Qapla: Policy compliance for database-backed systems

Aastha Mehta\textsuperscript{1}, Eslam Elnikety\textsuperscript{1}, Katura Harvey\textsuperscript{1,2}, Deepak Garg\textsuperscript{1}, Peter Druschel\textsuperscript{1}

\textsuperscript{1}Max Planck Institute for Software Systems (MPI-SWS)
\textsuperscript{2}University Of Maryland, College Park
Web applications store confidential data in DBMS

Users

Application

DBMS

confidential data

Healthcare systems
- patient records

Personnel management systems
- salaries, ages

Conference management systems
- submissions, reviews

Requirement: applications must comply with data access policies

Challenge: specifying and enforcing complex data access policies is non-trivial
Web applications store confidential data in DBMS

- Users
  - Healthcare systems
    - patient records
  - Personnel management systems
    - salaries, ages
  - Conference management systems
    - submissions, reviews

Application architecture

Requirement: applications must comply with data access policies

Challenge: specifying and enforcing complex data access policies is non-trivial

this talk: HotCRP
Current approach: enforcing policy in application code

Example from HotCRP (simplified for illustration)

```php
review.php:
loadRows();

loadRows() {
    $getinfo["paperId"] = $_REQUEST["paperId"];
    if ($Me->isPC && $Conf->timePCReviewPreferences() || $Me->privChair)
        $getinfo["reviewerPreference"] = true;
    $prow = $Conf->paperRow($getinfo, $Me);
    $rrows = $Conf->reviewRow($getinfo, $prow);
    if (!can_view_paper($prow))
        return null;
    if (!$Me->can_view_review($prow, $rrow) && !$Me->privChair)
        return null;
    return $prow;
}

if (!$Me->can_view_review($prow)) report_error();

if ($Me->can_view_review($prow))
    echo "<html>$prow</html>";
```

Current application logic:

- **load reviews**
- **fetch reviews for paperId**
- **checks based on user roles, conference phase**
- **checks based on user roles, conflicts, conference phase**
- **more checks ...**
Policy checks inlined throughout application code

Application code

get_review($paper, $user) {
    $result = $db->query("SELECT * FROM Reviews WHERE Review.paperId=".$paper-
    paperId); $rows = $result->fetchObject();
    foreach ($rows as $row) {
        if (can_view_review($row, $user)) show($row->review);
    }
}

get_comments($reviewId, $user) {
    $result = $db->query("SELECT * FROM Comments WHERE reviewId=".$reviewId-
    ); $row = $result->fetchRow();
    if (!can_view_review($row, $user)) return "";
    $result = $db->query("SELECT name FROM Contacts WHERE roles=PC|CHAIR");
    $rows = $result->fetchObject();
    foreach ($rows as $row) {
        if (can_view_contact($row, $user)) {
            comments[] = $row;
        }
    }
    return $comments;
}

get_pc_members($paper, $user) {
    $result = $db->query("SELECT name FROM Contacts WHERE roles=PC|CHAIR");
    $rows = $result->fetchObject();
    foreach ($rows as $row) {
        if (can_view_contact($row, $user)) show($row->name);
    }
}

- Must update checks in several code paths as application or policies evolve
- Easy to miss or implement incorrect checks
Examples of data leaks in HotCRP

**HotCRP v2.58** 23.Mar.2013
More information leak plugging: explicit search for review fields that should be hidden from authors, and review rounds. Reported by John Heidemann.

**HotCRP v2.59** 14-Jun.2013
Bug fix: "Monitor external reviews" works. Reported by Peter Sewell.
Information leak fixes: During response periods, don’t notify authors of changes in PC-only fields. Don’t allow searches on review rounds for conflicted papers. Don’t show accept status via "Accepted papers" searches. Reported by Nickolai Zeldovich and Jeff Mogul.

