Conducting Technical Investigations on Apple iOS

Ken van Wyk
@KRvW

KRvW Associates, LLC

Further Reading / Study

"Hacking and Securing iOS Applications", Jonathan Zdziarski, O'Reilly, 2012

"iOS Hacker's Handbook", Charlie Miller, John Wiley & Sons, Inc., 2012

CS193p, Developing Apps for iOS, Stanford University, iTunes University

OWASP Mobile Top 10 Risks

M1- Insecure Data Storage	M6- Improper Session Handling
M2- Weak Server Side Controls	M7- Security Decisions Via Untrusted Inputs
M3- Insufficient Transport Layer Protection	M8- Side Channel Data Leakage
M4- Client Side Injection	M9- Broken Cryptography
M5- Poor Authorization and Authentication	M10- Sensitive Information Disclosure

Platform Architecture

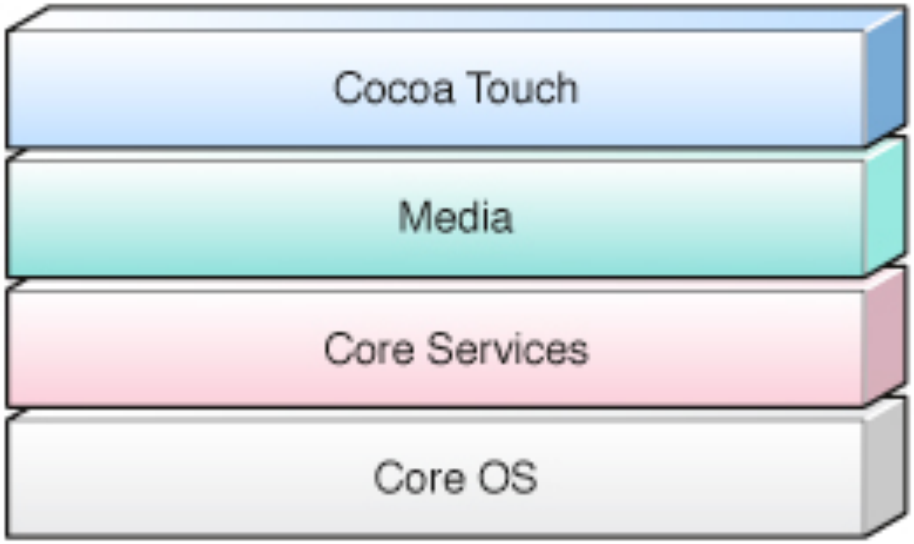
What the iOS / hardware platform offers us in the way of protection

KRvW Associates, LLC

iOS application architecture

The iOS platform is basically a subset of a regular Mac OS X system's

- From user level (Cocoa) down through Darwin kernel
- Apps can reach down as they choose to
- Only published APIs are permitted, however



Key security features

System hardening

Application sandboxing

App store protection

Hardware encryption

Keychains

SSL and certificates

System hardening features

- Attack surface reduction
- Stripped down OS
  - No /bin/sh
- Privilege separation
- Code signing
- Data execution prevention (DEP)
  - Vital for return oriented programming
  - No architectural separation of data and code segments
- Address space layout randomization (ASLR)

Application sandboxing

- By policy, apps are only permitted to access resources in their sandbox
  - Inter-app comms are by established APIs only
    - URLs, keychains (limited)
  - File i/o in ~/Documents only
- Sounds pretty good, eh?

App store protection

- Access is via digital signatures
  - Only registered developers may introduce apps to store
  - Only signed apps may be installed on devices
- Sounds good also, right?
  - But then there’s jailbreaking...
  - Easy and free
  - Completely bypasses sigs

App Store Review Limitations

- Don’t count on the App Store to find your app’s weaknesses
- Consider what they can review
  - Memory leaks, functionality
  - Playing by Apple’s rules
    - Published APIs only
  - Protecting app data?
    - Do they know your app?
  - Deliberate malicious “features”?

more ▾
Google
the droids we're looking for
Google Search
I'm Feeling
Advertising Programs - Business Solutions - Abo
©2009 - Privac
ogle - Mozilla F

Hardware encryption

- Each iOS device (as of 3Gs) has hardware crypto module
  - Unique AES-256 key for every iOS device
  - Sensitive data hardware encrypted
- Sounds brilliant, right?
  - Well...

iOS crypto keys

- GID key - Group ID key
- UID key - Unique per dev
- Dkey - Default file key
- EMF! - Encrypts entire file system and HFS journal
- Class keys - One per protection class
  - Some derived from UID + Passcode

iOS NAND (SSD) mapping

- Block 0 - Low level boot loader
- Block 1 - Effaceable storage
  - Locker for crypto keys, including Dkey and EMF!
- Blocks 2-7 - NVRAM parameters
- Blocks 8-15 - Firmware
- Blocks 8-(N-15) - File system
- Blocks (N-15)-N - Last 15 blocks reserved by Apple

File protection classes

- Pros
  - Easy to use, with key management done by iOS
  - Powerful functionality
  - Always available
  - Zero performance hit
- Cons
  - For Complete, crypto key is UDID + Passcode
    - 4 digit PIN problem
- Your verdict?

