Mobile Device and Platform Security

John Mitchell

Two lectures on mobile security

- Introduction: platforms and trends
- Threat categories
  - Physical, platform malware, malicious apps
- Defense against physical theft
- Malware threats
- System architecture and defenses
  - Apple iOS security features and app security model
  - Android security features and app security model
- Security app development
  - WebView – secure app and web interface dev
  - Device fragmentation

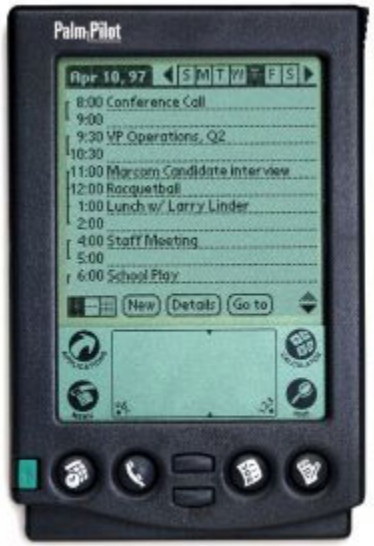
Tues

Thurs

MOBILE COMPUTING

Current devices have long history

Apple Newton, 1987



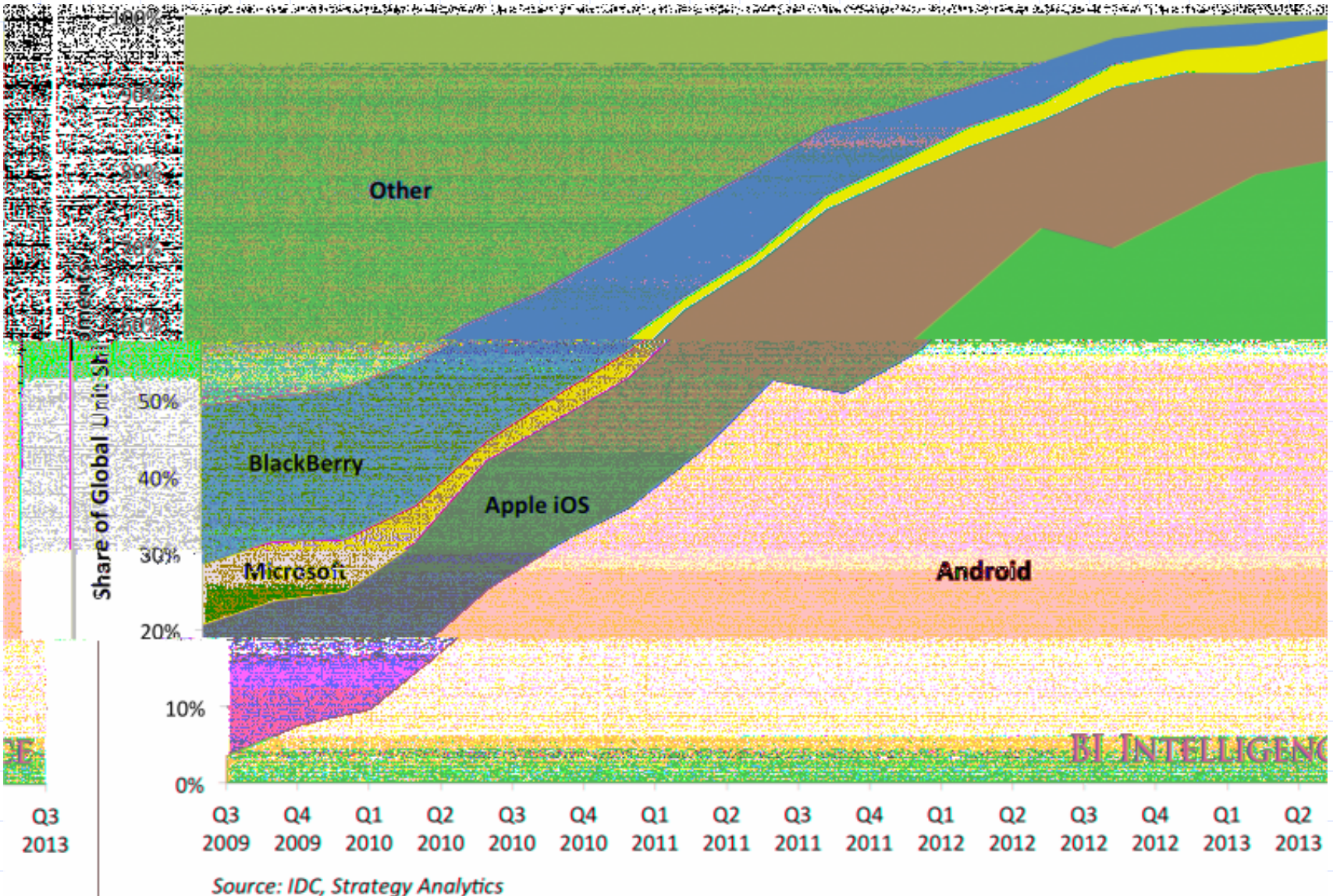
Palm Pilot, 1997

iPhone, 2007

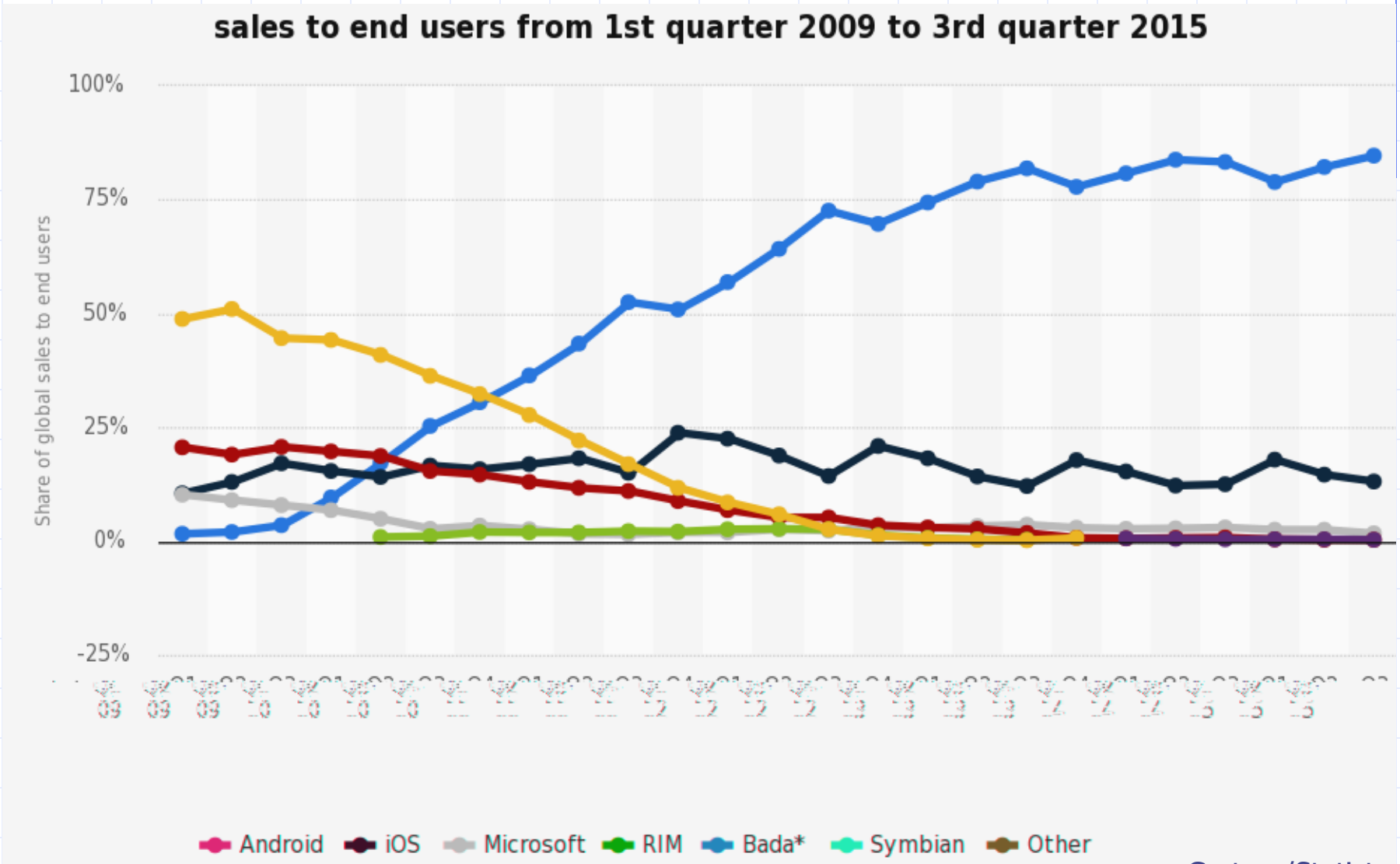
Mobile devices

- Mainframe -> desktop/server -> mobile/cloud
- Trends
  - Increasing reliance on person device
    - Communication, personal data, banking, work
    - Data security, authentication increasingly important
  - From enterprise perspective: BYOD
    - Mobile device management (MDM) to protect enterprise
  - Reliance on cloud: iCloud attack risks, etc
  - Progress from web use to mobile device UI
    - Apps provide custom interface, but limited screen size…
- System designs draw on best ideas of past

