Constructive Security
Using Information Flow Control

Andrew Myers
Cornell University
What is computer security?

- *Past*: can an attacker control my computer?
- *Future*: do networked systems sharing information provide security and privacy despite limited trust?
  - web applications, mashups
  - social networking platforms
  - medical information systems
  - government information systems
  - supply chain management
  - the Internet
Amazon.com Privacy Notice:

...We reveal only the last four digits of your credit card numbers when confirming an order. Of course, we transmit the entire credit card number to the appropriate credit card company during order processing.

...third-party Web sites and advertisers, or...advertising companies...sometimes use technology to send...Advertisements that appear on our Web site directly to your browser. They automatically receive your IP address.

...Examples of the information we collect and analyze include...[IP]address used to connect your computer to the Internet; login; e-mail address; password; computer and connection information...plug-in types and versions, operating system, and platform; purchase history...the full [URL]clickstream...the phone number you used to call our 800 number...cookies...we may use...JavaScript to measure and collect session information, including...scrolling, clicks, and mouse-overs.

...Sometimes we send offers to selected groups of Amazon.com customers on behalf of other businesses. When we do this, we do not give that business your name and address. If you do not want to receive such offers, please adjust your Customer Communication Preferences.

...We release account and other personal information when...appropriate to comply with the law; enforce or apply our Conditions of Use and other agreements; or protect the rights, property, or safety of Amazon.com, our users, or others.

...Lots of promises about confidentiality and integrity...
How does Amazon know this evolving system containing many nodes, code from many sources meets their legal obligations?
Cooperation with distrust

- *Past:* can an attacker control my computer?
- *Future:* do networked systems sharing information provide security and privacy despite limited trust?
  - web applications, mashups
  - social networking platforms
  - medical information systems
  - government agencies
  - supply chain management
  - the Internet
Security: bridges vs. software

• Bridges fail rarely (post-arch)
  – Assurance derived from construction process

• Software violates security/privacy (frequently)
  – Assurance is weak at best
  – Much “destructive” security research
Constructive security?

- **Idea**: build secure systems with:
  - explicit, declarative security policies capturing security requirements
  - higher-level language-based abstractions
- Compiler, runtime *automatically* employ mechanisms to achieve security and performance
  - synthesizing implementation-level mechanisms (access control, partitioning, replication, encryption, signatures, logging, …)
- Security by construction!
Language-based security

- Developer writes code in a safe language (e.g., Jif) with explicit security policies
- Software construction process checks policies are enforced, adds run-time enforcement mechanisms
- Can verify target code to ensure policy enforcement
- Policies exposed for checking against rest of system at load time and run time

Diagram:
- Source code
- Policy
- Static checking
- Transformation
- Target code
- Policy
  - Synthesized security mechanisms
- Compare policies
- External policies
- Running code
  - Compare policies
Policies and end-to-end security

• System-wide, end-to-end enforcement of policies for information security $\Rightarrow$ need *compositional* policies

  ![Diagram showing code modules, network nodes, services, ...]

• **Information flow** policies on interfaces constrain end-to-end behavior
  $\Rightarrow$ are compositional
  $\Rightarrow$ enable raising the level of abstraction
Plan

1. **Jif**: Java + information flow control

2. **Swift**: synthesizing secure web applications

3. **Fabric**: a distributed platform for secure computation, sharing, and storage
Jif: A security-typed language

• Jif = Java + information flow control [POPL99]
  – Types include explicit (but simple) security policies
  – Enforcement: compile-time and run-time

• Trust and access control: principals and authority

• Information flow: decentralized labels
Principals in Jif

A principal is an abstraction of authority and trust

- represents users, groups, roles; privileges; access rights; host nodes and other system components.
- acts-for relation $p \succeq q$ means $p$ can do whatever $q$ can. “$q$ trusts $p$”. (related to speaks-for in authentication logic [e.g., ABLP93])

Top, bottom principals:
“acts for everyone” = $\top \succeq p \succeq \bot$ = “acts for no one”

Principals form a lattice with meet ($\wedge$) and join ($\vee$).
Programming with authority

- Code can run with the authority of a principal.

```java
class C authority(Alice) {
  int m() where authority(Alice) {
    f(); // use authority of Alice
  }
  int f() where caller(Alice) { ... }
}
```

