FASE: Functionality-Aware Security Enforcement

Petar Tsankov
ETH Zurich

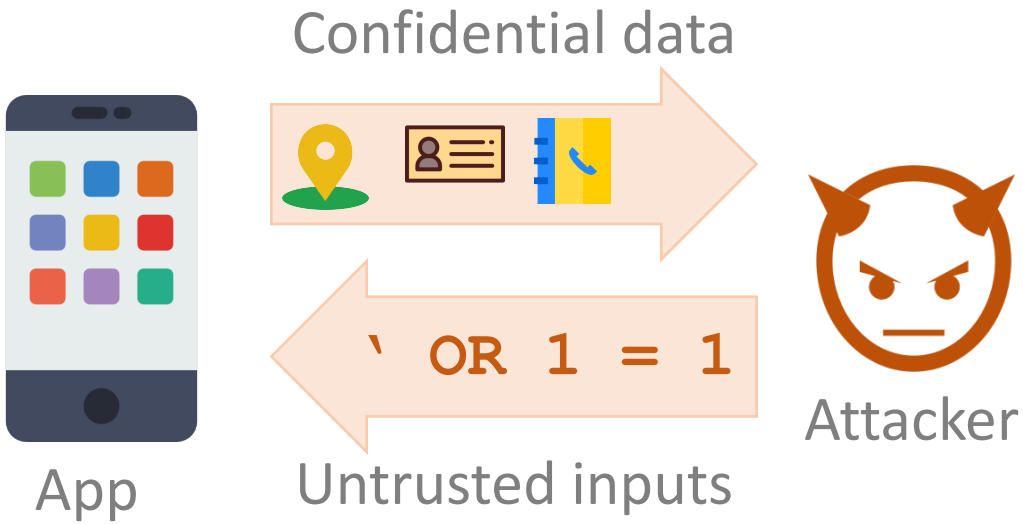
Marco Pistoia
IBM T.J. Watson Research Center

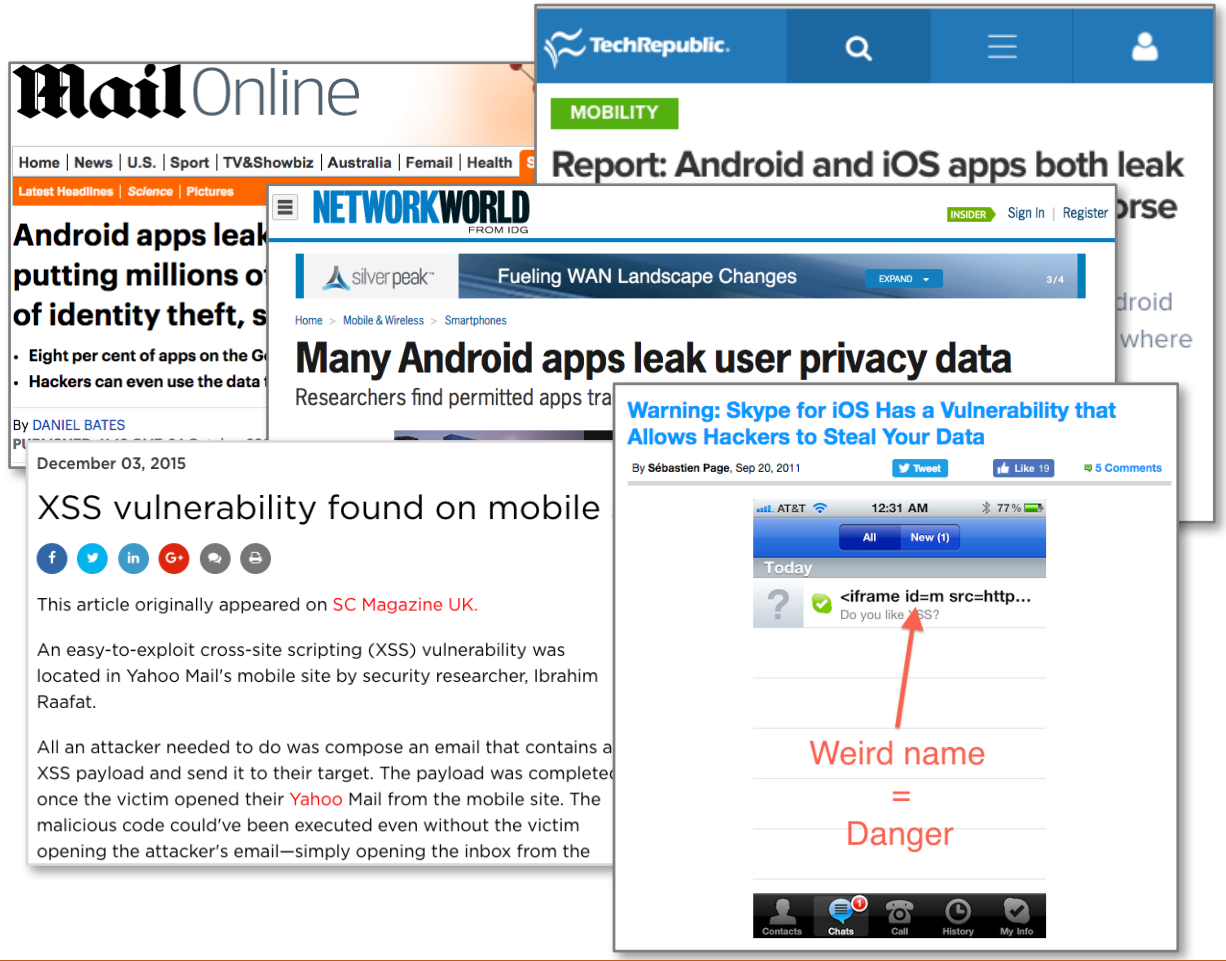
Omer Tripp
Google Inc.

