PacSec 2011 Tokyo

How Security Broken?

Android Internals and Malware Infection Possibilities

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

Research Engineer – Tsukasa OI

Background: Android and Threats

- Increasing Share + Increasing Malware
  - 3x malware increases in 2010[1]
  - 2010/08 : SMS malware identified (FakePlayer.A)
  - 2011/03 : “Undeletable” malware found (DroidDream)
- Vulnerabilities and Exploits
  - 2003- : Implementation to prevent exploits (DEP, ASLR...)
  - Mobile devices also can be exploited
    - 2007- : JailbreakMe (payload for iOS)
    - 2011/03 : DroidDream (utilizing two *root*ing exploits)
- Countermeasure : Anti-virus Software for Android
  - Android should be protected like PC

[1] http://www.adaptivemobile.com/

Agenda

- Security in Low Layer
  - Protection in Kernel level
- Android Internals
  - Packages / Permissions
  - Intent / Activity / Broadcast
- Threats and Countermeasures
  - Malware Infection and Impact
  - *root*ing issues
  - Anti-virus software issues

Kernel-level Memory Protection and Android

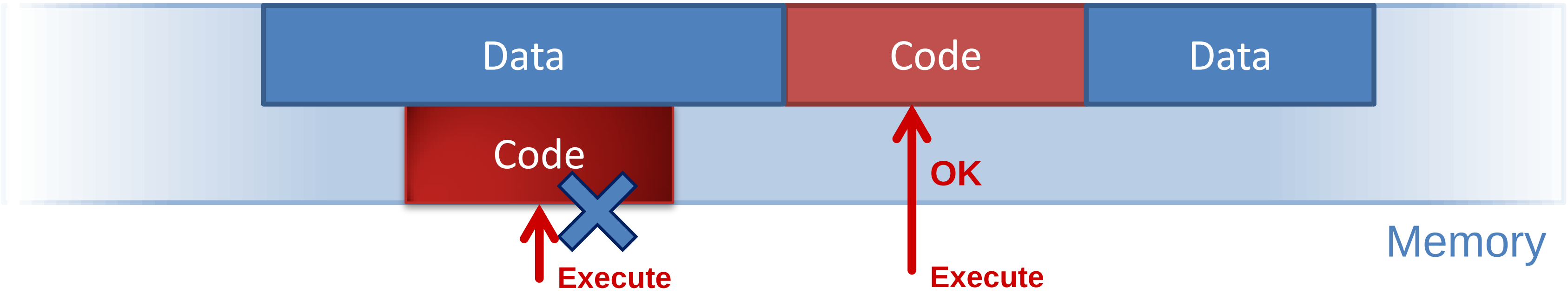
SECURITY IN LOW LAYER

Kernel-level Protection : Implementation

	-2.2	2.3-,3.0-	4.0-	iOS
DEP (Stack)	- [1]	✔ [1]	✔	Supported: 2.0-
DEP (Others)	- [2]	✔	✔	
ASLR (Stack)	✔	✔	✔	Supported: 4.3-
ASLR (Heap)	-	-	? / - [3]	
ASLR (Modules)	-	-	✔ / - [3]	Partially supported: 4.3-[4]

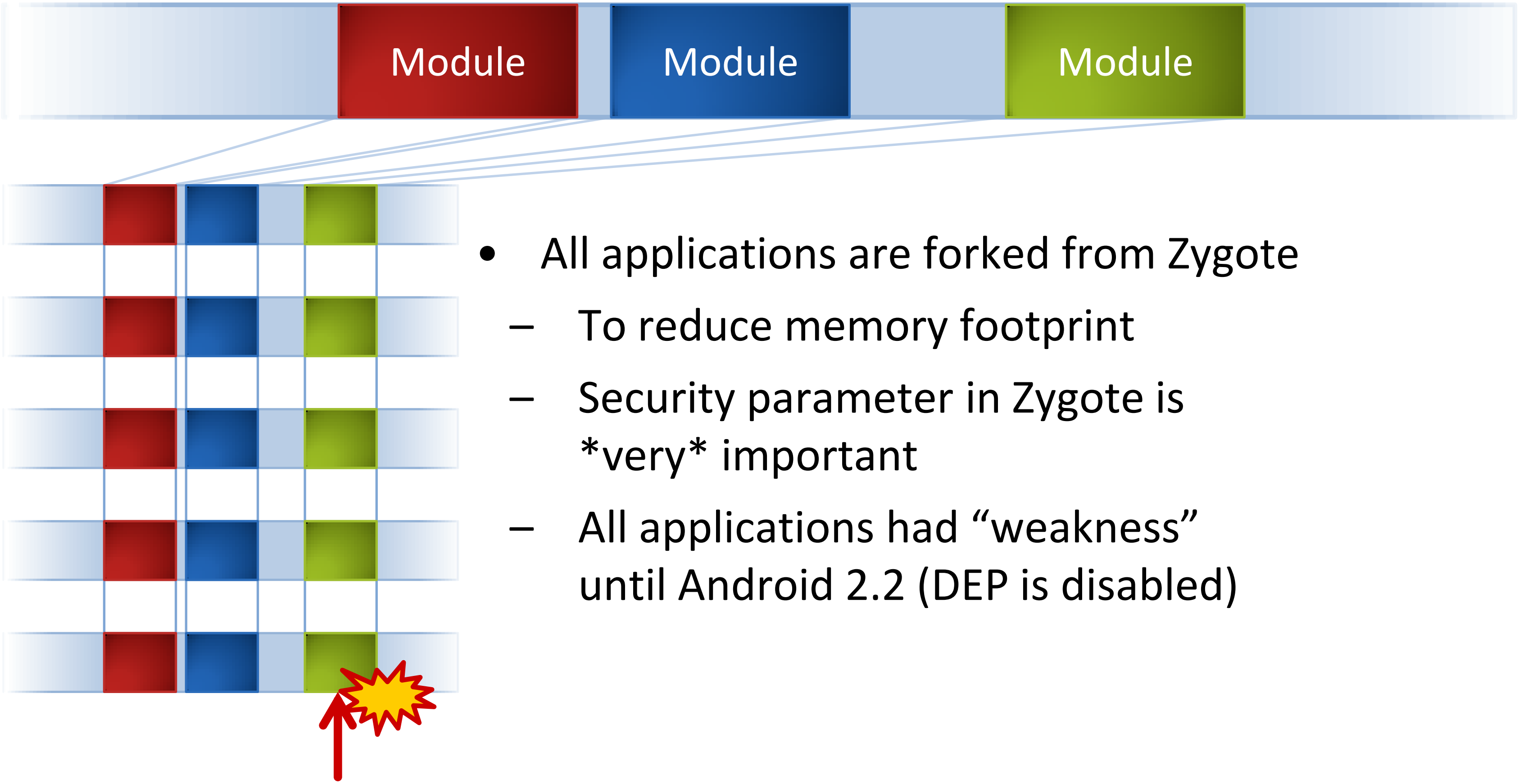
(1) May vary in compiler flags for native applications.
(2) Allocation in portable way
(3) According to the Release note / Result in Android 4.0 emulator image
(4) Only if application supports ASLR

