Article review: Secure web application via automatic partitioning

Department of Computer Science
Cornell University

Ivan Hristov
Computational Engineering
Dresden University of Technology

20.05.2008

iv.hristov@yahoo.com
Part I

Introduction
Motivation

Problem

*How can one *easily* create ...

*... secure web applications?*

*... a dynamic, responsive user interface?*

*... both?*

Solution

*By using Swift!*
Web Programming

Increased responsiveness?
Some code and data on the client side.

The problem
Security vulnerabilities:
- confidentiality
- integrity
- explicit/implicit information flow

Solution
- right placement
- automation
- correctness

Article review: Secure web application via automatic partitioning
Overview

Swift Aspects

- secure by construction - annotations based paradigm
- easy to write - less awkwardness
- aids the programmer - automatic protocol and code generation

Article review: Secure web application via automatic partitioning
Stephen Chong, Jed Liu, Andrew C. Myers, Xin Qi, K. Vikram, Lantian Zheng, and Xin Zheng; 2007 Department of Computer Science Cornell University
Swift Architecture

Motivation and Overview

Architecture

Programming process

Transformation process

The Swift runtime

Evaluation

Article review: Secure web application via automatic partitioning
Stephen Chong, Jed Liu, Andrew C. Myers, Xin Qi, K. Vikram, Lantian Zheng, and Xin Zheng; 2007 Department of Computer ScienceCornell University
### Basic step

#### 1st step: Jif Source Code
- Labels - information security policies
- Static check - label consistency check

#### 2nd step: WebIL
- Annotations for placement

#### 3rd step: WebIL optimization
- Decision of exact placement
- Code and data replication
- Placement cost minimization

#### 4th step: Source Code Splitting
- Divide the original Java program into two

Article review: Secure web application via automatic partitioning
Stephen Chong, Jed Liu, Andrew C. Myers, Xin Qi, K. Vikram, Lantian Zheng, and Xin Zheng; 2007 Department of Computer Science Cornell University
Basic step

1st step: Jif Source Code
- Labels - information security policies
- Static check - label consistency check

2nd step: WebIL
- Annotations for placement

3rd step: WebIL optimization
- Decision of exact placement
- Code and data replication
- Placement cost minimization

4th step: Source Code Splitting
- Divide the original Java program into two
Basic step

1st step: Jif Source Code
- Labels - information security policies
- Static check - label consistency check

2nd step: WebIL
- Annotations for placement

3rd step: WebIL optimization
- Decision of exact placement
- Code and data replication
- Placement cost minimization

4th step: Source Code Splitting
- Divide the original Java program into two

Article review: Secure web application via automatic partitioning
### Basic step

<table>
<thead>
<tr>
<th>1st step: Jif Source Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>- Labels - information security policies</td>
</tr>
<tr>
<td>- Static check - label consistency check</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>2nd step: WebIL</th>
</tr>
</thead>
<tbody>
<tr>
<td>- Annotations for placement</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>3rd step: WebIL optimization</th>
</tr>
</thead>
<tbody>
<tr>
<td>- Decision of exact placement</td>
</tr>
<tr>
<td>- Code and data replication</td>
</tr>
<tr>
<td>- Placement cost minimization</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>4th step: Source Code Splitting</th>
</tr>
</thead>
<tbody>
<tr>
<td>- Divide the original Java program into two</td>
</tr>
</tbody>
</table>

Article review: Secure web application via automatic partitioning
Things to remember

Additional step: Java to JavaScript
- Client side transformation

Client side code and data
- Implementation of UI
- Faster interaction and higher responsiveness

Information flow
Should be strictly controlled

Functionality replication
- Responsiveness
- Security reasons

Article review: Secure web application via automatic partitioning
Stephen Chong, Jed Liu, Andrew C. Myers, Xin Qi, K. Vikram, Lantian Zheng, and Xin Zheng; 2007 Department of Computer Science Cornell University
Labels, principals, flows

Labels - set of security policies
- confidentiality: alice → bob
- integrity: alice ← bob

Implicit flows

```plaintext
int {alice → bob, alice; bob ← alice} y;
int {bob → bob} x;
int {alice → bob; bob ← alice} z;
if (x == 0) {
z = y; explicit information flow
}
```

NOTE! Implicit flow: from x to z
Acts for relationship

Principals
- Server (*) - maximally trusted
- Client (client) - untrusted

Acts for examples
* acts for client
client acts for bob and/or alice

Problem
Role misconfusion (object schizophrenia)

Solution
Static variables must not reference directly or indirectly the principle client!

