Foundations of Software Engineering

Taint Analysis
Miguel Velez
Learning goals

• Define taint analysis.
• Compare the dynamic and static approaches, as well as their benefits and limitations.
• Apply the analysis to several examples
• Understand how dynamic and static analyses can be combined to overcome the limitations of each other.
DYNAMIC ANALYSIS
Dynamic Analysis

• Learn about program’s properties by executing it.
• Examine program state throughout/after execution by gathering additional information.
Performance Analysis

How would you learn about method execution time?
1. void main(a) {
2.     if(a > 0) {
3.         sleep_ms(a);
4.     } else {
5.         sleep_ms(1000);
6.     }
7. }
1. void main(a) {
2.     start("main");
3.     if(a > 0) {
4.         sleep_ms(a);
5.     } else {
6.         sleep_ms(1000);
7.     }
8.     end("main");
9. }
Benefits
Benefits

• Analyzes the state of the program in a runtime environment.
• If the property we are looking for is found, we can be sure that it exists.
• Validate static analysis findings.
Limitations
Limitations

• Input dependent
• Cannot explore all paths
• Cost of tracking information
• Heisenbuggy behavior
Static Analysis

• Learn about program’s properties without executing it.
• Systematic examination of an abstraction of a program
Zero Analysis

How would you learn if you divide by 0?
1. x = 10;
2. y = x;
3. z = 0;
4. while(y > -1) {
5. x = x/y;
6. y = y-1;
7. Z = 5;
8. }

1. x = 10;
2. y = x;
3. z = 0;
4. while(y > -1) {
5.   x = x/y;
6.   y = y-1;
7.   Z = 5;
8. }

\[
\begin{array}{c}
Z \\
\downarrow \\
\downarrow \\
\downarrow \\
MZ & & & & NZ \\
\end{array}
\]
1. \( x = 10; \)
2. \( y = x; \)
3. \( z = 0; \)
4. while\( (y > -1) \) {
   5. \( x = x/y; \)
   6. \( y = y-1; \)
   7. \( Z = 5; \)
   8. }

(MZ)
1. x = 10;
2. y = x;
3. z = 0;
4. while(y > -1) {
   5. x = x/y;
   6. y = y-1;
   7. z = 5;
   8. }

Join

y > -1

x = x/y

y = y-1;

z = 5;

(exit)
1. \( x = 10; \)
2. \( y = x; \)
3. \( z = 0; \)
4. while(\( y > -1 \)) {
   5.    \( x = x/y; \)
   6.    \( y = y-1; \)
   7.    \( z = 5; \)
   8. }

Join

\( y > -1 \)

Fixpoint

\( x = x/y \)

\( y = y-1; \)

\( z = 5; \)

(exit)
1. x = 10;
2. y = x;
3. z = 0;
4. while(y > -1) {
5.   x = x/y;
6.   y = y-1;
7.   z = 5;
8. }

\[
\text{Join}
\]
\[
y > -1
\]
\[
x = x/y
\]
\[
y = y-1;
\]
\[
z = 5;
\]
\[
\text{Fixpoint}
\]
\[
\text{(exit)}
\]
Benefits
Benefits

• Analyzes all possible executions of the program.
• Pinpoint in code where issues occur.
• Detects issues in the early stages of development.
Limitations
Limitations

• Rice’s Theorem: Every static analysis is necessarily incomplete or unsound or undecidable (or multiple of these).
• Difficult to track runtime properties.
• Can analyze parts of the program that are never executed.
TAINT ANALYSIS
Taint Analysis

- Information flow analysis.
- Used in the security domain.
- Tracking how private information flows through the program and if it is leaked to public observers.
Example

1. input = get_input();
2. tmp = “select …” + input;
3. query(tmp);
4. log(tmp);
Example

1. input = get_input();
2. tmp = “select …” + input;
3. query(tmp);
4. log(tmp);

Warning!

input

priv
tmp

Warning!
Terminology

• Sources
  • Private data of interest

• Sinks
  • Locations of interest
  • Check taints of incoming information
  • Determines if there is a leak in the program.
Example

1. input = get_input();
2. tmp = “select …” + input;
3. query(tmp);
4. log(tmp);
Example

1. input = Source();
2. tmp = “select …” + input;
3. Sink(tmp);
4. log(tmp);
Example

1. input = Source();
2. tmp = “select …” + input;
3. Sink(tmp);
4. log(tmp);

Warning!
Example

1. `input = Source();`
2. `tmp = “select ...” + input;`
3. `tmp = encode(tmp)`
4. `Sink(tmp);`
5. `log(tmp);`
DYNAMIC TAINT ANALYSIS
Dynamic Taint Analysis

