Precise, Dynamic Information Flow for Database-Backed Applications

Jean Yang, Travis Hance, Thomas H. Austin, Armando Solar-Lezama, Cormac Flanagan, and Stephen Chong

PLDI 2016
An oil skimming operation works in a heavy oil slick after the spill on April 1, 1989. (Photo from Huffington Post).
Oil-covered otter. (Photo from the Human Impact Project)
The Relationship Between Design and Accidents

Single hull

Crude oil

Required by the Oil Pollution Act of 1990.

Double hull

Crude oil
But what about information leaks?
Wanted: Double Hull for Information Security

Single hull

Double hull

Sensitive data

Sensitive data

Research in language-based security looks at designs for double hulls [Sabelfeld and Myers, JSAC 2003].

Our goal: make double hulls that are as easy to construct as possible!
This Talk: Making It Easier to Secure Web Programs

1. Why it’s hard to prevent information leaks.

2. A programming model that makes writing secure web programs easier.

3. How we support that programming model in database-backed applications.
Social Calendar Example

Let’s say Arjun and I want to throw a surprise paper discussion party for Emery.
Challenge: Different Viewers Should See Different Events

Guests

Surprise discussion for Emery at Chuck E. Cheese.

Emery

Pizza with Arjun/Jean.

Strangers

Private event at Chuck E. Cheese.
Policies May Depend on Sensitive Values

Must be on guest list.

Leaky enforcement: when the programmer neglects dependencies of policies on sensitive values.

Must be member of list and the list must be finalized.

Policy for event depends on policy for guest list!

Finalized list
A Story of Leaky Enforcement

1. We add Armando to non-final guest list.

   Guest List
   🗓 Finalized list

2. Armando sees the event on his calendar.

3. We run out of space and remove Armando.

4. Armando figures out he was uninvited.

   There was a party on my calendar...
A Story of Leaky Enforcement

We add Armando to non-final guest list.

Problem: implementation for event policy neglected to take into account guest list policy.

We run out of space and remove Armando.

Armando figures out he was uninvited.

There was a party on my calendar...

This arises whenever we trust programmers to get policy checks right!
Need to Track Policies and Viewers Across the Code

“What is the most popular location among friends 7pm Tuesday?”

Update to all calendar users

Need to track how information flows through derived values and where derived values flow!
“Policy Spaghetti” in HotCRP

Conditional permissions checks everywhere!
Jacqueline Web Framework to the Rescue!

1. Programmer specifies information flow policies separately from other functionality.

2. Runtime prevents information leaks according to policy annotations.

3. Enhanced runtime encompasses applications and databases, preventing leaks between the two.
Contributions

• *Policy-agnostic* programming model for database-backed web applications.

• Semantics and proofs for policy-agnostic programming that encompasses SQL databases.

• Demonstration of practical feasibility with Python implementation and application case studies.
Jacqueline Web Framework

Framework shows appropriate values based on viewer and policies.

Object-relational mapping propagates policies and sensitive values through computations.

Framework attaches policies based on annotations.
class Event(JacquelineModel):
    name = CharField(max_length=256)
    location = CharField(max_length=512)
    time = DateTimeField()
    description = CharField(max_length=1024)

@jacqueline
def has_host(self, host):
    return EventHost.objects.get(
        event=self, host=host) != None

@jacqueline
def has_guest(self, guest):
    return EventGuest.objects.get(
        event=self, host=host) != None

@staticmethod
@label_for('location')
def restrict_event(event, ctxt):
    return event.has_host(ctxt) or event.has_guest(ctxt)

@staticmethod
def jacqueline_get_private_location(event):
    return "Undisclosed location"
Centralized Policies in Jacqueline

Centralized policies! No checks or declassifications needed anywhere else!
Closer Look at the Policy-Agnostic Runtime


1. Runtime propagates values and policies.

```python
userCount = 0
if location == 'C:
    userCount += 1
return userCount
```

2. Runtime solves for values to show based on policies and viewer.

```
print { }
print { }
print { 1 } 0
print { }
```
Labels Track Sensitive Values to Prevent Leaks

Labels follow values through all computations, including conditionals and assignments.

Emery can’t see secret party information or results of computations on those values!
The Dangers of Interacting with Vanilla Databases

Database queries can leak information!

Challenge: Support faceted execution when interacting with an unmodified SQL database.

Need faceted queries!
Semantics of a Faceted Database

Too expensive! Too difficult to extend the formal semantics!

Conceptual row

<table>
<thead>
<tr>
<th>Primary key</th>
<th>Location</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td></td>
</tr>
</tbody>
</table>

Store facets as strings?

