Security Testing Guidelines for mobile Apps

Florian Stahl
Johannes Ströher

Who we are

Florian Stahl

- Lead Consultant for Information Security, CISSP, CIPP/IT
- Security & Privacy advocate
- Works in Munich for msg systems ag, Germany's 5th largest IT consulting and system development company
- Florian.Stahl@msg-systems.com

Johannes Ströher

- Consultant for Information Security
- Expert for Mobile App Testing
- Developed the Mobile Security Testing Guide in his Master's Thesis
- Johannes.Stroeher@msg.de

Agenda

1. Motivation for Mobile Security Testing Guidelines
   - Current mobile threat landscape and current situation
   - Challenges
2. Mobile Security Testing Guide (MSTG)
   - Overview
   - Intelligence Gathering, Threat Modeling & Vulnerability Analysis in specific
   - Tools and examples
3. Summary

Mobile App Threat Landscape

- Location-independent (mobile)
- “Always online” and traceable
- Consumerization – devices are built for personal use
- Focus on functionality and design rather than security
- Raise of sensitive use cases for mobile apps
- 163% increase of mobile malware in 2012 *
- “Hidden” business cases for free apps

* Source: NQ Mobile Security Report

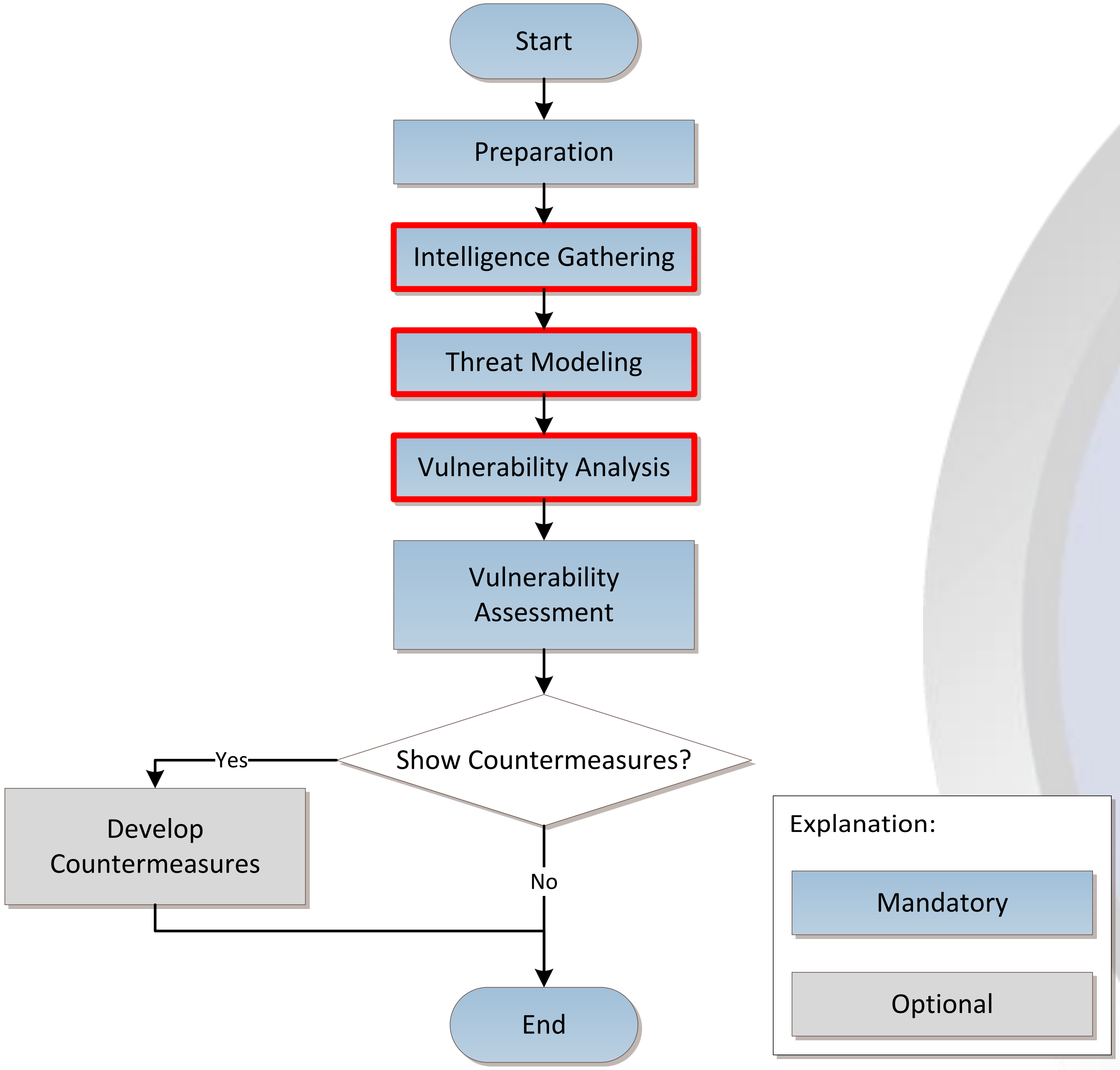
Situation Mobile Security Testing

- Mobile apps have some specific characteristics regarding penetration testing
- Custom guidelines have not been available
- msg systems decided to develop guidelines (MSTG) with Munich University of Applied Sciences
- Similar guidelines published by OWASP: OWASP Mobile Security Testing

Challenges