Martin Vechev
ETH Zurich

Pietro Ferrara
Julia

Information Flow Vulnerabilities in Mobile Apps





Manual analysis of information flow threats is challenging

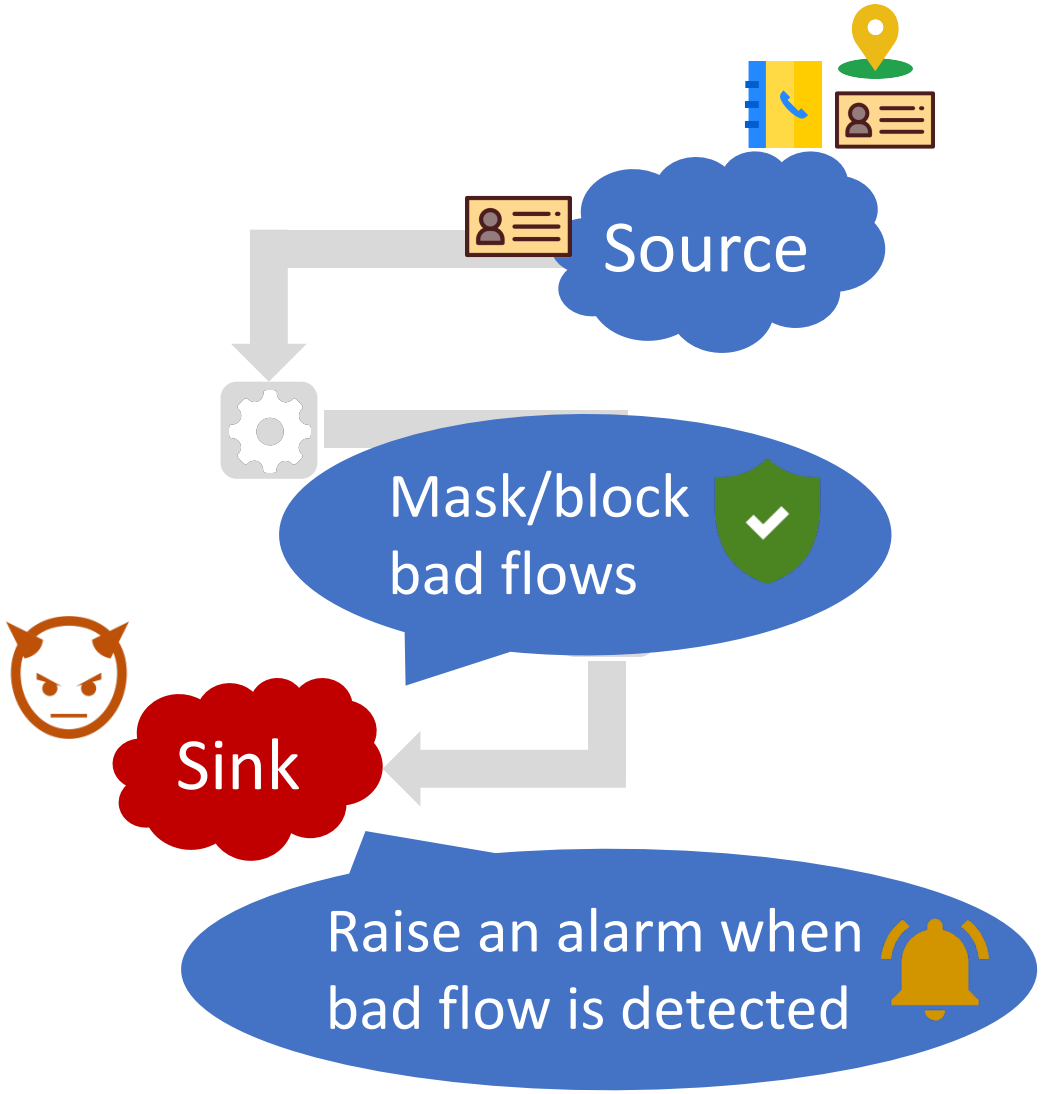
Existing Solutions

Detection

- TaintDroid (dynamic)
- FlowDroid (static)

Enforcement

- AppFence
  (masking & blocking)



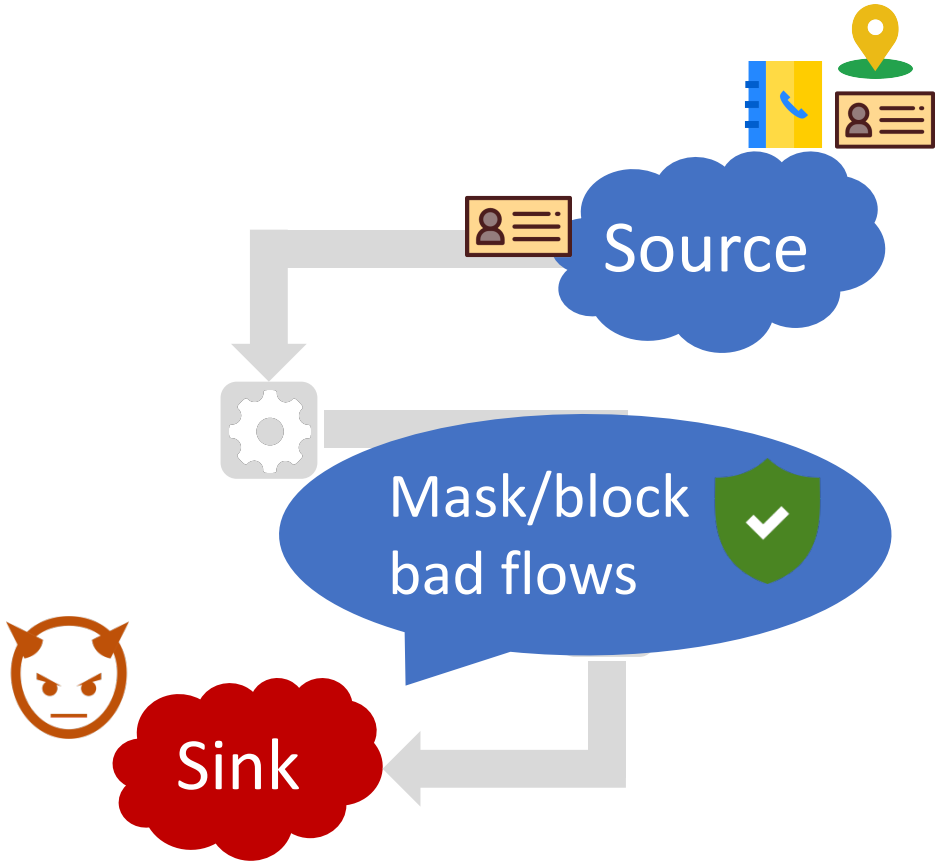
Existing Solutions

Detection

- TaintDroid (dynamic)
- FlowDroid (static)

Enforcement

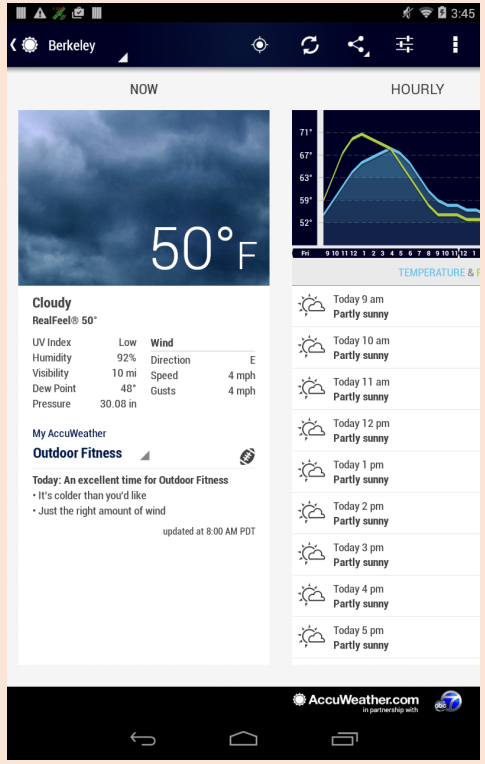
- AppFence
  (masking & blocking)



