Preventing Information Leaks with Jeeves

Jean Yang, MIT

April 8, 2015
Data, Data Everywhere

**Social media**

**Electronic health records**

**Online courses**

**Wearable devices**
All Kinds of People Are Writing All Kinds of Code

Open source lines of code

Medical researchers

Social scientists

Journalists

Children
Even Trained Developers Leak Information

Bang With Friends Isn't Actually That Anonymous

Facebook apps leak personal data, says Symantec

Vivino wine-lovers’ app leaked personal information

Not OK, Cupid: dating site email security gaffe leaves your account wide open
Why Aren’t Existing Approaches Enough?

Encrypting Data

But people are still showing the data wrong.

Defensive protection

But leaves system builders a step behind.

Jean Yang / Jeeves
My Approach: Privacy by Construction

Factor out privacy to reduce opportunity for leaks.

• Programmer specifies high-level policies about how sensitive data can be used.
• Rest of program is policy-agnostic.
• System manages policies automatically.
Alice and Bob throw a surprise party for Carol.
Even seemingly simple policies have subtleties.

Problem:

Even seemingly simple policies have subtleties.

- **Guests**
  - Surprise party for Carol at Chuck E. Cheese.
  - **Policy:** Must be guest.

- **Carol**
  - Pizza with Alice/Bob.
  - **Policy:** Only visible to hosts until finalized.

- **Strangers**
  - Private event at Chuck E. Cheese.

Policies can depend on sensitive values and other policies.
Enforcing Policies Can Leak Information!

Policy: Only visible to hosts until finalized.

Policy: Must be guest.

• Subtle mistake: check for policy 1 neglects dependency on policy 2.
• Problem arises when programmers trusted to get dependencies right.
Problem:

Policies Are Intertwined Across the Code

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

- Track information flow through derived values.
- Track where derived values flow.

Update to event subscribers
“Policy Spaghetti” in Real Systems

Highlighted: conditional permissions checks everywhere.

Code from HotCRP conference management system
Automatic Enforcement with Jeeves

The well-intentioned programmer writes same code no matter what policies are.
Jeeves Factors Out Policies

- Centralized policies.
- Policy-agnostic program.
- Runtime differentiates behavior.

```python
class event(Model):
    VISIBILITY = [(E, 'Everyone'), (S, 'Guests')]
    name = CharField(max_length=255)
    location = CharField(max_length=512)
    time = DateTimeField()
    description = CharField(max_length=1000)
    visibility = CharField(max_length=3, choices=VISIBILITY, default='E')

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

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

@staticmethod
def event_get_private_name(event):
    return 'Private event'

@staticmethod
def event_get_private_location(event):
    return 'Undisclosed location'

@staticmethod
def event_get_private_time(event):
    return datetime.now(tz=pytz.utc)

@staticmethod
def event_get_private_description(event):
    return 'An event.'

@staticmethod
def event_for(name, location, time, description):
    @jidees
def event_restrict_event(event, ctx):
        if event.visiblity == 'G':
            return event.has_host(ctx) or event.has_guest(ctx)
        else:
            return True

def register_account(request):
    if request.user.is_authenticated():
        return HttpResponseRedirect("index")

    post_method = "POST"
    form = UserCreationForm(request.POST)
    form.is_valid():
        user = form.save()
        user.save()
    UserProfile.objects.create(
        username=request.POST.get('username'),
        email=request.POST.get('email'),
    )

# View
@login_required
@request_wrapper
def jidees

    if request.method == 'POST':
        host_profile = UserProfile(request.POST.get('host_profile'))
        guest_events = EventGuest.objects.get(host_profile=host_profile)

        return ('profile.html',
            "profile": profile,
            "is_uor_profile": request.user.username == user_profile.username,
            "host_events": host_events,
            "guest_events": guest_events,
            "view_page": "profile",
        )
```

Jean Yang / Jeeves
Background.
Automatically Managing Information Flow

1. Specifying policies.
2. Specifying differentiated behavior.
3. Factoring out policies from other code.
4. Enforcing policies that depend on sensitive values.

Jeeves supports more expressive policies than prior work.

Only Jeeves supports 3 and 4.
Related Work:

Checking Information Flow

Programmer tags data.