Kernel-level Protection : DEP



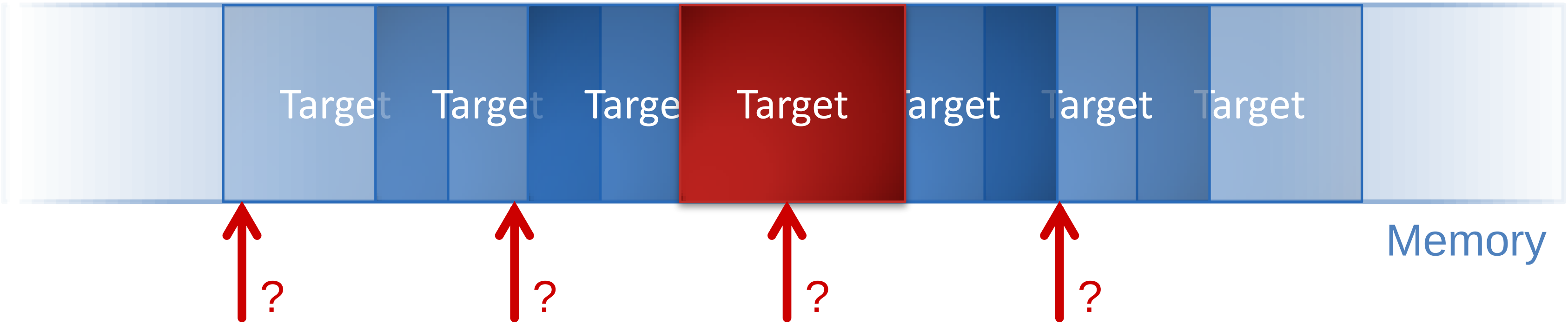
- Distinguish between “data” and “code” in hardware level and Prevent “data” to be executed
- Need a Compiler Flag to enable DEP
  - Not enabled until Android 2.2
  - Kernel *disables* DEP for compatibility
- Solved in Android 2.3

Android Internals : Zygote



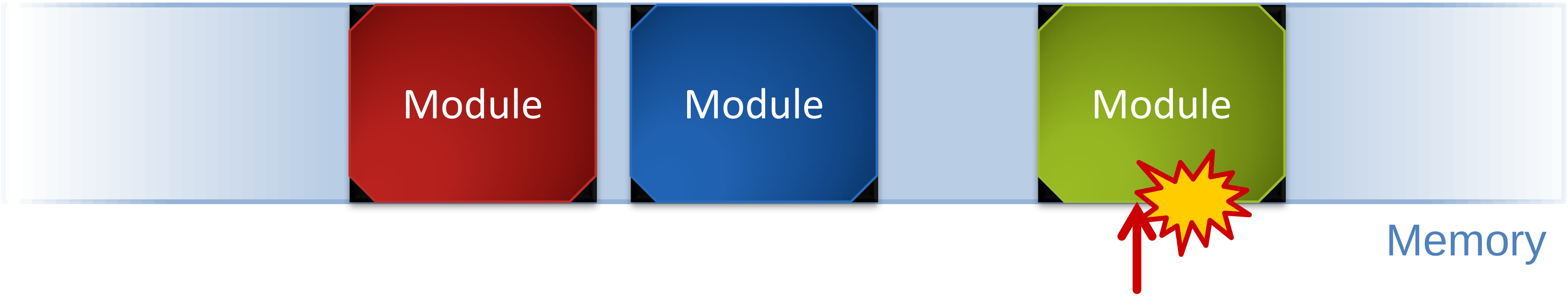
- All applications are forked from Zygote
  - To reduce memory footprint
  - Security parameter in Zygote is *very* important
  - All applications had “weakness” until Android 2.2 (DEP is disabled)

Kernel-level Protection : ASLR



- Randomize Memory Layout to prevent exploits
  - Many of recent exploits utilize *specific* address
- Kernel settings : Randomize everything except heap (OK)
  - But actually, modules (libraries) are not randomized (no good)
  - Because of *Prelinking*

Security Concerns : Prelinking



- Prelinking (user-mode mechanism)
  - Locates system libraries to fixed addresses
  - ASLR is effectively *neutralized* because of Prelinking
- Makes exploitation very easy

Kernel-level Protection : ASLR in Android 4.0?

- 2011/10 : Still no real Android 4.0 device...
  - Android 4.0 SDK emulator image is available now
- Google have announced ASLR is introduced in Android 4.0 [1]
  - Still no ASLR in the emulator image...
  - I expect “proper” ASLR is implemented!

[1] http://developer.android.com/sdk/android-4.0-highlights.html

Conclusion

- Kernel-level Protections are not so effective
  - Possibility: Native Code exploitation
- Improper build settings can be fixed
  - Fixed by default in Android 2.3
- Prelinking can weaken kernel-level protection
  - CPU performance increasing
  - Could be fixed! (Android 4.0)

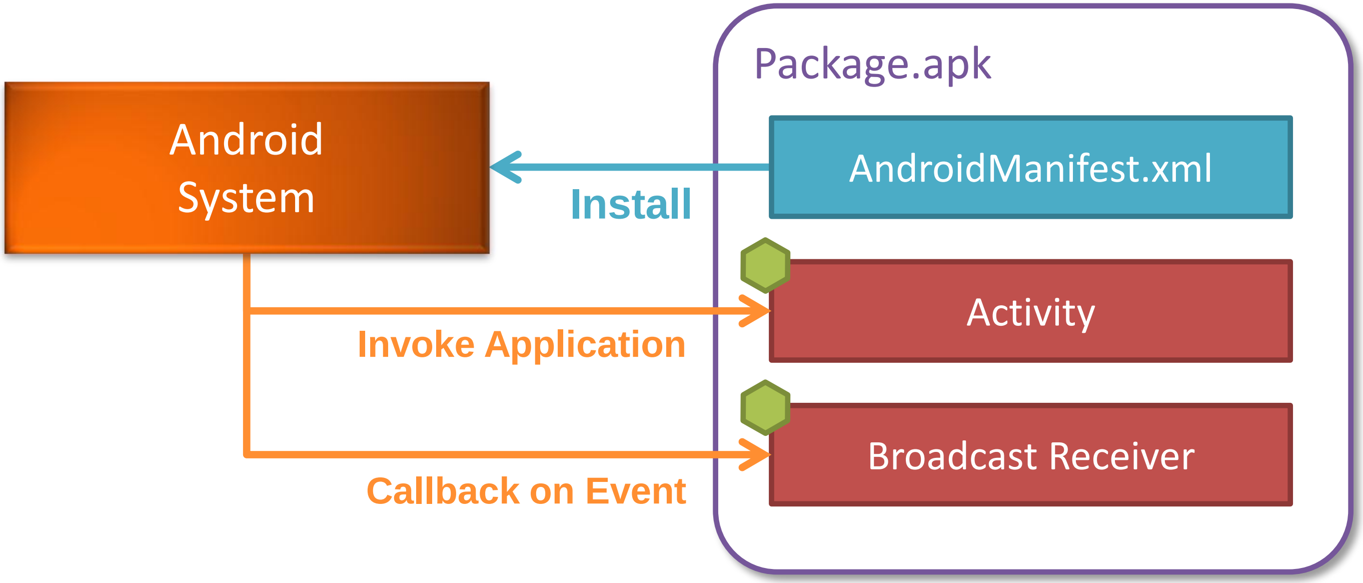
How Android system works?

