

Break 'em and Build 'em iOS

SecAppDev 2016
Ken van Wyk, @KRvW

*Leuven, Belgium
7-11 March 2016*

KRvW Associates, LLC

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

Copyright© 2016 KRvW Associates, LLC



Part I - Break 'em!

Biggest issue: 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

Data encryption helps, but we can't count on developers using it



Second biggest: insecure comms

Without additional protection, mobile devices are susceptible to the “coffee shop attack”

Anyone on an open WiFi can eavesdrop on your data

No different than any other WiFi device really

Your apps **MUST** protect your users’ data in transit



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



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 at boot time



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



WHAT?!

Yes, these keys are stored
in plaintext

No, you shouldn't be able
to access them

This has improved greatly in
the past couple of years

Requires run-time exploit or
replaced boot loader



Jailbreaks

Apple's protection architecture is based on a massive digital signature hierarchy

Starting from bootloader

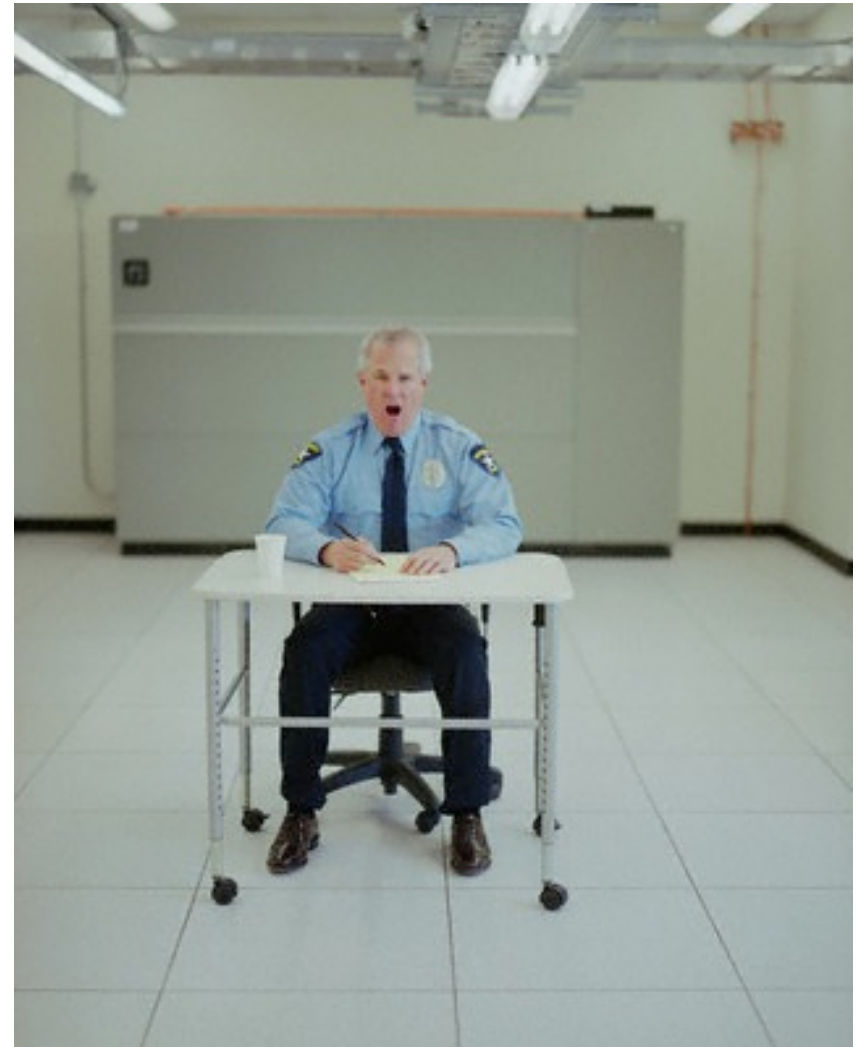
Through app loader

Jailbreak software breaks that hierarchy

Current breaks up to 9.0.2

DFU mode allows USB vector for boot loader

Older iPhones mostly, but...



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...

Key is as strong as your device passcode



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

Your verdict?



Built-in file protection classes

iOS (since 4) supports file protection classes

NSFileProtectionComplete

NSFileProtectionCompleteUnlessOpen

NSFileProtectionCompleteUntilFirstUserAuthentication

NSFileProtectionNone



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...