Global smartphone market share



Global smartphone market share

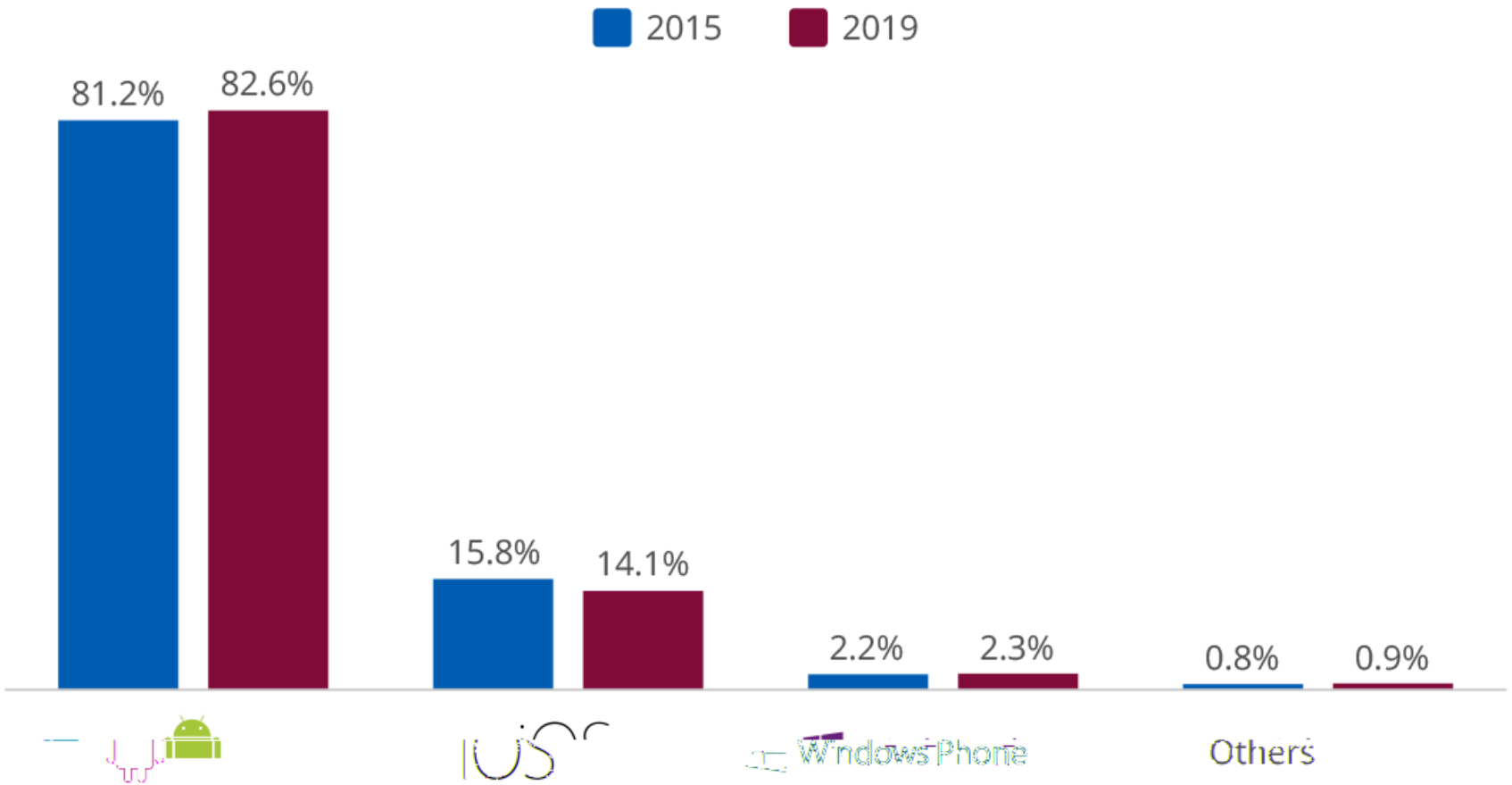


Gartner/Statista

Tech Chart of the Day

Smartphone Platform Market Share

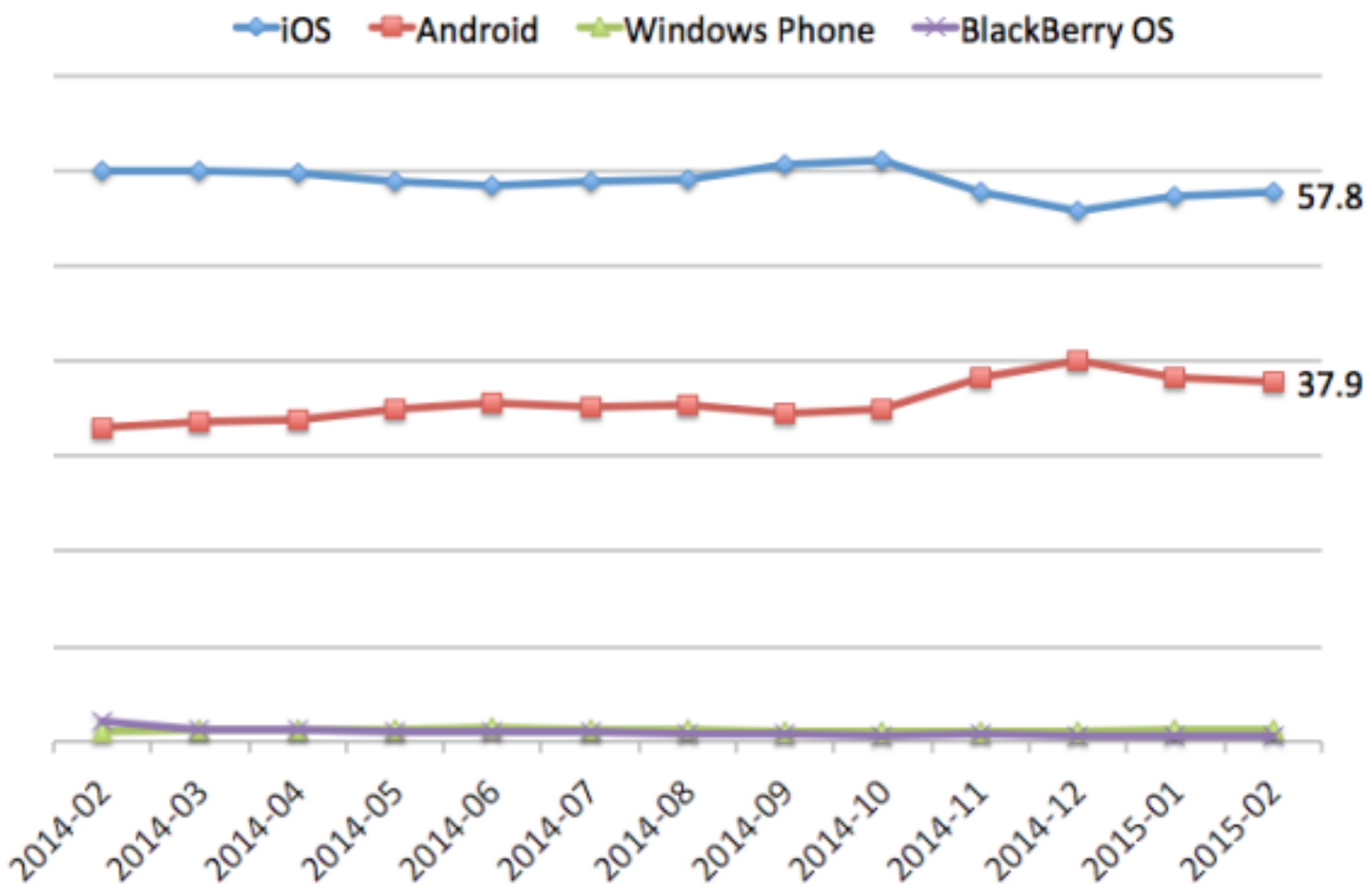
Worldwide smartphone operating system market share (% of new device shipments)*



* Forecast

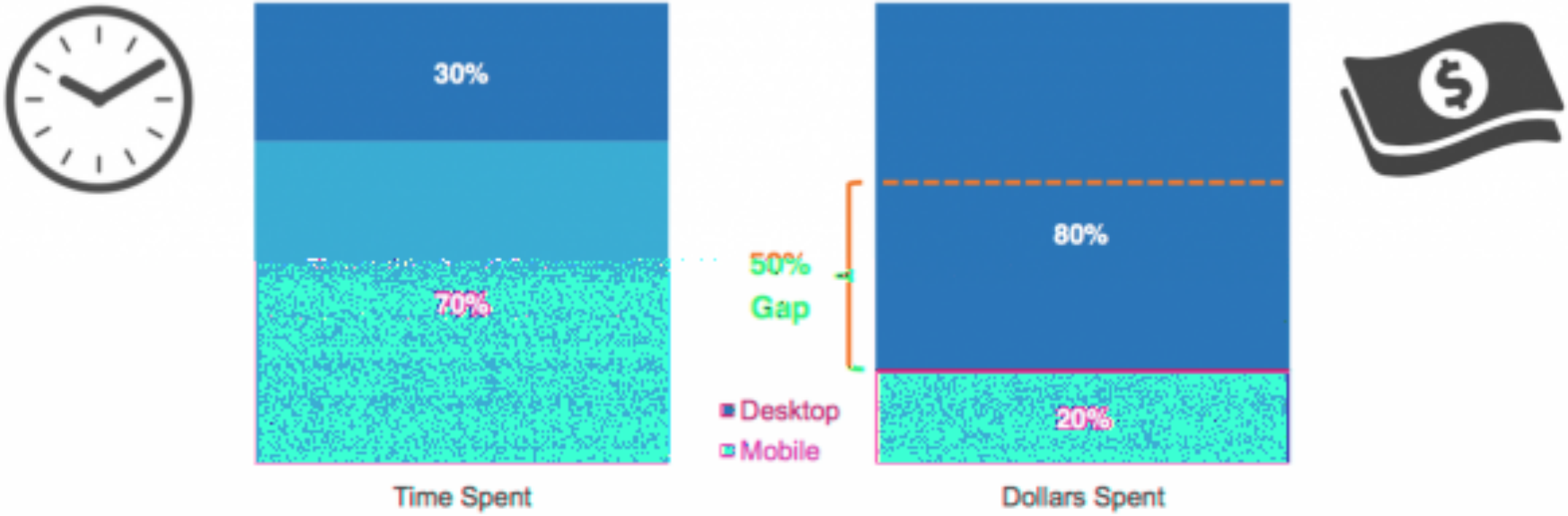
Source: IDC

US Mobile App Traffic



http://www.ironpaper.com/webintel/articles/web-design-statistics-2015/

Comparison with laptop



http://www.ironpaper.com/webintel/articles/web-design-statistics-2017/

Zillions of apps

App Marketplace

- Better protection, isolation than laptop install
- App review before distribution
  - iOS: Apple manual and automated vetting
  - Android
    - Easier to get app placed on market
    - Transparent automated scanning, removal via Bouncer
- App isolation and protection
  - Sandboxing and restricted permission
  - Android
    - Permission model
    - Defense against circumvention