Information flow controls check that only allowed flows occur [HiStar, Flume, Resin, LIO/Hails, Jif/Sif, flow locks, F*, and more].

My previous work.
Related Work:
Computing with Multiple Values

Must encode policies as tags.
- “Policy spaghetti.”
- Policies can leak information.

Multi-execution approaches differentiate behavior based on viewer [secure multi-execution (Devriese et al); faceted execution (Austin and Flanagan)].
Jeeves programming model.

**Challenges**

1. Factoring out information flow.
2. Supporting expressive policies that can depend on sensitive values.
Policies describe rules for how values may flow to output contexts.

Sensitive values encapsulate multiple behaviors.

Application Code
Separate from policies.
Jeeves Supports Expressive Policies

A policy is an arbitrary function that takes the output context and returns a Boolean value.

```python
def isNotCarol(oc):
    return oc != Jean Yang
```

Output context can be of arbitrary type.

Policies can depend on sensitive values.

```python
def isGuest(oc):
    return oc in .guests
```

Jean Yang / Jeeves
Jeeves Programming Model

1. Programmer specifies policies and facets.
2. Programmer writes policy-agnostic programs.
3. Runtime propagates values and policies.
4. Runtime produces differentiated output based on the viewer.

```
print { [people] }
```

true

```
print { [woman] }
```

false
The Jeeves Programming Model

• Well-defined runtime semantics for policy-agnostic programming with information flow policies.

• Can be implemented standalone or embedded as a library.

• Has been adapted across runtimes in web frameworks.
How Jeeves works.

Challenges

1. Handling policies with dependencies.
2. Enforcing policies when state changes.
Jeeves Execution Model

Runtime simulates simultaneous multiple executions.

1. Runtime propagates values and policies.

```
x = 0

if x == 1:
x += 1
return x
```

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

```
print { Girls }
print 1
```

```
print { Girl }
print 0
```
Using Policies to Produce Outputs

Jeeves uses policies to defacet appropriately.

```python
def isNotCarol(oc):
    return oc !=
```

```
print { }
```

```
policy()
```

```
( != ) ? 1 : 0
```

```
0
```
def isMaybeCarol(oc):
    return oc ==

def policy(oc):
    print { }

But What About Dependencies?

Possible solutions:

Jeeves runtime will pick the secret value if allowed.

Need to find a fixed point!
Using Constraints to Handle Dependencies

Evaluated with respect to state at time of output.

\[ a = \text{secret} \Rightarrow \text{in .guests} \]
\[ a = \text{secret} \Rightarrow \text{false} \]
\[ \neg (a = \text{secret}) \]

<table>
<thead>
<tr>
<th>Label</th>
<th>Policy</th>
</tr>
</thead>
</table>
| a     | `def` isGuest(oc):
|       | return oc in .guests |

\[ a \in \{\text{secret, public}\} \]

- Constraints contain only Boolean variables.
- Always a consistent assignment.
Handling Indirect Flows

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

```python
if a == true:
    x += 1
elif a == false:
    x += 1
x = x新征程 + 1
```
# Jeeves Runtime Semantics: Outputting to Sinks

## Statement evaluation

\[ \Sigma, S \downarrow V_p, oc: R \]

## Expression evaluation

\[ \Sigma, E \downarrow_{pc} \Sigma', V \]

### Evaluate output context and expression to print.

\[ \Sigma, E_{oc} \downarrow \emptyset \Sigma_{oc}, V_{oc} \]
\[ \Sigma_{oc}, E_r \downarrow \emptyset \Sigma_r, V_r \]

### Retrieve labels and policies.

\[ \{k_1 \ldots k_n\} = \text{closeK} (\text{labels}(E_{oc}) \cup \text{labels}(E_r), \Sigma_2) \]
\[ E_p = \lambda x. \text{true} \land \ldots \land \Sigma_2 (k_n) \]

### Evaluate policies applied to the output context.

\[ \Sigma_r, (E_p V_{oc}) \downarrow \emptyset \Sigma_p, V_p \]

### Defacet using satisfying policy assignment.

pick \( pc \) such that \( pc(V_{oc}) = oc, pc(V_r) = R, pc(V_p) = \text{true} \)

\[ \Sigma, \text{print} \{E_{oc}\} E_r \downarrow V_p, oc: R \]

---

Jean Yang / Jeeves
Theoretical guarantees.

