Seven Ways to Hang Yourself with Google Android

Yekaterina Tsipenyuk O'Neil

Principal Security Researcher

Erika Chin

Ph.D. Student at UC Berkeley

Yekaterina Tsipenyuk O'Neil

- Founding Member of the Security Research Group at Fortify (now an HP Company)
- Code audits, identifying insecure coding patterns, and providing security content for Fortify's software security products
- B.S. and M.S. in CS from UC San Diego
- Thesis focused on mobile agent security

Erika Chin

- Ph.D. student in Computer Science at UC Berkeley (Security research group)
- B.S. from University of Virginia
- Research interest in improving mobile phone security
- Recently presented at MobiSys 2011 on vulnerabilities stemming from inter-application communication in Android

Overview

- Introduction to Google Android
- Seven Ways to Hang Yourself
- Results of Empirical Analysis
- Conclusion

Introduction to

GOOGLE ANDROID

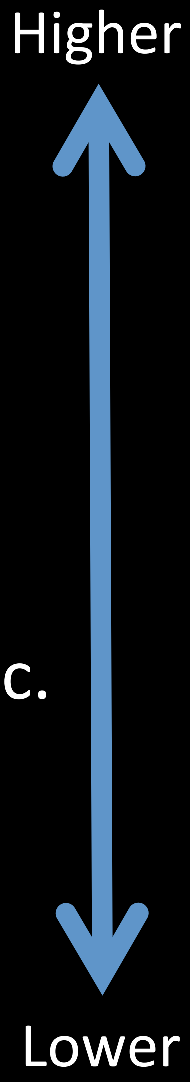
FORTIFY®
An HP Company

Introduction to Google Android

- Android architecture
- Security model
- Application breakdown
- Android manifest
- Inter-component communication

Android Architecture

- Applications
- Application framework (SDK)
- Dalvik virtual machine
  - Customized bytecode (.dex files)
- Native libraries
  - Graphics, database management, browser, etc.
  - Accessed through Java interfaces
- Linux kernel
  - Device drivers, memory management, etc.



Security Model

- Applications have unique UIDs
  - Run as separate processes on separate VMs
  - Typically cannot read each other's data and code
- Linux-style file permissions
- Android permissions protect
  - Access to sensitive APIs
  - Access to content providers
  - Inter- and intra-application communication

Application Breakdown

- Applications are divided into *components*
- 4 types of components
  - Activities
  - Services
  - Broadcast Receivers
  - Content Providers

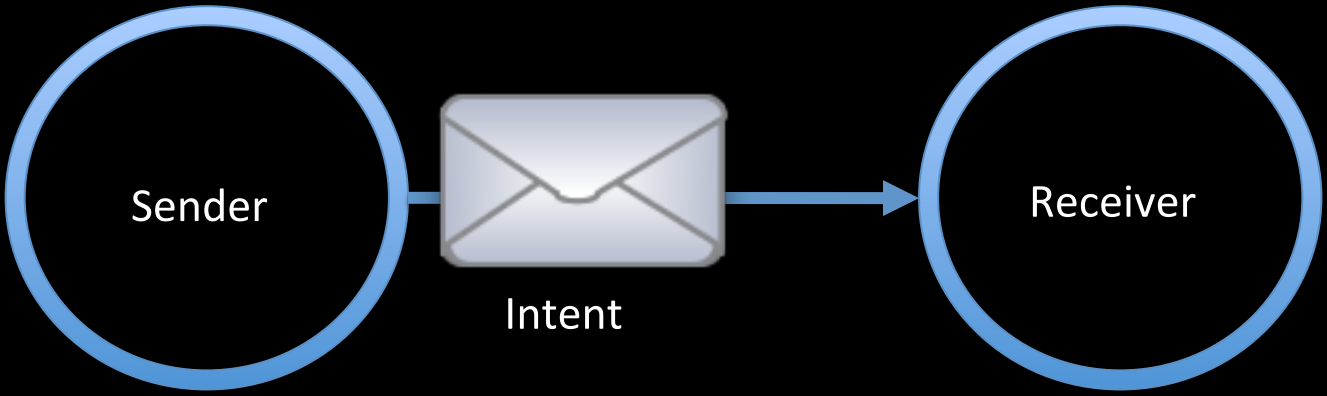
Android Manifest

Each application contains a manifest

```
<manifest ...>
  <application>
    <activity android:name=“.MyActivity”>...</activity>
    <receiver android:name=“.MyReceiver”>...</receiver>
  </application>

  <uses-sdk android:minSdkVersion=“8” />
  <uses-feature android:name=“android.hardware.CAMERA”/>

  <uses-permission
     android:name=“android.permission.INTERNET” />
  <uses-permission
     android:name=“android.permission.CAMERA” />

  <permission android:name=“com.emc.NewPermission” />
</manifest>
```

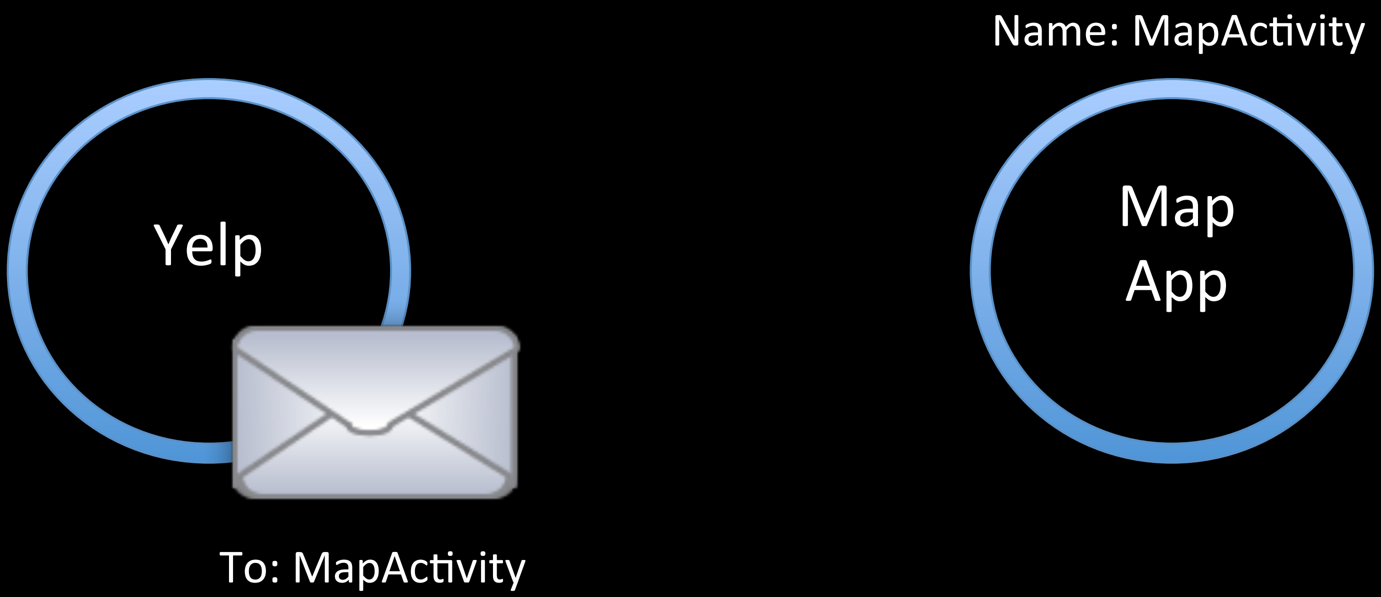
Inter-Component Communication

- Uses Intents (messages)
- Intents can be sent between components
  - Used for both intra- and inter-application communication
  - Event notifications (including system events)



