Mobile Platform Security Models

*Original slides by Prof. John Mitchell

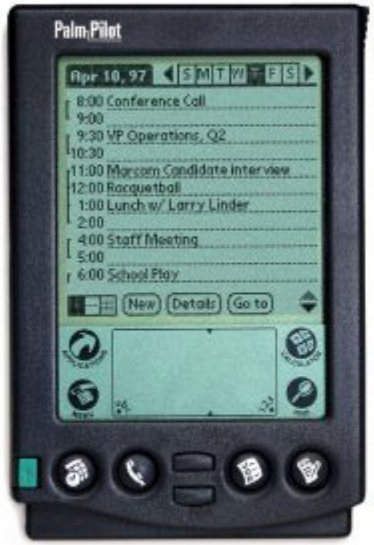
Outline

- Introduction: platforms and attacks
- Apple iOS security model
- Android security model
- Windows 7, 8 Mobile security model

Announcement: See web site for second homework, third project

Change takes time

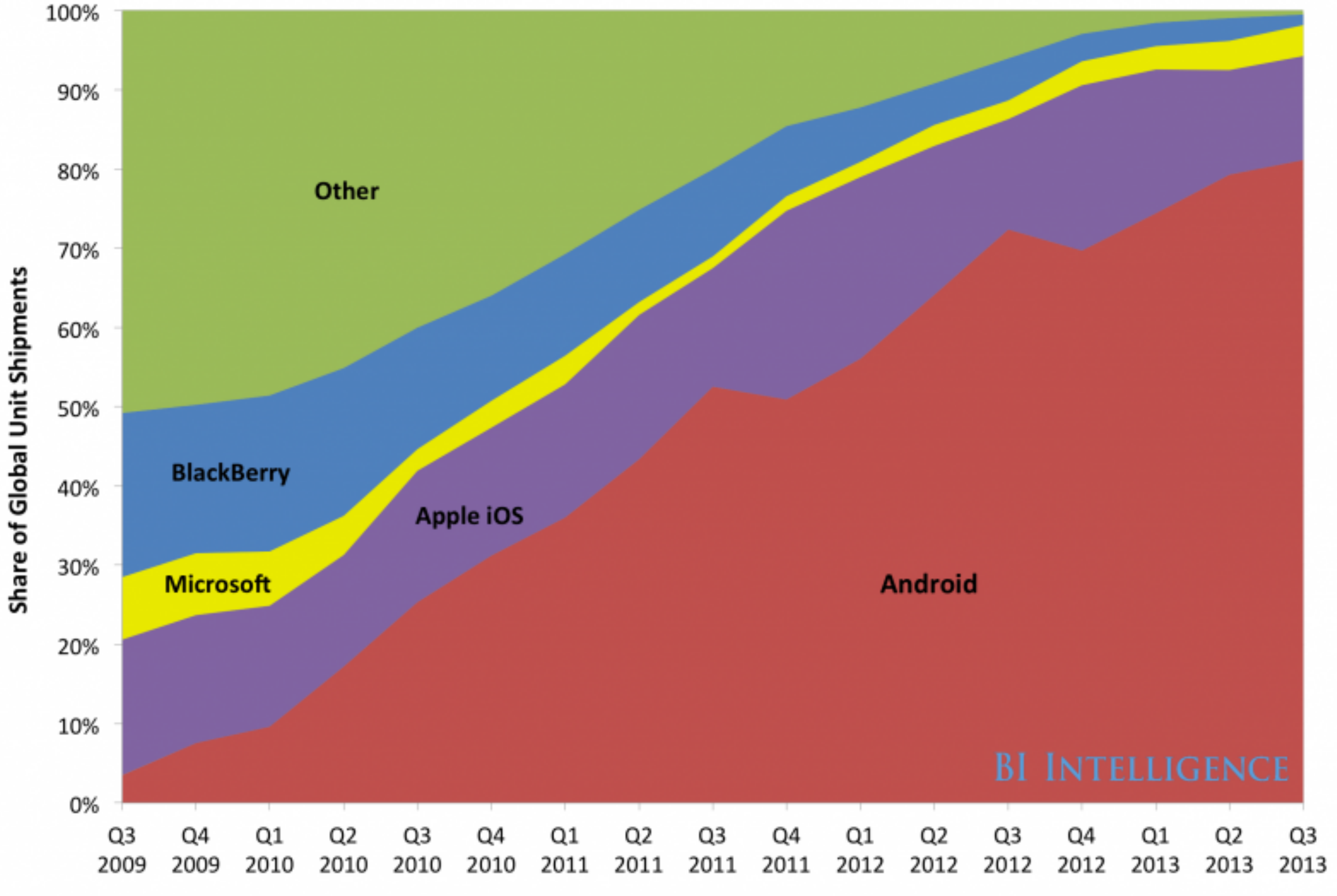
Apple Newton, 1987



Palm Pilot, 1997

iPhone, 2007

Global smartphone market share



Source: IDC, Strategy Analytics

Zillions of apps

Two attack vectors

- Web browser
- Installed apps

Both increasing in prevalence and sophistication

Mobile malware attacks

- Unique to phones:
  - Premium SMS messages
  - Identify location
  - Record phone calls
  - Log SMS
- Similar to desktop/PCs:
  - Connects to botmasters
  - Steal data
  - Phishing
  - Malvertising

Kaspersky: Aug 2013 – Mar 2014

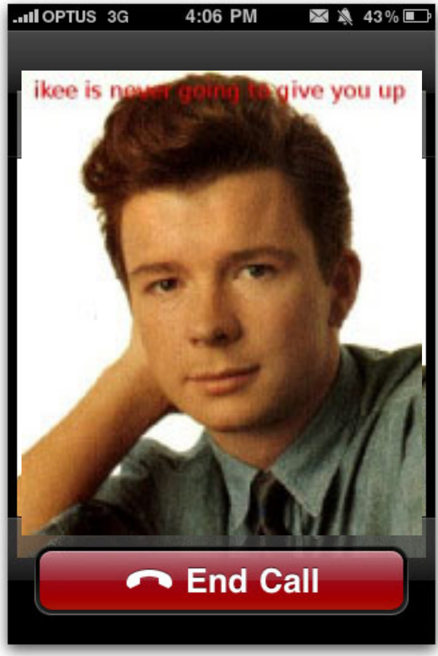
- 3,408,112 malware detections 1,023,202 users.
- 69,000 attacks in Aug 2013 -> 644,000 in Mar 2014
- 35,000 users -> 242,000 users
- 59.06% related to stealing users’ money
- Russia, India, Kazakhstan, Vietnam, Ukraine and Germany have largest numbers of reported attacks
- Trojans sending SMS were 57.08% of all detections

