A framework for on-device privilege escalation exploit execution on Android

IWSSI/SPMU 2011 at Pervasive 2011
12. June 2011, San Francisco, CA, US

Sebastian Höbarth, **Rene Mayrhofer**
Fachhochschule Hagenberg, AT
rene.mayrhofer@fh-hagenberg.at

Comparing mobile platforms

	Android	**iOS**	**Blackberry**	**Symbian**
Restricted to App Store	no	yes	no	no
Sandbox for applications	yes	some (Safari)	no (unknown)	no
Signed applications	yes	yes	yes	yes
Capabilities for applications	yes (all-or-nothing)	no	yes (configurable)	yes (configurable)
Garbage collection	yes (Java)	no (Objective C)	yes (Java)	no (C++)
Exploit prevention	no NX no ASLR	NX stack+heap no ASLR	unknown	no NX no ASLR
On-device encryption	no	yes, but problematic	yes	no

Comparing exploit prevention techniques

Table: Exploit mitigation techiques

Protection	Android	W. Mobile	Iphone
Stack NX	-	-	Yes
Heap NX	-	-	Yes
Cookie	-	Yes, 16 bit	-
Random Libs	-	-	-
Random Stack	Yes	-	-
SEH	-	stack	-

Table by Nicolas Economou and Alfredo Ortega, Presentation “Smartphone (in)security” at CanSecWest 2009