Source: http://read.seas.harvard.edu/~kohler/hotcrp/news.html
Limitations of existing approaches

Enforcing policy in application code:
Bugs in application code may lead to policy violations and cause data leaks

Using DBMS access control support:
Cannot support all policies without changes to DB schema or application queries
Our approach

Goals:
- Separate policy compliance from application code and DBMS
- Support complex, fine-grained data access policies
- Add only moderate performance overheads for end users
Outline

- Policy compliance today
  - Policy checks in application
  - DBMS access control

- Qapla
  - Design
  - Policy specification
  - Policy enforcement

- Evaluation
Qapla: compliance independent of application code and DBMS

User

request \(\rightarrow\) response

Application

query

query result

DBMS

Application data tables
Qapla: compliance independent of application code and DBMS

1 Declarative policies associated with the DB schema, stored in the DBMS
Qapla: compliance independent of application code and DBMS

1. **Declarative policies** associated with the DB schema, stored in the DBMS

2. Policies enforced by a reference monitor integrated with a DB adapter
Qapla: compliance independent of application code and DBMS

1. **Declarative policies** associated with the DB schema, stored in the DBMS
2. Policies enforced by a **reference monitor** integrated with a DB adapter
3. Reference monitor **rewrites query** with applicable policies
Qapla: compliance independent of application code and DBMS

1. **Declarative policies** associated with the DB schema, stored in the DBMS.
2. Policies enforced by a reference monitor integrated with a DB adapter.
3. Reference monitor rewrites query with applicable policies.
4. Forwards compliant query results to the application.

Diagram:
- User sends a request to the application.
- The application sends a query to the DB adapter.
- The Qapla reference monitor enforces policies by rewrites the query with applicable policies.
- The DB adapter forwards compliant query results to the application.

Key components:
- Application
- Query
- DB adapter
- Qapla reference monitor
- DBMS
- Application data tables
- Qapla policies
- Policies
Qapla: compliance independent of application code and DBMS

1. **Declarative policies** associated with the DB schema, stored in the DBMS

2. Policies enforced by a **reference monitor** integrated with a DB adapter

3. Reference monitor **rewrites query** with applicable policies

4. Forwards **compliant query results** to the application

Compliant application queries → **same results**

Non-compliant queries → **fewer results**
Our goal: prevent inadvertent data leaks due to application bugs
Threat model

Our goal: prevent inadvertent data leaks due to application bugs

Prototype Limitations

- application does not circumvent the reference monitor
  - can use software fault isolation, process or address space isolation
- application sends correct user identity to reference monitor
  - user can directly authenticate with the reference monitor
Policies on DB queries

1. single column policies
   - SELECT col

2. link policy (on joins, filters)
   - coll JOIN col2

3. policies on aggregate, group by
   - SELECT COUNT(col)
     GROUP BY col

4. policies on user defined functions (UDF)
   - SELECT UDF(col)

*in the paper*
Policy on queries that read single column

Authors can see their reviews only after decision notification

<table>
<thead>
<tr>
<th>Reviews</th>
<th>Papers</th>
</tr>
</thead>
<tbody>
<tr>
<td>paperId</td>
<td>contactId</td>
</tr>
<tr>
<td>123</td>
<td>9</td>
</tr>
</tbody>
</table>
Policy on queries that read single column

Authors can see their reviews only after decision notification

<table>
<thead>
<tr>
<th>Reviews</th>
<th>Papers</th>
</tr>
</thead>
<tbody>
<tr>
<td>paperId</td>
<td>paperId</td>
</tr>
<tr>
<td>contactId</td>
<td>author</td>
</tr>
<tr>
<td>review</td>
<td>outcome</td>
</tr>
<tr>
<td>123</td>
<td>123</td>
</tr>
<tr>
<td>9</td>
<td>Alice</td>
</tr>
</tbody>
</table>

Reviews.review :-
Policy on queries that read single column

Authors can see their reviews only after decision notification

Reviews

<table>
<thead>
<tr>
<th>paperId</th>
<th>contactId</th>
<th>review</th>
</tr>
</thead>
<tbody>
<tr>
<td>123</td>
<td>9</td>
<td></td>
</tr>
</tbody>
</table>

Reviews.review :-
CURRENT_DATE() >= 22-06-2017
AND
EXISTS(SELECT 1 FROM Papers
WHERE paperId = Reviews.paperId
AND author = $user)

/* date after decision notification */

/* user is an author of paper with paperId = Reviews.paperId */
# Policy on queries that read single column

Authors can see their reviews only after decision notification.