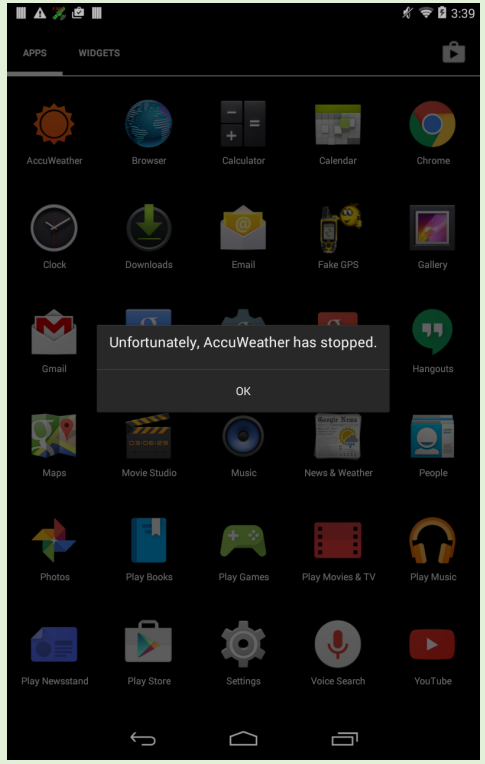
However, ***correct*** security enforcement depends on the app's ***functionality***

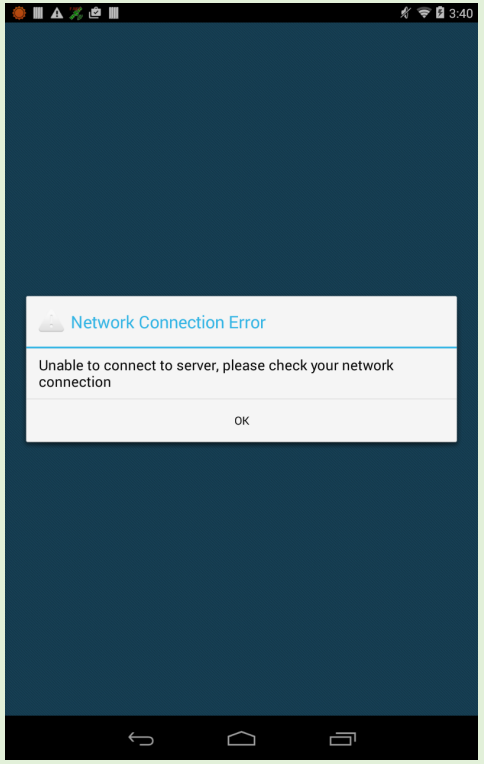
The Lack of Functionality-Awareness

Normal Behavior



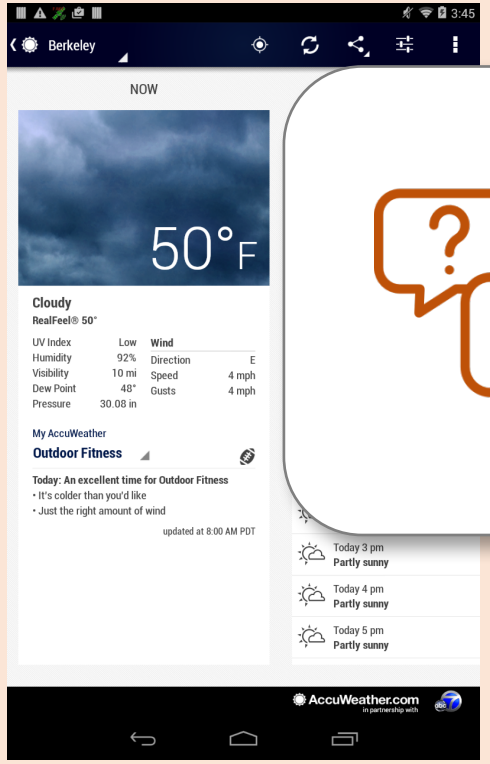
App secured with AppFence



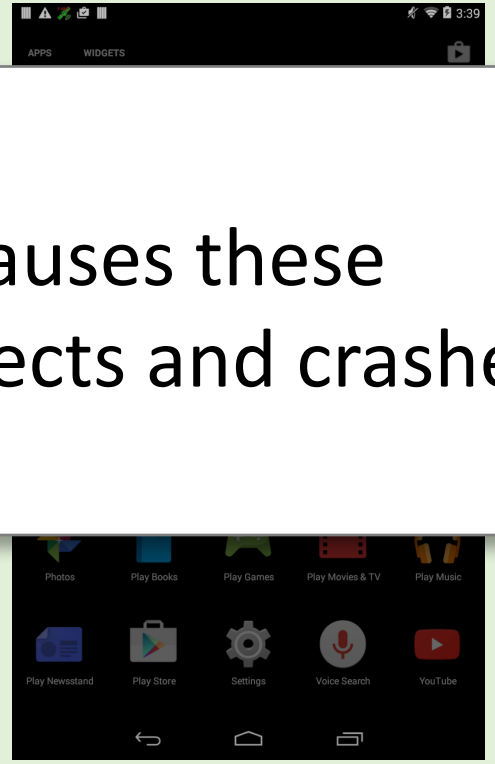


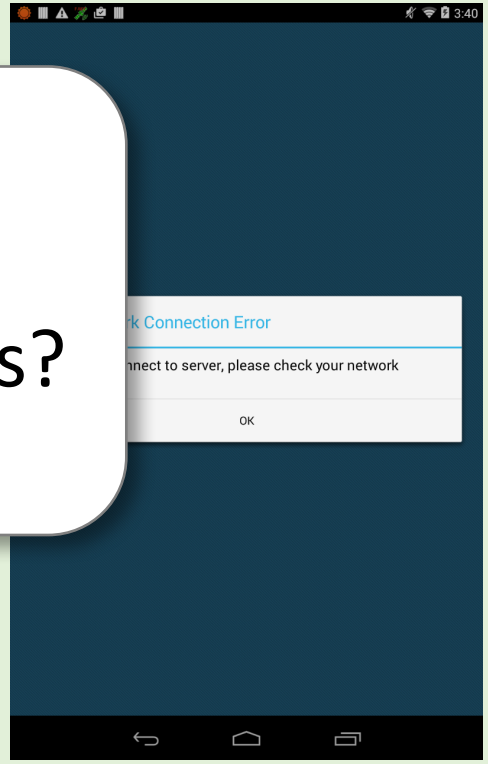
The Lack of Functionality-Awareness

Normal Behavior



App secured with AppFence





What causes these
side effects and crashes?

Illustrative Example

Illustrative Example

Source returns the International Mobile Subscriber Identity (IMSI)

```
String imsi = getSubscriberId(); // source
// imsi ↦ "310152843957264"
HttpGet request = new HttpGet("analytics.com?id=" + imsi);
// request.uri ↦ "analytics.com?id=310152843957264"
httpClient.execute(req); // sink
```

