Rewriting of Queries and Updates across XML Security Views

Sławek Staworko
joint work with
Iovka Boneva, Benoît Groz, Anne-Cécile Caron, Yves Roos, and Sophie Tison

Mostrare Project
INRIA Lille - Nord Europe
University of Lille

Dagstuhl Seminar: Security and Rewriting

August 2011
the view schema is derived from Access Spec. and Schema
the view is virtual (no materialization)
user queries are rewritten and then evaluated
Overview

This talk includes:

1. Framework (XML, Regular XPath, DTDs)
2. Security Access Specification (SAS) and views
3. Query rewriting
4. Static analysis of SAS
5. Updates and their propagations

Related to:

- “XML Security Views Revisited,” DBPL’09
- “The View Update Problem for XML,” Workshop on XML Updates’10
- “View Update Translation for XML,” ICDT’11

N.B.: Framework introduced and investigated to some extent before.
Basic Notions
XML and XPath

Regular XPath ($\mathcal{X}Reg$)

\[
\begin{align*}
\alpha & ::= \text{self} | \downarrow | \uparrow | \Rightarrow | \Leftarrow \\
\gamma & ::= \text{lab()} = a | Q | \text{true} | \neg \gamma | \gamma \text{ and } \gamma \\
Q & ::= \alpha | [\gamma] | Q/Q | Q \cup Q | Q^*
\end{align*}
\]

\[
\begin{align*}
\alpha^+ & ::= \alpha^*/\alpha \\
\alpha::a & ::= \alpha[\text{lab()} = a] \\
\alpha::* & ::= \alpha \\
Q[\gamma] & ::= Q/[\gamma]
\end{align*}
\]
SAS = DTD + Annotation

DTD
projects → projects*
project → name, (stable | dev), license
stable → src, bin, doc
dev → src, doc
license → free | propr
SAS = DTD + Annotation

**DTD**

- `projects → projects*`
- `project → name, (stable | dev), license`
- `stable → src, bin, doc`
- `dev → src, doc`
- `license → free | propr`

**Annotation**

- `A(stable) = false`
- `A(dev) = false`
- `A(doc) = true`
- `A(src) = [↑:*:|=::license/↓::free]`
\[ \text{SAS} = \text{DTD} + \text{Annotation} \]

**Accessibility:**
- Root always accessible
- If A defined for the node label, then evaluate the filter
- Otherwise, accessibility inherited from the parent

**View:** \( A(t) = \text{tree obtained from accessible nodes only.} \)

**Annotation**
- \( A(\text{stable}) = \text{false} \)
- \( A(\text{dev}) = \text{false} \)
- \( A(\text{doc}) = \text{true} \)
- \( A(\text{src}) = \left[ \uparrow: \ast \rightarrow::\text{license}/\downarrow::\text{free} \right] \)

- hide status of project (including binaries)
- Source visible only if under free license
SAS = DTD + Annotation

- hide status of project (including binaries)
- source visible only if under free license

**Accessibility:**
- root always accessible
- if $A$ defined for the node label, then evaluate the filter
- otherwise, accessibility inherited from the parent

**View:** $A(t) =$ tree obtained from accessible nodes only.

**Annotation**

- $A(stable) = \text{false}$
- $A(dev) = \text{false}$
- $A(doc) = \text{true}$
- $A(src) = [\uparrow:\star/\Rightarrow::license/\downarrow::free]$
Query Rewriting
Query Rewriting

Problem statement

**Given:**
- source DTD $D_S$
- annotation $A$

**Input:** Regular XPath query $Q$

**Output:** Regular XPath query $Q' = \text{Rewrite}(Q)$ such that for every $t \in L(D_S)$ we have $Q'(t) = Q(A(t'))$

Lemma 1

For any annotation $A$ there exists a filter expression $f_{acc}$ such that a node $n$ of a tree $t$ is accessible w.r.t. $A$ if and only if $(t, n) \models f_{acc}$
Rewrite(\downarrow) := [f_{acc}]/{\downarrow}/([not f_{acc}]/{\downarrow})^{*}/[f_{acc}]
Query Rewriting: Horizontal axes

\textbf{Rewrite(⇒): combine 3 tricks}
Query Rewriting: Summary

Theorem

Regular XPath is closed under rewriting over XML views. The size of the rewritten query is $O(|A| \times |Q|)$, where $Q$ is the original query.