- Can be used to implement access control
Decentralized labels

• Confidentiality policies: $u \rightarrow p$
  – $u$ is the **owner** of the policy (a principal), $p$ is a **reader**
  – meaning: $u$ trusts $p$ to learn information and not leak it
  – e.g., **Bob → Alice** means Bob trusts Alice (and Bob) to
    learn information about the data

• Integrity policies: $u \leftarrow p$
  – meaning: $u$ trusts $p$ not to influence the information in a
    way that damages it
  – $p$ is a **writer** of the information

• Decentralized label: set of owned policies
  e.g., \{Alice→Bob; Alice ← Alice\}
Decentralized label space

- Application-specific downgrading is needed by real applications
- Dangerous, so controlled in Jif by requiring authority (trusted code only) and integrity (for robust declassification)
Information security policies as types

- Confidentiality labels:
  ```
  int{Alice→Bob} a;
  "Alice says only Bob (&Alice) can learn a"
  ```

- Integrity labels:
  ```
  int{Alice←Alice} a;
  "Alice says only Alice can affect a"
  ```

Combined:
```
int{Alice→Bob ; Alice←} a;
```

- End-to-end static checking of flow $L_1 \rightarrow L_2$:
  $$L_1 \sqsubseteq L_2 ?$$

<table>
<thead>
<tr>
<th>Insecure</th>
<th>Secure</th>
</tr>
</thead>
<tbody>
<tr>
<td>( int{Alice\rightarrow} \ a_1, a_2; )</td>
<td>( b = a_1; )</td>
</tr>
<tr>
<td>( int{Bob\leftarrow} \ b; )</td>
<td>( a_1 = a_2; )</td>
</tr>
<tr>
<td>( int{Bob\leftarrow Alice} \ c; )</td>
<td>( b = c; )</td>
</tr>
<tr>
<td>&quot;Bob says only Alice (&amp; Bob) can affect c&quot;</td>
<td>( a_1 = b; )</td>
</tr>
</tbody>
</table>

But: ok if Alice $\geq$ Bob
Information flow control as type checking

- Jif label checking is type checking in a nonstandard type system: compositional!
- End-to-end security: noninterference (termination-insensitive)
  - caveat: proved for simplified models
  - challenges: objects, dynamic labels and principals, dependent types, parameterized types, exceptions, ...
Secure web applications?

- Ubiquitous, important, yet insecure
  - Cross-site scripting, SQL injection, information leakage, etc.

- Development methods make security assurance hard
  - Distributed system in multiple languages
    - Client: CSS, XHTML, JavaScript, Flash
    - Server: PHP, ASP, Ruby, SQL
  - Ajax/Web 2.0: Complex JavaScript UIs generating HTTP requests

*Symantec Internet Security Threat Report 2007*
Swift

- A programming system that makes secure, interactive web applications easier to write [SOSP 07]
- A higher-level programming model: one program in one language automatically split by the compiler
- Security by construction:
  - automatically partitioning code and data based on decentralized labels
- Automatic performance optimization
Guess-the-number

Random number between 1 and 10

Secret Number: 7

Tries: 3

Take a Guess!

(You have 3 chances)
Guess-the-number

Secret Number: 7

Bounds Check
Compare Guess

Tries: 3

6
Try Again

12
Out of range

4
Try Again

7
You win $500

Take a Guess!
(You have 3 chances)

Constructive Security Using Information Flow Control/Andrew Myers, Cornell University
Guess-the-number

Confidentiality Requirement

Integrity Requirement

Bounds Check

Secret Number: 7

Tries: 3

Tries: 10

Take a Guess!
(You have 3 chances)

I win $500

Buggy draw $500

malicious

Trusted

Integrity Requirement

Constructive Security Using Information Flow Control/Andrew Myers, Cornell University
A secure optimal split

Take a Guess!
(You have 3 chances)