The IMSI flows into a ***sink*** as part of the URI

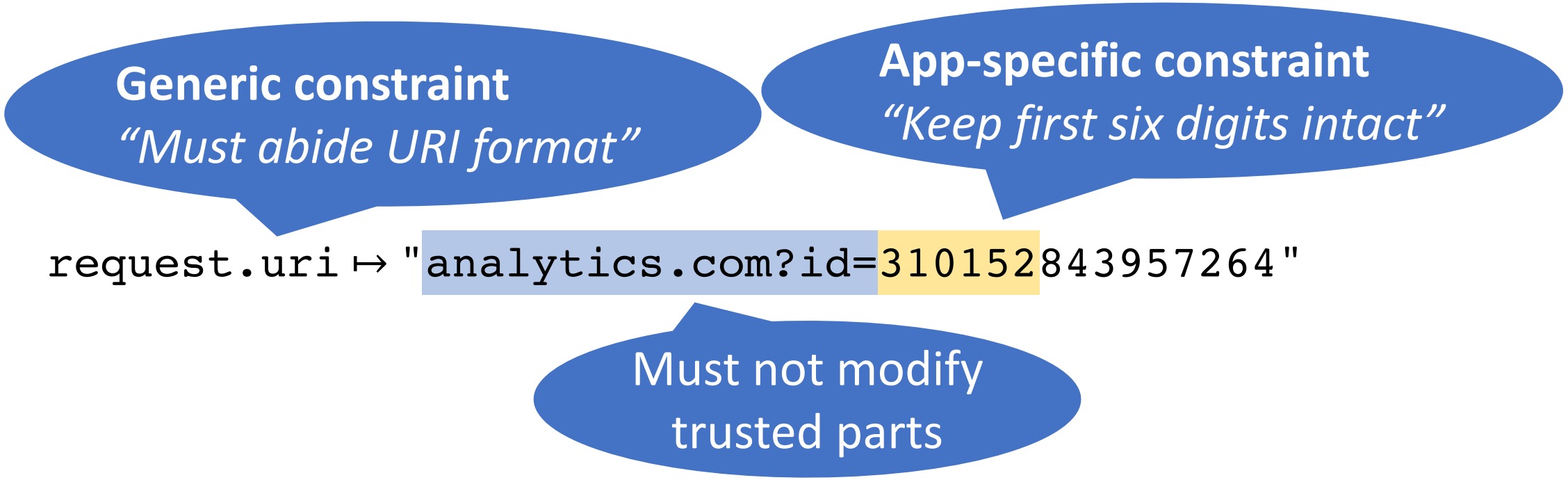
Illustrative Example

Source returns the International Mobile Subscriber Identity (IMSI)

```
String imsi = getSubscriberId(); // source
// imsi ↦ "310152843957264"
HttpGet request = new HttpGet("analytics.com?id=" + imsi);
// request.uri ↦ "analytics.com?id=310152843957264"
httpClient.execute(req); // sink
```

How can we correctly anonymize the URI that contains the IMSI?

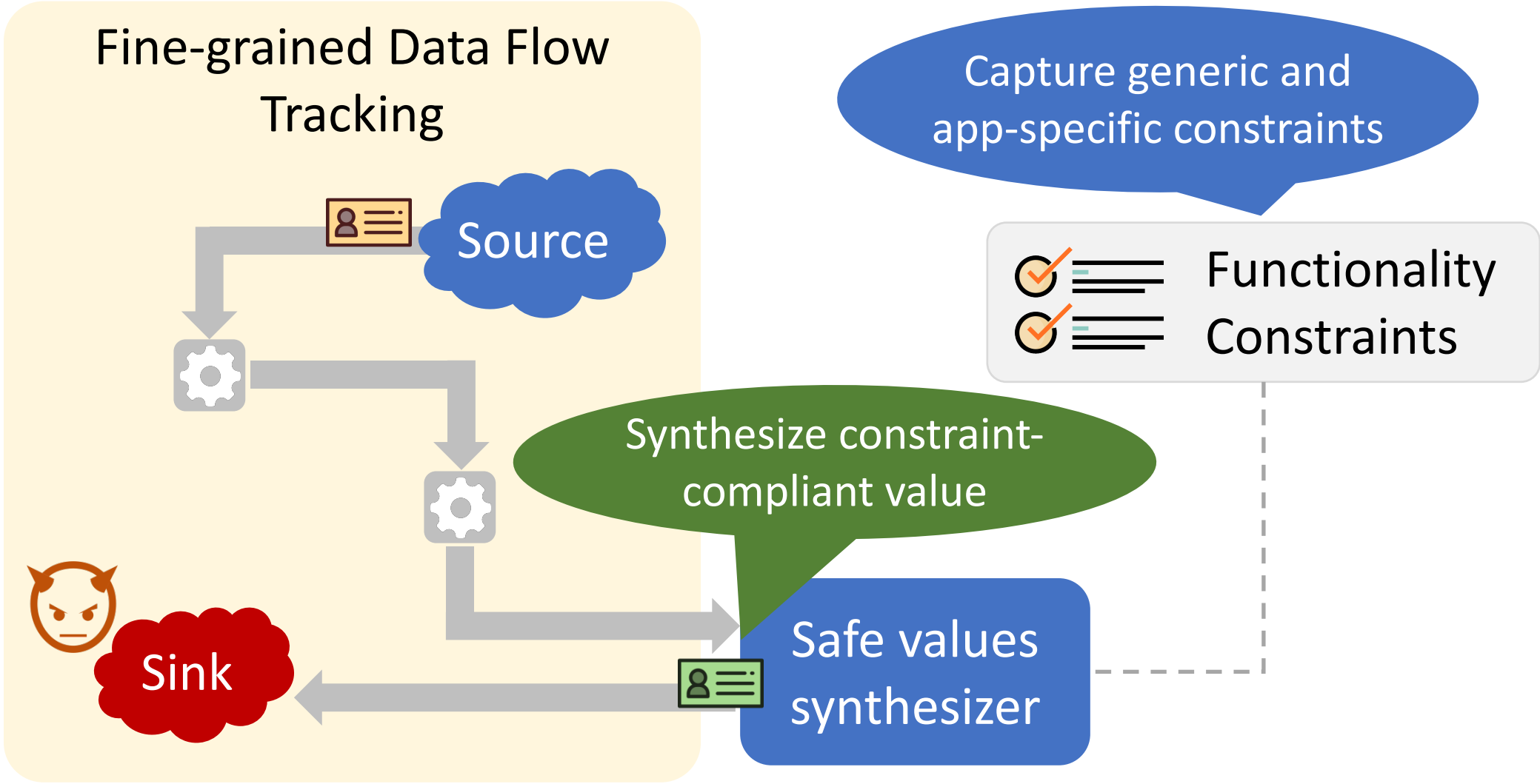
Common Functionality Constraints



`request.uri = "XYZ"`	**Incorrect**
`request.uri = "xyz.com?id=XYZ"`	**Incorrect**
`request.uri = "analytics.com?id=0000000000000000"`	**Incorrect**
`request.uri = "analytics.com?id=3101520000000000"`	**Correct**

How can we enforce security while satisfying such functionality constraints?

