

# Secure Program Partitioning

Ruy Ley-Wild

Computer Science Department  
Carnegie Mellon University

October 29, 2007

# Secure Program Partitioning

- ▶ Secure Program Partitioning  
S. Zdancewic, L. Zheng, N. Nystrom, A. C. Myers  
ACM Transactions on Computer Systems 20 (2002)
- ▶ Programming Languages for Information Security  
S. Zdancewic  
PhD Thesis (2002)

# Motivation

- ▶ Distributed system with mutually untrusted principals
  - ▶ No universally trusted principal
  - ▶ Compromised principal only threatens its trusters
- ▶ Protect end-to-end confidentiality and integrity
- ▶ Compile sequential program into sequential distributed system
  - ▶ Guide partitioning with security type annotations and trust relation
  - ▶ Find program partitioning that satisfies every principal's policy
  - ▶ No concurrency

# Features

- ▶ Selective declassification
  - ▶ Controlled information leaking
- ▶ Extend information flow to distributed setting
  - ▶ Address covert and overt storage channels
  - ▶ Elide timing and termination channels
- ▶ Decentralized computation despite incomplete trust
  - ▶ No universally trusted principal
- ▶ Force explicit policy decisions
- ▶ Automatic policy enforcement

# Information Flow

- ▶ Policies specify end-to-end information use
- ▶ Static (security-typed languages)
  - ▶ Type annotations restrict information flow
  - ▶ Protect information globally
  - ▶ May conservatively reject safe programs
- ▶ Dynamic (access control/capabilities)
  - ▶ Regulate principals' actions at particular program points
  - ▶ Protect access locally
  - ▶ Rejects individual executions

# Confidentiality and Integrity

- ▶ Confidentiality (a.k.a. secrecy)
  - ▶ Alice trusts sending her data to Bob
  - ▶ High confidentiality means few principals have data
- ▶ Integrity
  - ▶ Alice trusts receiving data from Bob
  - ▶ High integrity means many principals trust data
- ▶ Duality
  - ▶ High security means high confidentiality or low integrity

# Security Labels

- ▶ Security label  $\ell$ 
  - ▶ set of principals with distinguished roles
  - ▶ describe confidentiality and integrity policy
- ▶ Label preorder  $\ell_1 \sqsubseteq \ell_2$ 
  - ▶ less restrictive policy  $\ell_1$  is enforced by more restrictive policy  $\ell_2$
  - ▶ datum with label  $\ell_1$  can be treated as though it has label  $\ell_2$
  - ▶  $\ell_2$  must specify at least as much confidentiality as  $\ell_1$ 
    - ▶ **Public**  $\sqsubseteq$  **Secret**
  - ▶  $\ell_2$  must specify at most as much integrity as  $\ell_1$ 
    - ▶ **Untainted**  $\sqsubseteq$  **Tainted**
- ▶ Equivalence classes form distributive lattice
  - ▶ join  $\sqcup$  combines policy restrictions

# Confidentiality and Integrity Security Labels

- ▶ Confidentiality label  $\gamma = \{o_1 : \vec{r}_1; \dots ; o_n : \vec{r}_n\}$ 
  - ▶ Data has owners  $o_i$  who permit readers  $\vec{r}_i$
  - ▶  $\{o : r\} \sqsubseteq \{o\}$
  - ▶ Conjunctive restriction

$$\frac{x_1 : \{o_1 : r_1, r_3\} \quad x_2 : \{o_2 : r_2, r_3\}}{x_1 + x_2 : \{o_1 : r_1, r_3\} \sqcap \{o_2 : r_2, r_3\}}$$

Only  $r_3$  may read  $x_1 + x_2$

- ▶ Integrity label  $\iota = \{p_1, \dots, p_n\}$ 
  - ▶ Data is trusted by principals  $p_i$
  - ▶  $\{\} \sqsubseteq \{p_1\} \sqsubseteq \{p_1, p_2\}$
- ▶ Security type annotation  $e : \tau^{\langle \gamma, \iota \rangle}$ 
  - ▶ conservative  $\sqsubseteq$ -upper bound on security of expression

# Declassification and Trust Coercions

- ▶ Downgrading
  - ▶ Assign a less restrictive label (lower in lattice)
  - ▶ Loss of confidentiality or increase of claimed integrity
  - ▶ Explicit coercions for selective declassification and trust
- ▶  $\text{declassify}_{\gamma}(e)$ : downgrade expression  $e$  to confidentiality  $\gamma$ 
  - ▶ all owner's must delegate authority
- ▶  $\text{endorse}_{\iota}(e)$ : upgrade expression  $e$  to integrity  $\iota$ 
  - ▶ programmer must intend belief

# Implicit Flows

- ▶ Reaching a program point with label  $\langle \gamma, \iota \rangle$ 
  - ▶  $\gamma$  captures what information may be learned
  - ▶  $\iota$  captures data integrity and control flow
- ▶ Implicit information flow due to control flow

$x : \text{bool}^{\ell_x} \vdash^{\ell} \text{if } x \text{ then }^{\ell \sqcup \ell_x} y = \text{true} \text{ else }^{\ell \sqcup \ell_x} y = \text{false};^{\ell}$

$y$ 's security label must be at least as restrictive as  $x$ 's

- ▶ Implicit flow due to synchronization is an obstacle for concurrency

# Labelled Programs

- ▶ Distinguished label  $l_{\text{pc}}$  denotes the label at current program point
- ▶ Function types may have additional initial and final labels and authority constraints

$$\tau_2^{\ell_2} \ell_i \xrightarrow{(\vec{a})} \ell_f \tau_1^{\ell_1}$$

- ▶ Pre:  $l_{\text{pc}} \sqsubseteq l_i$  and principals  $\vec{a}$  delegated authority
- ▶ Post:  $l_{\text{pc}} \sqsubseteq l_f$

