Android Security
A Software Architectural Perspective

Sam Malek
Smartphones have fundamentally changed computing

Number of Global Users (Millions)

Source: Morgan Stanley Research
Smartphones have fundamentally changed computing

Internet Usage Worldwide
October 2009 – October 2016

Source: StatCounter Global Stats
Changes go beyond computing
Changes go beyond computing
Changes go beyond computing
Mobile devices were not always so exciting

- Highly rigid systems
- Software “hacked” in low-level languages
- Device shipped with a predefined set of apps
- Not much customization possible
Modern mobile computing drivers

App Ecosystem

Developer Productivity

Interoperability

Resource constraints

Security & Privacy

Role of architecture in mobile computing

Domain-specific software architecture

- App Ecosystem
- Developer Productivity
- Resource constraints
- Interoperability
- Security & Privacy

Architecture-based development in Android

- **Component types**
  - Activity, Service, Content Provider, and Broadcast Receiver

- **Connector types**
  - Messages-based explicit and implicit, RPC, Data Access, etc.

- **Events**
  - Intent messages
Manifest file specifies an app’s architecture

```xml
<uses-permission android:name="READ_CONTACTS"/>
<uses-permission android:name="INTERNET"/>
<uses-permission android:name="WRITE_EXTERNAL_STORAGE"/>

<activity
  android:name=".activity.MessageCompose"
  android:configChanges="locale"
  android:enabled="false"
  android:label="@string/app_name">
  <intent-filter>
    <action android:name="android.intent.action.SENDTO"/>
    <data android:scheme="mailto"/>
    <category android:name="android.intent.category.DEFAULT"/>
  </intent-filter>
</activity>

<service
  android:name=".service.RemoteControlService"
  android:enabled="true"
  android:permission="com.fsck.k9.permission.REMOTE_CONTROL"/>
```

- **App’s permissions**
- **Specification of an Activity component**
- **Specification of a Service component**
Elegant architectural solutions

- Component life cycle
  - Allows the framework to offload the components in a seamless fashion
K-9 mail client app
Recovered architecture of K-9 mail client
Android: an architecture-based development success story

Number of apps in Google Play store

Source: Statista
But facing increasing security problems

Source: NOKIA Threat Intelligence Report
Architectural missteps are at fault
Overprivileged resource access

<<Android system>>

App A

Comp 1

Comp 2

Comp 3

App B

Comp 1

Comp 2

Comp 3
Underspecified architecture

A provided interface is by default public

Ability to define the provided interfaces, but not the required interfaces
Overprivileged inter-component communication

<<Android system>>

App A

Comp 1

Comp 2

Comp 3

---

App B

Comp 1

Comp 2

Comp 3
Inter-component communication attacks

ICC Attacks

- App Collusion
- Unauthorized Intent Receipt
  - Broadcast Theft
  - Activity Hijacking
  - Service Hijacking
  - Pending Intents
    - Privilege Escalation
    - Malicious Broadcast Injection
    - Malicious Activity Launch
    - Malicious Service Launch
    - Fragment Injection

Intent Spoofing

Inter-component communication attacks

- ICC Attacks
  - Unauthorized Intent Receipt
    - Broadcast Theft
    - Activity Hijacking
    - Service Hijacking
    - Pending Intents
      - Privilege Escalation
      - Malicious Broadcast Injection
      - Malicious Activity Launch
      - Malicious Service Launch
      - Fragment Injection
  - Intent Spoofing

App collusion attack
Inter-component communication attacks

ICC Attacks

- App Collusion
- Unauthorized Intent Receipt
- Intent Spoofing
  - Broadcast Theft
  - Activity Hijacking
  - Service Hijacking
  - Pending Intents
    - Privilege Escalation
    - Malicious Broadcast Injection
    - Malicious Activity Launch
    - Malicious Service Launch
    - Fragment Injection

PhoneActivity defines a public interface (Intent Filter), but fails to check if the caller has the proper permission.

