Information Flow Security of Multi-Threaded Distributed Programs

Riccardo Focardi and Matteo Centenaro
Ca’ Foscari University of Venice
{focardi,centenaro}@dsi.unive.it

Third Workshop on Programming Languages and Analysis for Security (PLAS)
Tucson, Arizona, June 8, 2008
Noninterference of multi-threaded distributed apps
Cryptographic primitives are extensively used on today applications

Is it possible to check if programs are secure via NI?

send(encrypt(h,k))
Cryptographic primitives are extensively used on today applications

Is it possible to check if programs are secure via NI?

\texttt{send(encrypt(h,k))}

NI is too strong

Encryption depends on secret data, so we cannot treat its result as a public value
• cryptographic messages respect the Dolev-Yao model
• encryption is randomized using confounders

Abadi, Rogaway [JCRYPTOL’02]
Attacker’s view of an encrypted message: $\square_c$
Equivalent messages: $\square_{c_1} = (\square_{c_2})\rho$, $\rho[c_2 \mapsto c_1]$
Multi-threaded distributed programs must be checked for security

An attacker could be a parallel thread or a program running on a connected host

Sabelfeld, Sands [CSF’00]
A multi-threaded program is strongly secure if it satisfy strong low-bisimilarity requirements
Refine the notion of low-equivalent states to be able to deal with cryptographic messages
Refine the notion of low-equivalent states to be able to deal with cryptographic messages.

Re-use the notion of strong low-bisimulation on programs which uses explicit crypto primitives implemented as expressions.
Related works
Possibilistic NI

Distinguish different ciphertexts from copies

\[ l_1 := \text{encrypt}(h,k); \]
\[ \text{if } h \text{ then} \]
\[ \quad l_2 := \text{encrypt}(h,k); \]
\[ \text{else} \]
\[ \quad l_2 := l_1; \]
Possibilistic NI does not deal with possible concurrent threads

Possibilistic NI and multi-threading

\begin{verbatim}
\texttt{h := true;}
\texttt{if h then}
  \texttt{l := true;}
\texttt{else}
  \texttt{l := false;}
\end{verbatim}

Rejected by the type system, satisfy possibilistic NI
Investigates conditions under which the model proposed by Askarov et. al. is computationally sound.

Propose a variant of the model based on the same idea of patterns (Abadi, Rogaway [JCRYPTOL02]) we adopt in our work.

He still uses a possibilistic notion of NI
Computational probabilistic NI

No multi-threading
A language with Cryptography
The language

An extension with explicit cryptography of the Multi-Threaded While-Language with Message Passing [Sabelfeld and Mantel. SAS 2002].

\[ C,D ::= \text{skip} \mid \text{var} ::= \text{Exp} \mid C_1;C_2 \]
\[ \quad \mid \text{if } B \text{ then } C_1 \text{ else } C_2 \mid \text{while } B \text{ do } C \]
\[ \quad \mid \text{fork(}C\vec{C}\text{)} \mid \text{send(}cid,\text{Exp}\text{)} \mid \text{receive(}cid,\text{var}\text{)} \]

\[ \text{Exp} ::= \text{var} \mid \text{op(}Exp_1, \ldots, Exp_n\text{)} \]

- channels are all public
- we only consider synchronous blocking receive
- cryptographic operations are expressions
Values and expressions

\[ v ::= \bot \mid n \mid b \mid \{v\}_n \mid (v_1, v_2) \]

\[
\begin{align*}
\text{encrypt}(v, n) &= \{v, c\}_n \quad c \leftarrow C \\
\text{decrypt}(&\{v, c\}_n, n) = v \\
\text{newkey} &= k \quad k \leftarrow \text{KEY} \\
\text{pair}(v_1, v_2) &= (v_1, v_2) \\
\text{fst}((v_1, v_2)) &= v_1 \\
\text{snd}((v_1, v_2)) &= v_2
\end{align*}
\]

All undefined cases will be evaluated to \( \bot \)
Why not stop execution?