Built-in file protection classes

- iOS (since 4) supports file protection class
  - NSFileProtectionComplete
  - NSFileProtectionCompleteUnlessOpen
  - NSFileProtectionCompleteUntilFirstUserAuthentication
  - NSFileProtectionNone
    - This is the default protection class!

Keychains

- Keychain API provided for storage of small amounts of sensitive data
  - Login credentials, passwords, etc.
  - Encrypted using hardware AES
- Also sounds wonderful
  - Wait for it...

SSL and x.509 certificate handling

- API provided for SSL and certificate verification
  - Basic client to server SSL is easy
  - Mutual verification of certificates is achievable, but API is complex
- Overall, pretty solid
  - Whew!

And a few glitches...

Keyboard data

Screen snapshots

Hardware encryption is flawed

(And there’s no tooth fairy either)

Keyboard data

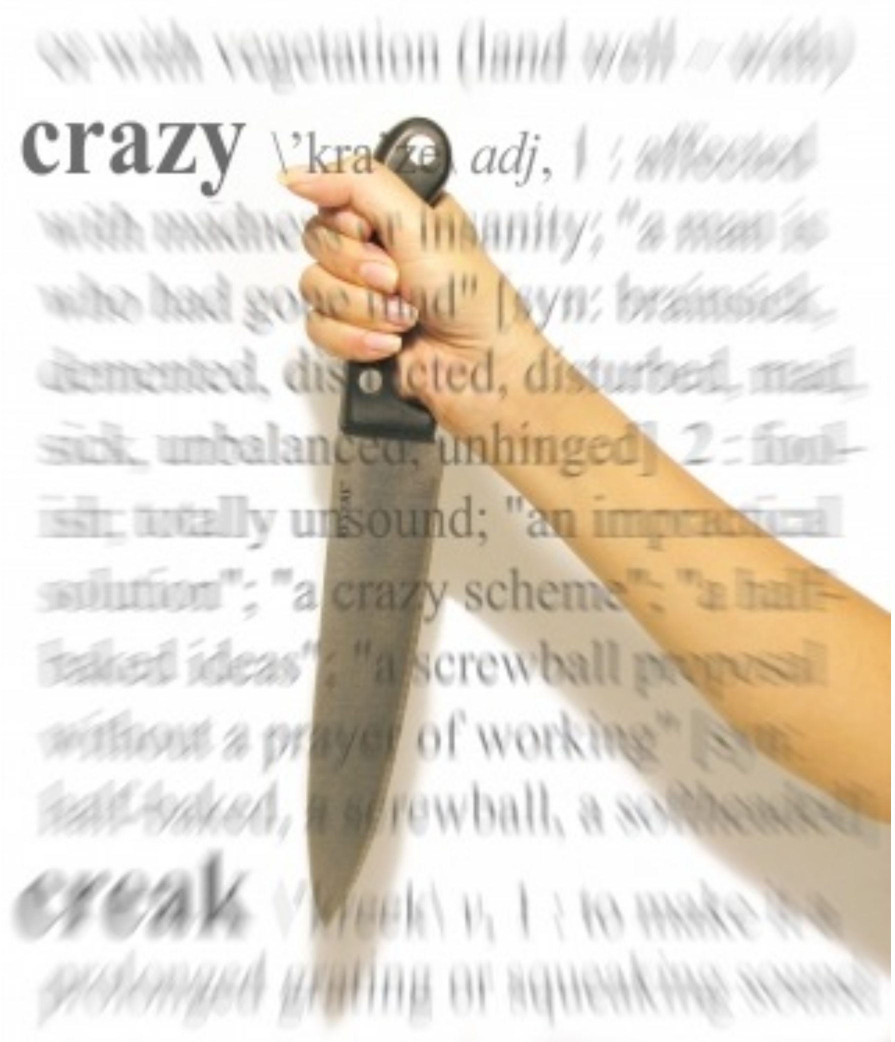
- All “keystrokes” are stored
  - Used for auto-correct feature
  - Nice spell checker
- Key data can be harvested using forensics procedures
  - Passwords, credit cards...
  - Needle in haystack?

