Kyoto, 2012 – FIRST Technical Colloquium

Smartphone Security and Finding "Third-party" Risks

Fourteenforty Research Institute, Inc.
http://www.fourteenforty.jp

Tsukasa Oi – Research Engineer

Self Introduction

- Fourteenforty Research Institute, Inc. (FFRI)
  - Tokyo, Japan
  - R&D in the field of computer security
- Tsukasa Oi : Research Engineer at FFRI
  - Currently focusing on mobile security
  - Recent Talks at:
    - PacSec 2011
      "How Security Broken?"
    - Black Hat Abu Dhabi 2011
      "Yet Another Android Rootkit */protecting/system/is/not/enough/*"
    - Black Hat USA 2012
      "Windows Phone 7 Internals and Exploitability"

Background

- Modern mobile operating systems
  - Sandbox to protect system and applications
  - Some kind of MAC (Mandatory Access Control)
  - Integrated application distribution (App Stores)
- Modifications by Third-party Vendors
  - Android
  - Windows Phone (7.x)

Agenda

- Security Design
  - Android
  - Windows Phone 7
- Risks and Vulnerabilities
  - What we find
- Third-Party Risks and Vulnerabilities
  - Remote DoS
  - Privilege Escalation
  - Access Control Vulnerability
- Finding Vulnerabilities

Caution!

We cannot disclose many of vulnerabilities we've found

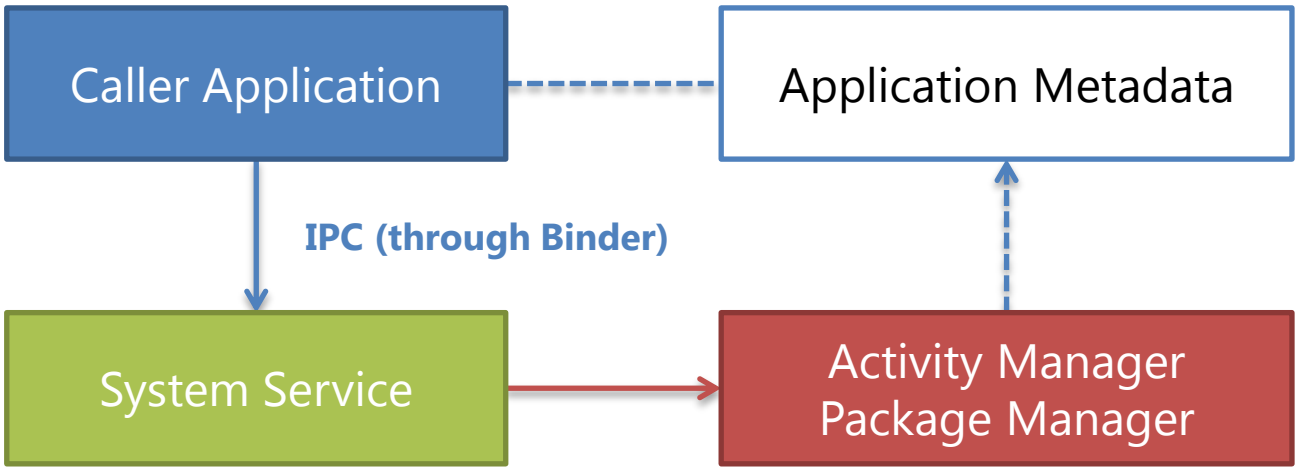
It looks pretty good. But is it enough then?