APPLICATION LAYER MECHANISMS

Android Applications

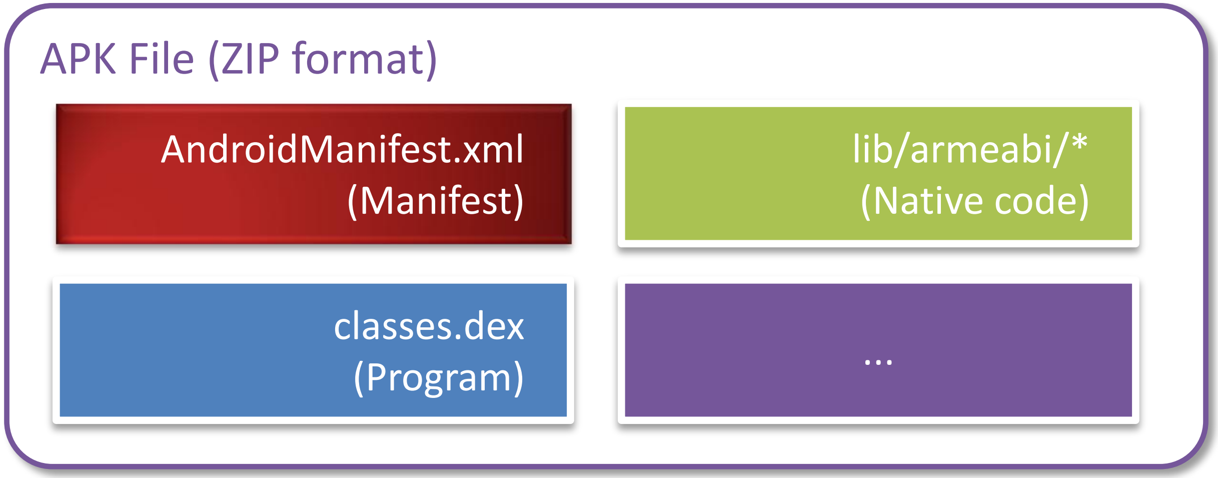
- Quite different than other platforms
  - Intent-based communication
- Android Internals
  - Package and Manifest
  - Permission system
  - Intent
    - Activity
    - Broadcast and BroadcastReceiver
    - …

Android : How application work



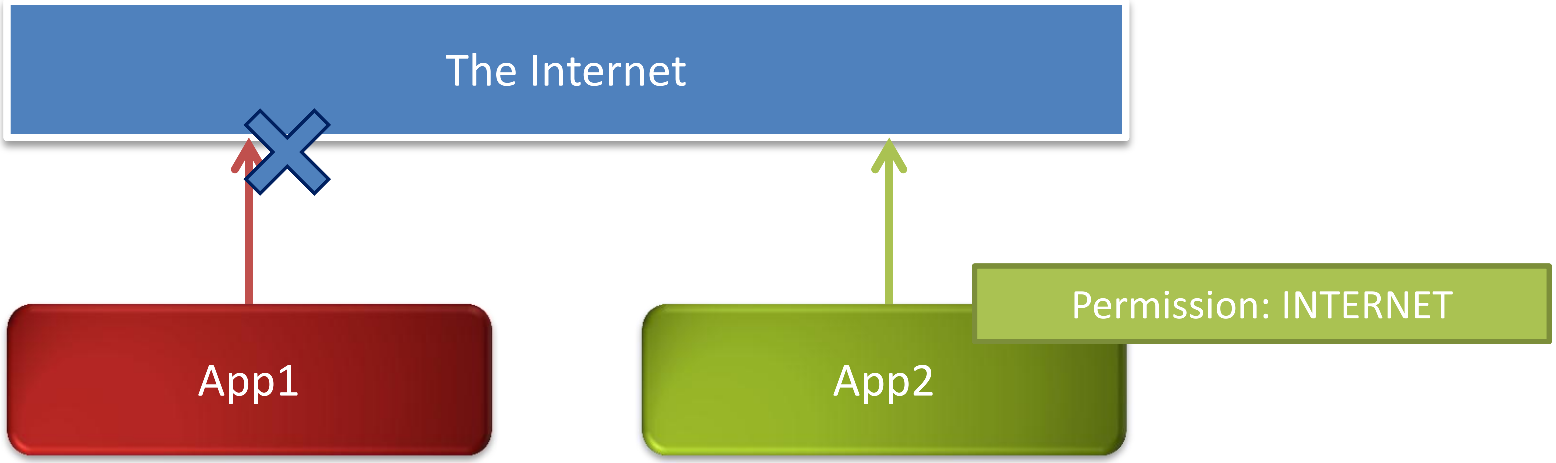
- Applications are contained in the Package
- Register how “classes” are invoked using Manifest
  - System calls application “classes” if requested
  - Activity, Broadcast, ...

Android : Package



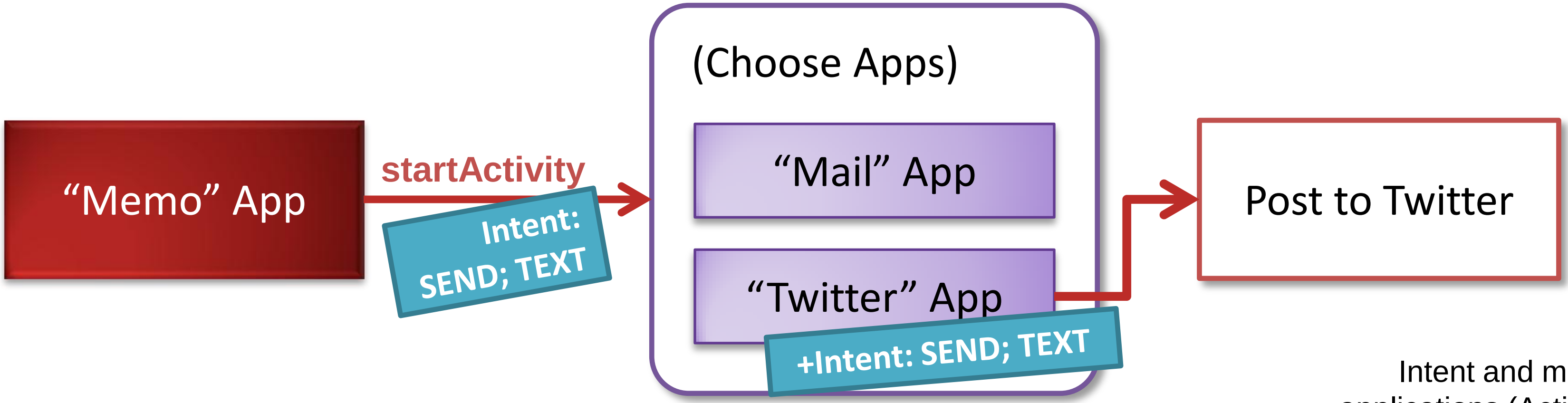
- Package itself is only a ZIP archive
- AndroidManifest.xml (Manifest)
  - Application information, permissions
  - How classes can be called (Activity, BroadcastReceiver...)
- Immutable on installation
  - Can be “updated” along with whole package

Android : Package (Permission)



