JAVAFORUM 2011
stuttgart

Safety First - Android sicher programmieren!

Benjamin Reimold & Stephan Linzner

7th July, 2011

Introducing

Stephan Linzner

Freelance Software Engineer

Mobile Developer

Founder of Stuttgart GTUG

Contact:

@onlythoughtwork
XING, Facebook
onlythoughtworks@gmail.com

Benjamin Reimold

Bachelor-Thesis 2011
(DH Stuttgart)

Mobile Developer

Member of Stuttgart GTUG

Contact:

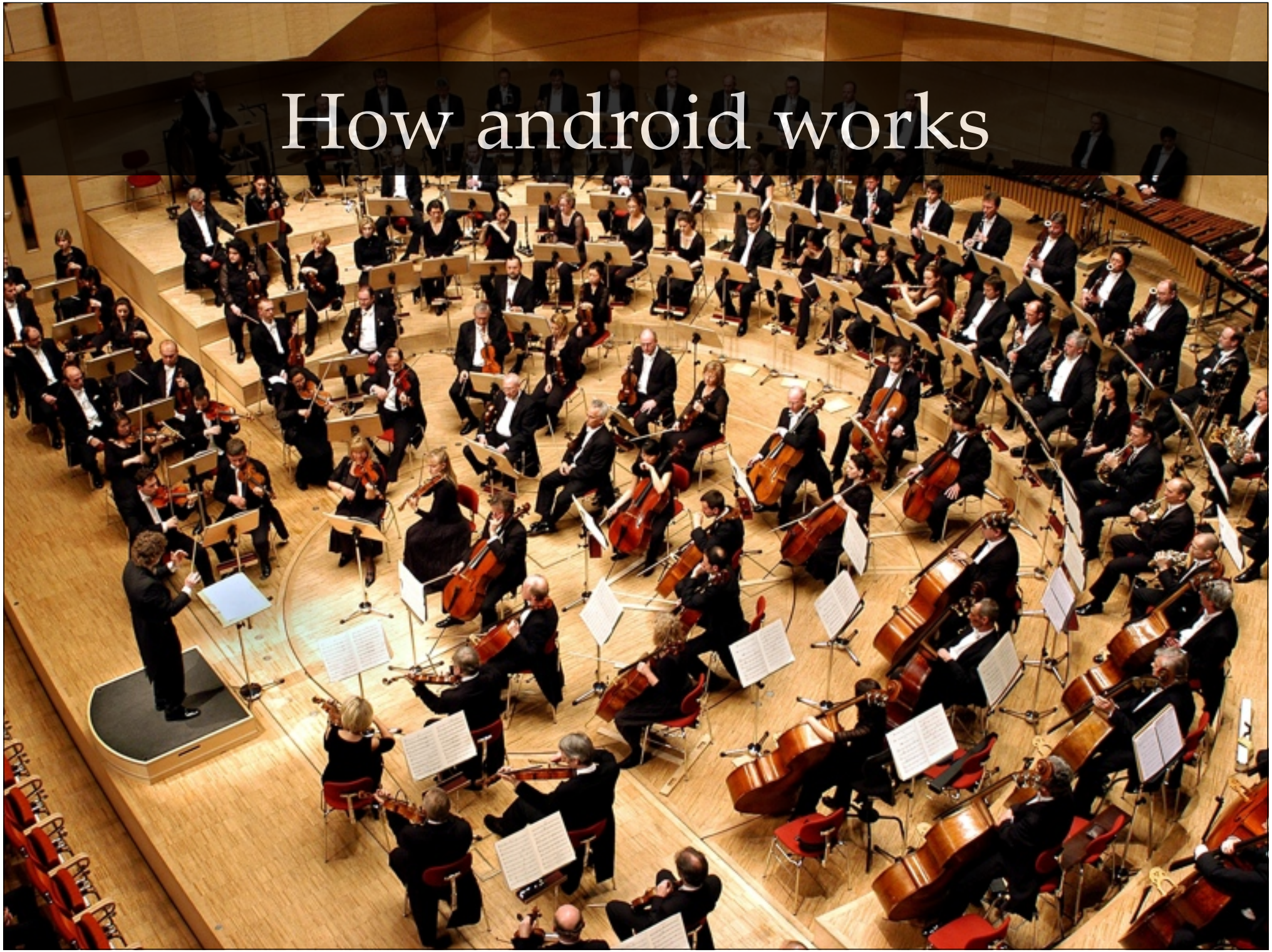
@elektrojunge
XING
benjamin.reimold@aformatik.de