Typical scenario

- Cybercriminals create an affiliate website and invite Internet users to become their accomplices
- A unique modification of the malware and a landing page for download is created for each accomplice
- Participants of the affiliate program trick Android users into installing malicious application
- Infected device sends SMS messages to premium numbers, making money for the cybercriminals
- Part of money is paid to the affiliate partners

http://media.kaspersky.com/pdf/Kaspersky-Lab-KSN-Report-mobile-cyberthreats-web.pdf

Mobile malware examples



ikee is never going to give you up

- DroidDream (Android)
  - Over 58 apps uploaded to Google app market
  - Conducts data theft; send credentials to attacker
- Ikee (iOS)
  - Worm capabilities (targeted default ssh pwd)
  - Worked only on jailbroken phones with ssh installed
- Zitmo (Symbian,BlackBerry,Windows,Android)
  - Propagates via SMS; claims to install a “security certificate”
  - Captures info from SMS; aimed at defeating 2-factor auth
  - Works with Zeus botnet; timed with user PC infection

Comparison between platforms

- Operating system (recall security features from lecture 5)
  - Unix
  - Windows
- Approval process for applications
  - Market: Vendor controlled/Open
  - App signing: Vendor-issued/self-signed
  - User approval of permission
- Programming language for applications
  - Managed execution: Java, .Net
  - Native execution: Objective C

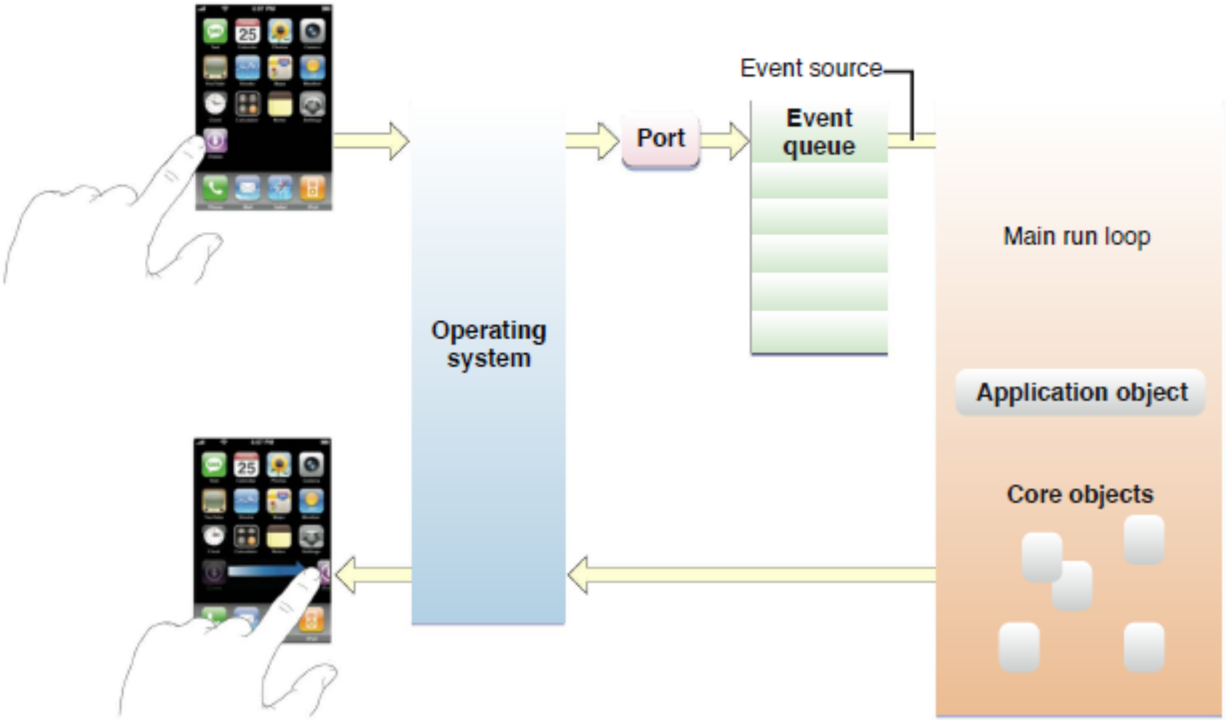
Outline

- Introduction: platforms and attacks
- ➡ Apple iOS security model
- Android security model
- Windows 7 Mobile security model

Apple iOS

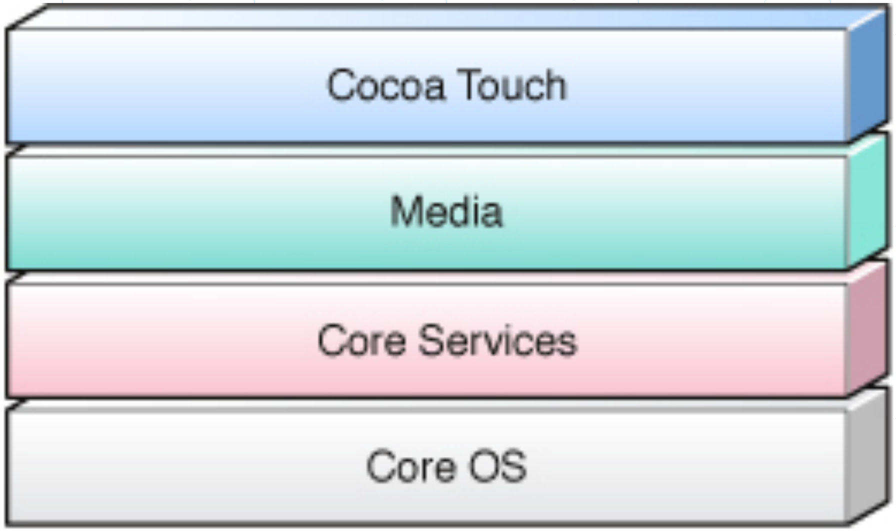
From: iOS App Programming Guide

iOS Application Development



- Apps developed in Objective-C using Apple SDK
- Event-handling model based on touch events
- Foundation and UIKit frameworks provide the key services used by all iOS applications

