A Language for Automatically Enforcing Privacy

Jean Yang

with Kuat Yessenov

and Armando Solar-Lezama
Displaying User Locations to Other Users

getLocation

Jean Yang @ POPL
def getLocation (user: User): Location = user.location
Only my friends can see my location.

Alice: Only my friends can see my location.

Simple policy:
- getLocation
- Owner
- Viewer

Secret club

Jean Yang @ POPL
Finer-Grained Policies

Only my friends can see my location.

Only members know this exists.

Policy interaction?

Which policies apply where?

Alice

Secret club

getLocation

Owner Locations

Viewer

Diner

Not a member!

Jean Yang @ POPL
def getLocation (user: User) (viewer: User) : Location = {
    if (isFriends user viewer) {
        if (canSee user.location viewer) {
            user.location;
        } else { scrub(user.location, "Diner"); }
    } else { undisclosedLocation; }
}

Policies

Views of sensitive values

Jean Yang @ POPL
Our Mission

Make it easier for the programmer to preserve confidentiality of user data.
What’s Hard?

Function

Scrubbed data

Function

Scrubbed data

Data

Functionality and policy are intertwined.

Programmer check/filter
Our Solution

- **Policy**
- **Data**
- **Separation of policies from functionality**

- Automatic enforcement
  - Function
    - Programmer check/filter
  - Tagged data
  - Function
    - Programmer check/filter
  - Scrubbed data
def getLocation (user: User) (viewer: User) : Location = {
    if (isFriends user viewer) {
        if (canSee user.location viewer) {
            user.location;
        } else {
            scrub(user.location, "Work");
        }
    } else {
        undisclosedLocation;
    }
}

State of the Art

Jeeves Goal

Policy + Functionality

Jean Yang @ POPL
Talk Outline

Jeeves language

How it works

Coding in Jeeves
Jeeves Language

Policy

Data

1. Sensitive values

2. Policies

3. Automatic contextual enforcement

Function

Tagged data

Scrubbed data

Jean Yang @ POPL
Jeeves for Locations

Low confidentiality  High confidentiality

Diner  Secret club

Jean Yang @ POPL
Using Jeeves

**Sensitive Values**

```scala
val a in { low, high }
val location: String = "school" | "MIT"
```

**Policies**

```
policy a: context != alice ➔ low
```

**Core Functionality**

```scala
val msg: String = "Alice is at " + location
```

**Contextual Enforcement**

```
print {alice} msg /* "Alice is at MIT" */
print {bob} msg /* "Alice is at school" */
```
Talk Outline

Jeeves language

How it works

Coding in Jeeves

Jean Yang @ POPL
How Jeeves Works

Constraints | Symbolic values

Symbolic evaluation

Function

Symbolic expressions

Function

Implicit parameter

SMT solving

Concrete value

Jean Yang @ POPL
Representing Sensitive Values in Jeeves

Without Jeeves

<table>
<thead>
<tr>
<th>Name</th>
<th>Location</th>
</tr>
</thead>
<tbody>
<tr>
<td>Alice</td>
<td>MIT</td>
</tr>
<tr>
<td>Bob</td>
<td>POPL</td>
</tr>
<tr>
<td>Claire</td>
<td>POPL</td>
</tr>
</tbody>
</table>

Jeeves

<table>
<thead>
<tr>
<th>Name</th>
<th>Location</th>
<th>Policy</th>
</tr>
</thead>
<tbody>
<tr>
<td>Alice</td>
<td>⟨ ?</td>
<td>MIT⟩ₐ</td>
</tr>
<tr>
<td>Bob</td>
<td>POPL</td>
<td></td>
</tr>
<tr>
<td>Claire</td>
<td>⟨ ?</td>
<td>POPL⟩ₐ</td>
</tr>
</tbody>
</table>
Symbolic Evaluation for Information Flow

<table>
<thead>
<tr>
<th>Name</th>
<th>Location</th>
</tr>
</thead>
<tbody>
<tr>
<td>Alice</td>
<td>$\langle I \rangle_a$</td>
</tr>
<tr>
<td>Bob</td>
<td>POPL</td>
</tr>
<tr>
<td>Claire</td>
<td>$\langle I \rangle_b$</td>
</tr>
</tbody>
</table>

How many people are at POPL?

$$1 + ((x_1 = \text{POPL}) \ ? \ 1 : 0) + ((x_2 = \text{POPL}) \ ? \ 1 : 0)$$

Outputs computed from sensitive values are symbolic & concretized under the policy environment.