Cut and paste

- That handy dandy pasteboard data is persistent
  - Reboot and see for yourself
  - Well, it’s gotta be stored somewhere, right?
    - It is
- Oh, and it has zero access control?
- Who cares?

Screen snapshots

- Devices routinely grab screen snapshots and store in JPG
  - Used for minimizing app animation
  - It looks pretty
- WHAT?!
  - It’s a problem
  - Requires local access to device, but still...



Then there are the self-inflicted

- Frameworks and other cached data too
  - Some store data in the name of persistency
    - Without warning
- It pays to study and update
  - "Update those apps with updated frameworks!"
    - Said no app developer, ever

But the clincher

- Passcode can trivially be bypassed
  - Jailbreak (or similar) software via DFU mode to boot custom kernel
  - Brute force break the 4-digit PIN
- No more protection...
  - Well, for PINsters, anyway

CERT operations

- OK, so how does all of this affect a CERT / CSIRT?
  - There is a lot to consider
  - Give up on perfection
  - It’s all about varying degrees of imperfect and how we can deal with them

Some tools we'll be using

We'll also later use a couple others

Burpsuite -- another web app proxy, but handles SSL really easily

iExplorer -- allows us to look at the files on an iOS device

- Non-destructively, of course
- Does NOT require any jailbreaking to work

Xcode, iPhone simulator, and Finder

- To build some apps and explore their file systems

Oh, and the "evasi0n" jailbreak too

Attack vector: lost/stolen device

- Anyone with physical access to your device can get to a wealth of data
  - PIN is not effective
  - App data
  - Keychains
  - Properties
- See forensics studies
- Your app must protect users' local data storage

M1- Insecure Data Storage

- Sensitive data left unprotected
- Applies to locally stored data + cloud synced
- Generally a result of:
  - Not encrypting data
  - Caching data not intended for long-term storage
  - Weak or global permissions
  - Not leveraging platform best-practices

Impact

- Confidentiality of data lost
- Credentials disclosed
- Privacy violations
- Non-compliance

Incident scenario - VIP lost device

Post facto, what is the exposure?

Restore backup onto new hardware

Jailbreak and examine data stores

- Starting points

  /private/var/mobile

- Let's explore...

Protecting secrets at rest

Encryption is the answer, but it’s not quite so simple

- Where did you put that key?
- Surely you didn’t hard code it into your app
- Surely you’re not counting on the user to generate and remember a strong key

Key management is a non-trivially solved problem

How bad is it?

- It’s tough to get right
  - Key management is everything
- We’ve seen many examples of failures
  - Citi and others
- Consider lost/stolen device as worst case
  - Would you be confident of your app/data in hands of biggest competitor?

Tools to use

- Mac tools
  - Finder
  - iExplorer
  - hexdump
  - strings
  - otool
  - otx (otx.osxninja.com)
  - class-dump (iphone.freecoder.org/classdump_en.html)
  - Emacs (editor)
- Xcode additional tools
  - Clang (build and analyze)
    - Finds memory leaks and others

What to examine?

- See for yourself
  - There is no shortage of sloppy applications in the app stores
  - Start with some apps that you know store login credentials

Static analysis of an app

- Explore folders
  - ./Documents
  - ./Library/Caches/*
  - ./Library/Cookies
  - ./Library/Preferences
- App bundle
  - Hexdump of binary
  - plist file
- What else?

Places To Look

In */private/var/ mobile*...

Library/Cookies/ - web page cookies

Media/Photos/ - thumbnails of photo albums

Media/DCIM/ - camera roll

Library/Caches/Safari/ - Safari history, bookmarks

Library/Keyboard/ - spellcheck kbd log

Library/caches/ Snapshots - recent screen shots

Many more...

Other Treasures

SMS - deleted and otherwise

Address book

Calendar

Phone log

Attack vector: coffee shop attack

- Exposing secrets through non-secure connections is rampant
  - Firesheep description
- Most likely attack targets
  - Authentication credentials
  - Session tokens
  - Sensitive user data
- At a bare minimum, your app needs to be able to withstand a coffee shop attack

M3- Insufficient Transport Layer Protection

- Complete lack of encryption for transmitted data
  - Yes, this unfortunately happens often
- Weakly encrypted data in transit
- Strong encryption, but ignoring security warnings
  - Ignoring certificate validation errors
  - Falling back to plain text after failures

Impact

- Man-in-the-middle attacks
- Tampering w/ data in transit
- Confidentiality of data lost

Incident scenario - employee account compromised

Post facto, what is the exposure?