This would be insecure on stopping by fail

receive(cid, x);
y := decrypt(x,k);
send(cid, l);

expression different from the six above and every boolean expression different from the equality test applyed to ciphertexts will fail
Standard Noninterference
Noninterference

Executing \( C \) the \textbf{observable} behaviour is the same

\begin{align*}
\text{H1} & \quad \text{H2} \\
\text{L} & \quad \text{L}
\end{align*}

\begin{align*}
\text{H'1} & \quad \text{H'2} \\
\text{L'} & \quad \text{L'}
\end{align*}
Low level users can observe every low level variables and every network channel

**equating states, \(\equiv_L\)**

Two states \(s_1 = (m_1, \sigma_1)\) and \(s_2 = (m_2, \sigma_2)\) are _low-equivalent_, noted \(s_1 \equiv_L s_2\), if

1. \(m_1 \equiv_L m_2: \forall l \in L, m_1(l) = m_2(l)\);
2. \(\sigma_1 \equiv_L \sigma_2: \forall cid \in CID, \sigma_1(cid) = \sigma_2(cid)\);
Strongly Secure Programs (Sabelfeld and Sands [CSFW 2000])

**Strong low bisimulation**

Let $\mathcal{R} \subseteq \bigcup_{n \in \mathbb{N}} (CMD^n \times CMD^n)$ be a symmetric relation on multi-threaded programs of equal size. $\mathcal{R}$ is a strong low-bisimulation if whenever

$$\langle C_1 \ldots C_n \rangle \mathcal{R} \langle D_1 \ldots D_n \rangle$$

then $\forall s_1, s_2, i$, such that $s_1 =_L s_2$:

$$\langle C_i, s_1 \rangle \rightarrow \langle C', s'_1 \rangle \text{ implies } \langle D_i, s_2 \rangle \rightarrow \langle D', s'_2 \rangle$$

for some $D', s'_2$ such that $C' \mathcal{R} D', s'_1 =_L s'_2$.

**Strong low-bisimilarity** $\cong _L$ is defined as the union of all strong low-bisimulations.
NI is too strong for crypto

Let $C$ be $l := \text{encrypt}(h, k)$

\[ h: 1234 \quad k: K \]
\[ l: 0 \]

\[ h: 5678 \quad k: K \]
\[ l: 0 \]

\[ = L \]
NI is too strong for crypto

Let $C$ be $l := \text{encrypt}(h, k)$

Let $h: 1234$, $k: K$, $l: 0$.

$C$, $\not= L$

Let $h: 5678$, $k: K$, $l: 0$.

$C$, $\not= L$

Let $h: 1234$, $k: K$, $l: \{1234, c\}_k$.

$\not= L$

Let $h: 5678$, $k: K$, $l: \{5678, c'\}_k$.

$\not= L$
Cryptographic Noninterference
Ciphertexts must be compared properly

Using confounders all encryptions will always produce **new** values

Ciphertexts cannot be considered indistinguishable

- copy of the same message
Ciphertexts must be compared properly

Using confounders all encryptions will always produce new values

Ciphertexts cannot be considered indistinguishable

- copy of the same message
- an attacker can share a key with a trusted entity
  - high level keys vs low level keys
Ciphertexts must be compared properly

Using confounders all encryptions will always produce **new** values

Ciphertexts cannot be considered indistinguishable

- copy of the same message
- an attacker can share a key with a trusted entity
  - high level keys vs low level keys
  - high level keys values $K$ range over $KEY$
Ciphertexts must be compared properly

Using confounders all encryptions will always produce \textbf{new} values

Ciphertexts cannot be considered indistinguishable

- copy of the same message
- an attacker can share a key with a trusted entity
  - high level keys vs low level keys
  - high level keys values $K$ range over $KEY$
  - Variables: $\mathcal{L}, \mathcal{H}, K$
Patterns

Call $PAT$ the set of values plus the undecryptable message $\Box_c$

$p(v)$: what can be observed?

