Blockaid
Data-access Policy Enforcement for Web Applications

Wen Zhang    Eric Sheng    Michael Chang
Aurojit Panda Mooly Sagiv    Scott Shenker

Berkeley EECS
ELECTRICAL ENGINEERING & COMPUTER SCIENCES

NETSYS
Web Applications Are Everywhere
Private Message

From: 🧑‍🎓Alice
To: 👤Bob
Content:
Here’s my little secret…
Web Applications Serve Sensitive Information

Sensitive information be released only to parties that should have access to it

Alice  Bob  (Valid)

Charlie  (Invalid)
What information can be accessed by which users?

Data-access Policy

• A direct message is accessible only to its participants.
• ...

A direct message is accessible only to its participants.
Data-access Policy
Enforced using access checks

- A **direct message** is accessible only to its **participants**.
- ...

Every time a message is displayed...

```python
if not message.has_participant(curr_user):
    return "Error"

return message.content
```
Access Checks Are Hard to Get Right

Missing/incorrect checks ➔ inadvertent data leaks in production software

Matthew Green
@matthew_d_green

Piazza offers anonymous posting, but does not hide each user’s total number of posts. Discuss.

10:37 AM · Oct 30, 2017 · Twitter for iPhone

Search leaks hidden tags #135

Closed

ben-stock opened this issue on Jun 21, 2018 · 4 comments

ben-stock commented on Jun 21, 2018

If you use the search to find a paper and start with a if, the auto-complete feature will leak which tags are around. This is not necessarily really bad (as you cannot actually see the papers tagged unless you have the permission to see the tags), but still inconsistent handling of "Hidden tags"
How to systematically ensure an application reveals only information allowed by its data-access policy?
How to systematically ensure an application reveals only information allowed by its data-access policy?

Blockaid
Run-time data-access policy enforcer for web applications
Coming Up Next...

1. Overview and goals.
2. Policy specification.
3. Policy enforcement.
4. Evaluation.
Coming Up Next...

1. Overview and goals.
2. Policy specification.
3. Policy enforcement.
4. Evaluation.
Blockaid: Overview

Web application -> SQL -> Blockaid -> SQL database

Data-access policy
Blockaid: Overview

Web application → SQL → Blockaid → SQL → SQL database

SQL Rows

Data-access policy
Blockaid: Overview

Web application → SQL → Blockaid → SQL database

Data-access policy
Blockaid: Overview

Web application ➔ SQL ➔ Blockaid ➔ SQL database

Data-access policy
Goals

Ensure application reveals only **information that the policy allows** the user to access.

1. Policy expressiveness.
2. Compatibility with existing frameworks.
   - **Complies** with policy ➔ Maintain application behavior.
   - **Violates** policy ➔ Raise error **visibly**.
4. Low performance overhead.
No Prior System Satisfies All Four Goals

Ensure application reveals only information that the policy allows the user to access.

1. Policy expressiveness.
2. Compatibility with existing frameworks.
   Complies with policy ➔ Maintain application behavior.
   Violates policy ➔ Raise error visibly.
4. Low performance overhead.
Prior Systems Don’t Meet Both Goals

Query Modification

- Compatible with existing frameworks
- Not semantically transparent

Towards Multiverse Databases

Precise, Dynamic Information Flow for Database-Backed Applications

Jean Yang
Carnegie Mellon University and Harvard Medical School, USA

Travis Hance
Dropbox, USA

Thomas H. Austin
San Jose State University, USA

Armando Solar-Lezama
Massachusetts Institute of Technology, USA

Cormac Flanagan
University of California, Santa Cruz, USA

Stephen Chong
Harvard University, USA

Static Verification

- Semantically transparent
- Incompatible with existing frameworks

Static Checking of Dynamically-Varying Security Policies in Database-Backed Applications

Adam Chipala
Impredicative LLC

STORM: Refinement Types for Secure Web Applications

Nico Lehmann
UC San Diego

Rose Kunkel
UC San Diego

Jordan Brown
Independent

Jean Yang
Akia Software

Niki Vazou
IMDEA Software Institute

Nadia Polikarpova
UC San Diego

Deian Stefan
UC San Diego

Ranjit Jhala
UC San Diego
Blockaid

Run-time data-access policy enforcer for web applications

- Compatible with existing frameworks
- Semantically transparent
- Supports expressive policies
- Incurs low performance overhead
Blockaid: A Close Look

Web application ➔ Request context ➔ Decision cache ➔ Blockaid ➔ SQL database

- (Q1, O1)
- (Q2, O2)
- (Q3, O3)

Data-access policy

- Error ➔ Not cached ➔ SMT solvers ➔ Compliant?
- Cached ➔ Compliant?

Decision cache

Q3

Compliant?

