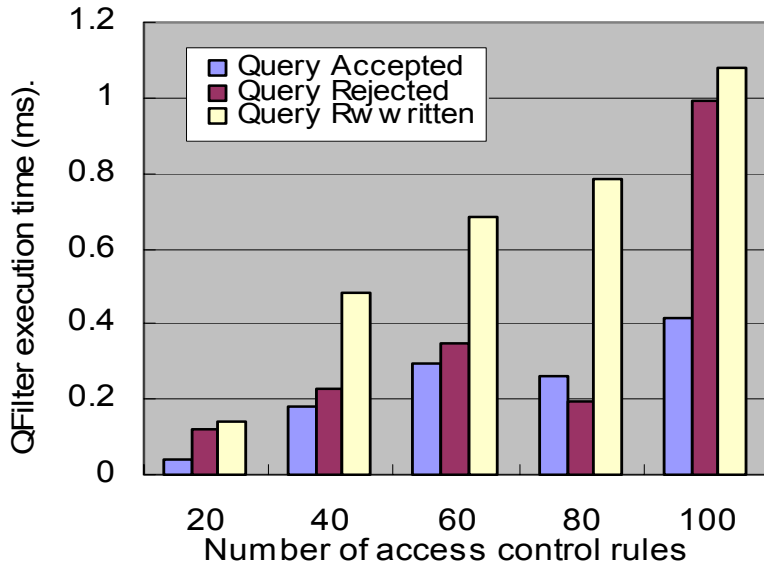
Set-Up

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

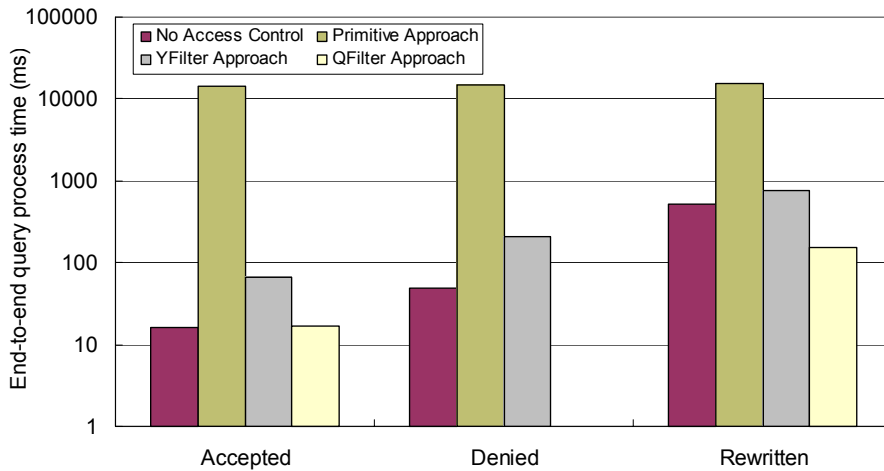
QFilter Performance



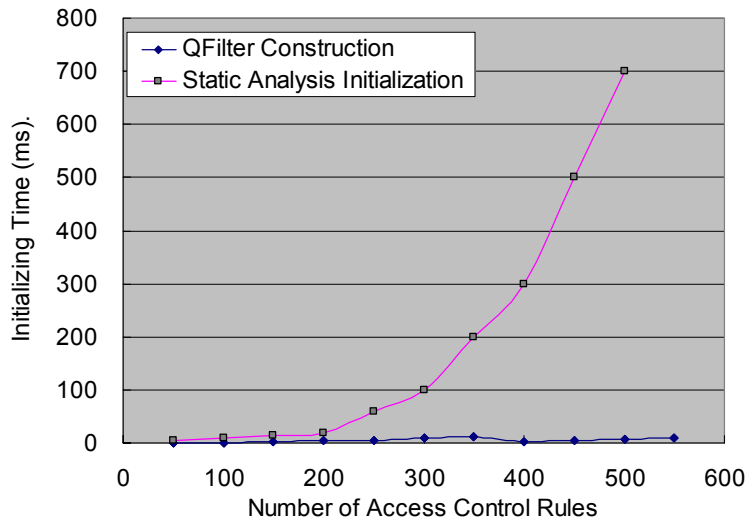
QFilter Performance



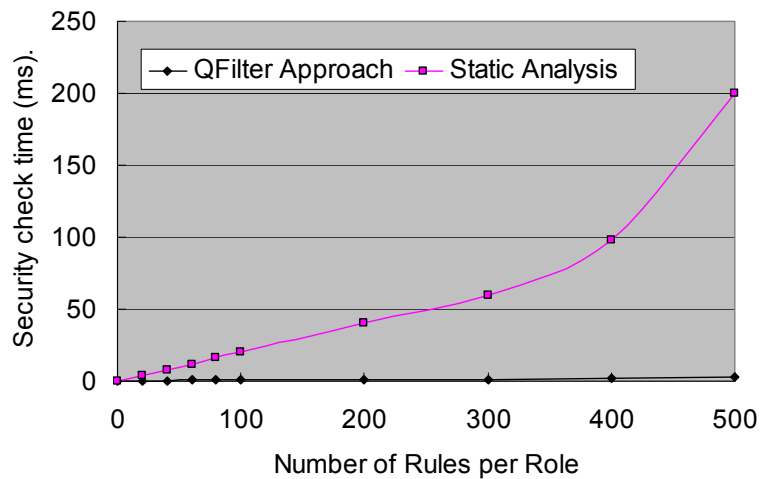
Comparison among Scenarios



Two Pre-Processing Methods



Two Pre-Processing Methods



Conclusion

- A framework with several solutions that are non view based and independent from database support.
- QFilter is superior to Post-processing and Primitive
- QFilter is superior to Static Analysis method

Thank You