Challenges

1. Defining what it means to enforce policies.

2. Proving enforcement with respect to our semantics.
Non-Interference

Secret values should not affect public output.

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

```python
if x += 1 == :
   print { }
```

Jeeves Theorem:
All executions where a must be public produce equivalent outputs.
Can’t tell apart secret values that require a to be public.
Jeeves Policy Compliance Theorem

Suppose we have

\[ S_1 = \text{print } \{e\} \ C[ E_1^a E_P ] \quad S_2 = \text{print } \{e\} \ C[ E_2^a E_P ] \]

Where \( a \) must be public.

Then the set of values \( S_1 \) evaluates to is equivalent to the set of values \( S_2 \) evaluates to.

\[ \{f: R \mid \exists V, \emptyset, S_1 \downarrow V, f: R\} = \{f: R \mid \exists V, \emptyset, S_2 \downarrow V, f: R\} \]

Need to account for policies that depend on sensitive values!
Challenges

1. Adapting the approach for web applications.

2. Scaling the approach.

Implementation and evaluation.
Facets in the Application and Database

Application

Queries

select * from Users where location =

Database queries can leak information!

SQL Database

Application

All data

select * from Users

Impractical and potentially slow!

SQL Database

Solution: Facet the database. Implemented as an object-relational mapping supporting uniform facet representation.
Jacqueline Web Framework

Jeeves runtime

Application

Frontend

Defacet values based on viewer.

Propagate policies.

Attach policies.

Dashboard

Viewer

Database

@jeeves

Programmer is responsible

Framework is responsible

Jean Yang / Jeeves
Django-like data schema for describing fields.

```python
class Event(Model):
    VISIBILITY = (('E', 'Everyone'), ('G', 'Guests'))

    name = CharField(max_length=256)
    location = CharField(max_length=512)
    time = DateTimeField()
    description = CharField(max_length=1024)
    visibility = CharField(max_length=1, choices=VISIBILITY, default='E')
```

Helper functions for policy include queries.

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

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

Policy for ‘location’ field.

```python
@staticmethod
@label_for('name', 'location', 'time', 'description')
@jeeves
def jeeves_restrict_event(event, ctxt):
    if event.visibility == 'G':
        return event.has_host(ctxt) or event.has_guest(ctxt)
    else:
        return True
```

Public value for ‘location’ field.

```python
@staticmethod
def jeeves_get_private_location(event):
    return "Undisclosed location"
```
Questions
1. How does faceted execution scale?
2. How does faceting the database affect queries?
Demo
Database size does not affect query speed.

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

Showing sensitive data grows linearly.
Evaluation Lessons

• Faceted execution is expensive.
• Policy-agnostic programming does not always need facet execution.
• Many opportunities for optimization in web frameworks.
Policy-Agnostic Programming in Jeeves

Design of a policy-agnostic programming language [POPL '12]

Sensitive values

Policies

Semantics and guarantees [PLAS '13]

Other functionality

Implementation, web framework, and case studies [in submission]
My Interests in Provable Guarantees

**Lessons**

- Difficult to write secure and private programs.
- Even more difficult to prove them correct.
- Easier if we can get privacy by construction.

---

Verve: a provably type-safe OS.
[PLDI '10 (Best Paper); CACM '10]

F*: type-checking security properties.
[ICFP '11; JFP '13]

Ask Reeves: verified privacy policies.
[patent filed]
The Future of Policy-Agnostic Programming

- Factor out policies from other code.
- Rely on compiler and runtime to differentiate behavior.
Making Policy-Agnostic Programming Work

• **Programmability.**
  – Languages for different kinds of policies.
  – Tools for verifying and visualizing policies.

• **Scaling.**
  – Policy-agnostic semantics with minimal faceting.
  – Building foundations with policy management in mind.

Web frameworks  Databases  Web browsers
Parting Thoughts

By reducing opportunity for programmer error, we can eliminate whole classes of privacy leaks.