FASE: Functionality-Aware Security Enforcement
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)

Source

Sink

Raise an alarm when bad flow is detected

Mask/block bad flows
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

String imsi = getSubscriberId(); \(\text{// source}\)

// imsi\(\mapsto\)"310152843957264"

HttpGet request = new HttpGet("analytics.com?id=" + imsi);
// request.uri\(\mapsto\)"analytics.com?id=310152843957264"

httpClient.execute(req); \(\text{// sink}\)

Source returns the International Mobile Subscriber Identity (IMSI)

The IMSI flows into a sink as part of the URI
Illustrative Example

```java
String imsi = getSubscriberId(); // source
// imsi \rightarrow "310152843957264"

HttpGet request = new HttpGet("analytics.com?id=" + imsi);
// request.uri \rightarrow "analytics.com?id=310152843957264"

httpClient.execute(req); // sink
```

How can we correctly anonymize the URI that contains the IMSI?
Common Functionality Constraints

**Generic constraint**
“Must abide URI format”

**App-specific constraint**
“Keep first six digits intact”

```plaintext
request.uri ← "analytics.com?id=310152843957264"
```

**Must not modify trusted parts**

<table>
<thead>
<tr>
<th>Original URI</th>
<th>Correctness</th>
</tr>
</thead>
<tbody>
<tr>
<td>request.uri = &quot;XYZ&quot;</td>
<td>Incorrect</td>
</tr>
<tr>
<td>request.uri = &quot;xyz.com?id=XYZ&quot;</td>
<td>Incorrect</td>
</tr>
<tr>
<td>request.uri = &quot;analytics.com?id=0000000000000000&quot;</td>
<td>Incorrect</td>
</tr>
<tr>
<td>request.uri = &quot;analytics.com?id=3101520000000000&quot;</td>
<td>Correct</td>
</tr>
</tbody>
</table>
How can we enforce security while satisfying such functionality constraints?
Functionality-Aware Security Enforcement (FASE)

Fine-grained Data Flow Tracking

Source

Sink

Capture generic and app-specific constraints

Functionality Constraints

Synthesize constraint-compliant value

Safe values synthesizer

Capture generic and app-specific constraints
Fine-grained Data Flow Tracking

Functionality-Aware Security Enforcement (FASE)

App secured with FASE
Fine-grained Data Flow Tracking

Source

Sink

Safe values synthesizer

Functionality-Aware Security Enforcement (FASE)

Functionality

Constraints
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
Functionality-Aware Security Enforcement (FASE)

Fine-grained Data Flow Tracking

Source

Sink

Safe values synthesizer

Functionality Constraints
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)

Fine-grained Data Flow Tracking

Source

Sink

Safe values synthesizer

Functionality Constraints
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=31015200000000000000"
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

<table>
<thead>
<tr>
<th>Coarse Tracking</th>
<th>No Constraints</th>
<th>Generic Constraints</th>
<th>FASE System</th>
</tr>
</thead>
<tbody>
<tr>
<td>✗</td>
<td>✗</td>
<td>✗</td>
<td>✗</td>
</tr>
<tr>
<td>✗</td>
<td>✗</td>
<td>✗</td>
<td>✗</td>
</tr>
<tr>
<td>✗</td>
<td>✗</td>
<td>✗</td>
<td>✗</td>
</tr>
</tbody>
</table>

Fine-grained Tracking
Generic Constraints
App-specific Constraints
The FASE system secures apps in a robust way.
Overhead Experiment

Task completion time (in seconds)

- Stock Android device
- With FASE

Roughly 10% overhead
Summary

Existing enforcement solutions often break functionality.

Functionality-aware security enforcement.

Robust security enforcement with low overhead.