Finding Architectural Flaws in Android Apps Is Easy

By Radu Vanciu and Marwan Abi-Antoun
Department of Computer Science
Wayne State University
Detroit, Michigan, USA

This work was supported in part by the Army Research Office under Award No. W911NF-09-1-0273
50% of security vulnerabilities are architectural [McGraw, Addison-Wesley’05]

- Other 50% are coding defects: local, found by analyzing one class at a time
- Architectural flaws: non-local, found by reasoning about usage context

<table>
<thead>
<tr>
<th>OWASP 2004* [web applications]</th>
</tr>
</thead>
<tbody>
<tr>
<td>A1 - Unvalidated Input</td>
</tr>
<tr>
<td>A2 - Broken Access Control</td>
</tr>
<tr>
<td>A3 - Broken Authentication and Session Management</td>
</tr>
<tr>
<td>A4 - Cross Site Scripting</td>
</tr>
<tr>
<td>A5 - Buffer Overflow</td>
</tr>
<tr>
<td>A6 - Injection Flaws</td>
</tr>
<tr>
<td>A7 - Improper Error Handling</td>
</tr>
<tr>
<td>A8 - Insecure Storage</td>
</tr>
<tr>
<td>A9 - Application Denial of Service</td>
</tr>
<tr>
<td>A10 - Insecure Configuration Management</td>
</tr>
</tbody>
</table>

*https://www.owasp.org/index.php/Top_10_2004
**https://www.owasp.org/index.php/Top_10_2013-Top_10
50% of security vulnerabilities are architectural [McGraw, Addison-Wesley’05]

- Other 50% are coding defects: local, found by analyzing one class at a time
- Architectural flaws: non-local, found by reasoning about usage context

<table>
<thead>
<tr>
<th>OWASP 2013** [web applications]</th>
</tr>
</thead>
<tbody>
<tr>
<td>A1 - Injection</td>
</tr>
<tr>
<td>A2 - Broken Authentication and Session Management</td>
</tr>
<tr>
<td>A3 - Cross-Site Scripting (XSS)</td>
</tr>
<tr>
<td>A4 - Insecure Direct Object References</td>
</tr>
<tr>
<td>A5 - Security Misconfiguration</td>
</tr>
<tr>
<td>A6 - Sensitive Data Exposure</td>
</tr>
<tr>
<td>A7 - Missing Function Level Access Control</td>
</tr>
<tr>
<td>A8 - Cross-Site Request Forgery (CSRF)</td>
</tr>
<tr>
<td>A9 - Using Components with Known Vulnerabilities</td>
</tr>
<tr>
<td>A10 - Unvalidated Redirects and Forwards</td>
</tr>
</tbody>
</table>

*https://www.owasp.org/index.php/Top_10_2004

**https://www.owasp.org/index.php/Top_10_2013-Top_10
Architectural flaws in mobile apps

- OWASP top 10 most critical risks in security of mobile applications*
- Most are architectural flaws
- >800K Android apps in Google Play store**
- >1 Billion Android devices activated*** (Sept 2013)

<table>
<thead>
<tr>
<th>OWASP 2013 [mobile applications]</th>
</tr>
</thead>
<tbody>
<tr>
<td>M1: Insecure Data Storage</td>
</tr>
<tr>
<td>M2: Weak Server Side Controls</td>
</tr>
<tr>
<td>M3: Insufficient Transport Layer Protection</td>
</tr>
<tr>
<td>M4: Client Side Injection</td>
</tr>
<tr>
<td>M5: Poor Authorization and Authentication</td>
</tr>
<tr>
<td>M6: Improper Session Handling</td>
</tr>
<tr>
<td>M7: Security Decisions Via Untrusted Inputs</td>
</tr>
<tr>
<td>M8: Side Channel Data Leakage</td>
</tr>
<tr>
<td>M9: Broken Cryptography</td>
</tr>
<tr>
<td>M10: Sensitive Information Disclosure</td>
</tr>
</tbody>
</table>

*** [http://www.androidcentral.com/android-passes-1-billion-activations](http://www.androidcentral.com/android-passes-1-billion-activations)
Architectural Risk Analysis helps to find architectural flaws [Howard and Lipner, Microsoft Press’06]

- Architects use forest-level view of system (not reading code)
  - Runtime architecture – not code architecture
  - Assign security properties to component instances