• Track what are the taints that are influencing the values of the program.
Example

1. \( x = \text{get\_input}(); \)
2. \( y = 1; \)
3. \( z = x; \)
4. \( w = y + z; \)
5. \( \text{print}(w); \)
Example

1. \( x = \text{Source}(0); \)
2. \( y = 1; \)
3. \( z = x; \)
4. \( w = y + z; \)
5. \( \text{Sink}(w); \)
Example

1. \( x = \text{Source}(0); \)
2. \( y = 1; \)
3. \( z = x; \)
4. \( w = y + z; \)
5. \( \text{Sink}(w); \)
Example

1. $x = \text{Source}(0);$  
2. $y = 1;$  
3. $z = x;$  
4. $w = y + z;$  
5. Sink($w$);
Example

1. \( x = \text{Source}(0); \)
2. \( y = 1; \)
3. \( z = x; \)
4. \( w = y + z; \)
5. Sink(w);
Example

1. \( x = \text{Source}(0); \)
2. \( y = 1; \)
3. \( z = x; \)
4. \( w = y + z; \)
5. Sink(w);
Example

1. \( x = \text{Source}(0); \)
2. \( y = 1; \)
3. \( z = x; \)
4. \( w = y + z; \)
5. \( \text{Sink}(w); \)
Example

1. \( x = \text{Source}(0); \)
2. \( y = 1; \)
3. \( z = x; \)
4. \( w = y + z; \)
5. \( \text{Sink}(w); \)

Leak in the program!
Is there a leak? Why? Why not?

1. \( x = \text{Source}(0) \);
2. \( y = x \);
3. \( \text{if}(y == 0) \) {
4. \( z = 2 \)
5. }
6. \( \text{else } \) {
7. \( z = 1 \)
8. }
9. \( \text{Sink}(z) \);
Implicit Flows

• Tainted data affects the value of another variable indirectly.
• Needed for sound analysis.
Implicit Flows

1. \( x = \text{Source}(0); \)
2. \( y = x; \)
3. \( \text{if}(y == 0) \) {
4. \( z = 2 \)
5. } \quad \text{Explicit information flow}
6. \( \text{else} \) {
7. \( z = 1 \)
8. } \quad \text{Implicit information flow}
9. \( \text{Sink}(z); \)
Implicit Flows

1. $x = \text{Source}(0)$;
2. $y = x$;
3. if($y == 0$) {
   4.     $z = 2$
   5. }
4. else {
   5.     $z = 1$
   6. }
7. Sink($z$);
Implicit Flows

1. \( x = \text{Source}(0); \)
2. \( y = x; \)
3. if \( y == 0 \) {
4.    \( z = 2 \)
5. }
6. else {
7.    \( z = 1 \)
8. }
9. Sink(z);
Implicit Flows

1. \( x = \text{Source}(\theta); \)
2. \( y = x; \)
3. if\( (y == 0) \) {
   4. \( z = 2 \)
5. }
6. else {
   7. \( z = 1 \)
8. }
9. Sink(z);

Leak in the program!
Try it yourself

1. \( x = \text{Source}(1); \)
2. \( y = 0; \)
3. while(\( x > 0 \)) {
4. \hspace{1em} y = y + 1;
5. \hspace{1em} x = x - 1;
6. }
7. \( z = y; \)
8. Sink(y);
9. Sink(z);
Try it yourself

1. \( x = \text{Source}(1); \)
2. \( y = 0; \)
3. \( \text{while}(x > 0) \) 
   
4. \( \quad y = y + 1; \)
5. \( \quad x = x - 1; \)
6. \( \quad } \)
7. \( z = y; \)
8. \( \text{Sink}(y); \)
9. \( \text{Sink}(z); \)

Leaks in the program!
Limits of Dynamic Analysis

• Results are input dependent.
• Implicit flows needed for sound analysis, but difficult to track*.

• *Stayed tuned for the end of lecture.
STATIC TAINT ANALYSIS
Static Taint Analysis

• Track, at each instruction, what are the taints that are influencing the variables of the program.
Example

1. \( x = \text{Source}(i); \)
2. \( y = 1; \)
3. \( z = x; \)
4. \( w = y + z; \)
5. \( \text{Sink}(w); \)
Example

1. $x = \text{Source}(i)$;  \hspace{1cm} x \rightarrow T
2. $y = 1$;  \hspace{1cm} x \rightarrow T
3. $z = x$;  \hspace{1cm} x \rightarrow T, z \rightarrow T
4. $w = y + z$;  \hspace{1cm} x \rightarrow T, z \rightarrow T, w \rightarrow T
5. Sink($w$);  \hspace{1cm} x \rightarrow T, z \rightarrow T, w \rightarrow T
Example