Explicit Intents

- Exact recipient is specified



Name: MapActivity

Yelp

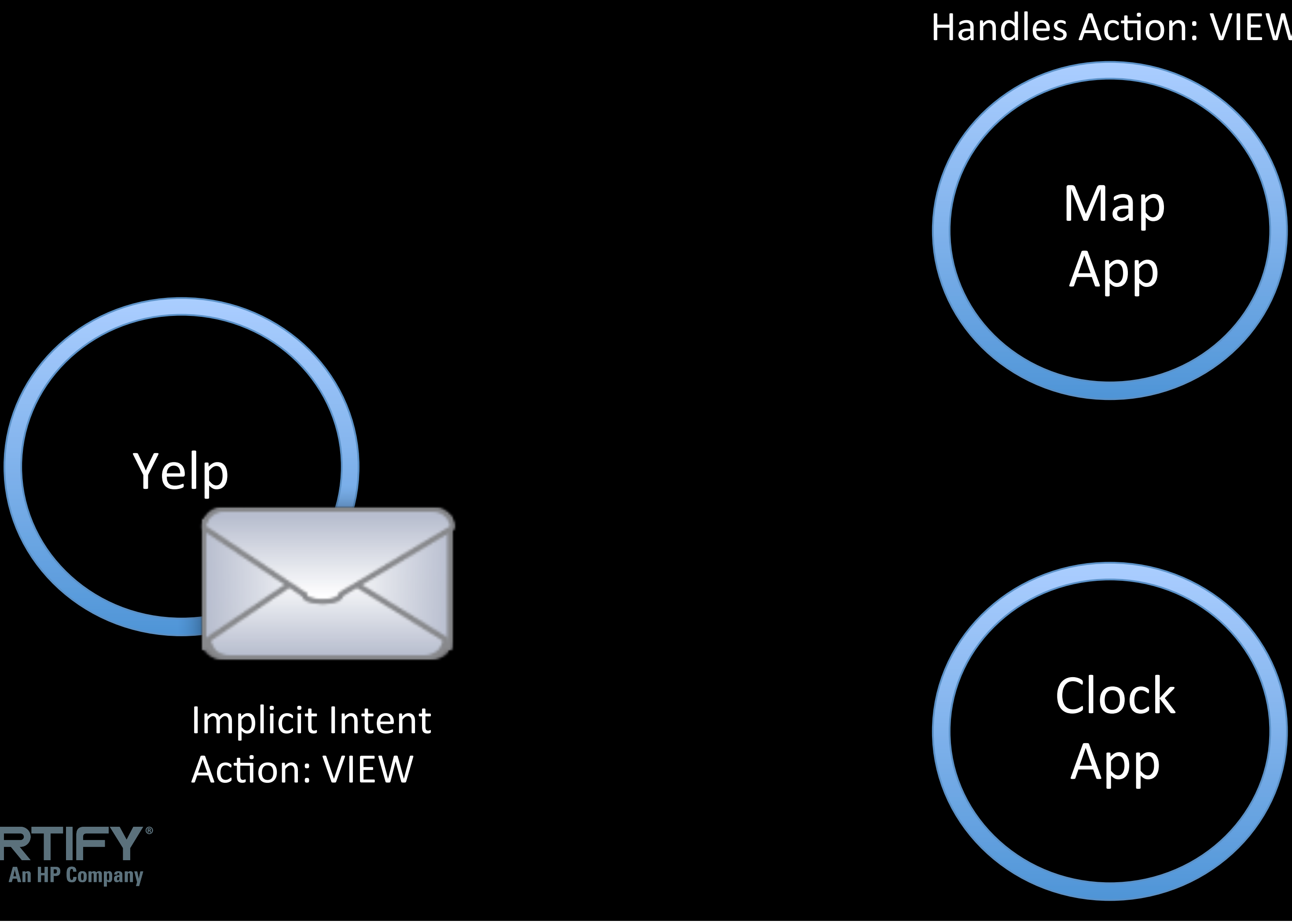
Map App

To: MapActivity

Only the specified destination receives this message

Implicit Intents

- Left up to the platform to decide where it should be delivered



Handles Action: VIEW

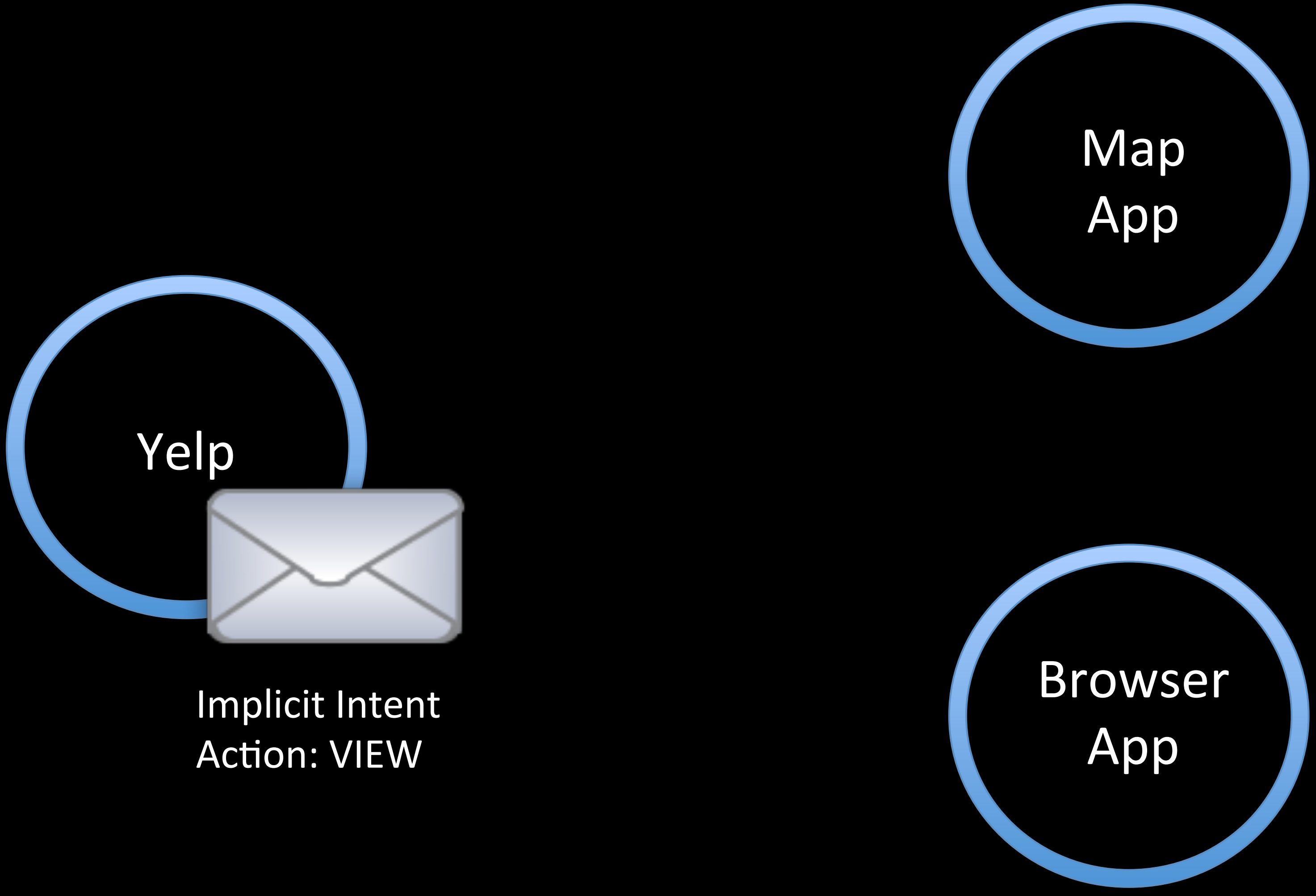
Map App

Yelp

Implicit Intent
Action: VIEW

Clock App

Implicit Intents



Handles Action: VIEW

Map App

Yelp

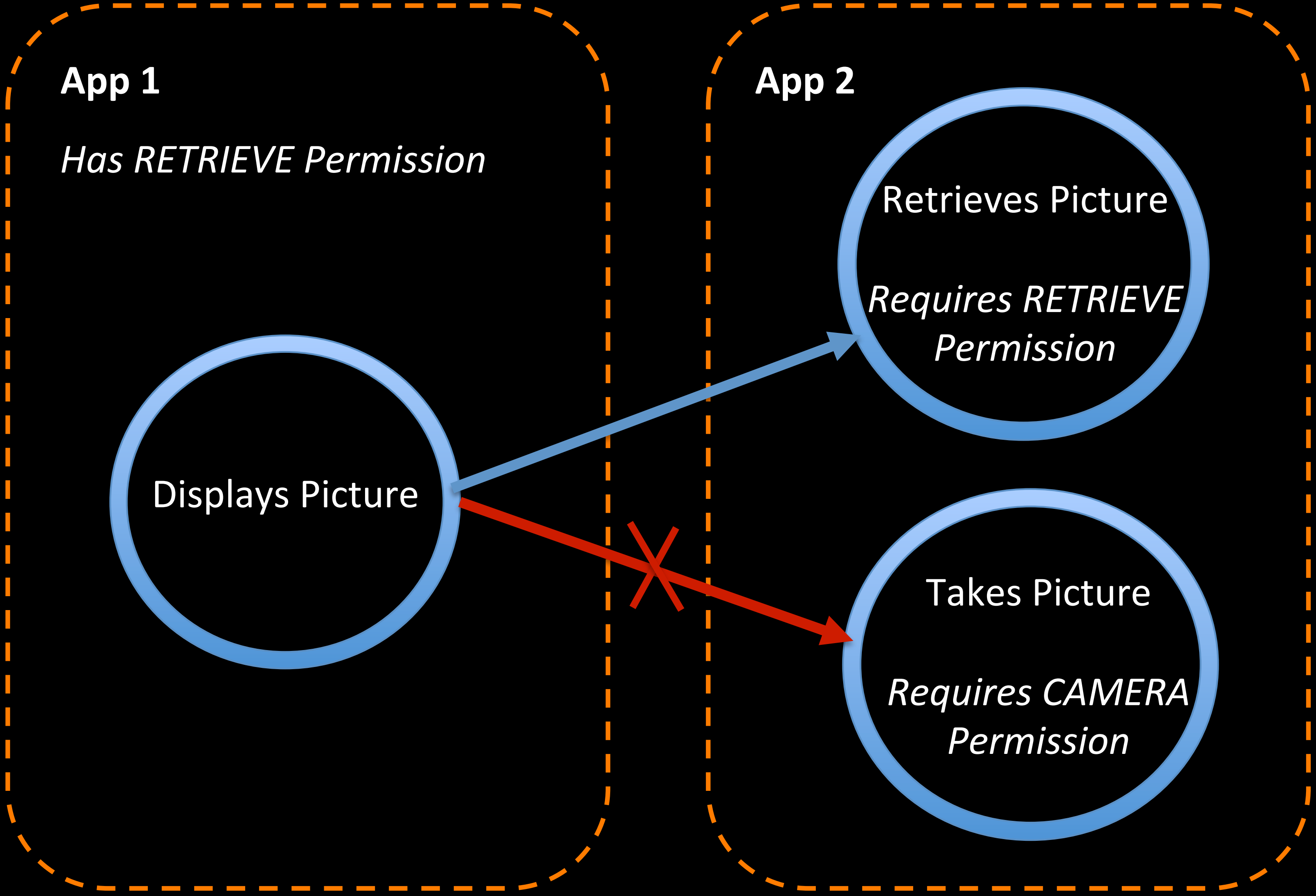
Implicit Intent
Action: VIEW

Browser App

Component Protection