iOS Platform

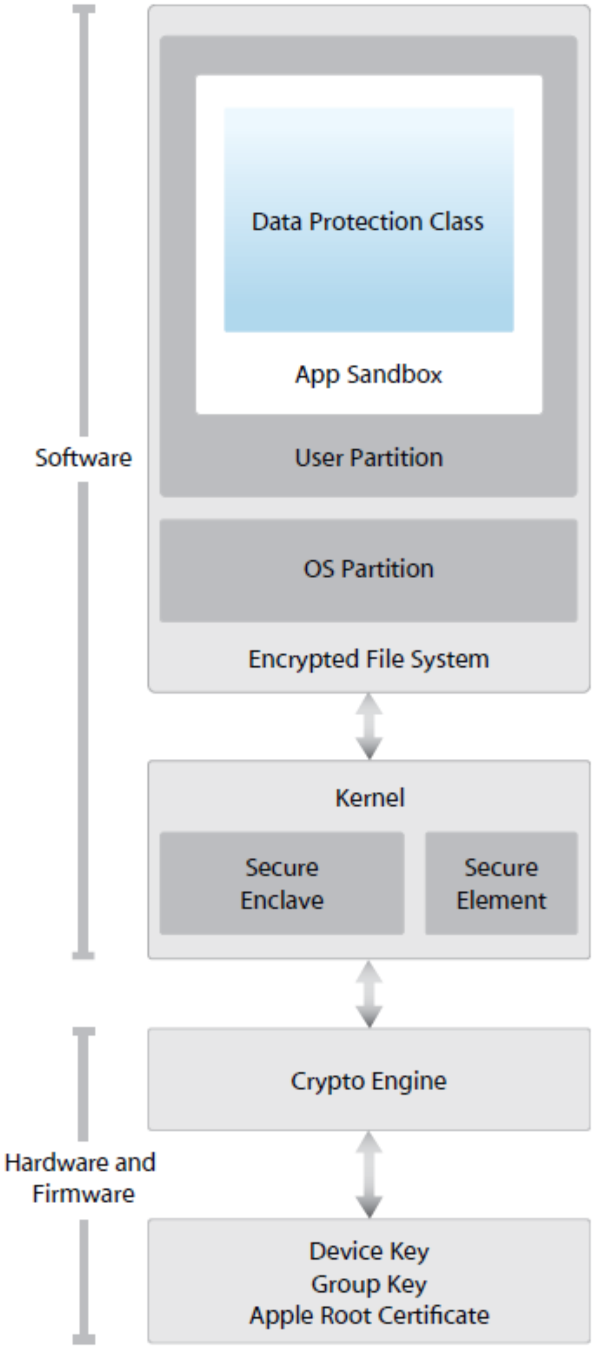


- Cocoa Touch: Foundation framework, OO support for collections, file management, network operations; UIKit
- Media layer: supports 2D and 3D drawing, audio, video
- Core OS and Core Services: APIs for files, network, … includes SQLite, POSIX threads, UNIX sockets
- Kernel: based on Mach kernel like Mac OS X

Implemented in C and Objective-C

Apple iOS Security

- Device security
  - Prevent unauthorized use of device
- Data security
  - Protect data at rest; device may be lost or stolen
- Network security
  - Networking protocols and encryption of data in transmission
- App security
  - Secure platform foundation