- Abstract “Capability” in Android system
  - More than 100 (internet connection, retrieve phone number...)
- No permission, No operation
  - Permission is the key of Capability

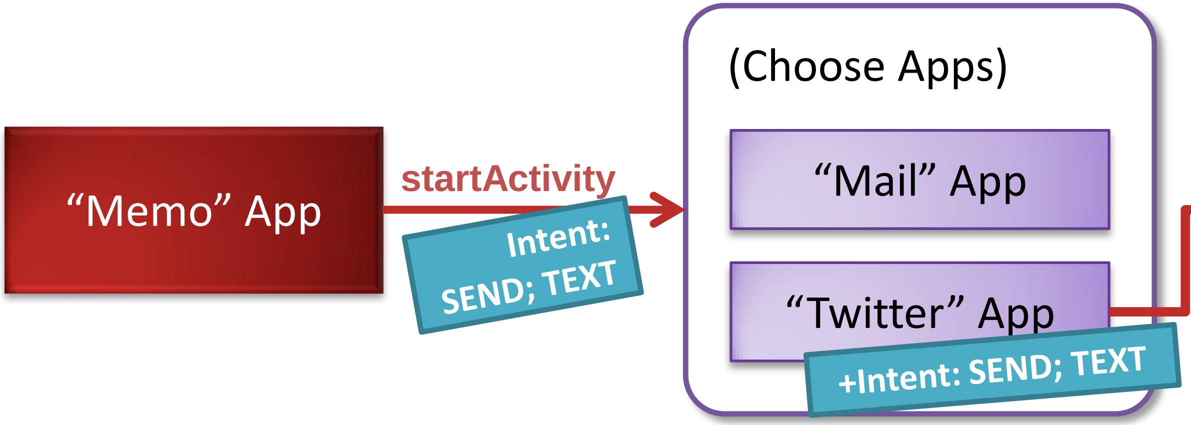
Android : Intent



Intent and multiple applications (Activities)

- Intent
  - Send/Receive Message containing action, target, ...
- Intent are used in many form
  - Inter-Application Communication
  - Event Notification

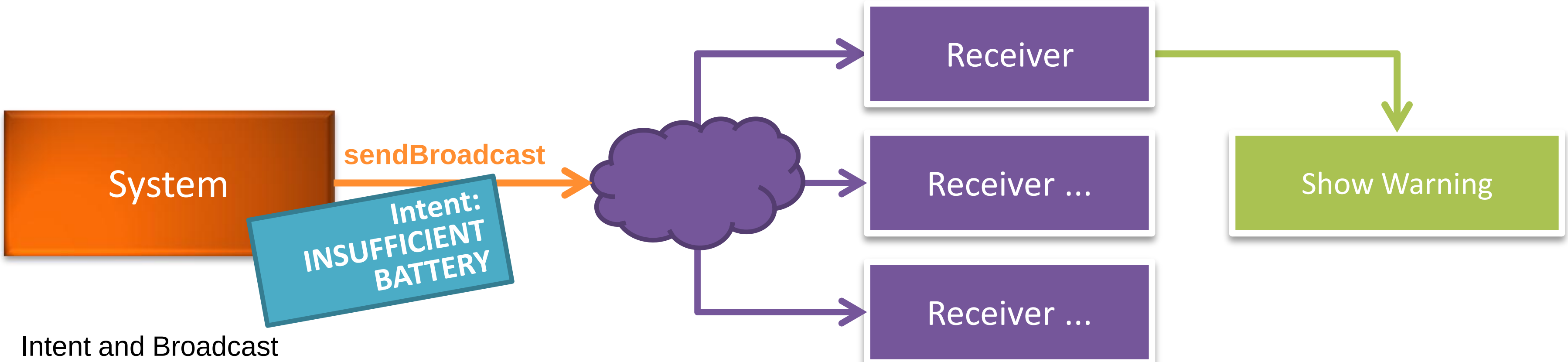
Android : Intent (Activity)



Intent and multiple applications (Activities)

- Activity = Unit of "Action" with User Interface
  - Specifying object type (target) and action, Activity is called by the system automatically

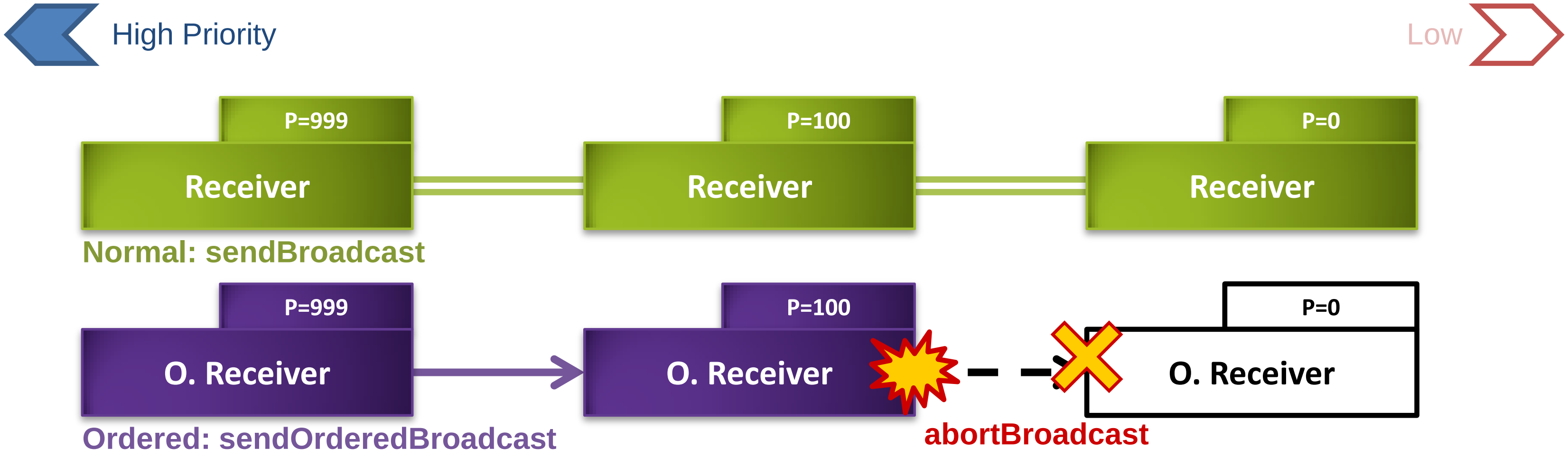
Android : Intent (Broadcast)



Intent and Broadcast
(Battery Event Notification)

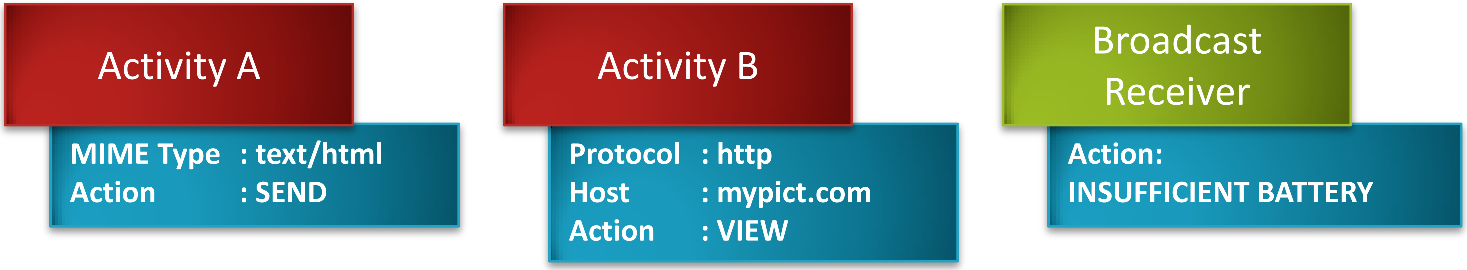
- Broadcast : Feature to Receive system/app-generated Events
  - All associated (and registered) BroadcastReceiver classes are invoked