Functionality-Aware Security Enforcement (FASE)



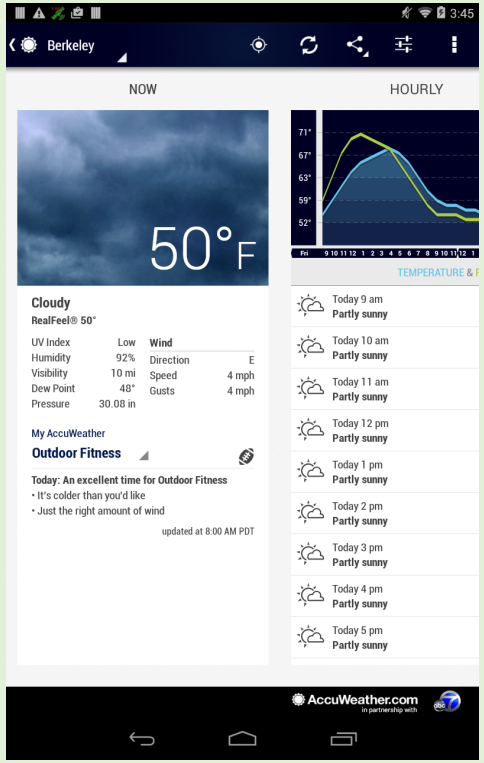
Functionality-Aware Security Enforcement (FASE)

Fine-grained Data F
Tracking

Sou

Sink

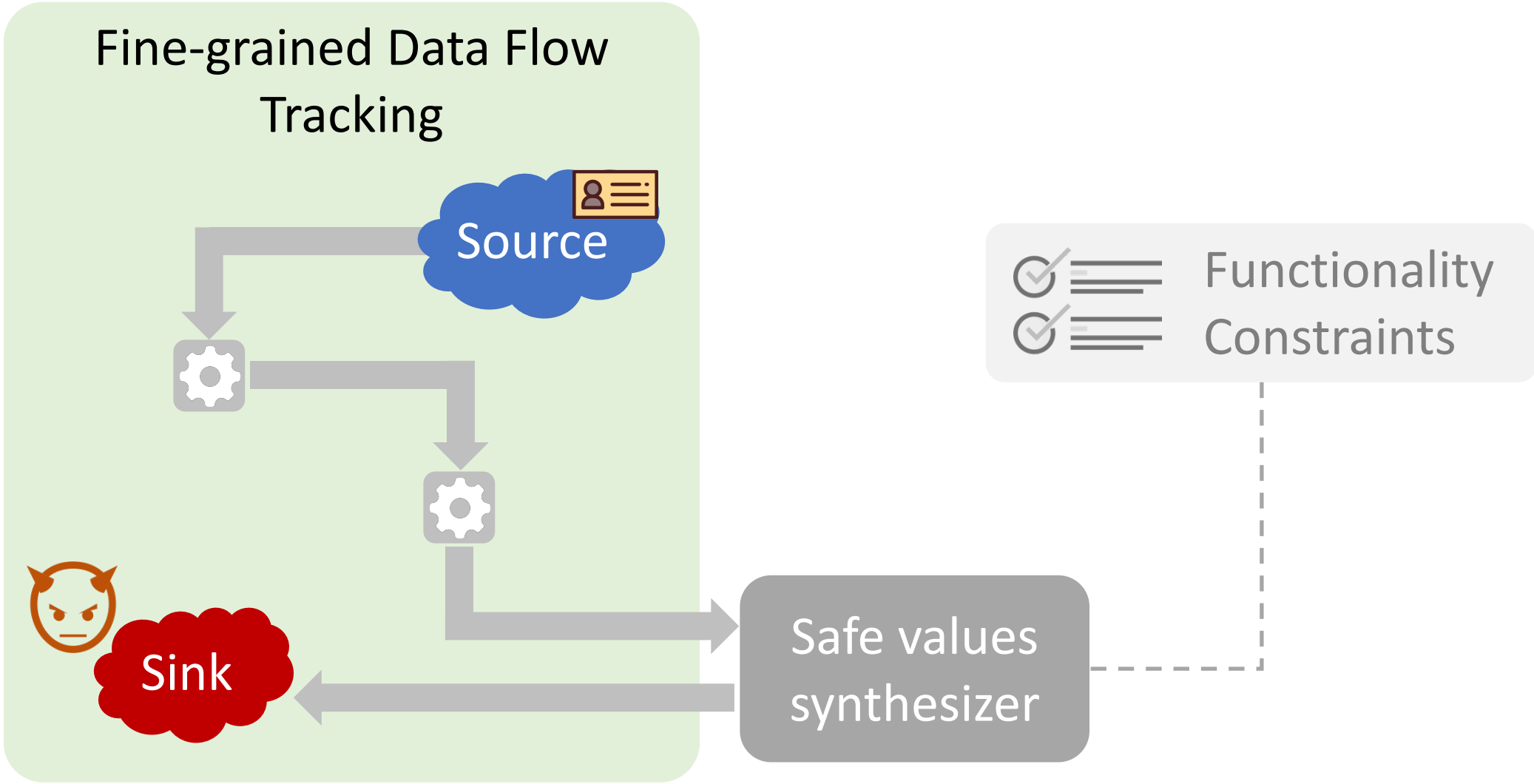
App secured with FASE



eric and
nstraints

Functionality
Constraints

Functionality-Aware Security Enforcement (FASE)



Fine-Grained Data Flow Tracking

Character-level Tracking for Strings

```
String imsi = getSubscriberId(); // source (IMSI)
// imsi ↦ "310152843957264"
HttpGet request = new HttpGet("analytics.com?id=" + imsi);
// request.uri ↦ "analytics.com?id=310152843957264"
```

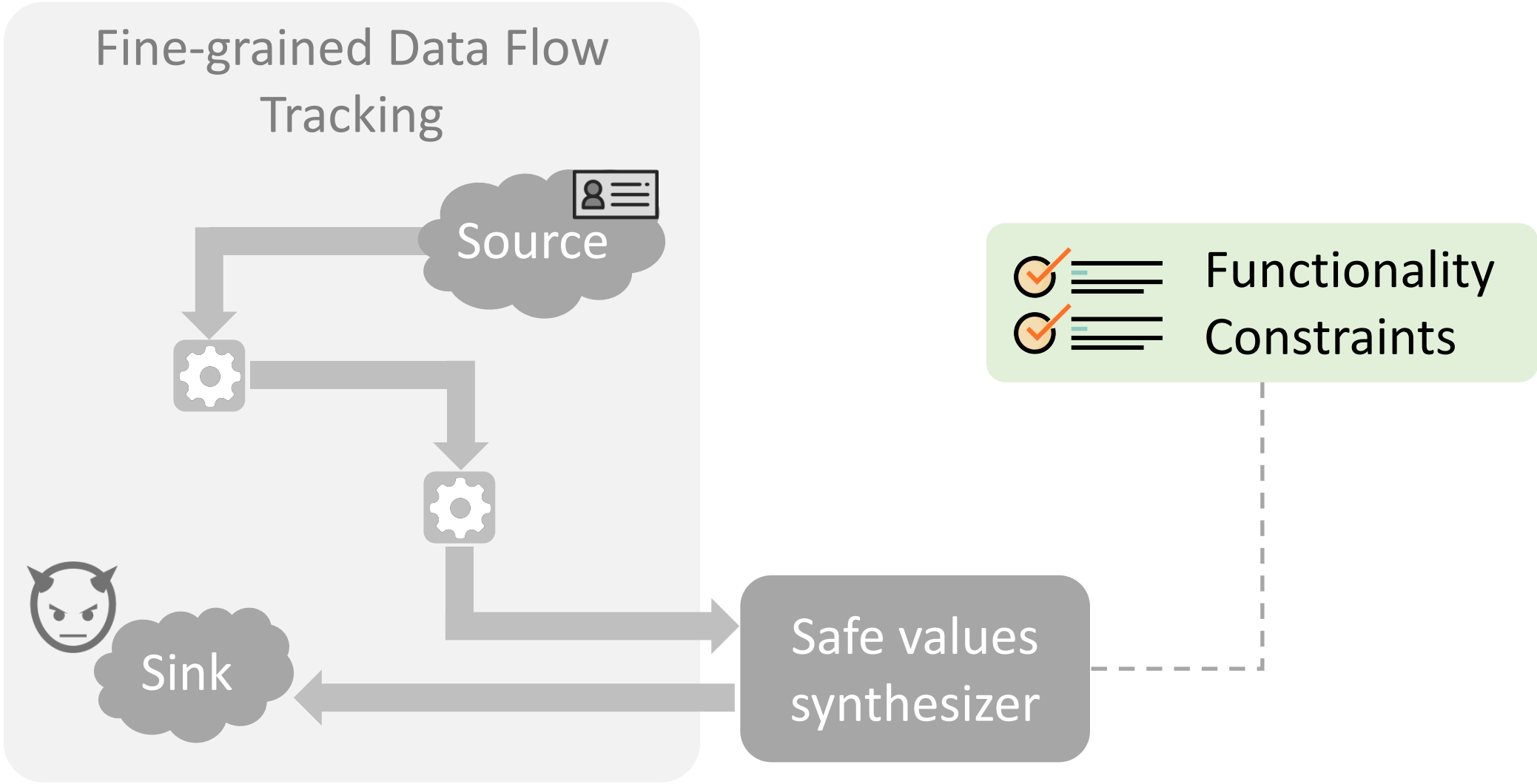
Value-based Tracking for Primitives

```
Location l = getLastKnownLocation(GPS);
// l.lat ↦ 37.3876, l.lon ↦ 122.0575
```

