Smelly APIs in Android ICC
2nd presentation

A bachelor thesis by Astrid Ytrehorn
Recap

- Further aspect of security smells need to be analyzed
- Part of effort to spread awareness of security issues in android apps
- Evaluate efficiency and correctness of linting tool
- “Human” factors (contributors etc.): How do they affect security?
- Assess if location of smells shows any patterns
Manual analysis

- High success rate of Tool with 3 smells.
- Interpretation of smells varies from one developer to another.
- Tool correctly reported 48% of cases.
- Comparison of automatic analysis and manual analysis
- False positives

Source: P. Gadient et al.
Contributor affiliation

- F-Droid repo parser written in C#.
- Parses repo index XML for repo URLs.
- Extract all contributors from repo with Octokit API.

Difficulties:
- Rate limitation of API (5000 per h.)
- Not all projects use Github or have source available

Some contributors worked on >10 projects.
Contributor affiliation

- Analysis of data shows correlation between # of contributors and # of smells.
- Trend: More contributors -> more smells.
- Most apps have team size of 1-2 developers
Contributor affiliation

- Further evaluation: Does a developer make the same mistake across multiple projects?
- Data from project contributors put into relation to smells present per project.
- Output: List of smells caused for each developer.

Difficulties:

- Could not see blame for smell for individual developers.
- Assumption: All developers responsible for every smell in project.

- Unibe Framework (Community Explorer) could have yielded similar results.
Contributor affiliation

Average smell contribution per developer where #Projects > 1, #Smells caused > 0

<table>
<thead>
<tr>
<th>Smell</th>
<th>Contribution</th>
</tr>
</thead>
<tbody>
<tr>
<td>SM01: Persisted Dynamic Permission</td>
<td>1.5</td>
</tr>
<tr>
<td>SM02: Custom Scheme Channel</td>
<td>1.38</td>
</tr>
<tr>
<td>SM04: Unauthorized Intent</td>
<td>2.52</td>
</tr>
<tr>
<td>SM05: Sticky Broadcast</td>
<td>1</td>
</tr>
<tr>
<td>SM06: Slack WebViewClient</td>
<td>1.06</td>
</tr>
<tr>
<td>SM07: Broken Service Permission</td>
<td>1</td>
</tr>
<tr>
<td>SM10: Unprotected Broadcast Receiver</td>
<td>1.7</td>
</tr>
<tr>
<td>SM11: Implicit Pending Intent</td>
<td>1.44</td>
</tr>
<tr>
<td>SM12: Common Task Affinity</td>
<td>2.85</td>
</tr>
</tbody>
</table>
App updates

- 33 projects had updates that introduced change in smells.
- Manual evaluation of these 33 projects incl. 3 updates.
- Time consuming due to manual work involved.

- Most updates focused on new functionality.
- Focus on new features and not security.
- Majority of apps introduced new security smells with update, dominant cause implementation of new ICC (Inter-component communication)
  - Data sharing or social interaction
- Developers should be careful with new updates.
Influence of project age and Activity

- Similar to contributor affiliation, C# tool that gets commit metadata:
  - Timestamp
  - Author
  - Commit message

- All commits saved using MongoDB and BSON.

Difficulties:
- Maximum MongoDB page size exceeded (projects with > 1k commits)
- DB fragmented into 5 pieces.
- Github API rate exceeded.
Influence of project age and Activity

- Analysis of extracted data shows that more mature apps tend to have more security issues than younger ones.
- Similarly, apps with more frequent updates tend to have more issues than ones with fewer.

Example: NewPipe
Unreported smells

- Manual analysis using GREP
  - Grep is a terminal application
  - **Globally** Search a **Regular Expression** and **Print**
- Regular expression used on codebase.
- Finds relevant code in project.

Difficulties

- Required a lot of manual evaluation, interpreting code for some smells non-trivial
  - Nested helper functions
  - Declaration of variables outside of functions
  - Arbitrary linebreaks
Unreported smells

- 99 projects assessed.
- SM04 present in nearly all projects.
- SM12 present in all projects.
- SM08 and SM09 never present.
- Others varying degree of presence.

<table>
<thead>
<tr>
<th>Smell ID</th>
<th>Total unreported smells</th>
</tr>
</thead>
<tbody>
<tr>
<td>SM01</td>
<td>96</td>
</tr>
<tr>
<td>SM02</td>
<td>71</td>
</tr>
<tr>
<td>SM03</td>
<td>99</td>
</tr>
<tr>
<td>SM04</td>
<td>2</td>
</tr>
<tr>
<td>SM05</td>
<td>88</td>
</tr>
<tr>
<td>SM06</td>
<td>81</td>
</tr>
<tr>
<td>SM07</td>
<td>95</td>
</tr>
<tr>
<td>SM08</td>
<td>99</td>
</tr>
<tr>
<td>SM09</td>
<td>99</td>
</tr>
<tr>
<td>SM10</td>
<td>17</td>
</tr>
<tr>
<td>SM11</td>
<td>52</td>
</tr>
</tbody>
</table>
Unreported smells

- 3 main reasons for absence of smells:
  - API not used
  - Failed to identify
  - Using securely

- Most smells absent due to API not used.

- Larger portions of SM02 and SM11 mitigated.

- No smell which evaded detection often.
Placement of smells

- Linting tool expanded to include further parameters.
- Report now includes following metadata:
  - Surrounding method
  - Class
  - Java Package
- Problem: Requires linting all projects again.
- SM08 & SM09 never present.
- SM02 & SM12 only in Android Manifest.
- Wrote C# tool that parses all reports and adds all information to an excel file.
Placement of smells - Methods

Statistics of Surrounding Methods per Smell

- Average of Distinct Surrounding Methods
- Maximum of Distinct Surrounding Methods
- Median of Distinct Surrounding Methods
Placement of smells - Classes

Statistics of Classes per Smell

- Average Distinct Classes per Smell
- Maximum Distinct Classes per Smell
- Median Distinct Classes per Smell
Placement of smells - Packages

Statistics of Packages per Smell

Astrid Ytrehorn
Placement of smells - Comparison

Statistics of Location for Smells

- Average of Distinct Surrounding Methods
- Average Distinct Classes per Smell
- Average Distinct Packages per Smell

Astrid Ytrehorn
Conclusions

- Linting tool gives trustworthy reports.
- Unreported smells mostly due to relevant API not being used.
- Larger teams and frequently updated apps have a tendency to be more smell-prone.
- Most apps suffer from 1 smell, fewer than 10% from > 2.
- Apps with large number of smells tend to have the smells distributed in different locations.
  - Number of average distinct methods, classes or packages is decreasing in order.
- Some API bugs in Android make avoiding specific methods necessary.