- Limitations of ARA approaches
  - Limited support for reverse engineering
  - Runtime architecture is missing or inconsistent
  - Lack of traceability to code
  - Analyses focused only on presence/absence of communication
Scoria: Security Constraints On Runtime Architecture

- Extract from code runtime architecture useful for security
  - Annotations convey design intent
  - Sound approximation of the runtime architecture
  - Supports hierarchical decomposition
    - Architectural relevant objects near the top of hierarchy
    - Implementation details further down
  - Reason about dataflow communication
    - Dataflow edge refers to objects

- Scoria helps architects to find architectural flaws
  - Support architects to write constraints as unit tests
Demo

- Find architectural flaws in one open-source Android app

- Subject system: Universal Password Manager Application (UPMA) (4KLOC)
  - Stores passwords in encrypted files
  - Annotated only code of UPMA, not for Android framework
  - Downloaded > 500K
Security is a worst-case analysis and requires soundness

- **Represent all** objects and relations that may exist at runtime

- Absence of a connector means absence of communication at runtime

- Find unexpected edges that may occur in exceptional or error handling cases

- Ensure that same runtime entity is not mapped to distinct components
Static analysis extracts sound approximation of runtime architecture [Vanciu and Abi-Antoun, FOOL’13]

- Nodes represent abstract objects
- Edges represent relations between abstract objects
- Directed graph
- Multiple types of edges
- Multigraph
At runtime, object oriented program appears as Runtime Object Graph.
Abstract multiple runtime objects into an abstract object

- Each runtime object has **exactly one** representative in extracted object graph
Abstract domain is a group of abstract objects

• Place each abstract object in exactly one conceptual group (abstract domain)
Abstract edge is between abstract objects

- Runtime edge between two objects maps to the abstract edge between the representatives of two objects
Edges between objects

- Points-to [label is field name]

- Dataflow [label refers to object]

- Creation [label refers to object]

- Control flow [label is method name]
Objects are organized hierarchically

- Abstract object can have abstract domains
- Each domain can have objects
- Hierarchy of objects extracted by analyzing code with annotations
- Domains provides precision
  - Distinguish between objects of same type in different domains
- At runtime object does not change domain
Architects distinguish between copying and sharing of object → Object identity

- Every abstract object is uniquely identified

- Enable comparison of references

Edges refer to same abstract object

Edges refer to distinct abstract objects of same type
Distinguish between objects of type File

- Selection queries
  - `getObjectsByCondition`
  - `getEdgesByCondition`
    - Return objects or edges that satisfy condition
- Condition based on:
  - Type + object hierarchy: `IsInDomain`, `isChildOf`
  - Type + object reachability: `IsInstOfRchblFromInstOf`
  - Type: `InstanceOf`
For information not directly extracted from code $\Rightarrow$ assign security properties

- Security property values for each component and connector
  - **TrustLevel**
    - Trusted(+)
    - Untrusted(-)
    - Unknown
  - **IsConfidential**
    - True
    - False
    - Unknown
  - **IsEncrypted**

- Using security properties
  - **Tampering:**
    - Untrusted(-) $\Rightarrow$ Trusted(+)
  - **Information Disclosure:**
    - Trusted(+) $\Rightarrow$ Untrusted(-)
Finding Architectural Flaws in Android app

- Intents are like command line arguments used to start an activity [Burns, Black Hat’09]

- Security policy: Don’t put sensitive data into Intents used to start Activities. Callers can’t easily require Manifest permissions of the Activities they start, and so your data might be exposed.
  - For example processes with the GET_TASKS permission are able to see ActivityManager.RecentTaskInformation which includes the base Intent used to start Activities.
Automating security reasoning \( \rightarrow \) queries

- **Property queries:** \( \text{setObjectsProperty}(\text{props, condition}) \), \( \text{setEdgesProperty}(\text{props, condition}) \)
  - Assign property values to objects or edges that satisfy condition

- **Condition based on:**
  - **Type + object hierarchy:** \( \text{IsInDomain, isChildOf} \)
  - **Type + object reachability:** \( \text{IsInstOfRchblFromInstOf} \)
  - **Type:** \( \text{InstanceOf} \)
Some objects that carry confidential data may be part of some other object → object hierarchy

• Only some objects are confidential, but architects also consider:
  • Descendant of object referred from dataflow edge is confidential
  • Ancestor of object referred from dataflow edge is confidential
Selection query in terms of architecturally relevant objects ➔ Indirect communication