New database for each label?

select * from Users where location =

save( )
Solution: Use ORM to Map Facets onto Database Rows

Conceptual row

<table>
<thead>
<tr>
<th>Primary key</th>
<th>Location</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>α</td>
</tr>
</tbody>
</table>

ORM refacets

<table>
<thead>
<tr>
<th>Jeeves key</th>
<th>Location</th>
<th>Labels</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td></td>
<td>{a}</td>
</tr>
</tbody>
</table>

select * from Users
where location = α

<table>
<thead>
<tr>
<th>Jeeves key</th>
<th>Location</th>
<th>Labels</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td></td>
<td>{a}</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Primary key</th>
<th>Location</th>
<th>Labels</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td></td>
<td>NULL</td>
</tr>
</tbody>
</table>
# Supporting Queries in Jacqueline

<table>
<thead>
<tr>
<th>Jacqueline Supports</th>
<th>SQL Implements</th>
<th>ORM Implements</th>
</tr>
</thead>
<tbody>
<tr>
<td>get</td>
<td>select</td>
<td>refaceting</td>
</tr>
<tr>
<td>all</td>
<td>select</td>
<td>refaceting</td>
</tr>
<tr>
<td>filter</td>
<td>select</td>
<td>refaceting</td>
</tr>
<tr>
<td>sort</td>
<td>order by</td>
<td>refaceting</td>
</tr>
<tr>
<td>foreign keys</td>
<td>join</td>
<td>-</td>
</tr>
<tr>
<td>save</td>
<td>delete, insert</td>
<td>turning a faceted value into multiple rows</td>
</tr>
<tr>
<td>delete</td>
<td>delete</td>
<td>keeping track of which facets to delete</td>
</tr>
</tbody>
</table>

Can use SQL implementations for many queries!
Early Pruning Optimization

Observation: Framework can often (but not always) track viewer.

Optimization: Can often explore fewer possible paths!
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

Abstract
We present an approach for dynamic information flow control across the application and database. Our approach reduces the amount of policy code required, yields formal guarantees across the application and database, works with existing relational database implementations, and scales for realistic applications. In this paper, we present a programming model that factors out information flow policies from application code and database queries, a dynamic semantics for the underlying $\lambda^{JDB}$ core language, and proofs of termination-insensitive non-interference and policy compliance for the semantics. We implement these ideas in Jacqueline, a Python web framework, and demonstrate feasibility through three application case studies: a course manager, a health record system, and

1. Introduction
From social networks to electronic health record systems, programs increasingly process sensitive data. As information leaks often arise from programmer error, a promising way to reduce leaks is to reduce opportunities for programmer error.

A major challenge in securing web applications involves reasoning about the flow of sensitive data across the application and database. According to the OWASP report [42], errors frequently occur at component boundaries. Indeed, the difficulty of reasoning about how sensitive data flows through both application code and database queries has led to leaks in systems from the HotCRP conference management system [3] to the social networking site Facebook [47]. The patch for the recent HotCRP bug involves policy checks
Review: Traditional Non-Interference

Secret values should not affect public output.

if guest == 0:
    userCount += 1

print {}

if guest == 1:
    userCount += 1

Challenge:
Compute labels from program—may have dependencies on secret values!
Policy-Agnostic Non-Interference

\[
\text{if } \begin{cases} \text{guest} \end{cases} = \begin{cases} \text{pizza Hut} \end{cases} : \text{userCount} += 1
\]

\[
\text{if } \begin{cases} \text{guest} \end{cases} = \begin{cases} \text{Domino's Pizza} \end{cases} : \text{userCount} += 1
\]

Theorem:
All executions where \text{guest} must be \text{public} produce equivalent outputs.

Can’t tell apart secret values that require \text{guest} to be public.
Application Case Studies

Course manager

Health record manager

Conference management system (deployed!)

Jacqueline reduces the number of lines of policy code and has reasonable overheads!
Demo
Conference Management System Running Times

Tests from Amazon AWS machine via HTTP requests from another machine.

*Different from numbers in paper.
Summary: Policy-Agnostic Web Programming with Jacqueline

1. Programmer specifies information flow policies separately from other functionality.

2. Runtime prevents information leaks according to policy annotations.

3. Enhanced runtime encompasses applications and databases, preventing leaks between the two.

We have strong formal guarantees and evidence that this can be practical!
You *can* factor out information flow policies from other code to avoid policy spaghetti!

You *can* enforce policies across the application and database by using a carefully-crafted ORM!

You *can* build realistic systems using this approach!

http://jeeveslang.org
http://github.com/jeanqasaur/jeeves