Article review: Secure web application via automatic partitioning
Stephen Chong, Jed Liu, Andrew C. Myers, Xin Qi, K. Vikram, Lantian Zheng, and Xin Zheng; 2007 Department of Computer Science Cornell University
Acts for relationship

Principals
- Server (*) - maximally trusted
- Client (client) - untrusted

Acts for examples
* acts for client
client acts for bob and/or alice

Problem
Role misconfusion (object schizophrenia)

Solution
Static variables must not reference directly or indirectly the principle client!
Type of labels

Method labels

- begin label
- end label

... 15 void makeGuess {client} (Integer{client} num)
16 where authority(*), endorse({*→*})
17 throws NullPointerException
18 {
... 39 } no end label needed in this case
...
Type of labels

Endorsement labels

Usage: Prevention of untrusted access to trusted variables.

Example - checked endorsement

...  
19 int i = 0;
20 if (num != null) i = num.intValue();
21 endorse (i, {*←client} to {*←*})
22 if (i >= 1 && i <= 10) {
endorsement succeeds, ’i’ is endorsed ...
23 if (tries > 0 && i == secret) {
...
25 tries = 0;
... and tries can be accessed! ...
27 }
Declassify labels

Usage: To allow updates over trusted variables and/or explicit information flow

Example - declassify statement body

24 declassify (\{*→*\} to \{*→client\}) {
25 tries = 0;
26 finishApp("You win");
27 }

Article review: Secure web application via automatic partitioning
Type of labels

Inheritance labels
Usage: To control inheritance.

Example - *authority* and *auto-endorse*

...  
15 void makeGuess{*->client}(Integer{*->client} num)  
16 where authority(*), endorse({*<-*})  
17 throws NullPointerException  
18 {  
...  
39 }  
...

Other labels
robust declassification - Usage: To control declassification.
WebIL

Transformation process

- Client data/code placement - defined by the Jif security policies
- Server data/code placement - defined by the Jif security policies
- Declassification and endorsement are removed
- Fine-grained placement control through splitting of compound expressions

Uses:

- Placement annotations
- An efficient algorithm based on a reduction of the maximum flow problem

Article review: Secure web application via automatic partitioning

Stephen Chong, Jed Liu, Andrew C. Myers, Xin Qi, K. Vikram, Lantian Zheng, and Xin Zheng; 2007 Department of Computer Science Cornell University
# WebIL Placement annotations

<table>
<thead>
<tr>
<th>Annotation</th>
<th>Possible placements</th>
<th>High integrity</th>
</tr>
</thead>
<tbody>
<tr>
<td>C</td>
<td>{client}</td>
<td>N</td>
</tr>
<tr>
<td>S</td>
<td>{server}</td>
<td>N</td>
</tr>
<tr>
<td>Sh</td>
<td>{server}</td>
<td>Y</td>
</tr>
<tr>
<td>CS</td>
<td>{both}</td>
<td>N</td>
</tr>
<tr>
<td>CSh</td>
<td>{both}</td>
<td>Y</td>
</tr>
<tr>
<td>CS?</td>
<td>{client, both}</td>
<td>N</td>
</tr>
<tr>
<td>C?S</td>
<td>{server, both}</td>
<td>N</td>
</tr>
<tr>
<td>C?Sh</td>
<td>{server, both}</td>
<td>Y</td>
</tr>
<tr>
<td>C?S?</td>
<td>{client, server, both}</td>
<td>N</td>
</tr>
</tbody>
</table>

**Table 1: WebIL placement constraint annotations**
Outcome

Example

5 C?Sh: boolean b1 = (i >= 1);
6 boolean b2;
...
11 Sh: if (c1) c2 = (i == secret);
12 Sh: else c2 = false;
13 Sh: if (c2) {
14 C?Sh: tries = 0;

NOTE!

