An Empirical Study of Information Flows in Real-World JavaScript

Cristian-Alexandru Staicu\textsuperscript{1}  Daniel Schoepe\textsuperscript{2}
Musard Balliu\textsuperscript{3}  Michael Pradel\textsuperscript{4}  Andrei Sabelfeld\textsuperscript{2}

\textsuperscript{1}TU Darmstadt
\textsuperscript{2}Chalmers University of Technology
\textsuperscript{3}KTH Royal Institute of Technology
\textsuperscript{4}University of Stuttgart

15\textsuperscript{th} of November 2019
// variable passwd is sensitive
var paddedPasswd = "xx" + passwd;
var gotIt = false;
var knownPasswd = null;
if (paddedPasswd === "xxacmccs2019") {
    gotIt = true;
    knownPasswd = passwd;
}
// function sink is insensitive
sink(gotIt);
// variable passwd is sensitive
var paddedPasswd = "xx" + passwd;
var gotIt = false;
var knownPasswd = null;
if (paddedPasswd === "xxacmccs2019") {
    gotIt = true;
    knownPasswd = passwd;
}
// function sink is insensitive
sink(gotIt);

// information flow?
Program Analyses of Different Complexity

<table>
<thead>
<tr>
<th>lightweight → heavyweight</th>
</tr>
</thead>
<tbody>
<tr>
<td>taint tracking</td>
</tr>
</tbody>
</table>

```
// variable passwd is sensitive
var paddedPasswd = "xx" + passwd;
var gotIt = false;
var knownPasswd = null;
if (paddedPasswd === "xxacmccs2019") {
  gotIt = true;
  knownPasswd = passwd;
}
// function sink is insensitive
sink(gotIt);
```
Program Analyses of Different Complexity (2)

<table>
<thead>
<tr>
<th>lightweight → heavyweight</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>taint tracking</strong></td>
</tr>
</tbody>
</table>

```javascript
var paddedPasswd = "xx" + passwd;
var gotIt = false;
var knownPasswd = null;
if (paddedPasswd === "xxacmccs2019") {
    gotIt = true;
    knownPasswd = passwd;
}
sink(gotIt);
```

- only track explicit flows
- `passwd = "acmccs2019"`
Program Analyses of Different Complexity (2)

<table>
<thead>
<tr>
<th>lightweight → heavyweight</th>
</tr>
</thead>
<tbody>
<tr>
<td>taint tracking</td>
</tr>
</tbody>
</table>

```javascript
var paddedPasswd = "xx" + passwd;
var gotIt = false;
var knownPasswd = null;
if (paddedPasswd === "xxacmccs2019") {
    gotIt = true;
    knownPasswd = passwd;
}
sink(gotIt);
```

- only track explicit flows
- passwd = "acmccs2019"
Program Analyses of Different Complexity (2)

<table>
<thead>
<tr>
<th>lightweight → heavyweight</th>
</tr>
</thead>
<tbody>
<tr>
<td>taint tracking</td>
</tr>
</tbody>
</table>

```javascript
var paddedPasswd = "xx" + passwd;
var gotIt = false;
var knownPasswd = null;
if (paddedPasswd === "xxacmccs2019") {
  gotIt = true;
  knownPasswd = passwd;
}
sink(gotIt);
```

- only track explicit flows
- passwd = "acmccs2019"
Program Analyses of Different Complexity (2)

<table>
<thead>
<tr>
<th>lightweight</th>
<th>heavyweight</th>
</tr>
</thead>
<tbody>
<tr>
<td>taint tracking</td>
<td>observable tracking</td>
</tr>
</tbody>
</table>

```javascript
var paddedPasswd = "xx" + passwd;
var gotIt = false;
var knownPasswd = null;
if (paddedPasswd === "xxacmccs2019") {
    gotIt = true;
    knownPasswd = passwd;
}
sink(gotIt);
```

- only track explicit flows
- `passwd = "acmccs2019"
- no security violation
Program Analyses of Different Complexity (3)

