PatchDroid: Third Party Security Patches for Android

TelSecDay, TROOPERS 2014

Collin Mulliner, March 17th 2014, Heidelberg
crm[at]ccs.neu.edu
$ finger collin[at]mulliner.org

- 'postdoc' Security Researcher
  - $HOME = Northeastern University, Boston, MA, USA
  - cat .plan
    - specialized in mobile handset security

- cat .project
  - OS and software security
  - Android security
  - SMS and MMS security (mostly handset related)
  - Mobile web usage and privacy
  - Early work on NFC phone security
  - old stuff: Windows Mobile, Symbian, Bluetooth, ...
Android

Collin Mulliner – “PatchDroid: Third Party Security Patches for Android”
Android

most popular smart phone platform

About 1 billion devices today
Android Security

- Android platform contains security vulnerabilities
  - New vulnerabilities are discovered all the time

- Android has built-in update mechanism
  - Over-the-Air (OTA) updates
  - No desktop computer needed

- Google patches a bug
  - Update arrive at Nexus devices (Google devices)
  - Patches are pushed to AOSP
  - Manufacturers are notified
Android Security

- Android platform contains security vulnerabilities
  - New vulnerabilities are discovered all the time

- Android has built-in update mechanism
  - Over-the-Air (OTA) updates
  - No desktop computer needed

- Google patches a bug
  - Update arrive at Nexus devices (Google devices)
  - Patches are pushed to AOSP
  - Manufacturers are notified

- Unfortunately, only few devices receive updates!
Missing Updates

- Manufacturer
  - Stop supporting devices after about 18 months
  - Manufacturer specific bugs

- Carrier
  - Customize firmware
  - Delay updates due to customization efforts
  - Do not update at all

- Result → many devices run out of date software
  - Software that contains publicly known vulnerabilities
### Android Platform Version Diversity

<table>
<thead>
<tr>
<th>Version</th>
<th>Codename</th>
<th>API</th>
<th>Distribution</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.2</td>
<td>Froyo</td>
<td>8</td>
<td>1.7%</td>
</tr>
<tr>
<td>2.3.3-2.3.7</td>
<td>Gingerbread</td>
<td>10</td>
<td>26.3%</td>
</tr>
<tr>
<td>3.2</td>
<td>Honeycomb</td>
<td>13</td>
<td>0.1%</td>
</tr>
<tr>
<td>4.0.3-4.0.4</td>
<td>Ice Cream Sandwich</td>
<td>15</td>
<td>19.8%</td>
</tr>
<tr>
<td>4.1.x</td>
<td>Jelly Bean</td>
<td>16</td>
<td>37.3%</td>
</tr>
<tr>
<td>4.2.x</td>
<td></td>
<td>17</td>
<td>12.5%</td>
</tr>
<tr>
<td>4.3</td>
<td></td>
<td>18</td>
<td>2.3%</td>
</tr>
</tbody>
</table>

Source: Google (Nov. 1, 2013)
Patching Vulnerabilities on Android Devices

- Overlooked problem until now
  - Google and manufacturers' duty

- Only solution so far is 3rd party firmware
  - Available for limited number of devices only
  - Manual process, limited follow-up updates

- Platform diversity is the key problem
  - Large number of different devices + software versions
  - Any solution has to address these problems
Challenges

- No access to source code
  - AOSP ≠ code running on devices
  - Modifications by the manufacturer

- Issue with modification of system files and partitions
  - Modified binaries might prevent system from booting
  - Cannot add/replace files on signed partitions

- Scalability vs. Testing
  - Too many different devices and OS versions
  - Patches need to be decoupled from the source code
Contributions

- **PatchDroid**: third-party security patches for Android
  - Includes attack detection and warning mechanism

- Scalable
  - Independent from device and Android version
  - Support for managed Dalvik bytecode and native code

- Reliable
  - No permanent modification (no bricked devices)

- Usable in practice
  - No noticeable overhead (no device slow down)
  - Does not rely on access to source code
Contributions

- **PatchDroid**: third-party security patches for Android
  - Includes attack detection and warning mechanism

- Scalable
  - Independent from device and Android version
  - Support for managed Dalvik bytecode and native code

**PatchDroid: Scalable Third-Party Security Patches for Android Devices**
Collin Mulliner, Jon Oberheide, William Robertson, Engin Kirda
In the Proceedings of the 29th Annual Computer Security Applications Conference (ACSAC), Dec. 2013
  - No permanent modification (no bricked devices)

- Usable in practice
  - No noticeable overhead (no device slow down)
  - Does not rely on access to source code
Overview

- Design
- Patches and Patching
- Implementation
- Technical Insights
- System Evaluation
- Case Study: Master Key bug(s)
The PatchDroid System

- In-memory patching at runtime
  - Need to patch processes at startup
    - Before process executes vulnerable code
    - Monitor system for new processes
  - No need to modify system files or system partitions

