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

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

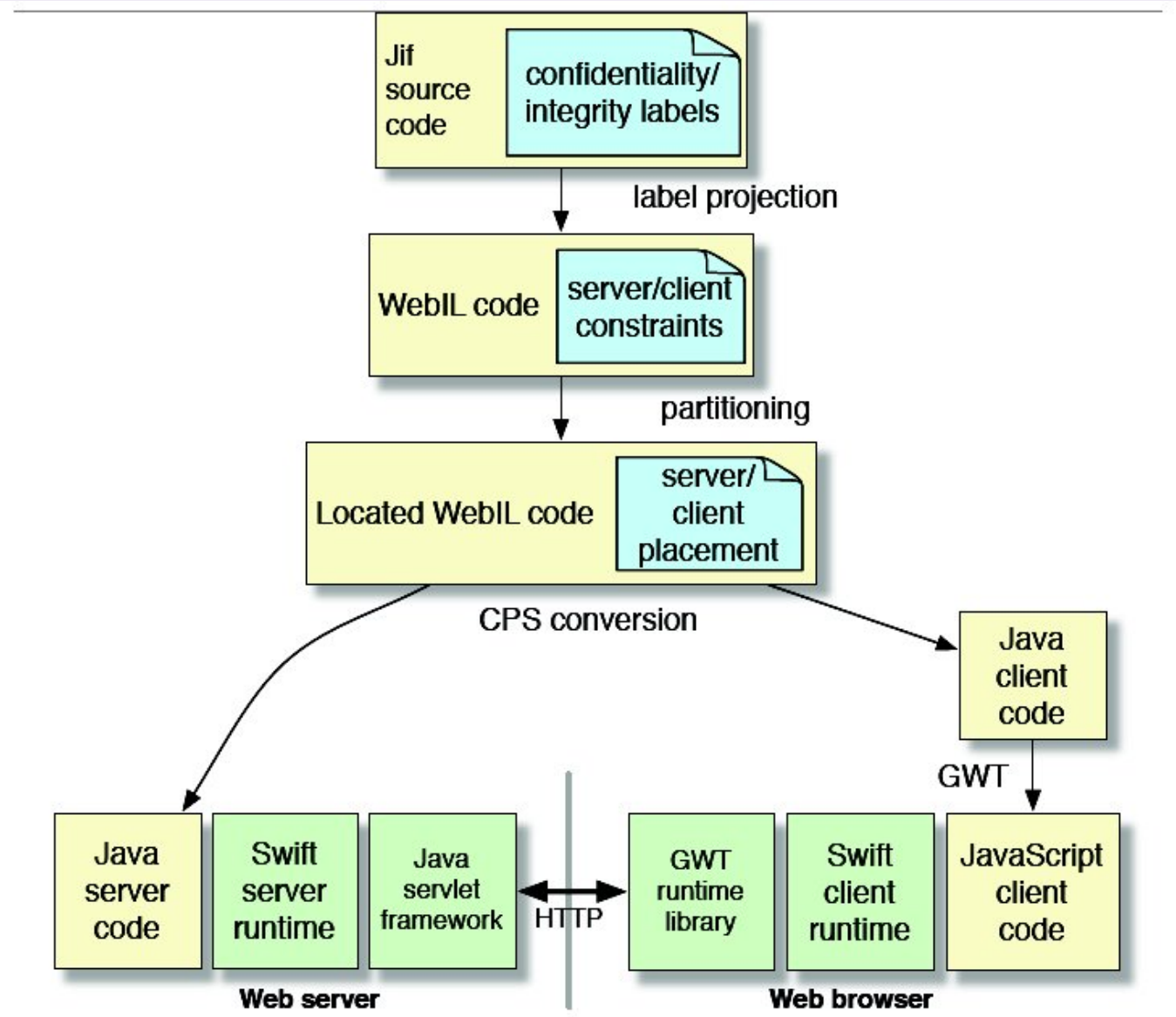
- right placement
- automation
- correctness

Overview

Swift Aspects

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

Swift Architecture



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

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

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

Labels, principals, flows

Labels - set of security policies

- confidentiality : alice $\rightarrow$ bob
- integrity : alice $\leftarrow$ bob

Implicit flows

int {alice $\rightarrow$ bob, alice; bob $\leftarrow$ alice} y;
int {bob $\rightarrow$ bob} x;
int {alice $\rightarrow$ bob; bob $\leftarrow$ 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!

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 }
```

Type of labels

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 }
```

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

WebIL Placement annotations

Annotation	Possible placements	High integrity
C	{*client*}	N
S	{*server*}	N
Sh	{*server*}	Y
CS	{*both*}	N
CSh	{*both*}	Y
CS?	{*client*, *both*}	N
C?S	{*server*, *both*}	N
C?Sh	{*server*, *both*}	Y
C?S?	{*client*, *server*, *both*}	N

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

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

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

Example	Task	Actual		Optimal	
		Server→Client	Client→Server	Server→Client	Client→Server
Guess-a-Number	guessing a number	1	2	1	1
Shop	adding an item	0	0	0	0
Poll	casting a vote	1	1	0	1
Secret Keeper	viewing the secret	1	1	1	1
Treasure Hunt	exploring a cell	1	2	1	1
Auction	bidding	1	1	1	1

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