MOBILE THREATS

Mobile platform threat models

- Attacker with physical access
  - Try to unlock phone
  - Exploit vulnerabilities to circumvent locking
- System attacks
  - Exploit vulnerabilities in mobile platform via drive-by web downloads, malformed data, etc.
- App attacks
  - Use malicious app to steal data, misuse system, hijack other apps

OWASP Mobile Top Ten

- M1: Improper Platform Usage
- M2: Insecure Data
- M3: Insecure Communication
- M4: Insecure Authentication
- M5: Insufficient Cryptography
- M6: Insecure Authorization
- M7: Client Code Quality Issues
- M8: Code Tampering
- M9: Reverse Engineering
- M10: Extraneous Functionality

https://www.owasp.org/index.php/Projects/OWASP_Mobile_Security_Project_-2015_Scratchpad

PROTECTION AGAINST PHYSICAL ATTACKER

PROTECTION AGAINST PHYSICAL ATTACKER

Device locking and unlocking

Today: PINs or Patterns

- Need PIN or pattern to unlock device
  - Once unlocked all apps are accessible
- Twist: set a PIN or pattern per app (per photo, video)
  - Protect settings, market, Gmail even if phone unlocked.
  - Examples: App Protector Pro, Seal, Smart lock, …
- Another twist:
  - Front camera takes picture when wrong PIN entered
  - Example: GotYa

Background: brute force pwd attack

- Offline attack
  - Traditionally: steal pwd file, try all pwd
  - Unix pwd file has hashed passwords
  - Cannot reverse hash, but can try dictionary

    hash(pwd, salt) = pwd_file_entry

    ↑ dictionary
- Online attack
  - Can you try all passwords at a web site?
  - What does this mean for phone pin attacks?

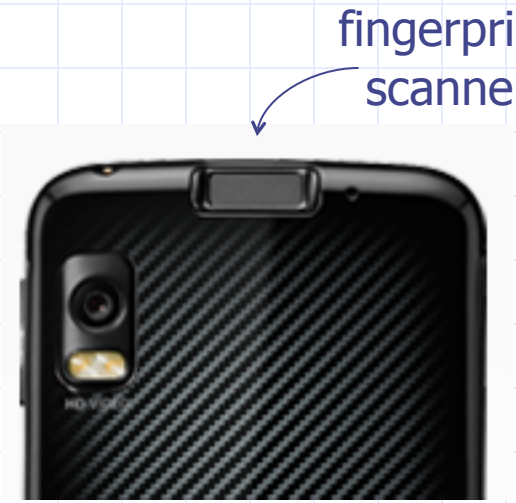
Attacks

- Smudge attacks [Aviv et al., 2010]
  - Entering pattern leaves smudge that can be detected with proper lighting
  - Smudge survives incidental contact with clothing
- Potential defense [Moxie 2011]
  - After entering pattern, require user to swipe across
- Another problem: entropy
  - People choose simple patterns – few strokes
  - At most 1600 patterns with <5 strokes

1
2
3
4
5

Biometric unlocking

- Biometric unlock:
  - Fingerprint (Morotola Atrix 4G)
  - Requires backup PIN
    - ⇒ no more secure than PIN



- Android ICS: Face Unlock
  - Concerns about security

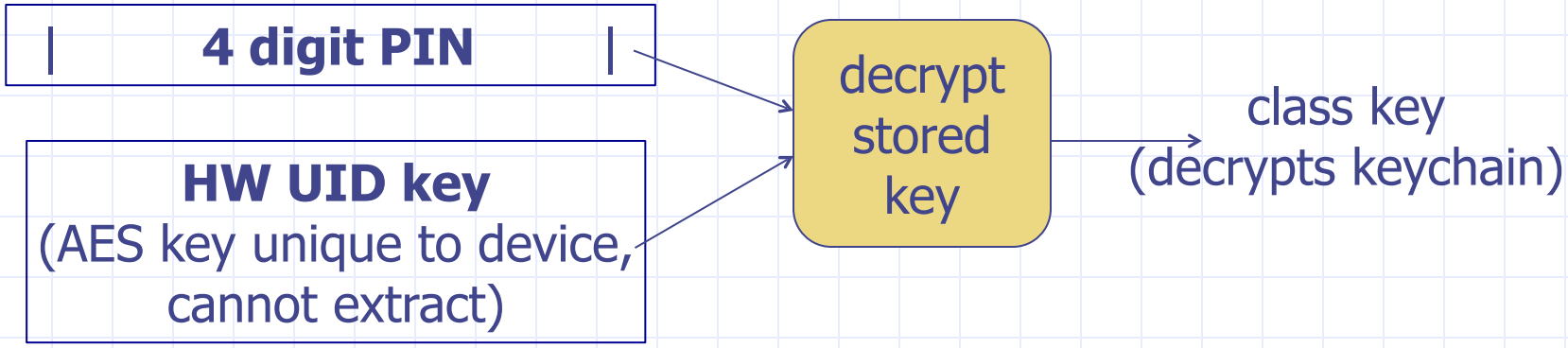
- Standard biometric security concerns:
  - Not secret and cannot be changed.

[Bedrune, Sigwald, 2011]

iOS 4.0: PIN brute force attack

- After device is jail broken, can PIN be extracted?
  - [Needed to read encrypted data partition (later topic)]

- iOS key management (abstract):



- Testing 10,000 PINs
  - for each, derive and test class key ≈ 20 mins on iPhone 4

(code.google.com/p/iphone-dataprotection)

Better Device Unlocking

- A more secure approach to unlocking:
  - Unlock phone using a security token on body
    wrist watch, glasses, clothing
- Requirements
  - Cheap token, should not require charging

Summary: locking and unlocking

- Protect from thief via user authentication
  - Commonly: pin, swipe, etc.
  - Future: Biometric? Token on body?
  - Can phone destroy itself if too many tries?
- Physical access can allow
  - Thief to jailbreak and crack password/pin
  - Subject phone to other attacks
- Next defense: erase phone when stolen

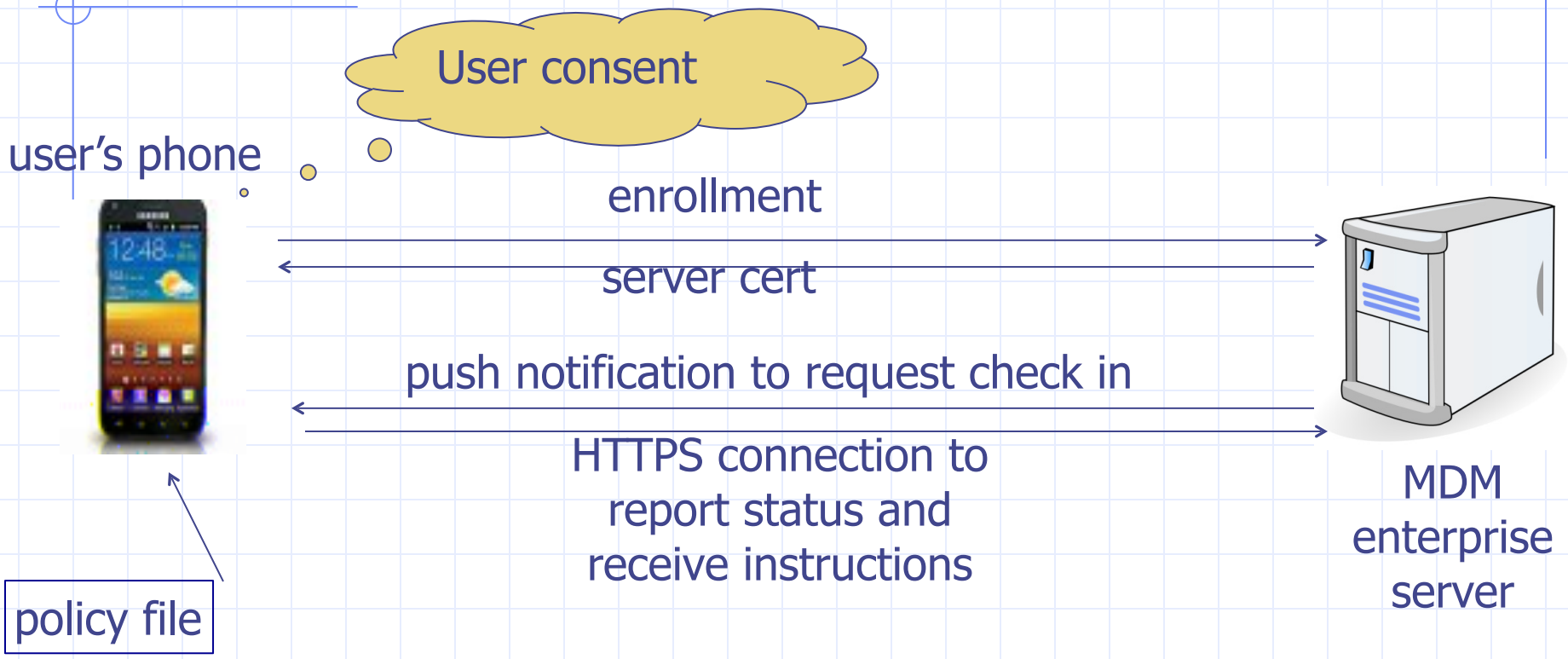
PROTECTION AGAINST PHYSICAL ATTACKER

Mobile device management (MDM)

MDM:Mobile Device Management

- Manage mobile devices across organization
  - Consists of central server and client-side software
- Functions:
  - Diagnostics, repair, and update
  - Backup/restore
  - Policy enforcement (e.g. only allowed apps)
  - **Remote lock and wipe**
  - **GPS tracking**

