Architectural Risk Analysis for Android Applications

Habilitation Presentation, March 17th, 2021

Dr. Karsten Sohr
TZI – University of Bremen
Center for Computing Technologies

Software Security as an Own Discipline

- Usual security mechanisms such as firewalls, anti-virus software or intrusion detection systems are reactive

- Cause of many security problems: security holes *in software*

- McGraw: Trinity of trouble
  1. Increasing complexity (Windows 8 up to 80 Mio. lines of code?)
  2. Increasing connectivity (SOA, Internet of Things, industrial controllers…)
  3. Extensibility of systems (installation of apps, plugins for browsers)

- Tools and processes to improve software security
  - Security development lifecycle (SDL)

Code Review through Static Code Analysis

- Security analysis of the source code of applications
  - Detection of common programming bugs, such as buffer overflows, SQL-injection- and cross-site-scripting vulnerabilities
  - Automated analysis
- Use of compiler-construction techniques
  - Intermediate representation of the program e.g. by abstract syntax trees, static single assignment (SSA), System Dependence Graphs (SDGs)
  - Data- and control flow analyses
- False positives, false negatives
  - Non-decidability
- Commercial tools: Fortify SCA (for Java), IBM AppScan, Checkmarx, Veracode, Coverity Prevent (for C/C++ code)

Dr. Karsten Sohr, sohr@tzi.de

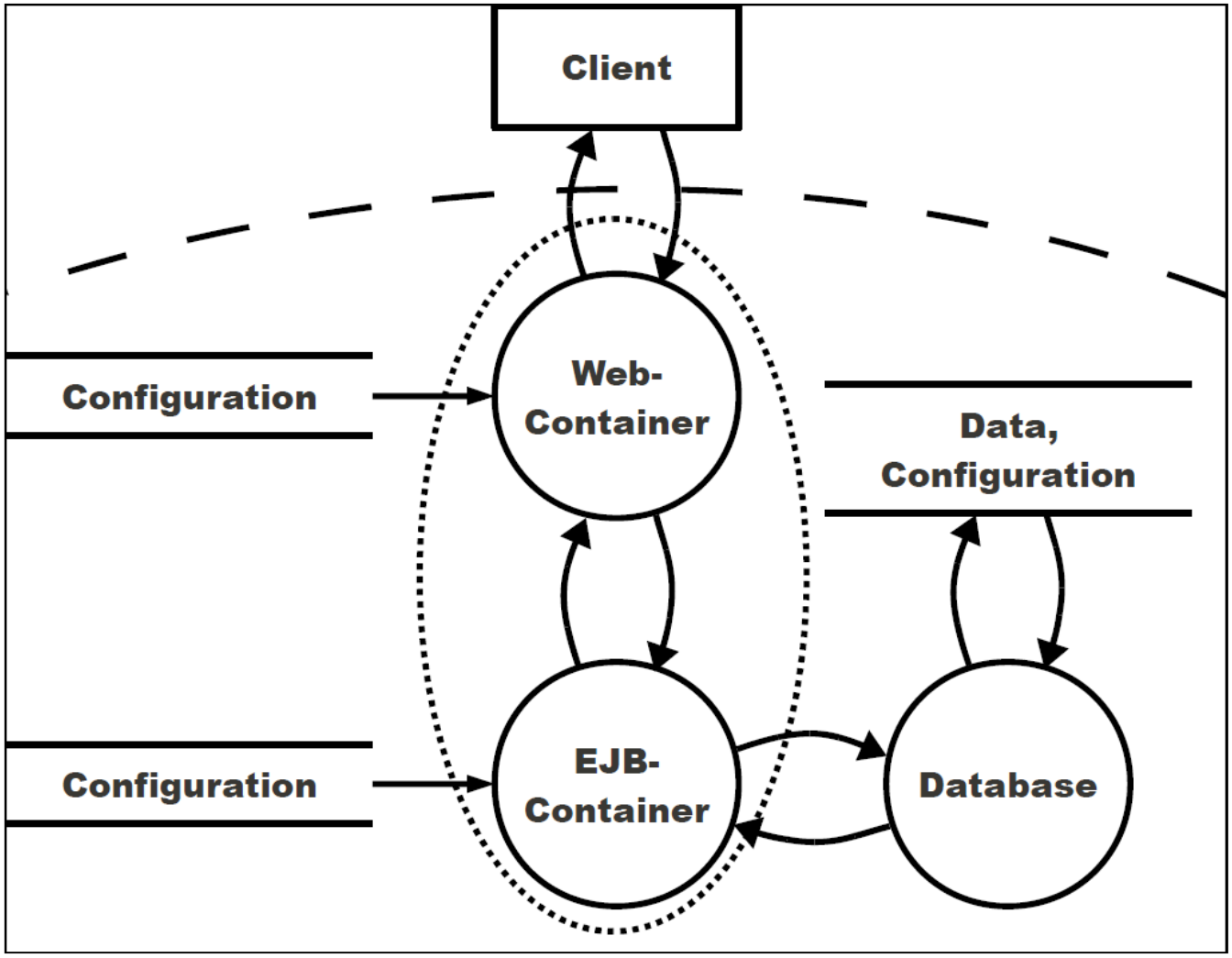
Architectural Risk Analysis as Part of the SDL