- Patches as independent code
  - Self-contained shared library
  - Patching via function hooking
  - No access to original source code required
  - Scale across different OS versions
PatchDroid: Architecture

- **PatchDroid**
  - process creation monitor
  - patch deployment & trace patcher
  - patch monitor
  - attack detection module
  - patch update system
  - central logging & remote reporting
  - GUI and Alerts (separate process)

- **Device**
  - init
  - zygote
  - patch (process A)
  - process B

- **Cloud**
  - patch repository
  - log event receiver
PatchDroid: Architecture

Identify newly created processes - trace init and zygote
PatchDroid: Architecture

Deploy patch into process - library injection
Monitor execution of patch code
- check for instabilities
- collect logs
PatchDroid: Architecture

- Analyze log for exploitation attempt

- Cloud components:
  - Log event receiver

- Device components:
  - PatchDroid
    - Process creation monitor
    - Patch deployment & trace patcher
    - Patch monitor
    - Patch update system
    - Central logging & remote reporting
    - GUI and Alerts (separate process)

- External components:
  - Init
  - Zygote
  - Patch process A
  - Process B

Collin Mulliner – “PatchDroid: Third Party Security Patches for Android”
PatchDroid: Architecture

PatchDroid App
- GUI
- display alerts
PatchDroid: Architecture

PatchDroid cloud infrastructure
- central logging + reporting
- patch repository

Collin Mulliner – “PatchDroid: Third Party Security Patches for Android”
Anatomy of a Patch

- Replacement for vulnerable function
  - Equivalent code that does not contain the vulnerability
  - Wrapper that adds input/output sanitization

- Installation
  - Hook vulnerable function(s)
    (original function needs to be kept in working condition)

- Communication link
  - Read configuration parameters
  - Write log messages
Patching Strategies

- Function replacement
- Proxy function
- Failed return value check
Example: Failed return value check

- int res = setuid(nobody);
  - res == 0 → success, privileges dropped
  - res == -1 → failure, privileges NOT dropped

- Missing check of result in zygote (zimperlich exploit)
  - fork() until setuid() failed due to resource limit
  → new process stays root!

- Patch: wrap setuid()
  - check result
  - terminate if res != 0
Patch Life Cycle

- Deployment
  - trace target process
  - setup communication
  - inject patch library
Patch Life Cycle

- Installation
  - connect communication
  - hook function
Patch Life Cycle

- Fixed function is called
  - log and report attack
  - collect patch telemetry
  - (call original function)
Patch Life Cycle

- Patch failure
  - detected using telemetry
  - failing patch is removed from system

- Enables scalable testing of patches in the field
Implementation

- *patchd*: the patch daemon
  - Monitor system for newly created processes
  - Inject patches into processes
  - Monitor patched processes

- PatchDroid Application
  - User interface
  - Helper service
  - Attack notification

- Patches
  - 3 patches for privilege escalation (native code)
  - 2 patches for bugs in managed code (Dalvik code)
Technical Insights

- **patchd**
  - uses `ptrace()` API for monitoring and injection
  - **requires root access**

- PatchDroid operates in `/data/../.com.patchdroid.patchdroid`
  - no need to modify 'system' FS
    - system FS could be signed
  - secure against other non root apps
  - can be removed like any other app (reliable)

- **patchd + PatchDroid (dalvik) app**
  - Need both to stimulate system services
    - e.g., get process unstuck while patching it
Technical Insights cont.

- **Patch**
  - Shared library
  - Self-contained → no external dependencies
  - Shared memory region to communicate with patchd

<table>
<thead>
<tr>
<th>Mode</th>
<th>Owner</th>
<th>Group</th>
<th>Size</th>
<th>Date/Time</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>drwxr-xr-x system</td>
<td>system</td>
<td>system</td>
<td>2014-03-05 20:42</td>
<td>lib</td>
<td></td>
</tr>
<tr>
<td>r-xr-xr-x app_69</td>
<td>app_69</td>
<td>app_69</td>
<td>71104 2014-03-05 20:43</td>
<td>patchd</td>
<td></td>
</tr>
<tr>
<td>rw-rw---- root</td>
<td>root</td>
<td>root</td>
<td>10 2014-03-07 14:40</td>
<td>checks</td>
<td></td>
</tr>
<tr>
<td>rw-r--r-- app_69</td>
<td>app_69</td>
<td>app_69</td>
<td>37932 2014-03-05 20:43</td>
<td>libvoldmeta.so</td>
<td></td>
</tr>
<tr>
<td>rw-r--r-- app_69</td>
<td>app_69</td>
<td>app_69</td>
<td>37932 2014-03-05 20:43</td>
<td>libsetuid.so</td>
<td></td>
</tr>
<tr>
<td>rw-r--r-- app_69</td>
<td>app_69</td>
<td>app_69</td>
<td>37932 2014-03-05 20:43</td>
<td>libsmsperms.so</td>
<td></td>
</tr>
<tr>
<td>rw-r--r-- app_69</td>
<td>app_69</td>
<td>app_69</td>
<td>37932 2014-03-05 20:43</td>
<td>libzipentry.so</td>
<td></td>
</tr>
<tr>
<td>rw-r----- root</td>
<td>root</td>
<td>root</td>
<td>8993 2014-03-07 14:45</td>
<td>log</td>
<td></td>
</tr>
</tbody>
</table>

| App_69 | 1602 | ... | com.patchdroid.patchdroid |
| Root   | 1613 | ... | /data/data/com.patchdroid.patchdroid/patchd |
| Root   | 1619 | ... | /data/data/com.patchdroid.patchdroid/patchd |
Patch 'Technology'