<table>
<thead>
<tr>
<th>Reviews</th>
<th>Papers</th>
</tr>
</thead>
<tbody>
<tr>
<td>paperId</td>
<td>paperId</td>
</tr>
<tr>
<td>contactId</td>
<td></td>
</tr>
<tr>
<td>review</td>
<td></td>
</tr>
<tr>
<td>123</td>
<td>123</td>
</tr>
</tbody>
</table>

```
Reviews.review :-
CURRENT_DATE() >= 22-06-2017
AND
EXISTS(SELECT 1 FROM Papers
WHERE paperId = Reviews.paperId
AND author = $user)
```

/* user is an author of paper with paperId = Reviews.paperId */

/* date after decision notification */
Policy on queries that link multiple columns

Authors can see PC **names** and **reviews** independently, but cannot link them.

<table>
<thead>
<tr>
<th>Reviews</th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>paperId</strong></td>
<td><strong>contactId</strong></td>
<td><strong>review</strong></td>
<td></td>
</tr>
<tr>
<td>123</td>
<td>9</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Contacts</th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>contactId</strong></td>
<td><strong>name</strong></td>
<td><strong>role</strong></td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>Bob</td>
<td>PC</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Papers</th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>paperId</strong></td>
<td><strong>author</strong></td>
<td><strong>outcome</strong></td>
<td></td>
</tr>
<tr>
<td>123</td>
<td>Alice</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Policy on queries that link multiple columns

Authors can see PC names and reviews independently, but cannot link them.

\[ \{ \text{Reviews.review, Reviews.contactId, Contacts.name, Contacts.contactId} \} \] :-

<table>
<thead>
<tr>
<th>Reviews</th>
<th>paperId</th>
<th>contactId</th>
<th>review</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>123</td>
<td>9</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Contacts</th>
<th>contactId</th>
<th>name</th>
<th>role</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>9</td>
<td>Bob</td>
<td>PC</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Papers</th>
<th>paperId</th>
<th>author</th>
<th>outcome</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>123</td>
<td>Alice</td>
<td></td>
</tr>
</tbody>
</table>
Policy on queries that link multiple columns

Authors can see PC names and reviews independently, but cannot link them.

\{Reviews.review, Reviews.contactId, Contacts.name, Contacts.contactId\} :-

`NOT EXISTS(SELECT 1 FROM Papers WHERE author=$user)`

*author cannot access these columns together*
Policy on queries that link multiple columns

Authors can see PC names and reviews independently, but cannot link them

\{Reviews.review, Reviews.contactId, Contacts.name, Contacts.contactId\} :-

\text{NOT EXISTS}(\text{SELECT 1 FROM Papers WHERE author=$user})

\text{authors can see their reviews only after notification}

\text{all users can see PC names}

\text{author cannot access these columns together}
Policy on aggregate queries

<table>
<thead>
<tr>
<th>paperId</th>
<th>author</th>
<th>outcome</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>accept</td>
</tr>
<tr>
<td>123</td>
<td>Alice</td>
<td>accept</td>
</tr>
<tr>
<td></td>
<td></td>
<td>reject</td>
</tr>
</tbody>
</table>

Authors can see their outcome only after notification.
Policy on aggregate queries

Anyone can see the number of submitted and accepted papers after the decision notification

{Papers.outcome[COUNT, GROUP BY]} :-
CURRENT_DATE() >= 22-06-2017 /* allow COUNT query with GROUP BY on outcome only after notification */

<table>
<thead>
<tr>
<th>paperId</th>
<th>author</th>
<th>outcome</th>
</tr>
</thead>
<tbody>
<tr>
<td>123</td>
<td>Alice</td>
<td>accept</td>
</tr>
<tr>
<td></td>
<td></td>
<td>reject</td>
</tr>
</tbody>
</table>
Summary: Qapla policy specification framework

**Qapla policy**
set of columns : - SQL *WHERE* clauses on tables

1. single column policies
2. link policy (on joins, filters)
3. policies on aggregate, group by
4. policies on UDFs
Policy enforcement

T1
<table>
<thead>
<tr>
<th>col1</th>
<th>col2</th>
</tr>
</thead>
</table>

Policies:
\{\text{col1}\} :\ P1
\{\text{col2}\} :\ P2
\{\text{col1,}\text{col2}\} :\ P3

Query on single column

Q: SELECT \text{col1} FROM T1

Policies = \{\text{P1}\}

T1': SELECT * FROM T1
WHERE \text{P1}

Q_r: SELECT \text{col1} FROM \text{T1}'