- Security analysis of the software architecture
  - At design time
  - Detection of basic security problems (“flaws” vs. “bugs”)
  - Example of flaws: Missing encryption; only encryption, although integrity is required; authorization checks on client side; overprivilege; wrong usage of SW frameworks
- Several approaches, e.g.
  - Threat Modeling/STRIDE from Microsoft
  - Architectural risk analysis (ARA) from McGraw
- Core idea in such approaches:

  Discussion of basic security aspects with the help of diagrams (forest-level overview)

Dr. Karsten Sohr, sohr@tzi.de

Threat Modeling with Dataflow Diagrams



Dr. Karsten Sohr, sohr@tzi.de

Security Holes in Mobile Apps

- Overprivileged apps
- Confused-deputy problems
- Massive vulnerabilities in TLS-client implementation of apps
- Wrongly implemented encryption (e.g., insecure algorithms and crypto modes, insecure key generation)
- Injection of JavaScript code into apps with web functionality
- Faulty usage of software frameworks (Android framework)
- Weaknesses in systems consisting of a remote-control app and a backend

Dr. Karsten Sohr, sohr@tzi.de

Insecure Usage of the Android Framework: Telekom Online Manager

```
Intent localIntent1 =
  new
  Intent("de.telekom.hotspot.intent.action.SMS_STATUS");
localIntent1.putExtra("status",
  CredSmsStatusType.SMS_STATUS_CREDENTIALS_RECEIVED);
localIntent1.putExtra("username", str2);
localIntent1.putExtra("password", str3);
sendBroadcast(localIntent1);
```

Exported Content Provider: SAP Mobile Documents

```
<provider
   android:name="com.sap.mcm.android.content.FileContentProvider"
   android:exported="true"
   android:authorities="com.sap.mcm.android.provider" />

public File getFile(Uri paramUri){
   McmDocument localMcmDocument = getDocument(paramUri);
   File localFile =
       new File(localMcmDocument.getUnencryptedPath());
   return localFile;
}

private void decryptNextDocument() {
   McmDocument localMcmDocument =
   (McmDocument)this.documents.get(this.nextDocumentIndex);
   localMcmDocument.copyUnencrypted(this);
```

Siemens Alarm System and Smart Building Apps: TLS/SSL Client Insecurity

```
public void onReceivedSslError(WebView
  paramWebView, SslErrorHandler
  paramSslErrorHandler, SslError
  paramSslError)
  {
    paramSslErrorHandler.proceed();
  }
```

Problem Statement

We need cost-efficient analysis methods and evaluation processes that assure that Android apps show an appropriate security level.

Dr. Karsten Sohr, sohr@tzi.de

Research Project ZertApps

- BMBF-funded project ZertApps: Certified security for mobile applications

- Project partners:
  - Universität Bremen
  - Fraunhofer SIT (Prof. Dr. Eric Bodden)
  - TU Darmstadt (Prof. Dr. Melanie Volkamer)
  - OTARIS Interactive Service GmbH
  - datenschutz cert GmbH
  - SAP SE (Prof. Dr. Achim Brucker)

Project Goals /1

- Development of precise static security analyses which support the Android Framework (using the Soot Java analysis framework)
- If necessary, dynamic analysis to improve static analysis
- Interaction of several apps (→ confused deputy problem)
- Consideration of hybrid apps (apps with Java and web parts)
  - E.g., analysis of Cordova-based apps

Dr. Karsten Sohr, sohr@tzi.de

Project Goals /2

- Comprehensible presentation of analysis results for different groups of users
  - Security administrators, evaluators, developers, users?
- Conception of a lightweight certification process / scheme
  - Low cost
  - Graded certification concept
- Tool-support for certification