Bounds Check

Secret Number: 7

Tries: 3

Bounds Check

Compare Guess

6
Try Again

12
Out of range

4
Try Again

7
You win $500

Constructive Security Using Information Flow Control/Andrew Myers, Cornell University
```swift
int secret;
int tries;
...
void makeGuess (int guess))
{
    if (guess >= 1 && guess <= 10) {
        …
    } else {
        message.setText("Out of range:" + guess);
    }
}
```
```java
int secret;
int tries;
...

void makeGuess (int guess) {

    if (guess >= 1 && guess <= 10) {
        boolean correct = (guess == secret);

        if (tries > 0 && correct) {
            finishApp("You win $500!");
        }
    }

    else {
        message.setText("Out of range:" + guess);
    }
}
```

Swift Guess-the-number

- **compare with stored secret**
- **successful guess**
int secret;
int tries;
...
void makeGuess (int guess)
{
    if (guess >= 1 && guess <= 10) {
        boolean correct = (guess == secret);
        if (tries > 0 && correct) {
            finishApp("You win $500!");
        } else {
            tries--;
            if (tries > 0)
                message.setText("Try again");
            else
                finishApp("Game over");
        }
    } else {
        message.setText("Out of range:" + guess);
    }
}
int secret;
int tries;
...

void makeGuess (int guess)
{
    if (guess >= 1 && guess <= 10) {
        boolean correct = (guess == secret);

        if (tries > 0 && correct) {
            finishApp("You win $500!");
        } else {
            tries--;
            if (tries > 0)
                message.setText("Try again");
            else
                finishApp("Game over");
        }
    } else {
        message.setText("Out of range:" + guess);
    }
}
int secret;
int tries;
...

void makeGuess (int guess)
{
    if (guess >= 1 && guess <= 10) {
        boolean correct = (guess == secret);

        if (tries > 0 && correct) {
            finishApp("You win $500!");
        } else {
            tries--;
            if (tries > 0)
                message.setText("Try again");
            else
                finishApp("Game over");
        }
    } else {
        message.setText("Out of range: " + guess);
    }
}
Security policies

- Swift adds two built-in principals: server, client
- Application can define more principals (Alice, Bob, ...)

\[
\begin{align*}
\text{Alice} \rightarrow \text{Bob} & \quad = \text{Alice permits Bob to learn info} \\
\text{Alice} \leftarrow \text{Bob} & \quad = \text{Alice permits Bob to affect info}
\end{align*}
\]

```swift
int { server → server ; server ← server } secret;
int { server → client ; server ← server } tries;
int { server → client } display;

display = secret;
```

Rejected at compile time
```java
int{server→server; server←server} secret;
int{server→client; server←server} tries;
...
{
    endorse (guess, {server←client} to {server←server})
    if (guess >= 1 && guess <= 10) {
        boolean correct = (guess == secret);
        {server→server} to {server→client});
        if (tries > 0 && correct) {
            finishApp("You win $500!");
        } else {
            tries--;
            if (tries > 0)
                message.setText("Try again");
            else
                finishApp("Game over");
        }
    } else {
        message.setText("Out of range:" + guess);
    }
}
```
Swift architecture

- Jif source code
- Confidentiality/Integrity labels
- label projection
- WebIL code
- partitioning
- Located WebIL code
- Server/Client Placement
- Java client code
- Javascript client code
- Swift client runtime
- GWT runtime library
- Java servlet framework
- Swift server runtime
- Java server code

Web Browser

Web Server

Constructive Security Using Information Flow Control/Andrew Myers, Cornell University
Swift architecture
### Labels \rightarrow \text{placement constraints}

