ModInf: Exploiting Reified Computational Dependencies for Information Flow Analysis

Jens Van der Plas · Jens Nicolay · Wolfgang De Meuter · Coen De Roover

Context

```
(define a #t)
(define a2 (source a))
(define (b x) x)
(define (set-b)
  (set! b (lambda (x) #f)))
(if a2 (set-b))
(define res (b 10))
(sink res)
```

Information flow analysis is used to detect flows of information that decrease the security of an application, e.g., statically using taint analysis.

```
(define x (read))
(define y (read))
(define z (+ x y))
(display z)
```

Explicit information flow: data dependence
"z depends on x and y"

```
(define result #f)
(define input (read))
(if (> input 0)
  (set! result #t))
```

Implicit information flow: control dependence
"result depends on input"

Information flow

```
(define x #t)
(define x-s (source x))
(define san (sanitize x-s))
(if x-s
  (begin
    (sink san)
    (display san)))
```

```
(define x #t)
(define a2 (source a))
(define (b x) x)
(define (set-b)
  (set! b (lambda (x) #f)))
(if a2 (set-b))
(define res (b 10))
(sink res)
```

Approach

We extend the inter-component data flow information of a modular analysis with intra-component flow information, tracked through accesses of the analysis store. We obtain IFC by tracing this flow information instead of by relying on specific lattices or analysis infrastructure. Our approach can handle both explicit and implicit flow information.

Validation

Preliminary validation
✓ 9 hand-crafted programs containing complex taint flows
  • Using sources, sinks and sanitizers
  • Containing explicit and implicit information flow
✓ Validation using type lattice and no context sensitivity
  • Other lattices and context sensitivities are supported
✓ Detected all harmful flows: no false negatives (sound)

Paper link