<table>
<thead>
<tr>
<th>lightweight → heavyweight</th>
</tr>
</thead>
<tbody>
<tr>
<td>taint tracking</td>
</tr>
</tbody>
</table>

```javascript
var paddedPasswd = "xx" + passwd;
var gotIt = false;
var knownPasswd = null;
if (paddedPasswd === "xxacmccs2019") {
  gotIt = true;
  knownPasswd = passwd;
}
sink(gotIt);
```
Program Analyses of Different Complexity (3)

<table>
<thead>
<tr>
<th>lightweight → heavyweight</th>
</tr>
</thead>
<tbody>
<tr>
<td>taint tracking</td>
</tr>
</tbody>
</table>

```javascript
var paddedPasswd = "xx" + passwd;
var gotIt = false;
var knownPasswd = null;
if (paddedPasswd === "xxacmccs2019") {
  gotIt = true;
  knownPasswd = passwd;
}
sink(gotIt);
```

- track some implicit flows
- passwd = "acmccs2019"
Program Analyses of Different Complexity (3)

<table>
<thead>
<tr>
<th>lightweight → heavyweight</th>
</tr>
</thead>
<tbody>
<tr>
<td>taint tracking</td>
</tr>
</tbody>
</table>

```
var paddedPasswd = "xx" + passwd;
var gotIt = false;
var knownPasswd = null;
if (paddedPasswd === "xxacmccs2019") {
    gotIt = true;
    knownPasswd = passwd;
}
sink(gotIt);
```

- track some implicit flows
- passwd = "acmccs2019"
- security violation
Program Analyses of Different Complexity (3)

<table>
<thead>
<tr>
<th>lightweight → heavyweight</th>
</tr>
</thead>
<tbody>
<tr>
<td>taint tracking</td>
</tr>
</tbody>
</table>

```javascript
var paddedPasswd = "xx" + passwd;
var gotIt = false;
var knownPasswd = null;
if (paddedPasswd === "xxacmccs2019") {
  gotIt = true;
  knownPasswd = passwd;
}
sink(gotIt);
```

- track some implicit flows
- `passwd = "foo"
- no security violation
Program Analyses of Different Complexity (4)

| lightweight → heavyweight | taint tracking | observable tracking | information flow control |

```javascript
var paddedPasswd = "xx" + passwd;
var gotIt = false;
var knownPasswd = null;
if (paddedPasswd === "xxacmccs2019") {
  gotIt = true; // violation
  knownPasswd = passwd;
}
sink(gotIt);
```

- `passwd = "acmccs2019"`
- insert upgrade statement on monitor violation
Program Analyses of Different Complexity (4)

<table>
<thead>
<tr>
<th>lightweight → heavyweight</th>
</tr>
</thead>
<tbody>
<tr>
<td>taint tracking</td>
</tr>
</tbody>
</table>

```javascript
var paddedPasswd = "xx" + passwd;
var gotIt = false;
var knownPasswd = null;
upgrade(paddedPasswd, gotIt);
if (paddedPasswd === "xxacmccs2019") {
  gotIt = true;
  knownPasswd = passwd;
}
sink(gotIt);
```

- passwd = "foo"
- insert upgrade statement on monitor violation
### Program Analyses of Different Complexity (4)

<table>
<thead>
<tr>
<th>lightweight → heavyweight</th>
</tr>
</thead>
<tbody>
<tr>
<td>taint tracking</td>
</tr>
</tbody>
</table>

```javascript
var paddedPasswd = "xx" + passwd;
var gotIt = false;
var knownPasswd = null;
upgrade(paddedPasswd, gotIt);
if (paddedPasswd === "xxacmccs2019") {
    gotIt = true;
    knownPasswd = passwd;
}
sink(gotIt);
```

- `password = "foo"`
- insert upgrade statement on monitor violation
Cost-Benefit Analysis

**Benefits:**
- more fine-grained flows
- more detected vulnerabilities

**Costs:**
- runtime cost
- label creep
- permissiveness
Overview

Formal semantics → Dynamic analysis → Metrics

Research questions → Benchmarks

Theoretical analysis → Empirical study

Results
Overview

- Formal semantics
- Dynamic analysis
- Benchmarks
- Metrics
- Research questions
- Theoretical analysis
- Empirical study
- Results
Microflows

Definition

A microflow is an operation, e.g., write, that causes a variable to become sensitive.

