Kratos: Discovering Inconsistent Security Policy Enforcement in the Android Framework

**Yuru Shao**, Jason Ott†, Qi Alfred Chen, Zhiyun Qian†, Z. Morley Mao
University of Michigan, †University of California Riverside
Security Policy Enforcement

• Security policies regulate access to
  – Sensitive data
  – System resources
  – Privileged operations

• Policies need to be correctly enforced
Inconsistencies exist

- According to the Android documentation:
  - apps that hold a `CALL_PHONE` permission can end phone calls

The enforcement of a security policy on different code paths can be inconsistent
Security implication

• Privilege escalation

Exploiting Inconsistencies

Request fewer permissions.
Security implication

- Privilege escalation

Besides app permissions, attackers can also bypass system permissions.
Inconsistent security policy enforcement

• Also found in SELinux and Xen\(^1\)
  – Unauthorized user account access
  – Permanent data loss

• No solution for the Android framework
  – Prior work is OS specific
  – Android has no explicitly defined policies

Problem statement

• Focusing on the Android framework, we answer the following question:
  – How can we systematically detect inconsistent security policy enforcement without any knowledge of the policies?
Our approach

• Discover **app-accessible service interfaces** that have overlaps in functionality
  – They’re expected to have consistent security enforcement

• Perform a *differential analysis* on security checks that two overlapping interfaces employ
Differential analysis

enforcePhone() checks if the caller’s UID is 1001 (RADIO)
enforcePhone() checks if the caller’s UID is 1001 (RADIO)
App-accessible service interfaces

• Analysis scope: system services
  – System services perform enforcement
• Service interfaces
  – AIDL methods
  – Broadcast receivers

AIDL: Android interface definition language
Security checks

• Security enforcement: a set of security checks
• We formulate four types of checks
  – Permission check
  – UID/PID check
  – Package name check
  – Thread status check
Explore the codebase to find
- All system services & interfaces
- Look at service registration

Build a precise framework call graph
- Points-to analysis
- IPC shortcuts

Identify security checks applied to each node (method)

Compare security enforcement of service interfaces if they
- Call the same privileged methods

Ranked list for manual investigation
Implementation

• Support AOSP and customized frameworks
  – Obtain Java classes from
    • Intermediate building output (AOSP)
    • Decompiled dex files (customized)

• Build a precise framework call graph
  – Points-to analysis using Spark
  – An artificial, static entry point including all app-accessible service interfaces

• Perform data flow analysis
  – Identify security check methods
  – Collect system services
Evaluation

• 6 different Android codebases
  – AOSP 4.4, 5.0, 5.1 and M Preview
  – HTC One, Samsung Galaxy Note 3

• Accuracy

<table>
<thead>
<tr>
<th>Codebase</th>
<th># Inconsistencies</th>
<th># TP</th>
<th># FP</th>
<th>Precision</th>
<th># Exploitable</th>
</tr>
</thead>
<tbody>
<tr>
<td>Android 4.4</td>
<td>21</td>
<td>16</td>
<td>5</td>
<td>76.2%</td>
<td>8</td>
</tr>
<tr>
<td>Android 5.0</td>
<td>61</td>
<td>50</td>
<td>11</td>
<td>82.0%</td>
<td>11</td>
</tr>
<tr>
<td>Android 5.1</td>
<td>63</td>
<td>49</td>
<td>14</td>
<td>77.8%</td>
<td>10</td>
</tr>
<tr>
<td>M Preview</td>
<td>73</td>
<td>58</td>
<td>15</td>
<td>79.5%</td>
<td>8</td>
</tr>
<tr>
<td>AT&amp;T HTC One</td>
<td>29</td>
<td>20</td>
<td>9</td>
<td>69.0%</td>
<td>8</td>
</tr>
<tr>
<td>T-Mobile Samsung Galaxy Note 3</td>
<td>128</td>
<td>102</td>
<td>26</td>
<td>79.7%</td>
<td>10</td>
</tr>
</tbody>
</table>
False positive and exploitability

• False positives exist
  – Two interfaces are not equivalent in functionality
  – Points-to analysis produces over-approximated results

• Not all inconsistencies are exploitable
  – Difficult to construct valid arguments
  – Difficult to trigger particular privileged methods
Vulnerabilities discovered

• We found 14 vulnerabilities

![Pie chart showing 8 zero-days and 6 previously reported or fixed issues.]

• 5 out of 14 affect all codebases
• Bug reports confirmed by Google
  – Results website: [http://tinyurl.com/kratos15](http://tinyurl.com/kratos15)
Case study 1

- Bypass system permission to change HTTP proxy settings
  - Documented in Android SDK
  - Hidden, undocumented

  updateNetwork(conf)
  
  save(conf)

  Check
  CHANGE_WIFI_STATE
  CONNECTIVITY_INTERNAL

  addOrUpdateNetworkNative(conf)

  CHANGE_WIFI_STATE
  ACCESS_WIFI_STATE

  CONNECTIVITY_INTERNAL is a system permission

  4.4.2_r1. Fixed in Android 5.0.0_r1

- Allows attackers to bypass the system permission
- MITM, eavesdropping, traffic interception, ...
Case study 2

- Send arbitrary requests to the radio hardware without any permissions

![Diagram showing the process of sending arbitrary requests to the radio hardware](image)

- Allows attackers to send arbitrary requests to radio on vulnerable Samsung phones
- Send SMS, make phone calls, ...
Other observations

• 11 vulnerable interfaces are hidden to apps
  – Not available in the Android SDK
  – Invoke using Java reflection
• AOSP frameworks
  – New system services introduce new inconsistencies, leading to new vulnerabilities
• Customized frameworks
  – Samsung added many system services
    • Introduced 2 additional vulnerabilities
    • One present in AOSP was fixed
Conclusions

• Inconsistent security policy enforcement gives rise to many vulnerabilities
• Our tool is practical and useful for AOSP, vendors, and carriers
• Our approach is general and can be applied to other systems
• To ensure system security, the implementation must faithfully realize the design
Q&A

• Thank you!