SAFETY FIRST
You never know when a cat is packed with explosives.
ICANHASCHEEZBURGER.COM

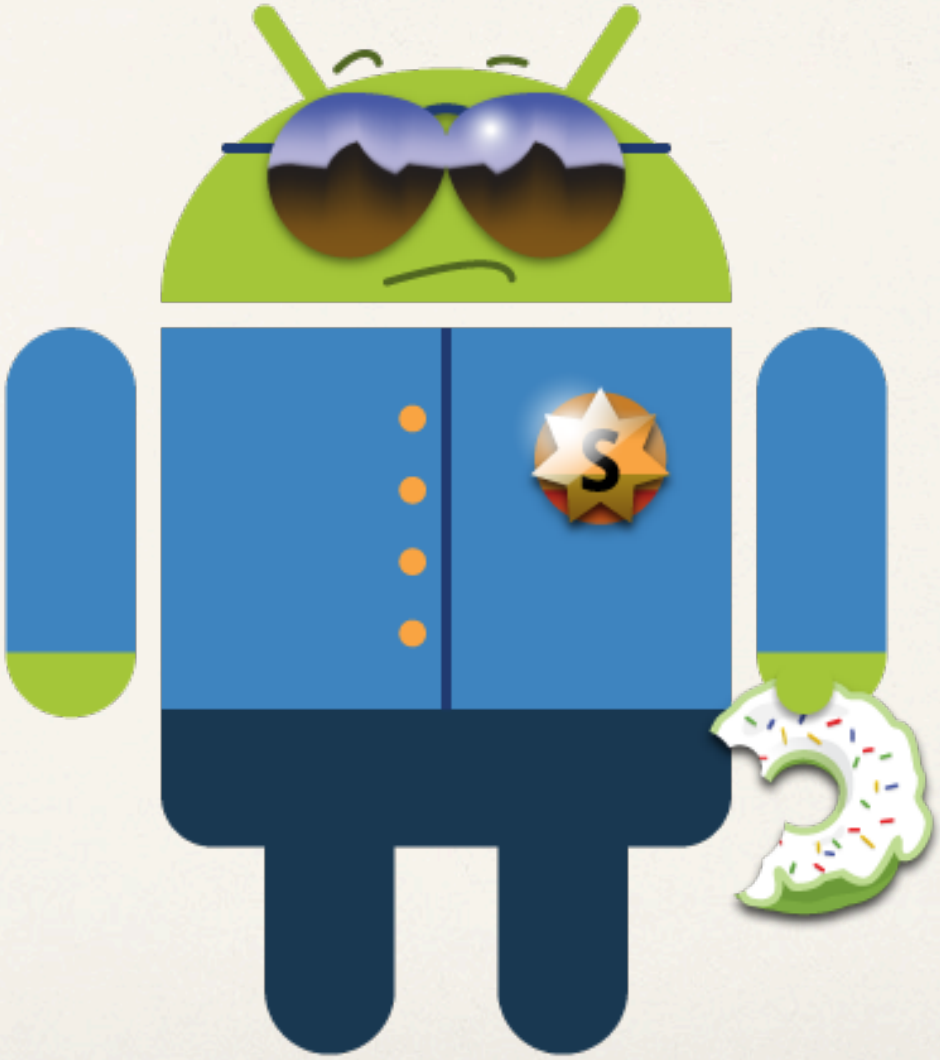
Agenda

- The android security model
- How to obtain information about installed apps
- Responsibility ahead!
- We‘re not done, yet!
- Conclusion