- Components can be made accessible to other applications (exported) or be made private
- Components can be protected by permissions

Component Permissions


App 1
Has RETRIEVE Permission
Displays Picture
App 2
Retrieves Picture
Requires RETRIEVE Permission
Takes Picture
Requires CAMERA Permission

Seven Ways to Hang Yourself with

GOOGLE ANDROID

Google Android Vulnerabilities

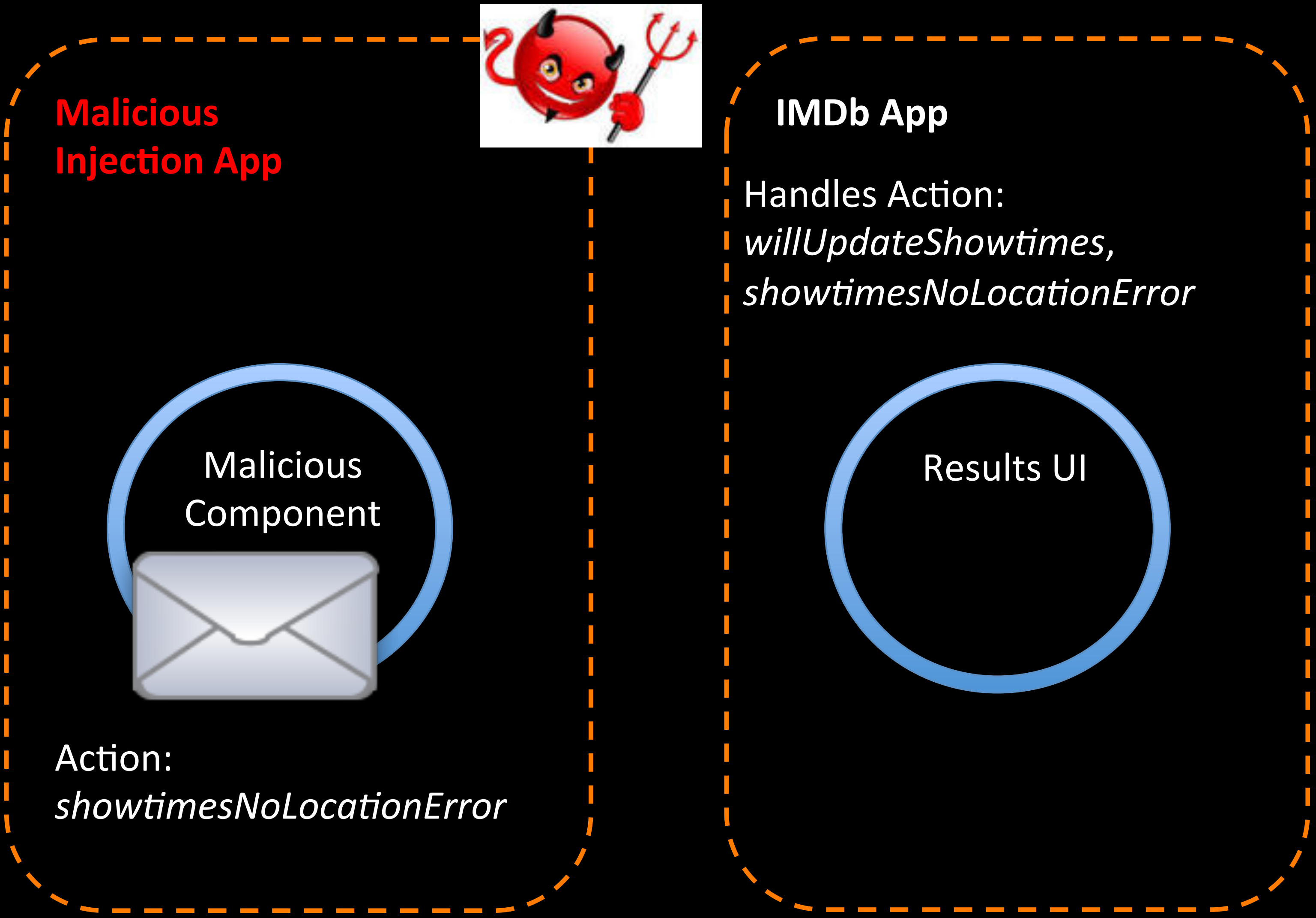
1. Intent Spoofing
2. Query String Injection
3. Unauthorized Intent Receipt
4. Persistent Messages: Sticky Broadcasts
5. Insecure Storage
6. Insecure Communication
7. Overprivileged Applications