SECURITY DESIGN

Android : Permission

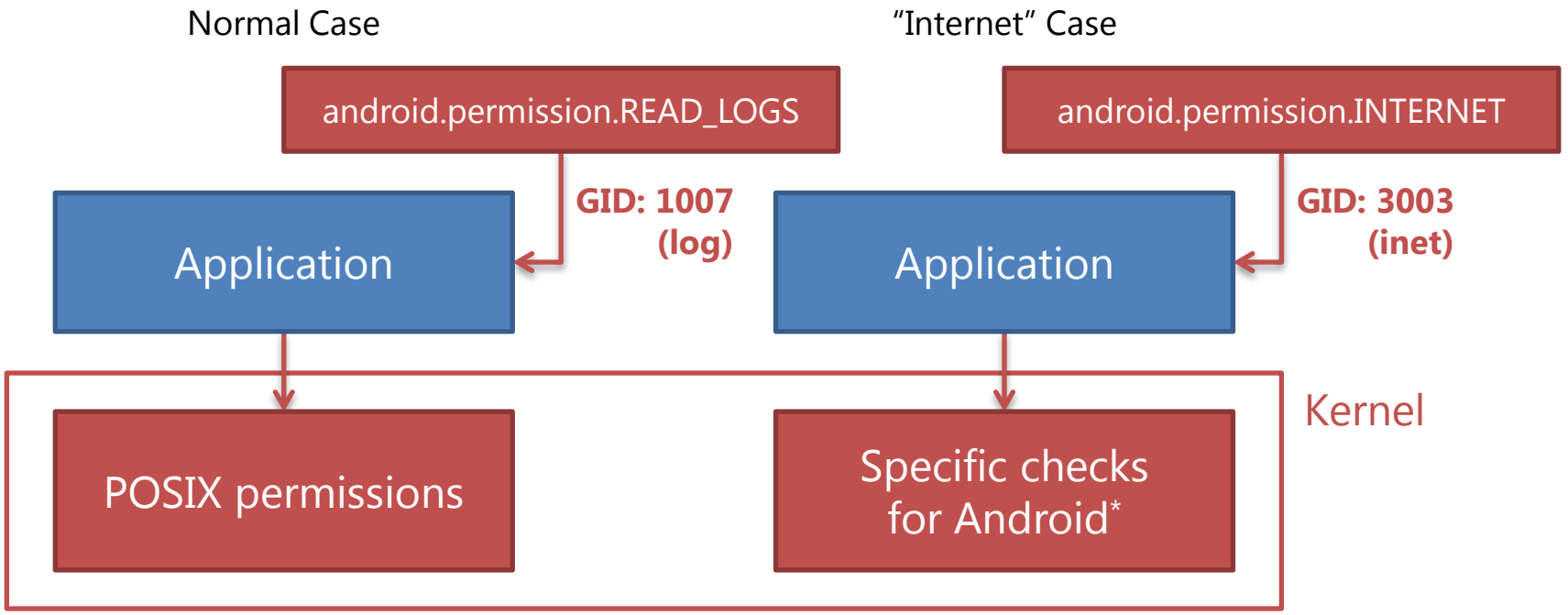
- Restrict access to specific resources
  - Need declaration to use specific features
    - Sensor data / Camera
    - Location
    - Access to system resources
  - Special GID or software checks
  - Some permissions are restricted for system apps (like INSTALL_PACKAGE; allows unattended installation)
    - Checks by package location / signature

Android : Permission Checks (1)



- Service Manager (or important method) checks callers permission
  - Achieve good isolation
    (IPC glue is automatically-generated)

Android : Permission Checks (2)



- Some permissions are associated with specific GIDs
  - Use POSIX permission checks except "Internet" permission

* Linux kernel for Android is modified to restrict Internet sockets to processes which have GID 3003 (inet).