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



Static analysis of an app

Explore folders

./Documents

./Library/Caches/*

./Library/Cookies

./Library/Preferences

App bundle

Hexdump of binary

plist files

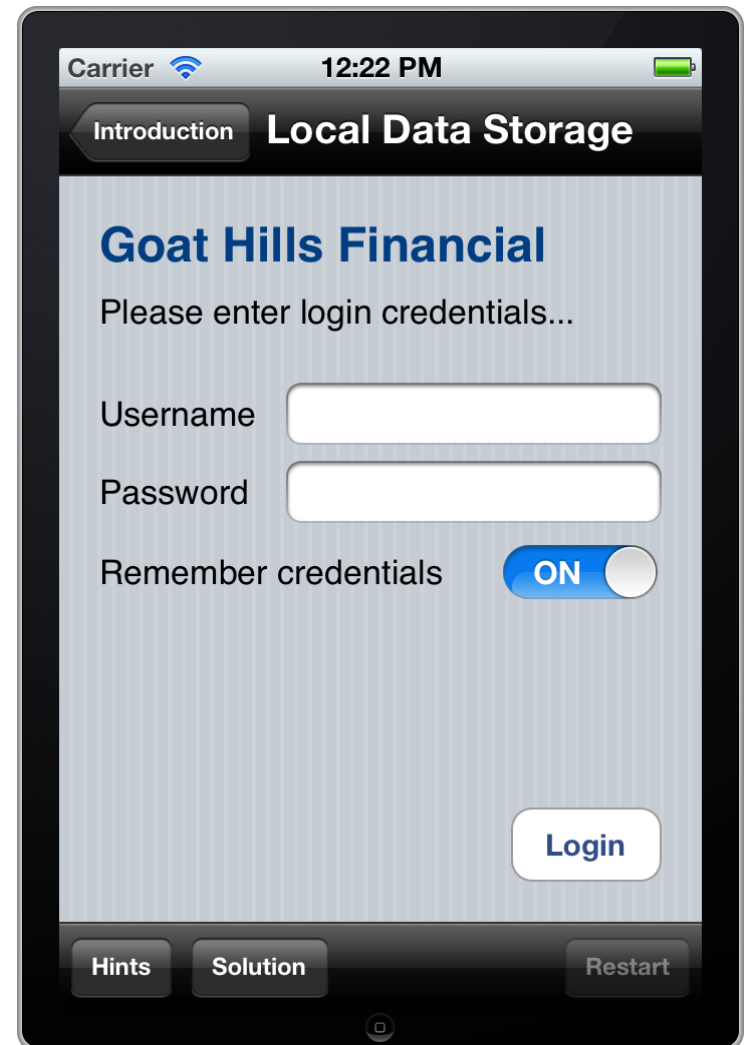
What else?



SQLite 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



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 protection...

Banking app

Framework cache revealed sensitive account data



Tools to use

Mac tools

Finder

iExplorer

hexdump

strings

otool

otx (otx.osxninja.com)

class-dump

([iphone.freecoder.org/
classdump_en.html](http://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



Let's go further

Consider jailbreaking to further analyze things

Get outside of app sandbox

All OS files exposed

- Keylog, SMS, email

Tethered vs. untethered

Tools and notes

Works up to 8.1.2 on iPhone

6

- Evasi0n and others
- Plus Cydia, of course...



