Hybrid Monitoring of Attacker Knowledge

Frédéric Besson, Nataliia Bielova, Thomas Jensen
INRIA

June 29, 2016
## Information flow control

<table>
<thead>
<tr>
<th>Noninterference</th>
<th>Quantitative Information Flow</th>
</tr>
</thead>
<tbody>
<tr>
<td>Secret input does not flow into public output</td>
<td>How much information (in bits) about secret a program leaks to the output?</td>
</tr>
</tbody>
</table>
## Information flow control

### Noninterference
- Secret input does not flow into public output

### Attacker’s knowledge
- What information about the secret is flown to the output in concrete program execution?

### Quantitative Information Flow
- How much information (in bits) about secret a program leaks to the output?
A program which is not secure

if h1 = 1 then x = 1
else skip;
if h2 = 1 then l = 1
else l = x;
output l

set of secure executions starting with l=1, x=1

set of insecure executions starting with l=1, x=0
Security Definition

- **TINI: Termination-Insensitive Noninterference**
  - Program P is TINI if for all low-equivalence classes:
What does an attacker learn?

```plaintext
if h1 = 1 then x = 1
else skip;
if h2 = 1 then l = 1
else l = x;
output l
```

insecure executions

[\text{\{l=1, x=0\}}

```
\text{h1=0} \land \text{h2=0}
```

0 1
What does an attacker learn?

\[
\begin{align*}
\text{if } h_1 = 1 & \text{ then } x = 1 \\
\text{else } & \text{ skip; } \\
\text{if } h_2 = 1 & \text{ then } l = 1 \\
\text{else } l = x; \\
\text{output } l
\end{align*}
\]

attacker knows values of both secrets

insecure executions

\( h_1 = 0 \land h_2 = 0 \)
What does an attacker learn?

\[
\begin{align*}
\text{if } h1 &= 1 \text{ then } x = 1 \\
\text{else skip;}
\end{align*}
\]
\[
\begin{align*}
\text{if } h2 &= 1 \text{ then } l = 1 \\
\text{else } l &= x; \\
\text{output } l
\end{align*}
\]

attacker knows values of both secrets

insecure executions

attacker knows some information about secrets
A program which is secure

```
l = 0;
if h = 1 then skip
else
  x = 5;
  while x > 0 do
    x = x-1; l = x;
  output l
```

- Does any dynamic/hybrid monitor accept all executions of this program?
A program which is secure

```plaintext
l = 0;
if h = 1 then skip
else
    x = 5;
    while x > 0 do
        x = x-1; l = x;
    output l
```

<table>
<thead>
<tr>
<th>Dynamic [h=0]</th>
<th>Hybrid [h=1]</th>
</tr>
</thead>
<tbody>
<tr>
<td>...</td>
<td>... branch taken</td>
</tr>
<tr>
<td>... branch taken</td>
<td>... static analysis</td>
</tr>
<tr>
<td>block execution</td>
<td>... block execution since</td>
</tr>
<tr>
<td>due to low assignment in</td>
<td>1 could be modified</td>
</tr>
<tr>
<td>high context</td>
<td>in else-branch</td>
</tr>
</tbody>
</table>

- **Dynamic monitors block too early**
  - [Zdancewic ’02, Austin and Flanagan ’10]

- **Hybrid monitors block due to imprecision of static analysis**
  - [Le Guernic ‘07, Russo and Sabelfeld ’10, Besson et al ‘13]
Challenges

• How to track attacker’s knowledge?

• How to make a monitor accept more secure executions?

Answer:
Hybrid monitoring of attacker’s knowledge
Hybrid monitor

Dynamic + Static analysis

- Dynamic analysis monitors one execution
- Static analysis is called on-the-fly for non-executed branches

- Two sets of rules: one for dynamic + one for static
Hybrid monitor

\[(P, \rho, \kappa) \downarrow (\rho', \kappa')\]

- \(\rho, \rho': \text{Env} \cup \{\cdot\}\)
- \(\kappa, \kappa': \text{Var} \rightarrow K\) labeling with knowledge
- \(\text{Env}\) for dynamic analysis
- \(\cdot\) for static analysis
Expressive knowledge domain

\[ \kappa: \text{Var} \to K \]

- \( \kappa(x) \) splits the initial environments in equivalence classes w.r.t. the possible values of \( x \)

\[ \kappa(x) : \]

- \([l = 0]\) insecure executions
- 0
- 1

T (unknown value)
Expressive knowledge domain

\[ \kappa : \text{Var} \rightarrow K \]

\[ K \triangleq \text{Env} \rightarrow \text{Value} \cup \{\top, \bot\} \]

- \( \kappa(x)(\rho) = v \) if the program terminates then \( x \) has value \( v \)
- \( \kappa(x)(\rho) = \top \) no information (\( x \) can have any value)
- \( \kappa(x)(\rho) = \bot \) the program certainly does not terminate on \( \rho \)
if h1 = 1 then x = 1
else skip;
if h2 = 1 then l = 1
else l = x;
output l
if $h_1 = 1$ then $x = 1$
else skip;
if $h_2 = 1$ then $l = 1$
else $l = x$;
output $l$
[h1 = 0, h2 = 1]

\begin{align*}
\text{if } h1 = 1 & \text{ then } x = 1 \\
\text{else skip;} \\
\text{if } h2 = 1 & \text{ then } l = 1 \\
\text{else } l & = x; \\
\text{output } l
\end{align*}

REAL KNOWLEDGE

[\ l = 0, x=0\]

\[\kappa(x) : \]

\[\kappa(l) : \]

0 1

0
\[ h_1 = 0, h_2 = 1 \]