Developer needs to remember to check the caller’s permission as follows:

```java
// receives the Intent
if (checkCallingPermission("permission.CALL_PHONE") == PackageManager.PERMISSION_GRANTED) {
  // makes a sensitive API call
}
```
How can we solve the security problems?

1. Accept Android “as is”
   - Develop tools to detect the security issues

2. Change Android
   - Develop mechanisms to prevent the security issues
Accept Android “as is”
Naïve approach

Scalability issues as the number of apps increases

Every time an app is updated or removed, or a new one installed, the entire process has to repeat
Requirements and Insights

• The analysis needs to be both scalable and compositional
  – Analyze each app in isolation, yet be able to reason about the security posture of the entire system

• Insight: security vulnerabilities are architectural in nature
  – Lift the analysis to the granularity of software architecture
COVERT

Compositional Analysis of Inter-app Vulnerabilities

Subset of Android specification in Alloy

- Formally **codifies Android’s architectural styles**
  - Signatures represent the elements
  - Fields represent the relations
  - Facts represent the constraints

```alloy
module androidDeclaration

abstract sig Application{
    usesPermissions: set Permission,
    appPermissions: set Permission
}

abstract sig Component{
    app: one Application,
    intentFilters: set IntentFilter,
    permissions: set Permission,
    paths: set Path
}

abstract sig Intent{
    sender: one Component,
    component: lone Component,
    action: lone Action,
    categories: set Category,
    data: set Data,
}

abstract sig IntentFilter{
    actions: some Action,
    data: set Data,
    categories: set Category,
}

fact IntentFilterConstraints{
    all i:IntentFilter | one i.^intentFilters
    no i:IntentFilter | i.^intentFilters in Provider
}
```
COVERT
Compositional Analysis of Inter-app Vulnerabilities

Static extraction of architecture

1. Architectural elements and properties defined in the manifest file
Static extraction of architecture

1. Architectural elements and properties defined in the manifest file
2. Architectural elements (e.g., Intent and Filters) that are latent in code
Static extraction of architecture

1. Architectural elements and properties defined in the manifest file
2. Architectural elements (e.g., Intent and Filters) that are latent in code
3. Event-driven behavior of each app
Static extraction of architecture

1. Architectural elements and properties defined in the manifest file
2. Architectural elements (e.g., Intent and Filters) that are latent in code
3. Event-driven behavior of each app
4. Sensitive paths
Specifications of architecture in Alloy

Each app’s architecture is specified declaratively, independent of other apps
COVERT

Compositional Analysis of Inter-app Vulnerabilities

Specification of privilege escalation in Alloy

```
assert privilegeEscalation{
    no disj src, dst: Component, i:Intent |
    (src in i.sender) &&
    (dst in intent Resolver[i]) && some dst.paths &&
    (some p: dst.app.usesPermissions |
        not (p in src.app.usesPermissions) &&
        not ((p in dst.permissions) ||(p in dst.app.
            appPermissions)))
}
```

An assertion states a security property that is checked in the extracted specifications
COVERT
Compositional Analysis of Inter-app Vulnerabilities
Given Android specification $S$, app specifications $M$, and vulnerability assertion $P$, assert whether $M$ does not satisfy $P$ under $S$.
Model checker finds the ICC vulnerabilities

... // omitted details of model instances
privilegeEscalation_src={MalApp/CallerActivity}
privilegeEscalation_dst={VicApp/PhoneActivity}
privilegeEscalation_i={intent1}
privilegeEscalation_p={appDeclaration/CALL_PHONE}
Experimental results

• 4,000 Android apps from four repositories
  – **Google Play** (1,000 most popular + 600 random)
  – **F-Droid** (1,100 apps)
  – **Malgenome** (1,200 random)
  – **Bazaar** (100 most popular)

• Partitioned into 80 non-overlapping bundles, each comprising 50 apps

• Total number of detected vulnerabilities: 385
  – Intent hijack: 97
  – Activity/Service launch: 124
  – Information leakage: 128
  – Privilege escalation: 36

• Manual analysis revealed 61% true positive rate in real-world apps
Example of a previously unknown vulnerability: service launch

- **Barcode Scanner** app
  - One of its services exposes an unprotected Intent filter
  - Allows a malicious app to make unauthorized payment through SMS
Performance compared to tools ignoring the architectural knowledge
Remaining challenge: hidden code

COVERT does not work if the code is not present
Change Android
What kind of change is acceptable?