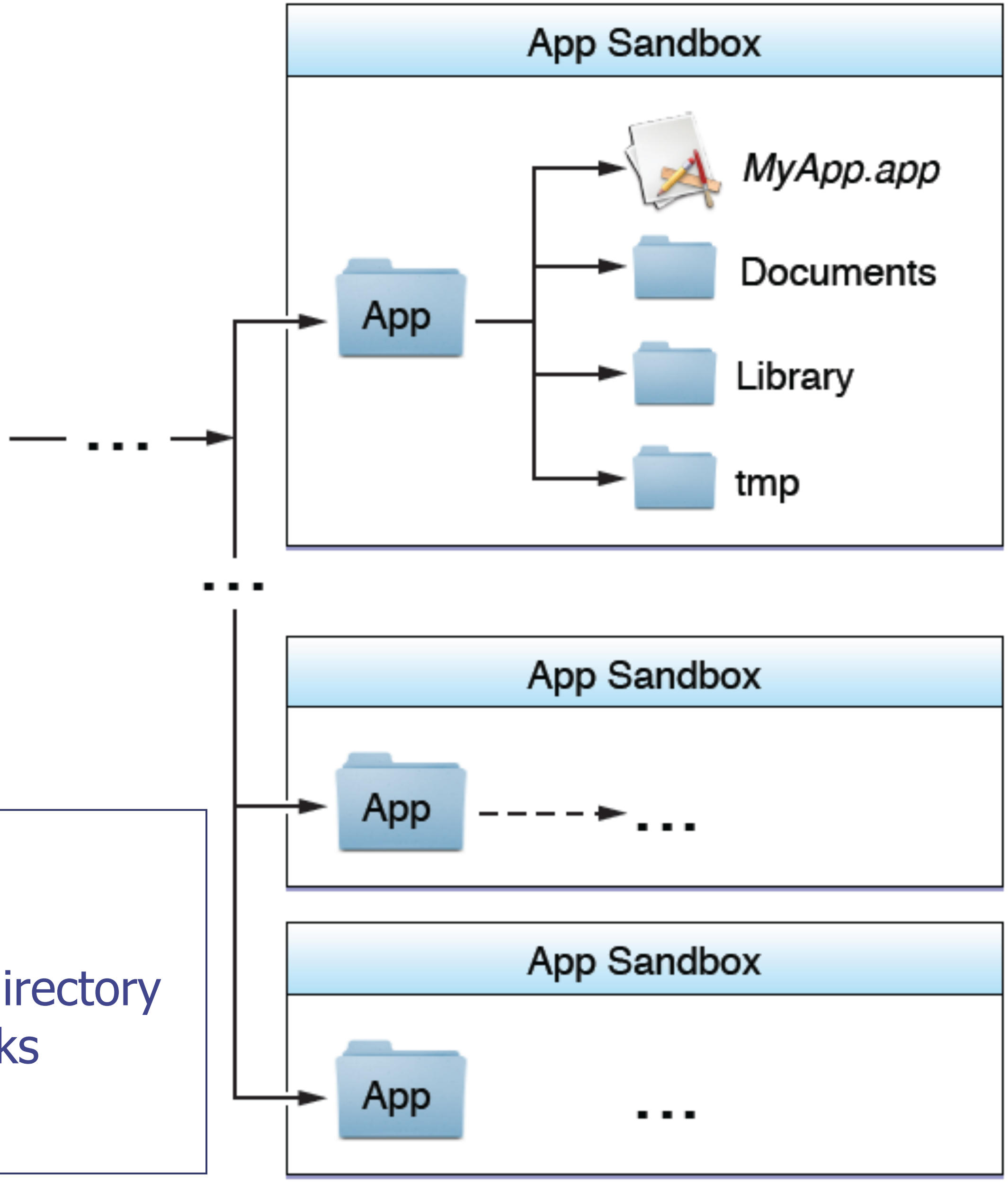


https://www.apple.com/business/docs/iOS_Security_Guide.pdf

App Security

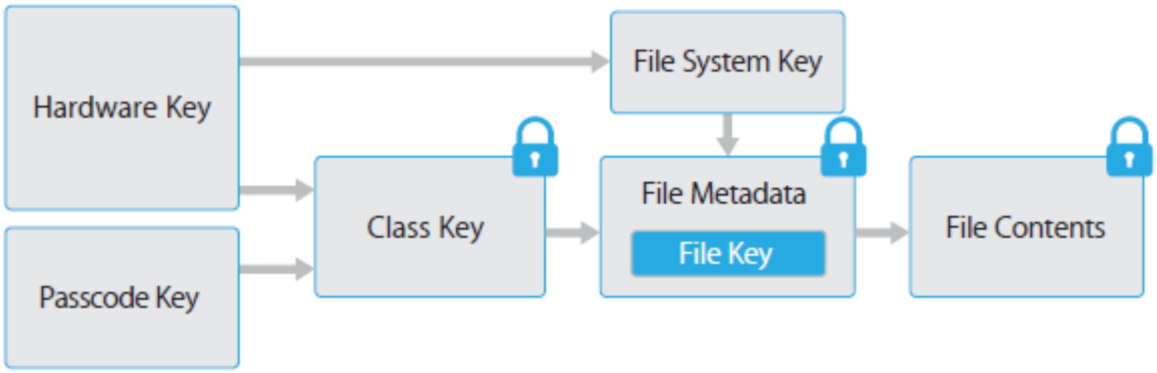
- Runtime protection
  - System resources, kernel shielded from user apps
  - App “sandbox” prevents access to other app’s data
  - Inter-app communication only through iOS APIs
  - Code generation prevented
- Mandatory code signing
  - All apps must be signed using Apple-issued certificate
- Application data protection
  - Apps can leverage built-in hardware encryption

iOS Sandbox



- Limit app’s access to files, preferences, network, other resources
- Each app has own sandbox directory
- Limits consequences of attacks
- Same privileges for each app

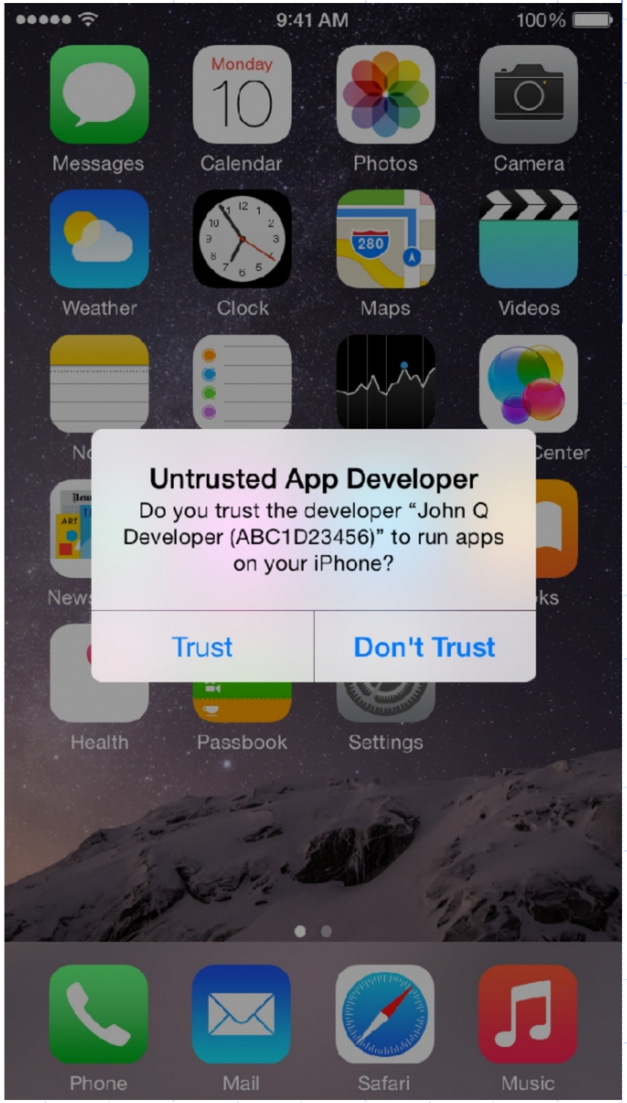
File encryption



- The content of a file is encrypted with a per-file key, which is wrapped with a class key and stored in a file's metadata, which is in turn encrypted with the file system key.
  - When a file is opened, its metadata is decrypted with the file system key, revealing the wrapped per-file key and a notation on which class protects it
  - The per-file key is unwrapped with the class key, then supplied to the hardware AES engine, decrypting the file as it is read from flash memory
- The metadata of all files is encrypted with a random key. Since it's stored on the device, used only for quick erased on demand.

“Masque Attack”

- iOS app installed using enterprise/ad-hoc provisioning could replace genuine app installed through the App Store, if both apps have same bundle identifier
- This vulnerability existed because iOS didn't enforce matching certificates for apps with the same bundle identifier



Untrusted App Developer

Do you trust the developer “John Q Developer (ABC1D23456)” to run apps on your iPhone?

Comparison

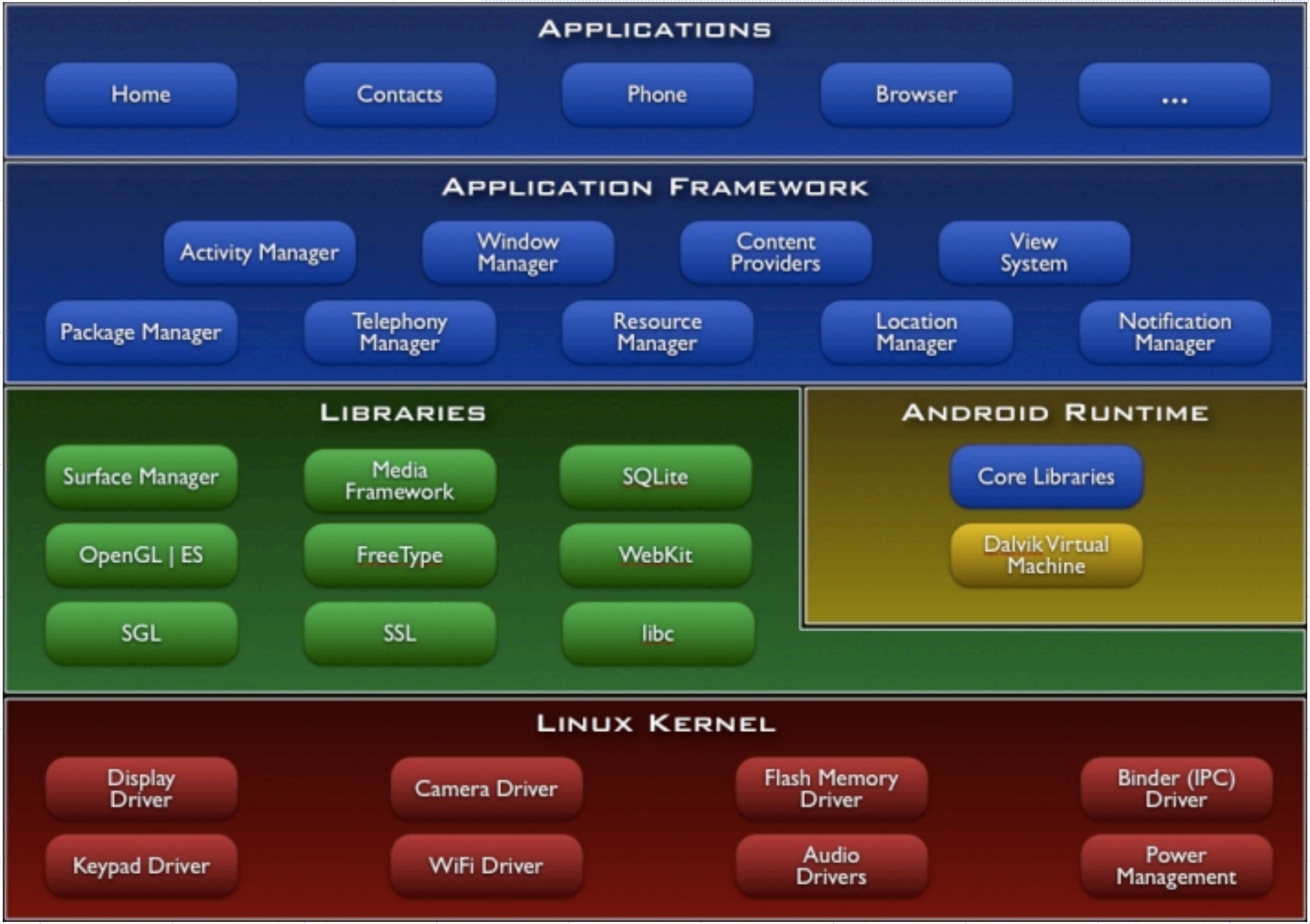
	iOS	Android	Windows
Unix	x		
Windows			
Open market			
Closed market	x		
Vendor signed	x		
Self-signed			
User approval of permissions			
Managed code			
Native code	x		

