

Source Analysis of Security Smells in Android Apps

BSc Thesis - Presentation #1

Patrick Frischknecht

28.11.2017



u^b

UNIVERSITÄT
BERN

Agenda

- 1) Relevance of mobile security
- 2) Android security smells
- 3) Bachelors thesis
 - *Familiarization with state-of-the-art*
 - *Implementation of detectors*
 - *Evaluation*

Relevance of Android Security

- 85% of the smartphone market [1]
- More than three million apps [2]

Issues emerge:

- Privacy
- Data sensitivity
 - *E-commerce*
 - *E-banking*
 - *Healthcare*



Android Security Smells

*Symptoms in the code
that indicate the prospect of
security and privacy vulnerabilities*

[03]

Android Security Smells #2

- Exposed Persistent Data
using `MODE_WORLD_READABLE/WRITEABLE`
- Dynamic Code Loading

Bachelors Thesis

I will NOT use insecure and outdated APIs any longer!
I will NOT use insecure and outdated APIs any longer!
I will NOT use insecure and outdated APIs any longer!
I will NOT use insecure and outdated APIs any longer!
I will NOT use insecure and outdated APIs any longer!
I will NOT use insecure and outdated APIs any longer!
I will NOT use insecure and outdated APIs any longer!
I will NOT use insecure and outdated APIs any longer!
I will NOT use insecure and outdated APIs any longer!
I will NOT use insecure and outdated APIs any longer!
I will NOT use insecure and outdated APIs any longer!
I will NOT use insecure and outdated APIs any longer!





Ambitions

- 1) Creation of an easy to use IDE tool
- 2) Detection of a security smells subset
- 3) Evaluation on a set of Android projects

Project Plan

- Existing tools evaluation
 - *Android Lint*
 - *Findbugs Security*
 - *Amandroid*
 - ...
- Selection and extension of a tool
- Automated analysis on a larger set of Android projects

Tool: Findbugs Security

The screenshot displays the FindBugs-IDEA interface within an IDE. The top part shows the project structure for 'InsecureBank2_studio' with the following hierarchy:

- Project
- app
- src
- main
- assets
- java
- com.android.insecurebank2
- ChangePassword

The code editor shows a snippet of Java code:

```
try {  
    jsonObject = new JSONObject(result);  
    acc1 = jsonObject.getString("from");  
    acc2 = jsonObject.getString("to");  
    System.out.println("Message:" + jsonObject.getString("message") + " From:" + from);  
    final String status = new String("\nMessage:" + "Success" + " From:" + from.getText());  
    try {  
        // Captures the successful transaction status for Transaction history tracking  
        String MYFILE = Environment.getExternalStorageDirectory() + "/Statements_" + u;  
        BufferedWriter out2 = new BufferedWriter(new FileWriter(MYFILE, true));  
        out2.write(status);  
        out2.close();  
    }  
}
```

The FindBugs-IDEA FindBugs Analysis Results panel shows the following structure:

- InsecureBank2 (found 14 bug items in 59 classes) more
- Security (14 items)
 - Static IV (2 items)
 - Cipher is susceptible to padding oracle attack (2 items)
 - Cipher with no integrity (2 items)
 - External File Access (Android) (4 items)
 - External File Access (Android) (4 items)
 - Files could be saved to external storage. (4 items)
 - WebView with JavaScript Enabled (Android) (1 item)
 - Potential Path Traversal (File Write) (2 items)
 - Broadcast (Android) (1 item)

The detailed view of the 'External File Access (Android)' issue is shown on the right:

External File Access (Android)

The application write data to external storage (potentially SD card). There are multiple security implication to this action. First, file store on SD card will be accessible to the application having the `READ_EXTERNAL_STORAGE` permission. Also, if the data persisted contains confidential information about the user, encryption would be needed.