MDM Sample Deployment

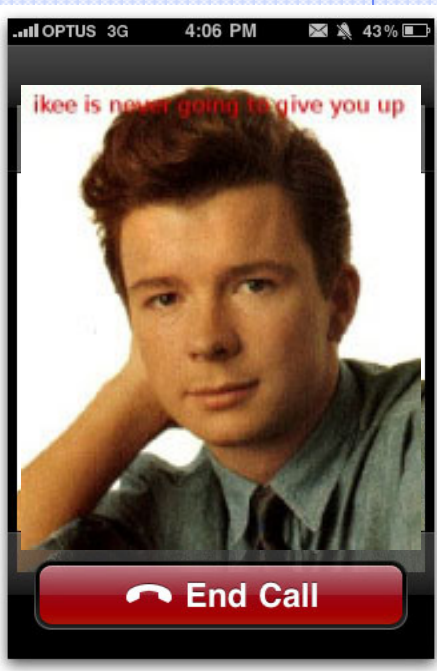


Summary: mobile device mgmt

- Protect stolen phone from thief
  - GPS: where's my phone?
  - Device wipe
- Preventing brute force attacks
  - Phone can "lock" if too many bad pin tries
  - Use MDM to reset to allow user pin
- Backup, backup, backup!
  - Frequent backup makes auto-wipe possible

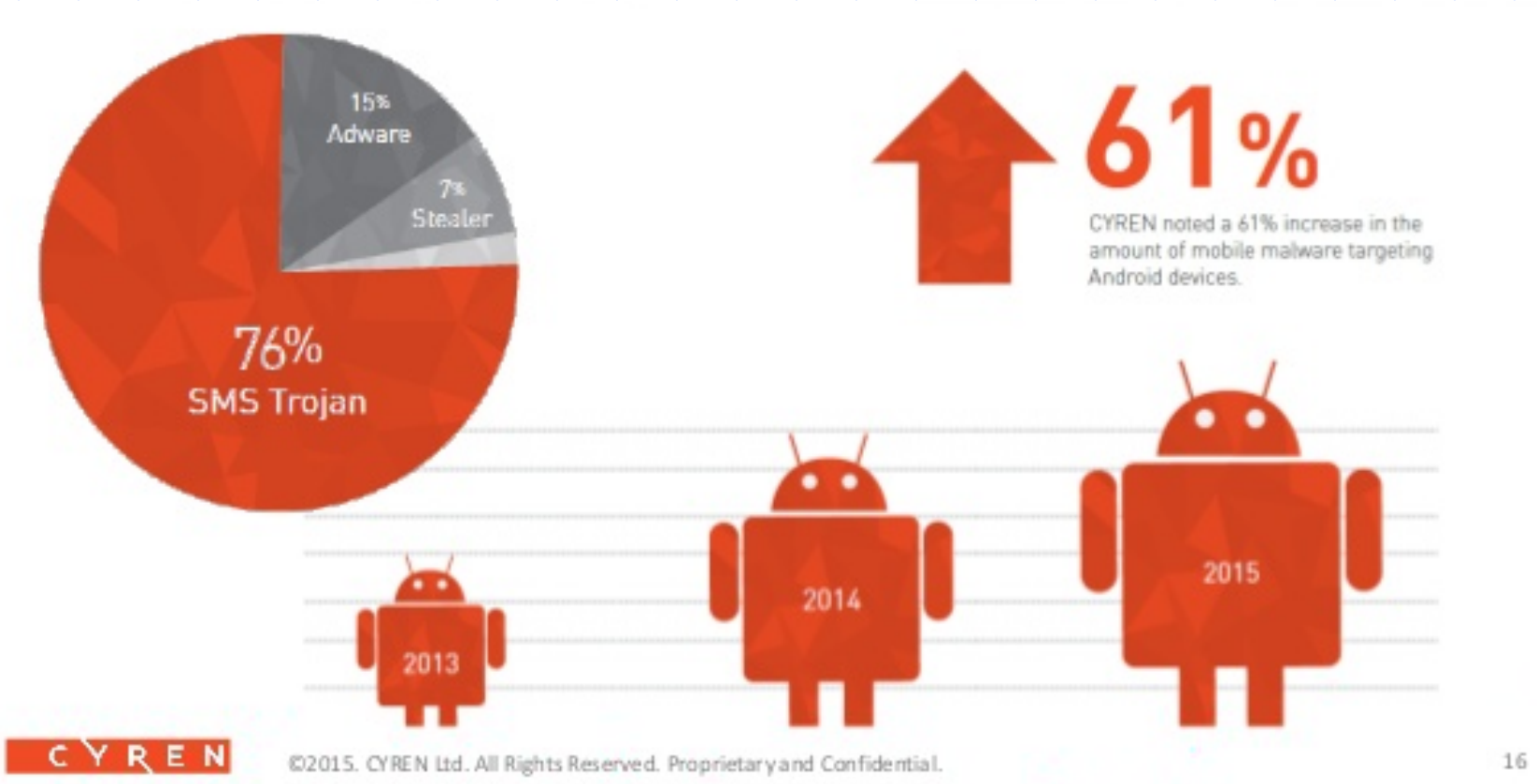
MALWARE ATTACKS

Mobile malware examples

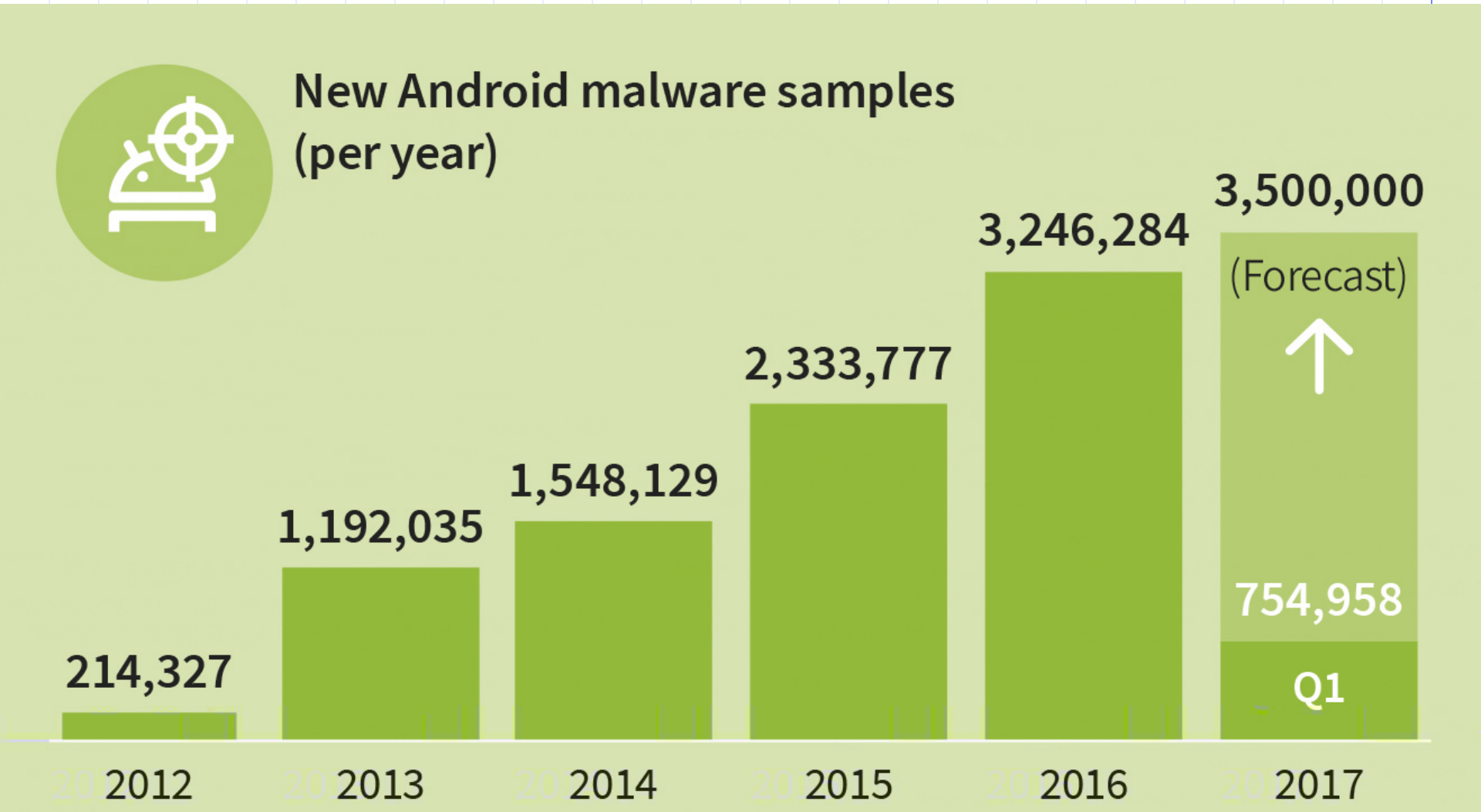


- DroidDream (Android)
  - Over 58 apps uploaded to Google app market
  - Conducts data theft; send credentials to attackers
- Ikee (iOS)
  - Worm capabilities (targeted default ssh pwd)
  - Worked only on jailbroken phones with ssh installed
- Zitmo (Symbian,BlackBerry,Windows,Android)
  - Propagates 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

Android malware 2015



Increasing Android app malware



https://blog.gdatasoftware.com/2017/04/29712-8-400-new-android-malware-samples-every-day

Recent Android Malware

Description

AccuTrack
This application turns an Android smartphone into a GPS tracker.

Ackposts
This Trojan steals contact information from the compromised device and uploads them to a remote server.

Acnetdoor
This Trojan opens a backdoor on the infected device and sends the IP address to a remote server.

Adsms
This is a Trojan which is allowed to send SMS messages. The distribution channel ... is through a SMS message containing the download link.