Android : Intent (Ordered Broadcast)



- Broadcast can have “Order”
  - Few broadcasts are sent “Ordered”
- Ordered Broadcast
  - BroadcastReceiver class is invoked in order of Priority (later)
  - Abort Processing Broadcast using “abortBroadcast” method

Android : Intent Filter



Activity A

MIME Type : text/html
Action : SEND

Activity B

Protocol : http
Host : mypict.com
Action : VIEW

Broadcast Receiver

Action:
INSUFFICIENT BATTERY

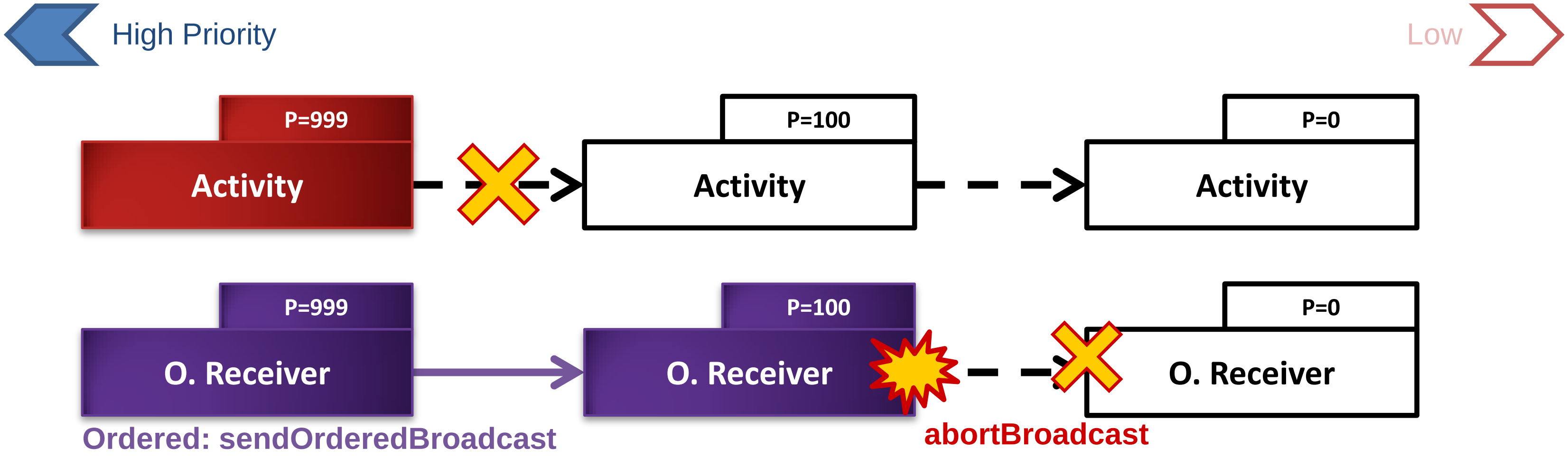
e.g. Application to upload text

e.g. Application for specific website

e.g. Battery-related service

- Similar to File/Protocol Association in Windows
  - Action (what to do), Category (how to do)
  - File Type (MIME), Location, Protocol...
- Specify in the Manifest (AndroidManifest.xml)
  - Android System manages all Intent Filters

Android : Intent Filter (Priority)



- Priority in Intent Filter (associated with Activity / Broadcast)
  - Higher Value = Higher Priority
  - Ordered Broadcast
  - Activity

Summary

- Android System
  - Package / Manifest
  - Permission System
- Intent-based Features
  - Activity
  - Broadcast
    - Ordered or not
- Intent Filter to help inter-application communication
  - Flexibleness
  - Priority

Android Malware and Countermeasure Issues

SECURITY AND THREATS

Android Security and Threats

- Many malwares and Many anti-virus software
  - Malware impacts
  - Is Anti-virus software effective?
- Malware
  - Trends and Characteristics
- How Anti-virus software work?
  - Issue: Insufficient Privileges
- *root*ing issues
  - How security has broken?
  - Countermeasure, and problems still left

Android Malware : 2009

- Found on 13 Jan (McAfee)
  - CallAccepter, Radiocutter, SilentMutter
  - Targeting *root*ed Android 1.0 devices
  - Denial of Service
- Released on 26 Oct : Mobile Spy
  - Paid Spyware (Record SMS, GPS, incoming/outgoing calls)
  - Similar to “Karelog” (2011) in many ways
- Different Type of Attack
  - Not so related to Cybercrime

Android Malware : 2010

- Found on 10 Aug (Symantec) : FakePlayer.A
  - First “real” Android threat
  - Distributed in Russian website masquerading as a harmless movie player
  - Making money utilizing Premium SMS

- Checkpoint : Modern Cybercrime and Android
  - Thereafter, Android malware became more “malicious”

Android Malware : 2011

- January : Repackaged Android Apps
  - Redistribute “tainted” Android applications
- March : Undeletable Malware
  - Install code to the System Partition
- June : Self-updating Malware
  - Download and Execute the code dynamically (DexClassLoader)
- July, October : Malware utilizing Application Updates
  - Updated application include malicious code

Android Malware : Characteristics

- Classification
  - Spyware
  - Backdoor
  - Dialer (utilizing premium services)
- China, Russia...
  - APN/telephone number in specific country
  - String resources
- Messaging Channel
  - HTTP
  - SMS

Android Malware : Characteristics (Premium Services)

- Paid SMS/telephone services
  - Japan : “Dial Q2”
  - Paid numbers/services have no borders
- Utilizing Premium Services : Dialer
  - Dial Premium Services and Make Money *directly*
  - Dialer is reborn in Japan
  - Android ≒ Telephone

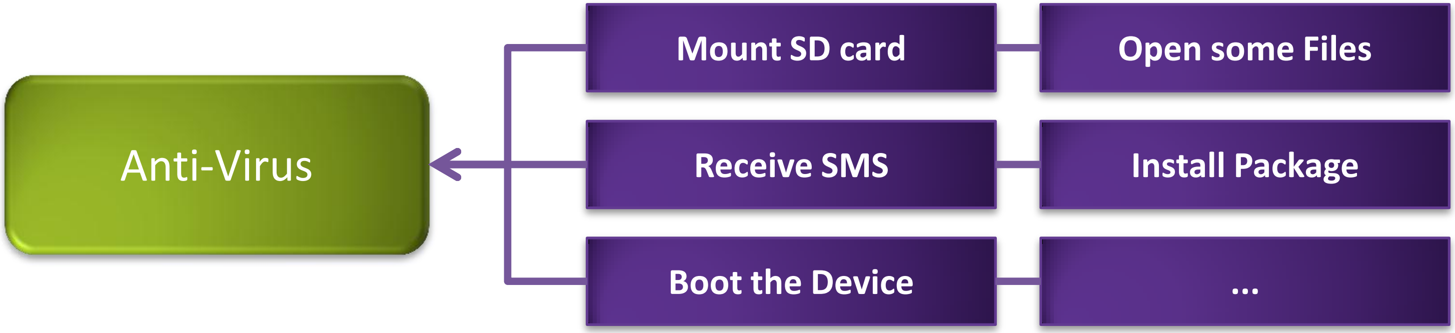
Android Malware : Utilizing Intent Filter