Cached

Not cached

SQL database

Web application

Q3

O3

O3
Coming Up Next...

1. Overview and goals.
2. Policy specification.
3. Policy enforcement.
4. Evaluation.
Specify Policy Using Database Views

Views: SQL SELECT statements defining data accessible to a user.
Example: Calendar Application

1. **SELECT * FROM Users**
   Each user can view information on all users.

2. **SELECT EId FROM Attendance WHERE UId = ?MyUId**
   Each user can view IDs of events they attend.

3. **SELECT * FROM Events e JOIN Attendance a ON e.EId = a.EId WHERE a.UId = ?MyUId**
   Each user can view details of events they attend.
Coming Up Next...

1. Overview and goals.
2. Policy specification.
3. Policy enforcement.
4. Evaluation.
Query Compliance

A **compliant query** reveals only information exposed by the views.
Example: Calendar Application

1. **SELECT * FROM Users**
   
   Each user can view information on all users.

2. **SELECT * FROM Attendance**
   
   WHERE UId = ?MyUId
   
   Each user can view which events they attend.

3. **SELECT * FROM Events e**
   JOIN Attendance a
   ON e.EId = a.EId
   WHERE a.UId = ?MyUId
   
   Each user can view information on events they attend.

Users(UId, Name)
Events(EId, Title, Date)
Attendance(UId, EId)
Query Compliance, Formally

Given:

• A set $\mathcal{V}$ of views (for the current request context),
• A trace of query-result pairs: $(Q_1, O_1), \ldots, (Q_n, O_n)$,

We say query $Q$ is **compliant** if for every pair of databases $D, D'$:

Consistent with the observed trace

$$Q_i(D) = Q_i(D') = O_i \quad \forall 1 \leq i \leq n$$

Contain the same accessible information

$$V(D) = V(D') \quad \forall V \in \mathcal{V}$$

$$\Rightarrow Q(D) = Q(D')$$
Checking Compliance

Translate SQL into first-order logic via Codd’s Theorem + approximation.

$Q_i(D) = Q_i(D') = O_i$

$V(D) = V(D')$

$\downarrow$

$Q(D) = Q(D')$

Compliance definition

Logical formula

SMT solvers

SMT solvers can be slow!

100 ms per query $\sim$ seconds per page

28
Checking Compliance Fast

Compliance definition

\[
\frac{Q_i(D) = Q_i(D') = O_i}{V(D) = V(D') \downarrow Q(D) = Q(D')}
\]

Logical formula
SMT solvers
Decision cache

Cache only compliant queries
Naive Caching \(\sim\) Low Cache Hit Rate

Assuming fixed database schema and policy...

\[
\text{IsCompliant}(\mathcal{U}, \text{query}, \text{trace}) = \checkmark
\]

Each entry specific to user & URLs visited!
Blockaid **Generalizes** Compliance Determinations

\[
\text{IsCompliant}(\text{query, trace}) = \checkmark
\]

![Diagram](image-url)
How Are Templates Generated?

Find a **small set of constraints** on the request context, query, & trace that is sufficient to **guarantee compliance**

- Is \( \{C_1, C_2, C_3\} \) sufficient?
- **Unsat core**: \( \{C_2, C_3\} \)
Coming Up Next...

1. Overview and goals.
2. Policy specification.
3. Policy enforcement.
4. Evaluation.
Setup

- Implemented Blockaid as **JDBC driver** wrapping a database connection.
- Applied Blockaid to **three applications** (with hand-crafted policies).
- Measured **page load time** of 5 URLs per application.
- Modified **~20 — 100 lines** of code per application.
  - To fetch only data that **can be revealed** to the user.
Page Load Times (median)

*Original*: 20 ms — 450 ms

![Bar Chart]

- **Original**: Blue bars
- **Modified**: Green bars
- **Blockaid: cached**: Orange bars
- **Blockaid: no cache**: Red bars

The chart shows the median page load times for different blocks and sites, with original times ranging from 20 ms to 450 ms.
Page Load Times (median)

*Modified*: up to 6% overhead for all but one page

19% (4 ms) overhead
Page Load Times (median)

Cached: up to 12% overhead over modified

D1 D2 D3 D4 D5 S1 S2 S3 S4 S5 A1 A2 A3 A4 A5

Original Modified Blockaid: cached Blockaid: no cache
Page Load Times (median)

No cache: up to 236x overhead over modified
More Evaluation in the Paper

- Breakdown of page load times.
- Solver performance.
- Template generalization case study.
Blockaid
Data-access Policy Enforcement for Web Applications

• Compatible with existing frameworks + semantically transparent.
• Uses SMT to verify query compliance with view-based policy.
• Generalization-based caching through decision templates.

https://github.com/blockaid-project/