
Radu Vanciu
Ebrahim Khalaj
Marwan Abi-Antoun
Department of Computer Science
Wayne State University
Motivating Example

- A hash code needs to be encrypted
- The developer follows the steps of an encryption algorithm
- The developer forgets one of the crucial steps of the algorithm
- The encryption is now broken

- Which approaches report the security vulnerability with good precision and recall?

```java
class CryptoStep {
    void missingStep() {
        MsgDigest md = new MsgDigest("");  
        IO io = new IO();
        String hashInput = new String("ABCDEFG123456");
        // Injected vulnerability: comment out next call
        // md.update(hashInput);
        io.writeLine(io.toHex(md.digest()));
    }
}
```
Contributions

• Comparative evaluation using test cases with injected vulnerabilities of:
  • **Scoria** as an **architecture-level** approach
  • **FlowDroid** as a **code-level** approach

• We compare approaches in terms of:
  • True positives (TP), Higher is better
  • False Positives (FP), Lower is better
  • False Negatives (FN), Lower is better

• Building a benchmark
  • Benchmarking is a common way to do the comparison, e.g., for compiler optimization
  • Enables reproducibility

• Introduce Architectural Flaw Index (AF-Index) to classify security vulnerability along the continuum
Research Questions

To find security vulnerabilities, approaches make tradeoffs

• Scoria tradeoffs: (Architecture-Level)
  • Sound and possibly less precise
  • Analyst-assisted approach
  • Special purpose constraints
  • Separate extraction and constraints
  • Extracts high-level representation for consumption by Security Information Workers (SIW)

• FlowDroid tradeoffs: (Code-Level)
  • Unsound and possibly more precise
  • More automated approach;
  • General purpose constraints
  • Combined extraction and constraints
  • Extract low-level representation used by the tools

• Which of the above tradeoffs leads to higher recall and precision?
Finding security vulnerabilities that are closer to **architectural flaws** is harder.

Architectural flaw
- e.g., missing authentication

Coding bug
- e.g., hard-coded password

- We are more interested in test cases that are closer to architectural flaws
**ScoriaBench**

- We built a benchmark with hand-selected test cases from different equivalence classes:
  - 43 hand-selected test cases
  - Android and Java applications
  - 13 different equivalence classes
- Selected test cases from:
  - DroidBench(DB)
  - NIST SAMATE Reference Dataset (SRD)
  - Examples from CERT Secure Coding Standard for Java
  - Designed by us (US)
ScoriaBench Equivalence Classes

<table>
<thead>
<tr>
<th>equivalence classes</th>
<th>#tests cases</th>
<th>DB</th>
<th>SRD</th>
<th>CERT</th>
<th>US</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arrays and Lists</td>
<td>1</td>
<td>✓</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Callbacks</td>
<td>5</td>
<td>✓</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Field and Object Sensitivity</td>
<td>7</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td></td>
</tr>
<tr>
<td>Inter-App Communication</td>
<td>3</td>
<td>✓</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Lifecycle</td>
<td>5</td>
<td>✓</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>General Java</td>
<td>3</td>
<td>✓</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Android-Specific</td>
<td>6</td>
<td>✓</td>
<td>✓</td>
<td></td>
<td>✓</td>
</tr>
<tr>
<td>Implicit Flows</td>
<td>4</td>
<td>✓</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Missing Encryption</td>
<td>4</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td></td>
</tr>
<tr>
<td>Bypass Authentication</td>
<td>2</td>
<td></td>
<td></td>
<td>✓</td>
<td></td>
</tr>
<tr>
<td>Command Injection</td>
<td>1</td>
<td></td>
<td>✓</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Exploitable Service</td>
<td>1</td>
<td></td>
<td></td>
<td>✓</td>
<td></td>
</tr>
<tr>
<td>Least Privilege Violation</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td>✓</td>
</tr>
<tr>
<td>Total</td>
<td>43</td>
<td>8</td>
<td>1</td>
<td>2</td>
<td>5</td>
</tr>
</tbody>
</table>

*Testcases in bold equivalence classes contain more architectural flaws. We added them on top of DroidBench equivalence classes.
Selection Process

- Common Weakness Enumeration (CWE)
  - Basis for a vulnerability
  - Parent-child relation

- We found such interesting CWEs
  - CWE-592: Authentication Bypass Issues
  - CWE-302: Authentication Bypass by Assumed-Immutable Data
  - CWE-290: Authentication Bypass by Spoofing
  - CWE-325: Missing Required Cryptographic Step