Android : Isolation

- One UID for One App
  - Unless apps by same developer declare to share UID
  - No apps can access other apps data unless its permission is world-accessible
    - Vulnerability in Skype for Android (CVE-2011-1717)
- Read-only access to some system resources
  - e.g. Data in SD card (will require READ_EXTERNAL_STORAGE permission in the future)
  - e.g. /data/system/packages.list (which enables to access package list without permission)

Android : Additional Security by Vendor

- Some vendors add security layer to avoid issues
  - NAND protection
    protect system partition of flash will not be overwritten
  - LSM (Linux Security Modules); except SEAndroid
    prohibit dangerous operations from being performed
  - Better security controls
    (e.g. 3LM Security)
- Some of them can be *effectively* broken
  - "Yet Another Android Rootkit */protecting/system/is/not/enough/*"
    Black Hat Abu Dhabi 2011

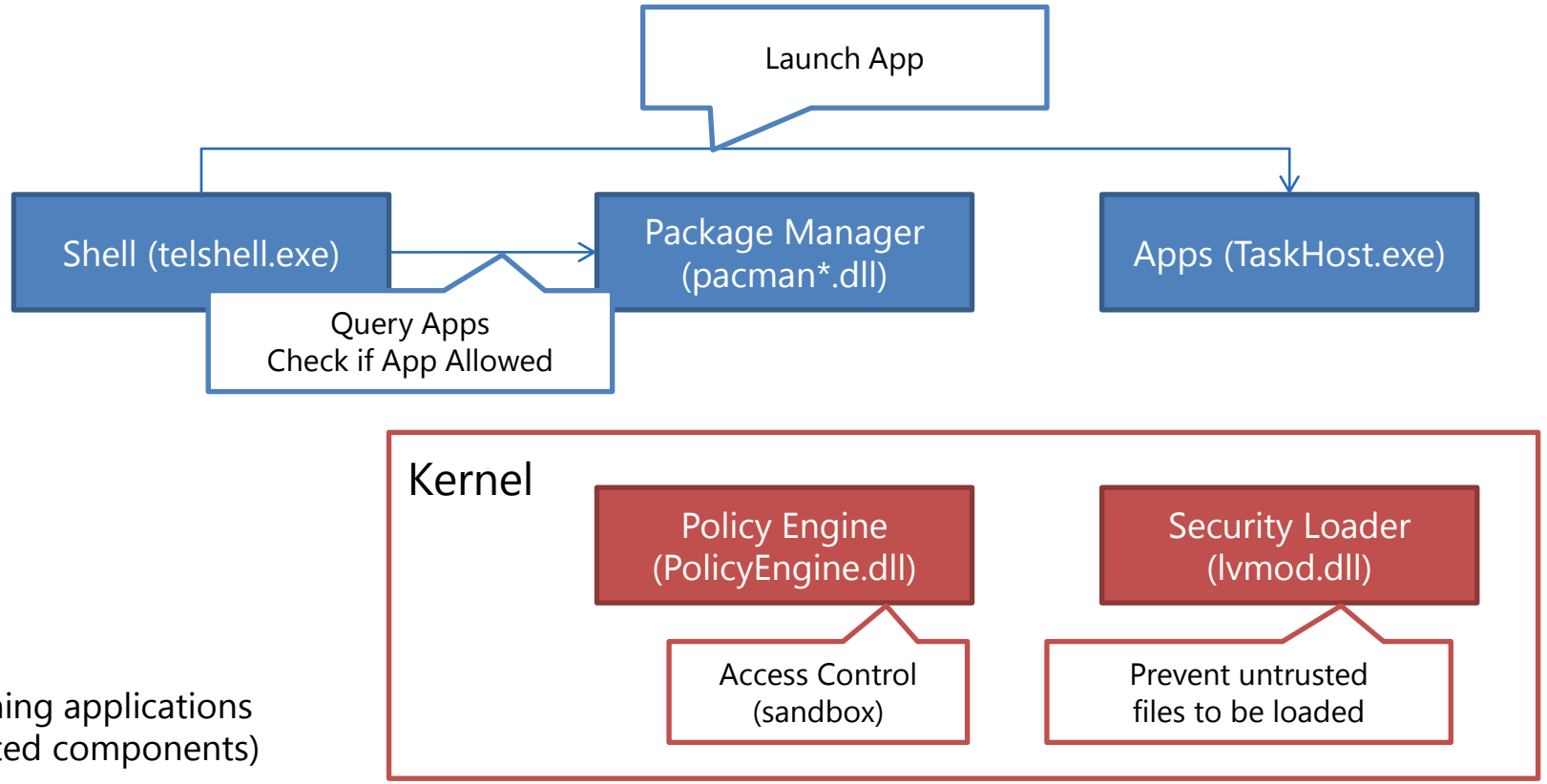
Windows Phone 7 : Capability

- Restrict access like Android's permission system
  - Fewer (and simple) capabilities
- Specific SID for capability
- Special Capabilities for limited apps
  - Some capabilities are not allowed for distribution (without explicit permission by Microsoft)
  - Use OEM's interop service (ID_CAP_INTEROPSERVICES)

Windows Phone 7 : Isolation

- One Chamber for One App
  - Windows Phone 7 creates “chamber” to isolate application data and program
- Almost no access to system resources
  - Normal developers can run only managed (.NET) code
    - Only few developers are allowed to run native code (with WPInteropManifest.xml in the package)
  - Almost no apps can access other apps data

Windows Phone 7 : Isolation Detailed



Running applications
(related components)

- Executable modules and resources are restricted

Conclusion

- Although there are some small “flaws”,
  these OS protect system from being compromised

In other words : what we always find

RISKS AND VULNERABILITIES

What we find : Access Control Vulnerability

- Access to resources which is not allowed (normally)
  - The risk of vulnerability will vary on the resource we can access using exploits
  - Critical one may lead to privilege escalation

What we find : Privilege Escalation

- Make malicious program to run on higher privileges
  - Normal users to System user