Theorem

Boolean and Unary MSO (expressed with tree automata) are closed under rewriting over XML views. Rewriting is polynomial.
Elements of Static Analysis
Static Analysis of SAS: What for?

Scenario: SAS Optimization

- Annotations are replaced with their streamlined versions
- Is the new SAS equivalent to the previous one?

Scenario: SAS Refinement

- SAS is changed to further restrict the access to the document.
- Is the new SAS strictly more restrictive than the previous one?
  Here, more restrictive may mean:
  1. Fewer nodes of the source document are visible
  2. Fewer queries can be executed on the source document
  3. Fewer information can be inferred about the source document
Node-based comparison

Equivalence

\[ A_1 \equiv^D A_2 \iff \forall t \in L(D). \text{Nodes}(A_1(t)) = \text{Nodes}(A_2(t)) \]
\[ \iff \forall t \in L(D). A_1(t) = A_2(t) \]

Node-based restriction

\[ A_1 \triangleleft^D_{NB} A_2 \iff \forall t \in L(D). \text{Nodes}(A_1(t)) \subseteq \text{Nodes}(A_2(t)) \]

Theorem

Testing equivalence and node-based restriction is EXPTIME-complete.
Node-based restriction too naïve?

Original annotation

\[ A_1(\text{stable}) = \text{false} \]
\[ A_1(\text{dev}) = \text{false} \]
\[ A_1(\text{doc}) = \text{true} \]
\[ A_1(\text{src}) = [\uparrow::*/\Rightarrow::\text{license}/\downarrow::\text{free}] \]

With the new annotation, the query
\[ \text{not} p::\text{src} \text{ and } \text{not} p::\text{license} \]
identifies a subset of projects under development which could not be selected before!
Node-based restriction too naïve?

Original annotation

\[ A_1(stable) = false \]
\[ A_1(dev) = false \]
\[ A_1(doc) = true \]
\[ A_1(src) = \left[ \uparrow::/* \Rightarrow::{license} \downarrow::free \right] \]
Node-based restriction too naïve?

Original annotation
\[A_1(stable) = false\]
\[A_1(dev) = false\]
\[A_1(doc) = true\]
\[A_1(src) = [\uparrow::*:\Rightarrow::license/\downarrow::free]\]

New annotation (hide sources of projects under development)
\[A_2(stable) = false\]
\[A_2(dev) = false\]
\[A_2(doc) = true\]
\[A(src) = [\uparrow::stabe/\Rightarrow::license/\downarrow::free]\]
Node-based restriction too naïve?

With the new annotation, the query

\[ \downarrow::\text{project}[\text{not}(\downarrow::\text{src}) \text{ and } \downarrow::\text{license}/\downarrow::\text{free}] \]

identifies a subset of projects under development which could not be selected before!
Query-based comparison

Identify queries executable on the source

\[ \text{Public}(D, A) = \{ Q \mid \exists Q'. \text{ Rewrite}(Q', A) \equiv^D Q \} \]

Definition (Query-based restriction)

\[ A_1 \leq_{QB}^D A_2 \iff \text{Public}(D, A_1) \subseteq \text{Public}(D, A_2) \]

Negative results

Testing query-based restriction is \textit{undecidable}.

Positive results

Testing query-based restriction for non-recursive DTDs is in \textsc{Exptime} and is \textsc{PSPACE}-hard.
Information-based restriction

What an attacker may suspect?

A well-informed attacker knows: the source DTD $D$, the annotation $A$, and the view instance $t_V$. The source document may be any of:

$$Inv(A, D, t_V) = \{ t \in L(D) \mid A(t) = t_V \}$$

What information that can the attacker infer?

$$Certain(D, A, t_S) = \{ Q \mid \forall t \in Inv(A, D, A(t_S)). t \models Q \}$$

Definition (Information-based restriction)

$$A_1 \leq_{IB}^D A_2 \iff \forall t \in L(D). Certain(D, A_1, t) \subseteq Certain(D, A_2, t)$$
Information-base comparison (cont’d)

Negative results
Testing information-based restriction is **undecidable**.