Code at risk:

```
file file = new File(getExternalStorageDirectory(), filename);  
fos = new FileOutputStream(file);  
fos.write(confidentialData.getBytes());  
fos.flush();
```

Better alternative:

```
fos = openFileOutput(filename, Context.MODE_PRIVATE);  
fos.write(string.getBytes());
```

References

- CERT: DRD00-J: Do not store sensitive information on external storage [...]
- Android Official Docs: Using the External Storage

A yellow callout box points to the code at risk section with the text: "This gives a longer description of the detected bug pattern".

Tool: Android Lint

- Static source code analysis ***tool for Android***
- Integrated in ***Android Studio***
- Provides ***a lot of built in security checks***
 - *14 different security smells checks built-in*
 - *Especially manifest smells are well covered*

Android Lint: In Action

```
29 public static KeyPair generateKeyPair() throws NoSuchAlgorithmException, NoSuchProviderException {  
30     KeyPairGenerator keyPairGenerator = KeyPairGenerator.getInstance("RSA");  
31     keyPairGenerator.initialize(1024);  
32  
33  
34 }  
35
```

RSA should be initialized with a key size of at least 2048 bits [InsufficientRSAKeySize] [more...](#) (Ctrl+F1)

- Code highlighting
- Tooltips
- Quickfixes

Android Lint: Extension

- Creation of a new detector class
XML or JavaDetector
- Registration in a registry class
- JAR repackaging and copying to Lint folder



Android Lint: Example

Weak Hash Function Detector

- Smell: Weak Crypto Algorithm
- Finds usages of MD5 hash function
MD5 is vulnerable to collision attack

Android Lint: Example

Weak Hash Function Detector

```
public void visitMethod(@NonNull JavaContext context, @Nullable AstVisitor visitor,
                       @NonNull MethodInvocation methodInvocation) {
    JavaParser.ResolvedNode resolved = context.resolve(methodInvocation);
    if (!(resolved instanceof JavaParser.ResolvedMethod)) {
        return;
    }
    JavaParser.ResolvedMethod method = (JavaParser.ResolvedMethod) resolved;
    if (!method.getContainingClass().isSubclassOf(MESSAGE_DIGEST, strict: false)) {
        return;
    }
}
```

```
@Override
public List<String> getApplicableMethodNames() { return Collections.singletonList("getInstance"); }
```

- New JavaScanner looking for **MessageDigest.getInstance("MD5")**

Android Lint: Example

Weak Hash Function Detector

```
private static void checkRequestedHashFunction(@NonNull JavaContext context,
                                               @NonNull MethodInvocation call, @NonNull Node node) {
    if (WEAK_HASH_FUNCTIONS.contains(value)) {
        String message = ""+value+" is considered a weak hash function and should not be used";
        context.report(ISSUE, call, context.getLocation(node), message);
    }
}
```

- Adds a report

Android Lint: Example

Weak Hash Function Detector

```
java.security.MessageDigest md5Digest = null;
try {
    md5Digest = java.security.MessageDigest.getInstance("MD5");
} catch (Exception e) {
    e.printStackTrace();
}
```

MD5 is considered a weak hash function and should not be used in a security critical context

- Visible in Android Studio
- ... and in the HTML report



Android Lint: Evaluation

- Execution on a subset of Android open-source projects
- Manual execution
- Fast & Scalable!



Roadmap

- Investigation of the relation between security smells and required detectors
- Implementation of more detectors
- Expansion of the Android app test set

References

[01]

IDC Annual Smartphone Market Share Report, accessed on 26.11.2017
<https://www.idc.com/promo/smartphone-market-share/os>

[02]

AppBrain Report, accessed on 26.11.2017
<https://www.appbrain.com/stats/number-of-android-apps>

[03]

Ghafari, Mohammad, Pascal Gadiant, and Oscar Nierstrasz. "Security Smells in Android." Source Code Analysis and Manipulation (SCAM), 2017 IEEE 17th International Working Conference on. IEEE, 2017.

Android Lint: Issues

- Only plain static analysis
 - *no taint analysis*
 - *no dynamic analysis*
- Android Lint API is unstable
- Projects must be compiled
(some detectors require class files)
- Executes on source code, thus we need
 - *Open-source projects*
 - *... or decompiled byte-code*

Android Lint: Preliminary Results