<table>
<thead>
<tr>
<th>{Alice \rightarrow Bob, Dave}</th>
<th>{Alice \rightarrow Bob; Alice \leftarrow Bob}</th>
</tr>
</thead>
<tbody>
<tr>
<td>{Eve \leftarrow Chuck, Alice}</td>
<td>{Chick \leftarrow Chuck, Alice}</td>
</tr>
<tr>
<td>{George \rightarrow Bob, Dave}</td>
<td>{Chuck \leftarrow Bob, Alice}</td>
</tr>
<tr>
<td>{Dave \rightarrow Bob, Heather}</td>
<td>{Chuck \leftarrow Chuck, Alice}</td>
</tr>
<tr>
<td>{Chuck \leftarrow Bob, Alice}</td>
<td>{Chuck \leftarrow Bob, Alice}</td>
</tr>
<tr>
<td>{George \rightarrow Bob, Dave}</td>
<td>{Chuck \leftarrow Bob, Alice}</td>
</tr>
<tr>
<td>{Fiona \rightarrow Bob, Eve, Alice; Bob \leftarrow Fiona}</td>
<td>{Chuck \leftarrow Chuck, Alice}</td>
</tr>
<tr>
<td>{Alice \rightarrow Bob, Dave}</td>
<td>{Chuck \leftarrow Chuck, Alice}</td>
</tr>
<tr>
<td>{Dave \rightarrow Bob, Heather}</td>
<td>{Chuck \leftarrow Bob, Alice}</td>
</tr>
<tr>
<td>{p \rightarrow Bob, q; n}</td>
<td>{p \leftarrow p}</td>
</tr>
</tbody>
</table>

#### Client or Server

- Client can read
  - Low confidentiality
  - \( L \subseteq \{ \top \rightarrow \text{client} \} \)

- Client cannot read
  - High confidentiality
  - \( D \not\subseteq \{ \top \rightarrow \text{client} \} \)

#### Server or Client

- Server only
  - \( S \)

- Client only
  - \( S \)

- Server and maybe client
  - \( S \)
Labels → placement constraints

- Client can read (low confidentiality)
- Client cannot read (high confidentiality)

(high integrity)  (low integrity)

Client can write  Client cannot write

S?C?    ShC?

S    Sh
More placement constraints

Security constraints
- S?C?
- ShC?

Architectural constraints

Database library calls

UI widget calls
int secret;
int tries;

void makeGuess (int guess)
{
    if (guess >= 1 && guess <= 10) {
        boolean correct = (guess == secret);
        if (tries > 0 && correct) {
            finishApp("You win $500!");
        }
    } else {
        tries--;
        if (tries > 0)
            message.setText("Try again");
        else finishApp("Game over");
    }
}

message.setText("Out of range:" + guess);
```java
int secret;
...  
void makeGuess(int guess) {
  if (guess >= 1 && guess <= 10) {
    boolean correct = guess == secret;
    if (tries > 0 && correct) {
      finishApp("You win $500!");
    } else {
      tries--;
      if (tries > 0)
        message.setText("Try again");
      else finishApp("Game over");
    } else {
      message.setText("Out of range:" + guess);
    }
  } else {
    message.setText("Out of range:" + guess);
  }
}
```
Evaluation: functionality

Guess-the-Number
142 lines

Poll
113 lines

Secret Keeper
324 lines

Shop
1094 lines

Treasure Hunt
92 lines

Auction
502 lines
## Evaluation: network messages

<table>
<thead>
<tr>
<th>Example</th>
<th>Task</th>
<th>Actual</th>
<th>Optimal</th>
</tr>
</thead>
<tbody>
<tr>
<td>Guess-the-Number</td>
<td>guessing a number</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>Shop</td>
<td>adding an item</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Poll</td>
<td>casting a vote</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>Secret Keeper</td>
<td>viewing the secret</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>Treasure Hunt</td>
<td>exploring a cell</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>Auction</td>
<td>bidding</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>
Related work

- **Unified web programming models**
  - Links [CLWY06]
  - Hop [SGL06]
  - Hilda [YGQDGS07, YSRG 06]

- **Web application security**
  - Static analysis [HYHTLK 04, LL05, X06, XA06, JKK06]
  - Information flow via dynamic taint tracking [HO05, NGGE05, XBS06, CVM07]

- **Security by construction**
  - Jif/split [ZZNM02, ZCMZ03] and provably sound impls of partitioning [FR08, FGR09]
  - Fairplay [MNPS04]
  - SMCL [NS07]
Swift summary

- Web applications with security assurance by construction
  - cleaner, higher-level programming model
  - enabled by declarative security annotations
  - automated enforcement $\Rightarrow$ greater security assurance
  - security-constrained optimization

- What about more general distributed computation?
Decentralized sharing?

