Scalable Software Testing for Android: Challenges & Opportunities

Angelos Stavrou & Jeff Voas
George Mason University & NIST
Maturity of Technologies (source Gartner)
### Top 10 CIO Business and Technology Priorities in 2012

<table>
<thead>
<tr>
<th>Top 10 Business Priorities</th>
<th>Ranking</th>
<th>Top 10 Technology Priorities</th>
<th>Ranking</th>
</tr>
</thead>
<tbody>
<tr>
<td>Increasing enterprise growth</td>
<td>1</td>
<td>Analytics and business intelligence</td>
<td>1</td>
</tr>
<tr>
<td>Attracting and retaining new customers</td>
<td>2</td>
<td>Mobile technologies</td>
<td>2</td>
</tr>
<tr>
<td>Reducing enterprise costs</td>
<td>3</td>
<td>Cloud computing (SaaS, IaaS, PaaS)</td>
<td>3</td>
</tr>
<tr>
<td>Creating new products and services (innovation)</td>
<td>4</td>
<td>Collaboration technologies (workflow)</td>
<td>4</td>
</tr>
<tr>
<td>Delivering operational results</td>
<td>5</td>
<td>Legacy modernization</td>
<td>5</td>
</tr>
<tr>
<td>Improving efficiency</td>
<td>6</td>
<td>IT management</td>
<td>6</td>
</tr>
<tr>
<td>Improving profitability (margins)</td>
<td>7</td>
<td>CRM</td>
<td>7</td>
</tr>
<tr>
<td>Attracting and retaining the workforce</td>
<td>8</td>
<td>ERP applications</td>
<td>8</td>
</tr>
<tr>
<td>Improving marketing and sales effectiveness</td>
<td>9</td>
<td>Security</td>
<td>9</td>
</tr>
<tr>
<td>Expanding into new markets and geographies</td>
<td>10</td>
<td>Virtualization</td>
<td>10</td>
</tr>
</tbody>
</table>

Source: Gartner Executive Programs (January 2012)
• Vetted apps ultimately go into an app store.
• Backflows of user feedback and in-field test data.
• If feedback is good, an app becomes app store accepted, and money is deposited; otherwise, a new version from the developers needed.
Why Do I Care?

Commercial Mobile Devices have access to a wide-range of functionality and ship with complex code-base:

- Fully Functional Linux system
- Proprietary device drivers with NO access to code
- Permissive policy model
- Capability to perform a wide range of operations
  - 3 (three) different types for location tracking
  - Many more through meta-data (geo-tagging)

BUT, I am secure: I am using Anti-Virus!!! Right?
Current Mobile Anti-Virus

Commercial AV vendors are not ready for mobile:

- Drain battery quickly
- Similar Results with their Desktop Counterparts
- There are no guaranteed for success in detection
  - Cannot Identify non-preclassified threats
  - CarrierIQ is an example, a “benign” and “legitimate” application
  - Some of them “call-back” home and require constant updates

But is it that bad?
The real picture: **Malicious Apps exist...**

Analyzed ~267,000 Applications from the Google Android Market

- **Thousands** with incorrect/permissive manifest
- **Hundreds** with excessive functionality that can be constituted as malicious
- **Hundreds** of Trojans (i.e. take over existing, legitimate applications)
  - Who will download these apps?
  - People who use SEARCH to find apps
  - Virtually everyone...
- **Two infection vectors:**
  - Regular Web Search
  - Search inside the Mobile App Market
The real picture: **Malicious Apps exist...**

A multifaceted problem:

- Developers maybe well-intended but...
  - They do not necessarily understand the mission or the security/policy requirements
  - They make mistakes
  - They use third-party libraries and code

- The Android permission model is **neither sound nor complete**
  - Intentions, Reflection, JNI, Webkit, others…
  - Android permissions are enforced inside Dalvik not everywhere in the device
What about existing Analysis Tools?

• Commercial application testing tools cover regular, non-Android specific Bugs:
  – No Security Analysis of the Code Functionality
  – No Power Analysis of the Application components and code
  – No Profiling of the resource consumption of individual applications
  – Cannot Regulate/Deny the access and use of phone subsystems (Camera, Microphone, GPS..)

