UCI University of California, Irvine

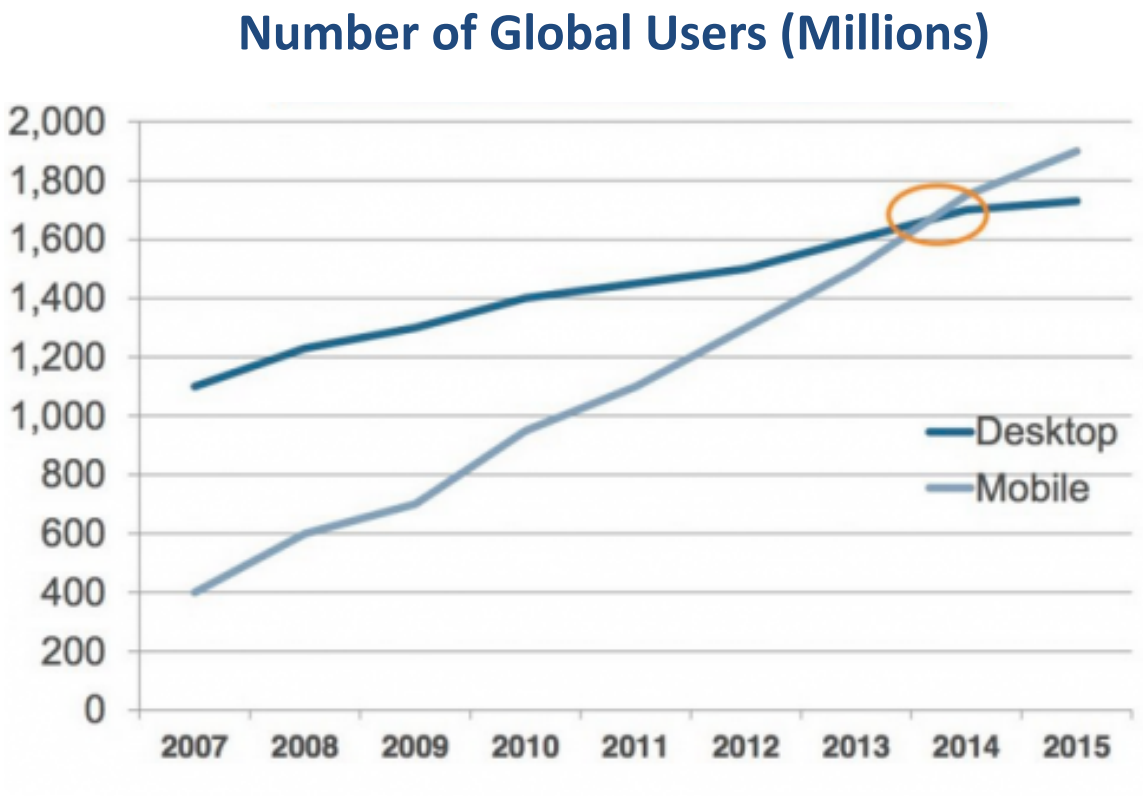
Android Security

A Software Architectural Perspective

Sam Malek

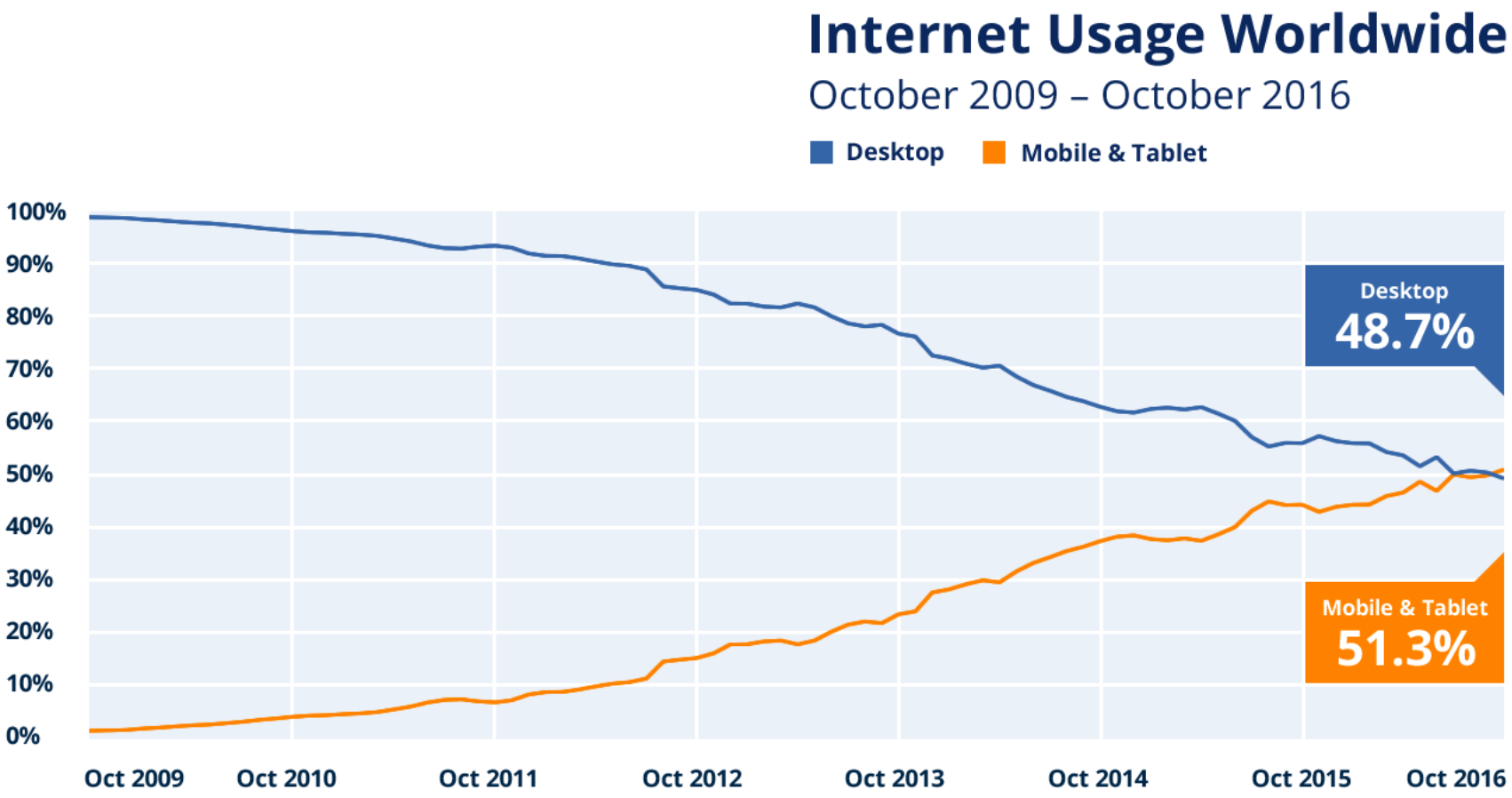
International Conference on Software Architecture (ICSA 2017)

Smartphones have fundamentally changed computing



Source: Morgan Stanley Research

Smartphones have fundamentally changed computing



Source: StatCounter Global Stats

Changes go beyond computing

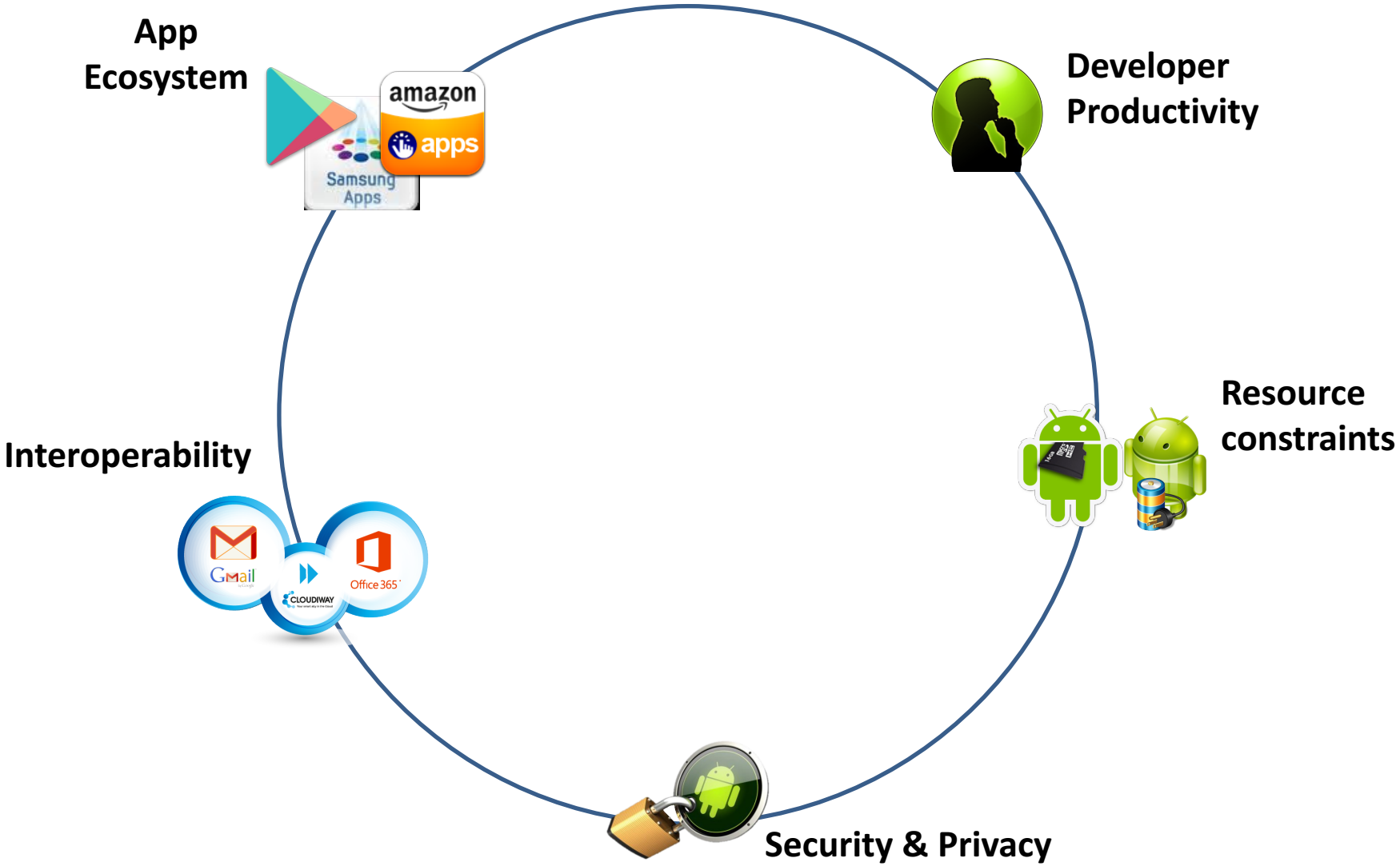
Changes go beyond computing

Changes go beyond computing

Mobile devices were not always so exciting

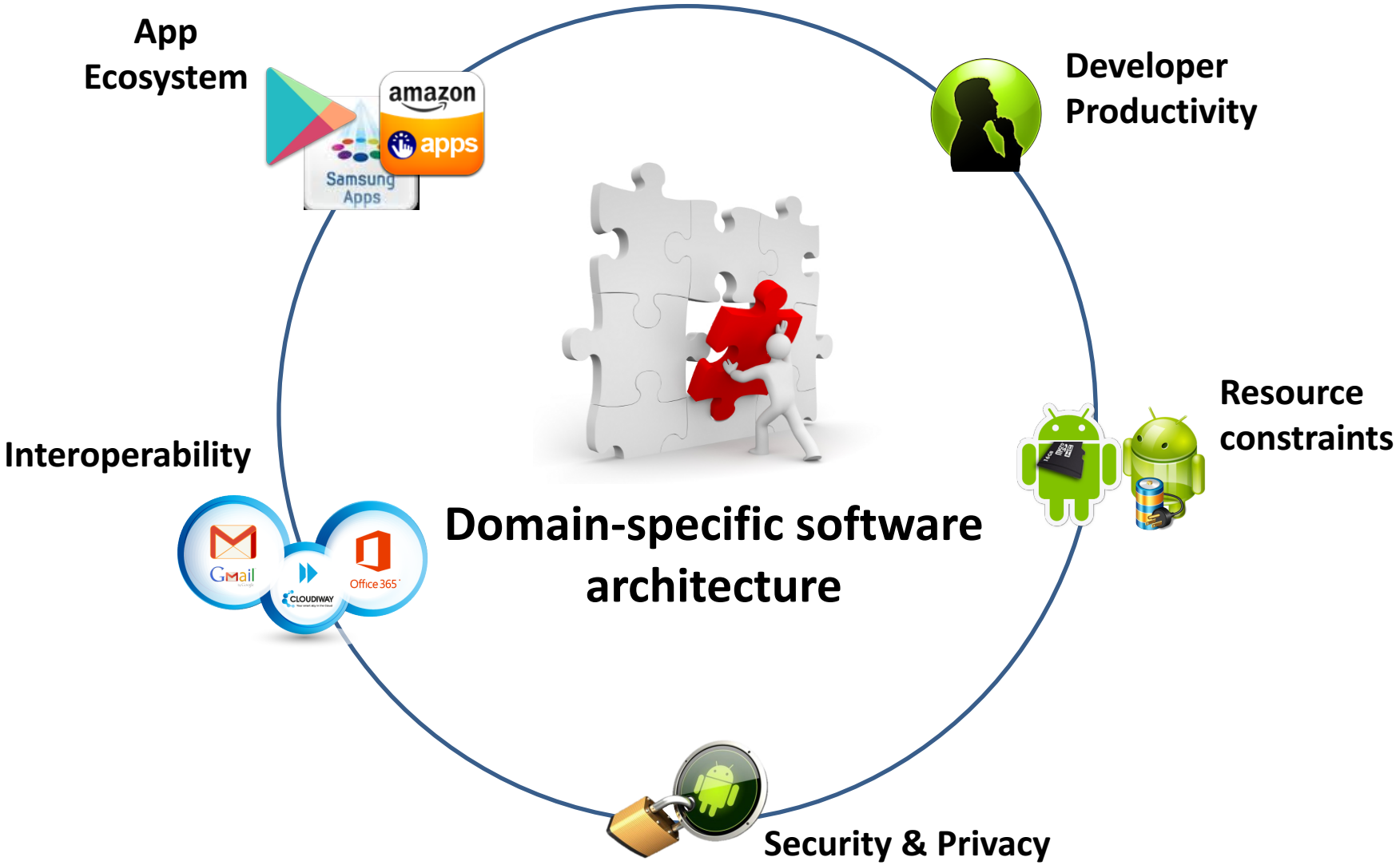
- Highly rigid systems
- Software “hacked” in low-level languages
- Device shipped with a predefined set of apps
- Not much customization possible

Modern mobile computing drivers



Bagheri, Garcia, Sadeghi, Malek, and Medvidovic. Software Architectural Principles in Contemporary Mobile Software: from Conception to Practice. *JSS* 2016.