    - "system" user in Android is allowed to use almost all system privileges and resources
    - This may lead to complete compromise
  - System user to Administrative user
    - Gaining "root" privilege
  - Keep admin privileges
    - Modify and infect the system permanently
    - This is complete compromise

What kind of vulnerability third-party made?

THIRD-PARTY RISKS

Android : Remote DoS Vulnerability

- "Data Wipe" vulnerability in Samsung and HTC devices
  - Clicking "tel:…" URL triggers "data wipe" feature
  - Special phone numbers (which trigger specific event) are not handled correctly
    - Demonstrated by IMEI display ("*#06#" from remote)
- Denial of Service (force-to-reboot) vulnerability in various Android devices (Sharp, Fujitsu-Toshiba, NEC-Casio…)
  - Similar example on a Japanese smartphone we've found
  - Clicking specific URL (more specifically, calling ***read*** system call for special location) triggers kernel panic and forces device to reboot

Reference:
http://www.guardian.co.uk/technology/2012/sep/27/samsung-htc-phones-remote-wipe

Android : Privilege Escalation Vulnerability

- ACER Iconia Tab / Motorola Xoom OS Command Injection
  - "/system/bin/cmdclient" setuid (and world-executable) program
  - Ability to run any command in root privilege

Reference:
http://forum.xda-developers.com/showthread.php?t=1138228 (ACER Iconia Tab A500)
http://www.xoomforums.com/forum/motorola-xoom-development/12997-rooting-family-edition.html (Motorola Xoom FE)

Android : Access Control Vulnerability

- ZTE Root Shell Vulnerability
  - "/system/bin/sync_agent" setuid (and world-executable) program
  - Ability to run a root shell with a hard coded password

Reference:
http://blog.mobiledefense.com/2012/05/zte-root-shell-vulnerability/

Windows Phone 7 : Vulnerability

- Heap overflow vulnerability in [not disclosed yet]
  - CVE-2005-2096 (vulnerability in zlib -1.2.2)
  - This showed us Windows Phone 7 apps are not vuln-free (such native vulnerabilities can be found)
- Risks of Exploitation
  - If a vulnerable native app has "Interop Services" capability, it can cause disaster (ID_CAP_INTEROPSERVICES)
  - Otherwise it's not much help for bypassing sandbox
    - Just taking control may be not enough for system compromise (because of strong isolation)
    - Fortunately, [not disclosed] didn't have one

