Secure Multi-Execution

Dominique Devriese   Frank Piessens

K.U.Leuven

May 14, 2010

Outline

Introduction

- Information Flow Analysis has received much attention:
  - Static analysis methods:
    From Denning to JFlow/JIF and FlowCaml
    But:
    - Substantial Programmer Effort
    - In general undecidable statically
    - Hard to handle exceptions, parallellism, timing covert channel
  - Dynamic methods:
    Many practical but unsound methods, some sound and somewhat practical methods
    But:
    - Some use cases require sound methods (e.g. web page scripts)
    - No existing monitor can precisely enforce non-interference
    - Hard to handle exceptions, parallellism, timing covert channel

Secure Multi-Execution

Secure Multi-Execution

- A novel dynamic enforcement technique for non-interference
- Nice theoretical properties
    - Strong soundness guarantee
    - The first (afawk) sound and precise enforcement method
- Practical in some scenario's
    - Performance measurements
    - Browser implementation possible?

Outline

Information Flow Analysis

```
var text = document.getElementById
        ('email-input').text;
var abc = 0;
if(text.indexOf('abc')!=-1) { abc = 1 };
var url = 'http://example.com/img.jpg'
    + '?t=' + escape(text) + abc;
document.getElementById('banner-img')
    .src = url;
```

Information Flow Analysis

```
var text = document.getElementById
        ('email-input').text;
var abc = 0;
if(text.indexOf('abc')!=-1) { abc = 1 };
var url = 'http://example.com/img.jpg'
    + '?t=' + escape(text) + abc;
document.getElementById('banner-img')
    .src = url;
```

← Input at level H (line `('email-input').text;`)

← Output at level L (line `.src = url;`)

Timing Covert Channel

```
function time(f) {
  var t = new Date().getTime();
  f();
  return new Date().getTime() - t;
}
function f() {
  if(abc != 0) {
    for(var i = 0; i < 10000; ++i) {}
  }
}
var abcLo = 0
if(time(f) > 10) {
  abcLo = 1;
}
```

Termination Covert Channel

```
while(abc == 0) {}
img.url = 'http://example.com/img.jpg';
```

Secure Multi-Execution

L

```
var t = (...).text
var abc = 0
if(t.indexOf('abc')!=-1)
 { abc = 1 }
var url = baseUrl + '?t='
 + escape(t) + abc
(...).src = url
```

H

```
var t = (...).text
var abc = 0
if(t.indexOf('abc')!=-1)
  { abc = 1 }
var url = baseUrl + '?t='
  + escape(t) + abc
(...).src = url
```

Secure Multi-Execution

L

`var t =` ~~`(...).text`~~ *undefined*

```
var abc = 0
if(t.indexOf('abc')!=-1)
 { abc = 1 }
var url = baseUrl + '?t='
 + escape(t) + abc
(...).src = url
```

H

```
var t = (...).text
var abc = 0
if(t.indexOf('abc')!=-1)
  { abc = 1 }
var url = baseUrl + '?t='
  + escape(t) + abc
(...).src = url
```

Secure Multi-Execution

L

undefined

```
var t = (...).text
var abc = 0
if(t.indexOf('abc')!=-1)
 { abc = 1 }
var url = baseUrl + '?t='
 + escape(t) + abc
(...).src = url
```

H

```
var t = (...).text
var abc = 0
if(t.indexOf('abc')!=-1)
  { abc = 1 }
var url = baseUrl + '?t='
  + escape(t) + abc
(...).src = url
```

Input Side Effects

L

undefined

```
var t = (...).text
var c = window.confirm
if( c("Send e-mail?") )
  { (...) }
var abc = 0
if(t.indexOf('abc')!=-1)
  { abc = 1 }
var url = baseUrl + '?t='
  + escape(t) + abc
(...).src = url
```

H

```
var t = (...).text
var c = window.confirm
if( c("Send e-mail?") )
  { (...) }
var abc = 0
if(t.indexOf('abc')!=-1)
  { abc = 1 }
var url = baseUrl + '?t='
  + escape(t) + abc
(...).src = url
```

Input Side Effects

L

```
var t = (...).text
var c = window.confirm
if( c("Send e-mail?") )
  { (...) }
var abc = 0
if(t.indexOf('abc')!=-1)
  { abc = 1 }
var url = baseUrl + '?t='
  + escape(t) + abc
(...).src = url
```

H

```
var t = (...).text
var c = window.confirm
if( c("Send e-mail?") )
  { (...) }
var abc = 0
if(t.indexOf('abc')!=-1)
  { abc = 1 }
var url = baseUrl + '?t='
  + escape(t) + abc
(...).src = url
```

Secure Multi-Execution

Properties

- "Obviously" sound:
  Only execution at high level can see the real high inputs
  Only execution at low level can produce low outputs
- "Obviously" precise:
  If a program is non-interferent, then changing high inputs in low executions will not change their low behaviour

Outline

Non-interference

A formalization of information flow policies

Assume:

- sets of input channels $\mathcal{C}_i$, output channels $\mathcal{C}_o$
- security level lattice $\mathcal{L}$
- $\sigma_{in} : \mathcal{C}_i \rightarrow \mathcal{L}$, $\sigma_{out} : \mathcal{C}_o \rightarrow \mathcal{L}$
- inputs $I : \mathcal{C}_i \rightarrow (\mathbb{N} \rightarrow Int)$, outputs $O : \mathcal{C}_o \rightarrow List[Int]$
- $I =_l I'$ iff $I(c_i) = I'(c_i)$ for all c_i such that $\sigma_{in}(c_i) \leq l$

Definition

A program P is (termination-insensitively) *non-interferent* if for all security levels l and inputs $I =_l I'$, where P terminates for I and I' with outputs O and O', we have that $O =_l O'$.