• Existing tools **do not cover Program Functionality**
  – We reveal the application capabilities and access
Application Static Analysis does not cover Program Functionality

Fortify, Coverity, and other application testing tools cover regular, non-Android specific Bugs:

- No Security Analysis of the Code Functionality
- No Power Analysis of the Application components and code
- No Profiling of the resource consumption of individual applications
- **Cannot Regulate/Deny** the access and use of phone subsystems (Camera, Microphone, GPS..)
App Vetting & Control

• App Signing – Prevent unauthorized App Execution
  – Approved Apps are signed by the program designated approval authority
  – Only program signed Apps can be installed on the device
    • Customizations made to Android package framework

• App Analysis & Testing
  – All Apps are analyzed for malware and potential vulnerabilities
    • AV Scans
    • Vulnerability Scans (Fortify)
  – Expose hidden & unwanted functionality
    • Hidden in Native Libraries
    • Dynamic or obfuscated code
  – Permissions manifest reconciliation against code
Application Vetting: Big Picture

Decision = Y

Portal

Artifacts

Results

Appstore

Developer

Assessor

IA Agent

communication
Progression of Testing
In-Lab Testing Process

Enhanced App Version

Initial App Version

Software Evaluation In Test Lab

Assessor

Developer

Software Documentation Including Requirements

Dynamic Testing, and Static Code Analysis Results (Manual and Automated)
In-Field Instrumentation Process

Enhanced App Version

Initial App Version

Assessor

Instrumented software

Software Documentation Including Requirements

Developer

Raw data: Quality/Security and Usage Profiling

Users 1...n
Android Application Control

• Application Signing – Prevent unauthorized App Execution
  – Approved Apps are signed by the program designated approval authority
  – Only program signed Apps can be installed on the device
    • Customizations made to Android package framework

• Application Stress Testing
  – Measure Power Consumption
  – Identify Input Errors / Find UI bugs
Application Analysis Framework

• Android Specific Analysis includes analysis of the Application Security Manifest
  – Tailored to the Android Permission Model

• Verify if the requested permissions are warranted by the submitted code
  – Remove excessive permissions & enforce a tighter security model

• Regulate access to critical/restricted resources
  – Modifications on the Android Engine to enable dynamic policies
  – Control the underlying Dalvik engine to report absence/depletion of resources instead of lack of permissions
Application Policy Enforcement

Solution: Per Application Policy Enforcement

Provide Dalvik mechanisms to

• Enforce application Access & Capabilities
  • Tailored to specific Location or Time
  • Tailored to specific Mission
• Application can still be installed but deprived access to resources and data selectively

Policy Enforcement paired with Device Security can significantly reduce the risk of Data Exfiltration
Android Specific Analysis includes analysis of the Application Security Manifest (not supported by third-party vendors)

- Tailored to the Android Permission Model
- Verify if the requested permissions are warranted by the submitted code
- Curtails excessive permissions and enforces a tighter security model

Modifications on the Android Engine to enable dynamic policies

- Control the underlying Dalvik engine to report absence/depletion of resources instead of lack of permissions
- Regulate access to critical/restricted resources
Power Metering Framework

• Design & Implement an accurate model for accounting and policing energy consumption

• Two-pronged approach
  • Meter the per-process CPU & Device utilization over time
  • Identify the relative impact of each device component on energy consumption

• Design an Android kernel subsystem to estimate energy
  • Meter energy consumption for each App/process
  • Use for characterizing application behavior
  • This behavior is Application dependent
  • Sometimes the behavior is also User dependent
ATP analyzes Android code bundles and returns messages, analysis reports, and signed APKs.