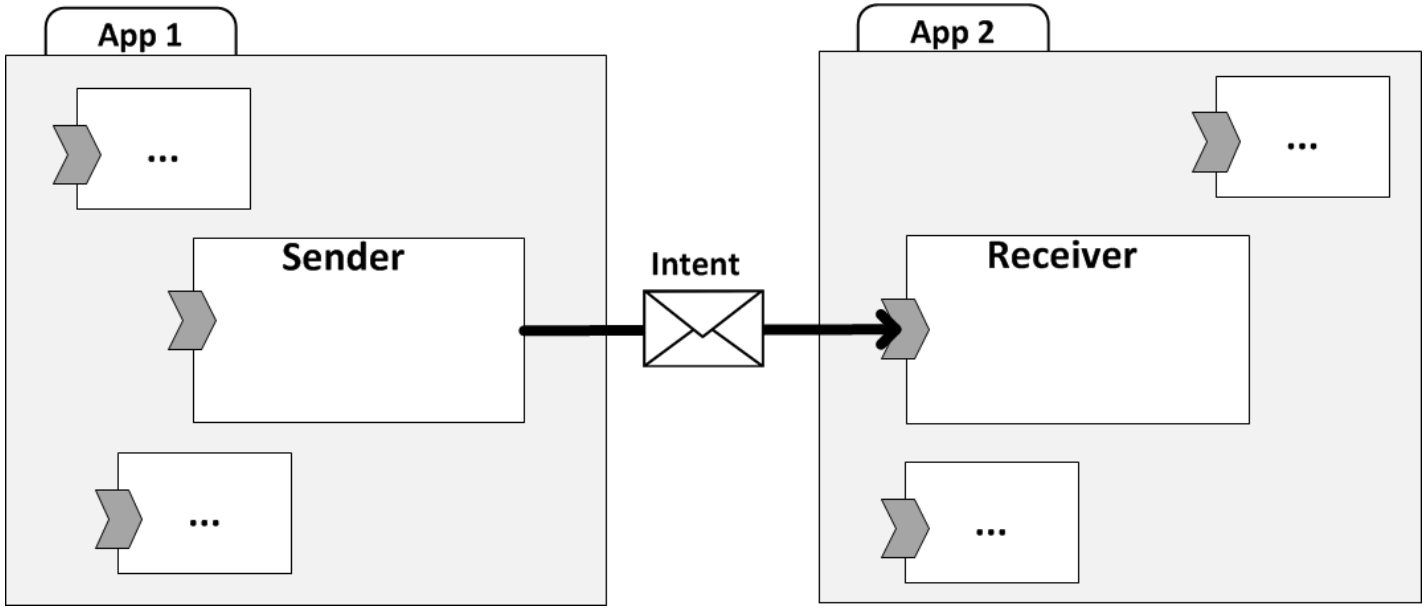
Role of architecture in mobile computing



Bagheri, Garcia, Sadeghi, Malek, and Medvidovic. Software Architectural Principles in Contemporary Mobile Software: from Conception to Practice. *JSS* 2016.

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

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

App's permissions

```
<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>
```

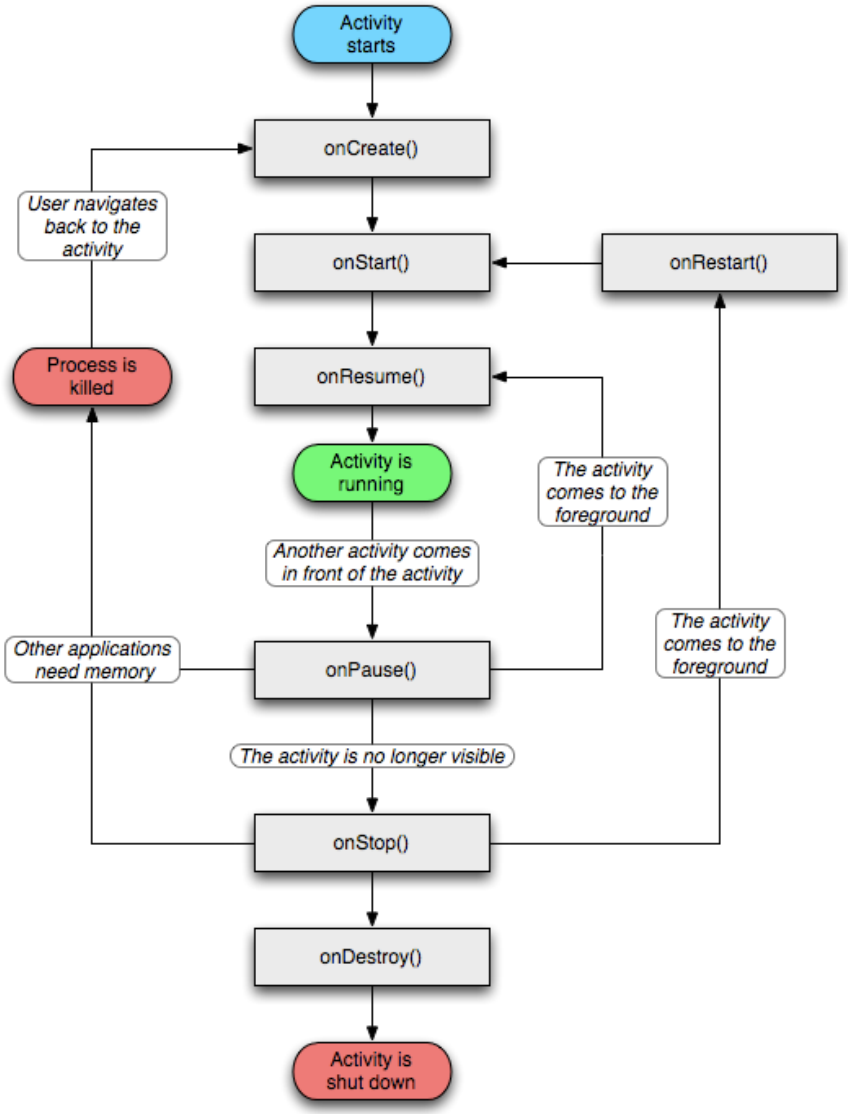
Specification of an Activity component

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

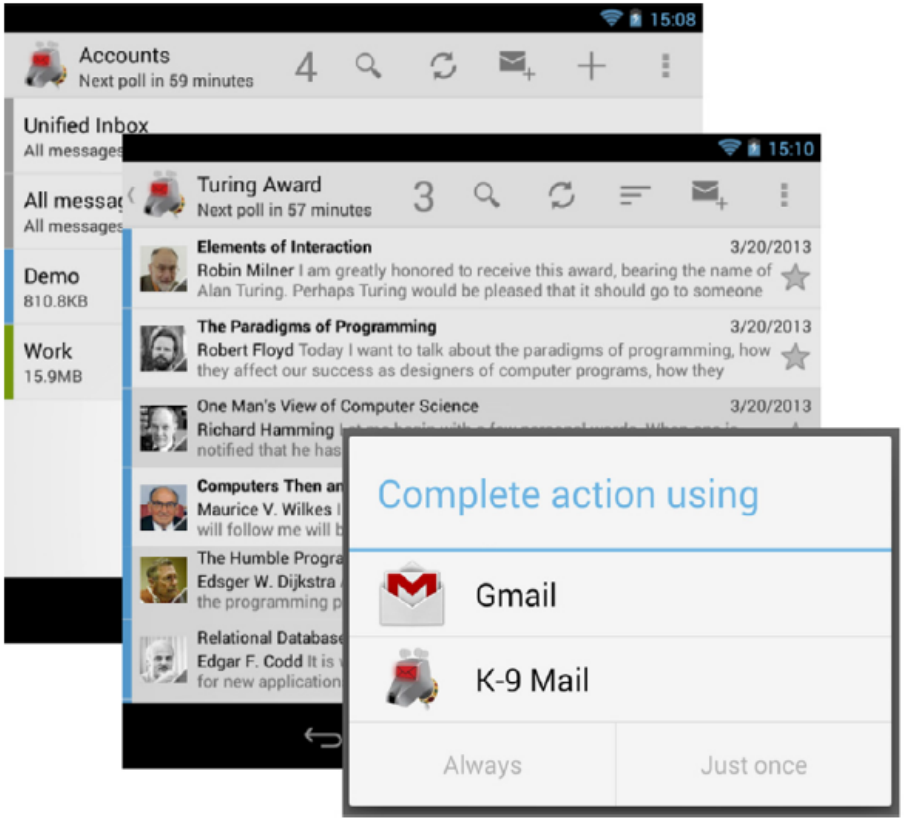
Specification of a Service component

Elegant architectural solutions

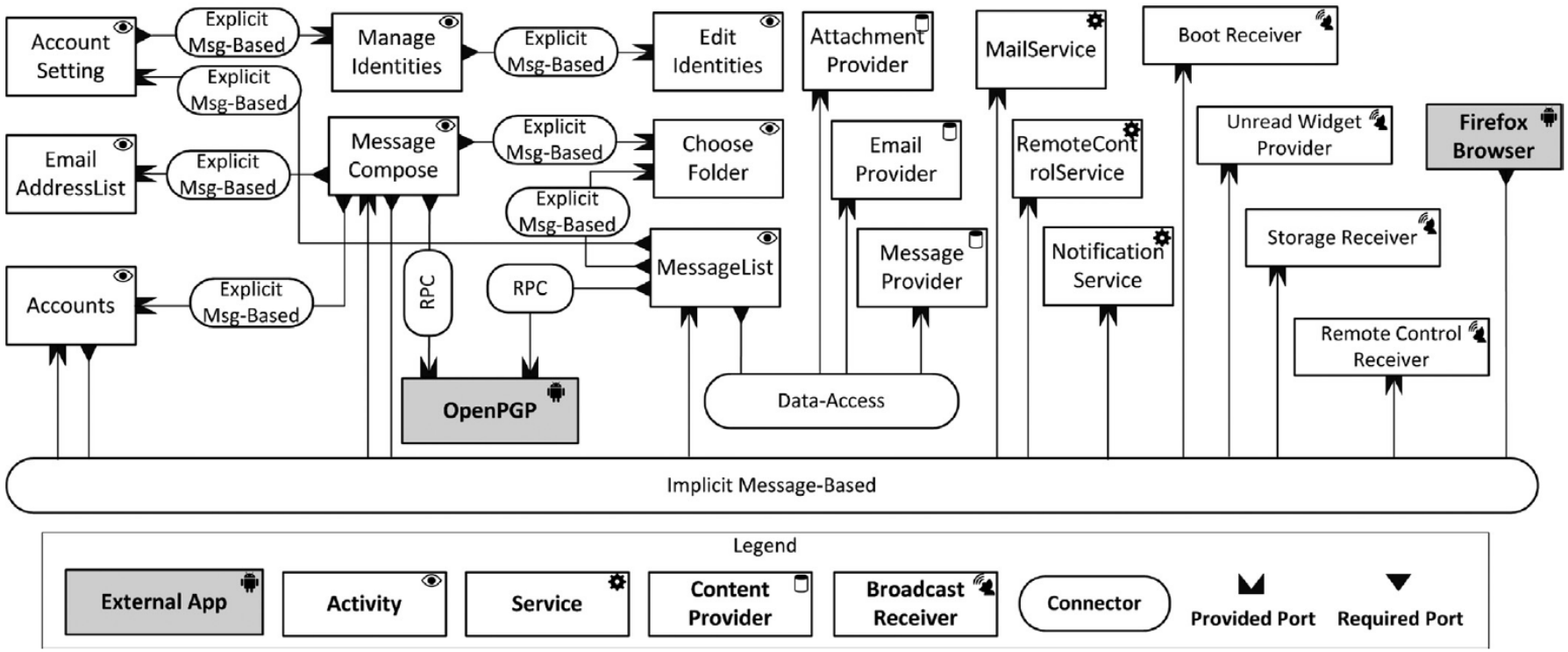
- Component life cycle
  - Allows the framework to offload the components in a seamless fashion



