A Security-Preserving Compiler for Distributed Programs

Cédric Fournet, Gurvan Le Guernic, Tamara Rezk

INRIA - Microsoft Research Joint Center
gleguern@gmail.com

June 19th, 2009
Outline

1. Introduction
2. Slicing
3. Control Flow Protocol
4. Static Single Remote Assigner
5. Cryptographic Protection
6. Experimental Results
7. Demonstration
Introduction
Introduction

Goal:
- simplify programming secure distributed programs
- automatically secure communications between hosts at the right level

Special purpose compiler:
- from sequential programs using shared memory protected by security levels
- to distributed programs using encrypted and signed messages
- main requirement: not introduce new covert flows
Overview

Input:
- localized sequential program
- information flow policy based on a lattice of security labels
- communication through a global shared memory

Output:
- distributed F# program
- communication of encrypted and signed (MAC) messages
Example: source

Role #HH# \( a \);
Role #HH# \( b \);
Role #HH# \( c \);
Role #LL# other;

global int#HL# \( x \);
global int#LH# \( y \);
global int#LH# \( z \);

\begin{verbatim}
a:[
x := 1; y := 2;
while \( y < 3 \) do {
y := y + 4;
b:[ if ((y \text{ mod } 2) = 1) then x := x + 9 \text{ else skip } ]
c:[ z := 5 ]
}
\end{verbatim}
4 steps compilation

- **Slicing**
  - every thread executes on a single host
  - merging on same host

- **Insertion of a control flow protocol**
  - $pc_I$ for every integrity $I$
  - test $pc$ to enforce source control flow

- **SSA-like transformation**
  - single remote last assignment

- **Cryptographic protection**
  - select adequate keys
  - ensure IF policy
Slicing
Slicing

- ensuring merges on single host
- loop counters for ensuring single execution

\[ b5 \ i \ j(b, L) : \]

\[
\text{if } (y_{\text{LH}} \mod 2) = 1 \\
\text{then } \{ x_{\text{HL}} := x_{\text{HL}} + 9 \} \\
\text{else } \{ \text{skip} \}; \\
\text{call}(a4 \ i \ j)
\]
Control Flow Protocol
Control Flow Protocol

- each thread (t) sets the pc at its integrity level
- test pcs of previous threads (t’) not checked before by:
  - a thread trusted by t
  - a thread more trusted than t’

\[
\text{check } \text{pc}\text{1}_LH \cong ("a8", [i; j]) \text{ do } \{
\text{pc}\text{2}_LH := ("b5", [i; j]);
\text{if } (y_LH \mod 2) = 1
\text{then } \{x_{HL} := x_{HL} + 9\}
\text{else } \{\text{skip}\};
\text{call}(a4 \ i \ j) \}
\]
Static Single Remote Assigner
Static Single Remote Assigner

- **goal:** statically know assigning thread if remote assignment
- **trick:** merging threads write in merger locals

```c
check (a8 i j.pc1) \equiv ("a8", [i; j]) do {
    b5 i j.pc2 := ("b5", [i; j]);
    if ((a8 i j.y) mod 2) = 1
    then {b5 i j.x := (a1 i j.x) + 9}
    else {skip; b5 i j.x := a1 i j.x};
    call(a4 i j)}
```
Cryptographic Protection


- encrypt and sign variables sent on the network
- use thread id as tag to compute MAC

\[
\text{check } \text{Verify}(b.pcl_s, "a8."}^i".^j".pcl", b.pcl_{mc}, K^s_{1HL}) \text{ do }
\]
\[
\text{check } \text{Verify}(b.ys, "a8."}^i".^j".y", b.ys_{mc}, K^s_{1HL}) \text{ do }
\]
\[
b.x_{mc} := \text{Decrypt}(b.x_e, K^e_{1HL});
b.x := \text{Unmarshal}(b.x_{mc});
b.y := \text{Unmarshal}(b.y_{mc});
b.pcl := \text{Unmarshal}(b.pcl_{mc});
\]
\[
\text{check } b.pcl \cong ("a8", [i; j]) \text{ do }
\]
\[
b.pc2 := ("b5", [i; j]);
\]
\[
\text{if } (b.y \mod 2) = 1
\]
\[
\text{then } \{b.x := b.x + 9\}
\]
\[
\text{else } \{b.x := b.x\};
b.x_{mc} := \text{Marshal}(b.x);
b.pc2_{mc} := \text{Marshal}(b.pc2);
b.x_e := \text{Encrypt}(b.x_{mc}, [K^e_{1HL}]);
\]
Experimental Results
### Experimental Results