**static analysis**

\[
\begin{align*}
\text{if } h_1 = 1 & \text{ then } x = 1 \\
\text{else} & \text{ skip; }
\end{align*}
\]

\[
\begin{align*}
\text{if } h_2 = 1 & \text{ then } l = 1 \\
\text{else} & \text{ } l = x;
\end{align*}
\]

output \( l \)

\[
\begin{align*}
\kappa(x): & \text{ } 0 \\
\kappa(l): & \text{ } 0
\end{align*}
\]

**REAL KNOWLEDGE**

\[ l = 0, x=0 \]

The result of static analysis only applies to environments where \( h_1 = 1 \).
if $h_1 = 1$ then $x = 1$
else skip;
if $h_2 = 1$ then $l = 1$
else $l = x$;
output $l$
if $h_1 = 1$ then $x = 1$
else skip;
if $h_2 = 1$ then $l = 1$
else $l = x$;
output $l$
The result of static analysis only applies to environments where h2=0.
if \( h_1 = 1 \) then \( x = 1 \) else skip;
if \( h_2 = 1 \) then \( l = 1 \) else \( l = x \); static analysis
output \( l \)

The new knowledge in these environments comes from the knowledge of \( x \)
if $h_1 = 1$ then $x = 1$
else skip;

if $h_2 = 1$ then $l = 1$
else $l = x$;

output $l$

The new knowledge in these environments comes from the knowledge of $x$. 
if $h_1 = 1$ then $x = 1$
else skip;
if $h_2 = 1$ then $l = 1$
else $l = x$;
output $l$

The new knowledge in these environments comes from the knowledge of $x$
if \( h_1 = 1 \) then \( x = 1 \) else skip;
if \( h_2 = 1 \) then \( l = 1 \) else \( l = x \);
output \( l \)

The knowledge in current execution applies to environments where \( h_2 = 1 \)
[h1 = 0, h2 = 1]

\[
\begin{align*}
\text{if } h1 &= 1 \text{ then } x = 1 \\
\text{else skip; } \\
\text{if } h2 &= 1 \text{ then } l = 1 \\
\text{else } l &= x; \\
\text{output } l
\end{align*}
\]

The knowledge in current execution applies to environments where h2=1
if \( h_1 = 1 \) then \( x = 1 \)
else skip;

if \( h_2 = 1 \) then \( l = 1 \)
else \( l = x \);

output \( l \)

\([h_1 = 0, h_2 = 1]\)

\(\kappa(x):\)

\(\kappa(l):\)

REAL KNOWLEDGE

\([l = 0, x=0]\)
if \( h_1 = 1 \) then \( x = 1 \)
else skip;
if \( h_2 = 1 \) then \( l = 1 \)
else \( l = x \);
output \( l \)
[h1 = 0, h2 = 1]

if h1 = 1 then x = 1
else skip;
if h2 = 1 then l = 1
else l = x;
output l

REAL KNOWLEDGE

= APPROXIMATED KNOWLEDGE
Implementation

• Symbolic representation of knowledge

\[ K^b \subset \mathcal{P}(\mathbb{F} \times \mathbb{E}) \times \mathbb{F} \]

- \text{propositional formulas}
- \text{program expressions}

• \( (f, e) \in \mathbb{F} \times \mathbb{E} \) returns the value of \( e \) when \( f \) holds in \( \rho \):
  - if \( \llbracket f \rrbracket_\rho \) then \( \llbracket e \rrbracket_\rho \) else \( \top \)

• \( \phi \in \mathbb{F} \) specifies when the knowledge is \( \bot \)
  - if \( \llbracket \phi \rrbracket_\rho \) then \( \bot \)
Result 1: Correctness guarantee

- Hybrid monitor over-approximates attacker’s knowledge

![Diagram showing insecure executions and knowledge approximations]

- Approximated knowledge: $h_1 = 1$
- Real knowledge: $h_1 = 1 \lor h_2 = 1$
- [public = 0]
Result 2: Precision

If $h_1 = 1$ then $x = 1$
else skip;
if $h_2 = 1$ then $l = 1$
else $l = x$;
output $l$

APPROXIMATED KNOWLEDGE
$h_1 = 1 \lor h_2 = 1$

REAL KNOWLEDGE
$h_1 = 1 \lor h_2 = 1$

insecure executions
Result 3: Enforcement of noninterference

\[ h = 1, \ x = 1, \ y = 0 \]

\[
\text{if } h = 1 \text{ then } l = x + y; \\
\text{else } l = x - y; \\
\text{output } l
\]

ACCEPTED

Approximated Knowledge

- No knowledge

Real Knowledge

- No knowledge

Secure executions
Result 4: Provably more permissive monitor

```plaintext
l = 0;
if h = 1 then skip
else
  x = 5;
  while x > 0 do
    x = x - 1; l = x;
output l
```

Our monitor combined with inlined dynamic monitor accepts all executions of this secure program

(More details in the paper)
Conclusions

• **Hybrid monitor tracks attacker’s knowledge**
  ▪ more precise than [Besson et al. CSF’13]
  ▪ modeled and proved correct
  ▪ enforces noninterference (TINI)
  ▪ has running prototype

• **Combination with another monitor**
  ▪ proved sound (TINI)
  ▪ proved more permissive than previous monitors
Postdoc position

- Information flow control
- Security monitors and type systems
- Soundness and permissiveness

- Starting date: flexible, Nov 2016 – Jun 2017
- Duration: 1 year
- Location: INRIA Sophia Antipolis (Nice, France)
Conclusions

- **Hybrid monitor tracks attacker’s knowledge**
  - more precise than [Besson et al.’13]
  - modeled and proved correct
  - enforces noninterference (TINI)
  - has running prototype

- **Combination with another monitor**
  - proved sound (TINI)
  - proved more permissive