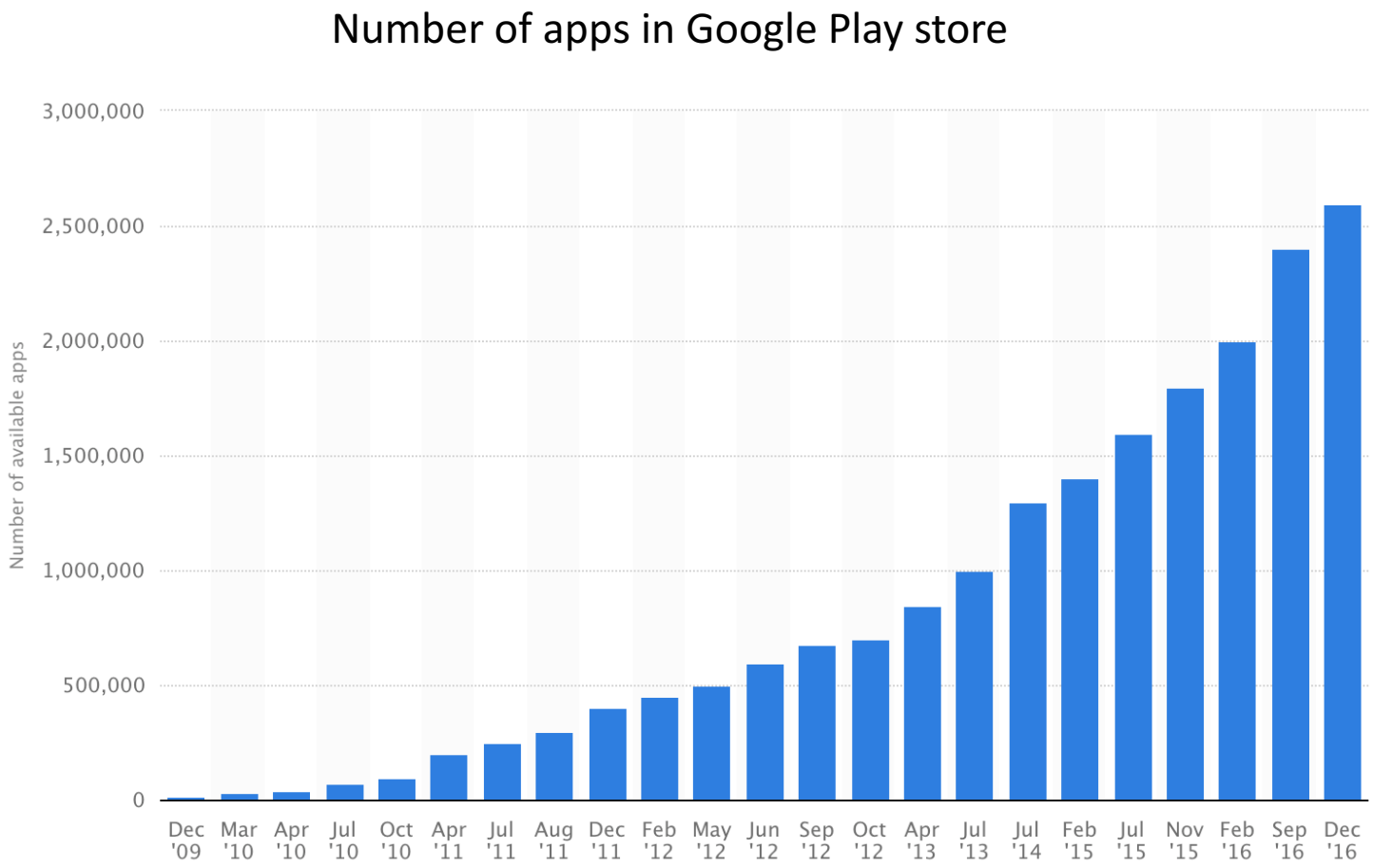
K-9 mail client app



Recovered architecture of K-9 mail client

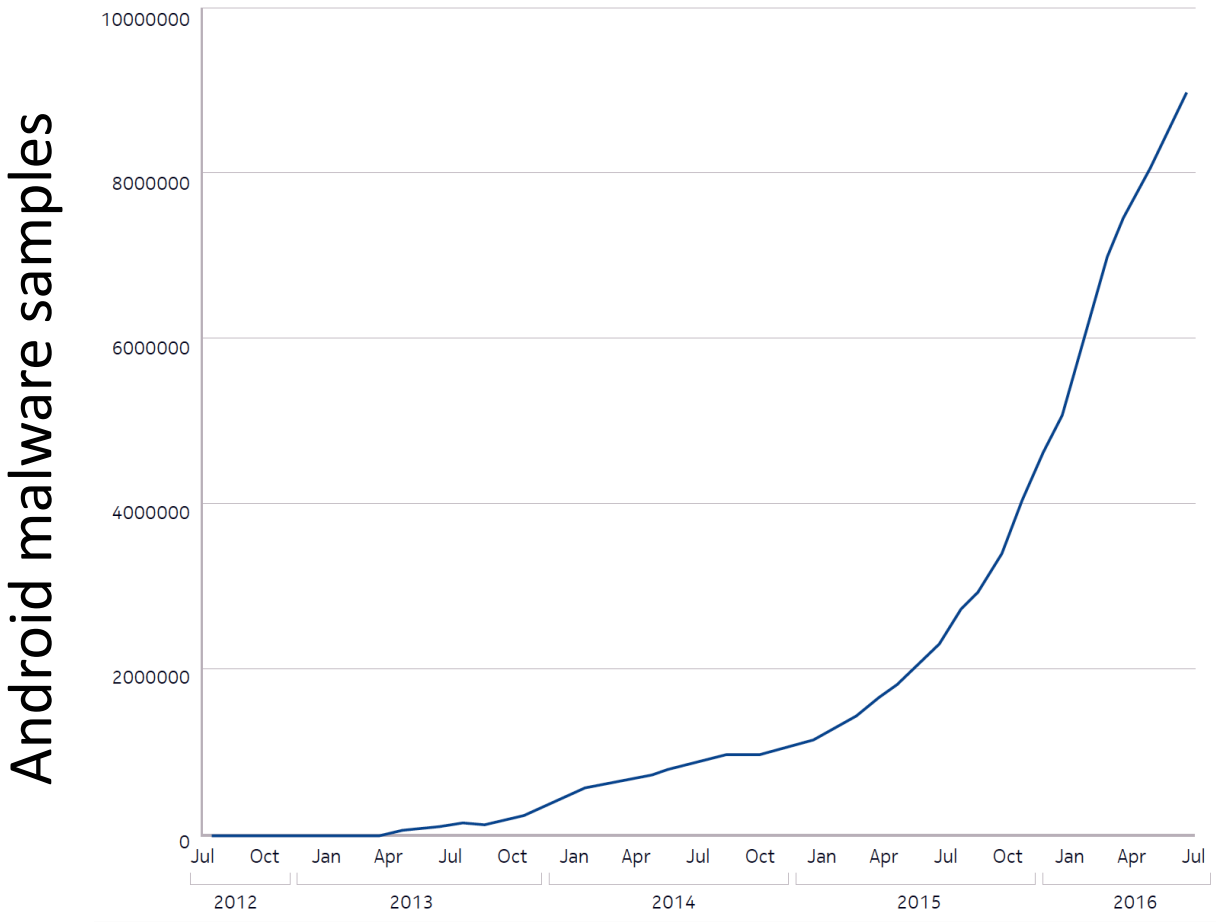


Android: an architecture-based development success story



Source: Statista

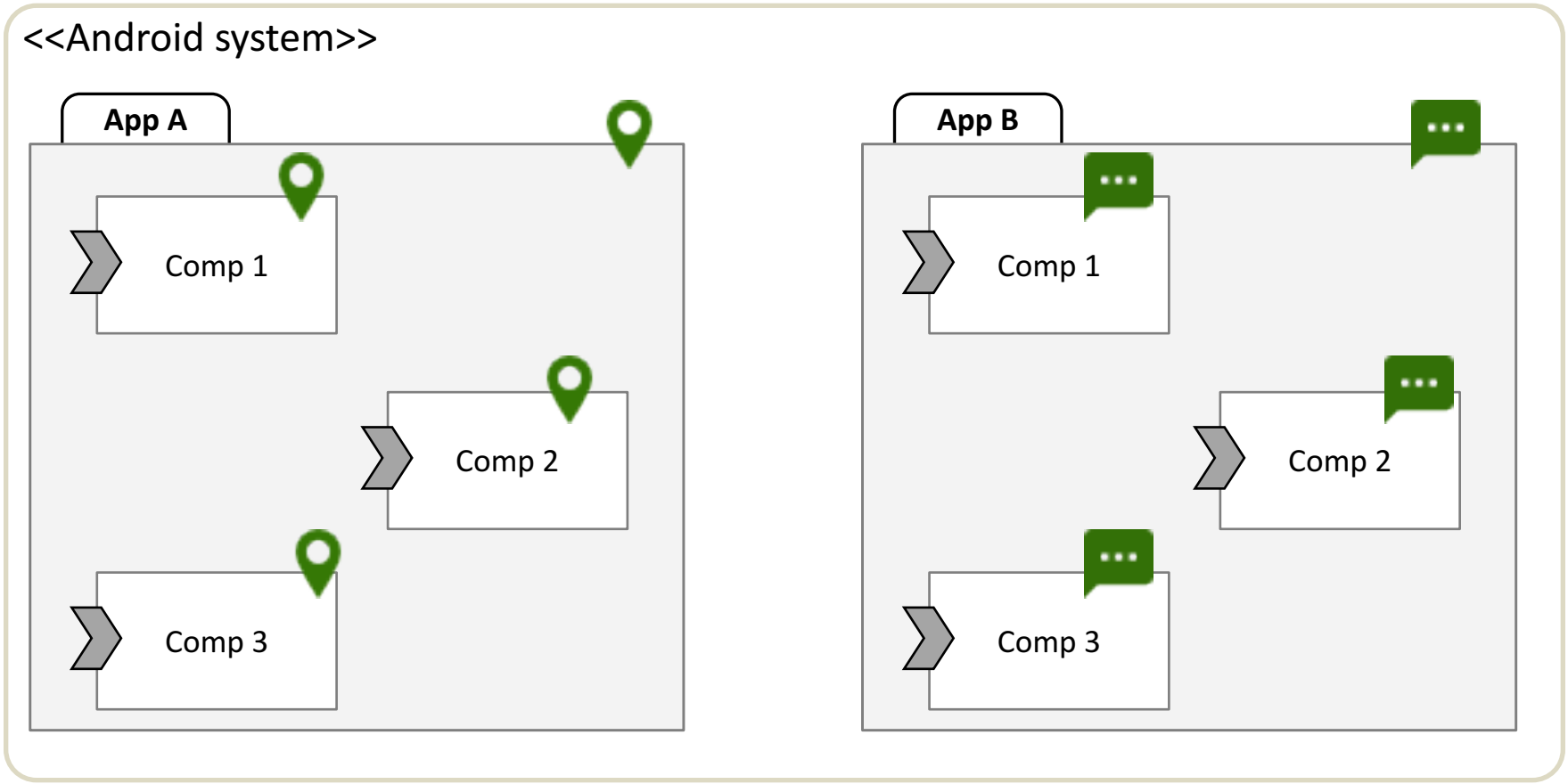
But facing increasing security problems



Source: [NOKIA Threat Intelligence Report]

Architectural missteps are at fault

Overprivileged resource access



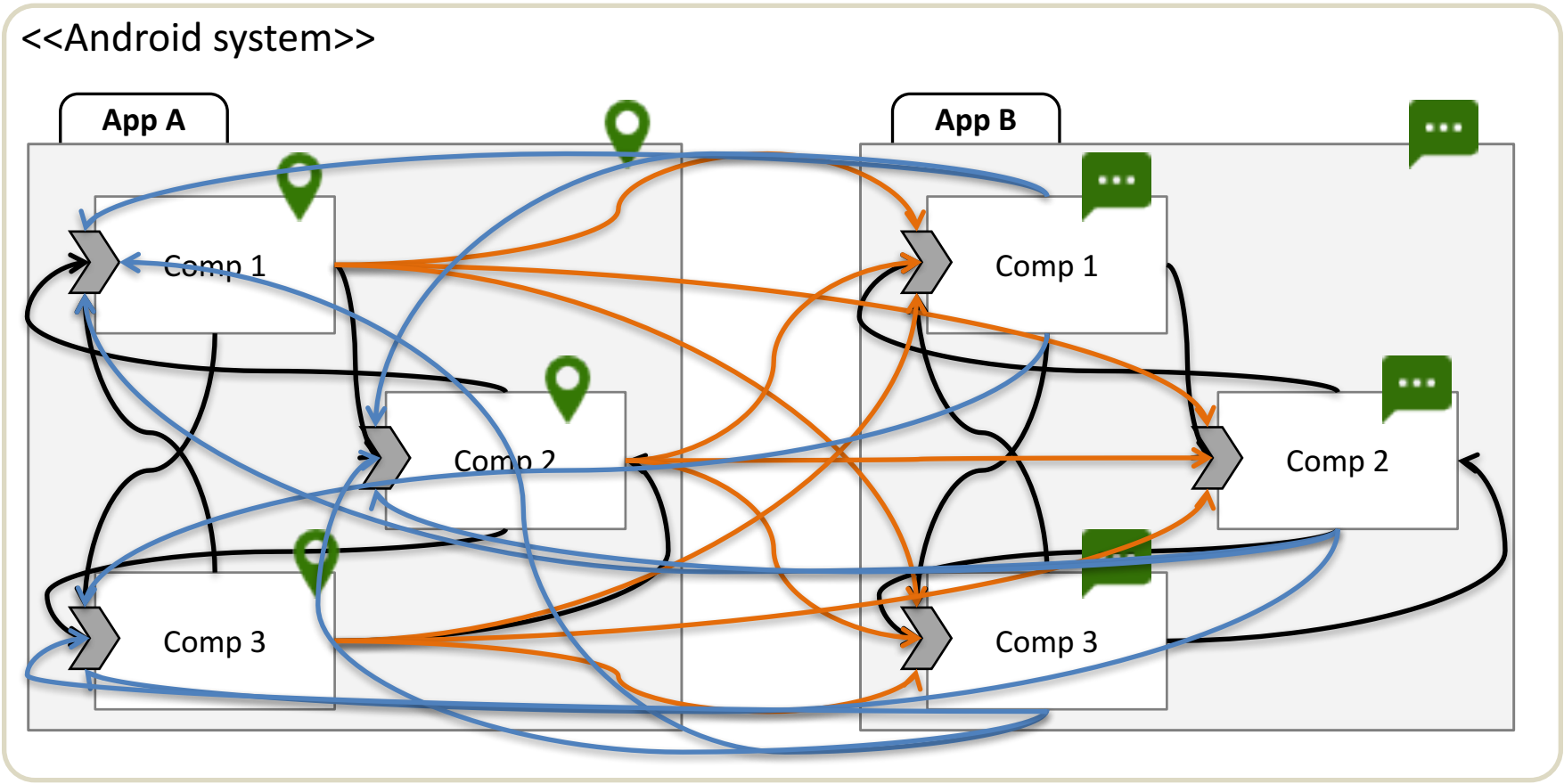
Underspecified architecture

```
<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.R    eControlService"
 android:enabled="true"
 android:permission="co      .k9.permission.REMOTE
/>
```

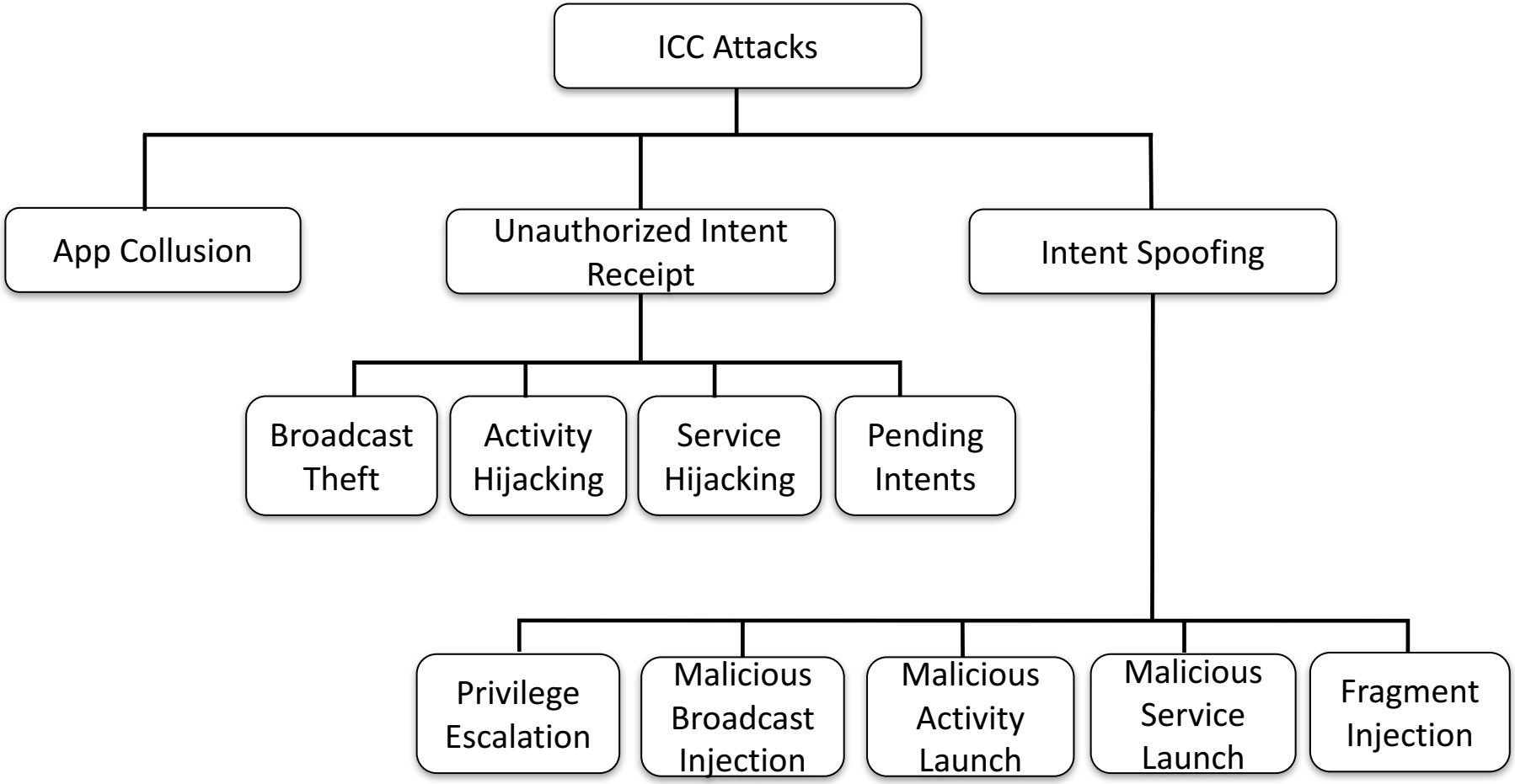
A *provided* interface is by default public