Positive results
Testing information-based restriction for non-recursive DTDs is in EXPTIME and is PSPACE-hard.
Further results: Interval-bounded SAS

Interval-bounded (IB) SAS

On a descending path in any source document the distance between two consecutive visible nodes is bounded by a fixed constant.

- IB (significantly) generalizes non-recursive DTDs.
- IB pushes the decidability frontier for IB.
- Enables the use of tree automata for a more powerful SAS and more fine-grained comparison of SAS.
Updates and their Rewritings
Alignment trees as Updates

Editing operations

- $(\epsilon, a)$ – insert a node
- $(a, \epsilon)$ – delete a node
- $(a, b)$ – rename $a$ to $b$
- $(a, a)$ – do nothing

Input

```
    r
   /|
  a  d  a
 /     |
|      c
(c, c)
```

Output

```
    r
   /|
  d  a  d
 /     |
| c  c  c
```

Editing script

- A tree over $\Sigma \times \{\epsilon\} \cup \Sigma \times \Sigma \cup \{\epsilon\} \times \Sigma$
- Downward-closed i.e., delete/insert whole subtrees
- Has associated cost (number of inserted and deleted nodes)
Update rewriting

**View update rewriting (propagation)**

**Given:**
- source DTD $D_S$
- annotation $A$ (downward-closed)
- view DTD $D_V = A(D_S)$
- source document $t \in L(D_S)$

**Input:** update of the view $S_o = A(t) \rightarrow t_o$ such that $t_o \in L(D_V)$

**Output:** update of the source document $S = t \rightarrow t'$ such that $S$:
- **side-effect free** i.e., $A(t') = t_o$
- **schema compliant** i.e., $t' \in L(D_S)$
- **optimal** i.e., the cost of $S$ is minimal among all updates of $t$ satisfying the two conditions above

---

**Theorem [Workshop on XML Updates 2010]**

An update rewriting can be constructed in polynomial time (DTD is fixed).
Update programs

Update program for a DTD $D$ (cf. XQuery Update Facility)

A set of updates $\mathcal{U}$ that is

- **schema compliant** i.e., $\forall S \in \mathcal{U}$ the input and output of $S$ satisfy $D$
- **functional** i.e., $\forall t \in L(D)$ there is exactly one $S \in \mathcal{U}$ matching $t$

$\mathcal{U}$ is **regular** if it is defined with a tree automaton

Constrained update program rewriting

**Given**: source DTD $D_S$, annotation $A$, view schema $D_V$, a set of allowed updates $\Omega$ of the source $D_S$

**Input**: view update program $\mathcal{U}_o \subseteq \Omega$

**Output**: source update program $\mathcal{U}$ such that

$\forall t \in L(D_S). \mathcal{U}_o(A(t)) = A(\mathcal{U}(t))$
... and their rewritings

**Unconstrained case (\(\Omega\) allows all updates)**
Rewritings of general (regular) update programs can be easily constructed.

**Constrained case**
Constrained rewritings of (regular) update programs cannot be constructed.

**Synchronized updates**
On a descending path in an alignment tree the distance between two consecutive node that are preserved (not deleted nor inserted) is bounded by a constant.

**Constrained case**
Rewritings of synchronized regular update programs can be constructed.
Thank you
And if there is more time...
Constructing View Schema
Deriving view schema

**DTD**

\[
\begin{align*}
\text{projects} & \rightarrow \text{projects}^* \\
\text{project} & \rightarrow \text{name}, \,(\text{stable} \mid \text{dev}), \,\text{license} \\
\text{stable} & \rightarrow \text{src}, \,\text{bin}, \,\text{doc} \\
\text{dev} & \rightarrow \text{src}, \,\text{doc} \\
\text{license} & \rightarrow \text{free} \mid \text{propr}
\end{align*}
\]