Usability

Compatibility with existing apps
What needs to change?

Systematic violation of the **least-privilege principle** in Android is the mother of all evil.

Build an analysis to determine the exact privileges required for each component from its implementation logic.
Resource access privileges

• Determine which permissions are actually used by each component
  – Use a mapping of API calls to permissions

<<Android system>>

App A
- Comp 1
- Comp 2
- Comp 3

App B
- Comp 1
  - sendText()
- Comp 2
- Comp 3
ICC privileges

- Determine the **required** ICCs for each component to run
  - Resolve the Intents and their recipients
DELDroid

Determination and Enforcement of Least-Privilege Architecture in Android

Hidden code

DELDDroid can effectively thwart such attacks
## Attack surface reduction

<table>
<thead>
<tr>
<th>Num of Apps</th>
<th>Num of Comp</th>
<th>Communication Domain</th>
<th>Permission Granted Domain</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>Original</td>
<td>LP</td>
</tr>
<tr>
<td>30</td>
<td>306</td>
<td>29,031</td>
<td>42</td>
</tr>
<tr>
<td>30</td>
<td>432</td>
<td>78,237</td>
<td>625</td>
</tr>
<tr>
<td>30</td>
<td>422</td>
<td>65,709</td>
<td>173</td>
</tr>
<tr>
<td>30</td>
<td>449</td>
<td>80,372</td>
<td>205</td>
</tr>
<tr>
<td>30</td>
<td>353</td>
<td>56,868</td>
<td>345</td>
</tr>
<tr>
<td>30</td>
<td>541</td>
<td>85,556</td>
<td>661</td>
</tr>
<tr>
<td>30</td>
<td>562</td>
<td>82,863</td>
<td>137</td>
</tr>
<tr>
<td>30</td>
<td>362</td>
<td>50,208</td>
<td>250</td>
</tr>
<tr>
<td>30</td>
<td>265</td>
<td>25,817</td>
<td>129</td>
</tr>
<tr>
<td>30</td>
<td>421</td>
<td>50,001</td>
<td>74</td>
</tr>
<tr>
<td>Average</td>
<td>411.3</td>
<td>60,466.2</td>
<td>264.1</td>
</tr>
<tr>
<td>Avg. (per app)</td>
<td>13.7</td>
<td>2,015.5</td>
<td>8.8</td>
</tr>
</tbody>
</table>

10 experiments with 30 randomly selected apps
## Attack surface reduction – ICC

<table>
<thead>
<tr>
<th>Num of Apps</th>
<th>Num of Comps</th>
<th>Communication Domain</th>
<th>Permission Granted Domain</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>Original</td>
<td>LP</td>
</tr>
<tr>
<td>30</td>
<td>306</td>
<td>29,031</td>
<td>42</td>
</tr>
<tr>
<td>30</td>
<td>432</td>
<td>78,237</td>
<td>625</td>
</tr>
<tr>
<td>30</td>
<td>422</td>
<td>65,709</td>
<td>173</td>
</tr>
<tr>
<td>30</td>
<td>449</td>
<td>80,372</td>
<td>205</td>
</tr>
<tr>
<td>30</td>
<td>353</td>
<td>56,868</td>
<td>345</td>
</tr>
<tr>
<td>30</td>
<td>541</td>
<td>85,556</td>
<td>661</td>
</tr>
<tr>
<td>30</td>
<td>562</td>
<td>82,863</td>
<td>137</td>
</tr>
<tr>
<td>30</td>
<td>362</td>
<td>50,208</td>
<td>250</td>
</tr>
<tr>
<td>30</td>
<td>265</td>
<td>25,817</td>
<td>129</td>
</tr>
<tr>
<td>30</td>
<td>421</td>
<td>50,001</td>
<td>74</td>
</tr>
<tr>
<td>Average</td>
<td>411.3</td>
<td>60,466.2</td>
<td>264.1</td>
</tr>
<tr>
<td>Avg. (per app)</td>
<td>13.7</td>
<td>2,015.5</td>
<td>8.8</td>
</tr>
</tbody>
</table>