Outline

- Introduction: platforms and attacks
- Apple iOS security model
- ⇨ Android security model
- Windows 7, 8 Mobile security model

Android

- Platform outline:
  - Linux kernel, browser, SQL-lite database
  - Software for secure network communication
    - Open SSL, Bouncy Castle crypto API and Java library
  - C language infrastructure
  - Java platform for running applications
  - Also: video stuff, Bluetooth, vibrate phone, etc.


Applications
Home
Contacts
Phone
Browser
...
Application Framework
Activity Manager
Window Manager
Content Providers
View System
Package Manager
Telephony Manager
Resource Manager
Location Manager
Notification Manager
Libraries
Surface Manager
Media Framework
SQLite
OpenGL | ES
FreeType
WebKit
SGL
SSL
libc
Android Runtime
Core Libraries
Dalvik Virtual Machine
Linux Kernel
Display Driver
Camera Driver
Flash Memory Driver
Binder (IPC) Driver
Keypad Driver
WiFi Driver
Audio Drivers
Power Management

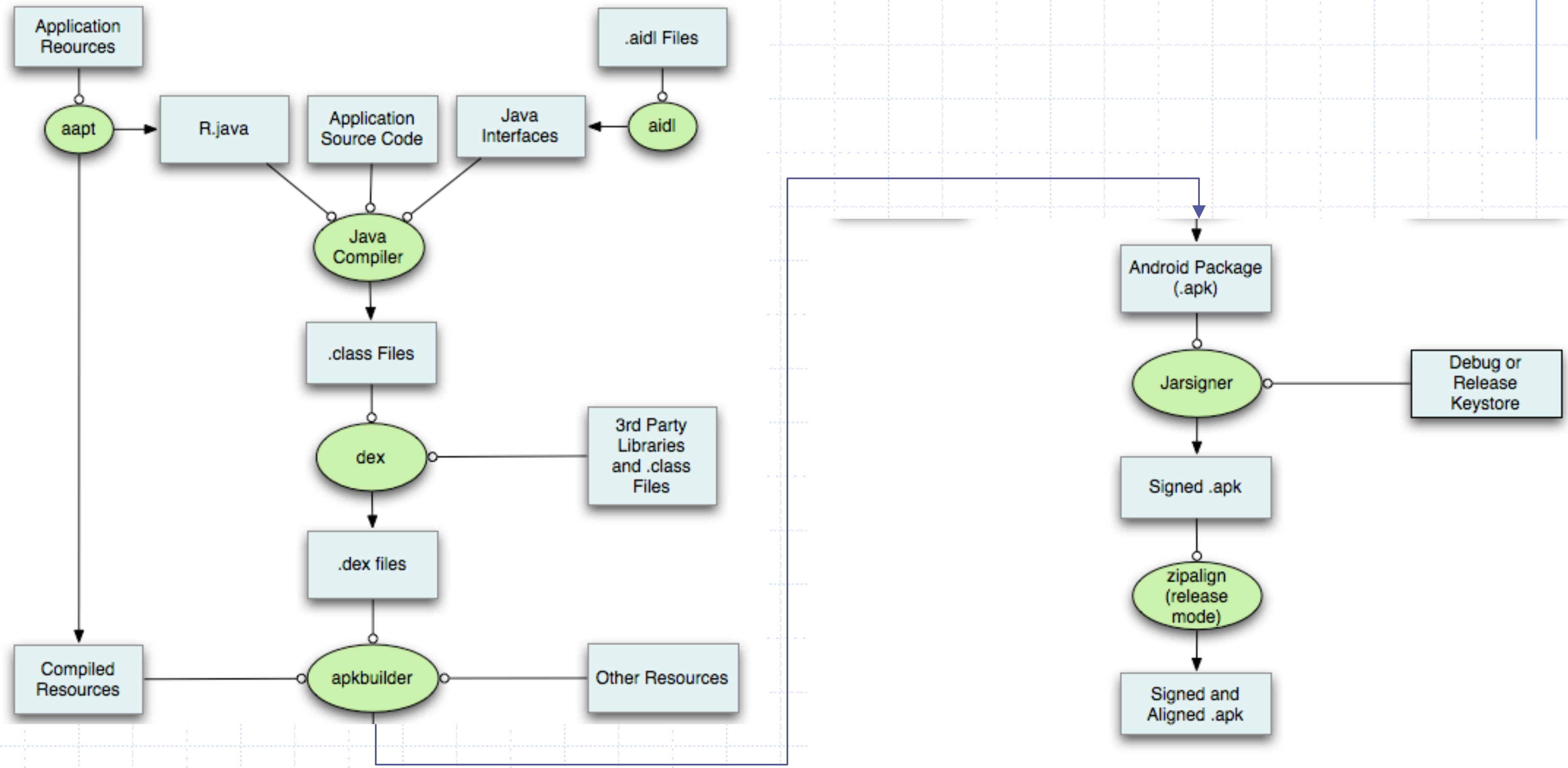
Android market

- Self-signed apps
- App permissions granted on user installation
- Open market
  - Bad applications may show up on market
  - Shifts focus from remote exploit to privilege escalation