Soundness

Definition (Strong non-interference)

A program P is *timing-sensitively non-interferent* or *strongly non-interferent* with relation to a given semantics $\hookrightarrow^*$ if for all security levels $l \in \mathcal{L}$, for all $n \geq 0$, for all program inputs I and I' such that $I =_l I'$ holds that if

$$(P, I) \hookrightarrow^n (p, O) \quad ,$$

then

$$(P, I') \hookrightarrow^n (p', O') \quad ,$$

and $p' =_l p$ and $O' =_l O$.

Theorem (Soundness of Secure Multi-Execution)

Any program P is strongly non-interferent under secure multi-execution, using the $\texttt{select}_{\text{lowprio}}$ scheduler function.

Precision

Theorem (Precision of Secure Multi-Execution)

Suppose we have a (termination-sensitively) non-interferent program P. Suppose that

$$(P, I) \rightharpoonup^* (p, O)$$

(terminates) for some I, p and O. Then

$$(P, I) \Rightarrow^* (p, O) \quad .$$

Outline

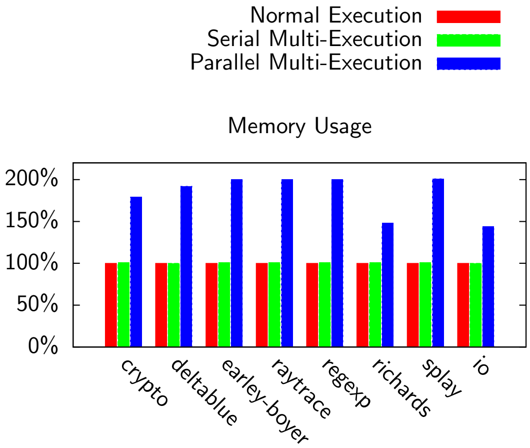
Experimental Results

- Spidermonkey Javascript engine, no real browser
- 2 security levels
- Dual-core PC
- 3 types of execution:
  - Standard execution
  - Serial Multi-Execution
  - Parallel Multi-Execution
- Benchmarks:
  - Google Chrome V8 Benchmark Suite: crypto, deltablue, earley-boyer, raytrace, regexp, richards, splay
  - io: model I/O functions: hi_input, hi_output, lo_input, lo_output: some calculations + I/O at different security levels

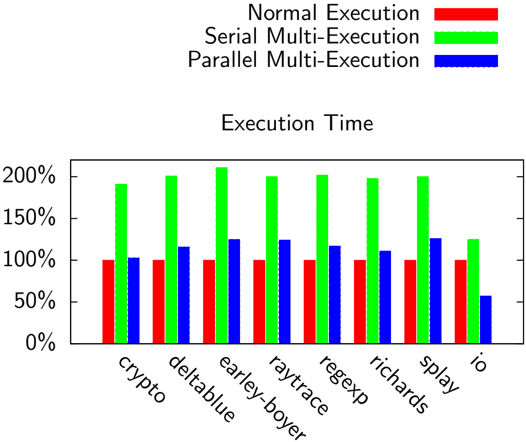
IO Benchmark

```
for (var i = 0; i < 100; ++i) {
  var test = 0;
  for (var j = 0; j < 10000; ++j) {
    test += j;
  }
  if (i % 10 == 0) {
    var hi_in = hi_input();
    var lo_in = lo_input();
    lo_output("#" + i + ". lo_in: '"
      + lo_in + "'. hi_in is: '"
      + hi_in + "'");
    hi_output("#" + i + ". hi_in: '"
      + hi_in + "'. lo_in is: '"
      + lo_in +"'");
  }
}
```

Experimental Results: Memory Usage



Experimental Results: Execution Time



Automatic parallellization

```
for (var i = 0; i < 100; ++i) {
  var test = 0;
  for (var j = 0; j < 10000; ++j) {
    test += j;
  }
  if (i % 10 == 0) {
    var hi_in = hi_input();
    var lo_in = lo_input();
    lo_output("#" + i + ". lo_in: '"
      + lo_in + "'. hi_in is: '"
      + hi_in + "'");
    hi_output("#" + i + ". hi_in: '"
      + hi_in + "'. lo_in is: '"
      + lo_in +"'");
  }
}
```

Latency

Latency

Latency

Latency

Automatic parallellization

```
for (var i = 0; i < 100; ++i) {
  var test = 0;
  for (var j = 0; j < 10000; ++j) {
    test += j;
  }
  if (i % 10 == 0) {
    var hi_in = hi_input();
    var lo_in = lo_input();
    lo_output("#" + i + ". lo_in: '"
      + lo_in + "'. hi_in is: '"
      + hi_in + "'");
    hi_output("#" + i + ". hi_in: '"
      + hi_in + "'. lo_in is: '"
      + lo_in +"'");
  }
}
```

L	H
Skip	Latency
Latency	Reuse
Latency	Skip
Skip	Latency

Outline

The technique's merits

Advantages:

Very strong	Soundness guarantee
Very general	No fundamental issues with parallellism, exceptions or other language features
Good precision	No change for (termination-sensitively) non-interferent programs
Acceptable imprecision	Interferent executions are modified in acceptable way (intuitive, no formalisation...)
Dynamic	Run-time assignment of I/O channels to security levels

Downsides:

Performance	Acceptable for some use cases?

Thank you for your attention.

Any questions?