1. \( x = \text{Source}(i); \)
2. \( y = 1; \)
3. \( z = x; \)
4. \( w = y + z; \)
5. \( \text{Sink}(w); \)

Leak in the program!
Implicit Flows

1. $x = \text{Source}(i)$;
2. $y = x$;
3. if($y == 0$) {
4. $z = 0$
5. }
6. else {
7. $z = 1$
8. }
9. Sink($z$);
1. \( x = \text{Source}(i, \text{“A”}); \)
2. \( y = x; \)
3. \( \text{if}(y == 0) \) {
4. \( \quad z = 0 \)
5. \( \}) \)
6. \( \text{else} \) {
7. \( \quad z = 1 \)
8. \( \}) \)
9. \( \text{Sink}(z); \)
1. $x = \text{Source}(i);$
2. $y = x;$
3. if($y == 0$) {
4.     $z = 0$
5. }
6. else {
7.     $z = 1$
8. }
9. $\text{Sink}(z);$
Kildall’s Worklist Algorithm

for Instruction i in program
    input[i] = ⊥
input[firstInstruction] = initialDataflowInformation
worklist = { firstInstruction }

while worklist is not empty
    take an instruction i off the worklist
    output = flow(i, input[i])
    for Instruction j in succs(i)
        if output ⊑ input[j]
            input[j] = input[j] □ output
            add j to worklist
1: \( x = \text{Source}(i); \)

2: \( y = x; \)

3: \( y == 0 \)

4: \( z = 0 \)

7: \( z = 1 \)

9: \( \text{Sink}(z); \)

Input

<table>
<thead>
<tr>
<th>Stmt</th>
<th>x</th>
<th>y</th>
<th>z</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>7</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>9</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
1:  x = Source(i);

2:  y = x;

3:  y == 0

4:  z = 0

7:  z = 1

9:  Sink(z);

Input
Stmt | x  | y  | z  |
-----|----|----|----|
1    | NT | NT | NT |
2    | T  | NT | NT |
3    | T  | T  | NT |
4    | T  | T  | NT |
7    | T  | T  | NT |
9    | T  | T  | T  |

Stmt | Worklist | x  | y  | z  |
-----|----------|----|----|----|
1    | 2        | T  | NT | NT |
2    | 3        | T  | T  | NT |
3    | 4,7      | T  | T  | NT |
4    | 7,9      | T  | T  | T  |
7    | 9        | T  | T  | T  |
9    |          | T  | T  | T  |
1: x = Source(i);
2: y = x;
3: y == 0
4: z = 0

Stmt | Worklist | x | y | z
--- | --- | --- | --- | ---
1 | 2 | T | NT | NT
2 | 3 | T | T | NT
3 | 4,7 | T | T | NT
4 | 7,9 | T | T | T
7 | 9 | T | T | T
9 | T | T | T

Leak in the program!
1: x = Source(i);

2: y = x;

3: y == 0

4: z = 0

7: z = 1

9: Sink(z);

