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 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 key and value types are left generic for extensibility should we need to use different types to represent variables and taint sources.

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	$k, \{v_1, v_2, ...\}$	$store[k] = \{v_1, v_2, ...\}$
setTaints	k_1, k_2	$store[k_2] = store[k_1]$
isTainted	k	**return** *true* **if** $\lvert store[k] \rvert > 0$
getTaints	k	**return** $store[k]$

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

2.1.3 flowThrough. We override `flowThrough` to check if the *unit* parameter is an instance of certain interface and classes to determine the rules that will be used to propagate taints. In Algorithm 1, *in* is the incoming taint store from the analysis of the previous statement, *out* is the statement that will be modified in the analysis of the current function, and then passed to the analysis of the next statement. Methods used to manipulate the taint store are described in Table 1.

Algorithm 1 Intra-procedural analysis with flowThrough

```
1:  Map sinkToSourceMap : sink ↦ {src | src taints sink}
2:  procedure FLOWTHROUGH(in, unit, out)
3:      in.COPY(out)                                  ▷ Sets out = in as a baseline
4:      if unit instanceof JAssignmentStmt then       ▷ Handles assignment statements
5:          rightOp ← jAssignmentStmt.GETRIGHTOP()
6:          leftOp ← jAssignmentStmt.GETLEFTOP()
7:          if rightOp instanceof StaticFieldRef then ▷ Handles static fields
8:              out.SETTAINTS(rightOp, leftOp)
9:          end if
10:         if rightOp instanceof InvokeExpr then     ▷ Handles method invokes
11:             if invokeExpr instanceof InstanceInvokeExpr then
12:                 out.SETTAINTS(instanceInvokeExpr.GETBASE(), leftOp)
13:             end if
14:             if invokeExpr instanceof StaticInvokeExpr then
15:                 out.SETTAINTS(staticInvokeExpr, leftOp)
16:             end if
17:             if invokeExpr instanceof DynamicInvokeExpr then
18:                 out.SETTAINTS(dynamicInvokeExpr, leftOp)
19:             end if
20:             for arg in invokeExpr.GETARGS() do    ▷ Weak update arguments
21:                 out.PROPAGATETAINTS(arg, leftOp)
22:             end for
23:         end if
24:     end if
25:     UPDATESINK()                  ▷ Add currently tainted sources to solution if sink
26: end procedure
```

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).