How android works



Android security model



Coarse-grained security model

- Process isolation enforced by underlying linux kernel
- Desktop == single UID
- Android == UID per application
- Components always launched with UID of application owner
- Applications signed with the same key can run with the same UID
- Communication of android components via Binder IPC

Coarse-Grained security model

- Sandboxing
  - Resources can be accessed by the owning application only
  - Each application running in it‘s own VM
  - Binder IPC to relax strict process boundaries
  - (Broadcast-) Intents, Services, Content Providers, AIDL interfaces to exchange data

Coarse-grained security model

- Signing of applications
  - Android uses a new reputation approach for code signing
  - Responsibility of the developer
  - Developers build trust by building good applications

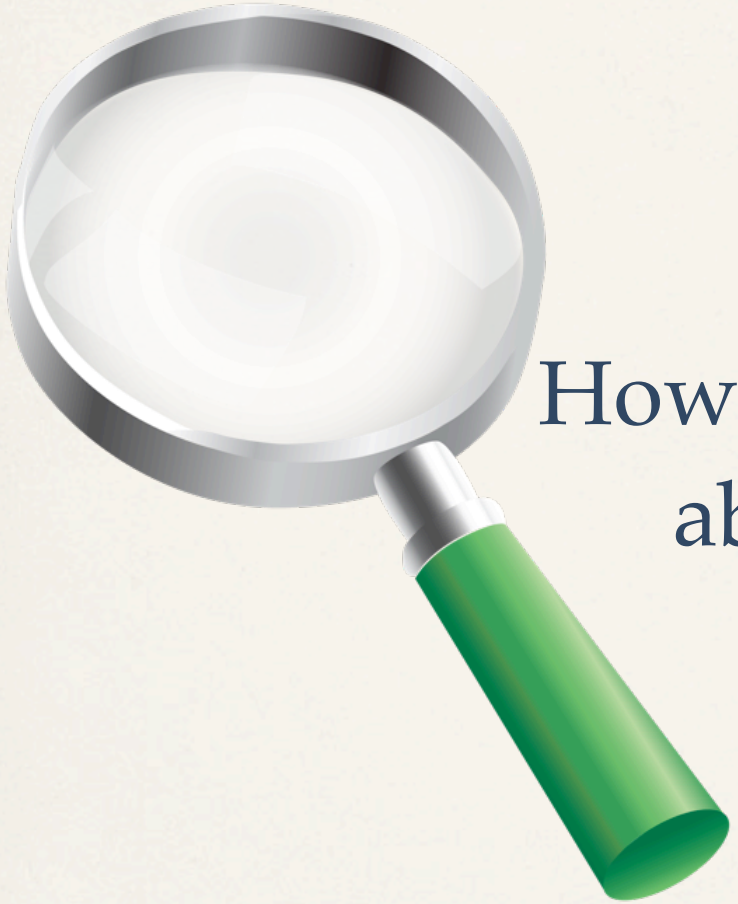
Fine-grained security model

- Permissions
  - "A permission is a mechanism that enforces restrictions on the specific operations that a particular process can perform"
  - End-user model

Permission Types

- `adb shell pm list permissions -s`

- System permissions `<uses-permission />`
- Custom permissions `<permission />`
- Permission groups `<permission-group />`
- Permission trees `<permission-tree />`



How to obtain information about installed apps

PackageManager utility methods

- Can be retrieved by calling `Context.getPackageManager()`
- Gather information about installed applications
  - `getInstalledPackages(int flags)`
  - `getInstalledApplications(int flags)`
  - `getLaunchIntentForPackage(String packageName)`

