An Empirical Study of Information Flows in Real-World JavaScript

Cristian-Alexandru Staicu[1] Daniel Schoepe[2]
Musard Balliu[3] Michael Pradel[4] Andrei Sabelfeld[2]

[1]TU Darmstadt

[2]Chalmers University of Technology

[3]KTH Royal Institute of Technology

[4]University of Stuttgart

15th of November 2019

Program Analyses of Different Complexity

```
// 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

```
// 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

lightweight $\rightarrow$ heavyweight		
taint tracking	observable tracking	information flow control

```
// 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 (2)

lightweight → heavyweight		
taint tracking	observable tracking	information flow control

```
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)

lightweight → heavyweight		
taint tracking	observable tracking	information flow control

```
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)

lightweight → heavyweight		
taint tracking	observable tracking	information flow control

```
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)

lightweight → heavyweight		
taint tracking	observable tracking	information flow control

```
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)

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

```
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)

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

```
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)

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

```
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)

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

```
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**

```
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)

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

```
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)

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

```
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

Cost-Benefit Analysis

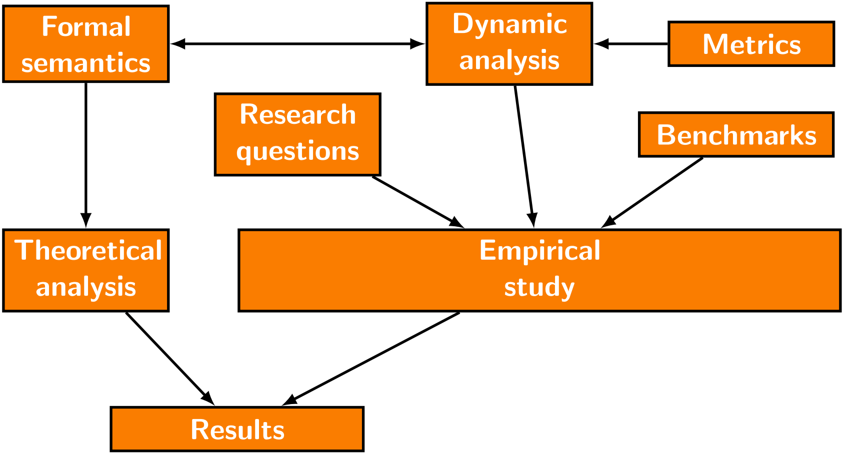
Benefits:

- more fine-grained flows
- more detected vulnerabilities

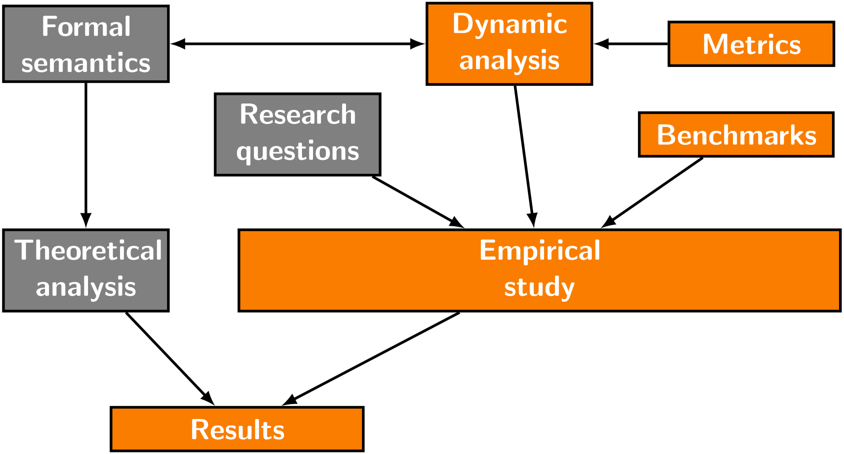
Costs:

- runtime cost
- label creep
- permissiveness

Overview


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

Overview


Formal semantics
Dynamic analysis
Metrics
Research questions
Benchmarks
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.

```
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.

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;
```

$LCR = 0.33$

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;
```

$LCR = 0.33$

```
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.