Application Testing Portal

- App Manager:
  - Request Handler
  - Registration Handler
  - Submission Validator
  - UI Handler
  - API Handler

- Analyses Engine:
  - Pre-Processor
  - Tool Invoker
  - Post-Processor
  - APK Compiler/Signer
  - Result Handler

Android Application

Security assessor examines submissions that do not pass ATP analysis.
Mobilize-ATP Workflow (PASS Use-Case)

ATP applies Testing to Analyze Android code bundles

1. Submit Android code bundle
2. Register submission
3. Tool 1 analysis
4. Tool 1 status message & analysis report
5. Tool 2 analysis
6. Tool 2 status message & analysis report
7. Tool n analysis
8. Tool n status message & analysis report

PASS?

9. Assess results

10. Sign APK

11. PASS message & APK

AVs and Testing Tools are invoked in parallel on received submissions

APKs are generated and signed only if all security analyses pass.
Failed: Application Appears to be Using a Different Functionality than what is requested. There is presence of code obfuscation.

The application HtcLoggers_v2 requests the following functionality and permission(s) through AndroidManifest.xml:

- android.permission.READ_LOGS - Constant
- android.permissionINTERNET - Constant
- android.permission.WRITE_EXTERNAL_STORAGE - Constant
- android.permission.ACCESS_CACHE_FILESYSTEM
- android.permission.DIAGNOSTIC - Constant
- android.permission.RECEIVE_BOOT_COMPLETED - Constant
- android.permission.SET_PROCESS_LIMIT - Constant
- android.permission.SET_ALWAYS_FINISH - Constant
- android.permission.SET_DEBUG_APP - Constant
- android.permission.SYSTEM_ALERT_WINDOW - Constant
- android.permission.PERSISTENT_ACTIVITY - Constant
- android.permission.WAKE_LOCK - Constant
- android.permission.ACCESS_FINE_LOCATION - Constant
- android.permission.ACCESS_LOCATION_EXTRA_COMMANDS - Constant
### ATP Monitor

#### NIST App Testing Portal

**Registered Apps**

<table>
<thead>
<tr>
<th>App ID</th>
<th>Name</th>
<th>Submitted</th>
<th>Status</th>
<th>Submitter</th>
<th>Approved</th>
</tr>
</thead>
<tbody>
<tr>
<td>3665043</td>
<td>Illumination-test</td>
<td>2011-11-16 14:44:26.0</td>
<td>ANDROID COMPIL ERROR</td>
<td>cnri</td>
<td>REJECTED</td>
</tr>
<tr>
<td>7238834</td>
<td>Illumination-test</td>
<td>2011-11-16 14:54:47.0</td>
<td>ANDROID COMPIL ERROR</td>
<td>cnri</td>
<td>REJECTED</td>
</tr>
<tr>
<td>1423329</td>
<td>Illumination-test</td>
<td>2011-11-16 14:56:26.0</td>
<td>ANDROID UPDATE OK</td>
<td>cnri</td>
<td>TBD</td>
</tr>
<tr>
<td>5766277</td>
<td>Illumination-test</td>
<td>2011-11-16 15:09:31.0</td>
<td>ANDROID COMPIL ERROR</td>
<td>cnri</td>
<td>REJECTED</td>
</tr>
<tr>
<td>130670</td>
<td>DariToEnglish2.3</td>
<td>2011-11-16 15:10:53.0</td>
<td>ANALYSIS COMPLETE</td>
<td>cnri</td>
<td>APPROVED</td>
</tr>
<tr>
<td>426641</td>
<td>Illumination-test</td>
<td>2011-11-16 15:33:06.0</td>
<td>ANDROID COMPIL ERROR</td>
<td>cnri</td>
<td>REJECTED</td>
</tr>
<tr>
<td>8276571</td>
<td>DariToEnglish2.3</td>
<td>2011-11-16 15:34:34.0</td>
<td>ANALYSIS COMPLETE</td>
<td>cnri</td>
<td>APPROVED</td>
</tr>
<tr>
<td>6052763</td>
<td>Illumination-test</td>
<td>2011-11-16 16:17:04.0</td>
<td>ANDROID COMPIL ERROR</td>
<td>cnri</td>
<td>REJECTED</td>
</tr>
<tr>
<td>6489049</td>
<td>Illumination-test</td>
<td>2011-11-16 16:20:23.0</td>
<td>ANDROID COMPIL ERROR</td>
<td>cnri</td>
<td>REJECTED</td>
</tr>
<tr>
<td>8460629</td>
<td>DariToEnglish2.3</td>
<td>2011-11-16 16:58:59.0</td>
<td>ANALYSIS COMPLETE</td>
<td>steveq</td>
<td>APPROVED</td>
</tr>
<tr>
<td>5809194</td>
<td>Illumination-test</td>
<td>2011-11-17 09:59:41.0</td>
<td>ANALYSIS COMPLETE</td>
<td>cnri</td>
<td>APPROVED</td>
</tr>
<tr>
<td>5560815</td>
<td>DariToEnglish2.3</td>
<td>2011-11-17 10:34:36.0</td>
<td>ANALYSIS COMPLETE</td>
<td>cnri</td>
<td>APPROVED</td>
</tr>
<tr>
<td>6130090</td>
<td>Illumination-test</td>
<td>2011-11-17 10:46:40.0</td>
<td>ANALYSIS COMPLETE</td>
<td>cnri</td>
<td>APPROVED</td>
</tr>
<tr>
<td>9740421</td>
<td>Illumination-test</td>
<td>2011-11-17 11:45:20.0</td>
<td>ANDROID COMPIL ERROR</td>
<td>cnri</td>
<td>REJECTED</td>
</tr>
<tr>
<td>982873</td>
<td>DariToEnglish2.3</td>
<td>2011-11-17 11:47:03.0</td>
<td>ANALYSIS COMPLETE</td>
<td>cnri</td>
<td>APPROVED</td>
</tr>
<tr>
<td>101711</td>
<td>Illumination-test</td>
<td>2011-11-17 12:50:58.0</td>
<td>ANDROID COMPIL ERROR</td>
<td>cnri</td>
<td>REJECTED</td>
</tr>
</tbody>
</table>
Defense in-Depth: Multiple Levels of Security

