Pen Testing iOS Apps

FIRST 2015
Ken van Wyk, @KRvW

Berlin, Germany
14-19 June 2015

KRvW Associates, LLC

Ken van Wyk, ken@krvw.com, @KRvW

Topics we’ll cover

- We’ll focus on how to break typical iOS apps
  - iOS topics
  - Application topics
- Simple analysis
  - Surface of app
  - Static analysis
  - Dynamic analysis
- Deeper analysis
  - Explore app binary
  - Run-time exploration and exploitation

Tools

Most tools we'll use are either open source or inexpensive

- iExplorer for exploring file system on an iOS device
- iOS device and a USB cable
  - Preferably jailbroken
  - Cydia
  - Cycript

Clear up some misconceptions

- Apple’s iOS has been a huge success for Apple
  - Together with Android, they have re-defined mobile telephony
- Apple has made great advances in security
  - They are still far from really good
  - Not even sure if they’re pretty good

Software developers still make silly mistakes

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

These rules don't always apply to Apple's own apps

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

Built-in file protection classes

iOS (since 4) supports file protection classes

- NSFileProtectionComplete
- NSFileProtectionCompleteUnlessOpen
- NSFileProtectionCompleteUntilFirstUserAuthentication
- NSFileProtectionNone

All but None are derived

Built-in file protection limitations

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

Keychains

- Keychain API provided for storage of small amounts of sensitive data
  - Login credentials, passwords, etc.
  - Credit card data often found here
- Stored in a SQLite database
  - Encrypted using hardware AES with derived key

Jailbreaks

- Apple’s protection architecture is based on a massive digital signature hierarchy
  - Starting from bootloader
  - Through app loader
- Jailbreak software breaks that hierarchy
  - Current breaks up to 8.1.2
- DFU mode allows USB vector for boot loader
  - Older iPhones mostly, but…

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?

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...

crazy \'krā-zē\ adj, 1 : affected
with madness or insanity; "a man is
who had gone mad" [syn: brainsick,
demented, distracted, disturbed, mad,
sick, unbalanced, unhinged] 2 : fool-
ish; totally unsound; "an impractical
solution"; "a crazy scheme"; "a half-
baked ideas"; "a screwball proposal
without a prayer of working" [syn:
half-baked, screwball, softheaded
creak

Let’s consider the basics

We’ll cover these (from the mobile top 10)

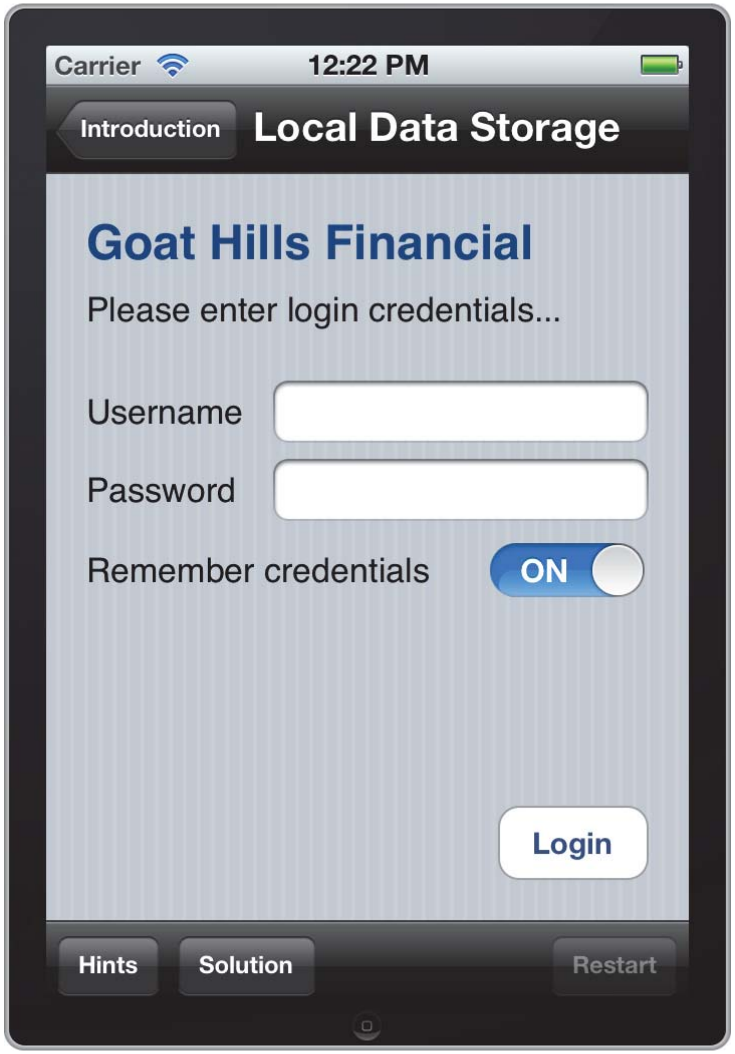
- Protecting secrets
  - At rest
  - In transit
- Input/output validation
- Authentication
- Session management
- Access control
- Privacy concerns