Over **99%** reduction in ICC privileges
### Attack surface reduction – resource access

<table>
<thead>
<tr>
<th>Num of Apps</th>
<th>Num of Comps</th>
<th>Communication Domain</th>
<th>Permission Granted Domain</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Original LP</td>
<td>Reduction (%)</td>
<td>Original LP</td>
</tr>
<tr>
<td>30</td>
<td>306</td>
<td>29,031</td>
<td>42</td>
</tr>
<tr>
<td></td>
<td>78,237</td>
<td>625</td>
<td>99.20</td>
</tr>
<tr>
<td></td>
<td>65,709</td>
<td>173</td>
<td>99.74</td>
</tr>
<tr>
<td></td>
<td>80,372</td>
<td>205</td>
<td>99.74</td>
</tr>
<tr>
<td></td>
<td>56,868</td>
<td>345</td>
<td>99.39</td>
</tr>
<tr>
<td></td>
<td>85,556</td>
<td>661</td>
<td>99.23</td>
</tr>
<tr>
<td></td>
<td>82,863</td>
<td>137</td>
<td>99.83</td>
</tr>
<tr>
<td></td>
<td>50,208</td>
<td>250</td>
<td>99.50</td>
</tr>
<tr>
<td></td>
<td>25,817</td>
<td>129</td>
<td>99.50</td>
</tr>
<tr>
<td></td>
<td>50,001</td>
<td>74</td>
<td>99.85</td>
</tr>
<tr>
<td>Average</td>
<td>411.3</td>
<td>264.1</td>
<td>99.58</td>
</tr>
<tr>
<td>Avg. (per app)</td>
<td>13.7</td>
<td>8.8</td>
<td>99.56</td>
</tr>
<tr>
<td></td>
<td>2,015.5</td>
<td>2.0</td>
<td>97.54</td>
</tr>
</tbody>
</table>

Over **97%** reduction in resource access privileges
How effective is attack surface reduction in preventing attacks?

- 54 malicious and vulnerable apps
  - The steps and inputs required to create the attacks are known

- The dataset contains
  - 18 privilege escalation attacks
  - 24 hidden ICC attacks through dynamic class loading
How effective is attack surface reduction in preventing attacks?

18 privilege escalation
24 hidden ICC attacks
42 attacks

- TP: Malicious behavior prevented (42)
- FP: Benign behavior prevented (1)
- FN: Malicious behavior allowed (0)

Precision = 97.76%
Recall = 100%

58
Recap

Role of architecture in mobile computing

Domain-specific software architecture

Interoperability

Security & Privacy

Resource constraints

Developer Productivity

Interoperability

App Ecosystem


Inter-component communication attacks

ICC Attacks

App Collusion

Unauthorized Intent Receipt

Intent Spoofing

Broadcast Theft

Activity Hijacking

Service Hijacking

Pending Intents

Privilege Escalation

Malicious Broadcast Injection

Malicious Activity Launch

Malicious Service Launch

Fragment Injection


COVERT

Compositional Analysis of Inter-app Vulnerabilities


DELDroid

Determination and Enforcement of Least-Privilege Architecture in Android

Broader takeaways

• Designing a new framework
  – Paramount to get it right the first time

• Think twice before choosing a framework
  – Determines the architecture of your application

• Frameworks + app ecosystems + program analysis
  – New opportunities to study architectural phenomena in action and at scale
Acknowledgement

Students and Collaborators

- Hamid Bagheri, UNL
- Joshua Garcia, UCI
- Alireza Sadeghi, UCI
- Mahmoud Hammad, UCI
- Nenad Medvidovic, USC
- David Garlan, CMU
- Daniel Jackson, MIT
- Bradley Schmerl, CMU
- Eunsuk Kang, UC Berkeley

Sponsors

- NSF
- DARPA
- NSWC
- CBIC