1. Intent Spoofing

- Attack: Malicious app sends an Intent, resulting in data injection/state change
- Arises when components are public and do not require senders to have strong permissions

```
<receiver android:name=“my.special.receiver”>
  <intent-filter>
    <action android:name=“my.intent.action” />
  </intent-filter>
</receiver>
```

1. Example



Malicious Injection App

Malicious Component

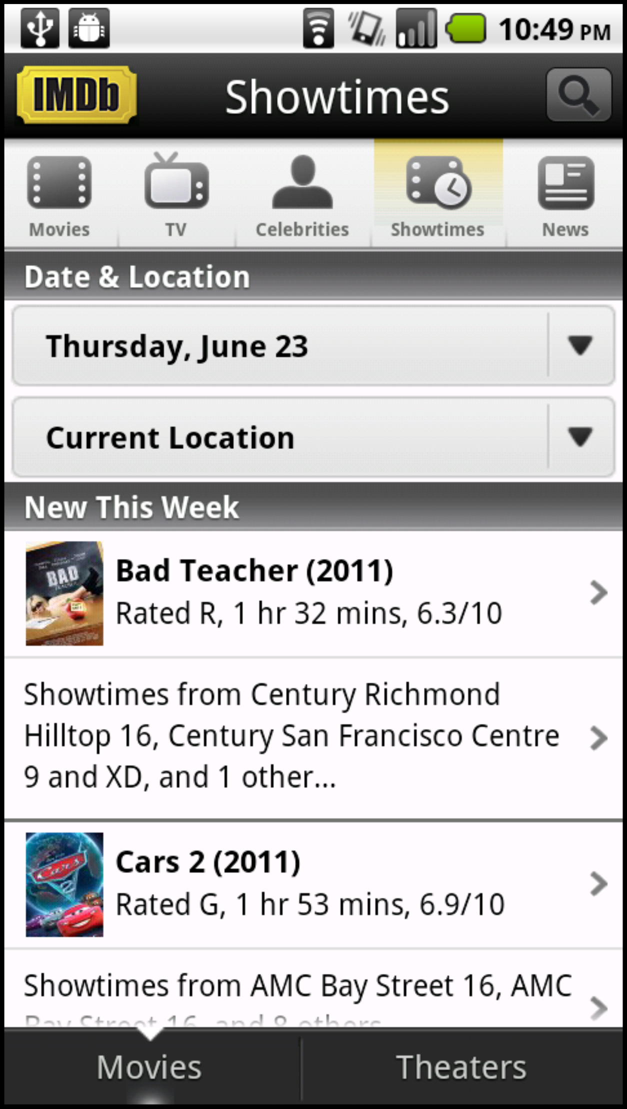
Action: *showtimesNoLocationError*

IMDb App

Handles Action: *willUpdateShowtimes*, *showtimesNoLocationError*

Results UI

Receiving Implicit Intents makes the component public



Date & Location

Thursday, June 23

Current Location

New This Week

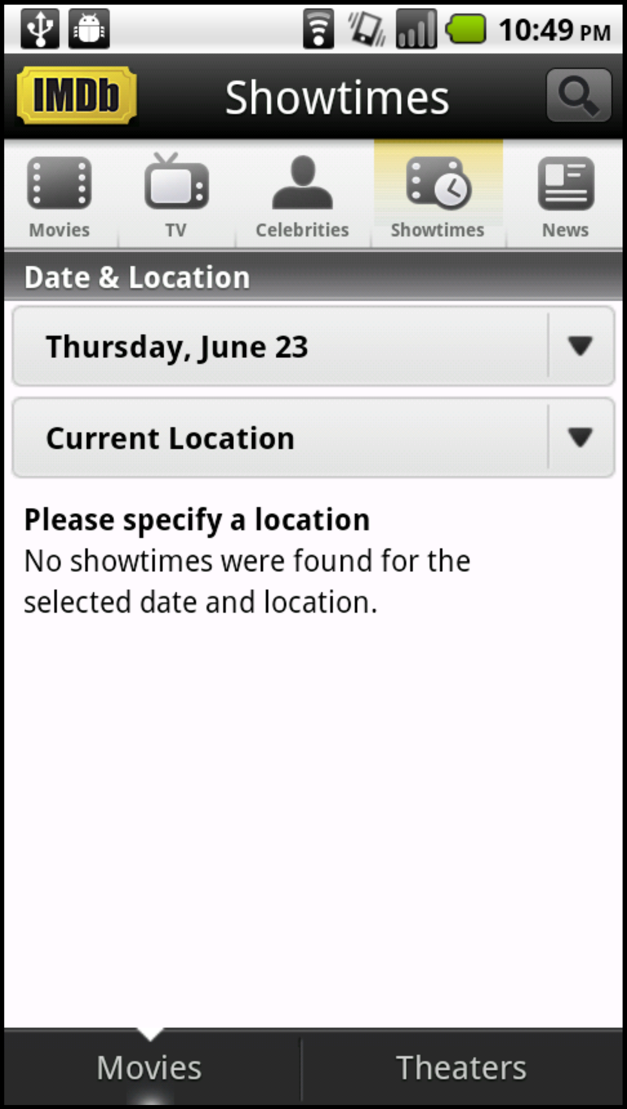
Bad Teacher (2011)
Rated R, 1 hr 32 mins, 6.3/10

Showtimes from Century Richmond Hilltop 16, Century San Francisco Centre 9 and XD, and 1 other...

Cars 2 (2011)
Rated G, 1 hr 53 mins, 6.9/10

Showtimes from AMC Bay Street 16, AMC Bay Street 16, and 8 others

Typical case



Date & Location

Thursday, June 23

Current Location

Please specify a location
No showtimes were found for the selected date and location.

