The Threat in Your Pocket:
Trends, Challenges, and Solutions in Mobile Application Security

Sam Malek
http://malek.ics.uci.edu
malek@uci.edu

ACM OC Chapter, May 20, 2020
Smartphones have fundamentally changed computing

Number of Global Users (Millions)

Source: Morgan Stanley Research
Smartphones have fundamentally changed computing

Source: StatCounter Global Stats
Explosive growth in mobile apps

Number of apps in Google Play store

Source:
Android; Google; App Annie; AppBrain
© Statista 2017

Additional Information:
Worldwide; Google; Android; App Annie; December 2009 to June 2017
Mobile app revenues

<table>
<thead>
<tr>
<th>Year</th>
<th>Revenue in billion U.S. dollars</th>
</tr>
</thead>
<tbody>
<tr>
<td>2014</td>
<td>97.7</td>
</tr>
<tr>
<td>2015</td>
<td>152</td>
</tr>
<tr>
<td>2016</td>
<td>218.2</td>
</tr>
<tr>
<td>2017</td>
<td>285.3</td>
</tr>
<tr>
<td>2018</td>
<td>365.2</td>
</tr>
<tr>
<td>2019*</td>
<td>461.7</td>
</tr>
<tr>
<td>2020*</td>
<td>581.9</td>
</tr>
<tr>
<td>2021*</td>
<td>693</td>
</tr>
<tr>
<td>2022*</td>
<td>808.7</td>
</tr>
</tbody>
</table>

Source: statista
Increasing security problems

Development of Android malware

- Total number of malware
- New malware

Source:

AVTEST
Increasing security problems

Source:

Android malware samples

[Graph showing the increase in Android malware samples from July 2012 to January 2019]
Android is the primary target

Market Distribution

Attack Distribution

Source: Kaspersky
Missteps in Android development framework are at fault
Getting the framework design “right” is exceptionally difficult
Architecture-based development in Android

• Component types
  – Activity
  – Service
  – Content Provider
  – Broadcast Receiver

• Events
  – Intent messages
Architecture-based development in Android

- Component types
  - Activity
  - Service
  - Content Provider
  - Broadcast Receiver

- Events
  - Intent messages
Principle of Least Privilege

• Each software component must be able to access only the information and resources that are necessary for its legitimate purpose

• Android systematically violates this principle
Overprivileged resource access

<<Android system>>

App A

Comp 1

Comp 2

Comp 3

App B

Comp 1

Comp 2

Comp 3
Overprivileged resource access

<<Android system>>

App A

Comp 1

Comp 2

Comp 3

App B

Comp 1

Comp 2

Comp 3
Overprivileged resource access

<<Android system>>

App A

- Comp 1
- Comp 2
- Comp 3

App B

- Comp 1
- Comp 2
- Comp 3
Overprivileged inter-component communication (ICC)

<<Android system>>
Overprivileged inter-component communication (ICC)
Overprivileged inter-component communication (ICC)
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

Inter-component communication attacks

App Collusion

Unauthorized Intent Receipt

Broadcast Theft
Activity Hijacking
Service Hijacking
Pending Intents

Intent Spoofing

Privilege Escalation
Malicious Broadcast Injection
Malicious Activity Launch
Malicious Service Launch
Fragment Injection

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

Privilege escalation attack

- **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
Naïve approach

Scalability issues as the number of apps increases

Combined Apps → Program Analysis

Vulnerability Report

App1: Malicious
App2: Vulnerable

App1
App2
App3
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

Vulnerability Report

App1: Malicious

App2: Vulnerable
Requirements and Insights
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

COVERT
Compositional Analysis of Inter-app Vulnerabilities

Subset of Android specification in Alloy

- Formally **codifies** Android’s architectural constructs
  - Signatures represent the constructs
  - 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

![Diagram showing MalApp, CallerActivity, VicApp, PhoneActivity, and permissions]
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
Specification of architecture in Alloy

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

Compositional Analysis of Inter-app Vulnerabilities

COVERT

Compositional Analysis of Inter-app Vulnerabilities

An assertion states a security property that is checked in the extracted specifications
COVERT
Compositional Analysis of Inter-app Vulnerabilities

