Optimizing the Secure Evaluation of Twig Queries

Divesh Srivastava
AT&T Labs–Research
http://www.research.att.com/~divesh/

Joint work with S. Amer-Yahia, S. Cho and L.V.S. Lakshmanan
Outline of Talk

• XML security model
• Secure query evaluation
• Optimizing for security
• Experimental results
Motivation: Why XML Security?

- XML: standard for data publication/exchange over the Web
  - query access is important
- Different data elements may have different security levels
  - low security: catalogs
  - medium security: customer accounts
  - high security: special customers
**XML Security Model: Multi-Level Security**

- **Data:** define security level \((\in [1..n])\) at elements
  - inherited if not defined, non-monotonicity allowed
  - schema/DTD: mandatory (M), optional (O), forbidden (F) at elements
- **User:** define security level \((\in [1..n])\)
  - accessibility: user.SecurityLevel \(\geq\) element.SecurityLevel
Twig Queries

- Core component of XQuery, XML-QL
  - child and descendant axes of XPath, one distinguished (*) node
- Semantics: existential matching of non-distinguished nodes
  - crucial to take security into account
Secure Query Evaluation: Incorrect Strategies

- **Postprocessing:** evaluate query, check results for accessibility
  - **problem:** witness elements may be inaccessible

- **Local checks:** test **LC** predicate at each query node
  - **problem:** element’s security level may be inherited
Secure Query Evaluation: An Inefficient Strategy

- Recursive checks: test \textbf{RC} predicate at each query node
  - computes element’s (possibly inherited) security level
- Correct, but inefficient (experimental results later)
  - scope for optimization of secure evaluation!
Secure Query Evaluation: Optimizations

- Annotate query nodes by security check annotation labels
  - **NC** (no check): a no-op
  - **LC** (local check): checks security level attribute only at the element
  - **RC** (recursive check): identifies closest ancestor with security level defined

- Find an optimal cost, correct security check annotation
  - **NC** < **LC** < **RC**
  - optimal = non-dominated annotation
  - examine only query and DTD, not the database
Example: Optimizing Parent-Child Edges

- Examine query nodes in a top-down fashion
  - natural, since security levels are inherited top-down
- Identify paths in DTD along which security levels are inherited
  - parent-child edges: children get LC or NC
Example: Optimizing Parent-Child Edges

- Query root node has an NC label
  - no path from DTD root has mandatory or optional security level
- Other query nodes also have an NC label
  - parent-child edges, DTD nodes have forbidden security levels
Example: Optimizing Parent-Child Edges

- Label of query root node depends on path from DTD root node
  - any node with mandatory or optional security level?
- Label of other query nodes depends on corresponding DTD nodes
  - mandatory/optional $\Rightarrow$ LC, else NC
Example: Optimizing Ancestor-Descendant Edges

- Examine query nodes in a top-down fashion
  - natural, since security levels are inherited top-down
- Identify paths in DTD along which security levels are inherited
  - possibility of non-query nodes with mandatory/optional security levels
Example: Insufficiency of Top-Down Analysis

- Sibling *item* elements with same security level
  - guaranteed to inherit same security level from common ancestor
- Need for a forward pass, in topological order of query nodes
  - state information maintained about visited query nodes
Example: Utility of a Forward Pass

• Sibling item elements with different security levels
  ○ possible to inherit different security levels from different ancestors

• A forward pass, in topological order of query nodes, suffices
  ○ track DTD node from which security level is inherited
Example: Dealing with Existential Checks

- Guaranteed accessible sibling element
  - payment_info always has a name sibling, with same security level
- A forward pass, in topological order of query nodes, suffices
  - track DTD nodes and edge labels along which security levels are inherited
Example: Dealing with Existential Checks

- No guarantee of accessible sibling element
  - *name* doesn’t always have a *payment_info* sibling
- A forward pass, in topological order of query nodes, suffices
  - track DTD nodes and edge labels along which security levels are inherited
Example: Dealing with a DAG DTD

- Desired **name** may be of a **customer**, not an **author**
  - such a name element may be inaccessible
- Need to check multiple paths between DTD nodes
Example: Dealing with a DAG DTD

- Existential check for a **name** descendant of **online_seller**
  - guaranteed accessible **author** of **book**
- Need to check **some** path between DTD nodes
Correctness and Optimality

Theorem 5.1 (Optimality of ForwardPassTree) Let $Q$ be an arbitrary XML twig query, and $D$ be any tree-structured DTD graph. Algorithm ForwardPassTree correctly optimizes the SC annotations of $Q$ on $D$. Further, it is optimal in that if $Q_a$ is the SC annotation of $Q$ computed by Algorithm ForwardPassTree, then there is no SC annotation $Q_b (\neq Q_a)$ of $Q$ such that $Q_b \leq Q_a$. 
Experimental Evaluation: Setup

- **XMark benchmark dataset**: 5MB to 30MB
  - DTDs with sparse (3) and dense (half) optional security levels
  - All elements with optional security level assigned values
- **Twig queries provided by XMark**

  ![Diagram](image)

- **Optimizer implemented on top of XALAN**
Evaluation Time Comparisons

- Optimized evaluation is much better than unoptimized evaluation
  - optimized secure evaluation did not add much overhead
- Benefits of optimization more for sparse DTDs
  - unoptimized: sparse > dense, optimized: dense > sparse
Optimization Overhead

- Optimization time for query with $pc$ edges independent of DTD
- With $ad$ edges, optimization time higher for sparse DTDs
  - more edges needed to be traversed in DTD
- Optimization is very fast (with 20-30 nodes, < 0.25ms)
Related Work

• Sophisticated access control models for XML
  ○ [B+00,D+00,B+01]: positive/negative authorizations, authorization propagation flexibility, accessible view per user
  ○ [KH00]: XML access control language, with authorization, non-repudiation, confidentiality, audit trail

• XML query rewritings using DTD [PV99,W99]
  ○ containment mappings, chase, unification
  ○ not applicable for optimizing secure evaluation
Conclusions

- Secure evaluation of XML twig queries
  - efficient algorithm for determining optimal annotations
  - experimental validation
- Open problems
  - dealing with DTDs with cycles, alternation
  - utilizing default values for security levels, monotonicity
  - ...
Experimental Evaluation: Setup

- HL7 clinical dataset: 5MB to 30MB
  - 25% patients with SL=2, all observations with SL=2
  - 75% patients with SL=1, 50% observations with SL=2
- Synthetic twig queries

  ![Diagram](image.png)

- Optimizer implemented on top of XALAN