Let $p : VAL \rightarrow PAT$ be defined as follows:

\[
\begin{align*}
p(\bot) &= \bot \\
p(n) &= n \\
p(b) &= b \\
p((v_1, v_2)) &= (p(v_1), p(v_2)) \\
p\{v, c\}_n &= \begin{cases} 
\{p(v), c\}_n & \text{if } n \notin KEY \\
\Box_c & \text{otherwise}
\end{cases}
\end{align*}
\]
IDEA: equal confounders correspond to equal, undecryptable messages

Call a bijection on confounders $\rho : \mathcal{C} \rightarrow \mathcal{C}$

confounder substitution

Two values $v_1$ and $v_2$ are cryptographically-low-equivalent, written $v_1 \approx_{\mathcal{C}} v_2$, if there exists a confounder substitution $\rho$ such that $p(v_1) = p(v_2)\rho$. 
In the previous example we failed to compare these two values:

\{1234, c_1\}_K \quad \{5678, c_2\}_K
Cryptographic-low-equivalent values: Example

In the previous example we failed to compare these two values:

\[ \{1234, c_1\}_K \approx_C \{5678, c_2\}_K \]

Indeed, \( p(\{1234, c_1\}_K) = \square c_1 \), \( p(\{5678, c_2\}_K) = \square c_2 \)
and by taking \( \rho(c_2) = c_1 \) we get \( \square c_1 = \square c_2 \rho \)
Comparing memories and network states

Comparing only correspondent values as done on the original definition does not suffice

```plaintext
if (l1 = l2) then
    send(cid,l3)
else
    send(cid,l4)
```

It looks like a secure program...
Comparing memories and network states (cont.)

$m_1$

$l_1 : \{1234, c_1\}_K$
$l_2 : \{1234, c_1\}_K$
$l_3 : \text{true}$
$l_4 : \text{false}$

$\sigma_1$

$cid :$

$m_2$

$l_1 : \{9999, c'_1\}_K$
$l_2 : \{5678, c'_2\}_K$
$l_3 : \text{true}$
$l_4 : \text{false}$

$\sigma_2$

$cid :$

if (l_1 = l_2) then
    send(cid, l_3)
else
    send(cid, l_4)
Comparing memories and network states (cont.)

$m_1$

$l_1 : \{1234, c_1\}_K$
l_2 : \{1234, c_1\}_K
l_3 : true
l_4 : false

$\sigma_1$

cid : true

$m_2$

$l_1 : \{9999, c'_1\}_K$
l_2 : \{5678, c'_2\}_K
l_3 : true
l_4 : false

$\sigma_2$

cid : false

if (l_1 = l_2) then
  send(cid,l_3)
else
  send(cid,l_4)
Consider these two programs

\[
\begin{align*}
l_1 & := \text{encrypt}(h,k); \\
l_2 & := \text{encrypt}(h,k);
\end{align*}
\]

\[
\begin{align*}
l_1 & := \text{encrypt}(h,k); \\
l_2 & := l_1;
\end{align*}
\]

They cannot be considered noninterfering because lead to memories similar to the one used on the previous example.
A whole state pattern

Extend the $p()$ function to streams:

$$p(v_1.\text{vals}) = p(v_1).p(\text{vals}), \quad \text{vals} \approx_C \text{vals}' \quad \text{if} \quad p(\text{vals}) = p(\text{vals}')_\rho$$
A whole state pattern

Extend the $p()$ function to streams:

$p(v_1.\text{vals}) = p(v_1).p(\text{vals})$, \(\text{vals} \approx_C \text{vals}'\) if
$p(\text{vals}) = p(\text{vals}')\rho$

\[
\begin{align*}
\text{sp}(m) &= \{ (l, p(m(l))) \mid \forall l \in L \} \\
\text{sp}(\sigma) &= \{ (cid, p(\sigma(cid))) \mid \forall cid \in CID \} \\
\text{sp}(m, \sigma) &= \text{sp}(m) \cup \text{sp}(\sigma)
\end{align*}
\]

Patterns equivalence $\equiv_C$

Two memories, networks or states $t_1$ and $t_2$ are cryptographically-low-equivalent, \(t_1 \equiv_C t_2\), if there exists a confounder substitution $\rho$ such that

$\text{sp}(t_1) = \text{sp}(t_2)\rho.$
Pattern: Example

\[m_1\]
\[\begin{align*}
l_1 & : \{1234, c_1\}_K \\
l_2 & : \{1234, c_1\}_K \\
l_3 & : \text{true} \\
l_4 & : \text{false}
\end{align*}\]

\[m_2\]
\[\begin{align*}
l_1 & : \{9999, c'_1\}_K \\
l_2 & : \{5678, c'_2\}_K \\
l_3 & : \text{true} \\
l_4 & : \text{false}
\end{align*}\]

\[\text{sp}(m_1) = \{(l_1, \square c_1), (l_2, \square c_1), (l_3, \text{true}), (l_4, \text{false})\}\]

\[\text{sp}(m_2) = \{(l_1, \square c'_1), (l_2, \square c'_2), (l_3, \text{true}), (l_4, \text{false})\}\]