Explicit flows

var a = b; // explicit flow from a to b

Observable implicit flow

if (a === true) {
    b = 23; // observable implicit flow from a to b
} else {
    b = 42; // observable implicit flow from a to b
}

Hidden implicit flow

if (a === true) {
    b = 23; // observable implicit flow from a to b
} else {
    // hidden implicit flow from a to b
}
Source-to-Sink Flows

Definition

A sequence of cascaded microflows between a source and a sink is called a source-to-sink (S2S) flow.

```javascript
var paddedPasswd = "xx" + passwd;
var gotIt = false;
var knownPasswd = null;
if (paddedPasswd === "xxacmccs2019") {
  gotIt = true;
  knownPasswd = passwd;
}
sink(gotIt);
```

One source-to-sink flow consisting of one explicit microflow and one observable implicit.
Definition

Label creep ratio (LCR) is the percentage of variables assigned a sensitive value in their lifetime out of the total number of variables ever assigned.

\[ LCR = \frac{\text{\# sensitive variables/fields ever assigned}}{\text{\# variables/fields ever assigned}} \]

```javascript
var paddedPasswd = "xx" + passwd;
var gotIt = false;
var knownPasswd = null;

LCR = 0.33
```
Label Creep

**Definition**

Label creep ratio (LCR) is the percentage of variables assigned a sensitive value in their lifetime out of the total number of variables ever assigned.

\[
LCR = \frac{\# \text{ sensitive variables/fields ever assigned}}{\# \text{ variables/fields ever assigned}}
\]

```
var paddedPasswd = "xx" + passwd;
var gotIt = false;
var knownPasswd = null;
if (paddedPasswd === "xxacmccs2019") {
    gotIt = true;
}
```

\[ LCR = 0.33 \]

```
var paddedPasswd = "xx" + passwd;
var gotIt = false;
var knownPasswd = null;
if (paddedPasswd === "xxacmccs2019") {
    gotIt = true;
}
```

\[ LCR = 0.66 \]
Inference of Upgrade Statements

**Definition**

*Sensitive branch coverage* (SBC) is the percentage of conditionals handling sensitive information that are covered on both branches.

\[
SBC = \frac{|\{ c \in C \text{ where both true and false branch covered}\}|}{|C|}
\]
Sensitive branch coverage (SBC) is the percentage of conditionals handling sensitive information that are covered on both branches.

\[
SBC = \frac{|\{c \in C \text{ where both true and false branch covered}\}|}{|C|}
\]

```javascript
var paddedPasswd = "xx" + passwd;
var gotIt = false;
if (gotIt === false) {
  console.log(23);
}
if (paddedPasswd === "xxacmccs2019") {
  gotIt = true;
}
```

**Input:** foo, acmccs2019  
**Branch coverage:** 75%, **SBC:** 100%
Implementation

- dynamic analysis based on **Jalangi**
- **expressive policy** language: function parameters, callback arguments
- **model** for native functions: arguments → return value
- **model** commonly used functions, e.g., Array.push, Object.call
- more than **100 unit tests** to test labels propagation
Benchmarks

- different security problems:
  - **integrity**: 19 injection vulnerabilities
    [Staicu et al., NDSS, 2018]
  - **availability**: 20 ReDoS vulnerabilities
    [Staicu et al.,USENIX Security, 2018]
  - **confidentiality**: 7 buffer vulnerabilities
  - **confidentiality**: 10 client-side leaks

- inputs for triggering the attack and for increasing coverage

- realistic security policies that capture at least one source-to-sink flow per benchmark

- 50,547 LOC, 65 upgrades, 0.68 average SBC
Benefit: Source-to-Sink Flows

![Bar chart showing the number of source-to-sink flows for different benchmarks and techniques.]