Each character is mapped to label `IMSI`

Each value is mapped to label `Location`

Functionality-Aware Security Enforcement (FASE)



Two Kinds of Functionality Constraints

Generic

- Specified once for all apps
- Capture sink pre-conditions

Example: *"URI strings must be valid"*

```
<Uri>   ::= "http" "s"? "://"
            <Chars> "." <Dom> <Args>
<Chars> ::= [a-zA-Z0-9]+
<Dom>   ::= "com" | "net" | ...
```

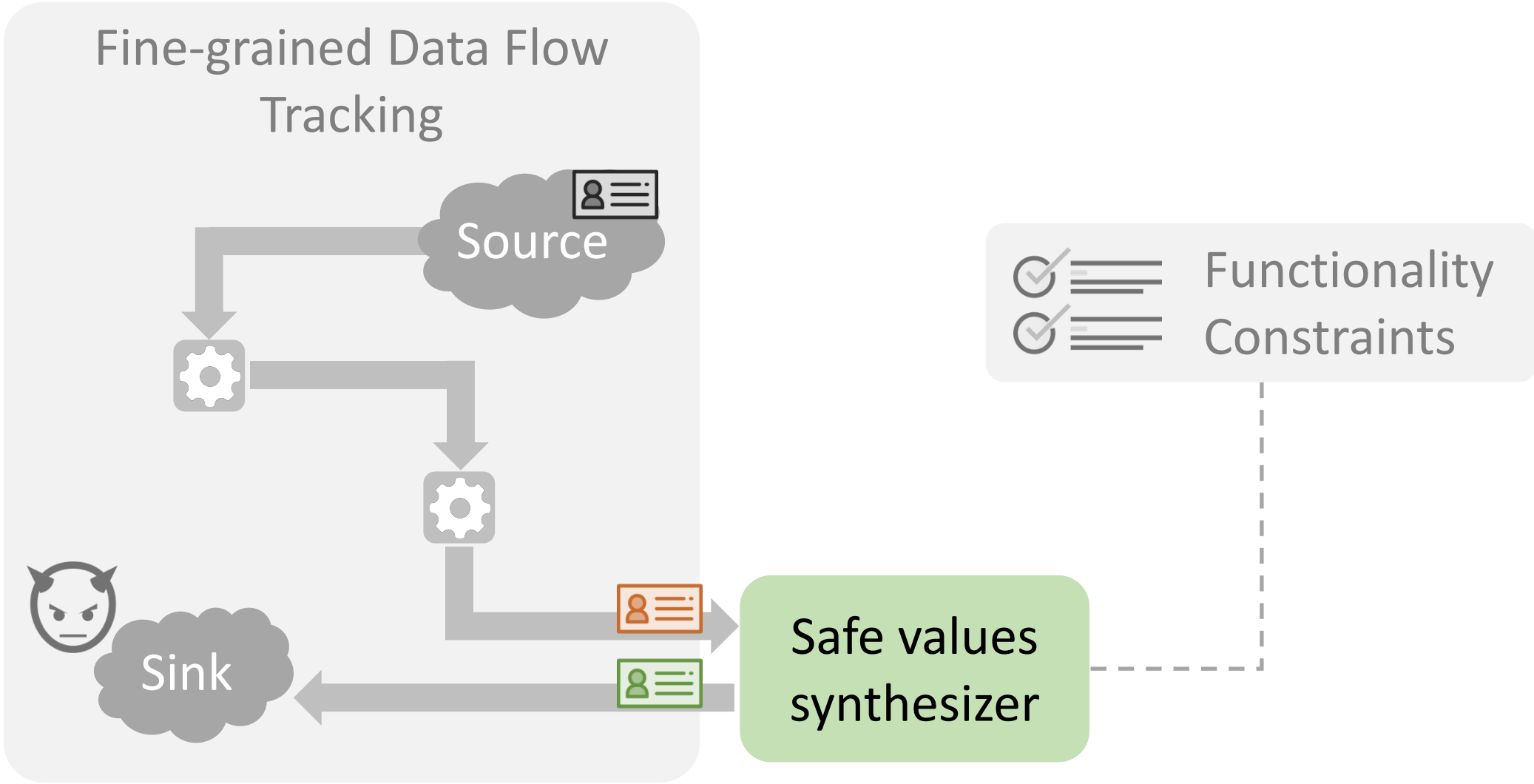
Application-specific

- Specified by developers
- Captured in a designated DSL

Example: *"First 6 chars of IMSI must be kept intact when sent to analytics.com"*

```
if uri.startsWith("analytics.com")
  constrain uri< IMSI >
    to val.substr(0,6).[0-9]^9
```

Functionality-Aware Security Enforcement (FASE)



Synthesizer

Labeled string

```
"analytics.com?id= 310152843957264 "
```

App-specific constraint

```
if uri.startsWith("analytics.com")
  constrain uri< IMSI >
    to val.substr(0,6).[0-9]⁹
```

Derived regular expression

```
"analytics.com?id=310152".[0-9]⁹
```

Generic constraint

```
<Uri> ::= [a-zA-Z0-9]⁺ . <Dom>
```

Constraint-compliant string

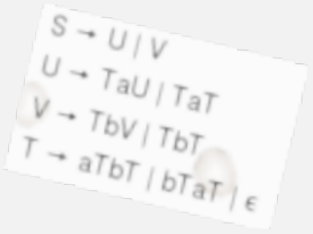
```
"analytics.com?id=3101520000000"
```

Implementation & Experiments

FASE System Implementation

Data Flow Tracking

- Instruments Android Libraries (String, StringBuilder, ...) as well as sources and sinks (>10K)
- Efficiency achieved by locality-aware memory allocation for labels



Synthesizer

- Uses the ACLA framework for analysis context-free and regular languages
- Efficiency achieved by combination of caching and short-circuiting heuristics

App-level Instrumentation

- Rewrites source and sink calls to invoke synthesizer

Experiments

Robustness

Can the FASE system secure apps while preserving functionality?