Clearly, \(m_1 \not\preceq C m_2\)
Why simultaneously observe patterns of memories and channels?

\[
l := \text{encrypt}(h1,k);
\]
\[
\text{send}(ch,l),
\]
\[
(*) \text{ if } h \text{ then}
\]
\[
l := \text{encrypt}(h2,k);
\]
\[
\text{else}
\]
\[
\text{skip};
\]

\begin{align*}
\text{\color{red}{m_1}} & \quad \text{\color{green}{\sigma_1}} \\
\begin{array}{l}
h : \text{true} \\
h_1 : 1234 \\
h_2 : 5678 \\
l : \{1234, c_1\}_K
\end{array} & \quad \begin{array}{l}
ch : \\
\{1234, \, c_1\}_K
\end{array}
\end{align*}

\begin{align*}
\text{\color{red}{m_2}} & \quad \text{\color{green}{\sigma_2}} \\
\begin{array}{l}
h : \text{false} \\
h_1 : 4443 \\
h_2 : 5556 \\
l : \{4443, \, c_1'\}_K
\end{array} & \quad \begin{array}{l}
ch : \\
\{4443, \, c_1'\}_K
\end{array}
\end{align*}

\[m_1 =_C m_2, \quad \sigma_1 =_C \sigma_2\]
Why simultaneously observe patterns of memories and channels?

\[
l := \text{encrypt}(h1,k);
\]
\[
\text{send}(ch,l),
\]
\[
\text{if } h \text{ then}
\]
\[
(1) \quad l := \text{encrypt}(h2,k);
\]
\[
\text{else}
\]
\[
(2) \quad \text{skip};
\]

\[
\begin{align*}
m'_1 & = \{k \}
\end{align*}
\]

\[
\begin{align*}
\sigma_1 & \quad \sigma_2
\end{align*}
\]

\[
\begin{align*}
h & : \text{true} & h & : \text{false} \\
h_1 & : 1234 & h_1 & : 4443 \\
h_2 & : 5678 & h_2 & : 5556 \\
l & : \{5678, c_2\}_K & l & : \{4443, c'_1\}_K
\end{align*}
\]

\[
\begin{align*}
\text{ch} & : \{1234, c_1\}_K & \text{ch} & : \{4443, c'_1\}_K
\end{align*}
\]

\[
m'_1 = C \quad m'_2, \quad \sigma_1 = C \quad \sigma_2 \quad : (\)
\]
Why simultaneously observe patterns of memories and channels?

\[ m'_1 \]
\[ h : \text{true} \]
\[ h_1 : 1234 \]
\[ h_2 : 5678 \]
\[ l : \{5678, c_2\}_K \]

\[ m_2 = m'_2 \]
\[ h : \text{false} \]
\[ h_1 : 4443 \]
\[ h_2 : 5556 \]
\[ l : \{4443, c'_1\}_K \]

\[ \sigma_1 \]
\[ ch : \{1234, c_1\}_K \]

\[ \sigma_2 \]
\[ ch : \{4443, c'_1\}_K \]

\[ sp(m'_1, \sigma_1) = \{(l, \square c_2), (ch, \square c_1)\} \]

\[ sp(m'_2, \sigma_2) = \{(l, \square c'_1), (ch, \square c'_1)\} \]

\[ (m'_1, \sigma_1) \cong (m'_2, \sigma_2) \]
Secure programs - Assumptions

Confounder unicity
Values encrypted with high level keys and with the same confounder are exactly the same.

High level key safety
High key variables \( k \in \mathcal{K} \) can only contain high key values \( K \in KEY \). On the other hand, we never allow high key values to occur unprotected in the low level memory and on the network.
Given a state $s$, $s \vdash k$ denote that $k \in \text{values}(\text{sp}(s))$ where $\text{values}(p)$ is the set of all atomic values occurring in pattern $p$.