Examples

- Airline app
  - Stores frequent flyer data in plaintext XML file
- Healthcare app
  - Stores patient data in plist file
    - But it’s base64 encoded for your protection…
- Banking app
  - Framework cache revealed sensitive account data
- Consumer ticket app
  - Accepted SSL from self signed key
  - Exposed credit card data

SQLlite example

- Let’s look at a database app that stores sensitive data into a SQLite db
  - We’ll recover it trivially by looking at the unencrypted database file



Goat Hills Financial

Please enter login credentials...

Username

Password

Remember credentials ON

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

Static analysis of an app

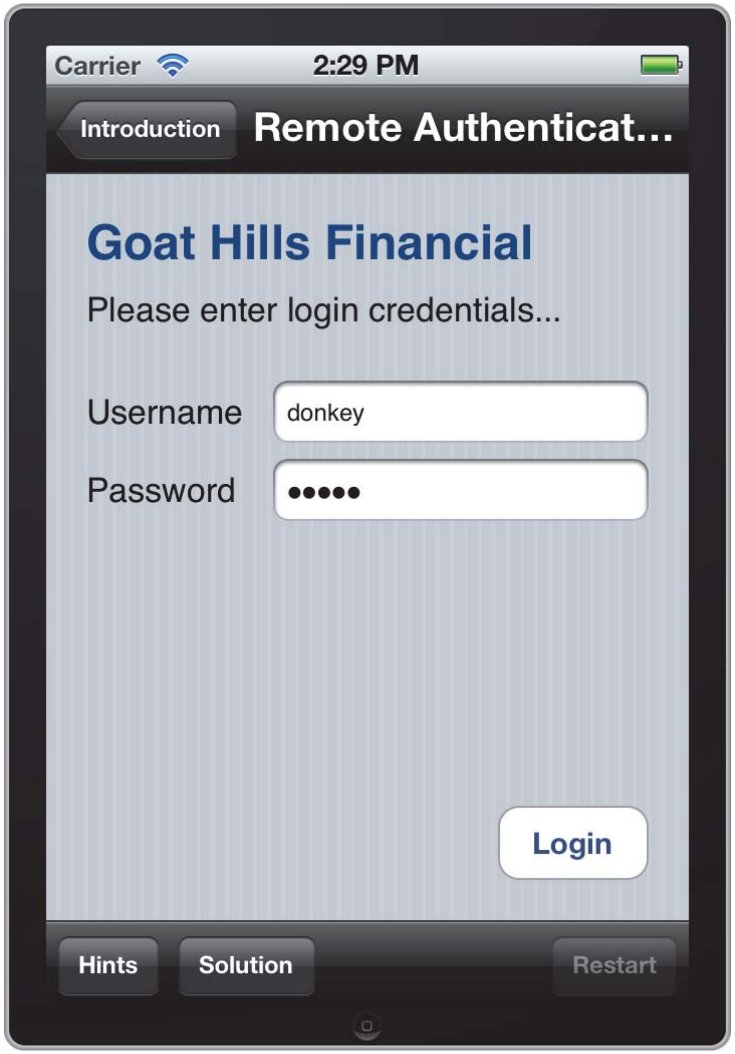
- Explore folders
  - ./Documents
  - ./Library/Caches/*
  - ./Library/Cookies
  - ./Library/Preferences
- App bundle
  - Hexdump of binary
  - plist files
- What else?