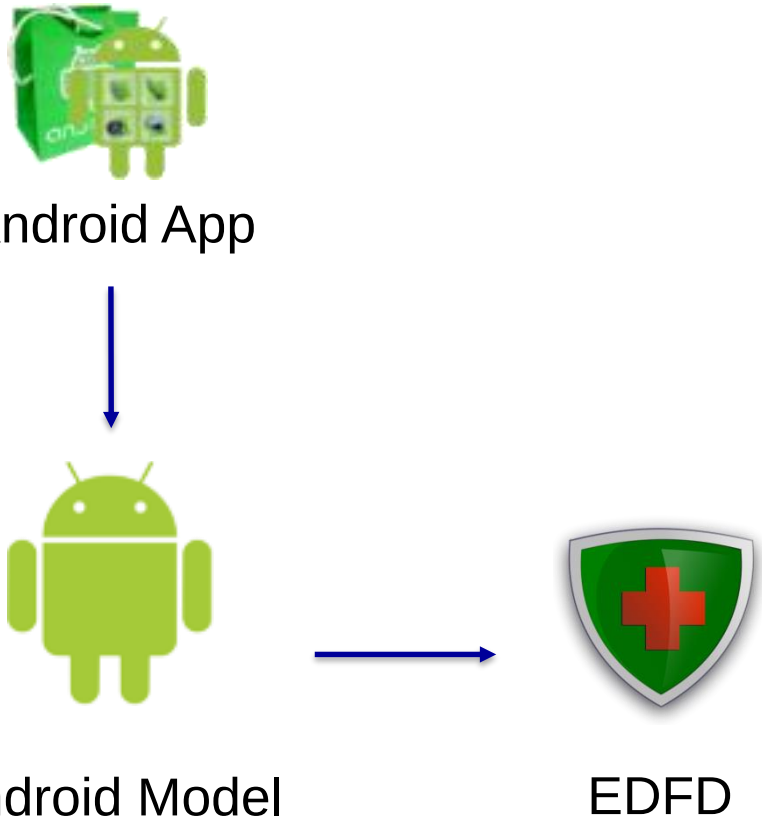
Dr. Karsten Sohr, sohr@tzi.de

Our Contribution: Architectural Risk Analysis

- Extraction and security analysis of the software architectures of apps (or parts of the architectures)
- Reverse engineering of **dataflow diagrams** (DFDs) or **extended dataflow diagrams** (EDFDs) with the help of static analysis (with the help of Soot)
- Automated analysis of these extracted (E)DFDs against known architectural weaknesses (e.g., CWE entries)
- Conception and implementation within in the context of a dissertation at the AG Softwaretechnik (Bernhard Berger)