Ability to define the *provided* interfaces, but not the *required* interfaces

Overprivileged inter-component communication

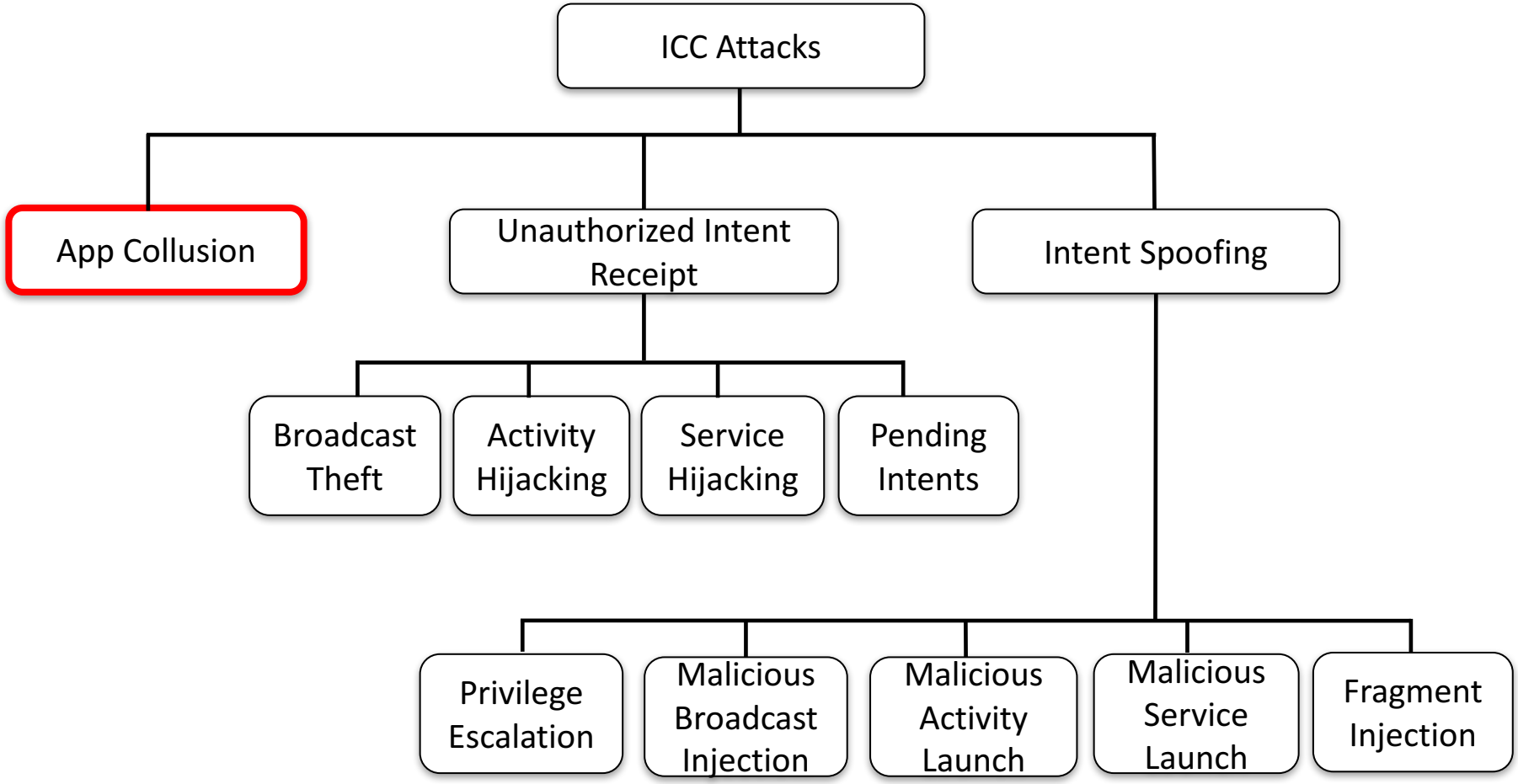


Inter-component communication attacks



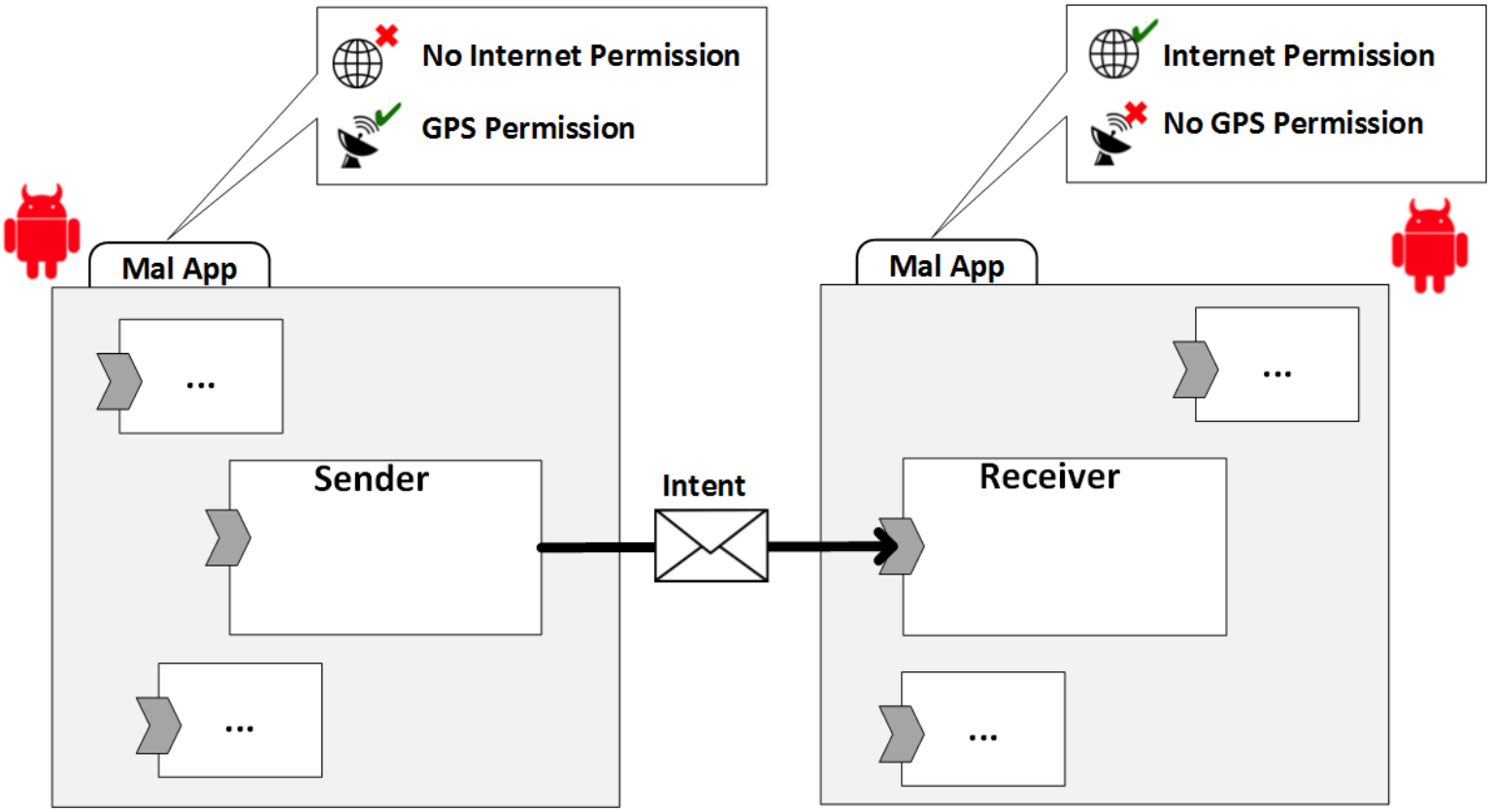
Sadeghi, Bagheri, Garcia, and Malek. A Taxonomy and Qualitative Comparison of Program Analysis Techniques for Security Assessment of Android Software. *TSE* 2017.

Inter-component communication attacks

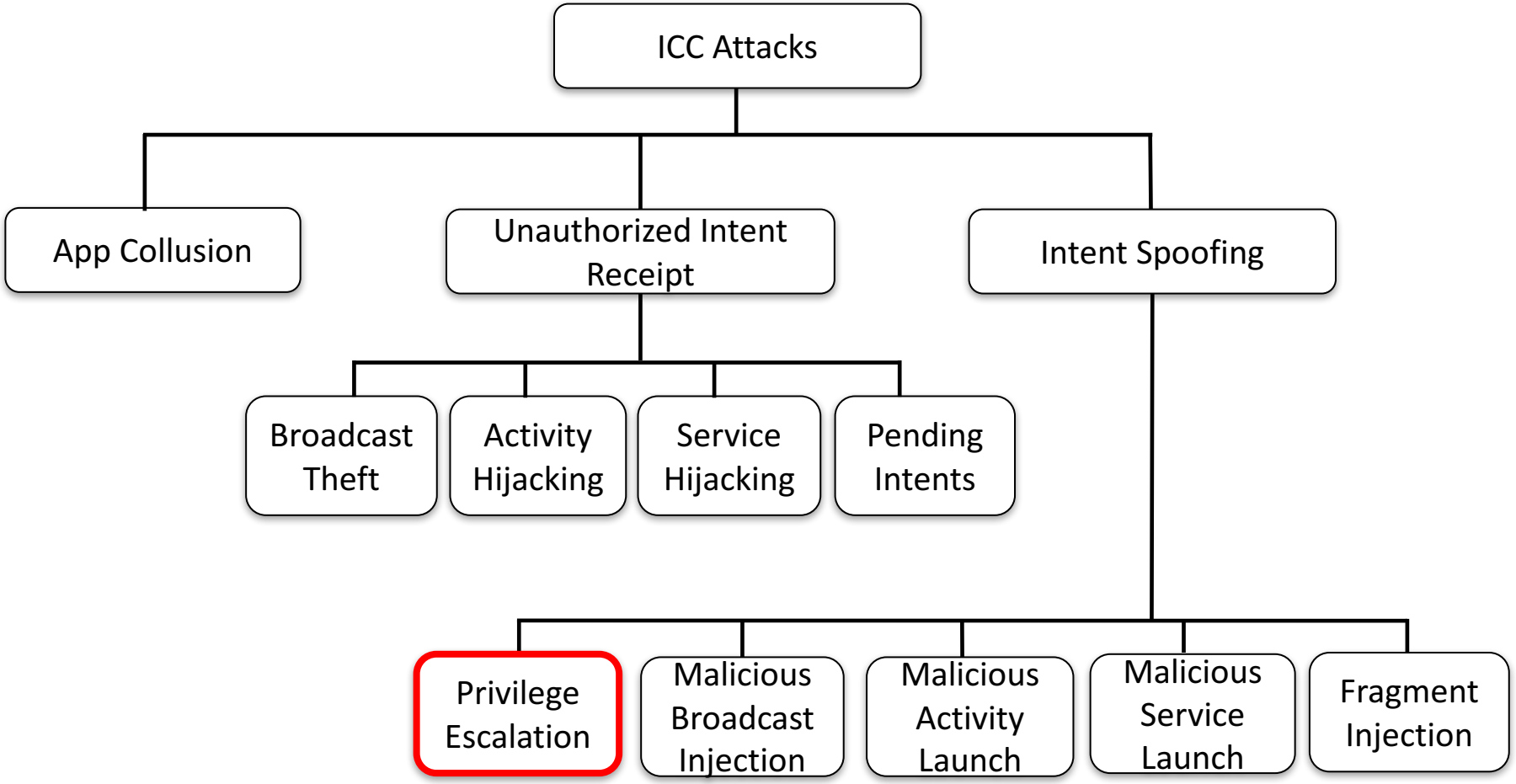


Sadeghi, Bagheri, Garcia, and Malek. A Taxonomy and Qualitative Comparison of Program Analysis Techniques for Security Assessment of Android Software. *TSE* 2017.

App collusion attack

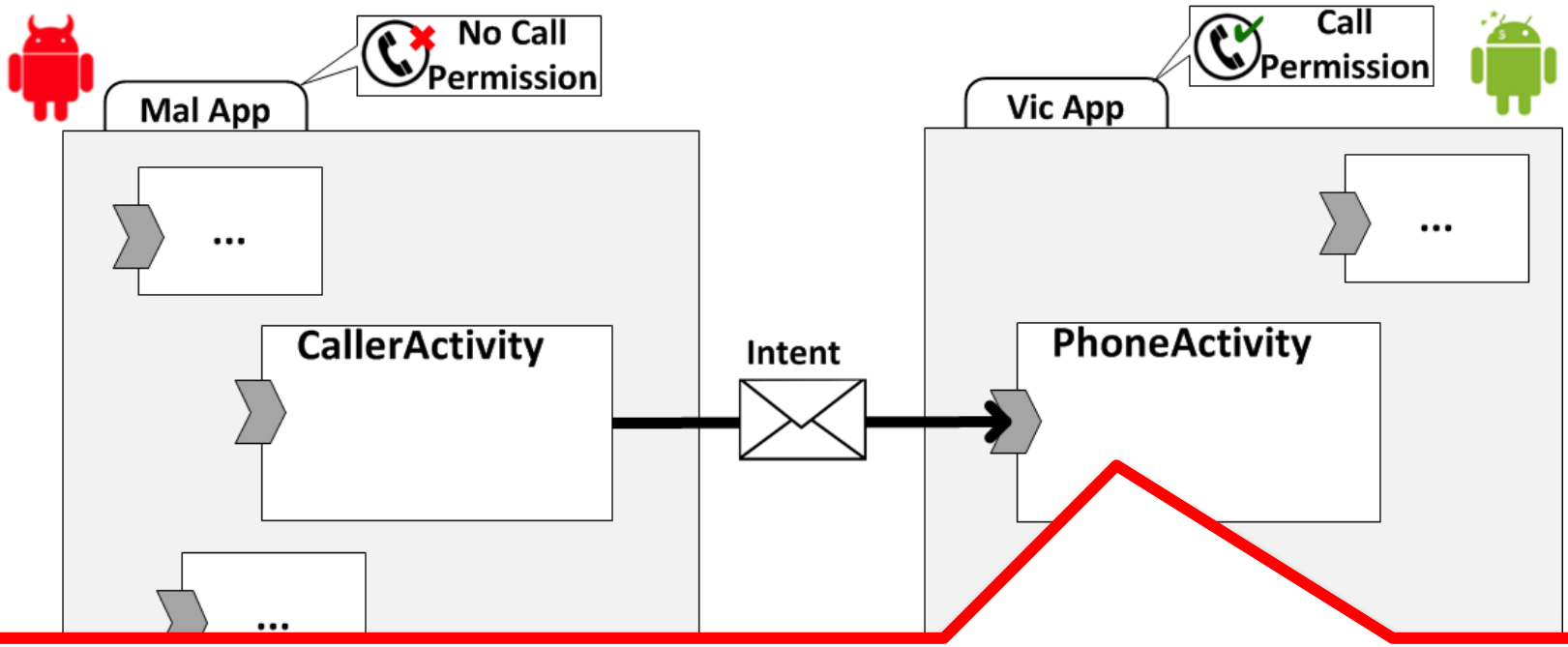


Inter-component communication attacks



Sadeghi, Bagheri, Garcia, and Malek. A Taxonomy and Qualitative Comparison of Program Analysis Techniques for Security Assessment of Android Software. *TSE* 2017.

Privilege escalation attack



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

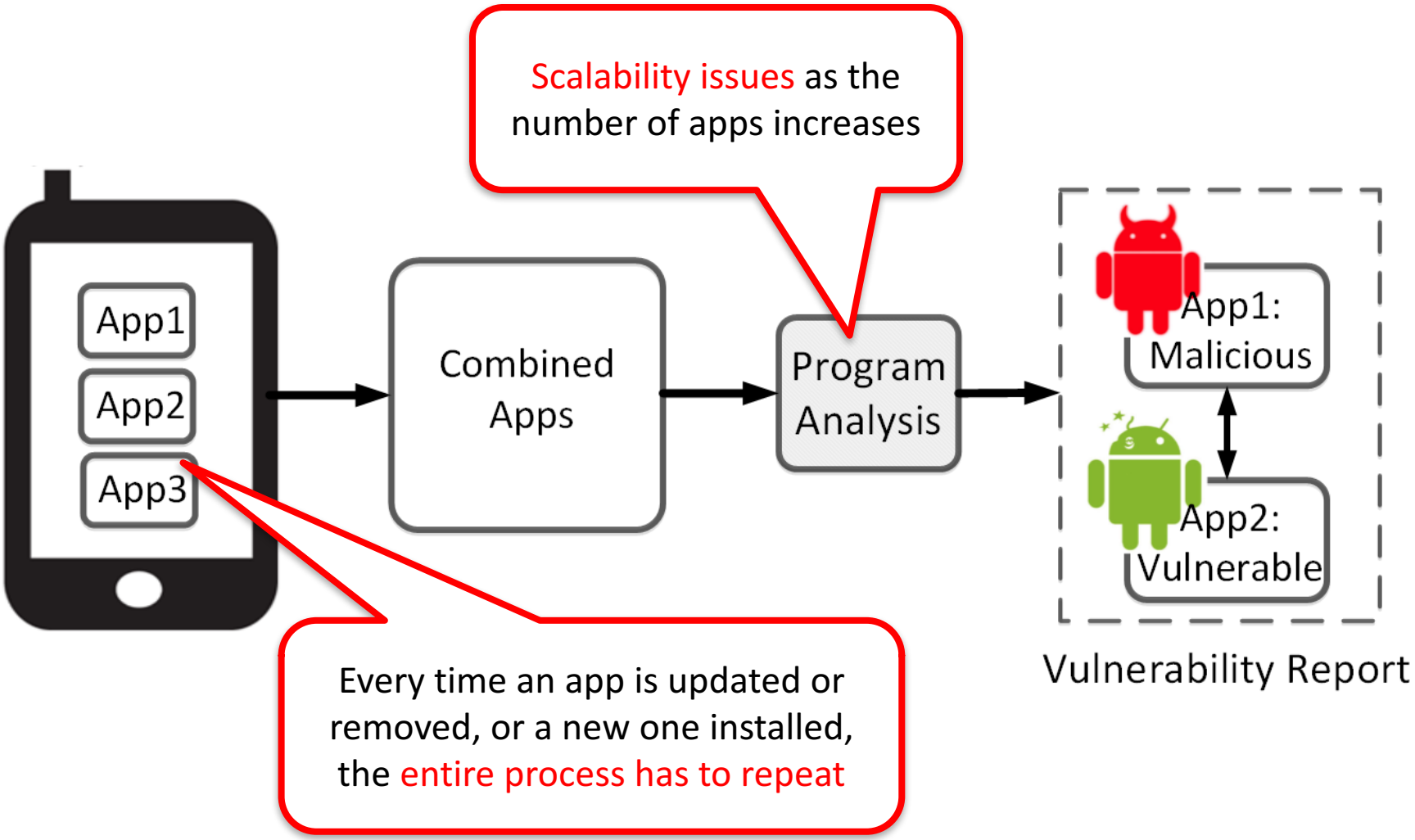
```
// receives the Intent
if (checkCallingPermission("permission.CALL_PHONE")==PackageManager.PERMISSION_GRANTED) {
        // makes a sensitive API call
}
```