- Receive Broadcasts to (steal information | run automatically | ...)
  - 39/44 malware samples
- “Receiving SMS” is a Ordered Broadcast event
  - BroadcastReceiver with higher priority can *hide* SMS message (hidden from preinstalled SMS application)
  - Can hide malicious commands
  - 14/44 malware samples

Android Malware : Evolution

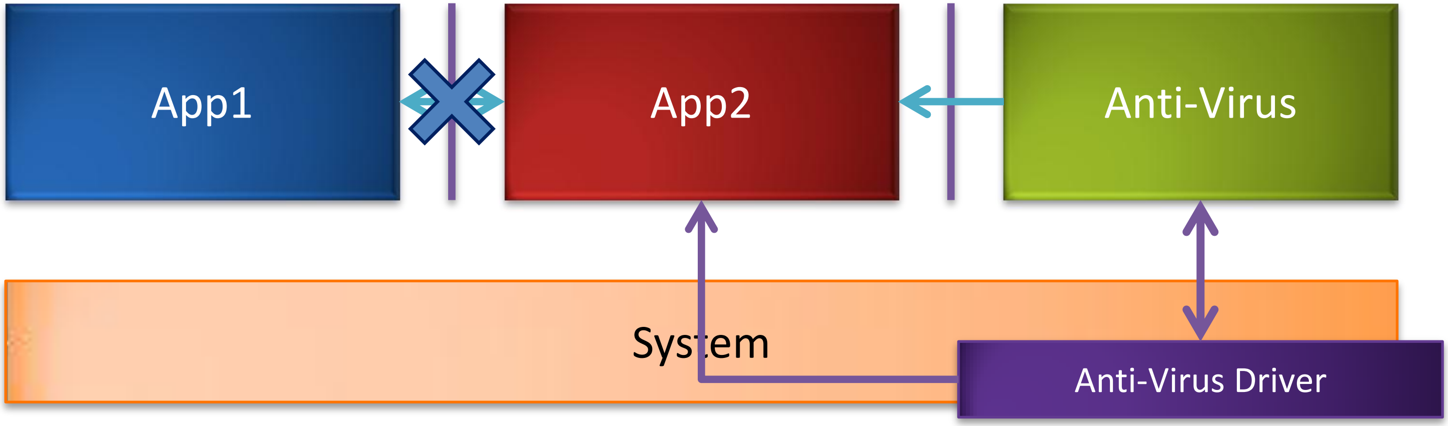
- Still no “real” obfuscation
  - Easy to analyze
- Evolving Rapidly
  - DroidDream
    Use exploits to gain root privilege and install malicious packages silently
  - Plankton
    Download DEX file (Dalvik byte code) and Execute it dynamically using class loader
- Refined Android malwares will cause problems (specially, the one utilizing *root*ing techniques)

Anti-virus : How it works?



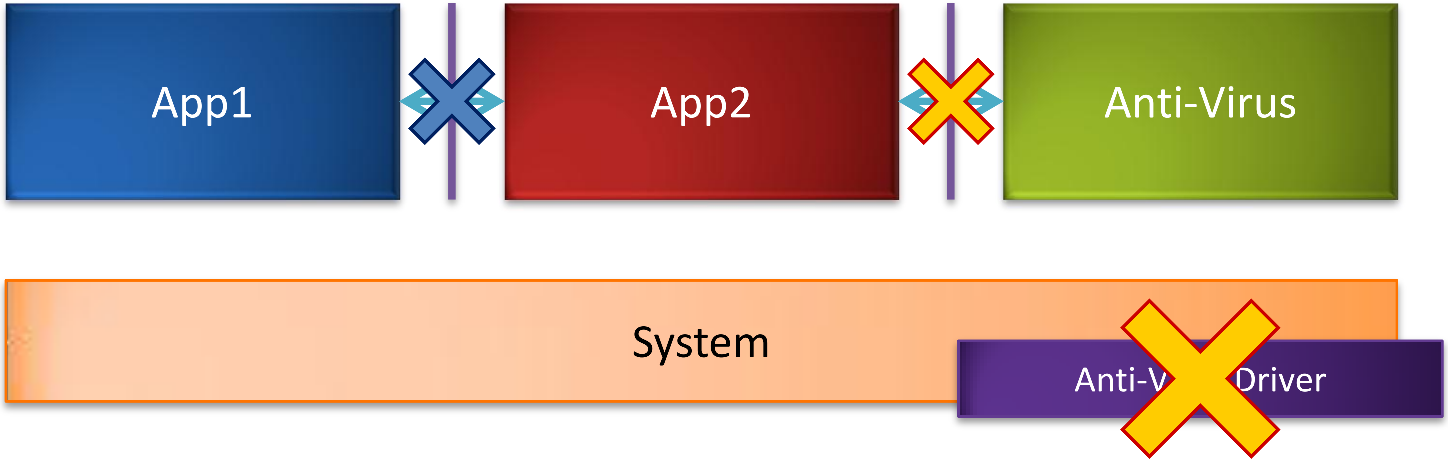
- Utilizing *many* of Intent Filters and Broadcasts
  - Real-time scan (partially)
  - Scan Downloaded Files / Applications
  - Scan SMS messages

Anti-virus : Issue by Android Security Design



- Anti-virus software is working as a normal Android app
  - Normally implemented as a driver (PC)

Anti-virus : Issue by Android Security Design



- Android as a Sandbox
  - Prevent Access to Other Processes
  - Blocks Anti-Virus software access as well
  - No driver can be installed

Anti-Virus : Issues

- Collecting Samples
  - Vary in Security Vendors
  - Android Market : Automated Crawler is Prohibited

Anti-virus : Same Privilege

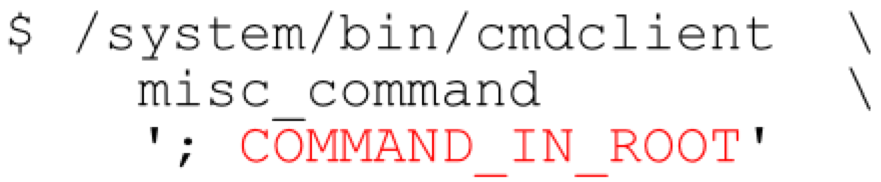
- Same Privilege : Malware and Anti-virus Software
  - Can Neutralize each other
- Dynamic Heuristics is not easy
  - No way to intercept system calls
  - Signature issues
  - Protect partially
    - Still, normal existing malware can be detected and warn to the user
- If malware can gain higher privilege...
  - Gaining root privilege = *root*ing

*root*ing

- Gain Administrator Privileges (not available by default)
  - Specially, utilizing local vulnerabilities
- *root*ing vulnerabilities
  - CVE-2009-1185 (exploid)
  - CVE-2010-EASY (rage against the cage)
  - CVE-2011-1149 (psneuter)
  - CVE-2011-1823 (Gingerbreak)
  - (no CVE number yet) (zergRush)
- Chip/Vendor-specific vulnerabilities!