1. Recommended Fix

```
<receiver android:name=“my.special.receiver”
    android:exported=false>
  ...
</receiver>
```

or

```
<receiver android:name=“my.special.receiver”
    android:exported=true
    android:permission=“my.own.permission”>
  ...
</receiver>
```

2. Query String Injection

- Unlike SQL injection, SQLite string injection allows malicious users to view unauthorized records, but not to alter the database
- Query string injection occurs when:
    1. Data enters a program from an untrusted source
    2. The data is used to dynamically construct a SQLite query string

2. Example

```
c = invoicesDB.query(
    Uri.parse(invoices),
    columns,
    "productCategory = '" +
        productCategory + "' and
        customerID = '" + customerID + "'",
    null,
    null,
    null,
    "'" + sortColumn + "'asc",
    null
);
```

2. Example

productCategory = “Fax Machines”
customerID = “12345678”
sortColumn = “price”

```
select * from invoices
   where productCategory = 'Fax Machines' and
   customerID = '12345678'
   order by 'price' asc
```

Returns invoice records for ONE customer

2. Example

productCategory = “Fax Machines’ or productCategory = \ “”
customerID = “12345678”
sortColumn = “\” order by ‘price”

```
select * from invoices
   where productCategory = ‘Fax Machines’ or
   productCategory = "' and customerID =
      '12345678' order by '"
   order by 'price' asc
```

Returns invoice records for ALL customers

2. Recommended Fix

Use parameterized queries!!!

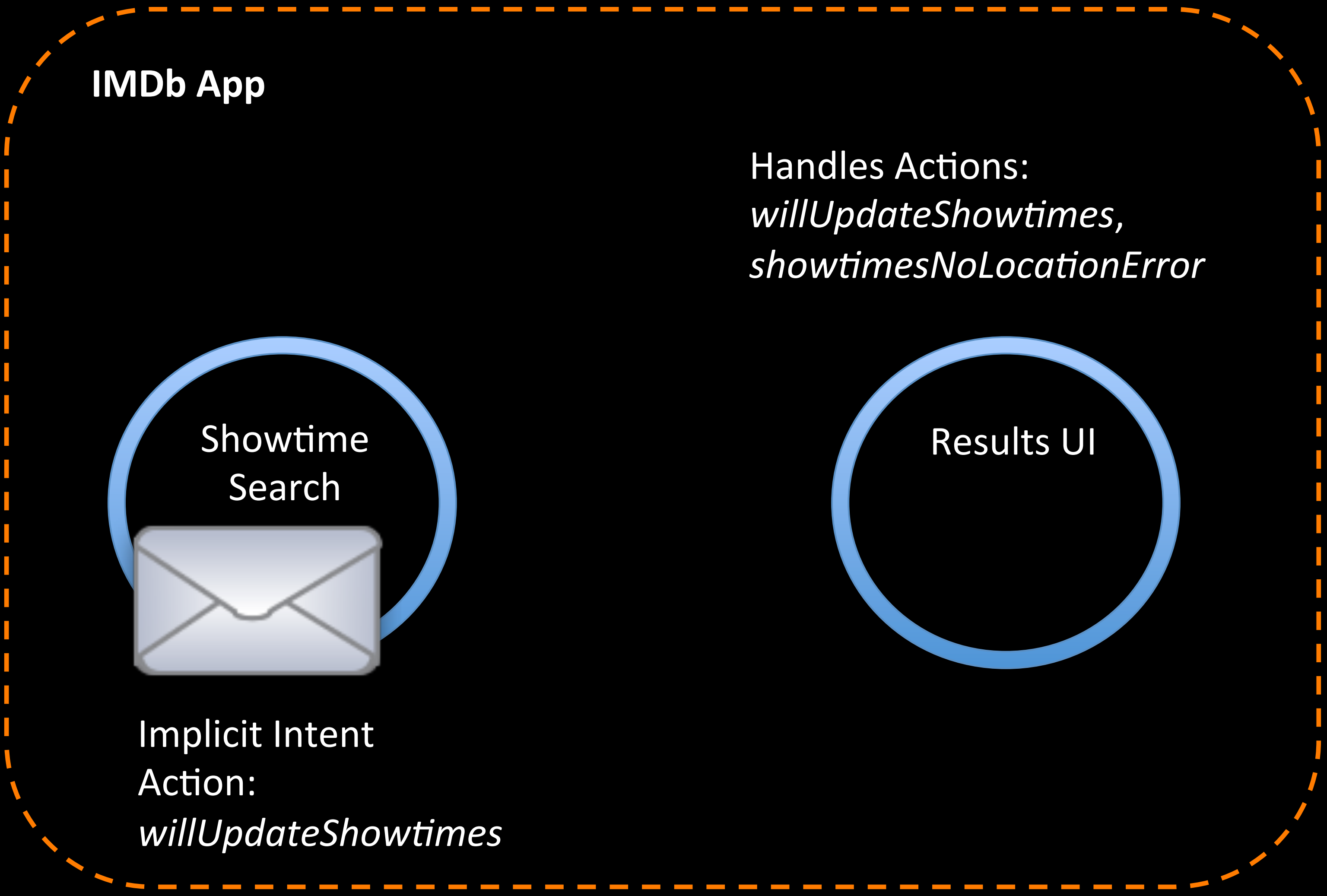
```
c = invoicesDB.query(
    Uri.parse(invoices),
    columns,
    "productCategory = ? and customerID = ?",
    {productCategory, customerID},
    null,
    null,
    "'" sortColumn + "'asc", null
);
```

3. Unauthorized Intent Receipt

- Attack: Malicious app intercepts an Intent
- Arises when Intents are implicit (public) and do not require receiving components to have strong permissions
- Can leak sensitive program data and/or change control flow

```
Intent i = new Intent();
i.setAction(“my.special.action”);
[startActivity|sendBroadcast|startService](i);
```