Leak in the program!
Try it yourself

1. \( x = \text{Source}(i); \)
2. \( y = x; \)
3. \( \text{if}(y == 0) \{ \)
4. \( \quad z = 0 \)
5. \( \} \)
6. \( \text{else} \{ \)
7. \( \quad z = 1 \)
8. \( \} \)
9. \( \text{Sink}(z); \)
Possible leak in the program!

```plaintext
1: x = Source(i);

2: y = 0;

3: x > 0

4: y = y + 1

5: x = x - 1

7: z = y;

8: Sink(y);

9: Sink(z);

(exit)

1: x = Source(i);

2: y = 0;

3: x > 0

4: y = y + 1

5: x = x - 1

7: z = y;

8: Sink(y);

9: Sink(z);

Possible leak in the program!

### Input

<table>
<thead>
<tr>
<th>Stmt</th>
<th>x</th>
<th>y</th>
<th>z</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>NT</td>
<td>NT</td>
<td>NT</td>
</tr>
<tr>
<td>2</td>
<td>T</td>
<td>NT</td>
<td>NT</td>
</tr>
<tr>
<td>3</td>
<td>T</td>
<td>MT</td>
<td>NT</td>
</tr>
<tr>
<td>4</td>
<td>T</td>
<td>MT</td>
<td>NT</td>
</tr>
<tr>
<td>5</td>
<td>T</td>
<td>MT</td>
<td>NT</td>
</tr>
<tr>
<td>7</td>
<td>T</td>
<td>MT</td>
<td>NT</td>
</tr>
<tr>
<td>8</td>
<td>T</td>
<td>MT</td>
<td>MT</td>
</tr>
<tr>
<td>9</td>
<td>T</td>
<td>MT</td>
<td>MT</td>
</tr>
</tbody>
</table>

### Worklist

<table>
<thead>
<tr>
<th>Stmt</th>
<th>Worklist</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>3</td>
<td>4, 7</td>
</tr>
<tr>
<td>4</td>
<td>5, 7</td>
</tr>
<tr>
<td>5</td>
<td>3, 7</td>
</tr>
<tr>
<td>3</td>
<td>4, 7</td>
</tr>
<tr>
<td>4</td>
<td>5, 7</td>
</tr>
<tr>
<td>5</td>
<td>7</td>
</tr>
<tr>
<td>7</td>
<td>8</td>
</tr>
<tr>
<td>8</td>
<td>9</td>
</tr>
<tr>
<td>9</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Stmt</th>
<th>Worklist</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>3</td>
<td>4, 7</td>
</tr>
<tr>
<td>4</td>
<td>5, 7</td>
</tr>
<tr>
<td>5</td>
<td>3, 7</td>
</tr>
<tr>
<td>3</td>
<td>4, 7</td>
</tr>
<tr>
<td>4</td>
<td>5, 7</td>
</tr>
<tr>
<td>5</td>
<td>7</td>
</tr>
<tr>
<td>7</td>
<td>8</td>
</tr>
<tr>
<td>8</td>
<td>9</td>
</tr>
<tr>
<td>9</td>
<td></td>
</tr>
</tbody>
</table>

Possible leak in the program!
Limits of Static Analysis

- Do not know what values might cause the leak.
- Overtainting
Overtainting anti-patterns

1. x = Source(args[0]);
2. Object o = foo();
3. v = o.equals(x);
Overtainting anti-patterns

1. x = Source(args[0]);
2. Object o = foo();
3. v = o.equals(x);

All implementation of equals analyzed!
1. x = Source(args[0]);
2. if(Math.max(1, x) == 0) {
3.    Sink(x);
4. }
Overtainting anti-patterns

1. \( x = \text{Source(args[0])}; \)
2. \( \text{if(Math.max(1, x) == 0) } \{ \)
3. \( \text{Sink(x); x} \rightarrow \top \)
4. \}
Overtainting anti-patterns

1. \( i = \text{foo}(); \)
2. \( j = i + 1; \)
3. \( a[i] = \text{Source}(); \)
4. \( a[j] = 0; \)
5. \( \text{Sink}(a); \)
6. \( \text{Sink}(a[i]); \)
7. \( \text{Sink}(a[j]); \)
Overtainting anti-patterns

1. \( i = \text{foo}(); \)
2. \( j = i + 1; \)
3. \( a[i] = \text{Source}(); \)
4. \( a[j] = 0; \)
5. \( \text{Sink}(a); \)
6. \( \text{Sink}(a[i]); \)
7. \( \text{Sink}(a[j]); \)

Taints the whole array

- \( a \rightarrow T \)
- \( a[i] \rightarrow T \)
- \( a[j] \rightarrow T \)
COMBINING DYNAMIC AND STATIC ANALYSIS
Implicit Flows in Dynamic Analysis

1. \( x = \text{Source}(0); \)
2. \( y = x; \)
3. \( \text{if}(y == 0) \{ \)
4. \( \quad z = 2 \)
5. \( \} \)
6. \( \text{else} \{ \)
7. \( \quad z = 1 \)
8. \( \} \)
9. \( \text{Sink}(z); \)

Leak in the program!
Implicit Flows in Dynamic Analysis

1. \( x = \text{Source}(3); \)
2. \( y = x; \)
3. \( \text{if}(y == 0) \) { 
4. \( z = 2 \)
5. } 
6. \( \text{else} \) { 
7. \( z = 1 \)
8. } 
9. \( \text{Sink}(z); \)