- **Benchmark**:
  - Injections
  - ReDoS
  - Buffer
  - Fingerprinting

- **Techniques**:
  - Taint Tracking
  - Observable Tracking
  - Permissive Upgrade

- **Axes**:
  - Y-axis: Number of source-to-sink flows
  - X-axis: Benchmark

- **Legend**:
  - Taint Tracking
  - Observable Tracking
  - Permissive Upgrade
  - Fingerprinting
Cost: Permissiveness

Number of code locations where the monitor would terminate the program

Benchmark

NSU  PU  

0  10  20  30  40  50  60  70  80
Cost: LCR - Label Creep Ratio

The diagram shows the relationship between the label creep ratio and the percentage of execution. The x-axis represents the percentage of execution, ranging from 0 to 100, while the y-axis represents the label creep ratio, ranging from 0 to 1. The lines and markers vary for different execution percentages, indicating the trend of label creep over execution. The label creep ratio is calculated as the average maximum label creep (LCR) and is shown to be approximately 20%.
Cost: LCR - Label Creep Ratio

Average $\max(LCR) = 20\%$
Cost: LCR - Label Creep Ratio (2)
Cost: Runtime Overhead

Measure taint relevant operations, e.g., binary operators, method calls, conditionals.

<table>
<thead>
<tr>
<th></th>
<th>Taint Tracking</th>
<th>Observable Tracking</th>
<th>Information Flow Control</th>
</tr>
</thead>
<tbody>
<tr>
<td>Command injection</td>
<td>59,339</td>
<td>59,383</td>
<td>59,540</td>
</tr>
<tr>
<td>ReDoS vulnerabilities</td>
<td>210</td>
<td>540</td>
<td>633</td>
</tr>
<tr>
<td>Buffer vulnerabilities</td>
<td>5,740</td>
<td>6,007</td>
<td>6,084</td>
</tr>
<tr>
<td>Client-side programs</td>
<td>5,919</td>
<td>19,555</td>
<td>20,890</td>
</tr>
</tbody>
</table>
Measure taint relevant operations, e.g., binary operators, method calls, conditionals.

<table>
<thead>
<tr>
<th></th>
<th>Taint Tracking</th>
<th>Observable Tracking</th>
<th>Information Flow Control</th>
</tr>
</thead>
<tbody>
<tr>
<td>Command injection</td>
<td>59,339</td>
<td>59,383</td>
<td>59,540</td>
</tr>
<tr>
<td>ReDoS vulnerabilities</td>
<td>210</td>
<td>540</td>
<td>633</td>
</tr>
<tr>
<td>Buffer vulnerabilities</td>
<td>5,740</td>
<td>6,007</td>
<td>6,084</td>
</tr>
<tr>
<td>Client-side programs</td>
<td>5,919</td>
<td>19,555</td>
<td>20,890</td>
</tr>
</tbody>
</table>

2.5-fold increase in runtime operations when considering implicit flows
Conclusions

56 benchmarks
integrity, availability, confidentiality; mostly non-malicious
Conclusions

56 benchmarks
integrity, availability, confidentiality; mostly non-malicious

Novel metrics
label creep ratio (LCR), sensitive branch coverage (SBC)
56 benchmarks
integrity, availability, confidentiality; mostly non-malicious

Novel metrics
label creep ratio (LCR), sensitive branch coverage (SBC)

Implicit flows
expensive to track, limited value
Conclusions

56 benchmarks
integrity, availability, confidentiality; mostly non-malicious

Novel metrics
label creep ratio (LCR), sensitive branch coverage (SBC)

Implicit flows
expensive to track, limited value

Taint analysis
suffices for integrity and availability benchmarks
56 benchmarks
integrity, availability, confidentiality; mostly non-malicious

Novel metrics
label creep ratio (LCR), sensitive branch coverage (SBC)

Implicit flows
expensive to track, limited value

Taint analysis
suffices for integrity and availability benchmarks