Exercise 3: Information Flow and Slicing

—Solution—

Deadline for uploading solutions via Ilias:
January 29, 2021, 11:59pm Stuttgart time

Task 1  Information Flow Analysis  [20 points]

This task is about dynamic information flow analysis. Consider the following JavaScript code to analyze:

```javascript
var age = getAge();
var flag = getFlag();
var count = 0;
var mult = 2;
if (age === 20){
    count++;
} else {
    if (age === 25){
        flag = 2;
    } else {
        if (age === 30) {
            flag = 3;
        }
    }
}
if (flag === 2){
    flag = flag * mult;
    print_public(flag);
} else {
    print_public(mult);
}
```

There are three security classes: secret, confidential, and public, which are ordered into a lattice such that:

secret > confidential > public
By default, all values are labeled as *public*. Values returned by `getAge()` are labeled as *secret* and values returned by `getFlag()` are labeled as *confidential*. The function `print(public)()` is an untrusted sink, which should only be reached by *public* information. Note that passing an argument to function should be handled like an assignment to the formal parameter of the function.

### Subtask 1.1 Execution 1

Consider a dynamic information flow analysis that considers both explicit and implicit flows. Suppose an execution where `getAge()` returns 20 and `getFlag()` returns 3.

- What are the security labels of variables and expressions during the execution? Use the following template to provide your answer.

  **Solution:**

<table>
<thead>
<tr>
<th>Line</th>
<th>Variable or expression</th>
<th>Security label of variable or expression (after executing the line)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>age</td>
<td>secret</td>
</tr>
<tr>
<td>2</td>
<td>flag</td>
<td>confidential</td>
</tr>
<tr>
<td>3</td>
<td>count</td>
<td>public</td>
</tr>
<tr>
<td>4</td>
<td>mult</td>
<td>public</td>
</tr>
<tr>
<td>5</td>
<td>age === 20</td>
<td>secret</td>
</tr>
<tr>
<td>6</td>
<td>count</td>
<td>secret</td>
</tr>
<tr>
<td>17</td>
<td>flag === 2</td>
<td>confidential</td>
</tr>
</tbody>
</table>

- Does the execution violate the information flow policy? Explain your answer.

  Yes, because the execution reaches the line 21 due to a confidential *if* condition. For this reason, a confidential information, in this case that `flag != 2`, reaches an untrusted sink.

### Subtask 1.2 Execution 2

Consider a dynamic information flow analysis that considers both explicit and implicit flows. Suppose an execution where `getAge()` returns 25 and `getFlag()` returns 1.

- What are the security labels of variables and expressions during the execution? Use the following template to provide your answer.

  **Solution:**

<table>
<thead>
<tr>
<th>Line</th>
<th>Variable or expression</th>
<th>Security label of variable or expression (after executing the line)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>age</td>
<td>secret</td>
</tr>
<tr>
<td>2</td>
<td>flag</td>
<td>confidential</td>
</tr>
<tr>
<td>3</td>
<td>count</td>
<td>public</td>
</tr>
<tr>
<td>4</td>
<td>mult</td>
<td>public</td>
</tr>
<tr>
<td>5</td>
<td>age === 25</td>
<td>secret</td>
</tr>
<tr>
<td>6</td>
<td>count</td>
<td>secret</td>
</tr>
<tr>
<td>17</td>
<td>flag === 1</td>
<td>confidential</td>
</tr>
<tr>
<td>Line</td>
<td>Variable or expression</td>
<td>Security label of variable or expression (after executing the line)</td>
</tr>
<tr>
<td>------</td>
<td>------------------------</td>
<td>---------------------------------------------------------------</td>
</tr>
<tr>
<td>1</td>
<td>age</td>
<td>secret</td>
</tr>
<tr>
<td>2</td>
<td>flag</td>
<td>confidential</td>
</tr>
<tr>
<td>3</td>
<td>count</td>
<td>public</td>
</tr>
<tr>
<td>4</td>
<td>mult</td>
<td>public</td>
</tr>
<tr>
<td>5</td>
<td>age === 20</td>
<td>secret</td>
</tr>
<tr>
<td>8</td>
<td>age === 25</td>
<td>secret</td>
</tr>
<tr>
<td>9</td>
<td>flag</td>
<td>secret</td>
</tr>
<tr>
<td>17</td>
<td>flag === 2</td>
<td>secret</td>
</tr>
<tr>
<td>18</td>
<td>flag</td>
<td>secret</td>
</tr>
</tbody>
</table>

- Does the execution violate the information flow policy? Explain your answer.
  Yes, because in line 19 a value with label *secret* flows in an untrusted sink.
Task 2  Universally Bounded Lattice  

In this task we will analyze the characteristics and properties of universally bounded lattices.

Subtask 2.1  Recognize a Universally Bounded Lattice  

Figure 1 shows structures that could represent universally bounded lattices.

![Diagram of two universally bounded lattice candidates](image)

Figure 1: Two universally bounded lattice candidates.

- Are the two candidates universally bounded lattices? Explain why (not).

