A Derivation Framework for Dependent Security Label Inference

Peixuan Li and Danfeng Zhang
Pennsylvania State University
University Park, PA, 16801
{pzl129, zhang}@cse.psu.edu
Background

- Information Flow Analysis
  - Study: how information **propagate** through the system
  - Goal: prevent leakage of sensitive information
Background

- Information Flow Analysis
  - Annotate the program with security labels
  - Track and validate flows in program
    - Explicit Flow
    - Implicit Flow

Security Specifications

```
a, c, d : P;
b : S;
```

Program (secure)

```plaintext
1  x := a + z;
2  if (d > 0) then y := b;
3  if (d < 0) then x := y;
4  c := x;
```
Program Annotation

- Annotation Burden
  - Time-consuming and error-prone
  - Requiring good knowledge of implementation details
  - Hard to distinguish incorrect label from false-positive
Security Label Inference

- **Goal:** to infer the security labels
  - **Lattice Labels** [Denning 1976]
    - Rehof-Mogensen Algorithm [Rehof et al. 1999]
    - Implemented in Jif [Myers et al. 2006]

- **Expressiveness**
  - Security levels are **unknown** in static time [Chen et al. 2018; Li and Zhang 2017]
  - Security levels are **not fixed** during execution [Murray et al. 2016; Lourenço and Caires 2015; Zhang et al. 2015;]

- **Precision**
  - Limited to **path-insensitive** analyses [Li and Zhang 2017; Zhang et al. 2015]
Dependent Security Labels

- Dependent Label: labels that may depend on concrete program states
  - Ternary Labels [Li and Zhang 2017]: $(d > 0)?S : P$
  - Predicate Labels [Murray et al. 2016, Polikarpova et al. 2018]: $\{d \leq 0\}$ ➤ Limited to two-point Lattice
  - Permission Labels [Chen et al. 2018]: $\{(\oplus p \oplus q, S), (\oplus p \oplus q, P), (\ominus p \oplus q, S), (\ominus p \oplus q, P)\}$
  - Dynamic Labels [Myers et al. 2006]: $l : Label; \quad x : l$

- Inference Engines
  - Liquid Type [Rondon et al. 2008]
  - SMT Solver
  - Permission Label Inference [Chen et al. 2018] ➤ Exponential
Goal: to efficiently infer the dependent security labels

- Ternary label
- Predicate as Labels
- Permission Predicates
- Dynamic Labels

Inference Engines

- One-Shot
- Early-Accept
- Early-Reject
- Hybrid

Evaluation

Derivation Framework

Core Constraint Language

Dependent Security Labels
Core Constraint Language

- **Syntax**
  - **Label**
  - **Label Constraint**
  - **Predicated Constraint**

$$\tau ::= \ell | \alpha, \text{ where } \ell \in \mathcal{L}, \alpha \in \text{CVars}$$

$$C ::= \tau_1 \sqsubseteq \tau_2 | C_1 \land C_2$$

$$C ::= P \rightarrow C | C \land C$$

Any decidable predicates

Security Specifications

- $a, c, d : P$
- $b : S$

$x, y, z : \ ???$

Program (secure)

1. $x := a + z$
2. if $(d > 0)$ then $y := b$
3. if $(d < 0)$ then $x := y$
4. $c := x$

Encoding details are in the paper…
Goal: to efficiently infer the **dependent** security labels

- Ternary label
- Predicate as Labels
- Permission Predicates
- Dynamic Labels

**Inference Engines**

- One-Shot
- Early-Accept
- Early-Reject
- Hybrid

**Dependent Security Labels**

- Core Constraint Language

**Derivation Framework**

**Evaluation**
Insights: deriving constraints into a more manageable format.

Constraints (satisfiable)
- true → $P \subseteq \alpha_x \land \alpha_x \subseteq P$
- $d > 0 \rightarrow S \subseteq \alpha_y$
- $\neg (d > 0) \rightarrow P \subseteq \alpha_y$
- $d < 0 \rightarrow \alpha_y \subseteq \alpha_x$

Ideal constraint format:

1. **Partition**
   - $d > 0$:
     - $\alpha_y$: $S$;
     - $\alpha_x$: $P$;
   - $d \leq 0$:
     - $\alpha_y$: $P$;
     - $\alpha_x$: $P$;

2. **Derive**
   - $d > 0$ → $P \subseteq \alpha_x \land \alpha_x \subseteq P \land S \subseteq \alpha_y$
   - $d \leq 0$ → $P \subseteq \alpha_x \land \alpha_x \subseteq P \land P \subseteq \alpha_y \land \alpha_y \subseteq \alpha_x$

- No overlapping predicates
  → Construct a global solution by merging local solutions

Derivation Framework
Derivation Framework

- **Partitioning**
  - Non-overlapping predicates

- **Derivations**
  - **Sound Derivation:**
    - solutions on sound-derivation always work for original constraints.
  - **Complete Derivation:**
    - solutions on original constraints always work for complete-derivations.
  - **Equivalent Derivation**
Solution from Sound-Derivations always work for original constraints.