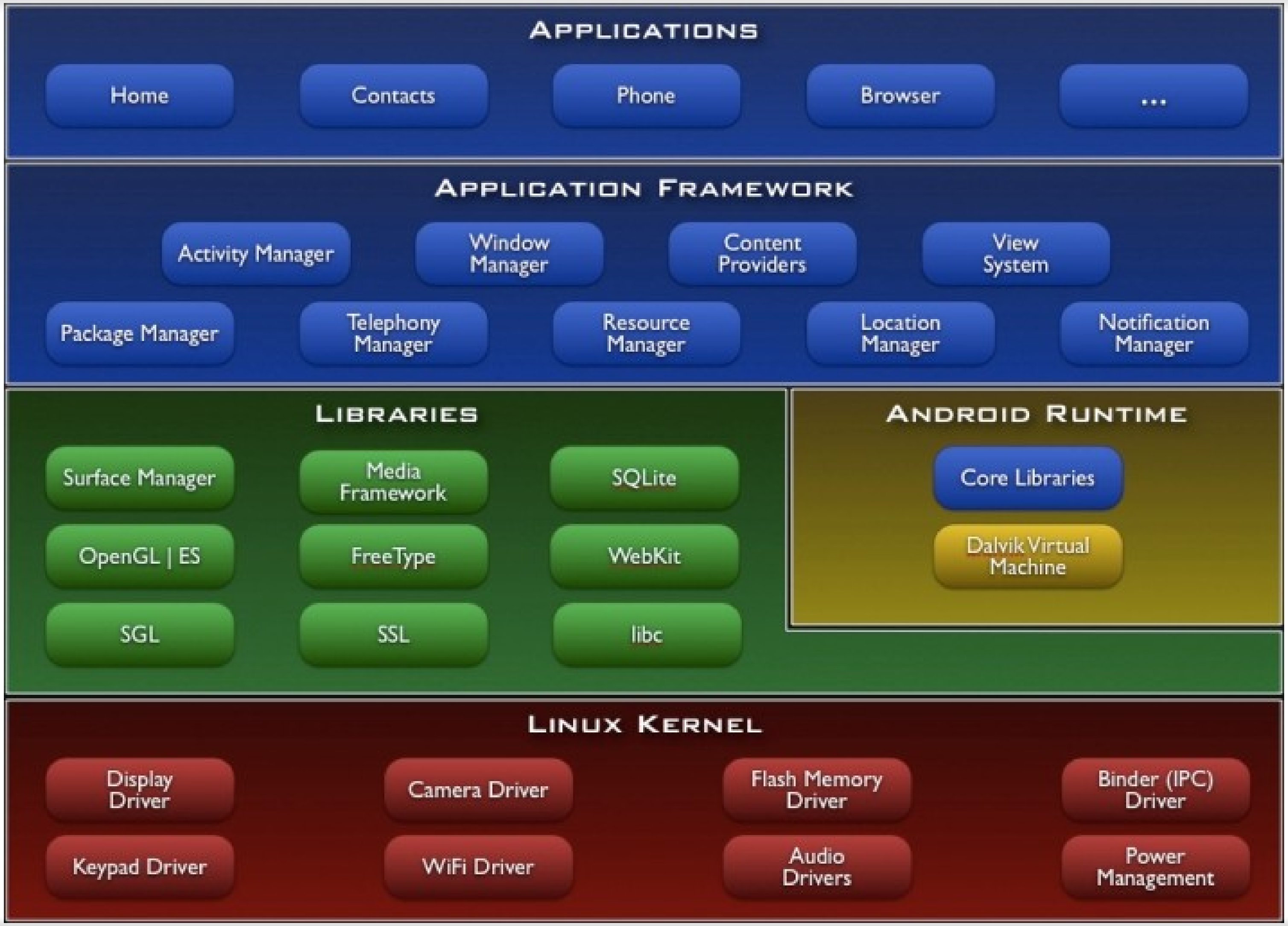
iPhone vs. Android security concepts

- **Closed** eco-system
- All applications need to go through Apple Store, checked by Apple

  unless device has been "jailbroken"…
- Sandboxing restricted to few applications (e.g. MobileSafari)
- Some exploit protection measures
- No further security measures
  - Data Execution Prevention (DEP) using ARM XN feature
  - but no ASLR in kernel

- **Open** eco-system
- Arbitrary applications can be installed by the user (may need to enable "USB debugging for installation from PC or download on-device and install)
- All applications are sandboxed
- No further security measures
  - no ASLR, no DEP
  - no on-device encryption

Android Architecture



Android Security Architecture

Applications must be signed for installation

- may be self-signed by the developer, therefore no requirement for centralized application Q/A or control
- signature allows non-repudiability (if the public key/certificate is known)
- signature by same private key allows applications to share data and files
- automatic application updates possible when signed by same private key

Upon installation, the package manager creates a dynamic user ID for each application ⇒ Application sandbox

- all application files and processes are restricted to this UID
- enforced by Linux kernel and therefore same restrictions for all code (Java + native)
- by default, even the user and debugging shells are restricted to a special UID (SHELL)
- permissions granted at installation time allow to call services outside the application sandbox

"rooting" to gain "root" access (super user / system level access on UNIX without further restrictions)

What can be done after rooting?

Development

- Installing custom root, recovery, and system images
- [on devices without NAND lock] changing files on the `/system` partition

General

- Reading all files on `/system` and `/data` partitions (including centralized system databases, e.g. accounts)
- Changing kernel settings, e.g. CPU over-/underclocking, IPv6 address privacy
- custom routing, WiFi/3G connection sharing on devices that do not support “tethering” out of the box or have it disabled by the manufacturer/carrier
- Installing/using applications that require root access, e.g. Backup/restore, OpenVPN, Webkey, SSH daemon, etc.

Rooting Overview

(1)Achieve temporary root privileges

- on “developer” devices (Google ADP1 / G1, Nexus One, Nexus S) simply with
  - `fastboot oem unlock`
- booting into recovery mode on devices that allow this without restriction
- exploiting an implementation bug
  - either during normal system run-time
  - or in the boot loader code to get into recovery mode (see above)
    e.g. flawed signature checking allows booting custom `recovery` image and from within that running image, flashing a new `system` image

(2)Achieve permanent root privileges

- by flashing a new `system` image with pre-installed binaries for root access
- by installing a `/system/(x)bin/su` binary with “setuid” Bit set and the “SuperUser” application that will ask the user on each (first-time) access
- by modifying the `root` image, e.g. to set the global property `ro.secure=0`

(3)Secure system against abuse by other applications

Details: Linux kernel

- “**root**” is the super-user, equivalent to system-level access
- kernel implements DAC (Discretionary Access Control) on filesystem (includes kernel virtual files, e.g. /proc, /sys, etc.)
- optional MAC (Mandatory Access Control) schemes available in upstream kernel
  - SELinux (NSA, flexible, comprehensive, but complex)
  - SMACK (simpler, path-based, used on MeeGo)
  - TOMOYO (path-based, more flexible than AppArmor)
  - AppArmor (simpler, path-based, used on Novell and Ubuntu servers)
- could all be used to further restrict root user, often based on application context

  But none of them used in Android at this time!
- goal is therefore to achieve root privileges