*root*ing : Vulnerabilities (1)

- Logic Error in *suid* program
  - Some Android Tablet: OS command injection

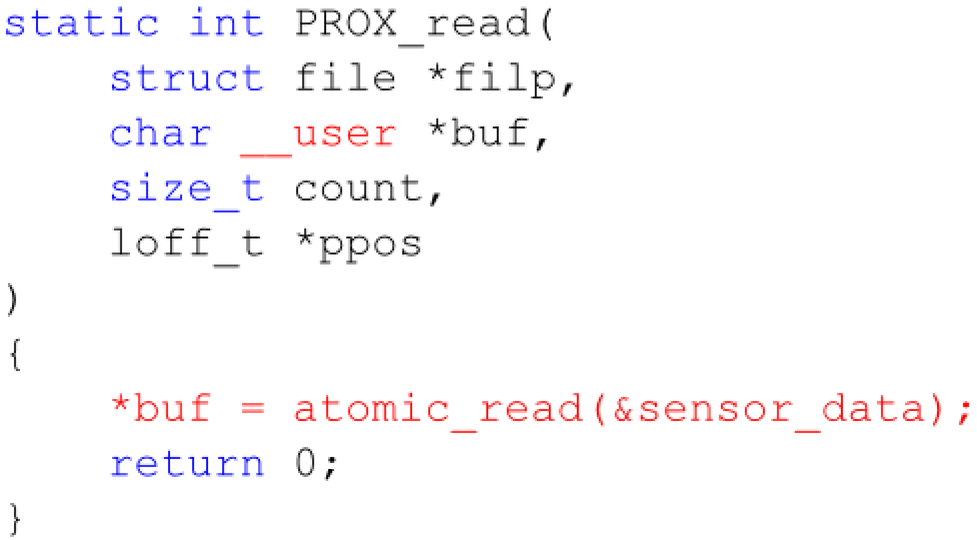


```
$ /system/bin/cmdclient   \
     misc_command          \
     '; COMMAND_IN_ROOT'
```

Can invoke arbitrary command in root privileges.

*root*ing : Vulnerabilities (2)

- Improper User-supplied buffer access
  - Some Android smartphone: Sensor Device



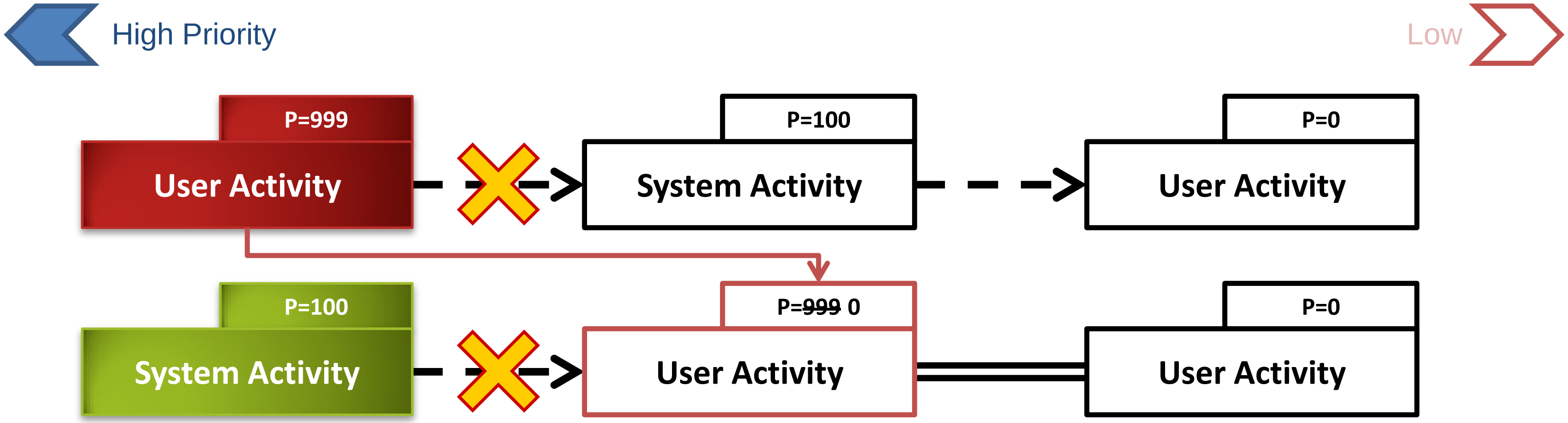
```
static int PROX_read(
        struct file *filp,
        char __user *buf,
        size_t count,
        loff_t *ppos
)
{
        *buf = atomic_read(&sensor_data);
        return 0;
}
```

Can write 0 or 7 (according to the sensor data) to arbitrary user memory, bypassing copy-on-write.
Destroying *setuid* function can generate root-privilege process.

*root*ing : The Real Problem

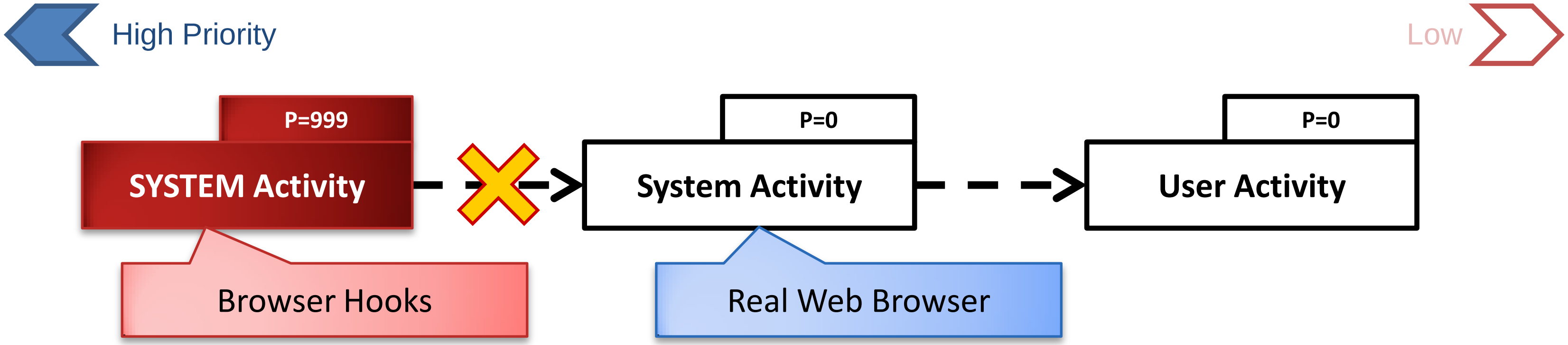
- Malware can Exploit same Vulnerability
  - Malware could gain higher privileges
  - Avoid Anti-virus software
- *root*ing breaks some security mechanisms
  - Intent Filter priority value (associated with Activity)
  - Permission System
- Security software may be neutralized

Broken Security : Activity Priority (1)