- We searched SRD to find test cases for those CWEs
Selection Process (cont'd)

• Some CWEs had an interesting test case
  • CWE-325: Missing Required Cryptographic Step
  • We added the test case to ScoriaBench

• Some CWEs had no corresponding test case:
  • CWE-592: Authentication Bypass Issues
  • CWE-290: Authentication Bypass by Spoofing
  • CWE-302: Authentication Bypass by Assumed-Immutable Data

• There is a gap that needs to be filled
Architecture-Level Approach: Scoria

Add and typecheck annotations
- Annotations express design intent

Extract high-level representation
- Sound over-approximation of runtime structure

Refine annotations

Write constraints to find vulnerabilities
- Enriched representation with security properties and queries
Scoria: Add Annotations

Code:
```java
class CryptoStep {
    void missingStep() {
        MsgDigest md = new MsgDigest(""); 
        IO io = new IO();
        String hashInput = new String("ABCDEFG123456");
        io.writeLine(io.toHex(md.digest()));
    }
}
```

Annotated code:
```java
@Domains("OWNED")
class CryptoStep {
    void missingStep() {
        @Domain("OWNED") MsgDigest md = new MsgDigest(""); 
        @Domain("OWNED") IO io = new IO();
        @Domain("unique") String hashInput = new String("ABCDEFG123456");
        io.writeLine(io.toHex(md.digest()));
    }
}
```

- Express design intent
- Impose arbitrary hierarchy on objects [in the object graph]

Consistent with the code:
- Implement a type system
- Not tags, they have meaning

- Support legacy code
Scoria: Extraction Static Analysis

- Soundness
- Aliasing
- Precision
- Summarization
- High-level View
- Traceability
Scoria: Constraints

Object Provenance

Indirect Communication

Object Hierarchy

Object Reachability
Architectural Flaw Index (AF-index)

- Measures if a test case is an architectural flaw or coding bug
- In ScoriaBench we have AF-index from 0 to 10:
  - 0 is the completely code-level vulnerability
  - 10 is the completely architecture vulnerability
- **Higher** AF-index means a test case is placed closer to the left of the continuum
How to compute AF-index?

- Assign a weight to each Scoria constraint
- Compute the summation of weighted constraints that are used to find the vulnerability using Scoria
Code-Level Approach: FlowDroid

- **FlowDroid** [Arzt et al., PLDI, 2014]
  - Reasons about information flow at the level of variables
  - Combines extractions and constraints
  - Find a path form a source to a sink
  - Mainly developed for Android apps
class CryptoStep {
    void missingStep() {
        MsgDigest md = new MsgDigest("");
        IO io = new IO();
        String hashInput = new String("ABCDEFG123456");
        md.update(hashInput);
        io.writeLine(io.toHex(md.digest()));
    }
}

class Main {
    public static void main(String[] args) {
        Main m = new Main();
        m.run();
    }
    void run() {
        IO io = new IO();
        CryptoStep cs = new CryptoStep();
        cs.missingStep();
    }
}

The SIW can notice the missing edge.
Scoria’s constraint for ACryptographic

• Found it using object provenance

(MsgDigest, cs.OWNED) -> (CryptoStep, main.LOGIC)

(CryptoStep, main.LOGIC) -> (MsgDigest, cs.OWNED)
FlowDroid’s constraint for ACryptographic

- The vulnerability in ACryptographic cannot be found by FlowDroid
- FlowDroid always looks for a confidential information flows to an untrusted sink
  - The confidential data that is supposed to flow to an untrusted sink is missing in ACryptographic
- So we cannot map the test case information flow into FlowDroid’s constraint form
  - \((C1, md1, \text{Property1}) \xrightarrow{*} (C2, md2, \text{Property2})\)
  - Cannot write such a constraint

Legend
C: class
md: method declaration
Property: security property, e.g., Source, Sink, Sanitizer
Overall Precision/Recall on ScoriaBench

Comparing Scoria and FlowDroid in terms of precision and recall

- Precision: Scoria 86%, FlowDroid 82%
- Recall: Scoria 82%, FlowDroid 70%
Precision per AF-index

