Taint Analysis in Simple Java Programs

ZHIMING MENG, University of San Francisco, USA
JIAHE TIAN, University of San Francisco, USA

Abstract placeholder. test test

ACM Reference Format:
Zhiming Meng and Jiahe Tian. 2024. Taint Analysis in Simple Java Programs. 1, 1 (May 2024), 2 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 INTRODUCTION

Introduction placeholder.

2 APPROACH

2.1 Intra-procedural Analysis

2.1.1 ForwardFlowAnalysis. Soot provides an abstract class `ForwardFlowAnalysis` that we subclass to implement our own taint analysis. We define a concrete implementation of the abstract method `flowThrough` which utilizes set implementing the `FlowSet` interface to compute a fixed-point in the dataflow through a worklist algorithm. The `flowThrough` method traverses the method, with every program point having its own `FlowSet`. Since we are working with Jimple, program points are of type `Stmt`, and they implement the `Unit` interface, needed to denote a unit of execution within the intermediate representation.

2.1.2 TaintStore. To keep be able to map variables to their respective taint sources for each program point, we create a `TaintStore` class that implements the `FlowSet` interface. By setting the generic parameter of the `FlowSet` interface to `Map.Entry<`K`, Set<`V`>>`, we are able to have an underlying $store : var \mapsto \{s \mid s \text{ is a taint source}\}$ mapping structure. A `LinkedTreeMap` is used to preserve the order that the individual statements are traversed. The set operations are implemented by considering each $(var, srcset)$ mapping as an individual element in the `FlowSet`. The key and value types are left generic for extensibility should we need to use different types to represent variables and taint sources.

Algorithm 1 Intra-procedural analysis of a method.

```
1: procedure FLOWTHROUGH(in, unit, out)
2:     in.copy(out)                          ▷ Sets out = in
3:     if unit instanceof criteria then      ▷ Handles different types of stmts
4:         UPDATETAINTS(lhs, rhs)            ▷ Propagate taint with specific method
5:     end if
6:     UPDATESINK()                          ▷ Add currently tainted sources to solution if sink
7: end procedure
```

Authors' Contact Information: Zhiming Meng, University of San Francisco, San Francisco, California, USA; Jiahe Tian, University of San Francisco, San Francisco, California, USA.

2024. ACM XXXX-XXXX/2024/5-ART
https://doi.org/10.1145/nnnnnnn.nnnnnnn

Table 1. Methods for interacting with taint store.

method	params	operation
addTaint	k, v	$store[k] = store[k] \cup \{v\}$
addTaints	$k, \{v_1, v_2, ...\}$	$store[k] = store[k] \cup \{v_1, v_2, ...\}$
propagateTaints	k_1, k_2	$store[k_2] = store[k_1] \cup store[k_2]$
setTaint	k, v	$store[k] = \{v\}$
setTaints	$k, \{v_1, v_2, ...\}$	$store[k] = \{v_1, v_2, ...\}$
isTainted	k	**return** *true* **if** $\|store[k]\| > 0$
getTaints	k	**return** $store[k]$

3 RESULTS

Results placeholder.

4 DISCUSSION

Discussion placeholder.

5 FUTURE WORK

Future work placeholder. [1]

REFERENCES

[1] Author Test. 2015. Test Title. *Test Journal* (Nov 2015).