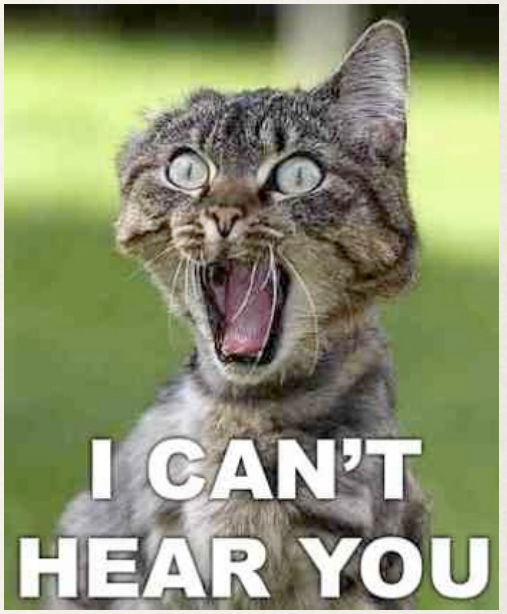
ActivityManager utility methods

- Gather information about running tasks
- `getRecentTasks(int maxNum, int flags)` if the App has `GET_TASKS` permission
- Use `ActivityManager.RecentTask` class to get the base Intent of an Activity


RESPONSIBILITY
AHEAD

Hello World Activity Manager!

- Components are exported when...
  - Declaring an IntentFilter
  - Exporting a component explicitly using `android:exported`
- Good News: Components are private by default



Intents

- Intents don’t enforce security policy themselves, they are just messengers
- Never put sensitive data i.e. password "into" an Intent!
- **Tip: Limit the scope of your Intent by adding categories!**

IntentFilters

- `IntentFilters` do not filter malicious Intents!
- Attackers can raise priority of their `IntentFilter`
  - `IntentFilter.setPriority(int priority)`
  - `android:priority` attribute
- Be specific! Add Actions/Categories and Data filters to your IntentFilters to permit Intents to pass and save you from unwanted consequences

Activities

- Use `<activity android:permission="de.otw.android.HARM_USER_DATA"/>`
- Permissions are checked during `Activity.startActivity()` or `Activity.startActivityForResult();`
- If the caller does not have the required permission then `SecurityException` will be thrown (same as `Context.enforceCallingPermissions()`)
- **Tip: Show a dialog to the user!**

Services

- Client:
  - Use `Intent.setComponent()` to explicitly specify the service
  - Beware when using Binder interfaces!
  - Check permission in ServiceConnection callback using the `PackageManager` before the exchange of sensitive data
  - Control access with Binder utility methods + `Context.check*Permission()` methods when using an `IInterface` (i.e. with `RemoteCallableList`)

Services

- Server
  - Enforce permission with `<service android:permission />`
  - Finer access control, when using a Binder and Context utility methods

BroadcastReceiver

- Receiver
  - Enforce permission by using `android:permission` attribute in the `<receiver />` - Tag and `ActivityManager` will take care
- Sender
  - Take control who will receive your Intent --> Common source of data leakage
  - Enforce a permission by using `Context.sendBroadcast(Intent intent, String receiverPermission)`
  - **Tip: Don´t use sticky broadcasts for sensitive data!**