Dr. Karsten Sohr, sohr@tzi.de

Procedure: Android App to EDFD



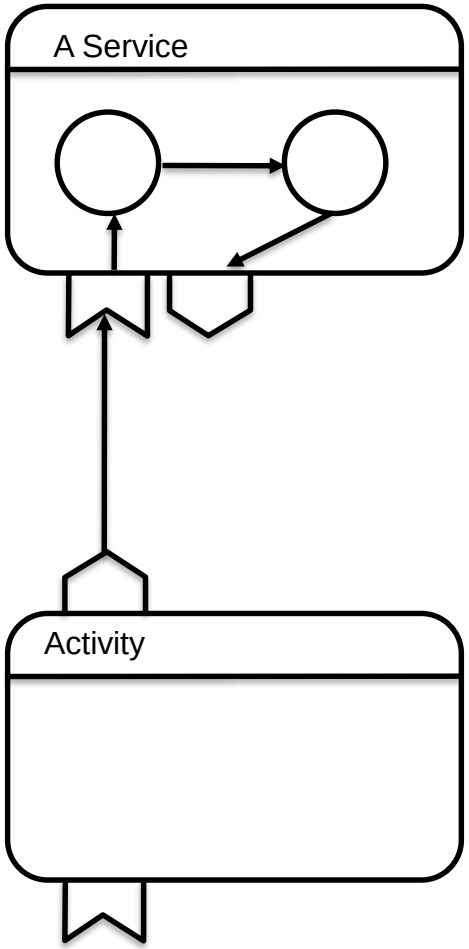
Dr. Karsten Sohr, sohr@tzi.de

Static Analysis Steps

Android app

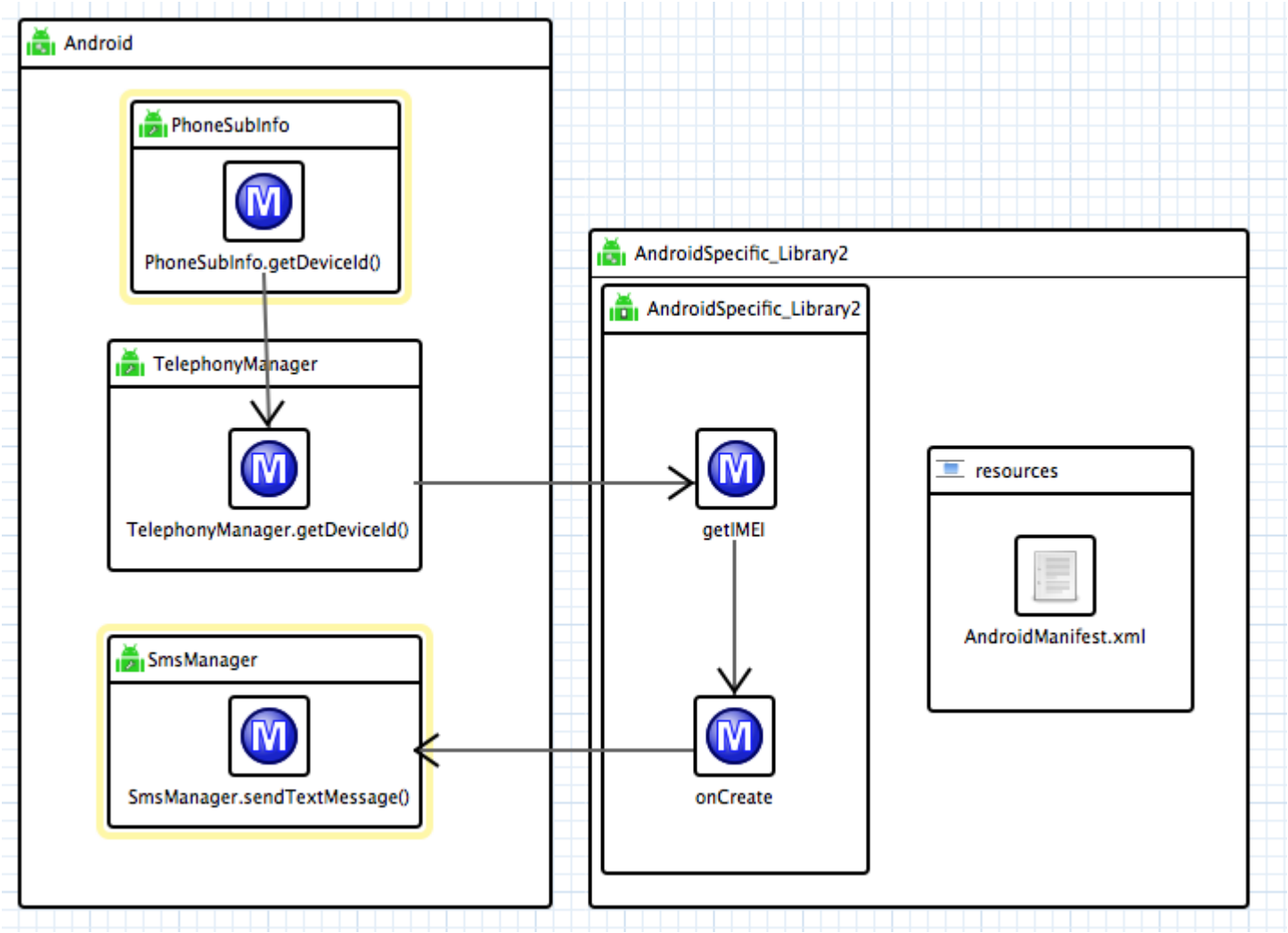
Android model

- Component detection
- Entry point detection
- Detection of exit points
- Identification of intra-component flows
- Identification of inter-component flows
- Identification of security mechanisms
- Determination of external communications
- Usage of shared preferences
- Usage of encryption