3. Example



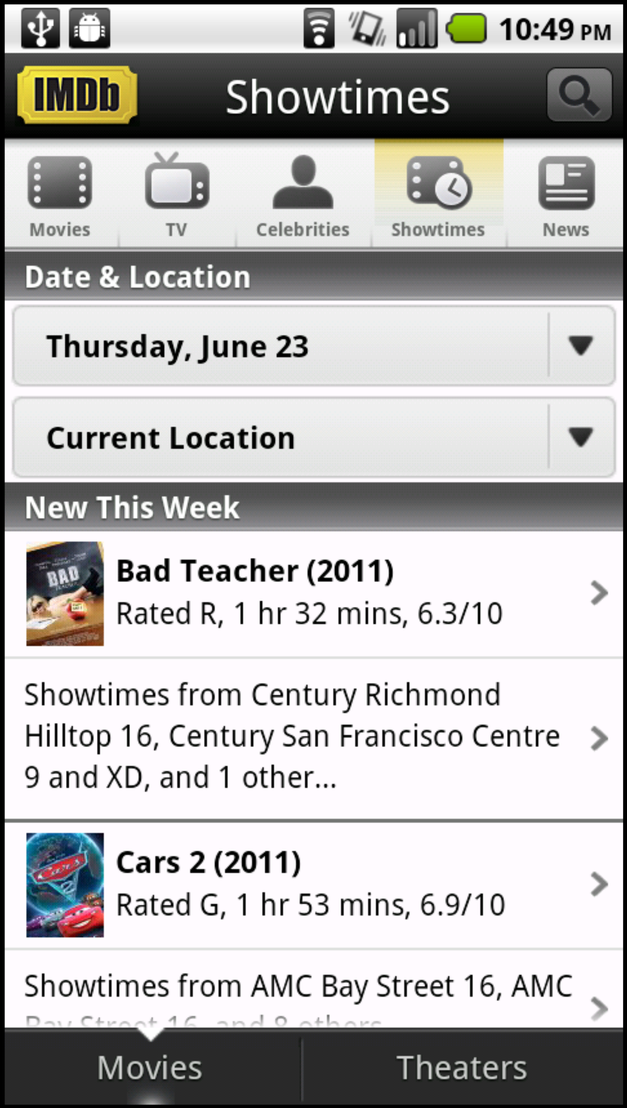
IMDb App

Handles Actions:
willUpdateShowtimes,
showtimesNoLocationError

Showtime Search

Results UI

Implicit Intent
Action:
willUpdateShowtimes



Date & Location

Thursday, June 23

Current Location

New This Week

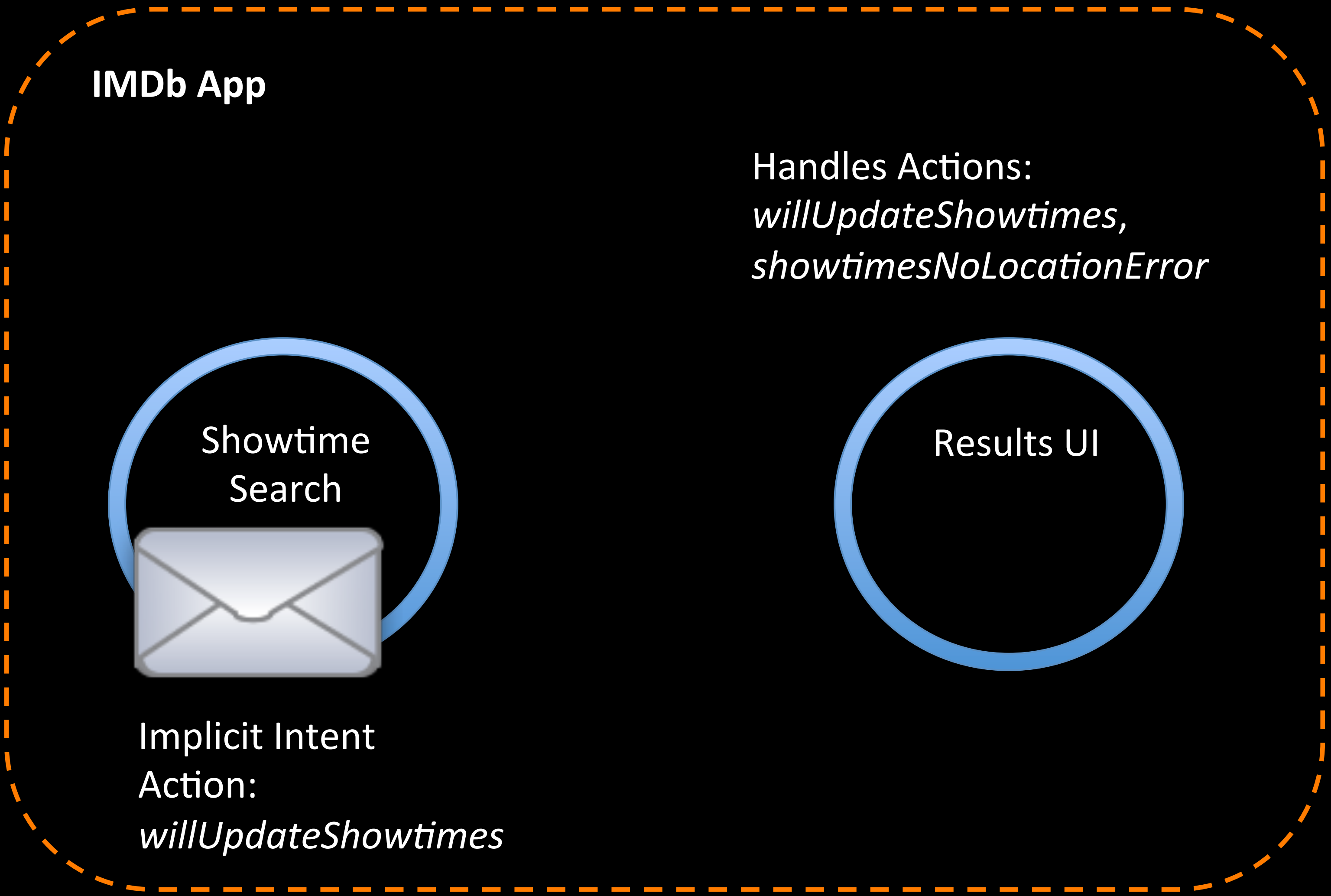
Bad Teacher (2011)
Rated R, 1 hr 32 mins, 6.3/10

Showtimes from Century Richmond Hilltop 16, Century San Francisco Centre 9 and XD, and 1 other...

Cars 2 (2011)
Rated G, 1 hr 53 mins, 6.9/10

Showtimes from AMC Bay Street 16, AMC Bay Street 16, and 8 others

3. Example



IMDb App

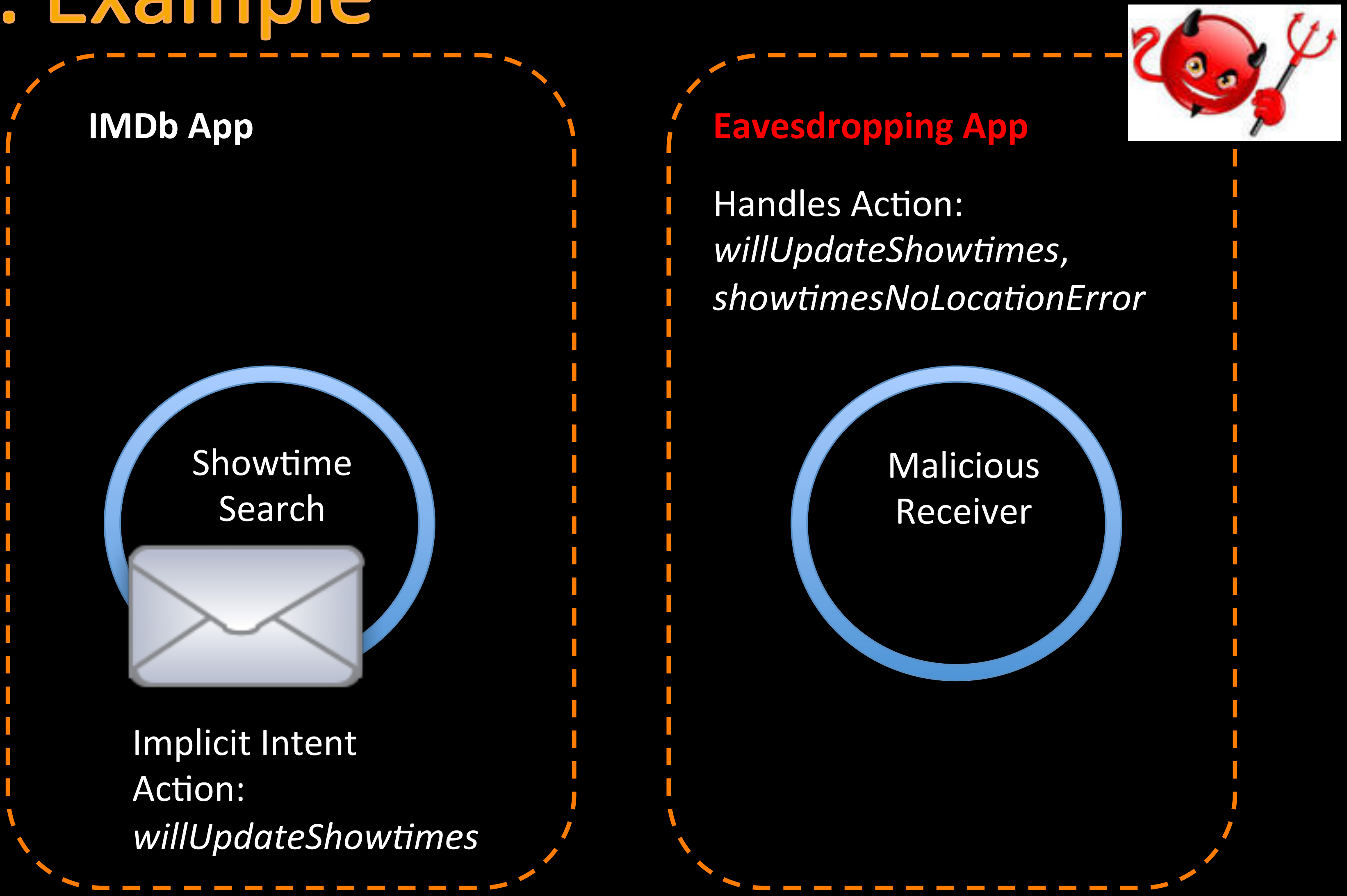
Handles Actions:
willUpdateShowtimes,
showtimesNoLocationError