- **Federated systems** integrate data and computation across administrative boundaries
  - can add functionality, increase automation
  - Web is federated but not very programmable
  - Need **security and consistency**
Fabric: a system and a language [SOSP 09]

- Goal: a undergraduate can write secure, reliable programs for the Internet Computer
- All information (persistent or otherwise) looks like an ordinary program object
- Objects connected by references
  - Any object can be referenced uniformly from anywhere
  - References look like ordinary object pointers but can cross nodes and trust domains

Compiler and runtime enforce security and consistency despite distrust
Fabric enables federated sharing
Example: Filling a prescription

Order medication

Verify prescription

Check for conflicts

Get current medications

Pharmacist

Psychiatrist

General Practitioner

Constructive Security Using Information Flow Control/Andrew Myers, Cornell University
Example: Filling a prescription

**Security issues**
- Pharmacist shouldn’t see entire record
- Psychiatrist doesn’t fully trust pharmacist with update
  - Need secure distributed computation

**Consistency issues**
- Need atomicity
- Doctors might be accessing medical record concurrently

- **Pharmacist**
  - Fill order
  - Update inventory
  - Mark prescription as filled

- **Psychiatrist**
  - Update inventory

- Must be done by pharmacist
- Must be done by psychiatrist
Order orderMed(PatRec psyRec, PatRec gpRec, Prescription p) {
    atomic {
        if (!psyRec.hasPrescription(p)) return Order.INVALID;
        if (isDangerous(p, gpRec.getMeds())) return Order.DANGER;
    }
    Worker psy = psyRec.getWorker();
    psyRec.markFilled@psy(p);
    updateInventory(p);
    return Order.fill(p);
}
Order orderMed(PatRec psyRec, PatRec gpRec, Prescription p) {
  atomic {
    if (!psyRec.hasPrescription(p)) return Order.INVALID;
    if (isDangerous(p, gpRec.getMeds())) return Order.DANGER;

    Worker psy = psyRec.getWorker();
    psyRec.markFilled@psy(p);
    updateInventory(p);
    return Order.fill(p);
  }
}
Fabric: a high-level language

Order orderMed(PatRec psyRec, PatRec gpRec, Prescription p) {
  atomic {
    if (!psyRec.hasPrescription(p)) return Order.INVALID;
    if (isDangerous(p, gpRec.getMeds())) return Order.DANGER;
  }

  Worker psy = psyRec.getWorker();
  psyRec.markFilled@psy(p);
  updateInventory(p);
  return Order.fill(p);
}
Remote calls

Order orderMed(PatRec psyRec, PatRec gpRec, Prescription p) {
    atomic {
        if (!psyRec.hasPrescription(p)) return Order.INVALID;
        if (isDangerous(p, gpRec.getMeds())) return Order.DANGER;

        Worker psy = psyRec.getWorker();
        psyRec.markFilled@psy(p);
        updateInventory(p);
        return Order.fill(p);
    }
}

Remote call — pharmacist runs code at psychiatrist’s node
Federated transactions

```java
Order orderMed(Patient psyRec, Patient gpRec, Prescription p) {
    atomic {
        if (!psyRec.hasPrescription(p)) return Order.INVALID;
        if (isDangerous(p, gpRec.getMeds())) return Order.DANGER;

        Worker psy = psyRec.getWorker();
        psyRec.markFilled@psy(p);
        updateInventory(p);
        return Order.fill(p);
    }
}
```
Fabric security model

• Decentralized system
  – *anyone* can join
  – No centralized enforcement

• Decentralized security principle:
  – *You can’t be hurt by what you don’t trust*
Security labels in Fabric

<table>
<thead>
<tr>
<th>Confidentiality:</th>
<th>Alice → Bob</th>
<th>Alice permits Bob to learn</th>
</tr>
</thead>
<tbody>
<tr>
<td>Integrity:</td>
<td>Alice ← Bob</td>
<td>Alice permits Bob to affect</td>
</tr>
</tbody>
</table>

```java
class Prescription {
    Drug{Psy→A_{pharm} ; Psy←Psy} drug;
    Dosage{Psy→A_{pharm} ; Psy←Psy} dosage;
    ...
}
```

- Compiler and runtime together ensure policies are not violated by any information flows in system.
Trust management

• Fabric principals are objects

```java
class Principal {
    boolean delegatesTo(principal p);
    void addDelegatesTo(principal p) where caller (this);
    ...
}
```