- Identify differences to common penetration tests
- Flexible Preconditions
  - App Security also depends on device security (jailbreak, different platforms, versions, interfaces, MDM, etc.)
  - Different attackers (internal, external, network or device access, blackbox / whitebox, etc.)
- Keep it flexible AND give specific hints to the penetration tester
- Result: General process (mandatory) and supporting tools and practices (optional)

Mobile Security Testing Guide Overview



Start

Preparation

Intelligence Gathering

Threat Modeling

Vulnerability Analysis

Vulnerability Assessment

Show Countermeasures?

Yes

Develop Countermeasures

No

End

Explanation:

Mandatory

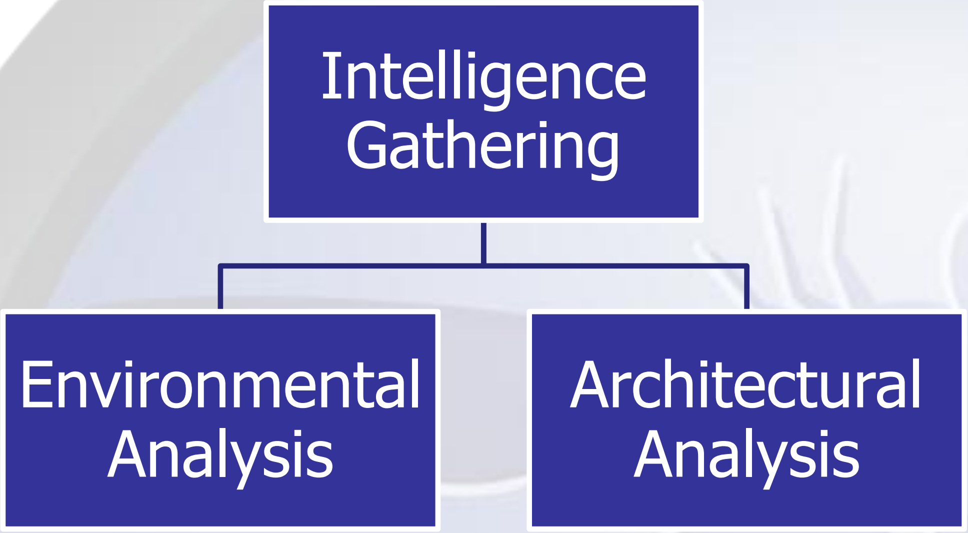
Optional

Annotation for app specific sub-processes

- The specific sub-processes were elaborated in detail for Android and iOS
- An iOS native CRM app is used for illustration because …
  - The CRM app supports many testable functions (authentication, …)
  - It is open source → more possibilities to demonstrate static methods
  - It is a native app → provides more attack surface for the tester
  - We can install the relating CRM service on an own server → no need for taking care of impacts during the tests
- The CRM App was tested on an iPhone 4 with iOS 6

Intelligence Gathering

- Try to catch as much as possible information about the app
- Consists of 2 analysis



Intelligence Gathering

Environmental Analysis

Architectural Analysis

- Differences to conventional process
  - Focus mainly on the architectural/technical part
  - Not considering mobile specific requirements