Runtime Environment

```
context != alice $\Rightarrow$ a = low
... $\Rightarrow$ b = low
```
Jeeves Non-Interference Guarantee

Consider the sensitive value

\[ \langle \text{Low component} \mid \text{High component} \rangle^a \]

Given a fixed \( L \), all executions where \( a \) must be \textit{low} produce equivalent outputs no matter the value of \( H \).
Standard Non-Interference

Program

\( L \rightarrow H_1 \rightarrow H_2 \rightarrow H_{n-1} \rightarrow H_n \rightarrow a = \text{low} \)

\( \text{Does not depend on the H-value} \)

Output

\( \text{Does not depend on the H-value} \)
Jeeves
Non-Interference

Program

$H_1 \quad H_2 \quad H_n$

$L \quad a = \text{low} \quad a = \text{high}

\quad H_{n-1}

Output

Cannot distinguish between $H$-values that imply $a = \text{low}$

Depends on the $H$-value

Jean Yang @ POPL
Jeeves
Non-Interference

Program does not leak information about $H$. $a = \text{low}$
Language Restrictions

<table>
<thead>
<tr>
<th>Constraints</th>
<th>Symbolic values</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arithmetic and Boolean constraints with conditionals &amp; implications.</td>
<td>Function</td>
</tr>
<tr>
<td>No functions, quantifiers, or theory of lists.</td>
<td>Symbolic expressions</td>
</tr>
<tr>
<td>SMT solving</td>
<td>Concrete value</td>
</tr>
</tbody>
</table>

Primitives and objects. No functions.
Static Checks

Symbolic expressions

Function

SMT solving

Concrete value

Contexts are well-formed.

Outputs are concrete.

Constraints

Symbolic values

Symbolic evaluation

Symbolic values flow only where expected.

Evaluation does not introduce nontermination.

Jean Yang @ POPL
Stateful Policies

Only people near me can see my location.

policy a:
(distance context alice \geq \text{radius}) \rightarrow \text{low}

But Alice’s location is changing…

Jeeves: Delay policy evaluation until output.
Jeeves System

Jeeves runtime

Policies evaluated.

Function

Symbolic expressions

Evaluation produces well-formed values.

输出

Guarantee: outputs shown according to policies.
Scala Implementation

Overload operators to create symbolic expressions.

Use an SMT solver as a model finder.

Delay evaluation of policies until output.

Propagate policies.

Jean Yang @ POPL
Talk Outline

Jeeves language

How it works

Coding in Jeeves
JConf Architecture

Core Program
- Search papers.
- Display papers.
- Add and remove tags.
- Assign and submit reviews.

Does not need to know about policies.

Context
Viewer: User
CStage: Stage

Functionality
- Paper
  - Title
  - Author
  - Reviews
  - Tags
  - Policy

- Review
  - Reviewer
  - Content
  - Policy

- User
  - Role
  - Policy

Submission, review, rebuttal, decision, public

Policy
## Functionality vs. Policy

<table>
<thead>
<tr>
<th>File</th>
<th>Total LOC</th>
<th>Policy LOC</th>
</tr>
</thead>
<tbody>
<tr>
<td>ConfUser.scala</td>
<td>59</td>
<td>17</td>
</tr>
<tr>
<td>PaperRecord.scala</td>
<td>103</td>
<td>48</td>
</tr>
<tr>
<td>PaperReview.scala</td>
<td>21</td>
<td>11</td>
</tr>
<tr>
<td>ConfContext.scala</td>
<td>6</td>
<td>0</td>
</tr>
<tr>
<td>Backend</td>
<td>123</td>
<td>0</td>
</tr>
<tr>
<td>Frontend (Scalatra)</td>
<td>161</td>
<td>0</td>
</tr>
<tr>
<td><strong>Total</strong></td>
<td><strong>473</strong></td>
<td><strong>76</strong></td>
</tr>
</tbody>
</table>
Conclusions

The Jeeves language: pushing responsibility of privacy to the runtime.

How we designed a language with constraints using symbolic evaluation to provide execution guarantees.

Evaluation of Jeeves in practice: conference management example.

Website: sites.google.com/site/jeevesprogramming
Google Code: code.google.com/p/scalaSimt
Contact: jeanyang@mit.edu