Showtime Search

Results UI

Implicit Intent
Action:
willUpdateShowtimes

3. Example



IMDb App

Showtime Search

Implicit Intent Action: *willUpdateShowtimes*

Eavesdropping App

Handles Action: *willUpdateShowtimes*, *showtimesNoLocationError*

Malicious Receiver

Sending Implicit Intents makes communication public

3. Recommended Fix

```
Intent i = new Intent();
i.setClassName(“some.pkg.name”,
     “some.pkg.name.TestDestination”);
```

or

```
Intent i = new Intent();
i.setAction(“my.special.action”);
sendBroadcast(i, “my.special.permission”);
```

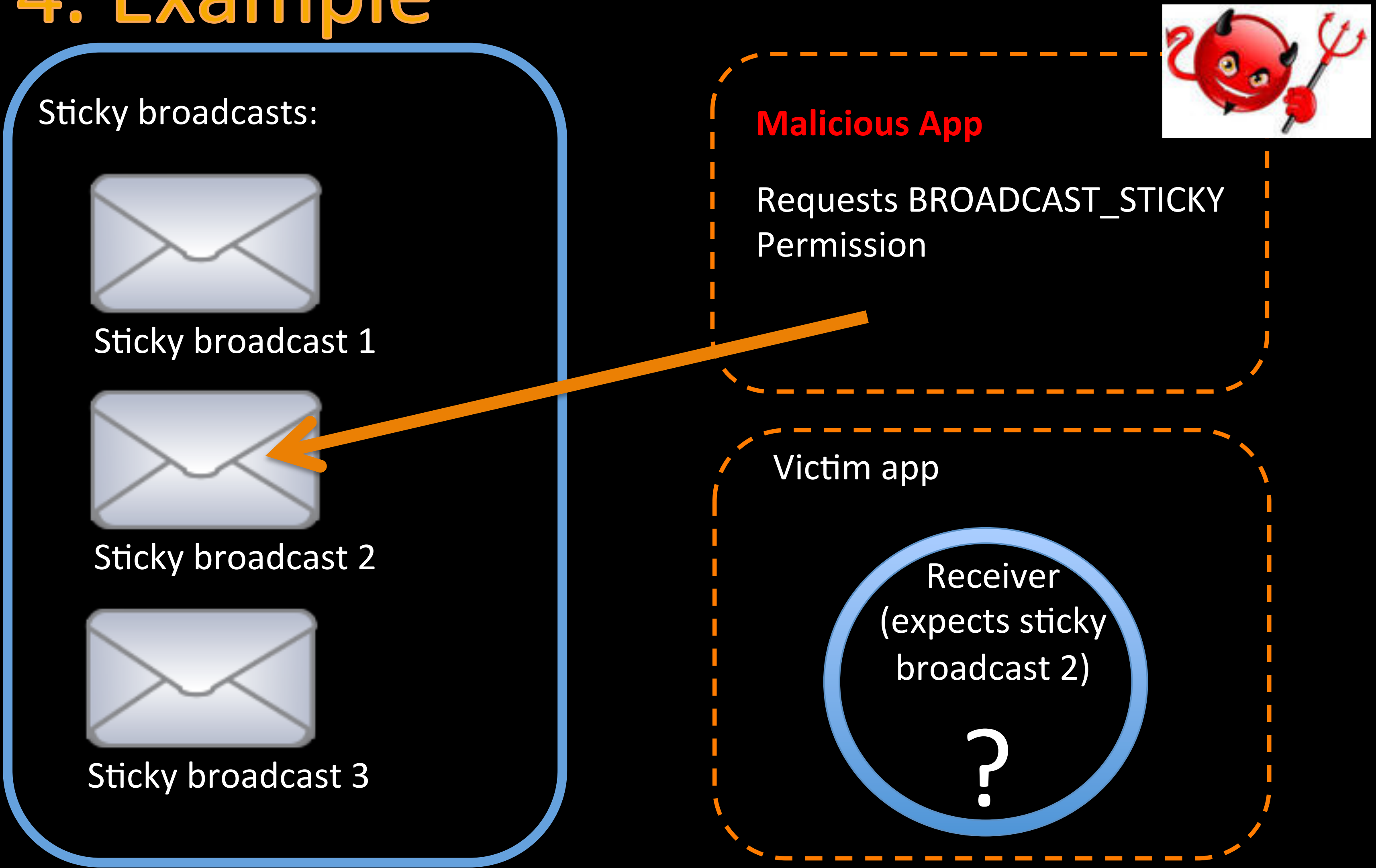
4. Persistent Messages: Sticky Broadcasts

- Broadcast Intent
  - One-to-many message
  - Delivered to all components registered to receive them
- "Sticky" Broadcast Intents are broadcasts that persist
  - Remain accessible after they are delivered
  - Re-broadcast to future Receivers

4. Problems with Persistent Messages

- Cannot be restricted to a certain set of receivers (cannot require a receiver to have a permission)
- Accessible to any receiver, including malicious receivers
- Can compromise sensitive program data
- Stays around after it has been sent
  - But anyone with BROADCAST_STICKY permission can remove a sticky Intent you create

4. Example



Sticky broadcasts:

Sticky broadcast 1

Sticky broadcast 2

Sticky broadcast 3

Malicious App

Requests BROADCAST_STICKY Permission

Victim app

Receiver (expects sticky broadcast 2)

?