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



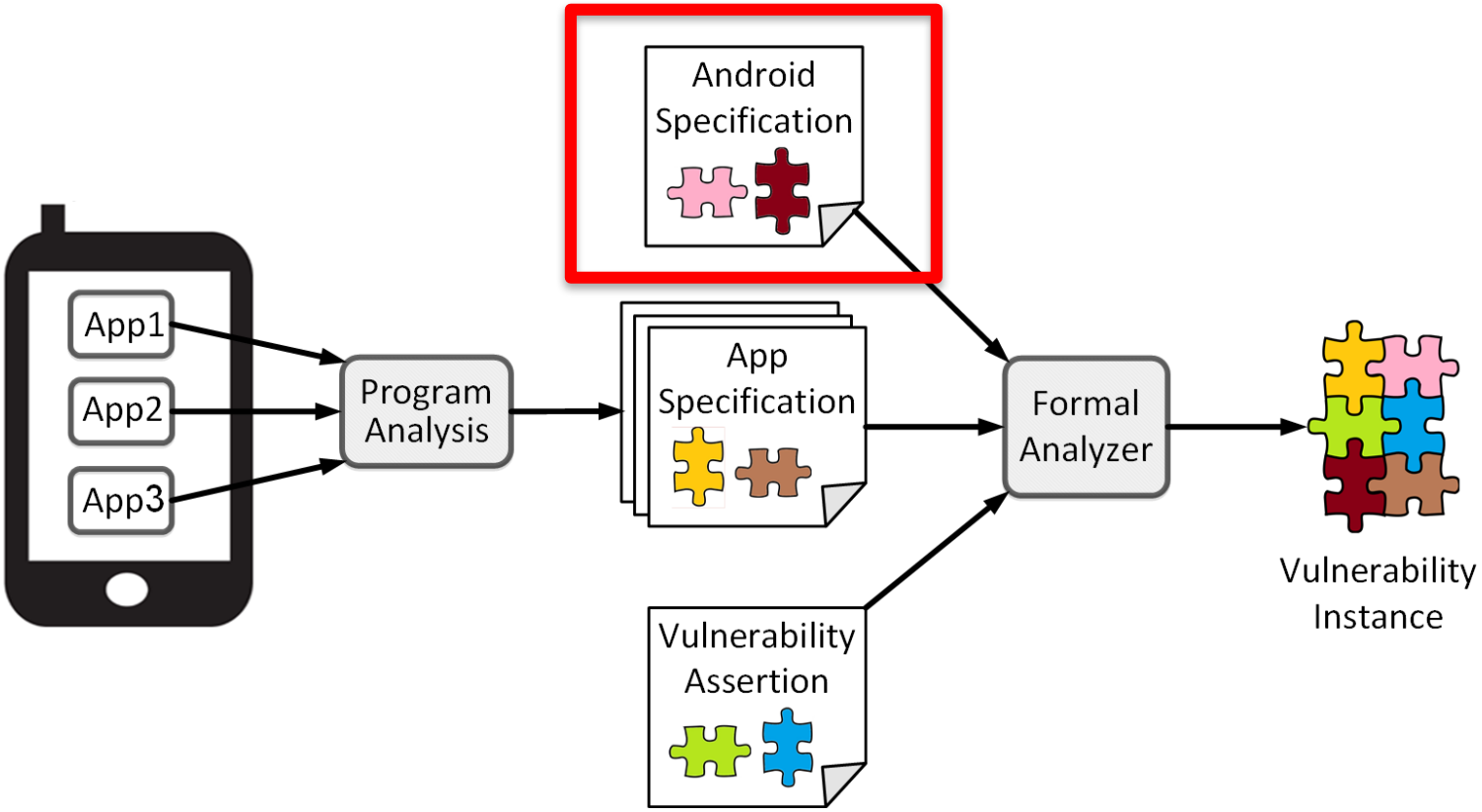
Requirements and Insights

- 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 **o**f Inter-app **V**uln**er**abili**t**ies



Bagheri, Sadeghi, Garcia, and Malek. COVERT: Compositional Analysis of Android Inter-App Permission Leakage. *TSE* 2015.

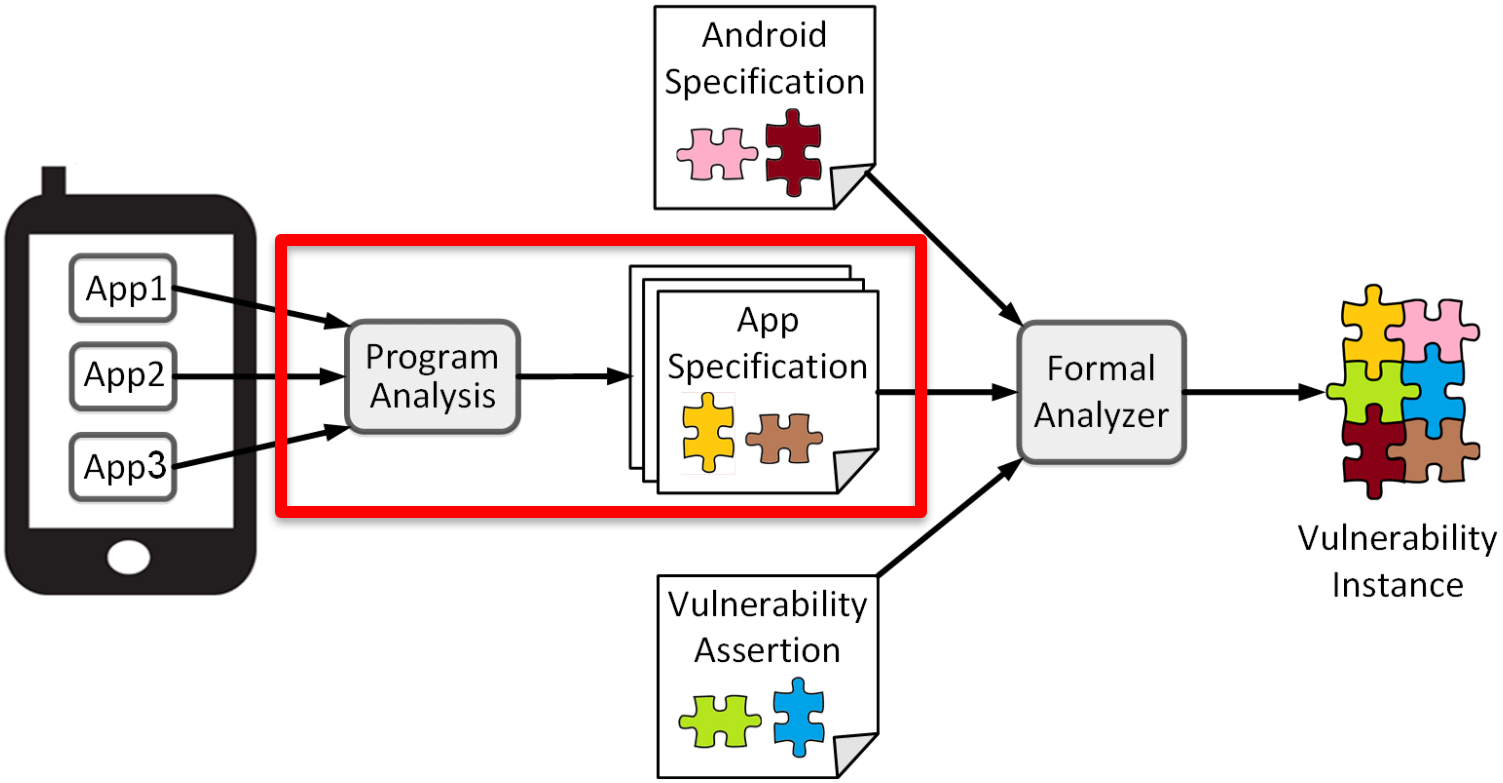
Subset of Android specification in Alloy