• Explicit trust delegation via method calls

```java
// Assert “Alice acts-for Bob”
bob.addDelegatesTo(alice)
```

– Compiler and run-time ensure that caller has proper authority
Fabric abstraction

- Fabric language combines:
  - Information flow policy annotations
  - Remote calls
  - (Optimistic) nested atomic transactions

- Fabric system is a decentralized platform for secure, consistent sharing of information and computation
  - Nodes join freely
  - No central control over security

How to build a system that implements this abstraction?
Fabric Architecture

- **Worker nodes** (Workers)
  - Transaction
  - Remote call

- **Storage nodes** (Stores)
Fabric Architecture

- **Storage nodes** securely store persistent objects.
- Each object specifies its own security policy, enforced by store.

**Worker nodes** (Workers)
Fabric Architecture

- **Worker nodes** compute on cached objects
- Computation may be distributed across workers in **federated transactions**

- **Storage nodes** securely store persistent objects
- Each object specifies its own security policy, enforced by store
Fabric Architecture

- **Worker nodes** compute on cached objects
- Computation may be distributed across workers in **federated transactions**
- **Dissemination nodes** cache signed, encrypted objects in peer-to-peer distribution network for high availability
- **Storage nodes** securely store persistent objects
- Each object specifies its own security policy, enforced by store

• Dissemination nodes cache signed, encrypted objects in peer-to-peer distribution network for high availability

• Worker nodes compute on cached objects

• Computation may be distributed across workers in federated transactions

• Storage nodes securely store persistent objects

• Each object specifies its own security policy, enforced by store
Fabric run-time system

- Nodes are principals in Fabric language

- Root of trust: X.509 certificates bind hostnames to node principal objects
  - Store `getStore(String hostname)` checks certificate
  - Nodes act for principals stored at them.
Secure data placement

- Placing objects with label $L$ securely: is node $n$ trusted to enforce label $L$?

  \[
  \text{new \texttt{Foo}@n(\ldots)} \quad \rightarrow \quad \text{Static check} \quad \{T \leftarrow n; \ T \rightarrow \} \geq L
  \]

- **Trust ordering** $\geq$ on labels lifts principal acts-for ordering $\geq$ to relate information flow policies.

\[
\begin{aligned}
\text{secure information flow} & \quad \sqsubseteq \\
\text{confidentiality} & \quad \{T \rightarrow T; \ \bot \leftarrow \bot\} \\
\text{integrity} & \quad \{\bot \rightarrow \bot; \ T \leftarrow T\} \\
\text{trust} & \quad \sqsubseteq \\
\text{labels} & \quad \{T \rightarrow T; \ T \leftarrow T\}
\end{aligned}
\]
Secure remote calls

Is callee trusted to see call?
• Call itself might reveal private information
• Arguments might be private

Is caller trusted to make call?
• Caller might lack sufficient authority to make call
• Method arguments might have been tampered with by caller

Is callee trusted to execute call?
• Result might have been tampered with by callee

Is caller trusted to see result?
• Call result might reveal private information

Result: secure information flow enforced end-to-end across network
and more mechanisms...

- Writer maps for secure propagation of updates
- Automatic ‘push’ of updated objects to dissemination layer
- In-memory caching of object groups at store
- Object-group clustering and prefetching
- Caching and incrementally updating acts-for relationships
- Secure distributed transaction logs
- Hierarchical two-phase commit protocol

(see the SOSP’09 paper)
Implementation

- Fabric prototype implemented in Java and Fabric
  - Total: 35 kLOC
  - Compiler translates Fabric into Java
    - 15 k-line extension to Jif compiler using Polyglot [NCM03]
  - Dissemination layer: 1.5k-line extension to FreePastry
    - Popularity-based replication (à la Beehive [RS04])
  - Store uses BDB as backing store
Object overheads

- Extra overhead on object accesses at worker
  - Run-time label checking
  - Logging reads and writes
  - Cache management (introduces indirection)
  - Transaction commit
- Overhead at store for reads and commits
- Ported non-trivial web application to evaluate performance: a course management system.
CMS experiment