Newly connected receiver will be unaware of the change

4. Recommended Fix

- Use regular broadcasts protected by the receiver permission instead, if possible
- Thoroughly scrutinize data in broadcasted messages

5. Insecure Storage

- Files on the SD Card are world-readable
- Files stay even after application that wrote them is uninstalled
- Can compromise sensitive program data
  - Passwords
  - Location
  - SMS
  - Etc.

5. Examples

- Skype for Android exposes your name, phone number, chat logs and more
- Citibank iPhone app “accidentally” saved account numbers, bill payments and security access codes in a secret file
- iPhone location file contains information about your location

5. Recommended Fix

- Write to an application's SQLite database
- Write to the device's internal storage and use Context.MODE_PRIVATE

6. Insecure Communication

- Be careful of leaking sensitive data through HTTP connections
- When using WebViews, connect to HTTPS when possible
- Treat your mobile app as you would a web app
- Don’t send passwords in the clear

FORTIFY®
An HP Company

6. Examples

- Twitter: Tweets are sent in the clear
- Google Calendar: Calendar traffic is sent in the clear
- Facebook: Despite having a fully encrypted traffic option on the web app, the mobile app sends everything in the clear

FORTIFY® An HP Company

Courtesy: https://freedom-to-tinker.com/blog/dwallach/things-overheard-wifi-my-android-smartphone

7. Overprivileged Applications

- Overprivileged applications – applications that request more permissions than the app actually requires

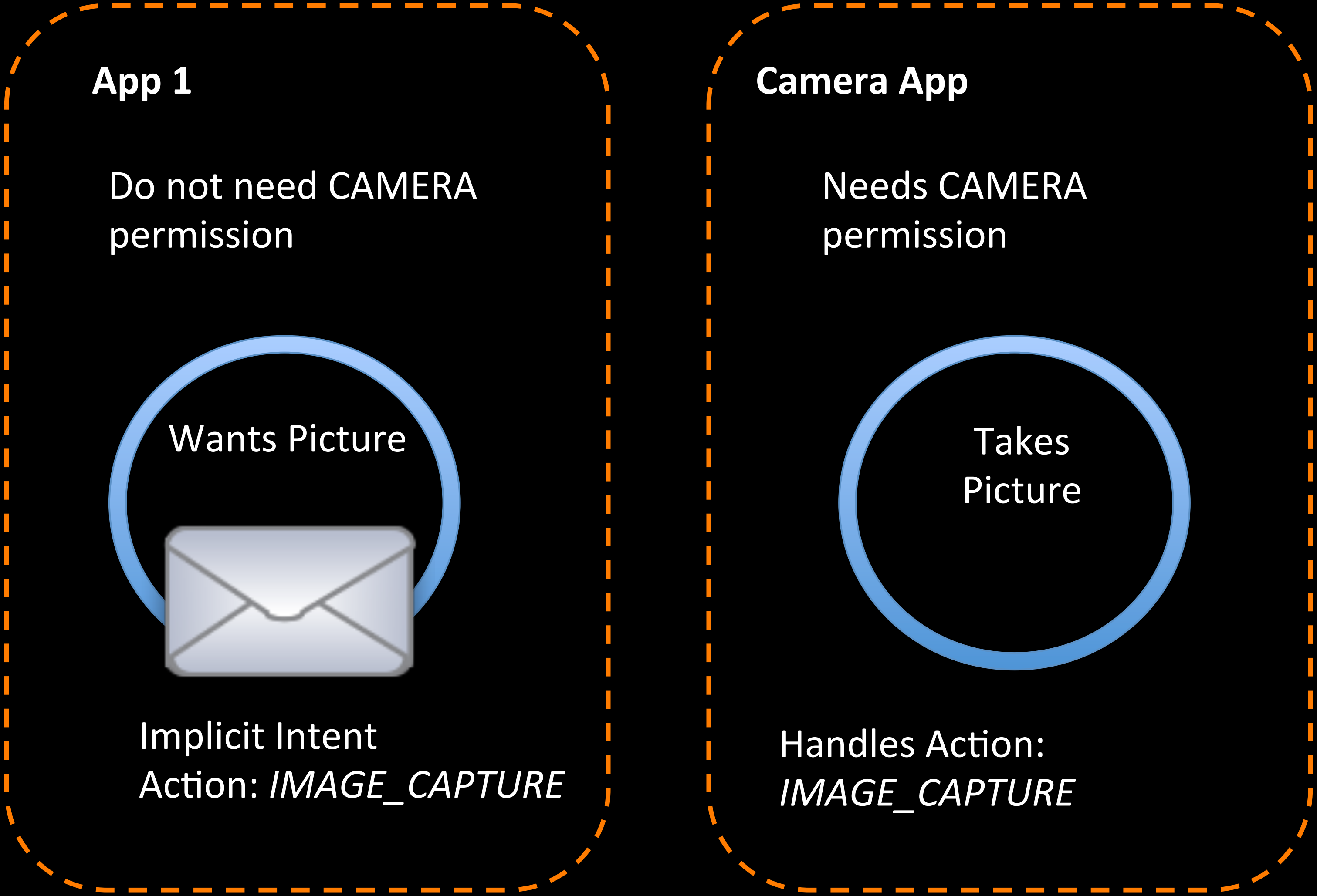
7. Why is this dangerous?

- Violates the principle of least privilege
- Any vulnerability may give the attacker that privilege
- Users may get accustomed to seeing and accepting unnecessary permission requests from third party applications

7. How can this occur?

- Common causes
  - Confusing permission names
  - Testing artifacts
  - Using deputies
  - Error propagation through message board advice
  - Related methods

7. Example



App 1

Do not need CAMERA permission

Wants Picture

Implicit Intent
Action: *IMAGE_CAPTURE*

Camera App

Needs CAMERA permission

Takes Picture

Handles Action:
IMAGE_CAPTURE

Empirical Results Analyzing Applications Built on

GOOGLE ANDROID

FORTIFY®
An HP Company

Summary of Results

Vulnerability Type	% of Apps that are Vulnerable
Intent Spoofing	40%
Unauthorized Intent Receipt	50%
Overprivileged Applications	31%

Challenges

- Obtaining application source code
  - – Dedexers available fail to generate *valid* Java
  - – Many applications are not open source
- Coding conventions
  - – Callbacks and other implicit control flow are a challenge for traditional static analysis techniques
- Documentation
  - – Google provides little documentation, which is often incomplete or out-of-date

Lacking Documentation

- Analysis of overprivileged applications showed that:
  - Android 2.2 documents permission requirements for only 78 out of 1207 API calls
  - 6 out of 78 are incorrect
  - 1 of the documented permissions does not exist

Vulnerability Identification

- Of the 7 vulnerabilities presented:
  - 5 vulnerability categories currently can be identified by Fortify's SCA tools
  - 4 vulnerability categories currently can be identified by UC Berkeley's tools
  - 6 categories will be integrated into the current tools

Related Work

- Adrienne Porter Felt, David Wagner, UC Berkeley ['11] - Overprivilege
- Will Enck, Penn State ['09] – information leakage through Broadcast Intents
- Jesse Burns ['09] – other common developers' errors
- Dan Wallach – WiFi leaks

Conclusion

- Android has its own set of security pitfalls
- Static analysis can help developers avoid these problems
- UC Berkeley and Fortify are working to incorporate state-of-the-art static analysis into Fortify's tools

Seven Ways to Hang Yourself with Google Android

Yekaterina Tsipenyuk O'Neil

Principal Security Researcher

Erika Chin

Ph.D. Student at UC Berkeley