Windows Phone 7 : Design Flaw

- Some Windows Phone 7 devices have “backdoor” interop services which enables access resources in many regions
  - Files
  - Registry
  - Physical RAM (?!)
- These services can be accessed from apps with ID_CAP_INTEROPSERVICES capability
  - There are some **non-OEM** native apps (which can access **all** interop services)
- Microsoft should have been separated such services
  - If an application need **an** interop service, **all** interop services will be permitted

Reference: http://labs.mwrinfosecurity.com/assets/128/mwri_wp7-bluehat-technical_2011-11-08.pdf

It was not so difficult.

FINDING VULNERABILITIES

General : Find Similar Hacks

- If device A have been hacked by others, device B (which has similar configuration) may have similar vulnerability
  - Same/Similar chipset
  - Same/Similar vendor

General : Focus on “System” interface

- Original system interface may be disaster
  - Buffer overflow
  - Directory traversal
  - Improper access to file system
- Using…
  - IDA Pro to figure out what interface the device has
  - Custom tools to exploit (or try to exploit)

Android : Diffing source tree

- Applicable for GPL/LGPL portions
  - Diffing between original source code and vendor one
  - AOSP and some vendors (like Qualcomm) serves git repository and makes diffing easier
    - Download every history by cloning git repository and compare each commits to find neighborhood
    - Take a complete diff and investigate “vendor” parts
- 1.3GB total for “Android” Linux kernel trees and thousands of appropriate commits
  - It may require optimization for diffing (if you don’t know which chipset the device uses)

Android : Diffing files and directories

- Access all the files and directories which we can access
  - Just doing this can reveal vulnerability
- Find “third-party” daemons
  - This will help efficient reverse engineering
- Disassemble/Decompile important programs and extract path information (to figure out)
  - Some locations which have “improper access” are difficult to find without reverse engineering

Android : Modules to load

- Check which module is loaded
  and make sure the way to load module is secure
  - If the module is loaded insecurely,
    we could “insert” module to be loaded
  - Symbolic link may help
    (many programs cannot handle symbolic links correctly)

Windows Phone 7 : Updates

- Windows Phone 7 updates are completely separated between Microsoft updates and OEM updates
  - Downloading OEM updates will make reverse engineering very easy (no need to “jailbreak” real device!)
  - *.cab.pkg (CAB files) : Separate update package
- Package file is a gold mine of reverse engineering
  - *.rgu : Registry file (driver information, configurations…)
  - *.policy.xml : Policy XML (used for access control)
  - *.dll, *.exe : Drivers / PE files (to disassemble)

Windows Phone 7 : Symbols

- System symbols for Windows Phone 7
  - If you can retrieve WP7 system binaries (e.g. extract ROM), you can download the symbols from well-known URL <http://msdl.microsoft.com/download/symbols>
  - Loading symbols may break IDA Pro but can be fixed:
    - Start analyzing module without loading symbols
    - Save “Thumb” functions
    - Load symbols
    - For each “Thumb” functions, restore register “T”. (to make functions really “Thumb” again)
    - Reanalyze module from options menu

So, what is the problem?

CONCLUSION

Problems of third-party vulnerabilities

- May not be easy to know
  - Many zero-days
- May not be fixed so fast
  - Varying on vendors
  - May be same on “common” Android vulnerabilities
- May be easy to exploit
  - If the third-party vendor didn’t properly design security
- Definitely easy to find
  - Find vulnerability from 1 million lines or 1 thousand lines

Conclusion

- Vulnerability made by third-party modification may be disaster
- There are some points to find such vulnerabilities
- Vendors must consider security design

Thanks!

Fourteenforty Research Institute, Inc.
http://www.fourteenforty.jp

Tsukasa Oi – Research Engineer
<oi@fourteenforty.jp>