ContentProvider

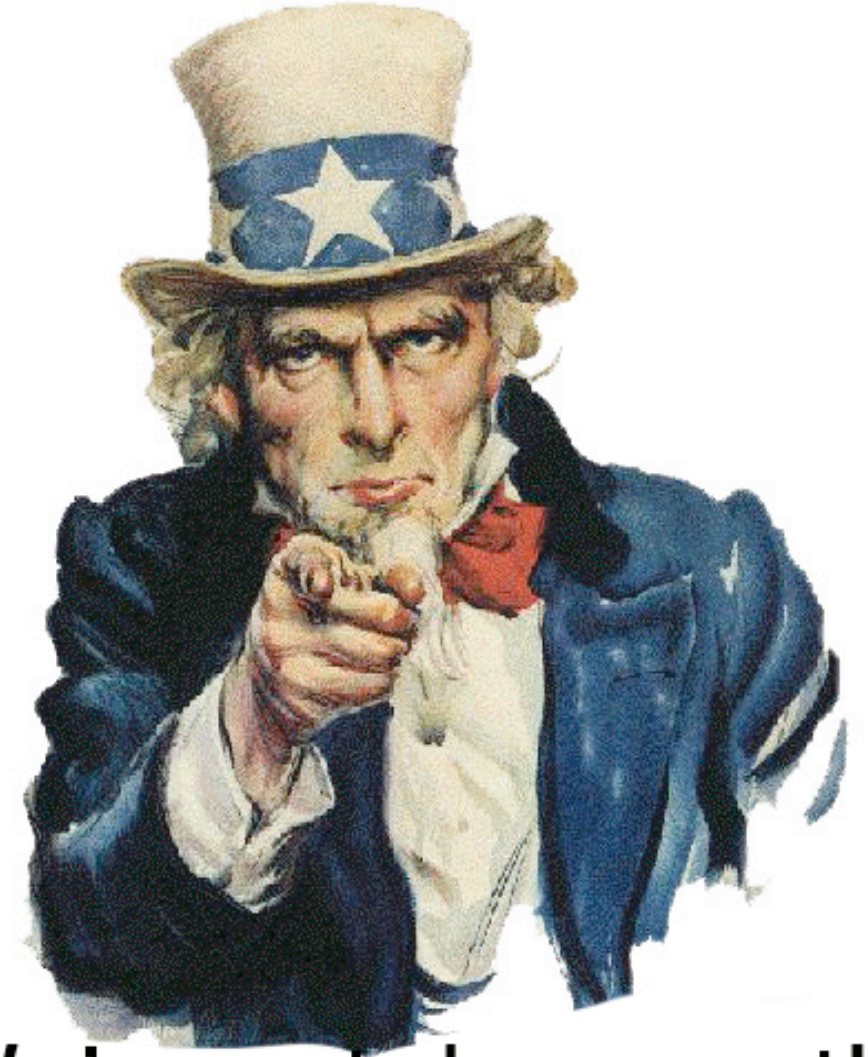
- Beware when using `ContentProvider` internally, **explicitly** set `android:exported="false"`
- Separate read and write permissions
  - `android:readPermission`
  - `android:writePermission`
  - `android:permission`
- Use `<path-permission>`-Tag to control access to specific uris

ContentProviders

- Enable uri access for all resources with `android:grantUriPermission="true"`
- Use `<grant-uri-permission />` to gain more granular control over
- Normal use via `Intent.FLAG_GRANT_READ_URI_PERMISSION` or `Intent.FLAG_GRANT_WRITE_URI_PERMISSION`
- Implement your own policy with `grantUriPermission()` & `revokeUriPermission()`

ContentProviders

- **Tips:**
  - Always check any received data (Intents, Binder Interfaces), which you use in `ContentProvider`/`SQLiteDatabaseHelper` queries
  - Clearly separate SQL Statement and the data it contains, use parameterized queries!
  - Make your selection fields final to avoid accidental contamination
  - Completely avoid selections, defining `CONTENT_URIs` only!



We’re not done, yet!

File I/O

- Files, DB and shared preferences can be created with a Permission (>> SKYPE!)
  - `Context.MODE_PRIVATE`
  - `Context.MODE_WORLD_READABLE`
  - `Context.MODE_WORLD_WRITEABLE`

File I/O

- **Tips:**
  - Think about the consequences when creating files (sensitive data, permissions), especially if making a file world readable!
  - Ask the user to grant a permission if you do so!
  - Don´t store sensitive data on the SD Card/DB unless it is encrypted! (store the key inside the private file area)


We Can Do It!
WAR PRODUCTION CO-ORDINATING COMMITTEE

You are responsible!

- Consider how you will keep user's data safe!
- Protect all user input & data to prevent data leakage!
- Require a permission or show a dialog to the user that another component is about to access his data!
- Deal with bad input parameters (i.e. Intent data, queries on ContentProvider)!
- Minimize application permissions because it minimizes the consequences of potential security flaws!

Android in Stuttgart

Article in Android360 2.11!

- Stuttgart GTUG
  - http://stuttgart.gtugs.org
- SIG Android
  - http://jugs.de/sig-android.html