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

```
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

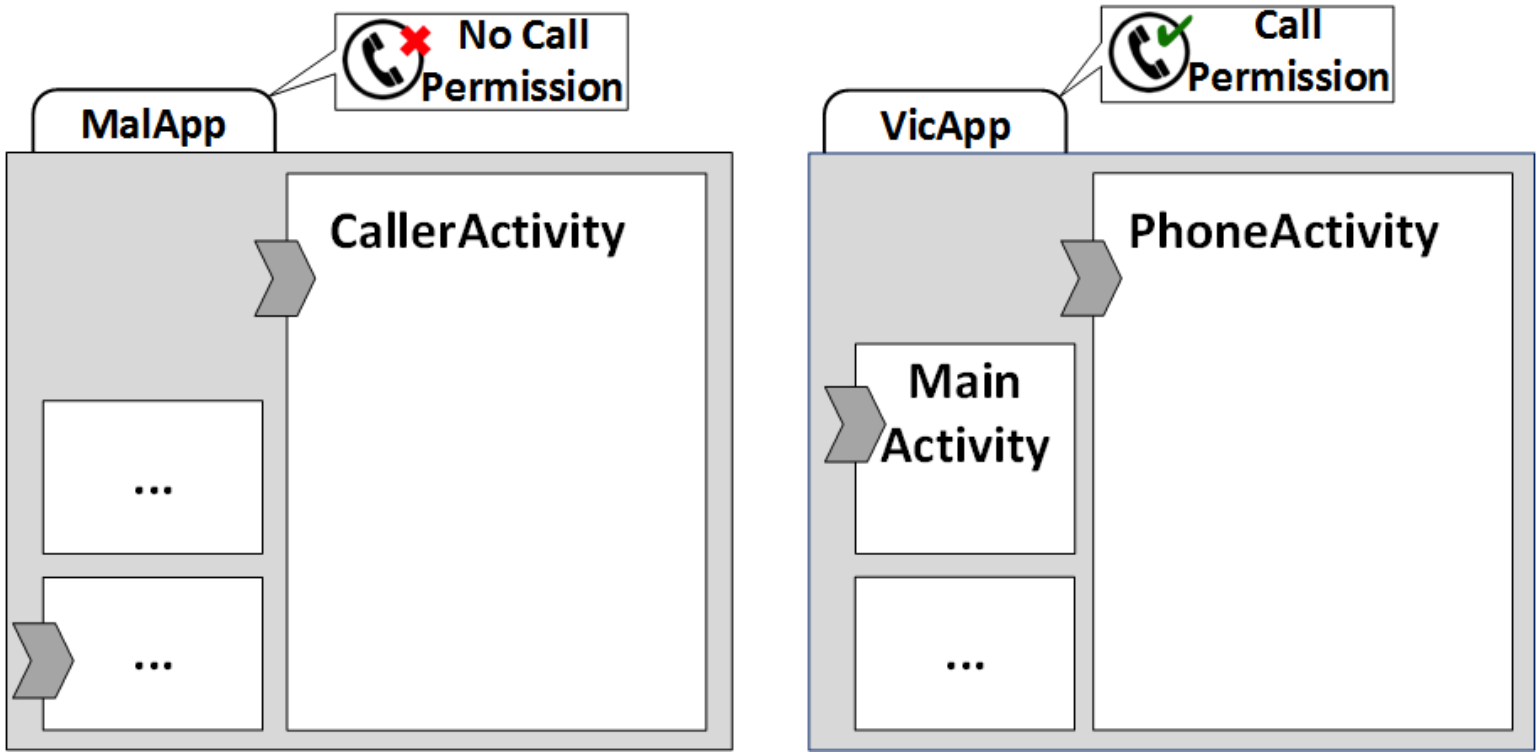
Compositional Analysis **o**f Inter-app **V**uln**er**abili**t**ies



Bagheri, Sadeghi, Garcia, and Malek. COVERT: Compositional Analysis of Android Inter-App Permission Leakage. *TSE* 2015.

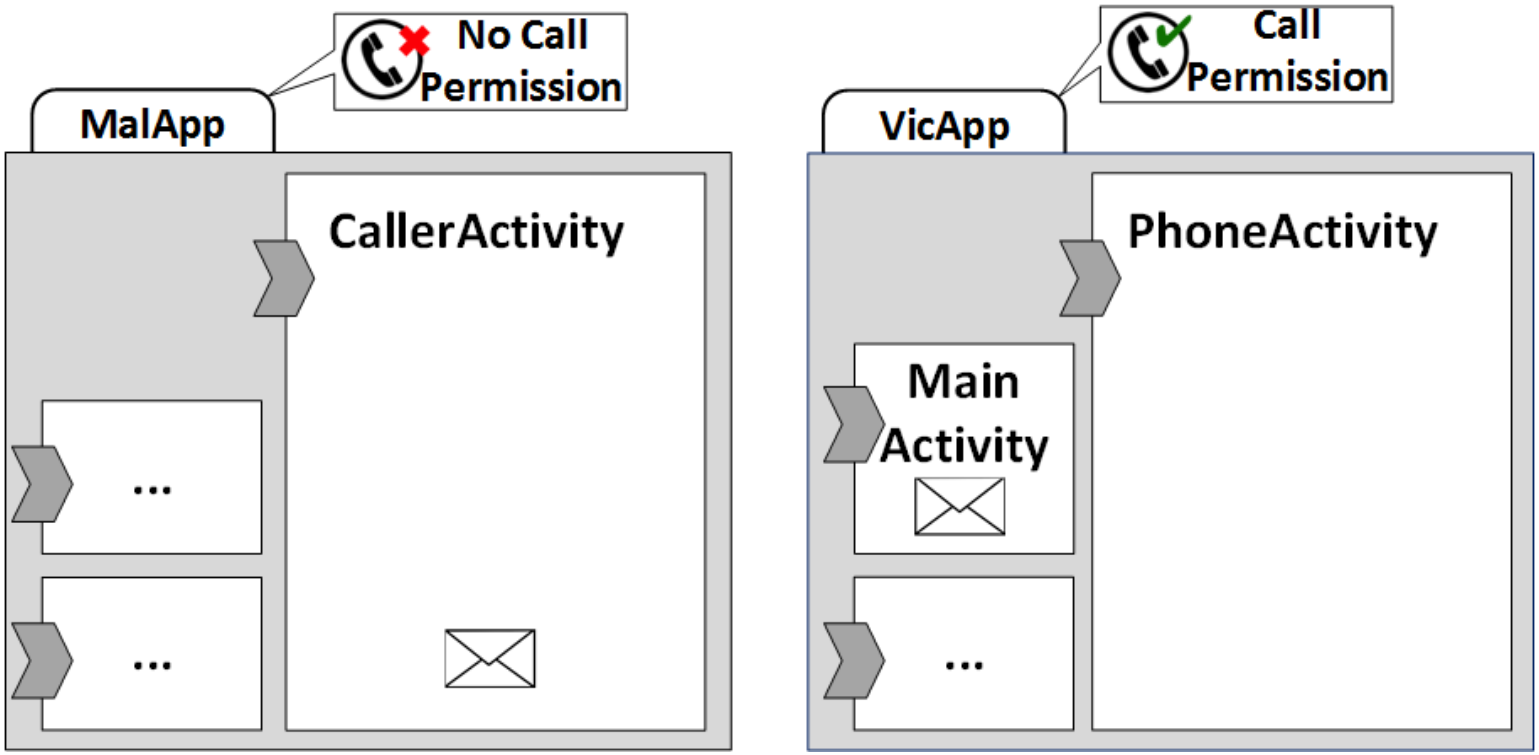
Static extraction of architecture

1. Architectural elements and properties defined in the manifest file



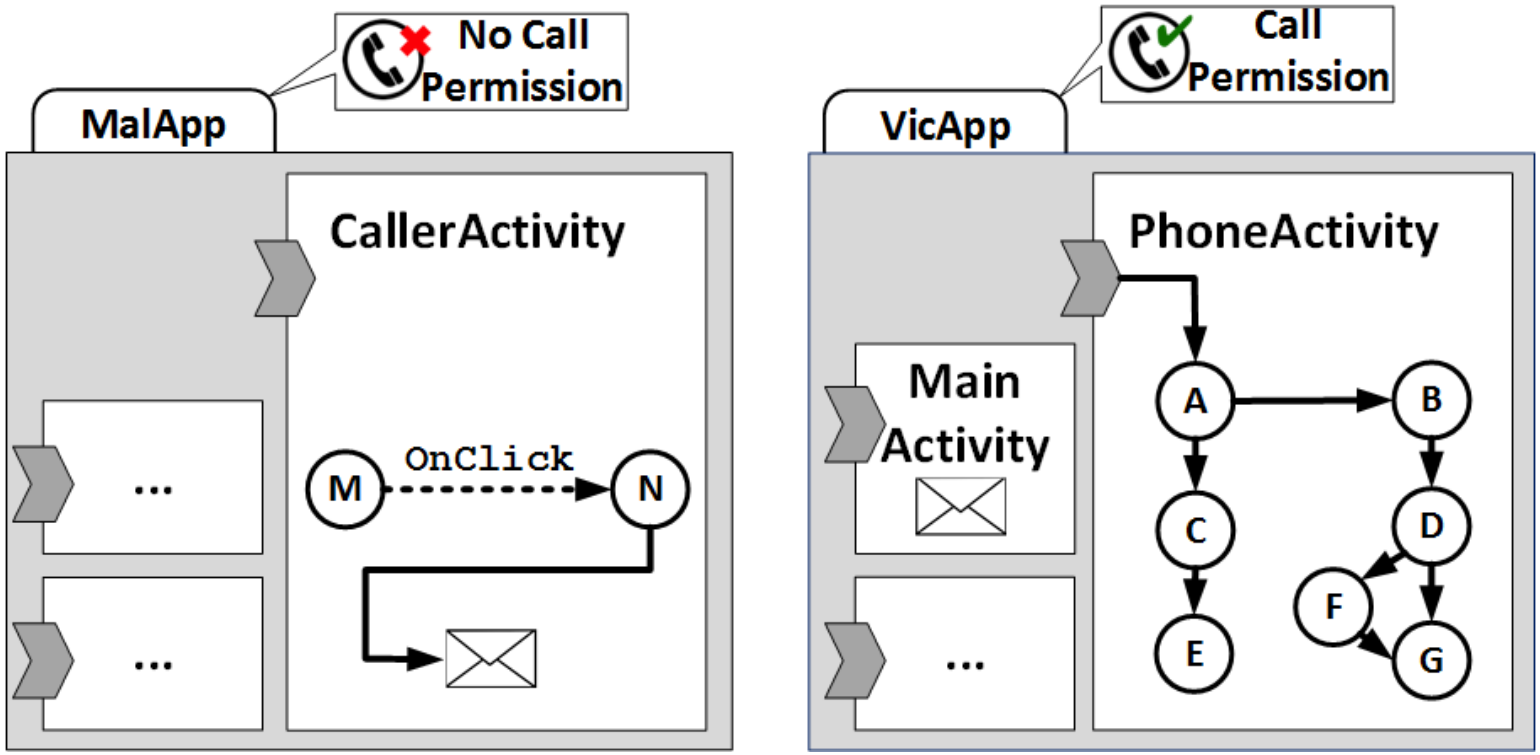
Static extraction of architecture

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



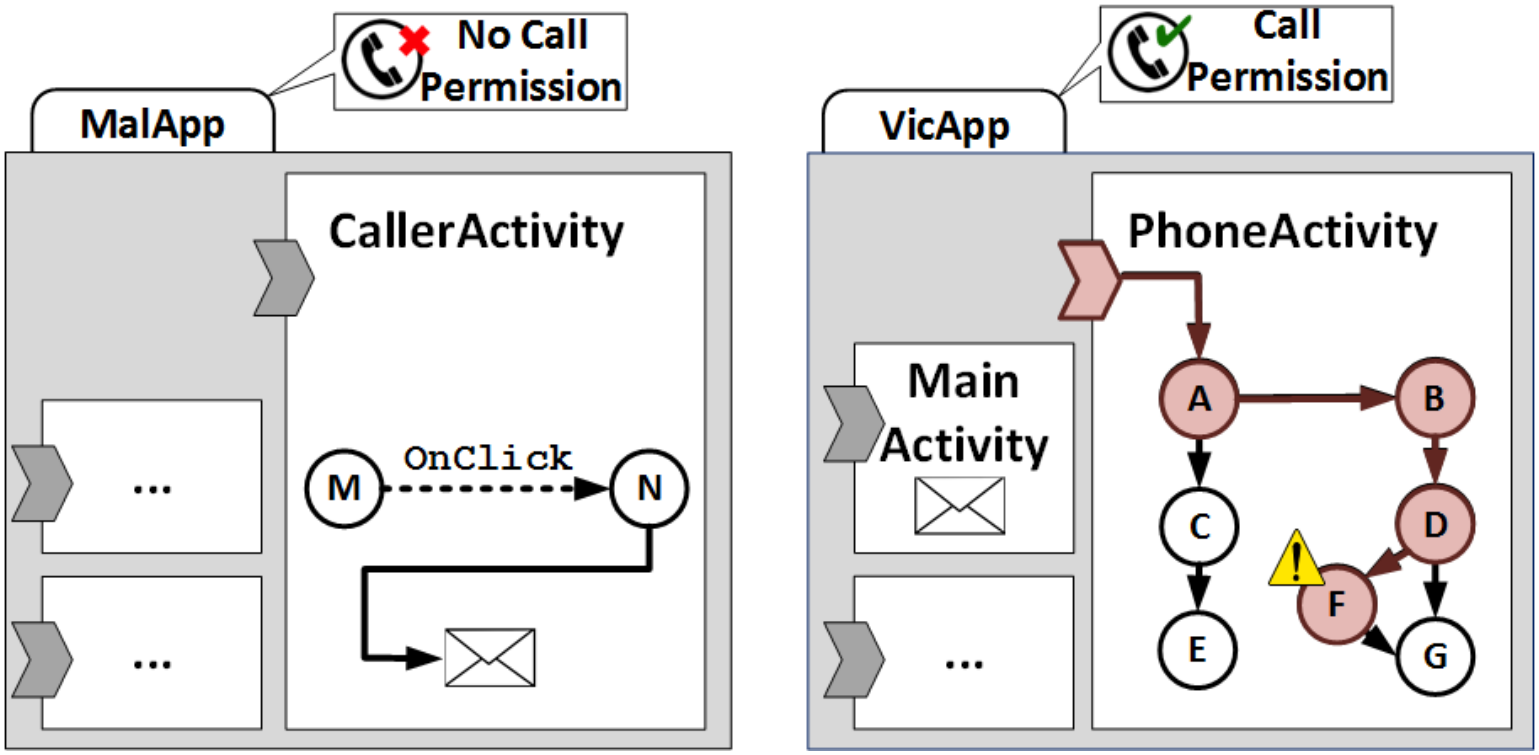
Static extraction of architecture

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



Specification of architecture in Alloy

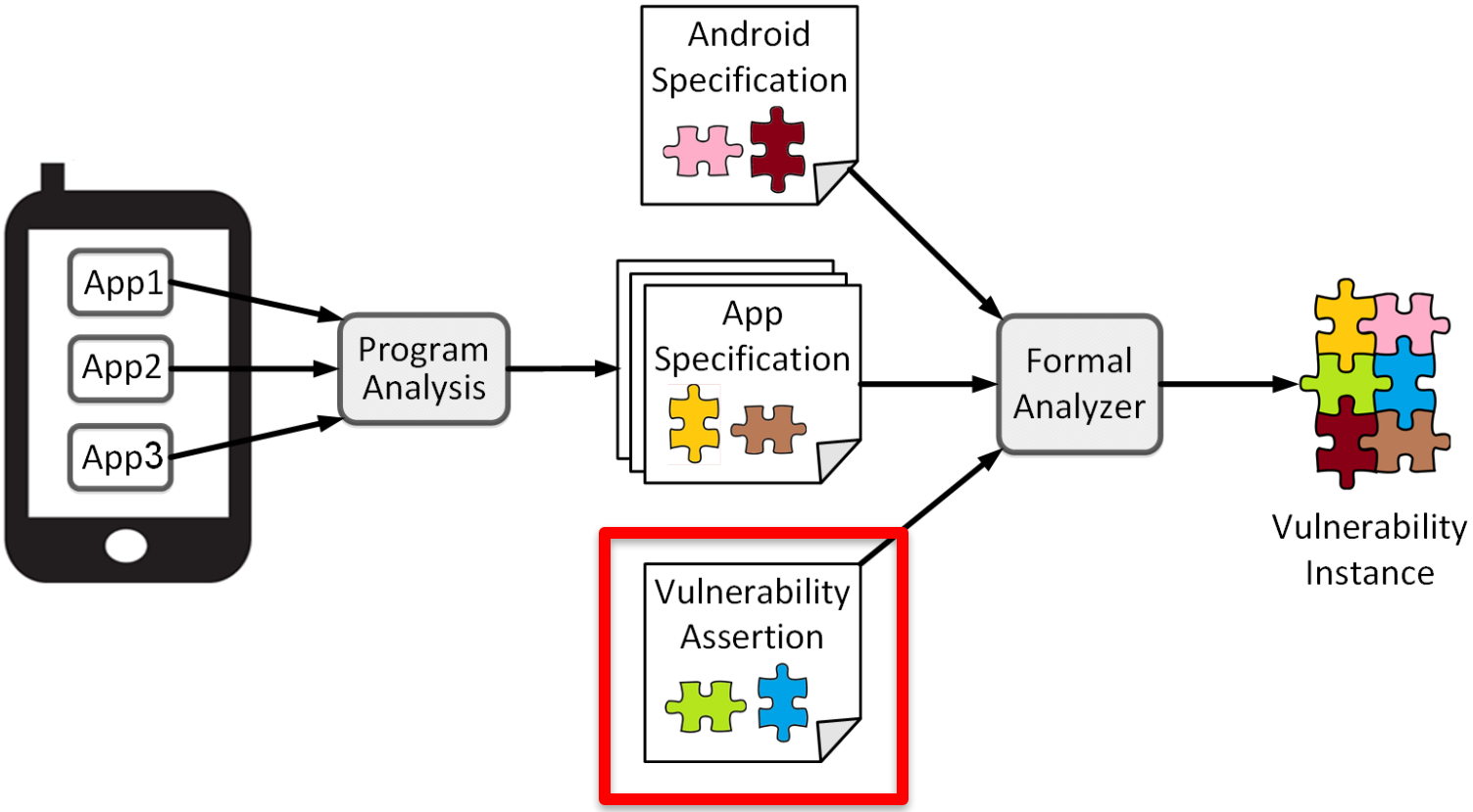
```
module MalApp

open appDeclaration

one sig MalApp extends Application{}{
   no usesPermissions
   no appPermissions
}

one sig CallerActivity extends Activity{}{
   app in MalApp
   intentFilter = IntentFilter1
   no permissions
}
one sig intent1 extends Intent{}{
    sender = CallerActivity
    component = PhoneActivity
    action = PHONE_CALL
    no categories
    extraData = Yes
}
```

```
module VicApp

open appDeclaration

one sig VicApp extends Application{}{
   usesPermissions = CALL_PHONE
   no appPermissions
}

one sig PhoneActivity extends Activity{}{...}
```

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

COVERT

Compositional Analysis **o**f Inter-app **V**uln**er**abili**t**ies



Bagheri, Sadeghi, Garcia, and Malek. COVERT: Compositional Analysis of Android Inter-App Permission Leakage. *TSE* 2015.

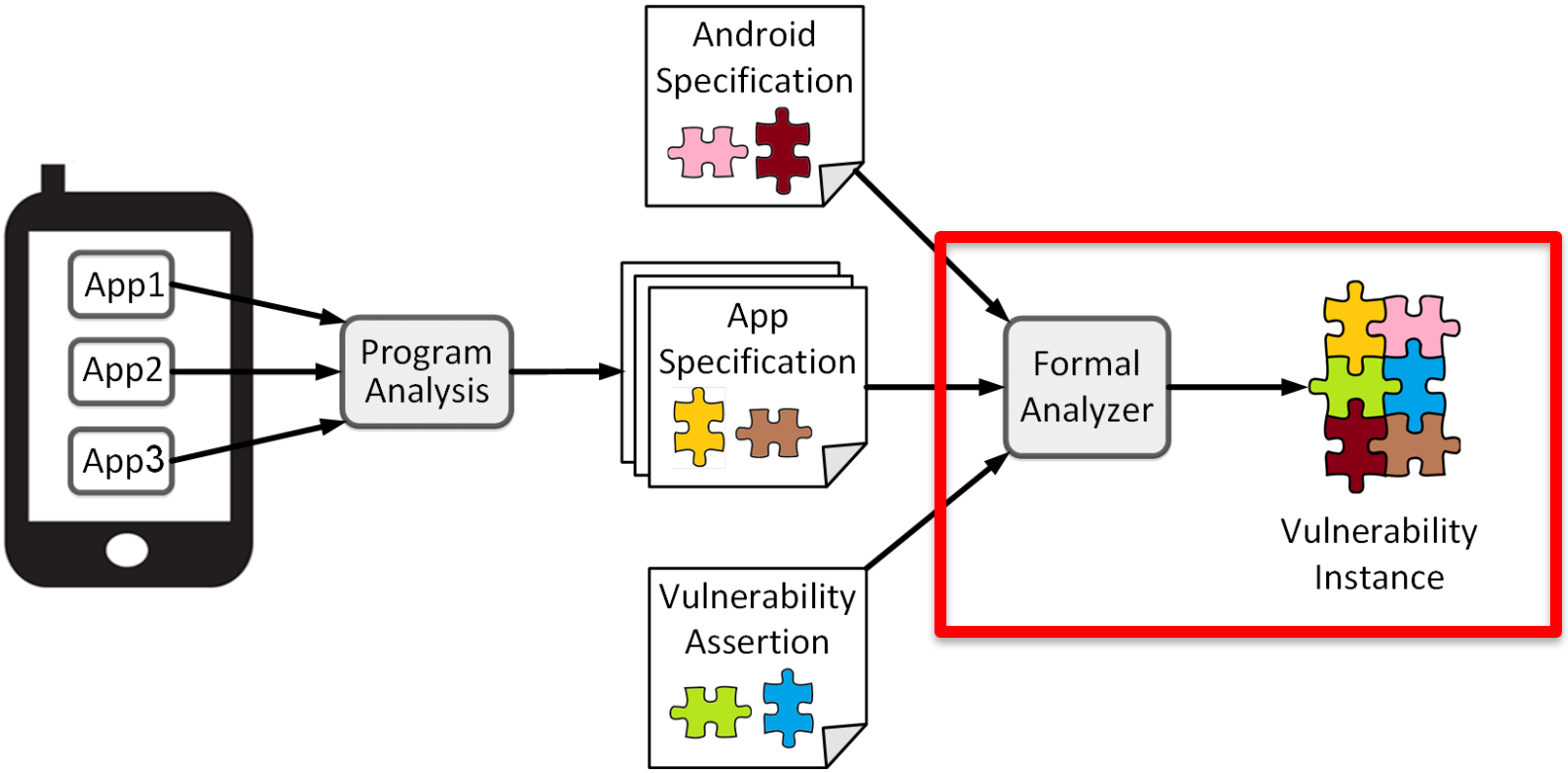
Specification of privilege escalation in Alloy

```
assert privilegeEscalation{
  no disj src, dst: Component, i:Intent|
  (src in i.sender) &&
  (dst in intentResolver[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

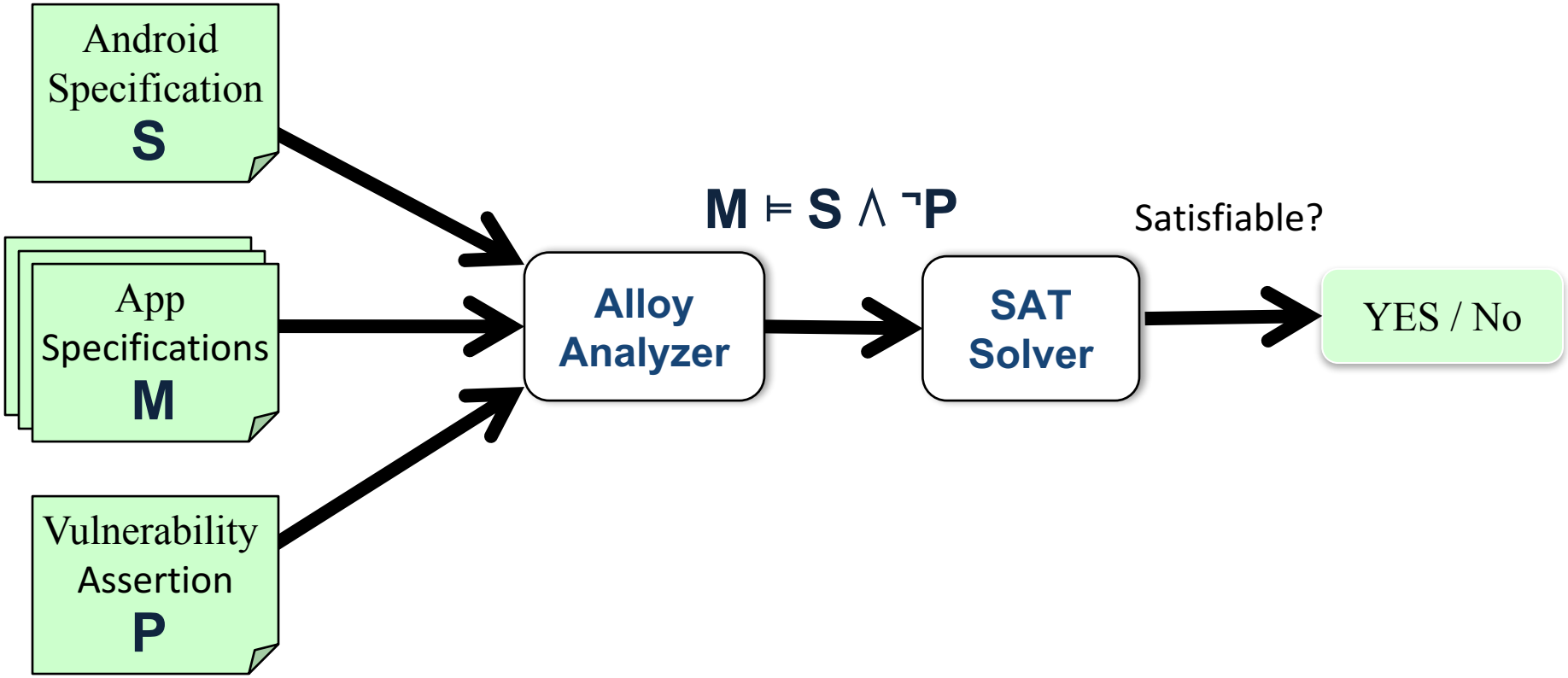
COVERT

Compositional Analysis **o**f Inter-app **V**uln**er**abili**t**ies



Bagheri, Sadeghi, Garcia, and Malek. COVERT: Compositional Analysis of Android Inter-App Permission Leakage. *TSE* 2015.