Solution:

- A is a universally bounded lattice. It has all the necessary characteristics:
  - A limited set of security classes
  - A partial order
  - A lower bound
  - An upper bound
  - A least upper bound operator
  - A greatest lower bound operator

- B is not a universally bounded lattice. Reason: It is impossible to define a least upper bound operator.
Subtask 2.2 Characteristics [10 points]

Consider the following structure that represents a universally bounded lattice:

Answer the following questions:

• Give the set $S$ of security classes.

  Solution:
  $S = ABCD, ABC, ABD, ACD, BCD, AB, BC, AD, BD, AC, CD, A, B, C, D, \emptyset$

• What is the lower bound $\perp$?

  Solution:
  $\perp = \emptyset$

• What is the upper bound $\top$?

  Solution:
  $\top = ABCD$

• Let $\bigoplus$ be the least upper bound operator. What is the result of the following operations?

  Solution:
  $B \bigoplus BC = BC$
  $ABCD \bigoplus CD = ABCD$
  $ABC \bigoplus \emptyset = ABC$
• Let \( \otimes \) be the greatest lower bound operator. What is the result of the following operations?

Solution:
\[
\begin{align*}
D \otimes DC &= D \\
ACD \otimes AD &= AD \\
ABCD \otimes B &= B
\end{align*}
\]
Consider the following JavaScript program:

```javascript
var year = getYear();
var flag = getFlag();
var count = 0;
var mult = 3;
if (year === 2020) {
    count++;
    mult = mult * count;
} else {
    if (year === 2021) {
        flag = 2 + flag;
    }
}
var sliceHere = count;
```

Compute the static backward slice for variable `sliceHere` at line 13. Use the slicing approach of Weiser (IEEE TSE, 1984) and its formulation as a graph reachability problem, as it has been introduced in the lecture. To describe your solution, follow the steps outlined below.

**Subtask 3.1 Data Flow Dependences**

Provide the data flow dependences between statements in the program. Use the following table to summarize the dependences. Each table cell represents a pair of statements. Mark all pairs of statements that have a definition-use relationship.

<table>
<thead>
<tr>
<th>Def</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>9</th>
<th>10</th>
<th>13</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td></td>
<td>x</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>x</td>
</tr>
<tr>
<td>2</td>
<td></td>
<td></td>
<td>x</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td></td>
<td></td>
<td></td>
<td>x</td>
<td></td>
<td></td>
<td>x</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>x</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>x</td>
<td></td>
<td>x</td>
<td></td>
<td></td>
</tr>
<tr>
<td>7</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>9</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>10</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>13</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

*Solution:*
Subtask 3.2 Control Flow Dependences [10 points]

Provide the control flow dependences between statements in the program. Describe your solutions as a sequence of “Statement .. is control-flow dependent on statement ..” sentences.

Solution:

- Statement 6 is control flow-dependent on statement 5.
- Statement 7 is control flow-dependent on statement 5.
- Statement 9 is control flow-dependent on statement 5.
- Statement 10 is control flow-dependent on statement 9.

Subtask 3.3 Program Dependence Graph [5 points]

Summarize the data flow dependences and the control flow dependences into a program dependence graph. Use the following template to draw your solution.

Solution:

```
1  10  9
   \  /   \
   2  3
   / \  /
  3  13 4
    v   \
    5
```

- - - - - - - destination is control-dependent on source

.. destination is data-dependent on source

Subtask 3.4 Slice [5 points]

What is the slice for the slicing criterion (variable sliceHere at line 13)? Write down the source code of the sliced program as a syntactically correct program.
Solution:

```javascript
var year = getYear();
var count = 0;
if (year === 2020) {
    count++;
}
var sliceHere = count;
```
Consider the following JavaScript program:

```javascript
var year = getInput();
var actual_year = 2021;
var count = 0;
var flag = false
var diff = actual_year - year
var i = 0;
while (i < diff) {
  i++;
  year++;
  if (year > 2000) {
    count++;
  }
}
if (count > 10) {
  flag = true
}
console.log(i)
```

**Subtask 4.1 Execution History**

Give the execution history with `getInput() = 2018`. You can use the line numbers to refer to statements. Do not include the lines with the closing curly brackets ( } ) into the history.

*Solution:* 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 7, 8, 9, 10, 11, 7, 8, 9, 10, 11, 7, 14, 17
Subtask 4.2  Dynamic Dependence Graph  

Suppose we want to compute the dynamic backward slice with the last statement (console.log(i)) as the slicing criterion. Use getInput() = 2019 (note that this input is different from above). Provide the dynamic dependence graph, using the “revised approach” presented in the lecture.

Solution:

Suppose we want to compute the dynamic backward slice with the last statement (console.log(i)) as the slicing criterion. Use getInput() = 2019 (note that this input is different from above). Provide the dynamic dependence graph, using the “revised approach” presented in the lecture.

Solution:
Subtask 4.3  Slice

Write the sliced program.

Solution:

```javascript
var year = getInput();
var actual_year = 2021;
var diff = actual_year - year
var i = 0;
while (i < diff) {
    i++;
}
console.log(i)
```