1 High-integrity marks mark data that should not be influenced by the client
2 The beginning of the high-integrity marks coincide with the endorsement
3 Auto-endorsement allow the code execution
Partitioning algorithm

Steps

- Approximate control-flow by weighted directed graph
- Placement algorithm based on integer program algorithm, which is reduced to an instance of the maximum flow problem

Note

"...the accuracy of this approach is limited by how closely the weighted directed graph approximates actual run-time behavior." Finding and evaluating all possible program’s paths in a complex problem might be very problematic, if not even impossible.
Integrity of control flow

Swift runtime

- communication and synchronization management

Definition

"A high-integrity closure is one whose execution block has high-integrity side effects, and is therefore annotated Sh or CSh."

Prevention of misbehaving clients

"A client may invoke a high-integrity closure only if it is at the top of the closure stack."

..."As a result, a misbehaving client cannot control the execution of high-integrity closures, even if it throws arbitrary exceptions and invokes arbitrary closures on the server."
Source code size

<table>
<thead>
<tr>
<th>Example</th>
<th>Jif</th>
<th>Java target code</th>
<th>JavaScript</th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>Server</td>
<td>Client</td>
<td>All</td>
<td>Framework</td>
<td>App</td>
</tr>
<tr>
<td>Null program</td>
<td>6 lines</td>
<td>0.7k tokens</td>
<td>0.6k tokens</td>
<td>73 kB</td>
<td>70 kB</td>
<td>3 kB</td>
</tr>
<tr>
<td>Guess-a-Number</td>
<td>142 lines</td>
<td>12k tokens</td>
<td>25k tokens</td>
<td>267 kB</td>
<td>104 kB</td>
<td>162 kB</td>
</tr>
<tr>
<td>Shop</td>
<td>1094 lines</td>
<td>139k tokens</td>
<td>187k tokens</td>
<td>1.21 MB</td>
<td>323 kB</td>
<td>889 kB</td>
</tr>
<tr>
<td>Poll</td>
<td>113 lines</td>
<td>8k tokens</td>
<td>17k tokens</td>
<td>242 kB</td>
<td>104 kB</td>
<td>137 kB</td>
</tr>
<tr>
<td>Secret Keeper</td>
<td>324 lines</td>
<td>38k tokens</td>
<td>38k tokens</td>
<td>639 kB</td>
<td>332 kB</td>
<td>307 kB</td>
</tr>
<tr>
<td>Treasure Hunt</td>
<td>92 lines</td>
<td>11k tokens</td>
<td>11k tokens</td>
<td>211 kB</td>
<td>99 kB</td>
<td>112 kB</td>
</tr>
<tr>
<td>Auction</td>
<td>502 lines</td>
<td>46k tokens</td>
<td>77k tokens</td>
<td>503 kB</td>
<td>116 kB</td>
<td>387 kB</td>
</tr>
</tbody>
</table>

Table 2: Code size of example applications

Notes

"Java target code" - only generated Java code
"JavaScript All" - the UI, Jif and Swift client framework
"JavaScript Framework" - the Swift, UI and Jif framework
"JavaScript App" - application’s JavaScript source code
## Performance/Responsiveness

<table>
<thead>
<tr>
<th>Example</th>
<th>Task</th>
<th>Actual</th>
<th>Optimal</th>
</tr>
</thead>
<tbody>
<tr>
<td>Guess-a-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>0</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>

*Table 3: Network messages required to perform a core UI task*
Flexibility

**Automatic repartitioning**

In case of change in the security policies, repartitioning is done automatically.
Part II

Discussion
Discussion

Points

- Pros & Cons
- Complexity
- Program’s paths problem
- Evaluation