Intelligence Gathering

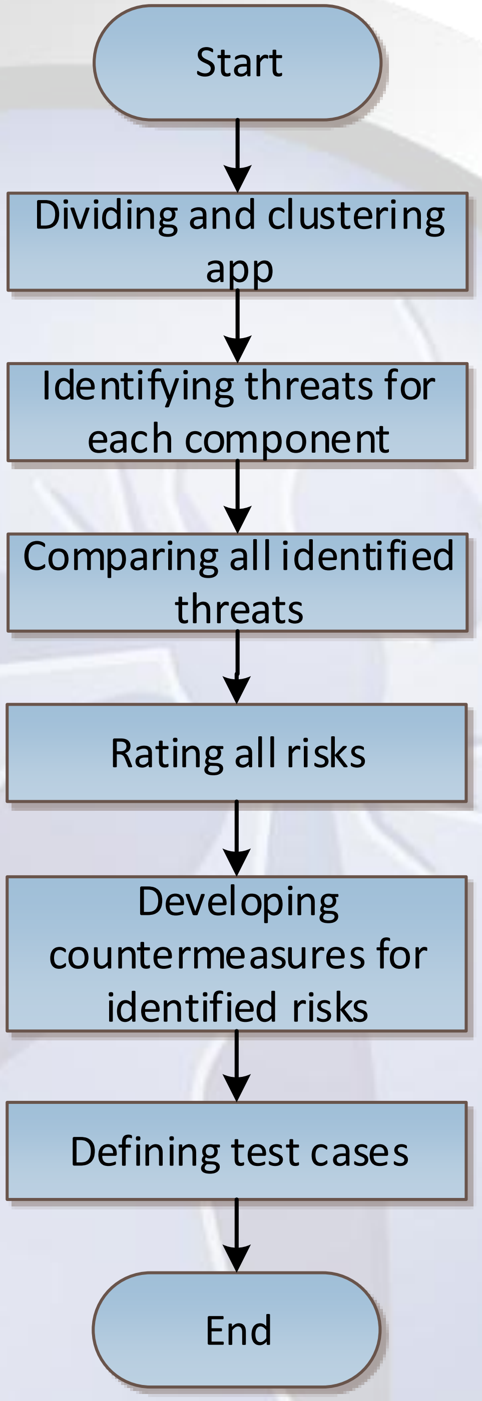
- Environmental Analysis
  - Focus on the company behind the app and their business case and the relating stakeholders
  - Analyze internal processes and structures
- Architectural Analysis
  - App (network interfaces, used data, communication with other ressources, session management, jailbreak/rooting detection, …)
  - Runtime environment (MDM, jailbreak/rooting, os version)
  - Backend services (application server, databases, firewall, …)

Intelligence Gathering - Example

- Examples for collected information from the Architectural Analysis for the CRM app
  - App
    - User session remains until the user logs off manually
    - No financial transactions are included
    - Runs on a jailbroken device → no jailbreak detection
    - Provides operations on server side CRM data for creating, reading, updating, deleting contacts, cases, calls, …
  - Runtime environment analysis is not relevant, because the app is running on a device from the tester
  - Backend services
    - Details about the version of the running CRM service

Threat Modeling

- Identifying threats for the app - specific or prepared threats (e. g. OWASP Top 10)
  - Should be done already in the development
- Risk rating e. g. with OWASP Risk Rating
- Developing countermeasures e. g. with best practices or developers guides
- Differences to conventional process
  - Most software testing processes do not include Threat Modeling
  - Threat Modeling makes the complete process more traceable and efficient for all participants