Airpush/StopSMS
Airpush is a very aggresive Ad-Network.

...

BankBot
This malware tries to steal users' confidential information and money from bank and mobile accounts associated with infected devices.

http://forensics.spreitzenbarth.de/android-malware/

Brief history of iOS attacks

- Find and call (2012)
  - Accesses user's contacts and spams friends
- Jekyll-and-Hyde (2013):
  - Benign app that turns malicious after it passes Apple's review
  - App can post tweets, take photos, send email and SMS, etc.
- Xsser mRat (2014)
  - Steal information from jailbroken iOS devices
- WireLurker (2014)
  - Infects iOS through USB to OSX machines
- Xagent (2015)
  - Spyware. Steals texts, contacts, pictures, …
- AceDeceiver (2016)
  - Infects by exploiting vulnerability in Fairplay (DRM)

Apple pulls popular Instagram client 'InstaAgent' from iOS App Store after malware discovery

By AppleInsider Staff
Tuesday, November 10, 2015, 03:51 pm PT (06:51 pm ET)

A popular Instagram profile analyzer was on Tuesday pulled from the iOS App Store after being outed as malware by a German developer who found the app harvesting usernames and passwords.

ACEDECEIVER: FIRST IOS TROJAN EXPLOITING APPLE DRM DESIGN FLAWS TO INFECT ANY IOS DEVICE

POSTED BY: Claud Xiao on March 16, 2016 5:00 AM

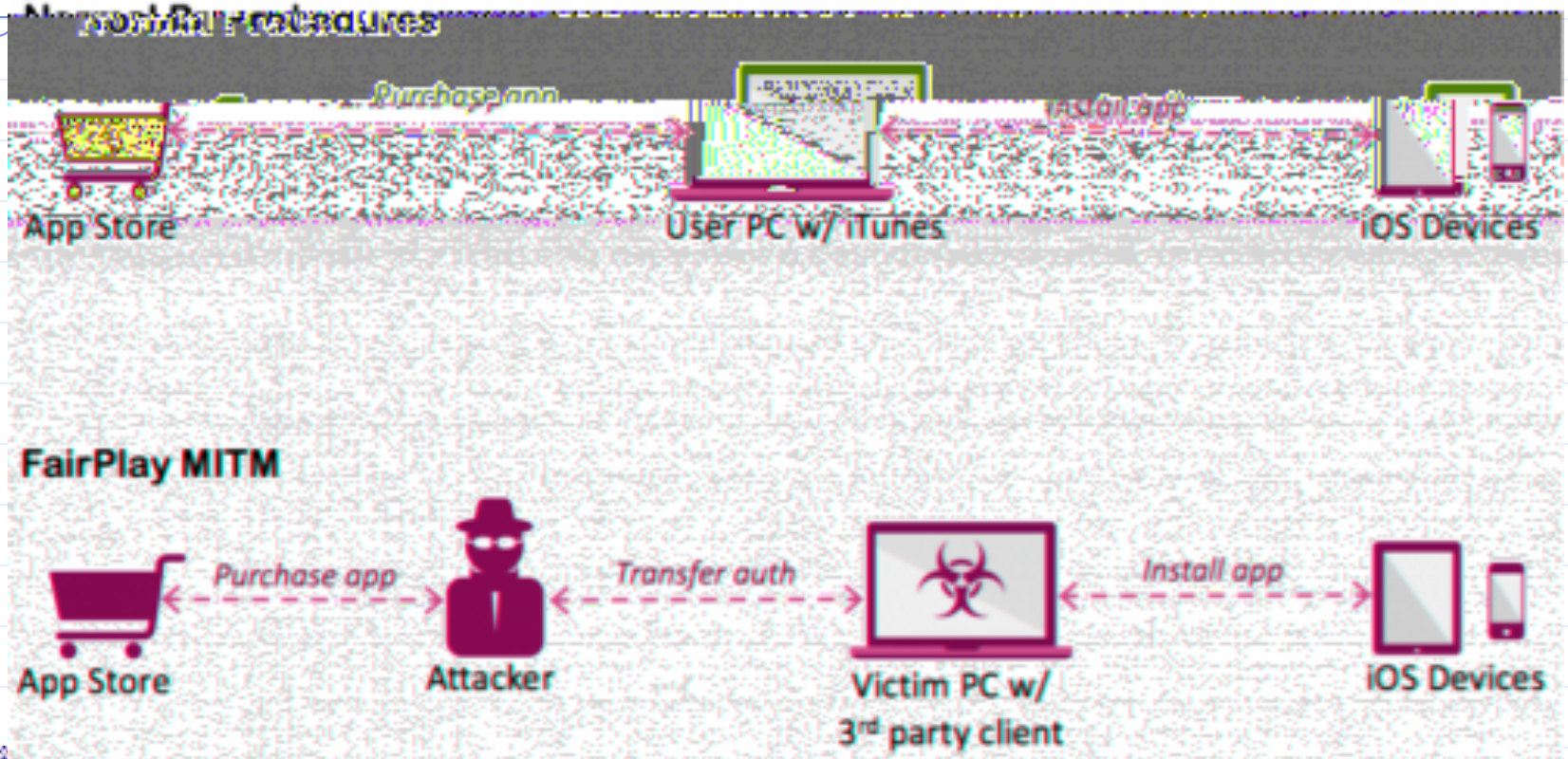
FILED IN: Unit 42
TAGGED: AceDeceiver, FairPlay, OS X, Trojan, ZergHelper

We've discovered a new family of iOS malware that successfully infected non-jailbroken devices we've named "AceDeceiver".

What makes AceDeceiver different from previous iOS malware is that instead of abusing enterprise certificates as some iOS malware has over the past two years, AceDeceiver manages to install itself without any enterprise certificate at all. It does so by exploiting design flaws in Apple's DRM mechanism, and even as Apple has removed AceDeceiver from App Store, it may still spread thanks to a novel attack vector.

AceDeceiver is the first iOS malware we've seen that abuses certain design flaws in Apple's DRM protection mechanism — namely FairPlay — to install malicious apps on iOS devices regardless of whether they are jailbroken. This technique is called "FairPlay Man-In-The-Middle (MITM)" and has been used since 2013 to spread pirated iOS apps, but this is the first time we've seen it used to spread malware. (The FairPlay MITM attack technique was also

Based on FairPlay vulnerability



- Requires malware on user PC, install of malicious app in App Store
- Continues to work after app removed from store
- Silently installs app on phone

IOS PLATFORM

Apple iOS

From: iOS App Programming Guide

Reference

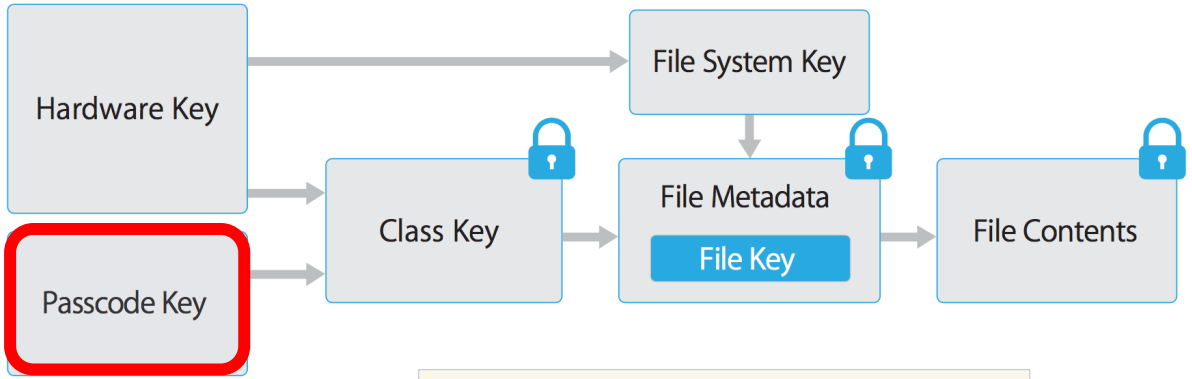
https://www.apple.com/business/docs/iOS_Security_Guide.pdf

Topics

- System Security — Protecting mobile platform (2)
- Encryption and Data Protection
- App Security — App isolation and protection (3)
- Network Security
- Apple Pay
- Internet Services
- Device Controls — User-level security features (1)
- Privacy Controls
- Apple Security Bounty

IOS DEVICE AND PRIVACY CONTROLS

Device unlock

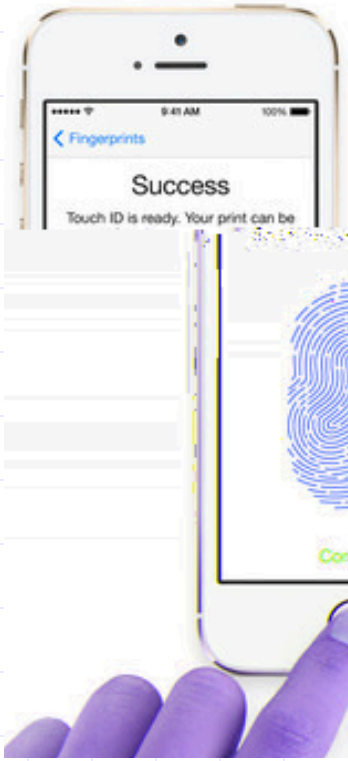


Can attacker try all 6-digit passcodes?

- Passcode key:
  derived by hashing passcode and device ID
- Hashing uses secret UID on secure enclave
  ⇒ deriving passcode key requires the secure enclave
- Secure enclave enforces 80ms delay per evaluation:
  - 5.5 years to try all 6 digits pins
  - 5 failed attempts ⇒ 1min delay, 9 failed attempts ⇒ 1 hour delay
  - >10 failed attempts ⇒ erase phone. Counter on secure enclave.

