FIRMSCOPE: Automatic Uncovering of Privilege-Escalation Vulnerabilities in Pre-Installed Apps in Android Firmware

Mohamed Elsabagh, Ryan Johnson, and Angelos Stavrou
Kryptowire

Chaoshun Zuo, Qingchuan Zhao, and Zhiqiang Lin
The Ohio State University
Android Application Sandbox

- Isolated process
- Isolated storage
- Secure IPC
- Restricted access to resources (permission based)
Privileged Pre-Installed Apps

User Apps
/data/app/...
● Netflix, SnapChat, ...
● User-Installed
● Low privilege
● Limited permissions
● User-granted permissions

System Apps
/system/priv-app/...
● Telephony, I/O, Package Mgmt, ...
● Pre-installed + Persistent
● High privilege
● Unrestricted permissions
● Pre-granted permissions
Pre-Installed Apps

Device OEMs (Firmware Signer)

Chipset OEMs

Affiliates

Carriers

AOSP+Google
Average No. Apps on an Android Phone

<table>
<thead>
<tr>
<th>Pre-Installed Apps</th>
<th>User Apps</th>
</tr>
</thead>
<tbody>
<tr>
<td>164</td>
<td>30</td>
</tr>
</tbody>
</table>
Package installer
Installed

Notifications
On

Permissions
No permissions requested

Storage
49.15 kB used in internal storage

Data usage
No data used

Advanced
Battery, Open by default
<table>
<thead>
<tr>
<th>Permission Category</th>
<th>Granted Status</th>
</tr>
</thead>
<tbody>
<tr>
<td>android.permission.KILL_UID</td>
<td>granted=true</td>
</tr>
<tr>
<td>android.permission.USE_RESERVED_DISK</td>
<td>granted=true</td>
</tr>
<tr>
<td>android.permission.CLEAR_APP_USER_DATA</td>
<td>granted=true</td>
</tr>
<tr>
<td>android.permission.INSTALL_PACKAGES</td>
<td>granted=true</td>
</tr>
<tr>
<td>android.permission.FOREGROUND_SERVICE</td>
<td>granted=true</td>
</tr>
<tr>
<td>android.permission.RECEIVE_BOOT_COMPLETED</td>
<td>granted=true</td>
</tr>
<tr>
<td>android.permission.INSTALL_GRANT_RUNTIME_PERMISSIONS</td>
<td>granted=true</td>
</tr>
<tr>
<td>android.permission.ACCESS_INSTANT_APPS</td>
<td>granted=true</td>
</tr>
<tr>
<td>android.permission.INTERACT_ACROSS_USERS_FULL</td>
<td>granted=true</td>
</tr>
<tr>
<td>android.permission.READ_INSTALL_SESSIONS</td>
<td>granted=true</td>
</tr>
<tr>
<td>android.permission.REVOKE_RUNTIME_PERMISSIONS</td>
<td>granted=true</td>
</tr>
<tr>
<td>android.permission.MANAGE_USERS</td>
<td>granted=true</td>
</tr>
<tr>
<td>android.permission.MANAGE_APP_OPS_RESTRICTIONS</td>
<td>granted=true</td>
</tr>
<tr>
<td>android.permission.CLEAR_APP_CACHE</td>
<td>granted=true</td>
</tr>
<tr>
<td>android.permission.GRANT_RUNTIME_PERMISSIONS</td>
<td>granted=true</td>
</tr>
<tr>
<td>android.permission.HIDE_NON_SYSTEM_OVERLAY_WINDOWS</td>
<td>granted=true</td>
</tr>
<tr>
<td>android.permission.MANAGE_APP_OPS_MODES</td>
<td>granted=true</td>
</tr>
<tr>
<td>android.permission.WAKE_LOCK</td>
<td>granted=true</td>
</tr>
<tr>
<td>android.permission.UPDATE_APP_OPS_STATS</td>
<td>granted=true</td>
</tr>
<tr>
<td>android.permission.OBSERVE_GRANT_REVOKE_PERMISSIONS</td>
<td>granted=true</td>
</tr>
<tr>
<td>android.permission.DELETE_PACKAGES</td>
<td>granted=true</td>
</tr>
<tr>
<td>android.permission.READ_EXTERNAL_STORAGE</td>
<td>granted=true</td>
</tr>
</tbody>
</table>
Privileged Functionality → Access Control

- **Authentication**: Who is allowed access?
- **Authorization**: What are they allowed to do?
- **Accounting**: What did they do?
What We Found

Thousands of privilege-escalation vulnerabilities in Android 4 to 9 due to improper access control in pre-installed apps
Real Example: *Lovely Fonts*
Real Example: *Lovely Fonts*

- **Local/Remote Command+Code Injection**
- **40+ ROMs, 10+ Vendors, Millions of users**
Automatic Discovery: FIRMSCOPE

Preprocessing

- Disassemble & lift to IL
- Extract & canonicalize apps
- Unpack firmware

Static Taint Analysis

- Build inter-procedural CFGs
- Build inter-procedural DFGs
- Perform custom taint analysis

Rules

Firmware Image

Vulnerabilities
Automatic Discovery: Preprocessing

Challenges:
- Non-standard image formats
- Different build/optimization settings
- Dalvik bytecode internals

Solutions:
- Unpacking heuristics
- Lift disassembly into IL
Automatic Discovery: Static Taint Analysis

Challenges:
- Flows through fields, callbacks, lifecycles, ...
- Analysis sensitivities
- Practical considerations

Solutions:
- Custom Def-Use
- Encode flows using custom gadgets
Automatic Discovery: Rules

- Command/Code Injection
- App (Un)Installation
- Audio/Video/Screen Recording
- Settings Modification
- SMS Reading, Sending
- Information Leakage
- Device Flashing/Resetting

- Source/Sink rules
Findings Summary

- Scanned 2017 Android 4 to 9 ROMs (331k apps)
- 850 unique vulnerabilities (3k+ total)
- 77% ROMs impacted
- 41% had Command Injection
- ⅓ the findings lead to code execution
- Disclosed 370+ in Android 7,8,9
- 200+ vendor-confirmed to date
Who Is to Blame?

- Overall lax security posture by vendors
- Most flaws from custom features, factory-mode apps, OTA providers, MDM apps, helpers, ...
- AOSP-like devices were the cleanest
Runtime Performance

- 7 min per app on average (53 s median)
- 81.7 min per ROM on average (55.7 min median)
- Significantly less FPs, FNs, CPU, MEM than prior solutions
FIRMSCOPE

- Accurate, efficient, static taint analysis
- Automatic privilege-escalation vulnerabilities detection
- Scanned 2017 ROMs (331k pre-installed apps)
- Discovered 850 unique privilege-escalation vulnerabilities
- Responsible disclosures for Android 7,8,9

Thank You!

Pilots: https://www.kryptowire.com/contact-us