Leak in the program!
Is there a leak? Why? Why not?

1. $x = \text{Source}(3);$  
2. $y = x;$  
3. $z = 1;$  
4. if($y == 0$) {  
5. \hspace{1em} $z = 2$  
6. }  
7. $\text{Sink}(z);$
Is there a leak? Why? Why not?

1. \( x = \text{Source}(3); \)
2. \( y = x; \)
3. \( z = 1; \)
4. \( \text{if}(y == 0) \{ \)
5. \( \quad z = 2 \)
6. \( \} \)
7. \( \text{Sink}(z); \)

No leak in the program!
Different result for Semantically the same Program?

1. \( x = \text{Source}(3); \)
2. \( y = x; \)
3. \( \text{if}(y == 0) \{ \)
   4. \( z = 3 \)
5. \} \)
6. \( \text{else} \{ \)
   7. \( z = 1 \)
8. \} \)
9. \( \text{Sink}(z); \)

Leak!

1. \( x = \text{Source}(3); \)
2. \( y = x; \)
3. \( z = 1; \)
4. \( \text{if}(y == 0) \{ \)
   5. \( z = 2 \)
5. \} \)
6. \( \text{Sink}(z); \)

No Leak!
Fundamental Issue

- In dynamic taint analysis, some implicit flows are hard to track
- If the code is not executed, we do not track its information.
How would you solve this issue?

1. \[ x = \text{Source}(3); \]
2. \[ y = x; \]
3. \[ \text{if}(y == 0) \{ \]
   \[ z = 2 \]
4. \[ \} \]
5. \[ \text{else} \{ \]
6. \[ z = 1 \]
7. \[ \} \]
8. \[ \text{Sink}(z); \]

Leak!

1. \[ x = \text{Source}(3); \]
2. \[ y = x; \]
3. \[ z = 1; \]
4. \[ \text{if}(y == 0) \{ \]
   \[ z = 2 \]
5. \[ \} \]
6. \[ \} \]
7. \[ \text{Sink}(z); \]

No Leak!
Branch-not-taken Analysis

1. \( x = \text{Source}(i); \)
2. \( y = x; \)
3. \( z = 1; \)

5. \( \text{if}(y == 0) \) {
6. \( z = 2 \)
7. \} 
8. \( \text{Sink}(z); \)
1. \( x = \text{Source}(3); \)
2. \( y = x; \)
3. \( z = 1; \)
5. \( \text{if}(y == 0) \) {
6.   \( z = 2 \)
7. } 
8. \( \text{Sink}(z); \)
Branch-not-taken Analysis

1. \( x = \text{Source}(3); \)
2. \( y = x; \)
3. \( z = 1; \)
5. \( \text{if}(y == 0) \{ \)
6. \( \quad z = 2 \)
7. \( \} \)
8. \( \text{Sink}(z); \)
Branch-not-taken Analysis

1. \( x = \text{Source}(3); \)
2. \( y = x; \)
3. \( z = 1; \)
4. \( 3 \rightarrow T \)
5. \( 1 \rightarrow NT \)
6. \( 1 \rightarrow T \)

5. \( \text{if}(y == 0) \) {
6. \( z = 2 \)
7. \} \)
8. \( \text{Sink}(z); \)

Leak in the program!
Is there a leak? Why? Why not?

1. \( x = \text{Source}(3); \)
2. \( y = x; \)
3. \( z = 1; \)
4. \( w = 1; \)
5. \( \text{if}(y == 0) \) {
  6. \( z = 2 \)
  7. \( \text{if}(x == 0) \) {
    8. \( w = 0; \)
    9. \} \)
  10. \} \)
11. \( \text{Sink}(w); \)
Limits of Branch-not-taken Analysis

1. \( x = \text{Source}(3); \)
2. \( y = x; \)
3. \( z = 1; \)
4. \( w = 1; \)
5. \( z \)
6. \( \text{if}(y == 0) \{ \)
7. \( z = 2 \)
8. \( \text{w} \)
9. \( \text{if}(x == 0) \{ \)
10. \( w = 0; \)
11. \} \)
12. \} \)
13. \( \text{Sink}(w); \)
INTERPROCEDURAL ANALYSIS
Interprocedural Analysis

1. main() {
2.   x = Source(1);
3.   y = 1;
4.   z = foo(x);
5.   Sink(z);
6.   z = foo(y);
7.   Sink(z);
8. }

1. foo(x) {
2.   y = x * 2;
3.   return x;
4. }
Interprocedural Analysis

1. main() {
   1.   foo(x) {
      2.   y = x * 2;
      3.   return x;
   4. }
  2.   x = Source(1);
  3.   y = 1;
  4.   z = foo(x);
  5.   Sink(z);
  6.   z = foo(y);
  7.   Sink(z);
  8. }

Information with context T
Interprocedural Analysis

1. main() {
2.   x = Source(1);
3.   y = 1;
4.   z = foo(x);
5.   Sink(z);
6.   z = foo(y);
7.   Sink(z);
8. }

1. foo(x) {
2.   y = x * 2;
3.   return x;
4. }

Information with context T

Information with context NT
Summary

• Taint analysis is an information flow analysis to detect if private data is leaked in the program.
• Compare benefits and limitations of dynamic and static approaches.
• Can be combined to overcome the limitations of the other.