$getFlowIntoSink(flow, sink)$

- Returns dataflow or creation edges
- Destination is descendant of $sink$, or object reachable from $sink$
- Edge refers to descendant of $flow$ or object reachable from $flow$
Use queries to assign security properties

- `setObjectsProperty(TrustLevel.Untrusted, InstanceOf(OutputStream))`

- `setObjectsProperty(IsConfidential.true, IsChildOf(String, AccountInformation))`

- `setObjectsProperty(IsConfidential.true, IsInDomain(String, PWD))`
Automating security reasoning ➔ machine checkable constraints on query results

getFlowIntoSink(IsConfidential.true, TrustLevel.Untrusted)

- Query in terms of security properties only
  - Return edges that refer to confidential object with an untrusted destination
  - Returned set is empty means: no confidential data flows to untrusted destination
  - Written in general terms — not system specific
Security Test: No confidential data flows to untrusted object

```java
@Test
public void checkInfDisclosureIntentPolicy() {
    secGraph.setObjectProperty(TrustLevelType.Low,
                                new InstanceOf(Intent.class));
    secGraph.setObjectProperty(IsConfidential.True,
                                new IsChildOf(AccountInformation.class, String.class));
    Property[] snkProps = { TrustLevelType.Low }; Property[] flwProps = { IsConfidential.True };
    if (secGraph.checkFlowIntoSink(snkProps, flwProps)) {
        Set<IEdge> sEdges = secGraph.getFlowIntoSink(snkProps, flwProps);
        scoria.displayWarnings(sEdges);
        Assert.fail("Information disclosure found");
    }
}
```
Multiple communication mechanisms for ClipboardManager in Android

- No explicit permissions needed to access clipboard
  - Most Password manager apps expose password in plaintext to clipboard
  - Developers criticized Android’s missing support for password manager apps [Fahl et al. FCDS’13]

Architects can reason about object provenance → return dataflow edges that refer to same object

- **Query: object provenance**

- **Constraint: return set is empty**
  - No object that flows from \(a:A\) to \(b:B\) also flows from \(c:C\) to \(d:D\)
Example of object provenance

- Same object `a:Account` that flows from `mgr:AccountManager` is saved by `act:Activity` into `outputStream:ObjectOutputStream`
Special case of object provenance ➔
object transitivity

- Destination of e1 is source of e2

- Constraint: return set is empty
  - Object that flows from a:A to b:B does not flow to c:C
Example of object transitivity

- Object a:Account flows from mgr:AccountManager to outStream:ObjectOutputStream through some intermediate objects.
Limitation: false positives

- UPMA - password is sent to a text view for a user to see
- This is the intended feature in UPMA, not an architectural flaw

```java
Class ViewAccountDetails{
    accountPasswordTextView.setText(new String(account.getPassword()));
}
```
Other limitations

- Scoria supports architectural flaws that deal with structure, rather than behavior (no protocols, no states of objects)
  - Spoofing
  - Tampering
  - Repudiation
  - Information disclosure
  - Denial of service
  - Elevation of privilege
Some related tools

- AST based analysis
  - SecureAssist
  - FindBugs
- Static Analysis
  - Fortify, IBMScan, FlowDroid, Blue Seal
- Reasoning about code architecture
  - Bauhaus
- Query object graph
  - VisualVM
- Monitoring
  - TaintDroid
Conclusion

- Found information disclosure in Android app
- Constraints implementing Java CERT rules for which automatic support is unavailable [Vanciu and Abi-Antoun, ASE’13]
- Future work
  - Compare Scoria to related approaches based on benchmarks
  - Study how security architects use Scoria

```java
@Test
public void checkInfDisclosureIntentPolicy() {
    secGraph.setObjectProperty(TrustLevelType.Low,
        new InstanceOf(Intent.class));
    secGraph.setObjectProperty(IsConfidential.True,
        new IsChildOf(AccountInformation.class, String);
    Property[] snkProps = { TrustLevelType.Low, new IsConfidential.True, Property[] flwProps = { IsConfidential.True });
    if (secGraph.checkFlowIntoSink(snkProps, flwProps)) { Set<IEdge> sEdges = secGraph.getFlowIntoSink;
        scoria.displayWarnings(sEdges);
        Assert.fail("Information disclosure found");
    }
```
Create, manage, and run configurations

Create a configuration that will launch a JUnit test.