Security Features

- Isolation
    - Multi-user Linux operating system
    - Each application normally runs as a different user
- Communication between applications
    - May share same Linux user ID
        - Access files from each other
        - May share same Linux process and Dalvik VM
    - Communicate through application framework
        - “Intents,” based on Binder, discussed in a few slides
- Battery life
    - Developers must conserve power
    - Applications store state so they can be stopped (to save power) and restarted – helps with DoS

Application development process



Application development concepts

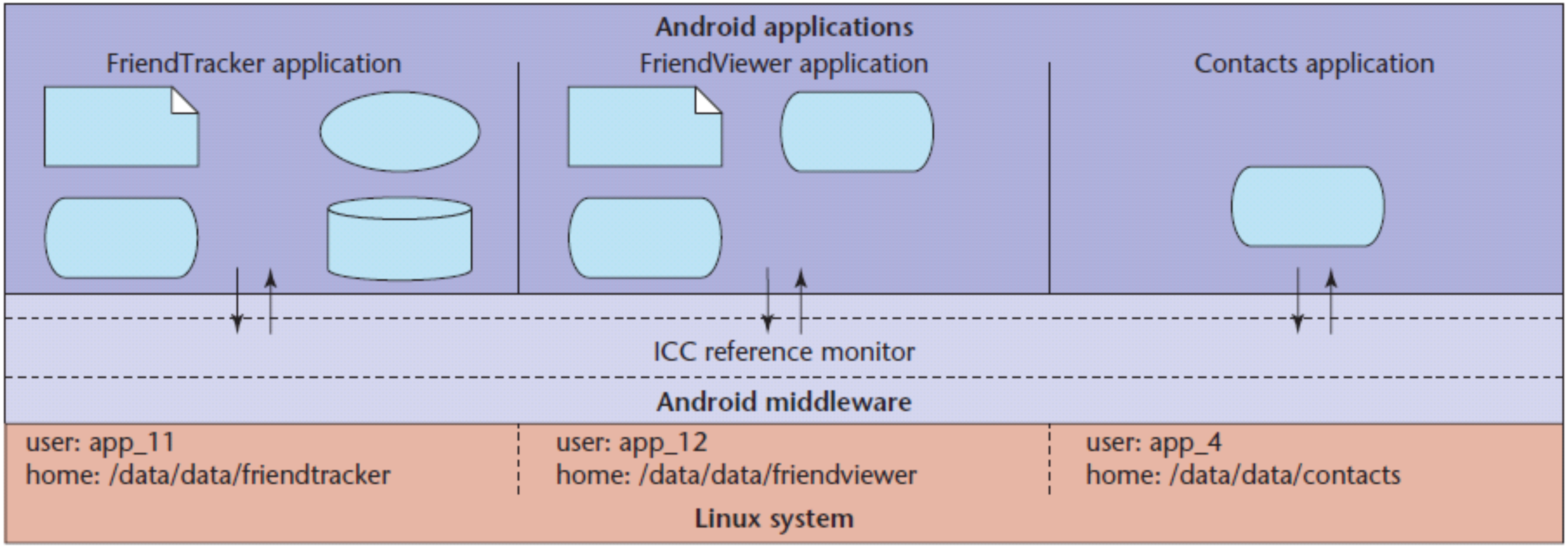
- Activity – one-user task
  - Example: scroll through your inbox
  - Email client comprises many activities
- Service – Java daemon that runs in background
  - Example: application that streams an mp3 in background
- Intents – asynchronous messaging system
  - Fire an intent to switch from one activity to another
  - Example: email app has inbox, compose activity, viewer activity
    - User click on inbox entry fires an intent to the viewer activity, which then allows user to view that email
- Content provider
  - Store and share data using a relational database interface
- Broadcast receiver
  - "mailboxes" for messages from other applications

Exploit prevention

- 100 libraries + 500 million lines new code
    - Open source -> public review, no obscurity
- Goals
    - Prevent remote attacks, privilege escalation
    - Secure drivers, media codecs, new and custom features
- Overflow prevention
    - ProPolice stack protection
        - First on the ARM architecture
    - Some heap overflow protections
        - Chunk consolidation in DL malloc (from OpenBSD)
- ASLR
    - Avoided in initial release
        - Many pre-linked images for performance
    - Later developed and contributed by Bojinov, Boneh

Application sandbox

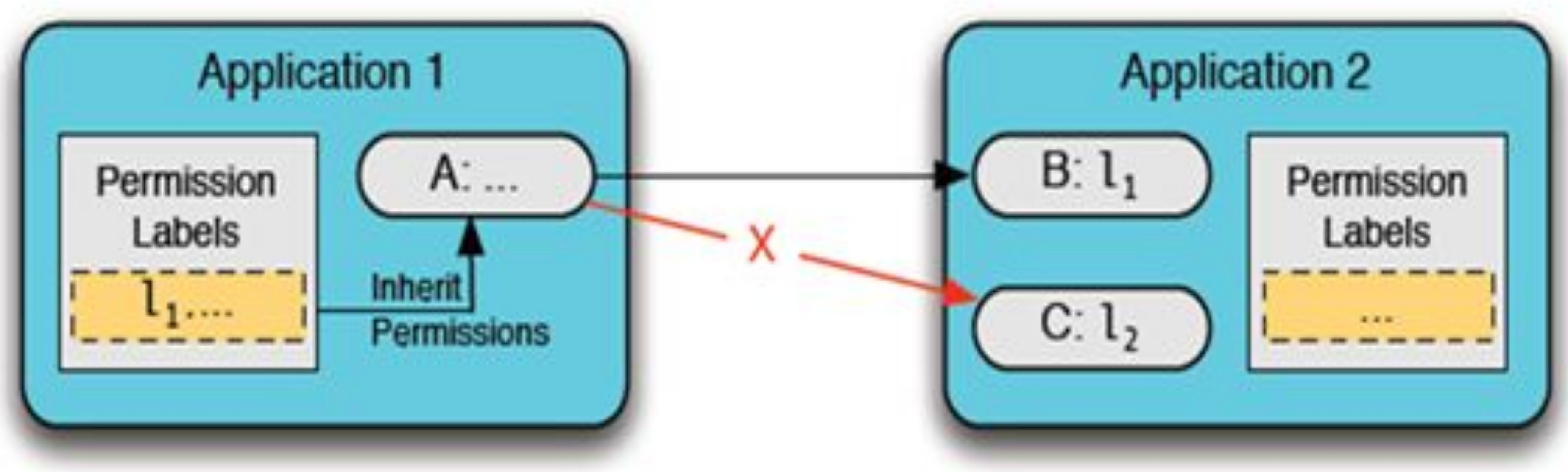
- Application sandbox
  - Each application runs with its UID in its own Dalvik virtual machine
    - Provides CPU protection, memory protection
    - Authenticated communication protection using Unix domain sockets
    - Only ping, zygote (spawn another process) run as root
  - Applications announces permission requirement
    - Create a whitelist model – user grants access
      - But don’t want to ask user often – all questions asked as install time
    - Inter-component communication reference monitor checks permissions



Layers of security

- Each application executes as its own user identity
- Android middleware has reference monitor that mediates the establishment of inter-component communication (ICC)

Source: Penn State group Android security paper



MAC Policy Enforcement in Android. This is how applications access components of other applications via the reference monitor. Component A can access components B and C if permission labels of application 1 are equal or dominate labels of application 2.