**Annotation**

\[
\begin{align*}
A(\text{stable}) &= \text{false} \\
A(\text{dev}) &= \text{false} \\
A(\text{doc}) &= \text{true}
\end{align*}
\]
Deriving view schema

```xml
<projects>
  <project>
    <name>stable</name>
    <license>propr</license>
    <src></src>
    <bin></bin>
    <doc></doc>
  </project>
  <project>
    <name>dev</name>
    <license>free</license>
    <src></src>
    <doc></doc>
  </project>
</projects>
```

**Annotation**
- \( A(\text{stable}) = \text{false} \)
- \( A(\text{dev}) = \text{false} \)
- \( A(\text{doc}) = \text{true} \)

**DTD**

```
projects → projects
project → name, (stable | dev), license
stable → src, bin, doc
dev → src, doc
license → free | propr
```
Deriving view schema

**DTD**
- `projects → projects*`
- `project → name, (stable | dev), license`
- `stable → src, bin, doc`
- `dev → src, doc`
- `license → free | propr`

**View DTD**
- `projects → projects*`
- `project → name, doc, license`
- `license → free | propr`

**Annotation**
- `A(stable) = false`
- `A(dev) = false`
- `A(doc) = true`
One problem: Size

**DTD (annotated)**

\[
\begin{align*}
  r & \rightarrow a_n \\
  a_n & \rightarrow a_{n-1}, a_{n-1} \\
  a_{n-1} & \rightarrow a_{n-2}, a_{n-2} \\
  \ldots \\
  a_1 & \rightarrow \text{empty} \\
  A(a_n) & = \text{false} \\
  A(a_1) & = \text{true}
\end{align*}
\]

**View DTD**

\[
\begin{align*}
  r & \rightarrow a_1, \ldots, a_1 \\
  a_1 & \rightarrow \text{empty}
\end{align*}
\]

**Observation**

The view DTD may be of *exponential* size!
And another one: Regularity

**Observation**

The view schema needs not be regular (in particular may not have a DTD)

**Proposition**

It is **undecidable** to test if the view schema can be captured with a DTD.
Approximation: Optimality criterion

Definition (Indistinguishability)

Two sets of trees $L_1$ and $L_2$ are \textit{indistinguishable} by a class of queries $C$ iff

$$\forall Q \in C. \left[ (\exists t_1 \in L_1. \; t_1 \models Q) \iff (\exists t_2 \in L_2. \; t_2 \models Q) \right].$$

Approximation

A DTD $D^*$ is a \textbf{good approximation} of the view schema of $D$ and $A$ if $L(D^*)$ and $\{ A(t) \mid t \in L(D) \}$ are indistinguishable by a relatively large class of queries.
Three approximations

**Parikh**
- $r \rightarrow (a, b)^*$
- $\mathcal{X} \text{Reg}(\downarrow, \uparrow, \lbracket, \text{not})$

**Subword**
- $r \rightarrow a^*, b^*$
- $\mathcal{X} \text{Reg}(\downarrow, \uparrow, \Rightarrow^+, \Leftarrow^+, \lbracket)$

**Subset**
- $r \rightarrow (a \mid b)^*$
- $\mathcal{X} \text{Reg}(\downarrow)$

**DTD (annotated)**
- $r \rightarrow c$
- $c \rightarrow (a, c\?, b)$
- $A(c) = \text{false}$
- $A(a) = \text{true}$
- $A(b) = \text{true}$
Further results

- $\mathcal{X} \text{Reg}(\downarrow, \Rightarrow)$ or $\mathcal{X} \text{Reg}(\downarrow, \Rightarrow^+, [], \text{not})$
  - No approximation

- $\mathcal{X} \text{Reg}(\downarrow, \uparrow, [], \text{not})$
  - Parikh approximation

- $\mathcal{X} \text{Reg}(\downarrow, \uparrow, \Rightarrow^+, \leftarrow^+, [])$
  - Subword approximation

- $\mathcal{X} \text{Reg}(\downarrow, \uparrow)$ or $\mathcal{X} \text{Reg}(\downarrow, [])$
  - Lower exponential bound

- $\mathcal{X} \text{Reg}(\downarrow)$
  - Subset approximation