FOSD Meeting 2014

Tracking Load-time Configuration Options

Max Lillack
SPL or one App to rule them all?
Challenge

- Apps must handle variability regarding hardware and software
  - Tablet?
  - Bluetooth?
  - NFC?
  - Old Android
  - Current Android?
- There must be features everywhere!?
Current Solution

- Use of (load-time) configuration options
- Use of normal Java variables and control structures
  - No preprocessor
Example

```java
public static ActionBarWrapper getActionBar(Activity activity) {
    if (PreferenceConstants.PRE_HONEYCOMB)
        return new DummyActionBar();
    else
        return new RealActionBar(activity);
}
```

```
public class PreferenceConstants {
    public static final int SDK_INT = Integer.parseInt(Build.VERSION.SDK);
    public static final boolean PRE_ECLAIR = SDK_INT < 5;
    public static final boolean PRE_FROYO = SDK_INT < 8;
    public static final boolean PRE_HONEYCOMB = SDK_INT < 11;
}
```
How to identify configuration options?

• There is no easy way to differentiate between a normal variable and a variable with a configuration value
• Common APIs to access configuration options (Build.VERSION.SDK) are known (from the documentation)
• We track the accessed information from the API through the program
Approach

• Extended static taint analysis
• Basic steps:
  1. Look for access of known configuration API
  2. taint value
  3. track tainted value along control and data flow
  4. check where tainted value is used to include/exclude code

```java
public static ActionBarWrapper getActionBar(Activity activity) {
    if (PreferenceConstants.PRE_HONEYCOMB) {
        return new DummyActionBar();
    } else {
        return new RealActionBar(activity);
    }
}
```

```java
public class PreferenceConstants {
    public static final int SDK_INT = Integer.parseInt(Build.VERSION.SDK);
    public static final boolean PRE_ECLAIR = SDK_INT < 5;
    public static final boolean PRE_FROYO = SDK_INT < 8;
    public static final boolean PRE_HONEYCOMB = SDK_INT < 11;
}
```
Results

• Constraint for each statement
  – Example: Feature A and not Feature B
• Whole functions and classes could be annotated this way
Comparison

- Slicing
  - Slicing would include all statements affected by a config value
  - We only look for optional statements
  - Slicing does not know how a config value affects a statement
  - We know Bluetooth must be enabled and version is $\geq 1.3$
Implementation

- New tool **LOTTRACK**
- Standing on the shoulders of giants:

\[1\] 1 https://github.com/MaxLillack/Lotrack
Android Case Study: What did we learn?

• Configuration options are used by the majority of apps
  – Framework version (SDK) is a popular option
  – Interactions happen but are rare and limited to first order interactions

• Feature localization? Depends …
  – Some apps have whole classes used only by certain configurations
    → Could easily be refactored to feature modules
  – Other uses only affect a single line of code within the app
    → Important info for testing / maintenance
Work Ahead

- LOTRACK currently only supports Boolean variables
  - Falls back to a “in some unknown way related to” for other types
  - We need at least handling of enumerable integer values (like possible versions)
- Limited to standard options
  - We only looked at options from the Android framework
  - What about user-defined options?
- Comparison to other approaches (such as program slicing)