A state $s = (m, \sigma)$ is key-safe if

1. $\forall k \in K, m(k) \in \text{KEY}$;
2. $s \vdash n$ implies $n \notin \text{KEY}$;

$KS$ is set of key-safe states.
Let $\mathcal{R} \subseteq \bigcup_{n \in \mathbb{N}} (CMD^n \times CMD^n)$ be a symmetric relation on multi-threaded programs of equal size.

**Strong cryptographic low-bisimulation**

$\mathcal{R}$ is a **strong cryptographic low-bisimulation** if whenever $\langle C_1 \ldots C_n \rangle \mathcal{R} \langle D_1 \ldots D_n \rangle$ then

$\forall i, \forall s_1, s_2 \in KS$, such that $s_1 =_C s_2$:

\[\langle C_i, s_1 \rangle \rightarrow \langle \vec{C}', s_1' \rangle \text{ implies } \langle D_i, s_2 \rangle \rightarrow \langle \vec{D}', s_2' \rangle\]

for some $\vec{D}', s_2'$ such that $\vec{C}' \mathcal{R} \vec{D}'$, $s_1' =_C s_2'$.

**Strong cryptographic low-bisimilarity** $\simeq_C$ is defined as the union of all strong cryptographic low-bisimulations.
Key-safe program

A program $C$ is key-safe if
- $\text{receive}(cid, k)$ never occurs in $C$;
- $k := \text{Exp}$ occurring in $C$ implies $\text{Exp} = k'$ or $\text{Exp} = \text{newkey}$.

Key-safe preservation

Let $C$ and $D$ be two key-safe programs. If $\forall s_1, s_2 \in KS$, with $s_1 =_C s_2$,

$$\langle |C, s_1| \rangle \rightarrow \langle |C', s'_1| \rangle$$ implies $$\langle |D, s_2| \rangle \rightarrow \langle |D', s'_2| \rangle$$

for some $D', s'_2$ such that $s'_1 =_C s'_2$

then $s'_1, s'_2 \in KS$. 

Riccardo Focardi and Matteo Centenaro

PLAS 2008
Secure program

A program $\tilde{C}$ is secure if it is key-safe and $\tilde{C} \simeq_C \tilde{C}$. 

Riccardo Focardi and Matteo Centenaro

PLAS 2008
Hook-up
Low expressions

Let $v_1 = \text{Exp} \downarrow^{m_1}$ and $v_2 = \text{Exp} \downarrow^{m_2}$

**Low expression**

$Exp$ is low if

1. $(m_1[l \mapsto v_1], \sigma_1) =_{C} (m_2[l \mapsto v_2], \sigma_2)$;
2. $(m_1, \sigma_1[cid \mapsto v_1.vals]) =_{C} (m_2, \sigma_2[cid \mapsto v_2.vals])$.

otherwise $Exp$ is *high*. 
Low expressions

Let $v_1 = \text{Exp} \downarrow^{m_1}$ and $v_2 = \text{Exp} \downarrow^{m_2}$

**Low expression**

$\text{Exp}$ is low if

- $(m_1[l \mapsto v_1], \sigma_1) =_C (m_2[l \mapsto v_2], \sigma_2)$;
- $(m_1, \sigma_1[cid \mapsto v_1.\text{vals}]) =_C (m_2, \sigma_2[cid \mapsto v_2.\text{vals}])$.

otherwise $\text{Exp}$ is high.

If $\text{Exp}$ is low then $\forall m_1 =_C m_2$, we have $\text{Exp} \downarrow^{m_1} \approx_C \text{Exp} \downarrow^{m_2}$
Secure contexts

\[ \mathbb{C}[\overrightarrow{C}_1, \overrightarrow{C}_2] ::= \text{skip} \mid h := \text{Exp} \mid l := \text{Exp}_L \mid [\overrightarrow{C}_1]; [\overrightarrow{C}_2] \]
\[ \mid k := k' \ (k, k' \in \mathcal{K}) \mid k := \text{newkey} \ (k \in \mathcal{K}) \]
\[ \mid \text{if } B_L \text{ then } [\overrightarrow{C}_1] \text{ else } [\overrightarrow{C}_2] \mid \text{while } B_L \text{ do } [\overrightarrow{C}_1] \]
\[ \mid \text{fork}([\overrightarrow{C}_1][\overrightarrow{C}_2]) \mid \text{send}(cid, \text{Exp}_L) \]
\[ \mid \text{receive}(cid, var) \ (var \not\in \mathcal{K}) \mid \langle [\overrightarrow{C}_1][\overrightarrow{C}_2] \rangle \]
Secure contexts