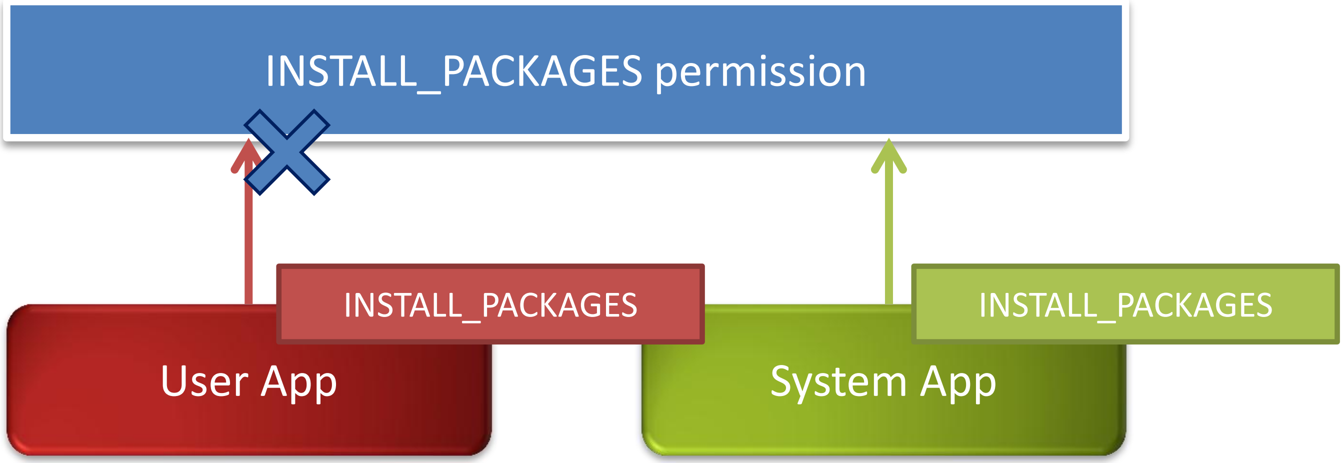
- High priority Activity enables hooking
  - Dangerous
  - Reserved for System Applications

Broken Security : Activity Priority (2)



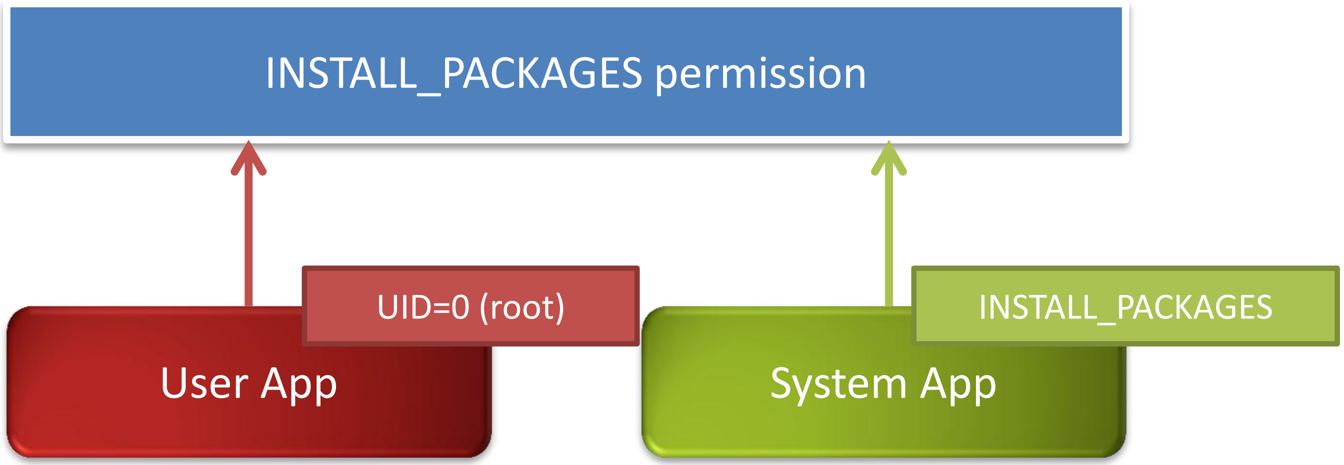
- If malicious package is installed in the System Partition, malware can utilize higher priority of Activity
  - Hook implicit Intents
  - e.g. Hook web browser-related Intents for phishing
    - Does not work since Android 3.0
      (because of Browser application changes)

Broken Security : Permission (1)



- Reserved Permissions
  - Only available to Vendor Packages or Preinstalled Packages
  - Bypassing : There's a way other than modifying System Partition...

Broken Security : Permission (2)



- In root process, all Permissions are granted
  - No additional security checks (not even manifest checks)
  - Enables silent installation for example
    - GingerMaster utilizes this behavior (indirectly)

*root*ing : Countermeasures and Issues (1)

- Remove found vulnerabilities
  - Not so easy to patch...
    (http://www.ipa.go.jp/about/technicalwatch/pdf/110622report.pdf)
- Limit root user : Linux Security Modules (LSM)
  - SHARP Corp. : Deckard / Miyabi
    - /system partition is prohibited (cannot be re-written)
    - ptrace (Debugging) is prohibited
    - Prevents DroidDream / DroidKungFu infection
  - Prevent root user to be utilized
    - Current LSMs are not enough though...
    - Black Hat Abu Dhabi 2011

*root*ing : Countermeasures and Issues (2)

- Limiting *root* user is not enough
  - Permission checks
  - Making secure OS policy is difficult
  - Anti-virus software privilege is left weak
- Protection specific to Android
- Enabling Privilege Escalation for Security is needed!

Conclusion

- Malware and Anti-virus software is evolving
  - But, we cannot protect whole system.
- *root*ing breaks security and neutralize Anti-virus software
  - Even if malware could be found, it could be undeletable.
  - To encounter, we need privilege improvement
    and whole new protection system.

Can Android protected?

BOTTOM LINE

Is Android Protected? (1)

- Vulnerability Attacks
  - Android depends on many of Native Code (e.g. WebKit)
  - Kernel-level protection is currently not so effective
    - Compiler Flag (DEP)
    - Prelinking (disabling ASLR)
  - If vulnerability is found in Android,
    it is not difficult to exploit.
  - It could possibly change in Android 4.0

Is Android Protected? (2)

- Malware vs. Anti-virus software
  - Malware (as a Trojan horse) works as a spyware, backdoor or dialer utilizing Android features
  - *root*ing can make Anti-virus software completely useless
- Currently, it is Difficult to protect Android devices

What to do (1)

- Technical Responsibility : Android Project (AOSP et al.)
  - Make security mechanism Strict
    - System Call-Level Protection (LSM)
    - Secure Android Framework
  - Help making Security Software
    - e.g. Giving higher privileges for specific software
  - Make Kernel-level Protection Better
    - Removing Prelinking, ...
    - ... it seems to be done!

What to do (2)

- Technical Responsibility : Device Vendor
  - Fix existing vulnerabilities (prevent existing malware)
  - Verify vendor customization
    - Not to break Android security mechanisms
      (and not to prevent user rights)

Conclusion

- Protection for Android is not enough, but not impossible to solve
  - Currently, Users must be aware of threats
  - Possibly, need to take resolute steps
- Work together to improve Android security whilst keeping platform open

Thank you

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

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