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;

a:[
  x := 1; y := 2;
  while y < 3 do {
    y := y + 4;
    b:[ if ((y 'mod 2) = 1) then x := x + 9 else skip ];
    c:[ z := 5 ]
  }
]
```

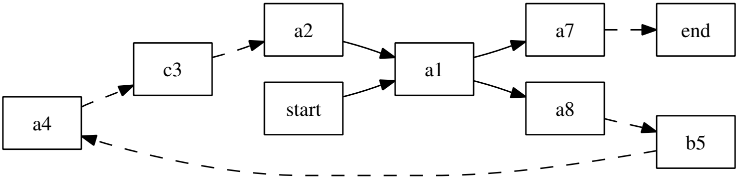
4 steps compilation

- 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



$b5\ i\ j(b, L)$:

```
if (y_LH mod 2) = 1
then {x_HL := x_HL + 9}
else {skip};
call(a4 i j)
```

Control Flow Protocol

Control Flow Protocol

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

```
check pc1_LH ≅ ("a8", [i; j]) do {
 pc2_LL := ("b5", [i; j]);
 if (y_LH mod 2) = 1
 then {x_HL := x_HL + 9}
 else {skip};
 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

```
check (a8 i j.pc1) ≅ ("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

Cryptographic Protection

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

```
check Verify(b.pcl_s, "a8."^i^"."^j^".pcl", b.pcl_mc, K^s_1HL) do {
 check Verify(b.y_s, "a8."^i^"."^j^".y", b.y_mc, K^s_1HL) do {
  b.x_mc := Decrypt(b.x_e, K^e_1HL);
  b.x := Unmarshal(b.x_mc);
  b.y := Unmarshal(b.y_mc);
  b.pcl := Unmarshal(b.pcl_mc);
  check b.pcl ≅ ("a8", [i; j]) do {
   b.pc2 := ("b5", [i; j]);
   if (b.y mod 2) = 1
   then {b.x := b.x + 9}
   else {b.x := b.x};
   b.x_mc := Marshal(b.x);
   b.pc2_mc := Marshal(b.pc2);
   b.x_e := Encrypt(b.x_mc, [K^e_1HL]);
```

Experimental Results

Experimental Results

Program	LOC		I/t		crypto		keys	Time (s)	
`empty`	2	102	1	(1+0)	0/0	0/0	0/0	1.59	1.63
`running`	18	464	3	(5+3)	2/2	4/4	1/2	1.58	1.71
`rpc`	11	321	2	(3+3)	2/2	4/4	1/1	1.63	2.58
`guess`	52	912	7	(13+3)	2/2	13/16	2/3	1.69	1.98
`hospital`	33	906	5	(9+0)	4/4	11/11	4/8	1.70	1.84
`taxes`	55	946	4	(7+2)	8/8	16/16	4/6	1.71	1.77

Experimental Results

Program	LOC		I/t		crypto		keys	Time (s)	
empty	2	102	1	(1+0)	0/0	0/0	0/0	1.59	1.63
running	18	464	3	(5+3)	2/2	4/4	1/2	1.58	1.71
rpc	11	321	2	(3+3)	2/2	4/4	1/1	1.63	2.58
guess	52	912	7	(13+3)	2/2	13/16	2/3	1.69	1.98
hospital	33	906	5	(9+0)	4/4	11/11	4/8	1.70	1.84
taxes	55	946	4	(7+2)	8/8	16/16	4/6	1.71	1.77

Experimental Results

Program	LOC		I/t		crypto		keys	Time (s)	
empty	2	102	1	(1+0)	0/0	0/0	0/0	1.59	1.63
running	18	464	3	(5+3)	2/2	4/4	1/2	1.58	1.71
rpc	11	321	2	(3+3)	2/2	4/4	1/1	1.63	2.58
guess	52	912	7	(13+3)	2/2	13/16	2/3	1.69	1.98
hospital	33	906	5	(9+0)	4/4	11/11	4/8	1.70	1.84
taxes	55	946	4	(7+2)	8/8	16/16	4/6	1.71	1.77

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