Hybrid monitoring of Attacker Knowledge

F. Besson, N. Bielova, T. Jensen

Dynamic Information Flow analysis

Assign "high-low (H-L)" security labels to

- values (track confidential information)
- program counter (track side effects under secret control)

Enforcement: halt execution before output of H values.

```
l = 0;
if h then l = 1;
return l;
```

No-sensitive-upgrade

Just tracking labels is not enough:

```
int secret s;    // s ∈ {0,1}
int public p,q;p := 0; q := 1;
                          s=0          s=1
if s == 0 then
  q := 0;                 p=0,q=0      skip
if q == 1 then
  p := 1;                 skip         p=1,q=1
                          p=0          p=1
```

No upgrade of security label under secret control

NSU is (too) restrictive

If the upgraded value is not read afterwards

```
l = 0;
if h then l = 1;
l = 2;
return l;
```

If the upgraded value is the same in two branches.

```
if h then l = 1 else l = 1;
return l;
```

Attacker knowledge

Compute a description of the knowledge that an attacker obtains by observing a given output.

Attacker model:

- "Gadget" attacker P : a program,
- Can see/set initial low values,
- Can observe low output.

Attacker knowledge

The set of initial environments E that lead to a given final output v after execution of P.

Definition of knowledge

Let ↓ denote the semantics of the language

- (P,E) ↓ v

Let [E] denote the low-equivalent environments of E.

The **knowledge K** of an attacker P that observes v when running P on initial environment E is defined by

$$K(E,v) = \{ E' \in [E] : (P,E') \downarrow v \}.$$

```
if h1 then l = 1;
if h2 then l = 1;
return l;
```

Knowledge about E
when seeing 1
with E(l) = 0
is
E(*h1) or E(h2) is true.*

Non-interference

Characterisation of non-interference in terms of knowledge (Askarov & Sabelfeld):

A program is (termination-insensitive) non-interferent if attacker learns nothing more by observing a given output v than by just observing non-interference.

Formally, for all initial environments E,

$$K(E,v) = \{ E' \in [E] \mid (P,E') \downarrow \}.$$

Approximating knowledge

Goal: compute attacker knowledge by program analysis.

Not always computable - have to accept safe approximations.

A safe approximation of attacker knowledge **over-estimates** the knowledge obtained by an attacker.

More knowledge ≈ smaller set of initial environments.

Eg. safe to approximate

h1 or h2

by

h1 and h2.

```
if h1 then l = 1;
if h2 then l = 1;
return l;
```

Hybrid monitor of knowledge

Compute an estimation of knowledge in each variable.

State: (E,K) where

- E is an environment of values
- K is an environment of knowledge

Hybrid monitor

- (P, (E,K)) $\Downarrow$ (E',K')
- executes the program
- computes the knowledge stored in each variable

Knowledge representation

Knowledge domain:

$$\text{Know} : \text{Env} \rightarrow \text{Val} \cup \{\top , \bot\}.$$

Environment of knowledge:

$$K : \text{Var} \rightarrow \text{Know}.$$

Intuition:

K(x) = a function that maps initial env. E to the value of x that E will lead to.

Hybrid monitor

A static and a dynamic part

- dynamic part executes program on concrete values,
- static part analyses code when value are not available.

For conditionals,

- use dynamic part to execute the taken branch,
- use static part to analyse non-executed branch, in order to refine knowledge computation.

Hybrid monitor rules

Static rules are used when no concrete values are available.

$$\textsc{assignDyn}\ \frac{[\![e]\!]_\rho = v \quad (\!|e|\!)_\kappa = e^\sharp}{(x := e, (\lfloor\rho\rfloor, \kappa)) \Downarrow (\lfloor\rho[x \mapsto v]\rfloor, \kappa[x \mapsto e^\sharp])}$$

$$\textsc{assignStat}\ \frac{(\!|e|\!)_\kappa = e^\sharp}{(x := e, (\bot, \kappa)) \Downarrow (\bot, \kappa[x \mapsto e^\sharp])}$$

Hybrid monitor rules

Conditionals

$$\textsc{ifDyn}\ \frac{C[\![e]\!]_\rho = \alpha \qquad (c_\alpha, (\lfloor\rho\rfloor, \kappa)) \Downarrow (\rho', \kappa_\alpha) \qquad (c_{\bar{\alpha}}, (\bot, \kappa)) \Downarrow (\bot, \kappa_{\bar{\alpha}})}{(\mathsf{if}\ e\ \mathsf{then}\ c_{tt}\ \mathsf{else}\ c_{ff}, (\lfloor\rho\rfloor, \kappa)) \Downarrow (\rho', \mathbb{IF}(C[(\!|e|\!)_\kappa], \kappa_{tt}, \kappa_{ff}))}$$

$$\textsc{ifStat}\ \frac{(c_{tt}, (\bot, \kappa)) \Downarrow (\bot, \kappa_{tt}) \qquad (c_{ff}, (\bot, \kappa)) \Downarrow (\bot, \kappa_{ff})}{(\mathsf{if}\ e\ \mathsf{then}\ c_{tt}\ \mathsf{else}\ c_{ff}, (\bot, \kappa)) \Downarrow (\bot, \mathbb{IF}(C[(\!|e|\!)_\kappa], \kappa_{tt}, \kappa_{ff}))}$$

where

$$\mathbb{IF}(c, \kappa_1, \kappa_2)(x) = IF(c(x), \kappa_1(x), \kappa_2(x))$$

$$\kappa(x) \quad (\!|n|\!)_\kappa = \lambda\rho.\lfloor n\rfloor \quad (\!|e_1 \oplus e_2|\!)_\kappa = (\!|e_1|\!)_\kappa \oplus^\sharp (\!|e_2|\!)_\kappa$$

Example

```
1 if h then                                    Program 5
2   z := x + y
3 else
4   z := y - x;
5 output z
```

$$\kappa(z) = \lambda\rho.if(C[\![h]\!]_\rho, [\![x+y]\!]_\rho, [\![y-x]\!]_\rho)$$

Results

The monitor is

- sound wrt the standard semantics,
- transparent (no change of semactics),
- can enforce non-interference,
- can be to other dynamic monitors
  - first use knowledge then use the dynamic monitor
  - track knowledge in shadow variables
  - applied to inprove NSU.