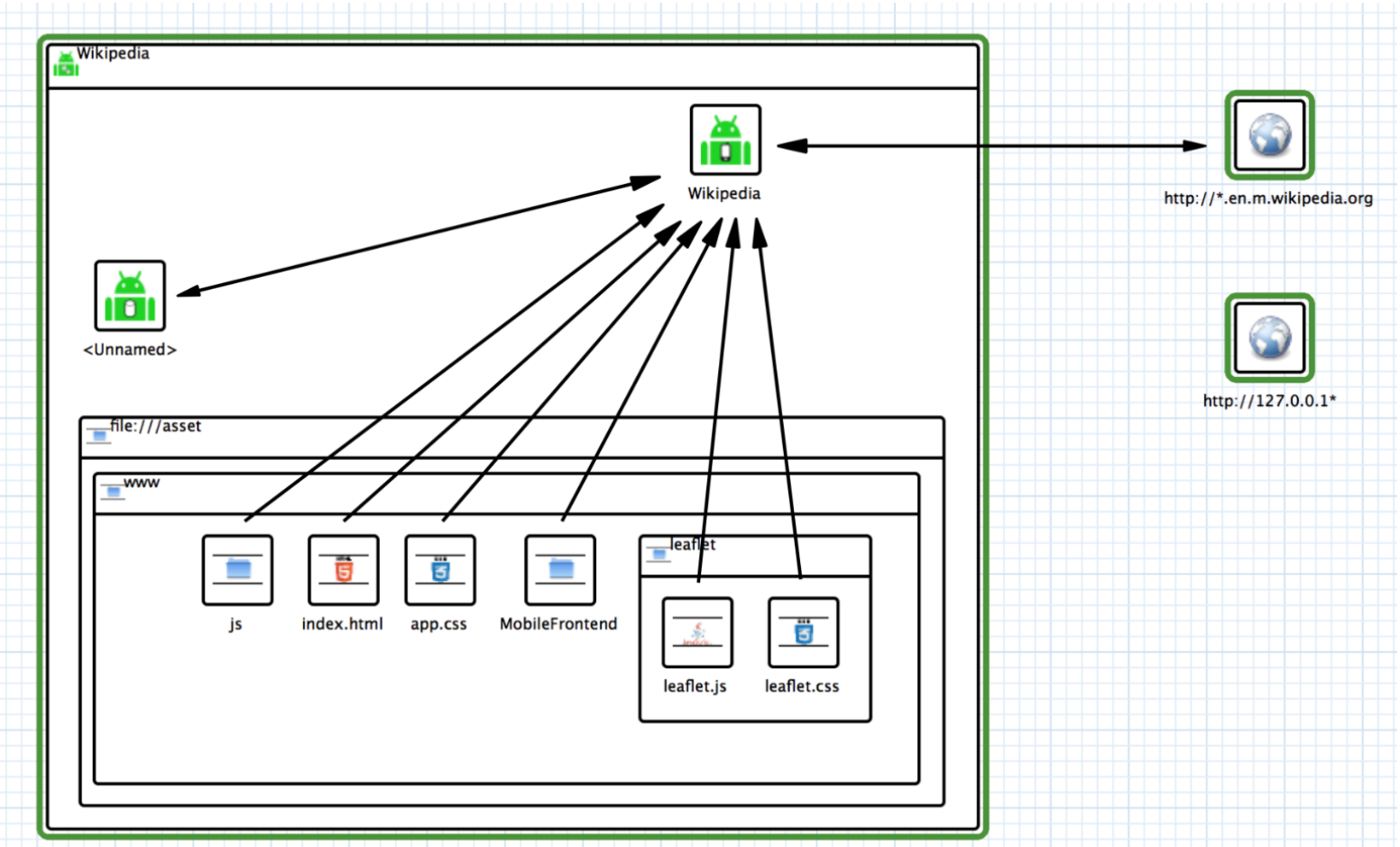


Dr. Karsten Sohr, sohr@tzi.de

An Example Dataflow Diagram



Another Dataflow Diagram: Hybrid App



Dr. Karsten Sohr, sohr@tzi.de

Procedure: Creating Risk Models

Android app

Pattern catalog

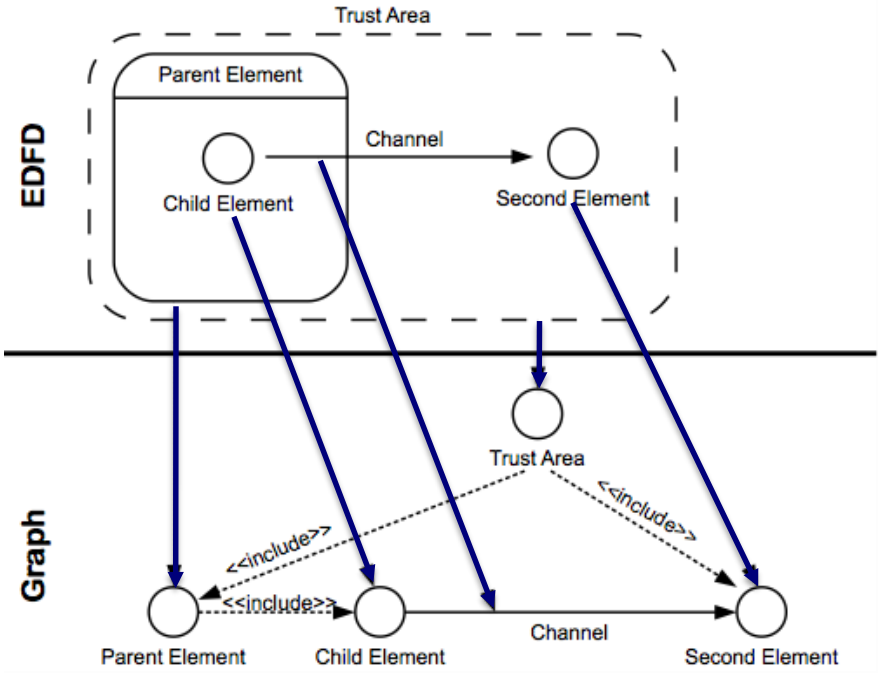
Android model

EDFD

Risk model

Security rules

Checking Security Rules




```
MATCH (source : Element)
      -[flow : Channel *]->
      (target : Element)
WHERE flow.data.IsConfidential
      and not flow.IsEncrypted
```


Dr. Karsten Sohr, sohr@tzi.de

Further Directions of this Work /1

- Applying this approach to Java/JEE applications and integrating it into a software certification platform
  - Project: CertifiedApplications (BMWi-funded, with datenschutz cert GmbH)
  - Paper: B. Berger, K. Sohr, R. Koschke. The Architectural Security Tool Suite ArchSec, 19th IEEE International Working Conference on Source Code Analysis and Manipulation, Cleveland, Ohio, 2019.
- Combined analysis: Android app and backend application
  - Constructing a common DFD (Android app + backend DFD)
  - Connection by using the external interface of the sever (e.g., SAP Mobile Documents app and its underlying content management system)
- Analysis of Android apps containing native code
  - Analysis supporting Java as well as C/C++ code
  - E.g., Java-based Android app with Qt parts (e.g., AusweisApp2 for Android)

Dr. Karsten Sohr, sohr@tzi.de

Further Directions of this Work /2

- Applying this approach to other Java-based software framework: E.g., Spring, Apache Shiro

- Applying this approach to other programming languages
  - C/C++:

    IoT applications or apps with C/C++ parts

    Basic analysis infrastructure: LLVM compiler infrastructure (similar to Soot and WALA, but for C/C+)
  - Microsoft C#:

    .NET framework, in practice often used, well-documented and clearly

    defined API

    Missing basic analysis infrastructure for C# (maybe, Bauhaus tool-suite)

Dr. Karsten Sohr, sohr@tzi.de

Other Approaches: Slicing-Based Code Review

- **Backward Slicing**: Static program analysis technique that calculates all statements influencing a given seed statement (via control and data dependency): Which statements influence the seed?
- Idea: Use security-critical API calls as slicing seed statements and calculate backward slice from them
- Analysts can use these slices within code review tasks
- Using the WALA program analysis framework for this purpose

- Paper: Mustafa, T., Sohr, K. Understanding the implemented access control policy of Android system services with slicing and extended static checking. *Int. J. Inf. Secur.* **14,** 347–366 (2015).