- CMS has been used at Cornell since 2004
  - Over 2000 students in over 40 courses
- Two prior implementations using SQL database:
  - J2EE/EJB2.0 (production system) [BCCDGGGLPRRYACGMS05]
    - 54k-line web app with hand-written SQL
    - Oracle database
  - Hilda [YGG+07]
    - High-level language for data-driven web apps
- Fabric implementation:
  - 3k lines → 740 lines
Performance results

- EJB, Hilda: DB server must be contacted frequently.
- Fabric: persistent objects can be cached at app server.
## Related work: Fabric

<table>
<thead>
<tr>
<th>Category</th>
<th>Examples</th>
<th>Fabric adds:</th>
</tr>
</thead>
<tbody>
<tr>
<td>Federated object store</td>
<td>OceanStore/Pond</td>
<td>• Transactions&lt;br&gt;&lt;br&gt;• Security policies</td>
</tr>
<tr>
<td>Secure distributed storage systems</td>
<td>Boxwood, CFS, Past</td>
<td>• Fine-grained security&lt;br&gt;&lt;br&gt;• High-level programming</td>
</tr>
<tr>
<td>Distributed object systems</td>
<td>Gemstone, Mneme, ObjectStore, Sinfonia, Thor</td>
<td>• Security enforcement&lt;br&gt;&lt;br&gt;• Multi-worker transactions with distrust</td>
</tr>
<tr>
<td>Distributed computation/ RPC</td>
<td>Argus, Avalon, CORBA, Emerald, Live Objects, Network Objects</td>
<td>• Single-system view of persistent data&lt;br&gt;&lt;br&gt;• Strong security enforcement</td>
</tr>
<tr>
<td>Distributed information flow systems</td>
<td>DStar, Jif/split, Swift</td>
<td>• Consistency for shared persistent data</td>
</tr>
</tbody>
</table>

No prior system has provided the security and expressiveness of Fabric.
Constructive security×3

- **Jif**
  - adding information flow policies to a real programming language
  - compiler supports programmer reasoning about security

- **Swift**
  - automatically, securely partitioning web applications

- **Fabric**
  - a general, high-level abstraction for secure, consistent, federated computing

- **A truly secure Internet Computer requires raising the level of abstraction even higher**
  - Decentralization and federation (ala Fabric) + automatic mapping of code and data (ala Swift)
  - Many challenges: mobile code; dynamic, adaptive partitioning; efficient, secure data management; richer compositional policies; formal security proofs; consistency policies; synthesizing more crypto protocols
Conclusions

Information flow policies enable a constructive approach to security:

- stronger, end-to-end, compositional security
- higher-level, more abstract programming model
- opportunities for greater efficiency and automatic optimization
Acknowledgments

- Steve Chong
- Jed Liu
- Nate Nystrom
- Xin Qi
- Mike George
- K. Vikram
- Steve Zdancewic
- Lantian Zheng
- Xin Zheng
Additional material

The following slides were not used in the talk but may help answer questions.
Covert channels

• Confidentiality depends on adversary not learning things from observations

• Information flow control prevents learning from observations at language level of abstraction (exception: termination vs. nontermination)

• Lower-level observations might still leak information:
  – Time and power
  – Size, existence, source, destination of network messages
  – Nondeterministic choices: addresses, interleavings, ...
  – Lower-level protocol message contents

• Run-time mechanisms exist for mitigating them
GUI interfaces

- Swift is a GUI toolkit similar to Swing (Java)
  - Layout is dynamic and user events are handled securely
- Information flow tracked through GUI widgets
  - `Out` and `In` labels bound information flowing up and down through hierarchy.

```java
class Widget[label Out, label In] { ... }
class Panel[label Out, label In] extends Widget[Out,In] {
  void addChild{Out}(label wOut, label wIn, Widget[wOut,wIn]{Out} w)
  where {*wOut} <= Out, {In;w} <= {*wIn};
}
class ClickableWidget[label Out, label In] extends Widget[Out,In] {
  void addListener{In}(ClickListener[Out,In]{In} li);
}
class Button[label Out, label In] extends ClickableWidget[Out,In] {
  String{Out} getText();
  void setText{Out}(String{Out} text);
}
interface ClickListener[label Out, label In] {
  void onClick{In}(Widget[Out, In]{In} b);
}
```