Is There Value in Reasoning about Security at the Architectural Level: a Comparative Evaluation

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

*This work was supported in part by the Army Research Office under Award No. W911NF-09-1-0273.
Finding security vulnerabilities that are closer to **architectural flaws** is harder.

### Architectural flaw
- e.g., missing authentication

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

#### Approaches make tradeoffs
- Sound and possibly less precise
- Analyst-assisted approach
- Special purpose constraints
- Separate extraction and constraints
- High-level representation of the system
- ...

- Unsound and possibly more precise
- More automated approach;
- General purpose constraints
- Combined extraction and constraints
- Code-oriented view of the system
- ...
Comparing approaches that find architectural flaws using a benchmark

- Some Common Weakness Enumerations (CWE) related to architectural flaws without corresponding testcases*
  - CWE-325: Missing Required Cryptographic Step (34 testcases)
  - CWE-311: Missing Encryption of Sensitive Data (no testcases)

- ScoriaBench
  - 43 hand-selected testcases
  - Android and Java applications
  - 13 different equivalence classes

- Selected test cases from
  - DroidBench(DB)
  - SAMATE Reference Dataset (SRD)
  - CERT rules examples
  - Designed by us (US)

*in SRD Juliet Test Suite for Java
Exploitable FindFriend Service

- No transitive information flow from Contacts to Client
- Brute force attack
  - reconstruct mapping user – phone number

Legend
- confidential
- untrusted
- trusted
- process
- data store
- data flow
Scoria process [Vanciu and Abi-Antoun, ASE'2013]

Add and typecheck annotations
- Annotations express design intent

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

Write constraints to find vulnerabilities
- Enriched representation with security properties and queries

Refine annotation

Annotated code:
```java
@DomainParams({"U","L","D"})
@Domains({"SLIST", "owned"})
class FFService {
    @Domain(owned<D,L>) HashMap<String,String> map = ...
    @Domain("SLIST>L") List<String> findByNumber(@Domain(">D") String num) {
        @Domain("SLIST>L") List<String> uNames = ...;
        //search for the number in the map
        return uNames;
    }
}
```

Constraint:
```
cp: ContactsProvider
number: String
mAct: ChatActivity
number: String
cln: Client
number: String
srv: FFService
```

Abstract object graph:
Results

• Compare in terms of precision and recall
  – Scoria
  – FlowDroid [Arzt et al., PLDI'2014]

Precision = \( \frac{TP}{TP+FP} \)
Recall = \( \frac{TP}{TP+FN} \)