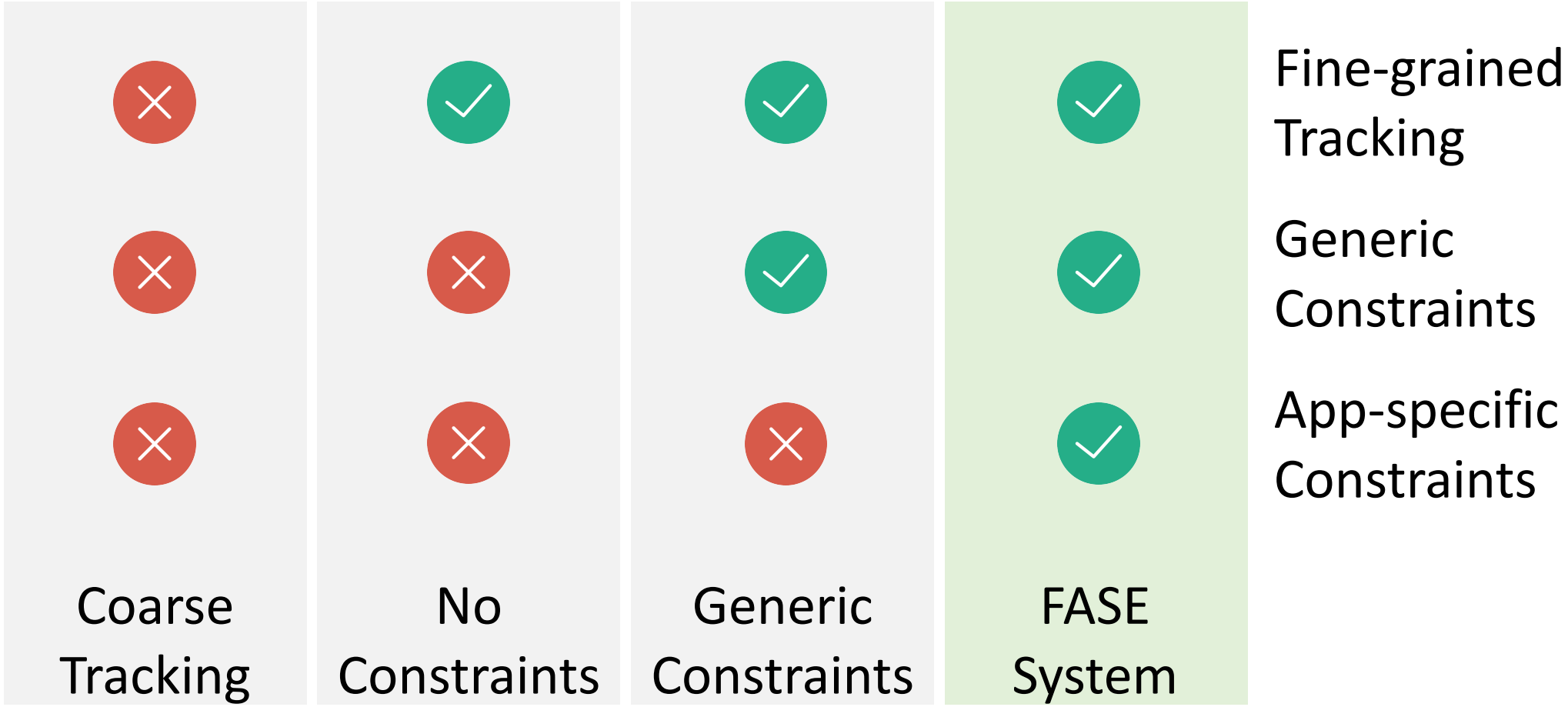
Overhead

What is the overhead caused by the FASE system?

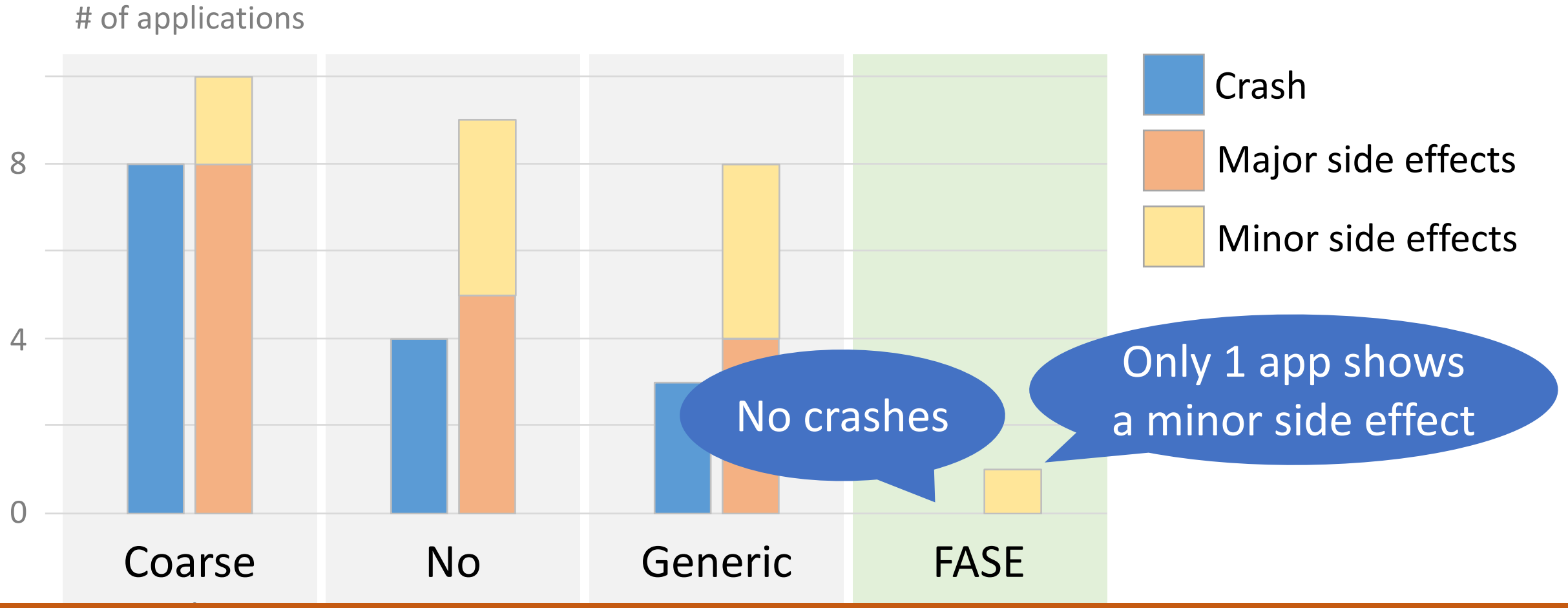
Benchmark Applications

- 20 apps used in prior studies
- On average, these apps have 500 source/sink call sites and 10 security-relevant flows

