Swift:
Secure Web Applications
via Automatic Partitioning

Robert Soulé and Robert Grimm
New York University
The Three Questions

- What is the problem?
- What is new or different?
- What are the contributions and limitations?
Web-applications are popular; security is crucial

- Banking, email, online shopping, social networking

When is it secure to place code & data on client?

- JavaScript, cookies help improve performance
  - No roundtrips across internet
  - Less load on server

- But clients must not be trusted
Backgrounder:
Information Flow
Motivation and History

The military cares about information flow
  - Everyone can read “Unclassified”
  - Very few can read “Top Secret”

So computer scientists care about information flow (the hand that feeds you...)
  - Bell and LaPadula 1973: “No read up, no write down”
  - Denning 1976: The lattice model
  - “Orange Book” 1985: Trusted Computer System Evaluation
We (Might Want to) Care Too

How do I know my software doesn’t steal my data?

- Passwords
- Social security numbers
- Health records
- Financial records
Java Information Flow (JIF)

- Based on decentralized label model
  - Avoids rigid constraints of traditional multilevel security
  - Allows users to control their own information flow
  - Supports declassification and endorsement as part of model
- All data is labeled
  - Propagated on assignments
- Compiler & runtime ensure that labels are not violated
A principal represents a real-life user or group
A principal can be owner, reader, or writer of data
A principal can “act for” another principal
  I.e., the principal is trusted by the other
Labels

保密性标签：限制读取访问
- 一个政策有两部分，拥有者和读者
  - 拥有者是数据的来源
  - 读者是可能的目的地
- 例如，{ Alice → Bob }

完整性标签：限制写入访问
- 一个政策有两部分，拥有者和作者
  - 拥有者是数据的汇合点
  - 作者是可能的起源
- 例如，{ Alice ← Bob }
Labeling by Example

So, how do we get from one legal label to another?

Apply a step-wise relabeling transformation

Where each step is safe, i.e., preserves information flow

```c
int { Alice → Bob } x;
int z;
z = x; // OK: z gets x’s label

int { Alice → Bob, Chuck } y;
x = y; // OK: x’s policy is stronger
y = x; // BAD: y’s policy is weaker
```
Relabeling Details

- Privacy labels may be safely changed
  - Remove a reader
  - Add a policy
  - Add a reader r2 if policy contains r1 and r2 acts for r1
  - Replace owner o1 with o2 if o2 acts for o1

- Integrity label may be safely changed
  - Add a writer
  - Remove a policy
  - Replace a writer w1 with w2 if w1 acts for w2
  - Add a policy
Explicit Relabeling

- Declassification by owner, i.e., process’ authority
  - Allows more readers
- Endorsement by owner, i.e., process’ authority
  - Allows more writers
JIF in Summary

**Contributions**
- Decentralizes authority
- Focuses on usable programming model
- Makes information flow accessible to developers

**Limitations**
- Assumes a trusted execution platform
- Assumes a trusted compiler
- Assumes a principal lattice with unique top $\ast$ and bottom $\perp$
  - Cannot express arbitrary, non-hierarchical relationships
Now Back to Our Regularly Scheduled Programming: Swift
Swift Architecture

Figure 1: The Swift architecture

- **Jif source code**: Jif extends the Java programming language and supports web browser code running on the web server and JavaScript code running on the server. Jif source code is transformed into annotated Java source code.

- **WebIL code**: WebIL is useful for web application programming in its own right, providing support for browser-based user interfaces. The initial WebIL annotations are merely constraints on code and data placements.

- **Located WebIL code**: The system attempts to maximize flow methods to find a good partitioning. Data placement and server/client constraints are considered.

- **Java server code** and **Swift server runtime**: Once code and data placements have been determined, the compiler synthesizes Java and Swift libraries.

- **GWT runtime library** and **Swift client runtime**: The GWT library is used to optimize interactive performance, and Swift applications are synthesized as JavaScript code.

- **JavaScript client code**: The placement automatically synthesizes JavaScript code on the client, with security labels required at both locations to protect against communication injection and cross-site scripting.

- **CPS conversion**: The system attempts to minimize the cost of the placement, particularly by avoiding unnecessary network messages.

- **Label projection**: The system uses labels to enforce information flow and expressiveness separately. The general enforcement of information integrity is performed by these labels.