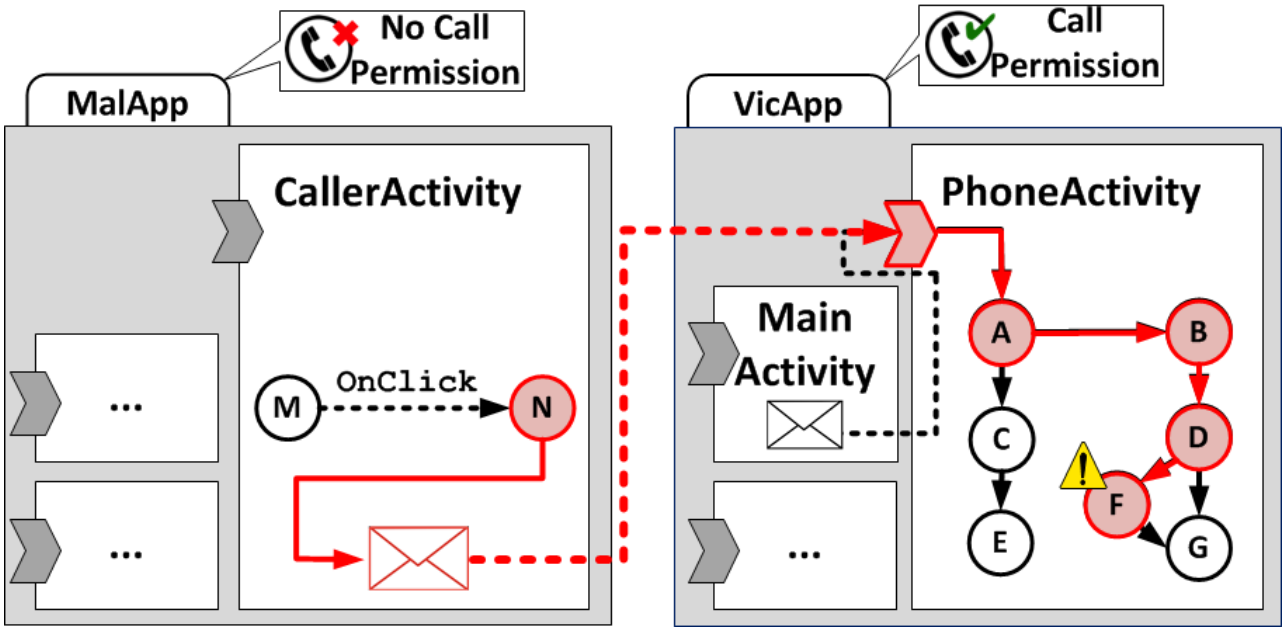
Formulation as a model checking problem



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

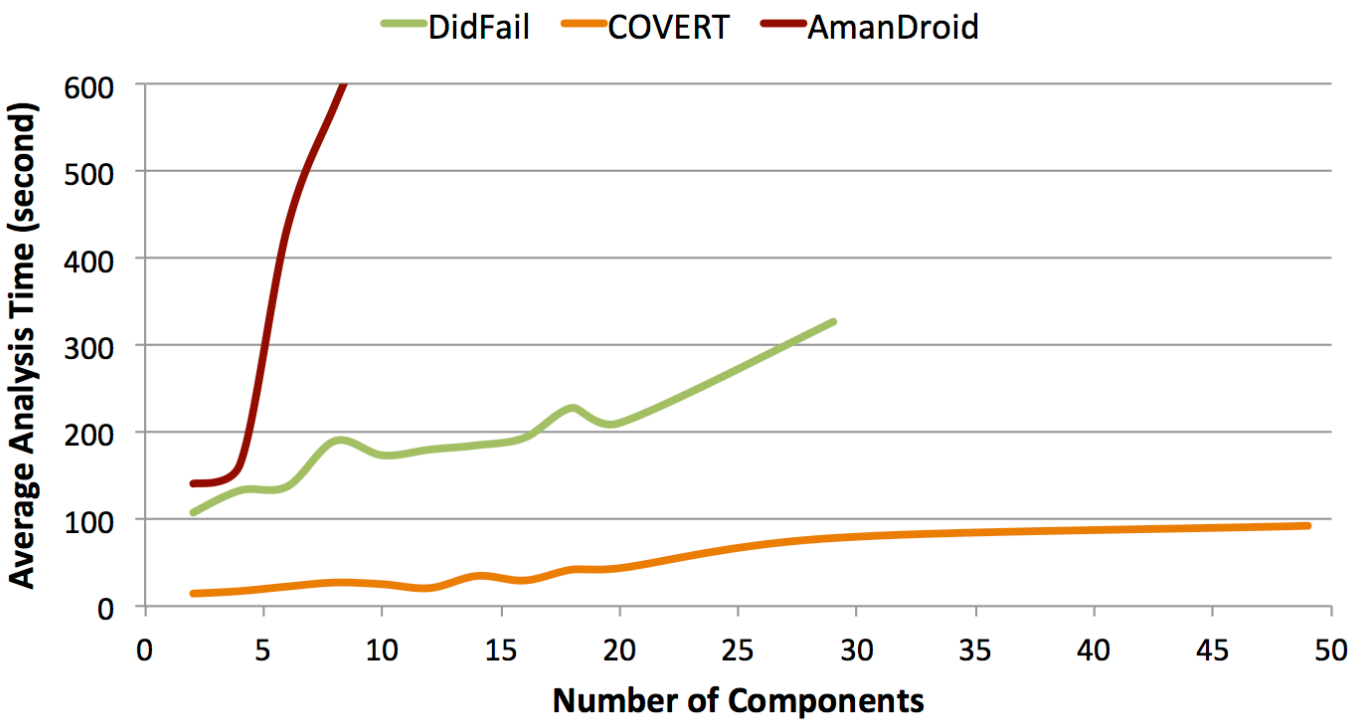
- 4,000 Android apps from four repositories
  - **Google Play** (1,000 most popular + 600 random)
  - **F-Droid** (1,100 apps)
  - **Malgenome** (1,200 random)
  - **Bazaar** (100 most popular)
- Partitioned into 80 non-overlapping bundles, each comprising 50 apps
- Total number of detected vulnerabilities: 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

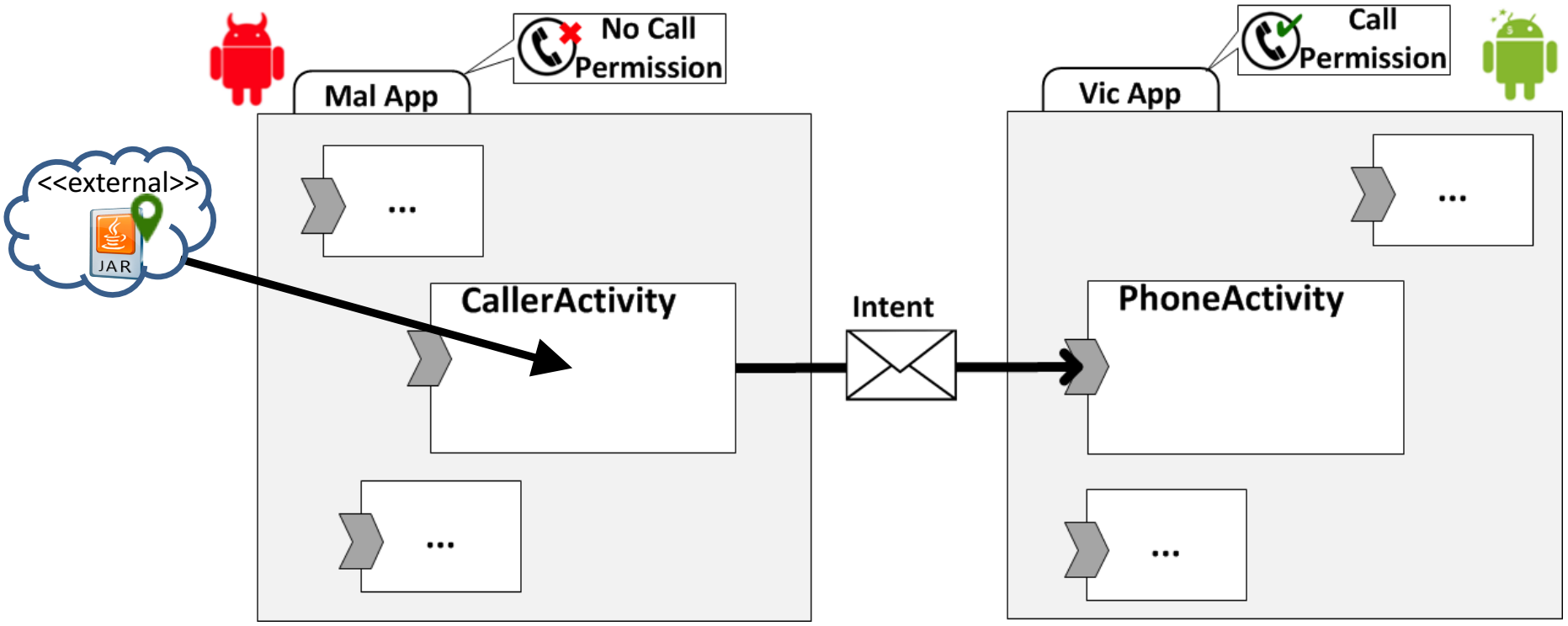
Any app

- *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

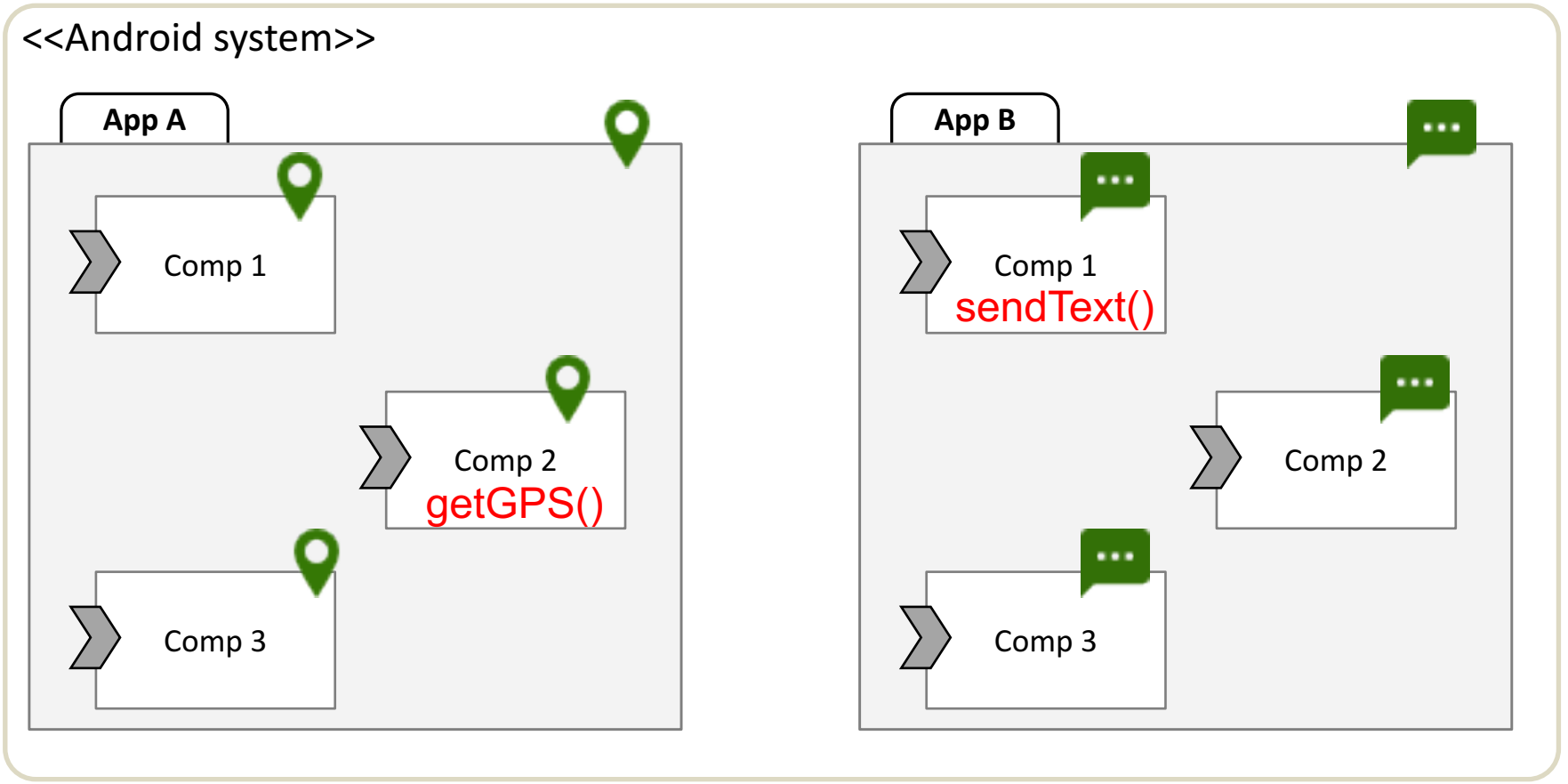
What needs to change?