Source: Penn State group, Android security tutorial

dlmalloc (Doug Lea)

- Stores meta data in band
- Heap consolidation attack
  - Heap overflow can overwrite pointers to previous and next unconsolidated chunks
  - Overwriting these pointers allows remote code execution
- Change to improve security
  - Check integrity of forward and backward pointers
    - Simply check that back-forward-back = back, f-b-f=f
  - Increases the difficulty of heap overflow

Java Sandbox

- Four complementary mechanisms
  - Class loader
    - Separate namespaces for separate class loaders
    - Associates *protection domain* with each class
  - Verifier and JVM run-time tests
    - NO unchecked casts or other type errors, NO array overflow
    - Preserves private, protected visibility levels
  - Security Manager
    - Called by library functions to decide if request is allowed
    - Uses protection domain associated with code, user policy

Comparison: iOS vs Android

- App approval process
  - Android apps from open app store
  - iOS vendor-controlled store of vetted apps
- Application permissions
  - Android permission based on install-time manifest
  - All iOS apps have same set of “sandbox” privileges
- App programming language
  - Android apps written in Java; no buffer overflow…
  - iOS apps written in Objective-C

Comparison

	iOS	Android	Windows
Unix	x	x	
Windows			
Open market		x	
Closed market	x		
Vendor signed	x		
Self-signed		x	
User approval of permissions		x	
Managed code		x	
Native code	x		

Outline

- Introduction: platforms and attacks
- Apple iOS security model
- Android security model
- ⇨ Windows Phone 7, 8 security model

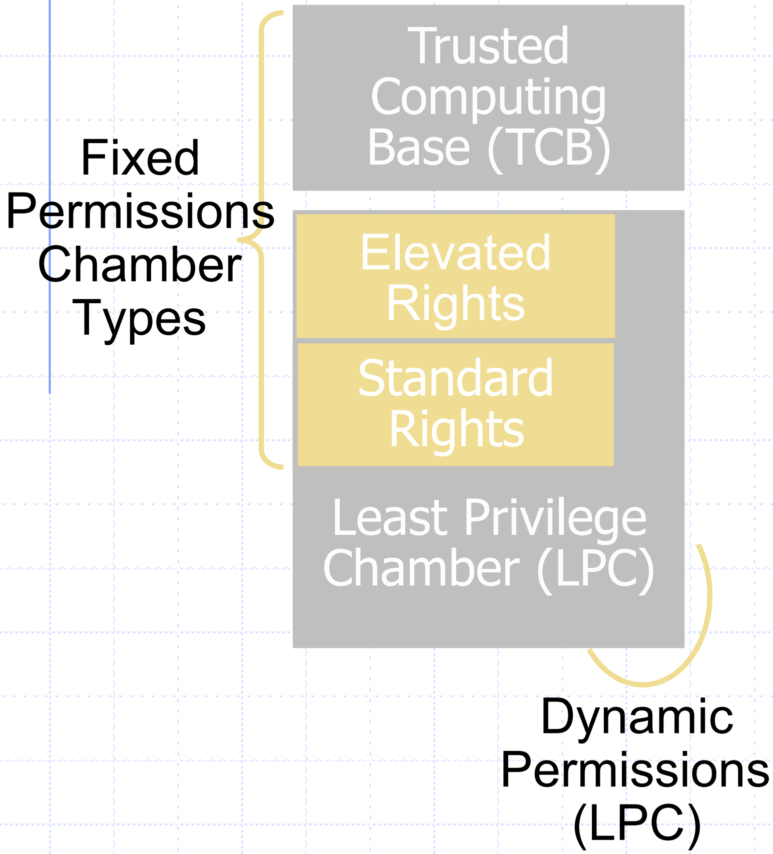
Windows Phone 7, 8 security

- Secure boot
- All binaries are signed
- Device encryption
- Security model with isolation, capabilities

Windows Phone OS 7.0 security model

- Principles of isolation and least privilege
- Each chamber
  - Provides a security and isolation boundary
  - Is defined and implemented using a policy system
- The security policy of a chamber
  - Specifies the OS capabilities that processes in that chamber can access

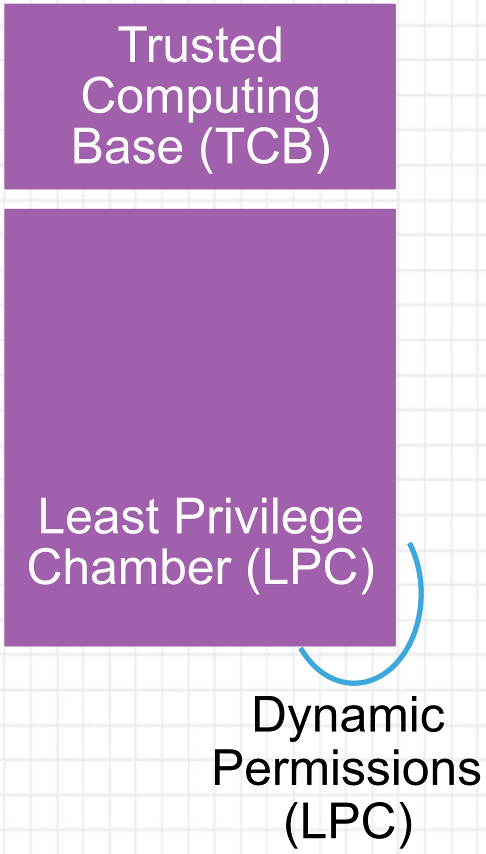
Windows Phone 7 security model



Fixed Permissions Chamber Types
Trusted Computing Base (TCB)
Elevated Rights
Standard Rights
Least Privilege Chamber (LPC)
Dynamic Permissions (LPC)

- Policy system
  - Central repository of rules
  - 3-tuple {Principal, Right, Resource
- Chamber Model
  - Chamber boundary is security boundary
  - Chambers defined using policy rules
  - 4 chamber types, 3 fixed size, one can be expanded with capabilities (LPC)
- Capabilities
  - Expressed in application manifest
  - Disclosed on Marketplace
  - Defines app's security boundary on phone

Windows Phone 8 security model


Trusted Computing Base (TCB)
Least Privilege Chamber (LPC)
Dynamic Permissions (LPC)