- **Swift architecture**: The Swift trust model addresses both problems at the same time, ensuring the placement is secure. Swift automatically partitions web application code while communicating by signing the protocol or interfaces by which client and server code interacts.
Same language with one built-in principal
- Use top "*" for server, acting for all possible principals
- Use "client" as default for client-side data

User interface framework
- Widgets represent user interface elements
- Events capture user actions
- Widget classes parameterized by labels
  - "Out" label is upper bound on information contained in widget
  - "In" label is upper bound on information gained thru events
WebIL

- Replaces labels with placement annotations
  - ‘C’ for client
  - ‘S’ for server
  - ‘h’ for high integrity (why?)

- Translation from JIF to WebIL
  - Enforces a-priori constraints
    - Database calls on server, UI interactions on client
  - Obeys label annotations
    - Data placed on client only if readable by client
    - Data accepted by server only if also writable by client
Goal: optimize performance without harming security

- Major performance factor: message latency

Constraints

- Fields and statements accessing fields *must* be consistent
  - May be replicated, i.e., client & server perform validity check
  - May still be partitioned, i.e., use temporary variable
- Consecutive statements *should* have same placement
- Number of messages *should* be minimized
Partitioning Algorithm

* Construct weighted directed graph
  * Weights approximates control flow of program
    * Both if branches, 10 times around loop, all dispatch receivers
  * Convert into integer programming problem
    * For each statement $u$, two variables $s_u$ and $c_u$ in $\{0, 1\}$
      * $s_u + c_u \geq 1$ ensures some placement
    * For each edge $e = (u, v)$, two variables $x_e$ and $y_e$
      * $x_e$ captures messages from client to server: $x_e \geq s_v - s_u$
      * $y_e$ captures messages from server to client: $y_e \geq c_v - c_u$
    * Minimize cost $\sum_e w_e (x_e + y_e)$ in polynomial time
Swift Runtime

- Execution blocks
  - Are contiguous segments of code with same placement
    - May contain more than one basic block
  - Are named by unique identifiers
- Activation records
  - Store local variables including arguments
  - Are also named by unique identifiers
- Closures
  - Combine execution blocks and activation records
  - Maintained as stack recording future returns and exceptions
Closure Results

- Closure results identify the next closure to execute
  - Simple results identify closure within same method
    - Activation record does not change
  - Method call results identify method to be invoked
    - Object, method, new activation record, return closure
  - Method return and exception results unroll closure stack
    - Up to next return closure for method returns
    - Up to matching exception closure for exception returns

- Continuation Passing Style (CPS)
  - Familiar technique from functional programming
Integrity of Control Flow

- Declassifications and endorsements are security critical
  - We can’t just let the client invoke such an execution block

- Solution
  - Compiler marks high integrity code in WebIL
  - Server pushes high integrity closures onto stack
    - Indicate legal control flow to a high integrity execution block
  - Client must invoke high integrity closures
    - High integrity closures cannot be popped without execution
    - High integrity blocks cannot be executed without closure
Example Applications

- Guess a number
  - 3 tries to find a randomly chosen number
- Poll
  - Vote for one of 3 options
- Secret keeper
  - Only creator gets to access the data
- Treasure hunt
  - aka Mine Sweeper
- Auction
  - Highest bidder wins
6.3 Performance results

Table 4 gives the number of round trips required for each of the example applications.

Table 2: Code size of example applications

<table>
<thead>
<tr>
<th>Example</th>
<th>Jif</th>
<th>Java target code</th>
<th>JavaScript</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>Server</td>
<td>Client</td>
</tr>
<tr>
<td>Null program</td>
<td>6 lines</td>
<td>0.7k tokens</td>
<td>0.6k tokens</td>
</tr>
<tr>
<td>Guess-a-Number</td>
<td>142 lines</td>
<td>12k tokens</td>
<td>25k tokens</td>
</tr>
<tr>
<td>Shop</td>
<td>1094 lines</td>
<td>139k tokens</td>
<td>187k tokens</td>
</tr>
<tr>
<td>Poll</td>
<td>113 lines</td>
<td>8k tokens</td>
<td>17k tokens</td>
</tr>
<tr>
<td>Secret Keeper</td>
<td>324 lines</td>
<td>38k tokens</td>
<td>38k tokens</td>
</tr>
<tr>
<td>Treasure Hunt</td>
<td>92 lines</td>
<td>11k tokens</td>
<td>11k tokens</td>
</tr>
<tr>
<td>Auction</td>
<td>502 lines</td>
<td>46k tokens</td>
<td>77k tokens</td>
</tr>
</tbody>
</table>

Table 3: Network messages required to perform a core UI task
What Do You Think?