- Dynamic instrumentation techniques
  - Mainly hooking functions
  - Need to preserve original function

- Native Code hooks
  - based on ADBI (http://github.com/crmulliner/adbi)

- Dalvik Code
  - based on DDI (http://github.com/crmulliner/ddi)
Patch Creation

- Extract patch from AOSP and transform to PatchDroid
  - Apply patch strategy best suited for vulnerability

- Develop custom patch
  - Bug known but no patch available
Patch Creation

- Extract patch from AOSP and **transform** to PatchDroid
  - Apply patch strategy best suited for vulnerability

- Develop custom patch
  - Bug known but no patch available

- **Walkthrough for creating a patch using source code**
Source Patch → PatchDroid Patch

- Source level patch

Source level patch:

```java
luni/src/main/java/java/util/zip/ZipFile.java
```

- Missing return value check:
  - `mEntries.put()` returns Non null if hash already has an entry with given key
- Duplicate key → multiple Zip entries with same name
Source Patch → PatchDroid Patch cont.

- **Hook**: `java.lang.LinkedHashMap.put()`
  - call original `put` method
  - check return value, throw exception if result != null

- But hashmap is used in other places how do we not break it?
  - Only hook it when used by ZipFile code

- **Hook**: `java.util.zip.ZipFile.readCentralDir()`
  - install hook for `LinkedHashMap`
  - call original `readCentralDir` method
  - unhook `LinkedHashMap`
Patch Deployment

- PatchDroid injects patches into processes
  - Patches need a mapping to a process

- Service processes
  - vold, adbd

- Android framework is separated into a number of processes
  - zygote (base VM)
  - system_server (runs most of the framework code)

- The ZipFile class is used by APK installer
  - Code is part of the Android runtime framework
Evaluation

- System performance
  - Low overhead during process creation
  - No runtime overhead

- Functional testing
  - Patch vs Exploit

- User trials
  - Users run PatchDroid
  - Try exploiting known vulnerabilities
    (details in the PatchDroid paper)
Overhead – creating new process

- One time hit at process creation
Patch vs Exploit

- Privilege escalation vulnerabilities (root exploits)
  - Zimperlich
  - GingerBreak
  - ZergRush

- Permission leak
  - local SMS spoofing (Dalvik)

- APK processing
  - MasterKey series of bugs (Dalvik)

- All patches prevent exploitation on the affected devices
  - PatchDroid warns the user about attack
Patch vs Exploit

- Privilege escalation vulnerabilities (root exploits)
  - Zimperlich
  - GingerBreak
  - ZergRush

- Permission leak
  - local SMS spoofing (Dalvik)

- APK processing
  - MasterKey series of bugs (Dalvik)

- All patches prevent exploitation on the affected devices
  - PatchDroid warns the user about attack
Case Study: MasterKey Bug(s)

- Bug(s) in handling of APK files
  - APK can be modified w/o breaking the signature

- **MasterKey can be used for privilege escalation**
  - Modify APK signed with platform/manufacturer key
  - Works on all devices from manufacturer

- Bug in manifested in Dalvik bytecode
  - First privilege escalation vulnerability in Dalvik code

- Present in AOSP before version 4.3
  - Affected almost all Android devices at the time
Patching MasterKey

- Multiple bugs → multiple patches
  - Different strategies, fix via:
    - return value check
    - input sanitization

- Fast implementation and testing
  - Initial version took only three (3) hours

- We wanted to release the patch to the general public
  - Provide possibility to protect user's devices
ReKey

- Limited version of the PatchDroid system
  - Only the MasterKey bug(s)

- Released ReKey on the Google Play store
  - July 16th, 2013

- +32K installs on play store
  - Only works on rooted devices!

- ReKey your device!
  - http://www.rekey.io
PatchDroid / ReKey - Demo Video

This version of ReKey contains all PatchDroid patches
Conclusions

- We are the first to address security patching on Android

- With PatchDroid we show that
  - third-party patching is possible without source code
  - patch development scales across different devices

- **PatchDroid**
  - supports Dalvik and native code
  - no noticeable performance overhead
  - no impact on device stability
  - safe against accidentally “bricking” devices

- Public release of ReKey was a huge success
Thank you!
Questions?
http://www.patchdroid.com