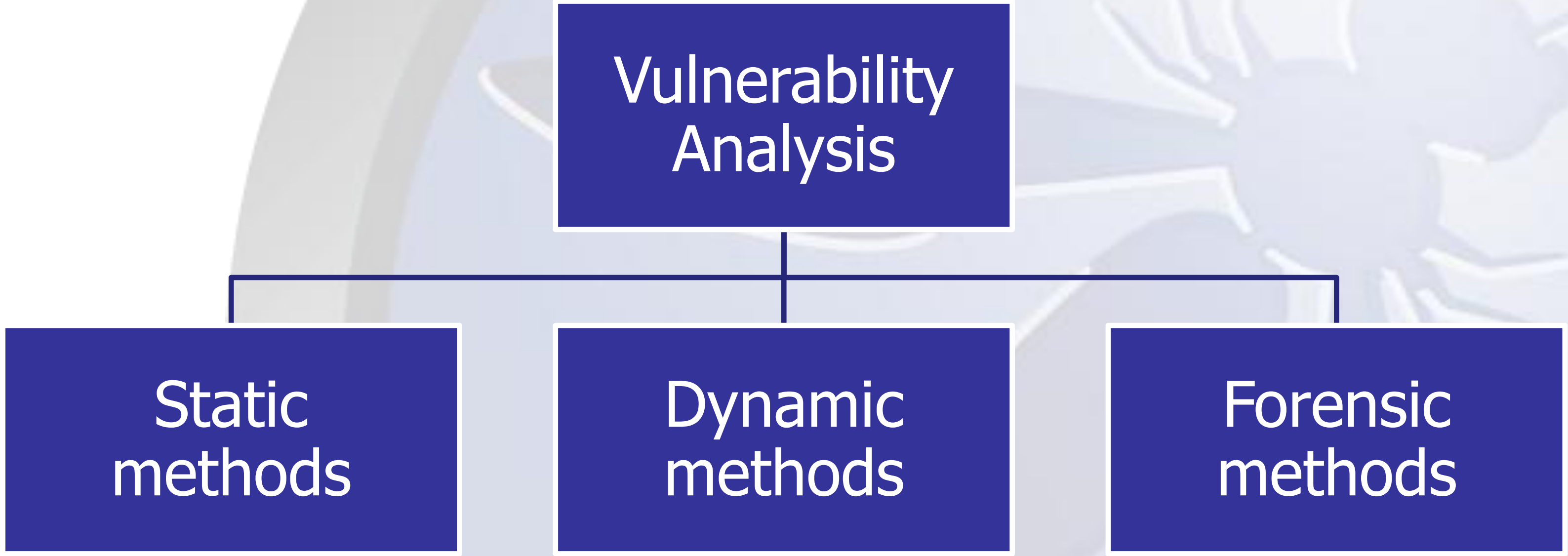

Start
Dividing and clustering app
Identifying threats for each component
Comparing all identified threats
Rating all risks
Developing countermeasures for identified risks
Defining test cases
End

Threat Modeling - Example

- Threat Modeling process example for the CRM App
  - Information from the Intelligence Gathering
    - App provides operations on CRM data on server side
  - Specific threat
    - Unauthorized reading of CRM data on the network traffic while communicating with the CRM backend
  - Relating countermeasure
    - Implementing a secure transport layer protection (e. g. SSL, TLS)
  - Relating test case
    - Try to catch and read the network traffic between the CRM App and the backend

Vulnerability Analysis

- Identifying vulnerabilities in the app with the previous created test cases
- Executing test cases with techniques from 3 different categories



Vulnerability Analysis

Static methods

Dynamic methods

Forensic methods

- Differences to conventional process
  - Most software testing processes not include so many categories of testing methods

Vulnerability Analysis

- Static methods
  - Reverse Engineering
  - Automatic and manual source code analysis

- Excursion: Tools for static methods
  - Reverse Engineering
    - Android: dex2jar, JD-GUI
    - iOS: otool, class-dump-z
  - Automatic and manual source code analysis
    - Android: Androwarn, Andrubis, ApkAnalyser
    - iOS: Flawfinder, Clang Static Analyzer

Vulnerability Analysis

- Dynamic methods
  - Passive network monitoring and analyzing
    - Network traffic analysis at different places in the network (at the device, gateway or in an own VPN)
  - Active network capturing and manipulating (Wifi and cellular)
    - Problems
      - Native apps do not use always device proxy settings
      - SSL encrypted connections
    - Solutions
      - Special apps that force the usage of device proxy settings or which break SSL encrypted connections (mostly for jailbroken or rooted devices)

Vulnerability Analysis

- Dynamic methods
  - Runtime analysis
    - Possible by analyzing the communicating process for internal components (Android: Intents; iOS: objc_msgSend calls)
  - Runtime manipulation
    - Call or manipulate specific functions
    - Read and write variable values
  - File activity analysis
    - Analysis file system changes during the runtime

Vulnerability Analysis

- Dynamic methods - CRM app example
  - Network traffic analysis reveals usage of HTTP and sending non-encrypted sensitive user data (session id, username and password)
    - Tools: Wireshark, BurpSuite, …
  - User authentication can be bypassed by runtime manipulation
    - iOS tools: GNU debugger, Snoop-it, Cycript, …
    - Android tools: Mercury, Intent Sniffer, Intent Fuzzer, …
  - File activity analysis shows that user credentials (username and password) are stored in and used from the iOS keychain
    - iOS tools: filemon.iOS, Snoop-it
    - Android tools: androidAuditTools

Vulnerability Analysis

- Forensic methods
  - Timeline analysis
    - Analyze timestamps created from the file system
  - Analysis of different file types
    - SQLite databases
    - Log files
    - Cookies
    - Screenshots (iOS)
    - Keyboard cache (iOS)
    - SharedPreferences (Android)
    - Keychain (iOS)

Vulnerability Analysis

- Forensic methods - CRM app example
  - Timeline analysis shows that the app updates several files during its runtime (*.plist file, database)
    - Tools: mac-robber, mactime
  - Analyzing identified files and standard file types reveal that the user credentials are stored in plain text in the iOS keychain
    - Tools: Keychain dumper, keychain viewer, …

Summary

Mobile Security Testing Guide …

- … considers mobile characteristics, but is independent from technologies
- … helps to improve transparency and repeatability for mobile penetration testing
- … is a holistic approach with sufficient flexibility
- … and ultimately helps to improve mobile app security

Thank you for your attention!

infosec@msg-systems.com

Full thesis (in German) available on request