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

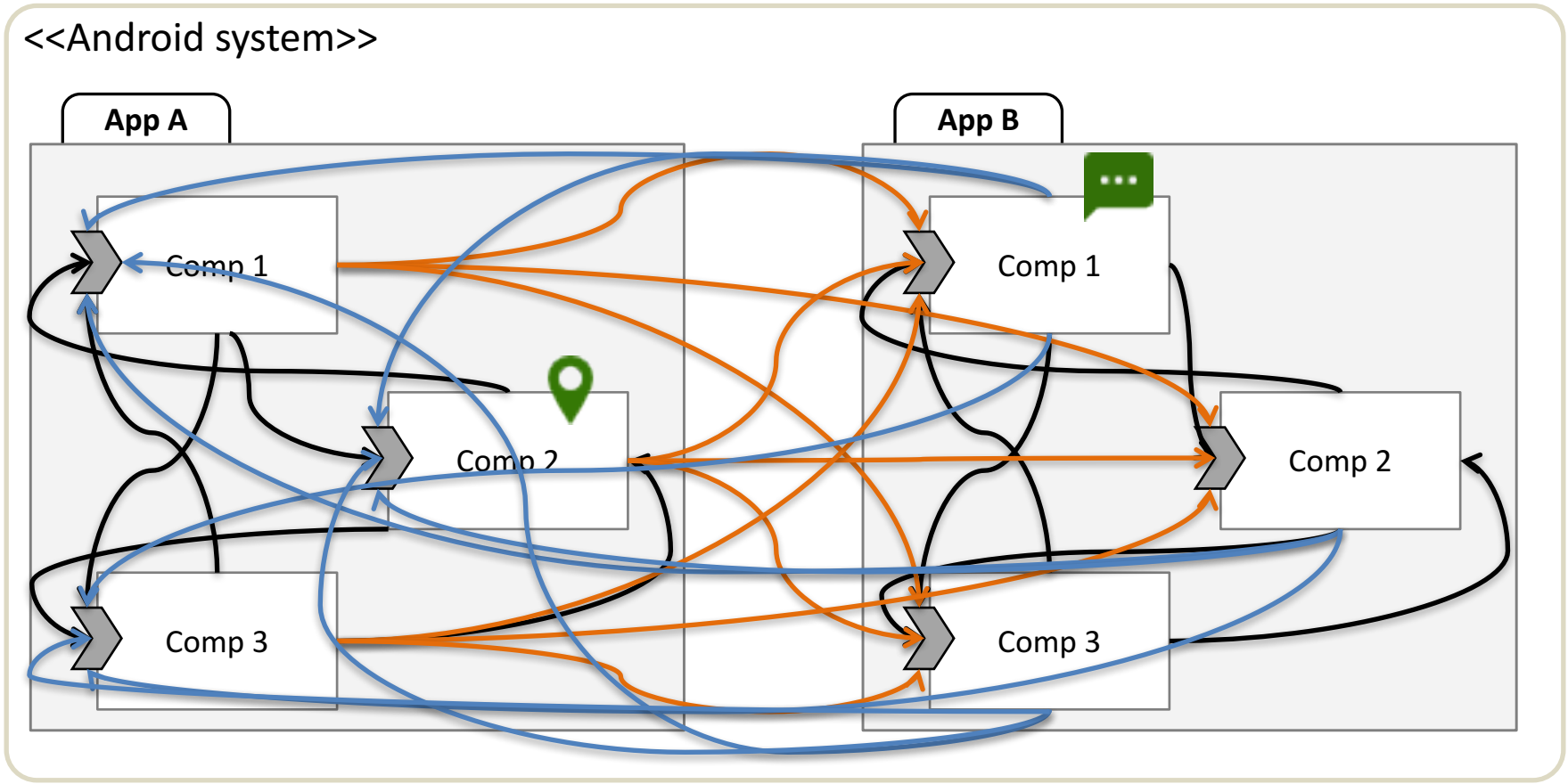
Resource access privileges

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



ICC privileges

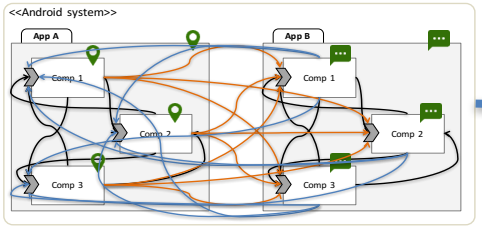
- Determine the required ICCs for each component to run
  - Resolve the Intents and their recipients



DELDroid

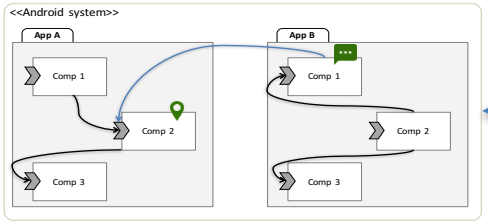
Determination and **E**nforcement of **L**east-Privilege Architecture in An**droid**

Over-privilege architecture



Static Analysis

Least-privilege architecture



Extra privileges to revoke

Android Framework

Privilege Manager Layer

Resource Monitor

ICC Monitor

ECA Rules

Android Runtime

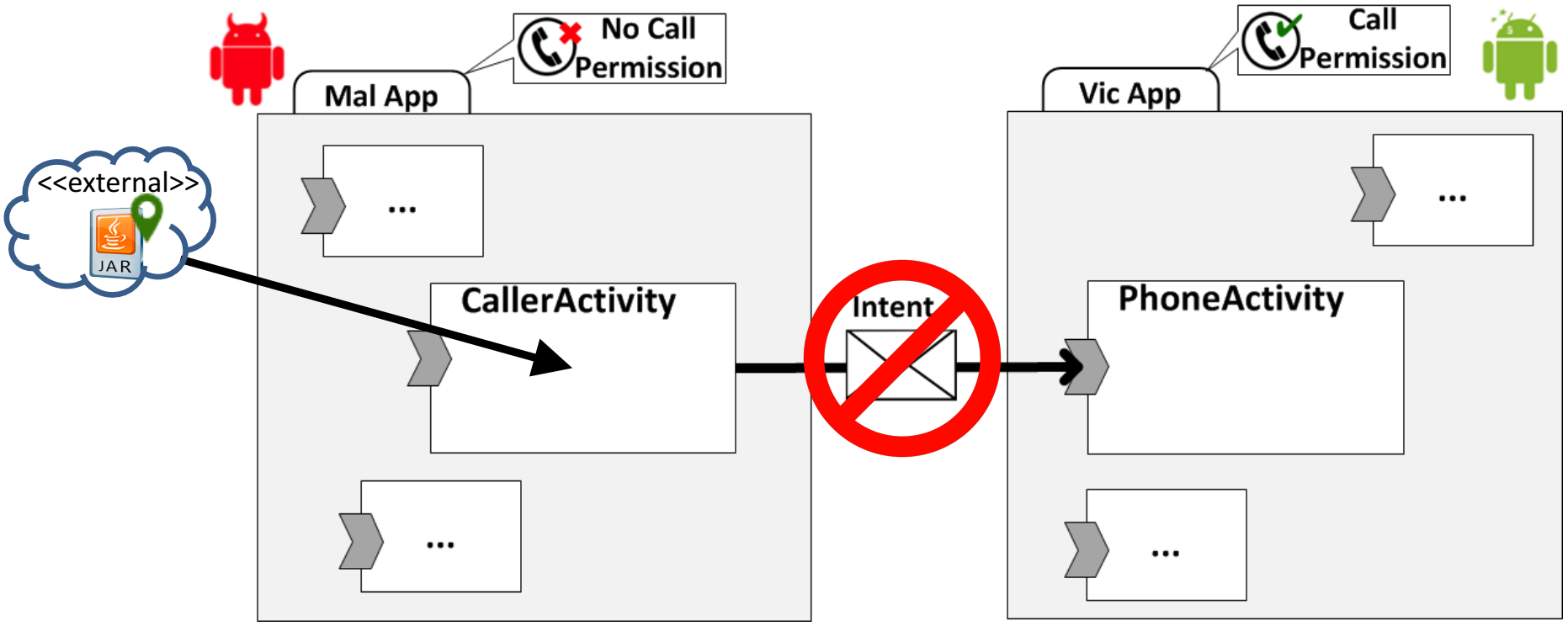
Libraries

Linux Kernel

Custom version of Android

Hammad, Bagheri, and Malek. DELDroid: Determination and Enforcement of Least Privilege Architecture in Android. *ICSA* 2017.

Hidden code



DELDroid can effectively thwart such attacks

Attack surface reduction

Num of Apps	Num of Comps	Communication Domain			Permission Granted Domain		
		Original	LP	Reduction (%)	Original	LP	Reduction (%)
30	306	29,031	42	99.86	1,642	45	97.26
30	432	78,237	625	99.20	2,954	61	97.94
30	422	65,709	173	99.74	2,510	54	97.85
30	449	80,372	205	99.74	4,234	78	98.16
30	353	56,868	345	99.39	1,536	51	96.68
30	541	85,556	661	99.23	4,461	181	95.94
30	562	82,863	137	99.83	1,577	58	96.32
30	362	50,208	250	99.50	1,946	24	98.77
30	265	25,817	129	99.50	1,568	30	98.09
30	421	50,001	74	99.85	2,386	28	98.83
Average	411.3	60,466.2	264.1	99.58	2,481.4	61.0	97.58
Avg. (per app)	13.7	2,015.5	8.8	99.56	82.7	2.0	97.54

10 experiments with 30 randomly selected apps

Attack surface reduction – ICC

Num of Apps	Num of Comps	Communication Domain			Permission Granted Domain		
		Original	LP	Reduction (%)	Original	LP	Reduction (%)
30	306	29,031	42	99.86	1,642	45	97.26
30	432	78,237	625	99.20	2,954	61	97.94
30	422	65,709	173	99.74	2,510	54	97.85
30	449	80,372	205	99.74	4,234	78	98.16
30	353	56,868	345	99.39	1,536	51	96.68
30	541	85,556	661	99.23	4,461	181	95.94
30	562	82,863	137	99.83	1,577	58	96.32
30	362	50,208	250	99.50	1,946	24	98.77
30	265	25,817	129	99.50	1,568	30	98.09
30	421	50,001	74	99.85	2,386	28	98.83
Average	411.3	60,466.2	264.1	99.58	2,481.4	61.0	97.58
Avg. (per app)	13.7	2,015.5	8.8	99.56	82.7	2.0	97.54

Over **99%** reduction in ICC privileges

Attack surface reduction – resource access

Num of Apps	Num of Comps	Communication Domain			Permission Granted Domain		
		Original	LP	Reduction (%)	Original	LP	Reduction (%)
30	306	29,031	42	99.86	1,642	45	97.26
30	432	78,237	625	99.20	2,954	61	97.94
30	422	65,709	173	99.74	2,510	54	97.85
30	449	80,372	205	99.74	4,234	78	98.16
30	353	56,868	345	99.39	1,536	51	96.68
30	541	85,556	661	99.23	4,461	181	95.94
30	562	82,863	137	99.83	1,577	58	96.32
30	362	50,208	250	99.50	1,946	24	98.77
30	265	25,817	129	99.50	1,568	30	98.09
30	421	50,001	74	99.85	2,386	28	98.83
Average	411.3	60,466.2	264.1	99.58	2,481.4	61.0	97.58
Avg. (per app)	13.7	2,015.5	8.8	99.56	82.7	2.0	97.54

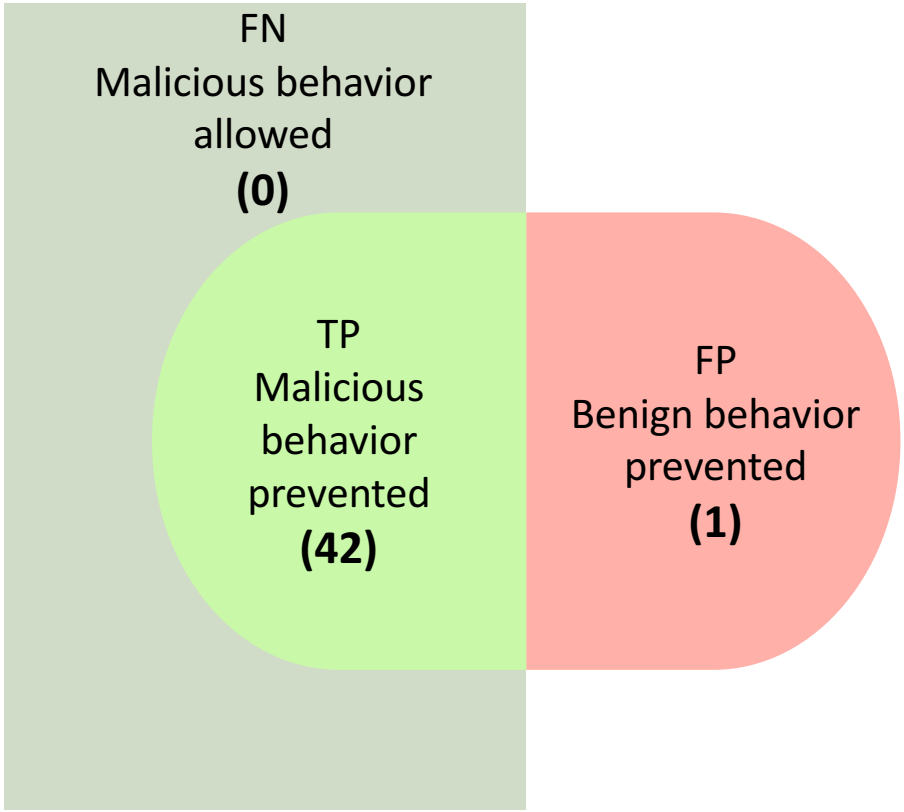
Over **97%** reduction in resource access privileges

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

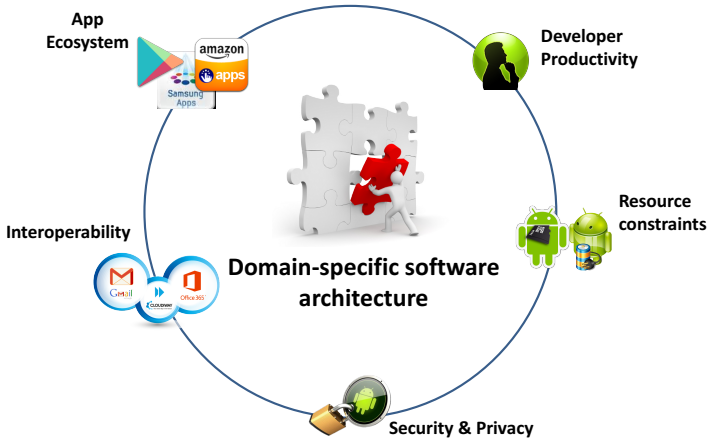


Precision ( ) = 97.76%

Recall ( ) = 100%

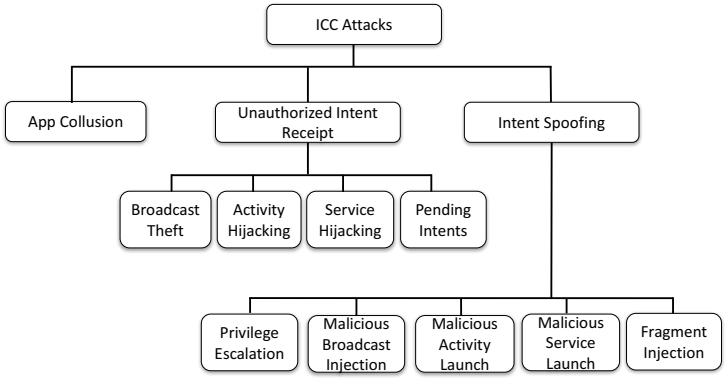
Recap

Role of architecture in mobile computing



Bagheri, Garcia, Sadeghi, Malek, and Medvidovic. Software Architectural Principles in Contemporary Mobile Software: from Conception to Practice. *JSS* 2016.

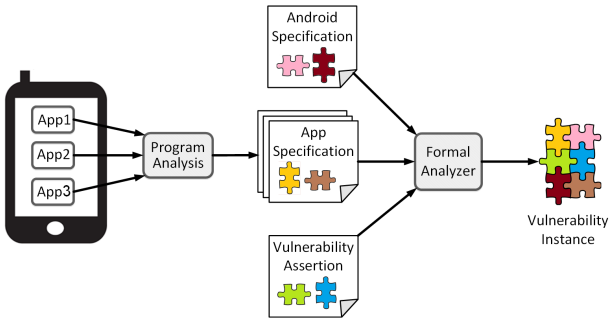
Inter-component communication attacks



Sadeghi, Bagheri, Garcia, and Malek. A Taxonomy and Qualitative Comparison of Program Analysis Techniques for Security Assessment of Android Software. *TSE* 2017.

COVERT

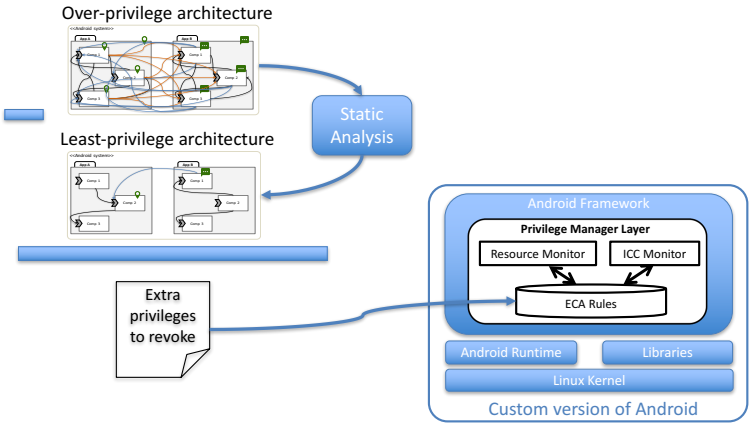
Compositional Analysis **o**f Inter-app **V**uln**er**abili**t**ies



Bagheri, Sadeghi, Garcia, and Malek. COVERT: Compositional Analysis of Android Inter-App Permission Leakage. *TSE* 2015.

DELDroid

Determination and **E**nforcement of **L**east-Privilege Architecture in An**droid**



Hammad, Bagheri, and Malek. DELDroid: Determination and Enforcement of Least Privilege Architecture in Android. *ICSA* 2017.

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