Derivation 1: (sound):

\[ \text{true } \rightarrow P \sqsubseteq \alpha_x \land \alpha_z \sqsubseteq \alpha_x \land \alpha_x \sqsubseteq P \land S \sqsubseteq \alpha_y \land P \sqsubseteq \alpha_y \land \alpha_y \sqsubseteq \alpha_x \]

Constraints (satisfiable)

- \( \text{true } \rightarrow P \sqsubseteq \alpha_x \land \alpha_x \sqsubseteq P \)
- \( d > 0 \rightarrow S \sqsubseteq \alpha_y \)
- \( \neg(d > 0) \rightarrow P \sqsubseteq \alpha_y \)
- \( d < 0 \rightarrow \alpha_y \sqsubseteq \alpha_x \)

\[ C_{\text{sound}} = \land(C_{\text{original}}), \text{ if } \exists(P_{\text{original}} \land P_{\text{sound}}) \]
Complete Derivation

- Counter-examples on Complete-Derivation always exist for original constraints

<table>
<thead>
<tr>
<th>Constraints (unsatisfiable)</th>
</tr>
</thead>
<tbody>
<tr>
<td>$a \leq 0 \rightarrow S \subseteq \alpha \land \alpha \subseteq S$</td>
</tr>
<tr>
<td>$a \geq 0 \rightarrow \alpha \subseteq P$</td>
</tr>
</tbody>
</table>

Derivation 1: (complete):

$a \leq 0 \rightarrow S \subseteq \alpha$

- A complete derivation can be generated by:

$$C_{complete} = \wedge(C_{original}), \text{ if } \forall (P_{complete} \Rightarrow P_{original})$$
Equivalent Derivation

- Equivalent Derivation

**Sound / Complete Derivation:**

Constraints (satisfiable)

- true $\rightarrow P \subseteq \alpha_x \land \alpha_x \subseteq P$
- $d > 0 \rightarrow S \subseteq \alpha_y$
- $(d > 0) \rightarrow P \subseteq \alpha_y$
- $d < 0 \rightarrow \alpha_y \subseteq \alpha_x$

\[
d > 0 \rightarrow P \subseteq \alpha_x \land \alpha_z \subseteq \alpha_x \land \alpha_x \subseteq P \land S \subseteq \alpha_y
\]
\[
d = 0 \rightarrow P \subseteq \alpha_x \land \alpha_z \subseteq \alpha_x \land \alpha_x \subseteq P \land P \subseteq \alpha_y
\]
\[
d < 0 \rightarrow P \subseteq \alpha_x \land \alpha_z \subseteq \alpha_x \land \alpha_x \subseteq P \land P \subseteq \alpha_y \land \alpha_y \subseteq \alpha_x
\]

**Project**

\[C_{\text{sound}} = \land (C_{\text{original}}), \text{ if } \exists (P_{\text{original}} \land P_{\text{sound}})\]

**Inferred**

\[C_{\text{complete}} = \land (C_{\text{original}}), \text{ if } \forall (P_{\text{complete}} \Rightarrow P_{\text{original}})\]
Overview

Goal: to efficiently infer the dependent security labels

- Ternary label
- Predicate as Labels
- Permission Predicates
- Dynamic Labels

Derivation Framework

One-Shot
Early-Accept
Early-Reject
Hybrid

Inference Engines

Core Constraint Language

Dependent Security Labels

Evaluation
Inference Algorithms

- One-Shot:
  - Equivalent Derivation
  - Derived constraint number grows **exponentially** with the number of predicates

Arbitrary Constraints

- $P_1 \rightarrow C_1$
- $P_2 \rightarrow C_2$
- $P_3 \rightarrow C_3$

Refined

Refined
Inference Algorithms

- Early-Accept: Sound Derivation
- Early-Reject: Complete Derivation

Arbitrary Constraints

\[ P_1 \rightarrow C_1 \]
\[ P_2 \rightarrow C_2 \]
\[ P_3 \rightarrow C_3 \]
\[ \ldots \]
Inference Algorithms

- Hybrid

Arbitrary Constraints

\[ P_1 \rightarrow C_1 \]
\[ P_2 \rightarrow C_2 \]
\[ P_3 \rightarrow C_3 \]
...

\[
P_1 \rightarrow C_s^{D_2} \rightarrow \text{Sat}
\]
\[
P_2 \rightarrow C_s^{D_2} \rightarrow \text{UnSat}
\]
\[
P_2 \rightarrow C_s^{D_2} \rightarrow \text{UnSat}
\]

\[
P_1 \rightarrow C_s^{D_1} \rightarrow \text{Sat}
\]
Evaluations

- Benchmark
  - Verified MIPS Processor
    - 1719 lines of Verilog Code, 2455 variables totally
  - Mutate the annotations
    - 1509 variables unlabeled by randomly removing annotated labels
    - 14 errors are injected by modifying annotations
  - SecVerilog [Zhang et al. 2015] compiler to generate constraints
  - 100 test files:
    - 82 involves dependent labels
Evaluations

- Overall Performance: Hybrid > Early-Accept > One-Shot
- Time-out: Hybrid < Early-Accept < One-Shot
- Scalability: Hybrid > Early-Accept > One-Shot
Conclusions

- **Goal:** to efficiently infer the **dependent** security labels

- **Dependent Security Labels**
  - Ternary label
  - Predicate as Labels
  - Permission Predicates
  - Dynamic Labels

- **Core Constraint Language**

- **Inference Engines**
  - One-Shot
  - Early-Accept
  - Early-Reject
  - Hybrid

- **Derivation Framework**

- **Evaluation**
THANKS!

Any questions?