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

Exercise - coffee shop attack

- This one is trivial, but let’s take a look
- In this iGoat exercise, the user’s credentials are sent plaintext
  - Simple web server running on Mac responds
  - If this were on a public WiFi, a network sniffer would be painless to launch



Goat Hills Financial

Please enter login credentials...

Username donkey

Password •••••

Most common SSL mistake

- We’ve all heard of CAs being attacked
  - That’s all important, but...
  - (Certificate pinning can help.)
- Failing to properly verify CA signature chain
  - Biggest SSL problem by far
  - Study showed 1/3 of Android apps fell to this

Testing for SSL problems

- Goal is to ensure client performs strong certificate verification
- MITM on the net setup
  - App proxy on laptop (e.g., Burpsuite)
  - Generate SSL cert signed by your own CA
  - Put your CA cert on test iOS device

Remember to remove fake CA before leaving lab environment!

But that’s not enough

ObjC Run-time is flawed

- Unlike in C, “functions” are not called
  - Messages are passed
  - Objects dynamically allocated
- Within process space, dynamic tampering also possible
  - Message traffic
  - Objects

Reverse engineering

- Attacker wants to learn how your app works
  - Deep internal details
- Attacker wants to attempt to trick your app into misbehaving
  - Tamper with runtime
- How? Jailbroken device and some free tools
  - And a *lot* of time

Prerequisite tools and env

- Mac with OS X and Xcode
- Jailbroken device
  - evasi0n works great
- Cydia and friends
  - Cydia installed with evasi0n
  - Shell access
    - OpenSSH - install with Cydia
  - Debugger
    - gdb - install with Cydia

Bare minimum essentials

Analysis techniques

- Static analysis
  - Observe attributes of the executable, app files
  - Yes, encrypted (app store) apps too
- Dynamic analysis
  - Run the app and learn how it works
- Tampering
  - Trick the run-time env

Static analysis

- Any binary can be examined
  - Usually reveal a map to classes, objects, text, symbols, etc.
- Common tools
  - otool
  - class-dump-z
  - nm

- Examples
  - Linked libs, methods
    - otool -L appname
    - otool -l appname
  - List of classes
    - class-dump-z appname
  - Symbol table
    - nm appname

It’s C underneath the hood

Beneath that nice OOP ObjC layer lies a C foundation

Pretty much everything in ObjC can be done in C

- Primitives for doing all the OO stuff
- *objc_msgSend()*, *objc_getClass()* are prime examples

This matters to us when analyzing statically or dynamically

Encrypted binaries too

- Basic process
  - Use app loader to decrypt
  - Calculate memory offsets
  - Store process to disk
    - dd is your friend
    - Will also need plutil and gdb
- HOWTO available
  - http://www.mandalorian.com/2013/05/decrypting-ios-binaries/

Let’s take a look…

Dynamic analysis

What can we learn from observing it running?

- A lot
- All those messages
- Memory contents
- CPU registers

You don’t have anything to hide, right?

Attacking a running app

Man in the app (MITA)

- The most dangerous form of on-host dynamic attack
- Internal access to everything

That ObjC run-time messaging architecture is going to haunt us

A few more tools

For these, you’ll want

- gdb
- Cycript (see slide)
- Network proxy (e.g., Burpsuite)
- SSLstrip (optional)

Message eavesdropping

Use gdb to build a simple but effective message eavesdropper

Example

```
gdb -q -p PID
break obj_msgSend
commands
x/a $r0
x/s $r1
c
```

Cycript

“Cycript allows developers to explore and modify running applications on either iOS or Mac OS X using a hybrid of Objective-C++ and JavaScript syntax through an interactive console that features syntax highlighting and tab completion”

— From http://www.cycript.org

It is an amazing utility for dynamically probing a running app

Fun with Cycript

Basics

```
# cycript
cy# var myString = [[ NSString alloc ]
cy> initWithString: @“Hello world”];
“Hello world”
cy# [ myString length ];
11
```

Combination of JavaScript and ObjC syntax gives amazing capabilities

Cycript (2)

Safari example

```
# cycript -p PID
cy# var app = [UIApplication sharedApplication];
“<UIApplication: 0x22f050>”
cy# [ app openURL: [ NSURL URLWithString:
cy> @“http://www.first.org”]];
1
cy# app.networkActivityIndicatorVisible = YES
```

Cycripting for fun and profit

- Break client-side logic
  - Alter PINs, booleans, semaphores
  - Replace methods
- Probe running app data
  - Can be verbose, but you get everything in an object

```
    cy# function appls(a){ var x={};
for(i in *a){ try{ x[i] = (*a)[i]; }
catch(e){}} return x; }
    cy# appls(object);
```

Client-side logic

You didn't think you could trust client-side logic, did you?