Robustness Experiment

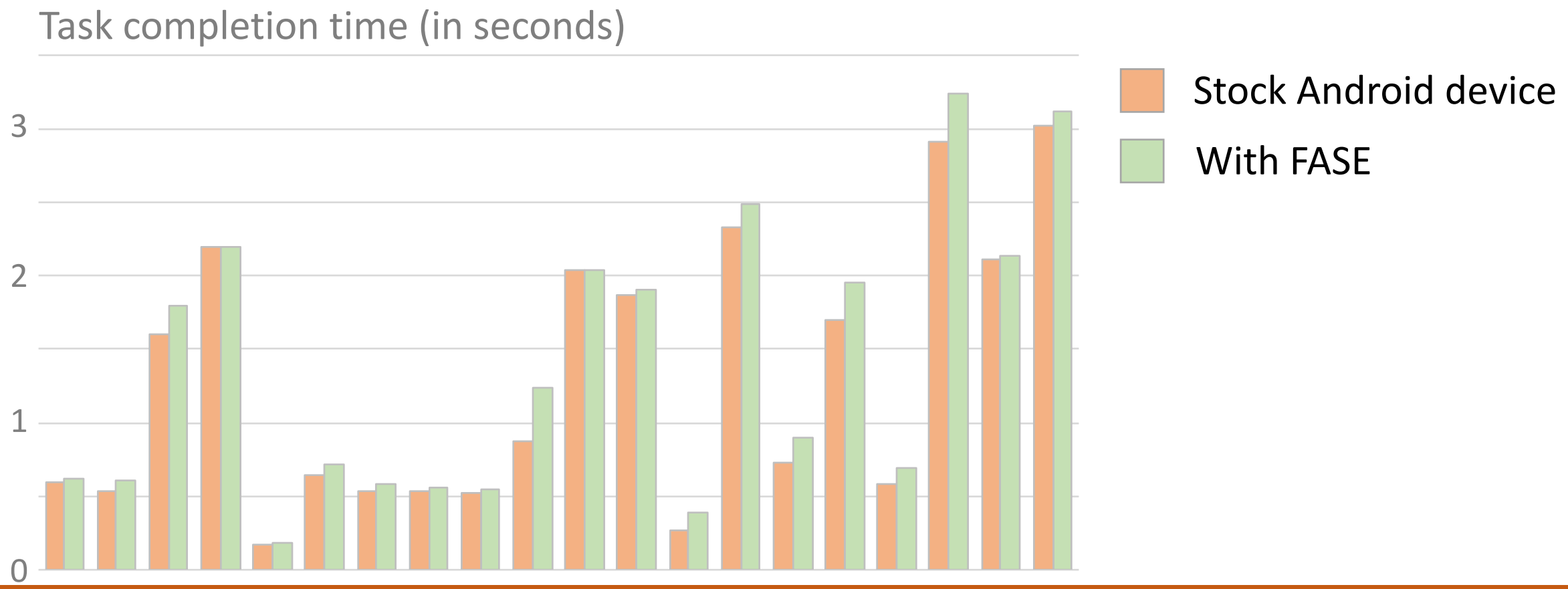


Robustness Experiment



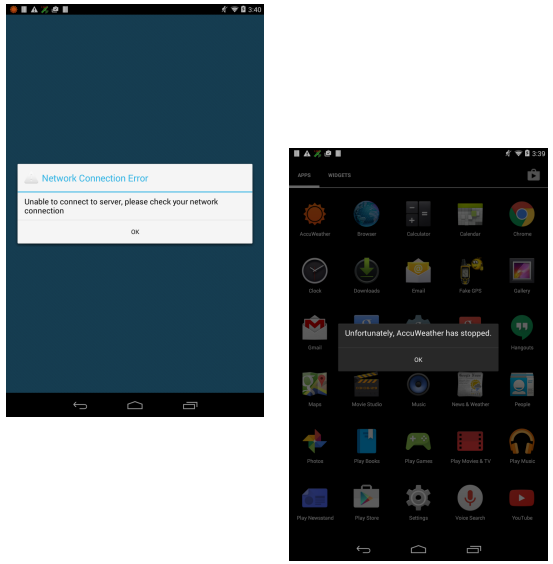
The FASE system secures apps in a robust way

Overhead Experiment

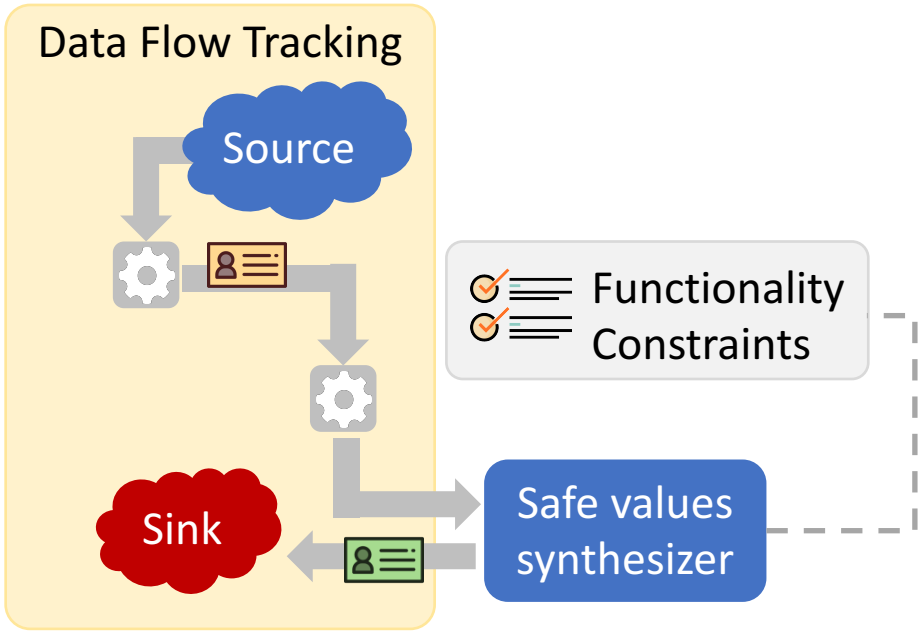


Roughly 10% overhead

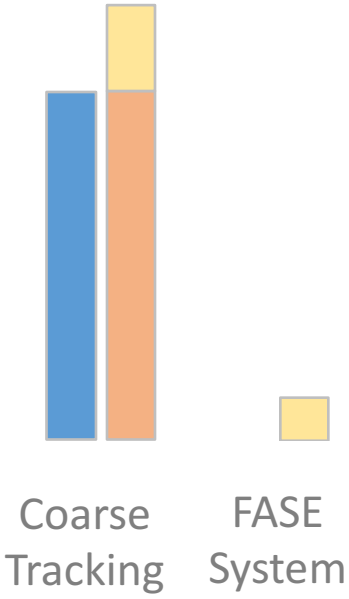
Summary



Existing enforcement solutions often ***break*** functionality



Functionality-aware security enforcement



Robust security enforcement with low overhead