- Application Vetting & Testing
- Device Lock-down and Encryption of ALL Data and Communications
- Enforcement of Security Policies in the Android Framework
- Second-level Defenses placed in the Android Linux Kernel
  - Prevent Attacks that bypass Android Security Framework
  - Android has Inherited some (if not all) of the Linux Vulnerabilities
  - Java Native Interface to Linux Libraries a potential Avenue for Exploitation
Risks in Mobile Security Supply Chain

Multi-Level Mobile Phone Security Architecture

- Applications
- Services
- Cell Space
- Android: Dalvik on modified Linux
- Encryption Layer for I/O
- Policy Manager (Probing & Policing)
- I/O Devices (USB, WiFi, GPRS, Bluetooth, Internal Flash, SD card)
- Enforce SELinux Policies on Data Flows for Android Apps
- Prevent unauthorized Hardware Flows (USB, Network, FG)

Secure Verify Test Deploy

Enterprise Security

MDM/Middleware Providers

Devices

Enterprise Security
Hardened Android Platform

- Custom modified Android OS and Linux Kernel
  - Additions, deletions, and modifications
  - Preference towards Open Source Solutions
- Security Stack
  - Data At Rest Encryption
  - Data In Transit Protections
  - Authentication
  - App Vetting and Control
  - Device Integrity Checks
Encrypted File System
Ultimately the Testing assists in POLICY Enforcement

- Tailored to the Android Permission Model
- Can allow Location-Based Policies
- Curtails excessive permissions and enforces a tighter security model

Modifications on the Android Engine to enable dynamic policies

- Control the underlying Dalvik engine to report *absence/depletion of resources* instead of lack of permissions
- Regulate access to critical/restricted resources
Conclusions

Assuring the Secure Operation of Smart Devices has a wide-range of requirements

- Application Testing
  - Static & Dynamic
  - In-Field Instrumentation
  - Power Behavior Metering & Policing

- Physical Device Security
  - Lock-Down of the Device I/O (USB, WiFi, etc.)
  - Encryption of Data both on the Phone & Network
  - Securing Provisioning Process
Thank you!

Questions ?