Dynamic analysis of apps in use

- Look for non-SSL data
- Look for inadequate SSL certificate validation

Dynamic analysis of an app

Test rig set up

- Web application proxy tool
- Point device network interface to proxy IP number
- Capture GETs and POSTs
- Configure to present a "proper" SSL certificate to mobile app

Protecting users' secrets in transit

Always consider the coffee shop attack as lowest common denominator

We place a lot of faith in SSL

- But then, it's been subjected to scrutiny for years

How bad is it?

Neglecting SSL on network comms is common

Consider the exposures

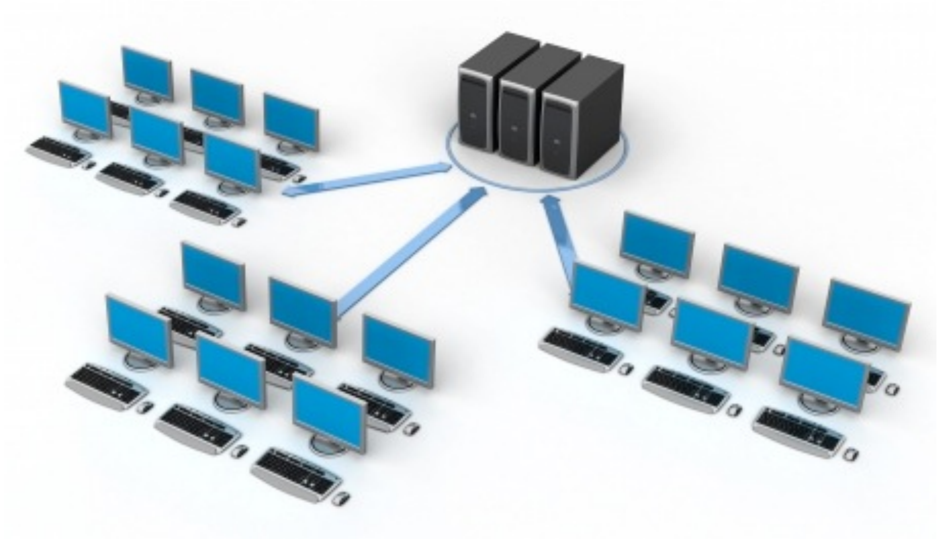
- Login credentials
- Session credentials
- Sensitive user data

Will your app withstand a concerted coffee shop attacker?

Attack vector: web app weakness

Remember, modern mobile devices share a lot of weaknesses with web applications

- Many shared technologies
- A smart phone is sort of like a mobile web browser
  - Only worse in some regards



Input and output validation

- Problems abound
  - Data must be treated as dangerous until proven safe
  - No matter where it comes from
- Examples
  - Data injection
  - Cross-site scripting

Where do you think input validation should occur?

SQL Injection

Most common injection attack

- Attacker taints input data with SQL statement
- Application constructs SQL query via string concatenation
- SQL passes to SQL interpreter and runs on server

Consider the following input to an HTML form

- Form field fills in a variable called "CreditCardNum"
- Attacker enters
  - ‘
  - ‘ --
  - ‘ or 1=1 --
- What happens next?

And introducing... The employee/ attacker

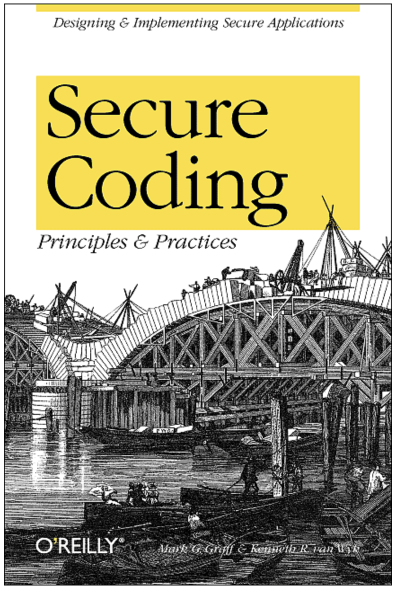
- Employee using a BYOD device to attack company assets
  - Authorized access to systems
  - Any of variety of motivations
    - Disgruntled for some reason
    - Personal gain

Where do we look?

- Logging (local)?
  - Not in /var/log/*
  - System logs are primarily for debugging, not security
- Files?
  - SMS, browser history, etc.
  - Helpful, but circumstantial
- System (server) logs can help corroborate

Kenneth R. van Wyk
KRvW Associates, LLC

Ken@KRvW.com
http://www.KRvW.com


Designing & Implementing Secure Applications
Secure Coding
Principles & Practices
O'REILLY
Mark G. Graff & Kenneth R. van Wyk