Unlocking with Touch ID

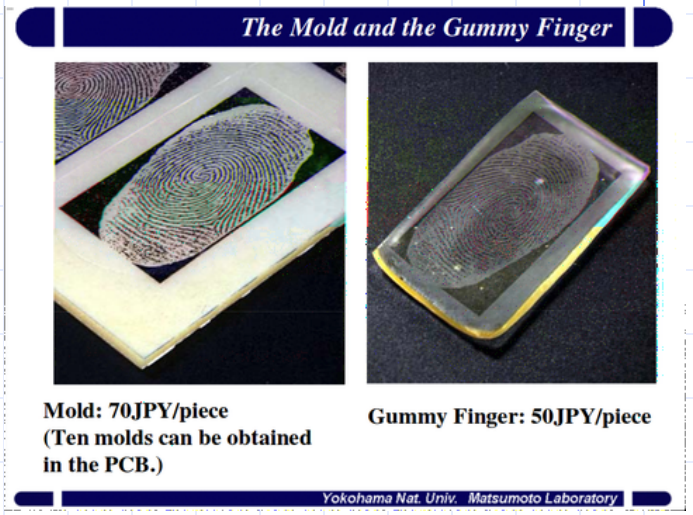
- Passcode can always be used instead
  - Passcode required after: Reboot, or five unsuccessful Touch ID attempts, …
- Other uses (beyond unlock):
  - Enable access to keychain items
  - Apple Pay
  - Can be used by applications



How does it work?

- Touch ID: sends fingerprint image to secure enclave (encrypted)
  - Enclave stores skeleton encrypted with secure enclave key
- With Touch ID off, upon lock, class-key Complete is deleted
- ⇒ no data access when device is locked
- With Touch ID on: class-key is stored encrypted by secure enclave
- Decrypted when authorized fingerprint is recognized
- Deleted upon reboot, 48 hours of inactivity, or five failed attempts

How secure is it?

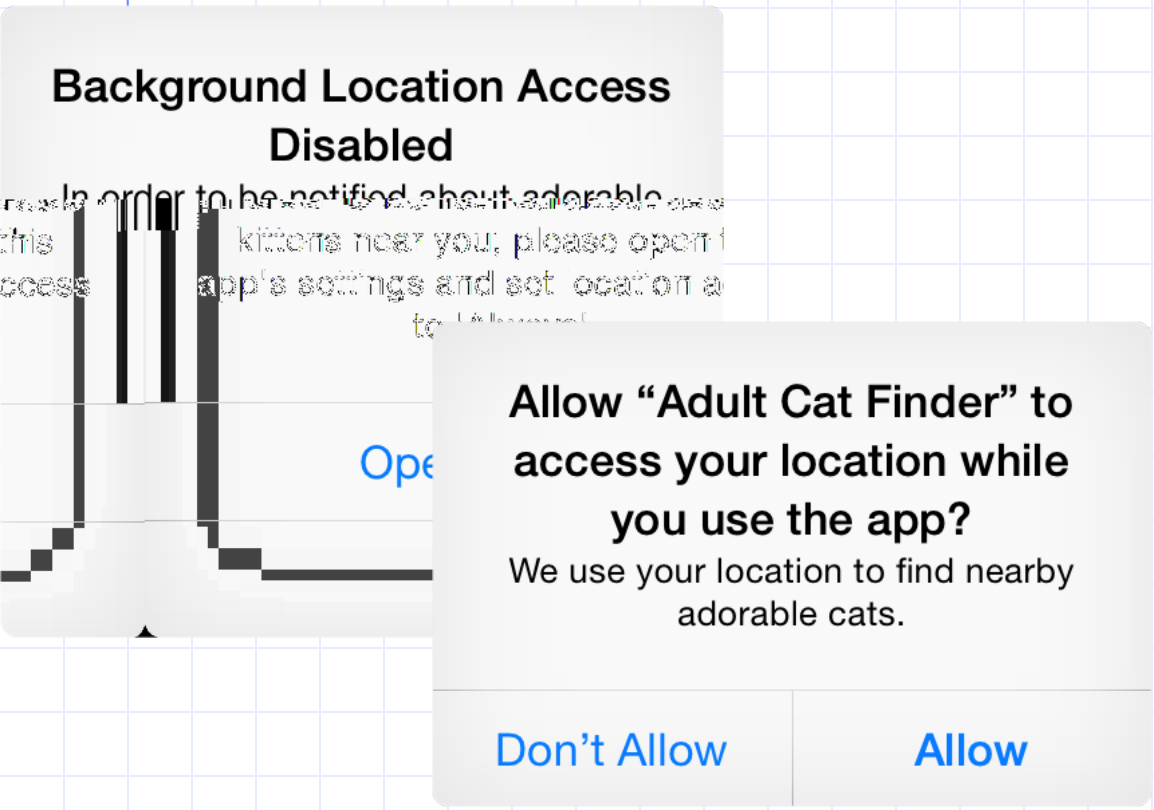


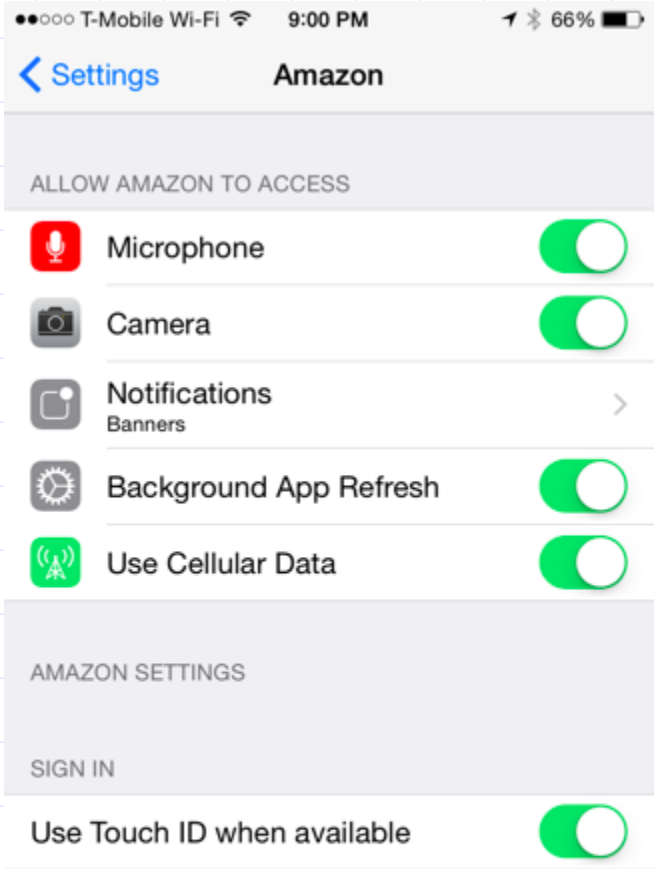
- Easy to build a fake finger
  - Several demos on YouTube
  - About 20 mins of work
  - If you have a fingerprint
- The problem: fingerprints are not secret
  - No way to reset once stolen

- Convenient, but more secure solutions exist:
  - Unlock phone via bluetooth using a wearable device
    ⇒ phone locks as soon as device is out of range
  - Enable support for both a passcode and a fingerprint