Dr. Karsten Sohr, sohr@tzi.de

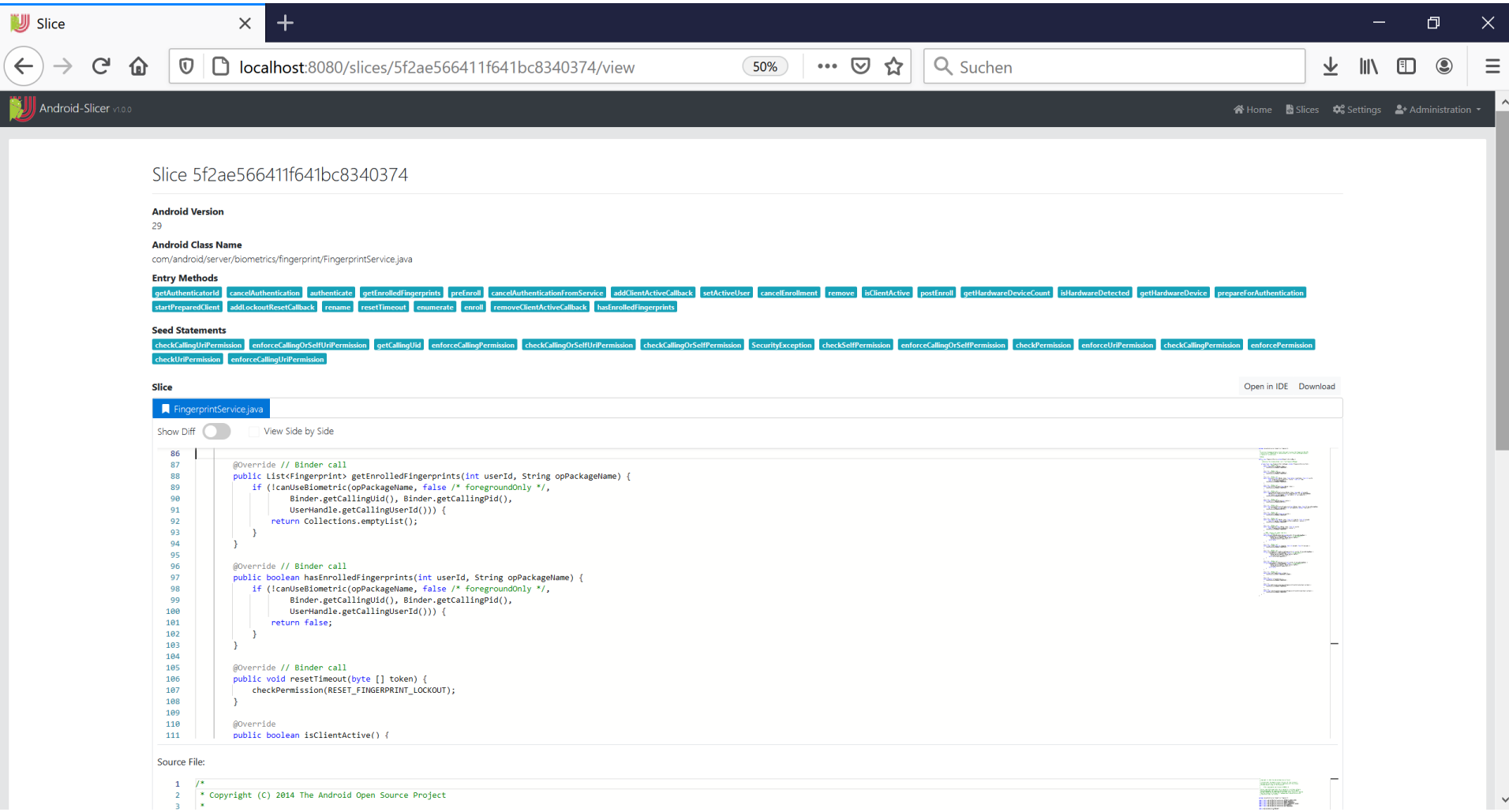
Application of the Slicing-Based Approach

- Android System Services (part of the Android platform)
  - Automated extraction of the implemented access control policy of Android system services by slicing
  - Seeds: E.g., `checkCallingPermission`, `checkPermission`
- JEE applications, programmatic access control: `isCallerInRole`
- Java crypto as wells as Java communications security (e.g., `Cipher.doFinal`, `URLConnection.connect`)
  - Has the crypto code been implemented correctly?
  - Have communications been secured appropriately?
- To improve results of data dependencies: Specific pointer analyses had to be implemented (or available points-to information had to be used)
- Slicing-based analysis tool available:

  https://github.com/BitFlipp3r/AndroidSlicer-Evaluation

Dr. Karsten Sohr, sohr@tzi.de

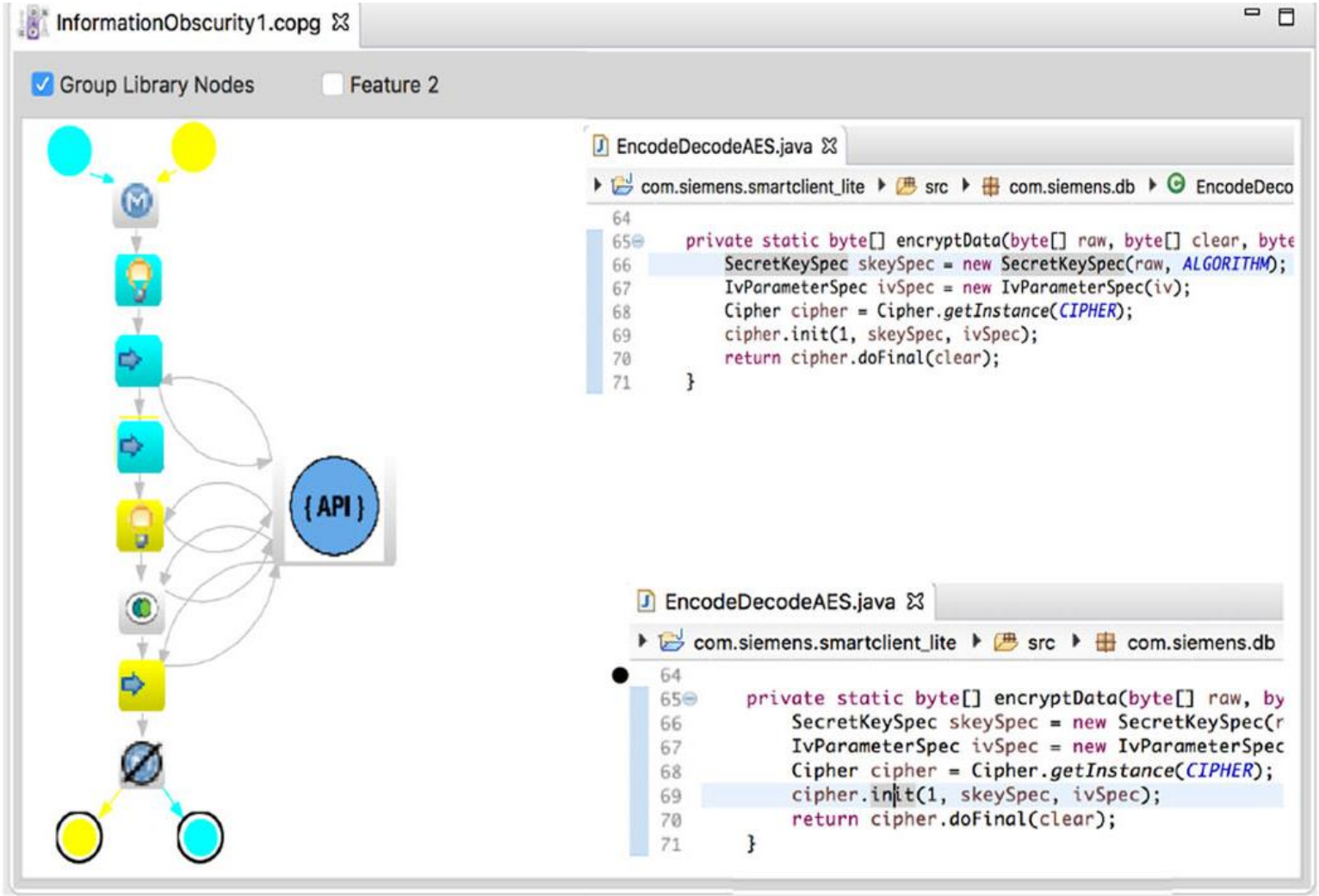
Slicing-Tool for Android: User Interface



Other Approaches: Extracting Security Patterns from Java Code

- Use static program analysis for extracting implemented **security patterns** from Java code
- Informally: Security patterns counterparts to design patterns
- Examples of security patterns:
  - Secure Channel
  - Secure Storage
  - Authenticator
- Idea: Represent manifestations of security patterns in code as **Connected Object Process Graphs** (COPGs)
  - Underlying observation: Security patterns are often represented in code by connected Java objects, e.g. Cipher, SecretKey, SecretKeySpec objects
- Paper: Bunke, M., Sohr, K. Towards supporting software assurance assessments by detecting security patterns. *Software Quality Journal* **28,** 1711–1753 (2020)

An Example COPG



Dr. Karsten Sohr, sohr@tzi.de

User Studies

- Still missing for all approaches: User studies
- Who are the addressed users of such tools?
  - Security analysts and evaluators
  - Architects
  - Developers
- Find adequate user groups and users (not that easy)
- Define user experiments
  - Controlled experiments
  - Baseline: Manual code review, bug finders for review
- Not necessary to come up with a complete tool
  - Demonstrate that *representations help (or not)*

Dr. Karsten Sohr, sohr@tzi.de

Summary & Outlook

- Software security becomes more relevant
  - Mobile apps, Internet of Things, industrial controllers, …
- Systematic and cost-efficient processes for software security are needed
- Especially relevant: security of apps
- Tool support
- Static (and dynamic) code analyses for the extraction and validation of the implemented security architecture
- Comprehensive user experiments/studies necessary to demonstrate efficacy/usability of the developed static analysis tools

Dr. Karsten Sohr, sohr@tzi.de