<table>
<thead>
<tr>
<th>Program</th>
<th>LOC</th>
<th>l/t</th>
<th>crypto</th>
<th>keys</th>
<th>Time (s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>empty</td>
<td>2</td>
<td>1</td>
<td>(1+0)</td>
<td>0/0</td>
<td>0/0</td>
</tr>
<tr>
<td>running</td>
<td>18</td>
<td>3</td>
<td>(5+3)</td>
<td>2/2</td>
<td>4/4</td>
</tr>
<tr>
<td>rpc</td>
<td>11</td>
<td>2</td>
<td>(3+3)</td>
<td>2/2</td>
<td>4/4</td>
</tr>
<tr>
<td>guess</td>
<td>52</td>
<td>7</td>
<td>(13+3)</td>
<td>2/2</td>
<td>13/16</td>
</tr>
<tr>
<td>hospital</td>
<td>33</td>
<td>5</td>
<td>(9+0)</td>
<td>4/4</td>
<td>11/11</td>
</tr>
<tr>
<td>taxes</td>
<td>55</td>
<td>4</td>
<td>(7+2)</td>
<td>8/8</td>
<td>16/16</td>
</tr>
</tbody>
</table>

RPC = 6000 symmetric-key cryptographic operations
## Experimental Results

<table>
<thead>
<tr>
<th>Program</th>
<th>LOC</th>
<th>l/t</th>
<th>crypto</th>
<th>keys</th>
<th>Time (s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>empty</td>
<td>2</td>
<td>102</td>
<td>(1+0)</td>
<td>0/0</td>
<td>1.59</td>
</tr>
<tr>
<td>running</td>
<td>18</td>
<td>464</td>
<td>(5+3)</td>
<td>2/2</td>
<td>1.58</td>
</tr>
<tr>
<td>rpc</td>
<td>11</td>
<td>321</td>
<td>(3+3)</td>
<td>2/2</td>
<td>1.63</td>
</tr>
<tr>
<td>guess</td>
<td>52</td>
<td>912</td>
<td>(13+3)</td>
<td>2/2</td>
<td>1.69</td>
</tr>
<tr>
<td>hospital</td>
<td>33</td>
<td>906</td>
<td>(9+0)</td>
<td>4/4</td>
<td>1.70</td>
</tr>
<tr>
<td>taxes</td>
<td>55</td>
<td>946</td>
<td>(7+2)</td>
<td>8/8</td>
<td>1.71</td>
</tr>
</tbody>
</table>
# Experimental Results

<table>
<thead>
<tr>
<th>Program</th>
<th>LOC</th>
<th>l/t</th>
<th>crypto</th>
<th>keys</th>
<th>Time (s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>empty</td>
<td>2</td>
<td>1 (1+0)</td>
<td>0/0</td>
<td>0/0</td>
<td>1.59</td>
</tr>
<tr>
<td>running</td>
<td>18</td>
<td>3 (5+3)</td>
<td>2/2</td>
<td>4/4</td>
<td>1.58</td>
</tr>
<tr>
<td>rpc</td>
<td>11</td>
<td>2 (3+3)</td>
<td>2/2</td>
<td>4/4</td>
<td>1.63</td>
</tr>
<tr>
<td>guess</td>
<td>52</td>
<td>7 (13+3)</td>
<td>2/2</td>
<td>13/16</td>
<td>1.69</td>
</tr>
<tr>
<td>hospital</td>
<td>33</td>
<td>5 (9+0)</td>
<td>4/4</td>
<td>11/11</td>
<td>1.70</td>
</tr>
<tr>
<td>taxes</td>
<td>55</td>
<td>4 (7+2)</td>
<td>8/8</td>
<td>16/16</td>
<td>1.71</td>
</tr>
</tbody>
</table>

RPC = 6000 symmetric-key cryptographic operations
Demonstration
A Security-Preserving Compiler for Distributed Programs

Cédric Fournet, Gurvan Le Guernic, Tamara Rezk

INRIA - Microsoft Research Joint Center
gleguern@gmail.com

June 19th, 2009