Query on multiple columns

Q: SELECT \text{col1} FROM T1 WHERE \text{col2} = \text{expr1}

Policies = \{\text{P3}\}

T1': SELECT * FROM T1
WHERE \text{P3}

Q_r: SELECT \text{col1} FROM \text{T1}' WHERE \text{col2} = \text{expr1}
Policy enforcement

<table>
<thead>
<tr>
<th>col1</th>
<th>col2</th>
</tr>
</thead>
</table>

Policies:

<table>
<thead>
<tr>
<th>{col1}</th>
<th>P1</th>
</tr>
</thead>
<tbody>
<tr>
<td>{col2}</td>
<td>P2</td>
</tr>
<tr>
<td>{col1, col2}</td>
<td>P3</td>
</tr>
</tbody>
</table>

Query on multiple columns

Q: SELECT col1 FROM T1 WHERE col2 = expr1

T1': SELECT * FROM T1
WHERE P3

Q_r: SELECT col1 FROM T1' WHERE col2 = expr1

Step 1: Identify the policies applicable to the set of columns in the query

Step 2: Replace each table with a subquery containing the applicable policies
Outline

- Policy compliance today
  - Policy checks in application
  - DBMS access control
- Qapla
  - Design
  - Policy specification
  - Policy enforcement
- Evaluation
Reference monitor implementation

- ~20K lines of C code (+ MySQL parser library)
- identifying set of columns, rewriting query with applicable policies
- API: create and set policies on columns

Ported reference monitor to:

**Web frameworks**
- PHP
- HotCRP
- MySQL

**Applications**
- commercial DBMS

**DBMSes**
- Python Django
- MPI-SWS job portal
- MySQL

*in the paper*
Policy compliance in HotCRP

- Schema: 22 tables, 215 columns
- Anonymized data from past conference hosting
- Implemented 30 policies for a typical configuration
  - double blind reviewing
  - chair conflict handling
  - review process with no rebuttal

- Application changes
  - overly general queries return fewer results with Qapla
  - changed ~150 out of ~52K LoC in application (< 0.3%)
End-to-end latency for user actions

Author clicks review URL: an author clicks the URL of paper to view reviews after notification

PC saves comment: a PC member clicks a button to save comments on a paper during review phase

Chair sets conflict: Chair assigns conflict for a paper and a PC member

Chair clicks assign button: Chair clicks a button automatic review assignment
End-to-end latency for user actions

Normalized latency vs. HotCRP end-user actions:

- **Baseline**
- **Qapla**

- **Absolute latency (ms):**
  - **Author clicks review URL:** 59.7 ms, 83.5 ms
  - **PC saves comment:** 101.6 ms, 127.0 ms
  - **Chair saves conflict:** 93.7 ms, 137.9 ms
  - **Chair clicks assign button:** 1517.5 ms, 6364.5 ms

- **Overhead below user-perceptible threshold:**
  - Overhead: 320%
  - Reviewer assignment is an infrequent task

- **< 45 queries (~3800 queries):**
End-to-end latency for user actions

Most of Qapla’s overhead corresponds to execution of rewritten queries

<table>
<thead>
<tr>
<th>HotCRP end-user actions</th>
<th>Baseline</th>
<th>Qapla</th>
</tr>
</thead>
<tbody>
<tr>
<td>Author clicks review URL</td>
<td>59.7</td>
<td>83.5</td>
</tr>
<tr>
<td>PC saves comment</td>
<td>101.6</td>
<td>127.0</td>
</tr>
<tr>
<td>Chair saves conflict</td>
<td>93.7</td>
<td>137.9</td>
</tr>
<tr>
<td>Chair clicks assign button</td>
<td>1517.5</td>
<td>6364.5</td>
</tr>
</tbody>
</table>

- *overhead below user-perceptible threshold*
- *< 45 queries*
- *overhead: 320%*
- *~3800 queries*

Reviewer assignment is an infrequent task
Summary

DB-backed systems → application bugs → data leaks due to policy violations

Qapla: an **effective policy compliance system**

- independent of DBMS support for access control
- independent of application code
- modest changes to application for functionality
- moderate overhead for end users

Questions?