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

Inference of Upgrade Statements

Definition

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

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

```
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%

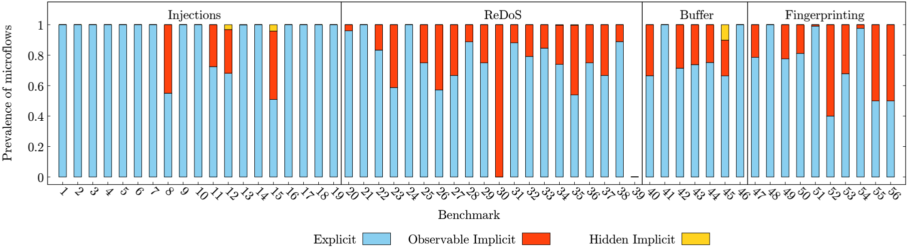
Implementation

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

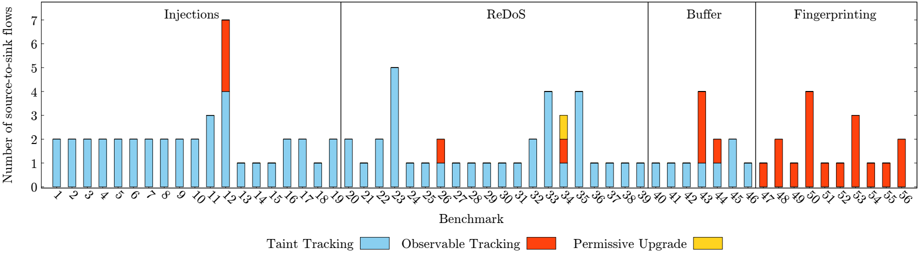
Benchmarks

- 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

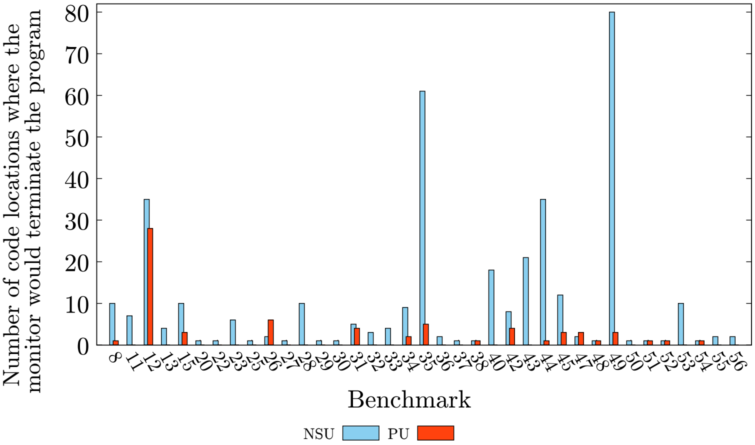
Benefit: Microflows



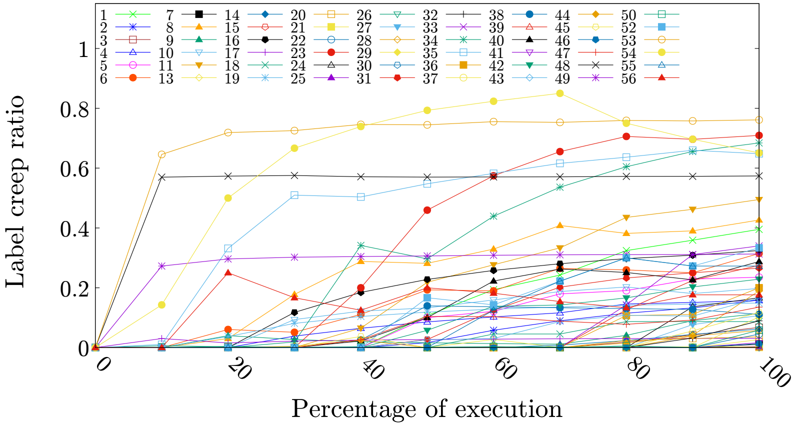
Benefit: Source-to-Sink Flows



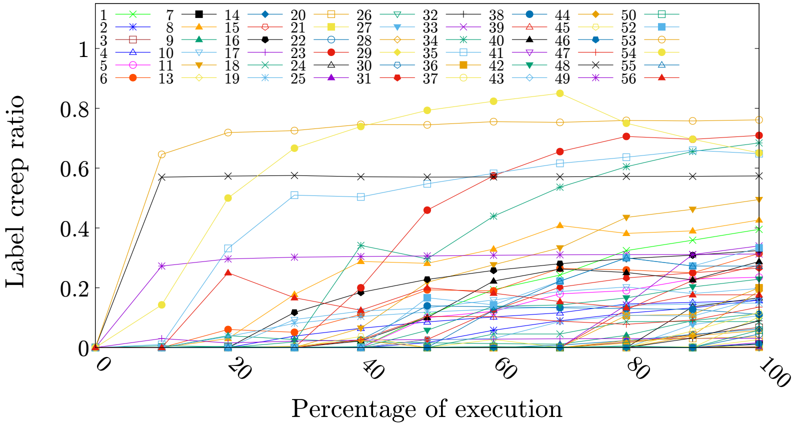
Cost: Permissiveness



Cost: LCR - Label Creep Ratio

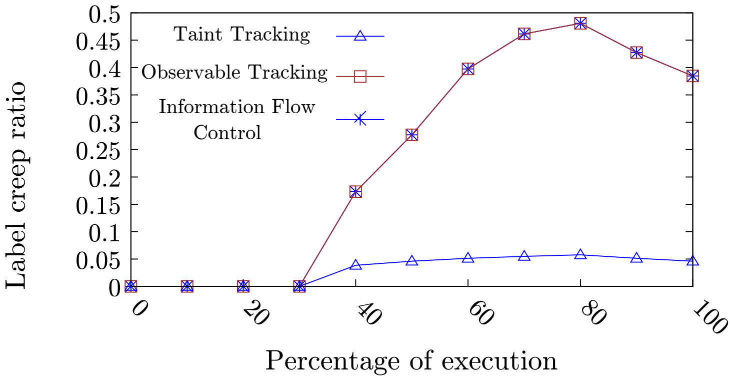


Cost: LCR - Label Creep Ratio



Average $max(LCR) = $ **20%**

Cost: LCR - Label Creep Ratio (2)



Cost: Runtime Overhead

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

	Taint Tracking	Observable Tracking	Information Flow Control
Command injection	59,339	59,383	59,540
ReDoS vulnerabilities	210	540	633
Buffer vulnerabilities	5,740	6,007	6,084
Client-side programs	5,919	19,555	20,890

Cost: Runtime Overhead

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

	Taint Tracking	Observable Tracking	Information Flow Control
Command injection	59,339	59,383	59,540
ReDoS vulnerabilities	210	540	633
Buffer vulnerabilities	5,740	6,007	6,084
Client-side programs	5,919	19,555	20,890

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)

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

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

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