![Bar chart showing precision per AF-index for FlowDroid and Scoria. The x-axis represents the AF-index, ranging from 0 to 10, and the y-axis represents the percentage, ranging from 0 to 100. The chart includes bars for each index point, with blue bars for FlowDroid and red bars for Scoria. Notable values include 92 for AF-index 2 for FlowDroid, 79 for AF-index 3 for FlowDroid, and 50 for AF-index 4, 5, 6, 7 for both FlowDroid and Scoria. There is a decrease in precision for AF-index 10 with FlowDroid at 33%.]
Recall per AF-index

![Graph showing recall percentage per AF-index]

- **FlowDroid**
- **Scoria**
Precision per Equivalence Class

[Bar chart showing the percentage of precision for different equivalence classes. The categories include Arrays, Lists, Callbacks, Field and Object, Sensitivity, Inter-App Communication, Lifecycle, General Java, Android-Specific, Implicit Flows, Missing Encryption, Bypass Authentication, Command Injection, Explorable Service, Least Privilege Violation. The bars are colored blue for FlowDroid and red for Scoria.]
Recall per Equivalence Class

![Bar Chart]

- **X-axis:** Equivalence Class
- **Y-axis:** Percentage
- **Legend:**
  - **FlowDroid**
  - **Scoria**

The chart compares the recall percentages across various equivalence classes for FlowDroid and Scoria.
Lessons Learned

- An architecture-level representation helps a SIW understand the system more than reading the code
- Many tools focus on coding bugs, not enough tools focus on architectural flaws
- Code-level approaches are less effective at finding architectural flaws than architecture-level ones
## Detailed Results

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>TP</td>
<td>FP</td>
</tr>
<tr>
<td>SRD</td>
<td>3</td>
<td>✔️</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>SRD</td>
<td>3</td>
<td>✔️</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>CERT</td>
<td>7</td>
<td>✔️</td>
<td>✔️</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>2</td>
<td>0</td>
</tr>
<tr>
<td>SRD</td>
<td>4</td>
<td>✔️</td>
<td>✔️</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>US</td>
<td>9</td>
<td>✔️ ✔️ ✔️</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>2</td>
<td>0</td>
</tr>
<tr>
<td>US</td>
<td>9</td>
<td>✔️ ✔️ ✔️</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>US</td>
<td>4</td>
<td>✔️</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>2</td>
<td>0</td>
</tr>
<tr>
<td>CERT</td>
<td>2</td>
<td>✔️</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>US</td>
<td>3</td>
<td>✔️</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>US</td>
<td>4</td>
<td>✔️</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

### Missing Encryption

- **ACipher**
- **ACipher2**
- **ASocket**
- **ACryptographic**

### Bypass Authentication

- **AToken1**
- **AToken2**

### Android Specific

- **AActivity**

### Command Injection

- **ARuntime**

### Exploitable Service

- **AChat**

### Least Privilege Violation

- **SecretViewer2**
Limitations

• We did not measure **effort** and **learnability**
  • In future work, we will measure effort
  • If an approach is good but requires a lot of effort to apply, people are not going to use it
Related Work

• Security benchmarks
  • SecuriBench Micro [Livshits, 2006]
    • Focuses on injection
    • Aliasing, collection and dataflow communication
  • MalGenome [Zhou et al., SSP, 2012]
    • A collection of 1200 malware Android applications

• Applications with injected vulnerabilities
  • Web apps
    • SecuriBench [Livshits et al., USS, 2005]
      • A collection of web applications with different sizes
  • Android apps
    • InsecureBank [Paladion, 2013]
      • Information disclosure to an external memory card

• Case studies on real-world applications
  • Evaluating some other approaches [Enck et al., USENIX, 2011]
  • Dynamic analyses [Enck et al., OSDI, 2010][Falcone et al., RV, 2013]
Future Work

• Add more test cases to ScoriaBench
  • More architectural flaws

• Evaluate more approaches
  • Architectural level [Almorsy et al., ICSE, 2013]
  • Type system such as Tainting Checker [Dietl et al., ICSE, 2011]
  • Sound static analysis such as JOANA [Graf et al., ATPS, 2013]
Conclusion

• Comparative evaluation of two approaches
  • Scoria, architectural level
  • FlowDroid, code level

• Introduce AF-index
  • A measure to classify test cases on the vulnerability continuum

• ScoriaBench
  • Consists of hand-selected test cases from different sources
  • Our extensions focus on architectural flaws
Call for action

• We encourage you to evaluate your approach on the benchmark
  • If you are interested let us know
• We encourage you to contribute test cases to the benchmark
  • We will run Scoria on your test cases