# Security Model

- ▶ Secure pairwise communication between principals
- ▶ Noninterference by failure or subversion of an untrusted principal
- ▶ Bad principals can only forge messages from bad principals
- ▶ Multiple concurrent bad principals vs. multiple sequential good principals

# Oblivious Transfer Problem

- ▶ Alice has two values  $v_1, v_2$
- ▶ Bob may request exactly one of the values
- ▶ Bob doesn't want Alice to know which value he requested
- ▶ A trusted third party is necessary

## Oblivious Transfer Problem: Unlabelled

```
val v1, v2: int
val done: bool ref = ref false
val transfer: bool -> int option
  fn b =>
    if !done
    then NONE
    else done := true;
      if b
      then SOME v1
      else SOME v2
```

# Trust Model

- ▶ A trust label is assigned to each principal
  - ▶ Alice can be trustfully sent data with confidentiality  $\gamma_A$
  - ▶ Data with integrity  $\iota_A$  can be trustfully received from Alice
- ▶ Only Alice trusts herself:  $\ell_A = \langle \{A : \}, \{A\} \rangle$
- ▶ Only Bob trusts himself:  $\ell_B = \langle \{B : \}, \{B\} \rangle$

## Oblivious Transfer Problem: Labelled and Declassified

```
val v1, v2: int<{A:},{A}>
val done: bool ref<{A:},{A}> = ref false
val transfer: bool{B:} {A}->(A) int option{B:} =
  fn b =>
    if !done
    then NONE
    else done := true;
      if b
      then SOME (declassify {B:} v1)
      else SOME (declassify {B:} v2)
```

# Partitioning Constraints

- ▶ Compiler partitions using annotated program and trust specification
- ▶ Assign a trustworthy principal to each program expression based on its label
- ▶ A principal  $p$  can execute an expression  $e$ 
  - ▶  $p$  is trustworthy enough to access read set  $R$

$$\sqcup_{r \in R} \gamma_r \sqsubseteq \gamma_p$$

- ▶  $p$  is trustworthy enough to modify write set  $W$

$$\iota_p \sqsubseteq \prod_{w \in W} \gamma_w$$

## Oblivious Transfer Problem: Partitioning

```
val v1, v2: int<{A:},{A}>
val done: bool ref<{A:},{A}> = ref false
val transfer: bool{B:} {A}->(A) int option{B:} =
  fn b =>
    if !done
    then NONE
    else done := true;
      if b
      then SOME (declassify {B:} v1)
      else SOME (declassify {B:} v2)
```

## Read Channel Constraints

- ▶ Remote read requests leak control flow information
  - ▶ Alice can infer the current program point from a read request
  - ▶ Control flow leakage should not cause information leakage
- ▶ Associate an upper bound  $\gamma_v^{\text{read}}$  to reads of  $v$  via implicit flows
- ▶ Datum may only be placed on a principal  $p$  satisfying

$$\gamma_v \sqcup \gamma_v^{\text{read}} \sqsubseteq \gamma_p$$

## Oblivious Transfer Problem: Read Channel

Alice and Bob trust Tom:  $\ell_T = \langle \{A ;; B : \}, \{A\} \rangle$

```
val v1, v2: int<{A:}, {A}>
val done: bool ref<{A:}, {A}> = ref false
val transfer: bool{B:} {A}->(A) int option{B:} =
  fn b =>
    if !done
    then NONE
    else done := true;
      if b
      then SOME (declassify {B:} v1)
      else SOME (declassify {B:} v2)
```

## Oblivious Transfer Problem: Read Channel

Alice and Bob trust Sam:  $\ell_S = \langle \{A :: B : \}, \{ \} \rangle$

```
val v1, v2: int<{A:},{A}>
val done: bool ref<{A:},{A}> = ref false
val transfer: bool{B:} {A}->(A) int option{B:} =
  fn b =>
    let val (v1',v2') = (v1,v2) in
      if !done
      then NONE
      else done := true;
         if b
         then SOME (declassify {B:} v1')
         else SOME (declassify {B:} v2')
```

# Declassification Constraints

- ▶ Principals  $P$  should only delegate declassification rights if they trust the data guiding the control flow

$$l_{\mathbf{pc}} \sqsubseteq \{p \mid p \in P\}$$

where declassify occurs at **pc**

## Oblivious Transfer Problem

```
val v1, v2: int<{A:},{A}>
val done: bool ref<{A:},{A}> = ref false
val transfer: bool{B:} {A}->(A) int option{B:} =
  fn b =>
    if !done
    then NONE
    else done := true;
      if endorse {A} b
      then SOME (declassify {B:} v1)
      else SOME (declassify {B:} v2)
```

# Correctness

- ▶ Security assurance: Bad principals  $B$  can only threaten an expression  $e$  with label  $\langle \gamma_e, \iota_e \rangle$  if
  - ▶ (Confidentiality)  $\gamma_e \sqsubseteq \sqcup_{b \in B} \gamma_b$
  - ▶ (Integrity)  $\prod_{b \in B} \iota_b \sqsubseteq \iota_e$
- ▶ Avoid direct threats from a principal  $p$ 
  - ▶ Never send data to  $p$  with confidentiality higher than  $\gamma_p$
  - ▶ Never receive data with integrity lower than  $\iota_p$  from  $p$
- ▶ Avoid indirect threats
  - ▶ Prevent less trusted principals from exploiting downgrading abilities of more trusted principals

# Conclusions

- ▶ Confidentiality and integrity policies don't encode high-level intent
- ▶ Trust endorsements require external reasoning
- ▶ Robustness in extending to concurrent systems?
- ▶ Labels at higher types?