Similar to WP7

WP8 chambers are built on the Windows security infrastructure

Services and Application all in chambers

WP8 has a richer capabilities list

Isolation

- Every application runs in own isolated chamber
  - All apps have basic permissions, incl a storage file
  - Cannot access memory or data of other applications, including the keyboard cache.
- No communication channels between applications, except through the cloud
- Non-MS applications distributed via marketplace stopped in background
  - When user switches apps, previous app is shut down
  - Reason: application cannot use critical resources or communicate with Internet–based services while the user is not using the application

Four chamber types

- Three types have fixed permission sets
- Fourth chamber type is capabilities-driven
  - Applications that are designated to run in the fourth chamber type have capability requirements that are honored at installation and at run-time

Overview of four chambers

- Trusted Computing Base (TCB) chamber
  - unrestricted access to most resources
  - can modify policy and enforce the security model.
  - kernel and kernel-mode drivers run in the TCB
  - Minimizing the amount of software that runs in the TCB is essential for minimizing the Windows Phone 7, 8 attack surface

Overview of four chambers

- Elevated Rights Chamber (ERC)
  - Can access all resources except security policy
  - Intended for services and user-mode drivers
- Standard Rights Chamber (SRC)
  - Default for pre-installed applications that do not provide device-wide services
  - Outlook Mobile is an example that runs in the SRC
- Least Privileged Chamber (LPC)
  - Default chamber for all non-Microsoft applications
  - LPCs configured using capabilities (see next slide)

Granting privileges to applications

- Goal: Least Privilege
  - Application gets capabilities needed to perform all its use cases, but no more
- Developers
  - Use the capability detection tool to create the capability list
  - The capability list is included in the application manifest
- Each application discloses its capabilities to the user,
  - Listed on Windows Phone Marketplace.
  - Explicit prompt upon application purchase
  - Disclosure within the application, when the user is about to use the location capability for the first time.

Windows Phone 7 “Capabilities”

- W7 Capability: a resource associated with user privacy, security, cost, or business concerns
- Examples: geographical location information, camera, microphone, networking, and sensors.

Managed code

- Application development model uses of managed code only

.NET Code Access Security

- Default Security Policy is part of the .NET Framework
  - Default permission for code access to protected resources
- Permissions can limit access to system resources.
  - Use EnvironmentPermission class for environment variables access permission.
  - The constructor defines the level of permission (read, write, …)
- Deny and Revert
  - The Deny method of the permission class denies access to the associated resource
  - The RevertDeny method will cause the effects of any previous Deny to be cancelled

Example: code requires permission

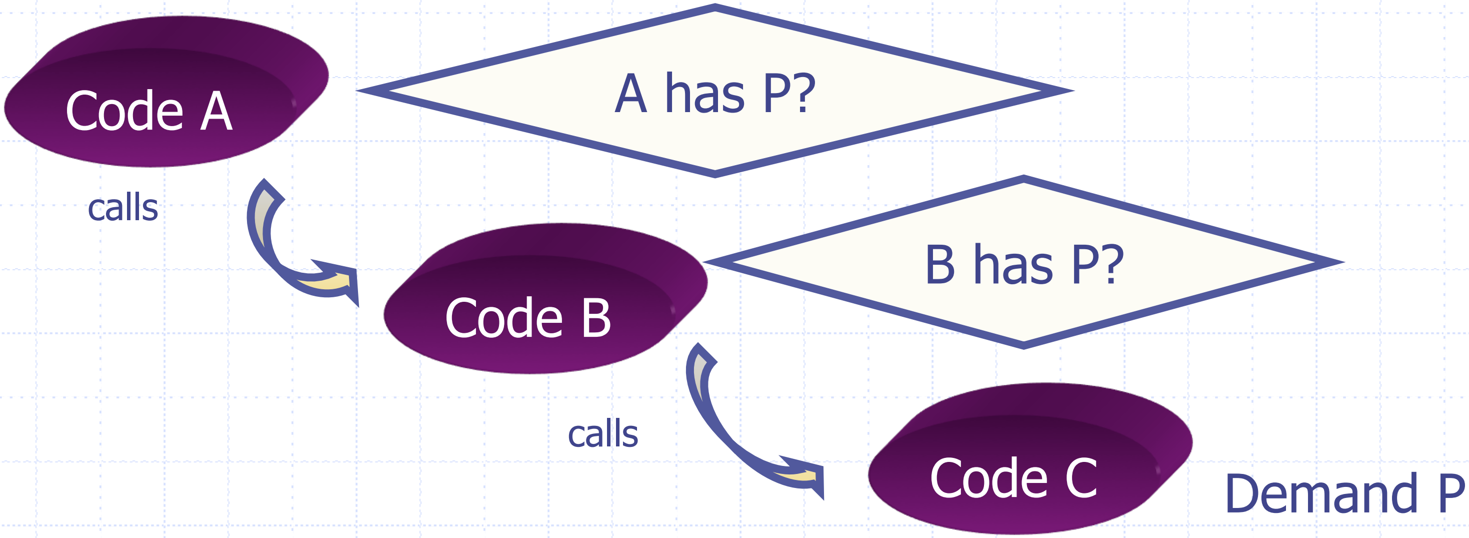
```
class NativeMethods
{
    // This is a call to unmanaged code. Executing this method
    // requires the UnmanagedCode security permission. Without
    // this permission, an attempt to call this method will throw a
    // SecurityException:
    [DllImport("msvcrt.dll")]
    public static extern int puts(string str);
    [DllImport("msvcrt.dll")]
    internal static extern int _flushall();
}
```

Example: Code denies permission not needed

```
[SecurityPermission(SecurityAction.Deny, Flags =
     SecurityPermissionFlag.UnmanagedCode)]
  private static void MethodToDoSomething()
  {   try
      {
          Console.WriteLine(“ … ");
          SomeOtherClass.method();
       }
      catch (SecurityException)
      {
          …
      }
  }
```

.NET Stackwalk

- Demand must be satisfied by all callers
  - Ensures all code in causal chain is authorized
  - Cannot exploit other code with more privilege



Stackwalk: Assert

- The Assert method can be used to limit the scope of the stack walk
  - Processing overhead decreased
  - May inadvertently result in weakened security

Comparison between platforms

- Operating system
  - Unix
  - Windows
- Approval process for applications
  - Market: Vendor controlled/Open
  - App signing: Vendor-issued/self-signed
  - User approval of permissions
- Programming language for applications
  - Managed execution: Java, .Net
  - Native execution: Objective C

Comparison

	iOS	Android	Windows
Unix	x	x	
Windows			x
Open market		x	
Closed market	x		x
Vendor signed	x		
Self-signed		x	x
User approval of permissions		x	7-> 8
Managed code		x	x
Native code	x		

Conclusion

- Introduction: platforms and attacks
- Apple iOS security model
- Android security model
- Windows 7, 8 Mobile security model

Announcement: See web site for second homework, third project