Tampering

- Now let's go beyond mere observation
- Replace existing methods
  - Change address in gdb
  - Dynamic linker attack
    - Put your library in DYLD_INSERT_LIBRARIES
- Automate dynamic linking
  - MobileSubstrate

Nothing is what it appears

Now we can change the entire universe your app runs in

(If this doesn't seem bad, go watch The Matrix)

Resources

Hacking and Securing iOS Applications, Jonathan Zdziarski, O’Reilly, 2012

Evasi0n, popular jailbreaking tool, http://www.evad3rs.com/

Hardening

User actions and client configurations

Architectural considerations

Hardening tips

But remember, nothing is perfect.

User actions and configurations

Strong passcodes help

MDMs can manage configurations of entire fleets

LOADING ZONE
15 MINUTE
AUTHORISED VEHICLES EXCEPTED
7-10 AM
S
AUTHORISED VEHICLES EXCEPTED
10 AM - 10 PM
S
7 - 10 PM
TAXIS EXCEPTED WHEN PICKING UP/ DROPPING OFF PASSENGERS
TAXI ZONE
ALL OTHER TIMES
P
10 PM - 7 AM

Architectural considerations

Design choices make a huge difference

Client cannot be trusted

- Sensitive data
- Sensitive functions
- Security controls

Client should provide presentation layer

- Minimal functionality
- Processing should be server

Hardening tips

- Non-obvious names
  - Obfuscate functional purpose
- Disable debugging

```
#define DENY_DEBUG 31
ptrace(DENY_DEBUG,0,0,0);
```

- Complicate disassembly
  - Compiler optimizer
  - Strip symbols

Hardening tips (2)

- Sensitive code
  - On server, but…
  - Write in C or ASM
  - Compile + link in-line
  - Expand loops manually
- Force your attacker to single step through
- Don’t give away anything

Hardening (3)

- Data storage
  - Encrypt
    - DataProtection API for consumer grade
    - Keys on server
  - Common Crypto Lib
- Secure file wiping
- SQLite data wiping
  - Update before delete

Tamper detection

How do we know?

Run-time integrity checks

- Memory offsets of sensitive objects

Sandbox integrity

- Attempt to fork
- Size and checksum of */etc/fstab*
- Symbolic links in */Applications*
- Common jailbreak files and apps

  /Applications/Cydia.app

Honeypots in app

There ain't a horse that can't be rode or a man that can't be throwed.

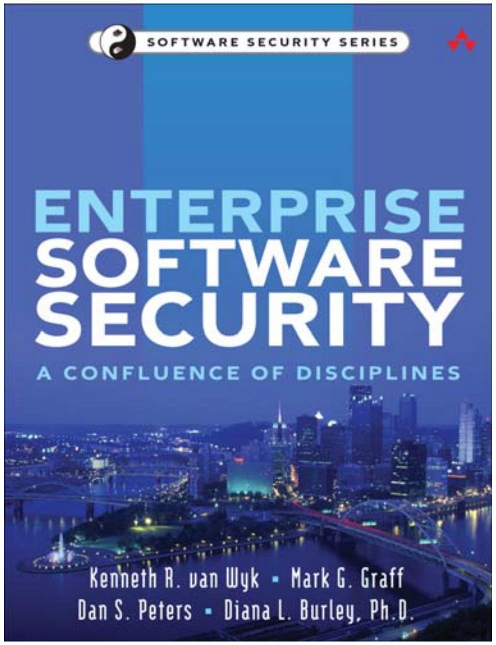
Tamper response

What to do?

- Remote wipe
- Phone home
- Log everything
- Wipe user data, keys
- Disable network access
- Et cetera

Kenneth R. van Wyk
KRvW Associates, LLC

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


SOFTWARE SECURITY SERIES
ENTERPRISE SOFTWARE SECURITY
A CONFLUENCE OF DISCIPLINES
Kenneth R. van Wyk • Mark G. Graff
Dan S. Peters • Diana L. Burley, Ph.D.