Details: filesystem ACLs

- **read**, **write**, **execute** bits for owning **user**, **group**, and all **others**
- additional bits available, e.g. **setuid** and setgid
  - ⇒ binary called with privileges of owner, not privileges of caller
  - ⇒ typical combination is a file owned by root with setuid Bit set
- setuid binaries used on many UNIX/Linux systems to allow normal users to perform administrative tasks (e.g. passwd) or for arbitrary code elevation (e.g. su, sudo)
- for controlled root access, simple and effective method on Android
  - no changes to any existing system binaries
  - only need one additional binary installed with setuid Bit set
  - typically `/system/(x)bin/su` as on standard UNIX systems, but with Android-specific GUI to ask user for permission on root access

Android Debug Bridge (adb)

- Must be enabled by user (USB Debugging)
  - but then available over USB, WiFi, or locally on device
- Supports debugging, file transfer, package installation, reboot control, etc.
- Normally runs as user SHELL (uid 2000)
- Can be restarted as user root (uid 0)
  - with global property `ro.secure=0`
  - then can call `adb root` to restart
  - or use one of the known exploits to force `adbd` to retain root privileges
  - will then support e.g. `adb remount` to remount `/system` with read-write option

Devices with NAND-Lock

- Some HTC devices have a “NAND lock” implemented in their kernel and boot loader
  - also called “S-On” in contrast to “S-Off” (no additional protection)
- Boot loader sets global flag “S-On” for booting into Android and “S-Off” for booting into recovery mode
- NAND access is moderated by baseband (radio) processor, application processor has to go through it
- Radio firmware prevents write access to `system` (and sometimes `recovery`) NAND partitions even with root privileges and when `/system` has been re-mounted for read-write access
- Currently known devices with NAND Lock: all HTC Desire and EVO variants
- Other devices “only” have a boot loader that verifies signatures before installing updates for `recovery` and `system` and before booting a kernel (e.g. Motorola Milestone)

Rooting: example exploits

Exploits used for gaining temporary root privileges

- Android <= 1.6: missing input sanitization in udev firmware loading (CVE-2009-1185)
- Android <= 2.2: remapping shared memory (and system properties)
- Android <= 2.2: overflowing limit of processes for restricted SHELL user (`rageagainstthecage`)
- Android <= 2.3: restricting access to shared memory (`psneuter`)

http://intrepidusgroup.com/insight/2010/09/android-root-source-code-looking-at-the-c-skills/

Framework steps for permanent rooting

Independent of specific exploit, as long as running code has (temporary) root privileges

(1) Remount /system for write access

(2) Install new binary with setuid Bit set ⇒ e.g. “su” binary with SuperUser companion application

- Exploit framework binary installs itself to /system/bin with setuid
- Alternative: set setuid Bit on existing binary /system/bin/sh, possible due to bugs in YAFFS2 code in combination with NAND lock

(3) Remove ACL restrictions from accounts and settings SQlite database files

(4) Binary with setuid Bit is called at any future time when root privileges are required ⇒ permanent privilege escalation

Future work: working around NAND lock

Automate process of creating custom NAND images

(1)Extract `boot` or `recovery` NAND partition to temporary files

(2)For `boot` partition:

   a)extract initramfs image

   b)modify main boot script to set `ro.secure=0`

   c)repack initramfs image

(3)Write `boot` partition to NAND

Conclusions

Android security measures are not sufficient

- Recommendation 1: ASLR, NX exploit prevention; audit system level code
- Recommendation 2: more fine-grained application permissions (e.g. Internet)
- Recommendation 3: allow user to choose which permissions to grant each application instead of all-or-nothing at installation time
- Recommendation 4: MAC (kernel policies) in addition to DAC (filesystem ACLs) at least for critical binaries, better for application sandboxing (cf. SELinux application sandbox in current Fedora distribution)

- Framework + example exploits source online at **http://openuat.org/android-exploit-framework**

“Portability is for people who cannot write new programs.”

Linus Torvalds, 1992-01-29, comp.os.minix

Linux is currently one of the most portable operating system kernels...

Thank you for your attention!

Slides: http://www.mayrhofer.eu.org/presentations
Later questions: rene.mayrhofer@fh-hagenberg.at
rene@mayrhofer.eu.org

OpenPGP keys: 0x249BC034 (new) and 0xC3C24BDE (old)
717A 033B BB45 A2B3 28CF B84B A1E5 2A7E 249B C034
7FE4 0DB5 61EC C645 B2F1 C847 ABB4 8F0D C3C2 4BDE