iOS Privacy Controls

- User can select which apps access location, microphone, a few other services

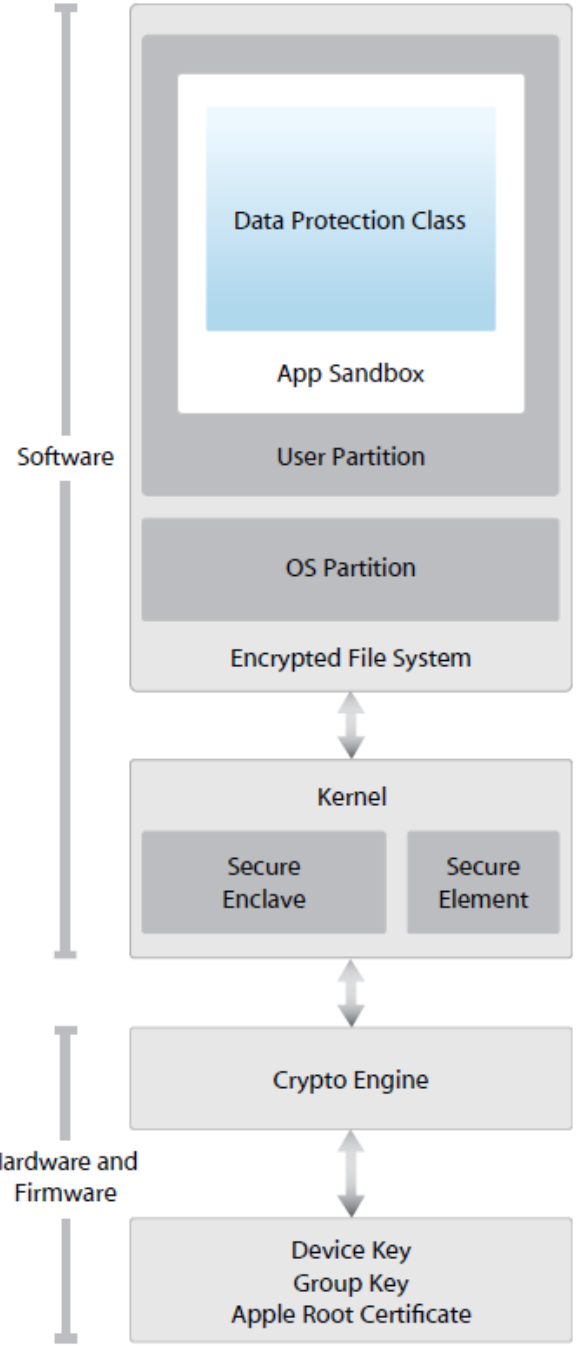




IOS SYSTEM AND DATA SECURITY

Apple iOS Security

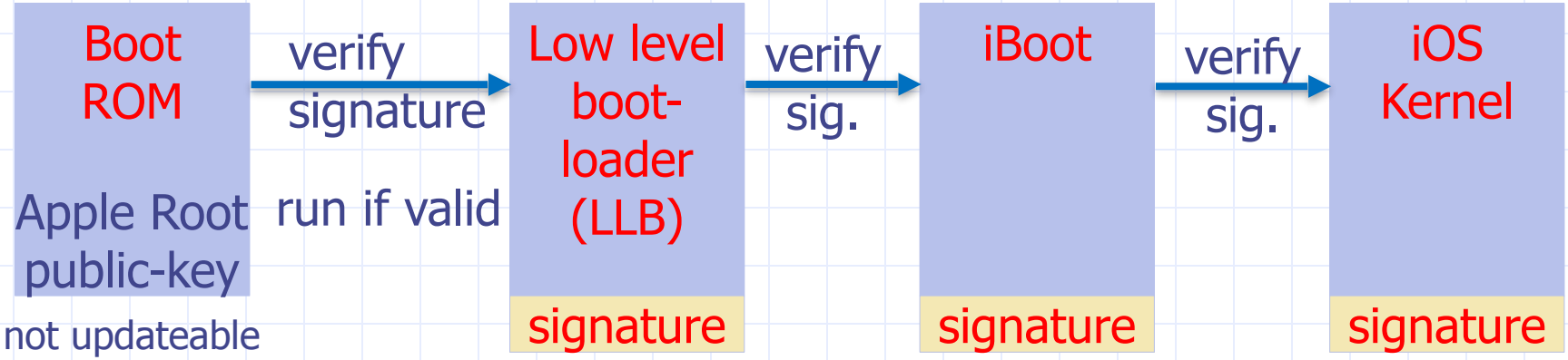
- Device security
  - Prevent unauthorized use of device
- Data security
  - Protect data at rest; device may be lost or stolen
- Network security
  - Networking protocols and encryption of data in transmission
- App security
  - Secure platform foundation



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

Secure boot chain

- Every layer ensures that the next layer is properly signed
- Root of trust: boot ROM, installed during fabrication



Secure boot chain

- Ensures only authorized iOS code can boot
- Jailbreaking works by exploiting bugs in the chain
  - Disables verification down the line
- Note: bugs in the boot ROM are especially damaging
  - Boot ROM cannot be updated

Software update

- All iOS software updates are signed by Apple
  - Signature from Apple's software update server covers:
    hash of update code,
    device unique ID (ECID) and nonce from device

⇒ Apple keeps track of which devices (ECID) updated to what

Why sign nonce and device ID? (harder for Apple to distribute patch)

- Cannot copy update across devices ⇒ Apple can track updates
- Nonce ensures device always gets latest version of update

Jailbreak detection

- Jailbreaking: install apps outside 3rd party sandbox
  - Apps in /Applications (not in sandboxed "mobile" dir)
- Jailbreak prevention
  - App wants to detect if device is jailbroken and not run if so, e.g., banking apps
- Some methods:

_dyld_get_image_name(): check names of loaded dynamic libs

_dyld_get_image_header(): inspect location in memory

- Can be easily bypassed – jailbreak detection is brittle
  - e.g., using Xcon tool (part of Cydia)

App exploit mitigation: XN and ASLR

- XN bit (eXecute Never): [a.k.a NX bit]
  - Mark stack and heap memory pages as non-execute, enforced by CPU
- ASLR (address space layout randomization):
  - At app startup: randomize location of executable, heap, stack
  - At boot time: randomize location of shared libs
- Harder to exploit memory corruption vulns

Data protection: protecting application data

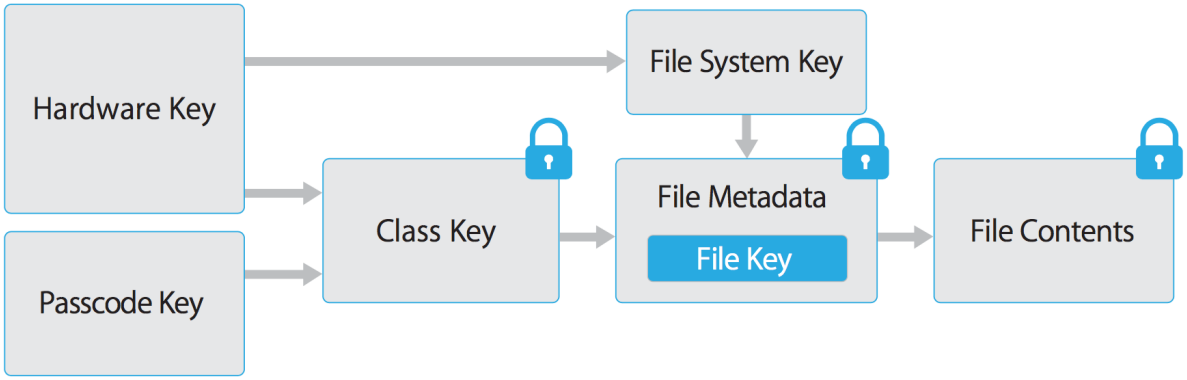
Application files written to Flash are encrypted:

- Per-file key: encrypts all file contents (AES-XTS)
- Class key: encrypts per-file key (ciphertext stored in metadata)
- File-system key: encrypts file metadata

Resetting device deletes file-system key

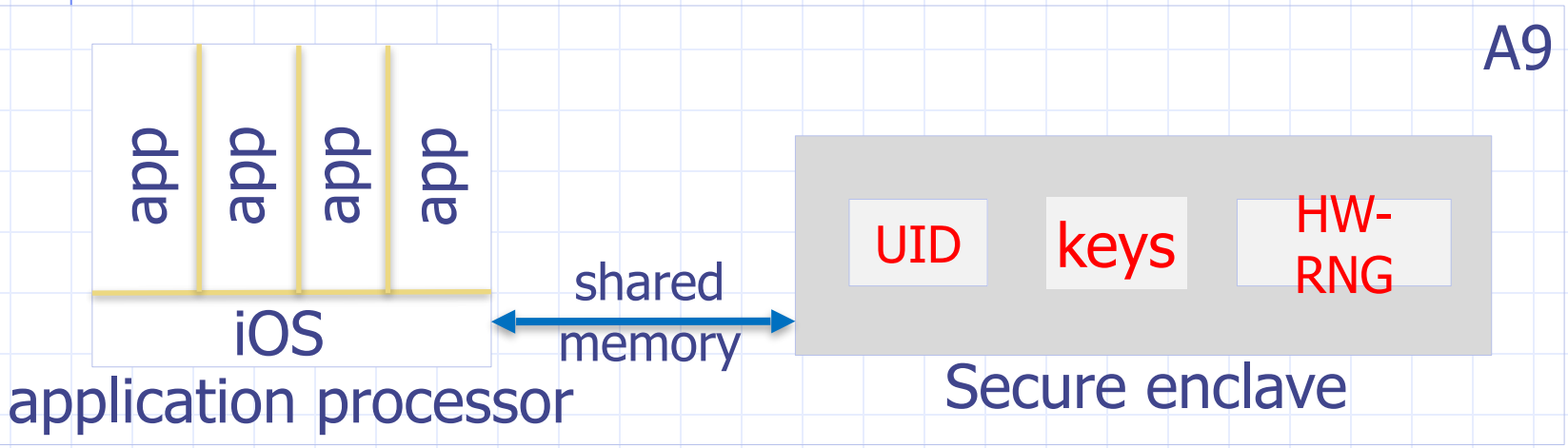
All key enc/dec takes place inside the secure enclave

⇒ key never visible to apps



Secure enclave (Apple A7 and later)

- Coprocessor fabricated in the Apple A7, A8, …
- All writes to memory and disk are encrypted with a random key generated in the enclave
- Used for device unlock, ApplePay, … (more on this later)



Backup to iCloud

- Data backup
  - Encrypted data sent from device to iCloud
  - But not applied to data of class NoProtection
  - Class keys backed up protected by "iCloud keys" (for device migration)
- Keychain class keys:
  - Non-migratory class keys

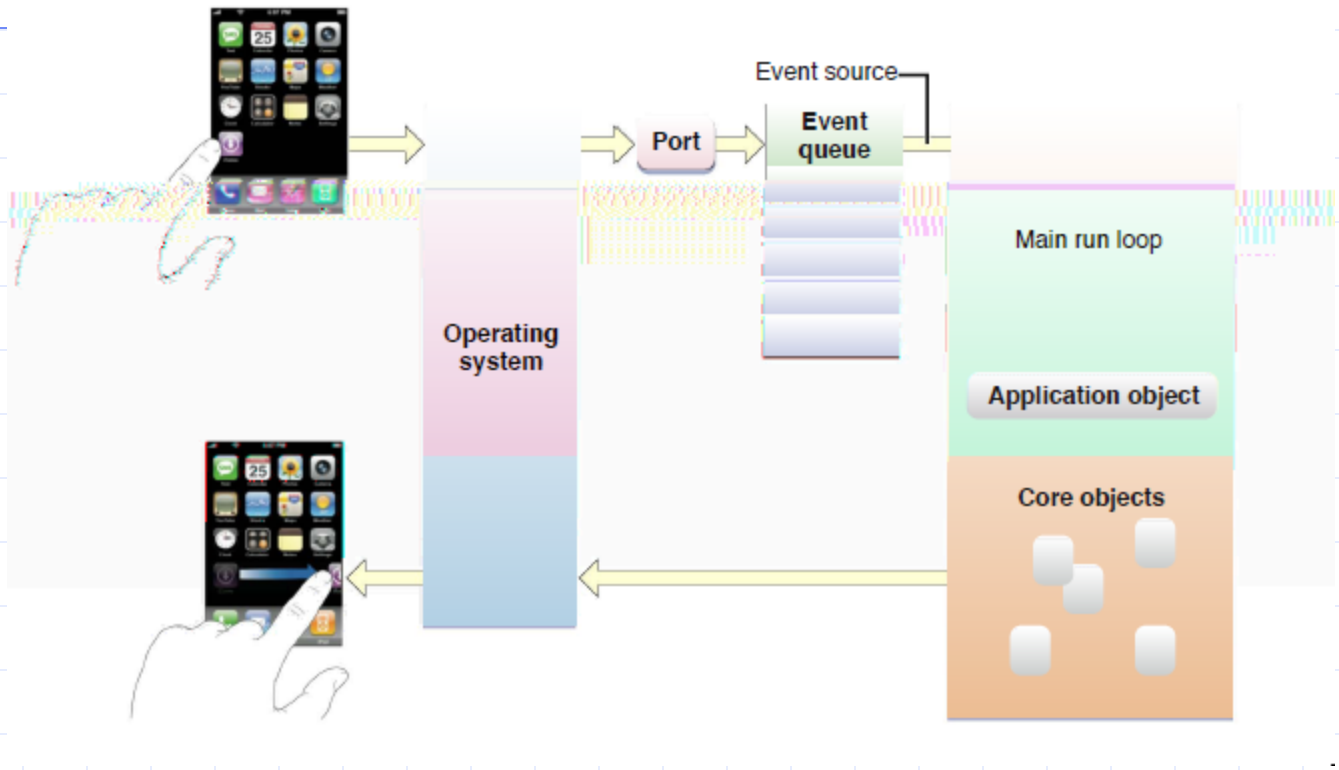
    wrapped with a UID-derived key

    ⇒ Can only be restored on current device
  - App-created items: not synced to iCloud by default

