Android

“Easy” to develop apps

Powerful

Omnipresent

Sophisticated IDE support & guides
Android app security

Complicated!

Knowledge is spread!
no centralized comprehensive help resource

Numerous threats!
privacy leak, data theft, denial of service, ...
Android security code smells

70.49% of apps have 1 concurrent smell, 19.95% have 2, 4.92% have 3, 3.14% have 4, and 1.09% have 5 or more concurrent smells.
The solution?

Build all the knowledge into the Android Studio IntelliJ IDE!
The solution?

Build all the knowledge into the Android Studio IntelliJ IDE!

... *but is that really a good idea?*
The solution?

Build all the knowledge into the Android Studio IntelliJ IDE!

... but is that really a good idea?

Yes, but the IDE must assist the developer!
Smell reports
+
Interactive feedback
=
Quickfixes
Not as easy as it seems

How to ....

... gather contextual information?
... design the UI?
... create reasonable workflows?
End User License Agreement

Please select your country.

United States

Please read and accept the following agreement to setup your profile.

End User License Agreement

NOTICE TO ALL USERS: PLEASE READ THIS CONTRACT CAREFULLY. BY CLICKING THE ACCEPT BUTTON OR INSTALLING THE SOFTWARE, YOU (EITHER AN INDIVIDUAL OR A SINGLE ENTITY) AGREE THAT THIS AGREEMENT IS ENFORCEABLE LIKE ANY WRITTEN CONTRACT SIGNED

Print  Privacy Policy

Accept  Decline
Example #01 - Missing Protection Level

Use case:
1) Permission limits access to feature
2) Another app requests permission to use that feature

A permission’s protection level defines the access scope:

- **normal** = automatically grants everything (default!)
- **dangerous** = user grants or denies permission
Example #01 - Missing Protection Level

Example:

```xml
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.example.demo" xmlns=""/>

<application.../>

<permission
    android:name="samplePermission"
    android:description="an example permission"
    android:protectionLevel="dangerous"/>

</manifest>
```
Example #01 - Missing Protection Level

Necessary considerations for the mitigation:

1) Detect missing protection level
   
   *How to detect incorrect protection levels?*

2) Protection level is a developer decision
   
   *Ask developer for context? How?*
Example #02 - Implicit Pending Intent
(Intent → task to be performed by other app)

Use case:

1) An app creates a background task
2) Background task will be executed later

There exist different kinds of intents:

- **implicit** → no target app specified
- **pending** → intent receiver gets permissions of sender
- **implicit pending** → security risk
Example #02 - Implicit Pending Intent

Example:

```java
// Pending intent with implicit intent argument
PendingIntent.getActivity(context, 0, new Intent().setAction("ACTION_VIEW"), 0);
```
Example #02 - Implicit Pending Intent

Necessary considerations for the mitigation:

1) Make intent explicit

What if target app cannot be inferred?

How to explain the security risk to the developer?
There are more quickfixes...

Persisted Dynamic Permission

Incorrect Protection Level

Unauthorized Intent

Sticky Broadcast

Implicit Pending Intent

Common Task Affinity
IntelliJ in practice...

DEMO!
IntelliJ syntax trees

code

internal representation

```
meth(hello);
```

```
PsiExpressionStatement
  PsiMethodCallExpression:meth(hello)
    PsiReferenceExpression:meth
      PsiReferenceParameterList
        PsiIdentifier:meth
    PsiExpressionList
      PsiReferenceExpression:hello
        PsiReferenceParameterList
          PsiIdentifier:hello
    PsiJavaToken:SEMICOLON
```
IntelliJ syntax trees

AST
Lowest level representation

PSI
Interface to facilitate file manipulations
Inspections, quick fixes

UAST
Unifies Kotlin and Java
Hardly documented
IntelliJ challenges

Lack of documentation

Internal bugs / behavior
  → Debugging the IntelliJ system

IntelliJ architecture
  → Threading rules, ...

Frequent updates
Quickfix evaluation

Still in progress

1) We let the tool run on existing apps
2) We investigate the false positives
Lessons learned

#01: Scope is important!

```java
public boolean notNothing(String str) {
    return str.equals("Nothing");
}
```

There are no known security issues for "String.equals()".
Lessons learned

#02: Start with the essentials, then extend

#03: Know-how takes time

#04: Importance of documentation
Summary

Smell reports
+ Interactive feedback
=
Quickfixes

Example #01 - Missing Protection Level

Use case:
1) Permission limits access to feature
2) Another app requests permission to use that feature

A permission's protection level defines the access scope:
  - normal = automatically grants everything (default!)
  - dangerous = user grants or denies permission

Lessons learned

#02: Start with the essentials, then extend
#03: Know-how takes time
#04: Importance of documentation