COVERT
Compositional Analysis of Inter-app Vulnerabilities

Formulation as a **model checking** problem

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
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
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
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

Graph showing the average analysis time (seconds) against the number of components for different tools: DidFail, COVERT, and AmAnDroid.
Remaining challenge: hidden code

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

Usability
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.
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
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 (getGPS())
  - Comp 3

- App B
  - Comp 1 (sendText())
  - Comp 2
  - Comp 3
Resource access privileges

- Determine which permissions are actually used by each component
  - Use a mapping of API calls to permissions
ICC privileges

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

DELDroid

**Determination and Enforcement of Least-Privilege Architecture in Android**

Over-privilege architecture
DELDroid: Determination and Enforcement of Least-Privilege Architecture in Android

Over-privilege architecture

Static Analysis

DELDroid: Determination and Enforcement of Least-Privilege Architecture in Android

Over-privilege architecture

Least-privilege architecture

DELDroid

Determination and Enforcement of Least-Privilege Architecture in Android

Over-privilege architecture

Least-privilege architecture

Static Analysis

DELDroid

Determination and Enforcement of Least-Privilege Architecture in Android

Over-privilege architecture

Least-privilege architecture

Extra privileges to revoke

DELDroid

Determination and Enforcement of Least-Privilege Architecture in Android

Over-privilege architecture

Least-privilege architecture

Static Analysis

Android Framework

Privilege Manager Layer

Resource Monitor

ICC Monitor

ECA Rules

Android Runtime

Libraries

Linux Kernel

Custom version of Android

Hidden code

Diagram showing a flow of activities involving Mal App, CallerActivity, Intent, and Vic App. The Mal App initiates an activity that prompts the CallerActivity, which then leads to an Intent to Vic App. The Vic App has a Call Permission while the Mal App does not. The diagram illustrates the interaction and potential security implications of such activities.
DELDroid can effectively thwart such attacks.
## Attack surface reduction

<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></td>
<td>411.3</td>
<td>264.1</td>
</tr>
<tr>
<td>Avg. (per app)</td>
<td></td>
<td>13.7</td>
<td>8.8</td>
</tr>
</tbody>
</table>

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

<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>

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
<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>

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

- 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?

<table>
<thead>
<tr>
<th>Category</th>
<th>Count</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TP</td>
<td>42</td>
<td>Malicious behavior prevented</td>
</tr>
<tr>
<td>FP</td>
<td>1</td>
<td>Benign behavior prevented</td>
</tr>
<tr>
<td>FN</td>
<td>0</td>
<td>Malicious behavior allowed</td>
</tr>
</tbody>
</table>

Precision = 97.76%
Recall = 100%

18 privilege escalation
24 hidden ICC attacks
42 attacks
Recap
Missteps in Android development framework are at fault
Recap

Missteps in Android development framework are at fault

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

Recap

Missteps in Android development framework are at fault

Inter-component communication attacks

COVERT
Compositional Analysis of Inter-app Vulnerabilities

Recap

Missteps in Android development framework are at fault

Inter-component communication attacks

COVERT
Compositional Analysis of Inter-app Vulnerabilities

DELDroid
Determination and Enforcement of Least-Privilege Architecture in Android


Broader takeaways
Broader takeaways

• Designing a new framework is hard
  – Paramount to get it right the first time
• Think twice before choosing a framework
  – Determines the security properties of your application
• Program analysis + software architecture
  – New opportunities to improve security properties of applications
Broader takeaways

• Designing a new framework is hard
  – Paramount to get it right the first time
• Think twice before choosing a framework
  – Determines the security properties of your application
• Program analysis + software architecture
  – New opportunities to improve security properties of applications
Acknowledgement

Students and Collaborators

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

Sponsors

NSF
DARPA
FBI
Homeland Security
Thank you
The Threat in Your Pocket: Trends, Challenges, and Solutions in Mobile Application Security

Sam Malek
http://malek.ics.uci.edu
malek@uci.edu