\[ \mathbb{C}[\vec{\bullet}_1, \vec{\bullet}_2] ::= \text{skip} \mid h := \text{Exp} \mid l := \text{Exp}_L \mid [\bullet_1]; [\bullet_2] \]
| \[ k := k' \ (k, k' \in \mathcal{K}) \mid k := \text{newkey} \ (k \in \mathcal{K}) \]
| \[ \text{if } B_L \text{ then } [\bullet_1] \text{ else } [\bullet_2] \mid \text{while } B_L \text{ do } [\bullet_1] \]
| \[ \text{fork}([\bullet_1][\vec{\bullet}_2]) \mid \text{send}(cid, \text{Exp}_L) \]
| \[ \text{receive}(cid, var) \ (var \not\in \mathcal{K}) \mid \langle[\vec{\bullet}_1][\vec{\bullet}_2]\rangle \]

Secure congruence

If \( \vec{C}_1 \approx_C \vec{C}_1', \vec{C}_2 \approx_C \vec{C}_2' \) then

- \( \mathbb{C}[\vec{C}_1, \vec{C}_2] \approx_C \mathbb{C}[\vec{C}_1', \vec{C}_2'] \);
- Let \( \mathbb{D}[\vec{\bullet}_1, \vec{\bullet}_2] = \text{if } B \text{ then } \vec{\bullet}_1 \text{ else } \vec{\bullet}_2, \) with \( B \text{ high} \). Then, \( \vec{C}_1 \approx_C \vec{C}_2 \) implies \( \mathbb{D}[\vec{C}_1, \vec{C}_2] \approx_C \mathbb{D}[\vec{C}_1', \vec{C}_2'] \).
Let $\vec{C}_1$, $\vec{C}_2$ be secure programs. Then

- $\mathbb{C}[\vec{C}_1, \vec{C}_2]$ is secure;
- Let $\mathbb{D}[\bullet_1, \bullet_2] = \text{if } B \text{ then } \bullet_1 \text{ else } \bullet_2$, with $B$ high. Then, $\vec{C}_1 \approx C \vec{C}_2$ implies that $\mathbb{D}[\vec{C}_1, \vec{C}_2]$ is secure.
Type system
Sabelfeld and Mantel (SAS 2002) type system can be adapted to our purposes.
Sabelfeld and Mantel (SAS 2002) type system can be adapted to our purposes.

\[ \vec{C} \leftrightarrow \vec{C}' : \vec{Sl} \]

\[ \text{Exp} : \tau, \quad \tau : \text{low, high, key} \]
Sabelfeld and Mantel (SAS 2002) type system can be adapted to our purposes.

\[ \vec{C} \rightarrow \vec{C}' : \vec{Sl} \]

Exp : \( \tau, \ \tau : low, high, key \)

What’s new?

- rule for typing expressions (encrypt and decrypt)
- key handling in commands
Results

If $\text{Exp} : \text{low}$ then $\text{Exp}$ is low according to definition given before

If $\vec{C} \hookrightarrow \vec{C}' : \vec{S}l$ then $\vec{C}' \approx_C \vec{S}l$.

Program Noninterference

If $\vec{C} \hookrightarrow \vec{C}' : \vec{S}l$, then $\vec{C}'$ is secure, i.e., $\vec{C}'$ is key-safe and $\vec{C}' \approx_C \vec{C}'$. 
Conclusions

Noninterference in a multi-threaded distributed setting, with insecure channels and explicit cryptographic primitives

Secure programs are proved to be compositional

Future works

- “session” keys
- public key cryptography
- . . .
Information Flow Security of Multi-Threaded Distributed Programs

Riccardo Focardi and Matteo Centenaro
Ca’ Foscari University of Venice
{focardi,centenaro}@dsi.unive.it

Third Workshop on Programming Languages and Analysis for Security (PLAS)
Tucson, Arizona, June 8, 2008

Thank you!