    [dict secObject:kCFBooleanTrue forKey:kSecAttrSynchronizable];

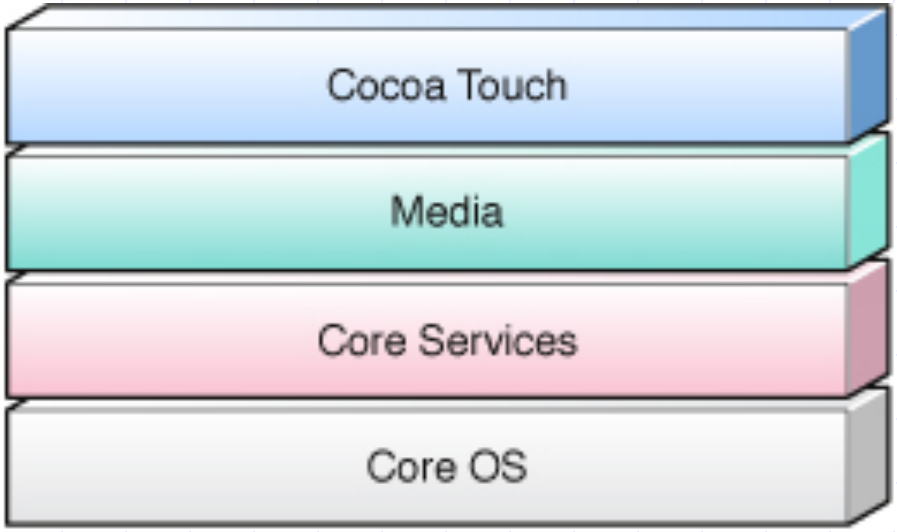
IOS APP DEVELOPMENT AND SECURITY

iOS Application Development



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

iOS Platform



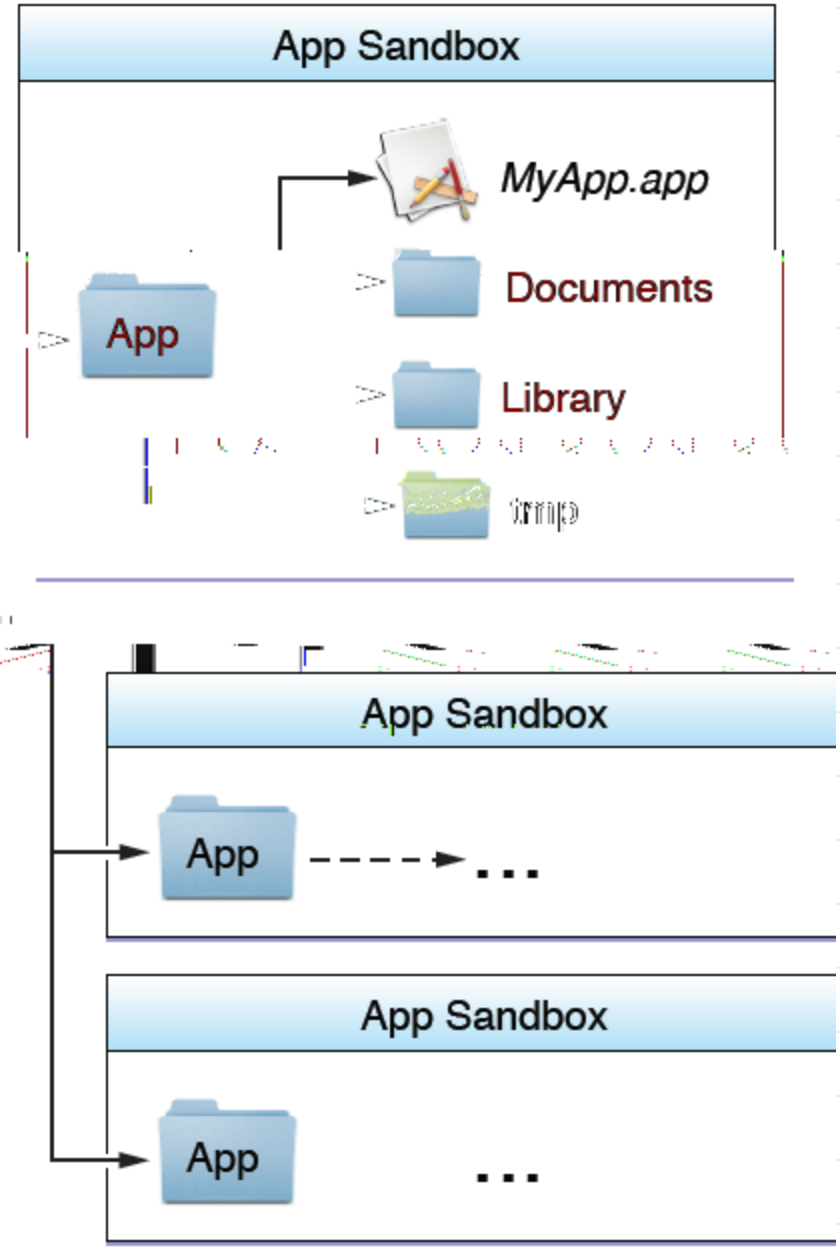
- Cocoa Touch

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

Implemented in C and Objective-C

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


App Sandbox
MyApp.app
App
Documents
Library
App Sandbox
App
...
App Sandbox
App
...

Runtime process security

- All 3rd party apps are sandboxed:

  run as the non-privileged user "mobile"

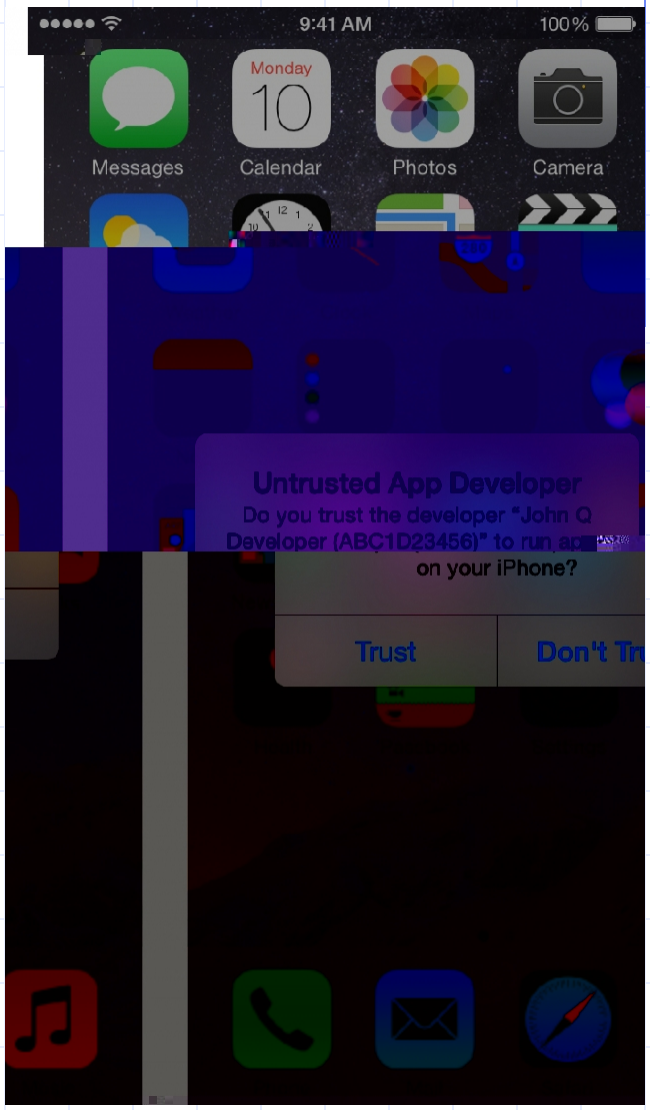
  - access limited by underlying OS access control
- Each app has a unique home directory for its files randomly assigned when the app is installed
- Accessing other info only through mediated services provided by iOS

App code signing

- All executable code must be signed by Apple certificate, including
  - Native apps
  - 3rd party apps (signed after Apple review)
  - Dynamic libraries
    - App can link against any dynamic library with the same TeamID (10-char string)
    - Example: an ad network library
- Not perfect: Charlie Miller's InstaStock app
  - stock ticker program: passed Apple review
  - After launch: downloads "data" from remote site, stores it in non-XN region, executes it ⇒ app becomes malicious
  - Why is there a non-XN region? Needed for Safari JIT.

"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



Several attacks occurred in 2015

Two lectures on mobile security

- Introduction: platforms and trends
- Threat categories
  - Physical, platform malware, malicious apps
- Defense against physical theft
- Malware threats
- System architecture anddefenses
  - Apple iOS security features and app security model
  - Android security features and app security model
- Security app development
  - WebView – secure app and web interface dev
  - Device fragmentation

Tues

Thurs