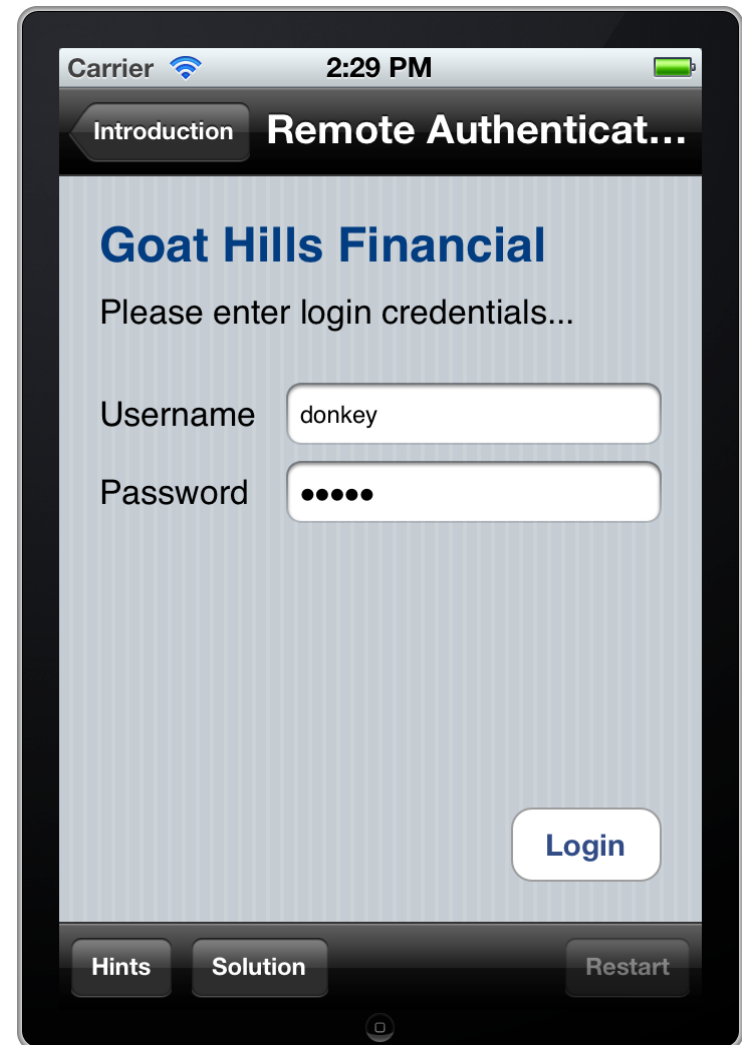
Example - coffee shop attack

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



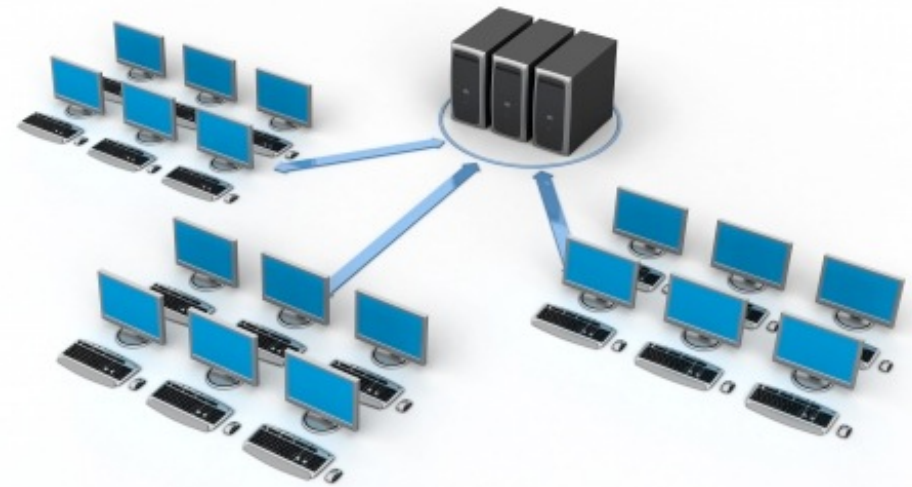
Attack vector: web app weakness

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



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?

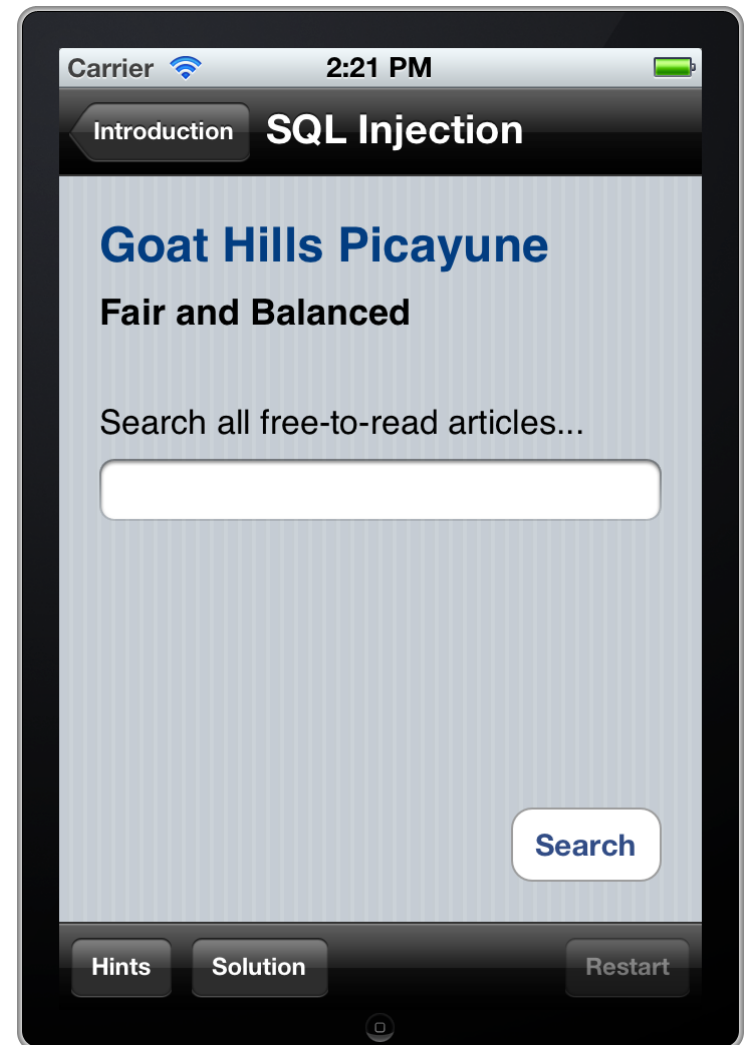
SQL injection exercise - client side

In this one, a local SQL db contains some restricted content

Attacker can use “SQLi” to view restricted info

Not all SQLi weaknesses are on the server side!

Question: Would db encryption help?



Part II - Build 'em!

Stanford Univ on iTunes



Apple resources

Excellent developer references and manuals on iOS Developer Portal

<http://developer.apple.com/devcenter/ios/index.action>

Several free iBooks also

Objective C

Cocoa Framework



Also look at OWASP

Numerous information resources that are relevant to mobile apps

Mobile Security Project

Growing community of mobile developers at OWASP



And then there's OWASP's iGoat

OWASP project for iOS
devs

iGoat

Developer tool for learning
major security issues on iOS
platform

Inspired by OWASP's
WebGoat tool for web apps

Released 15 June 2011



iGoat Layout

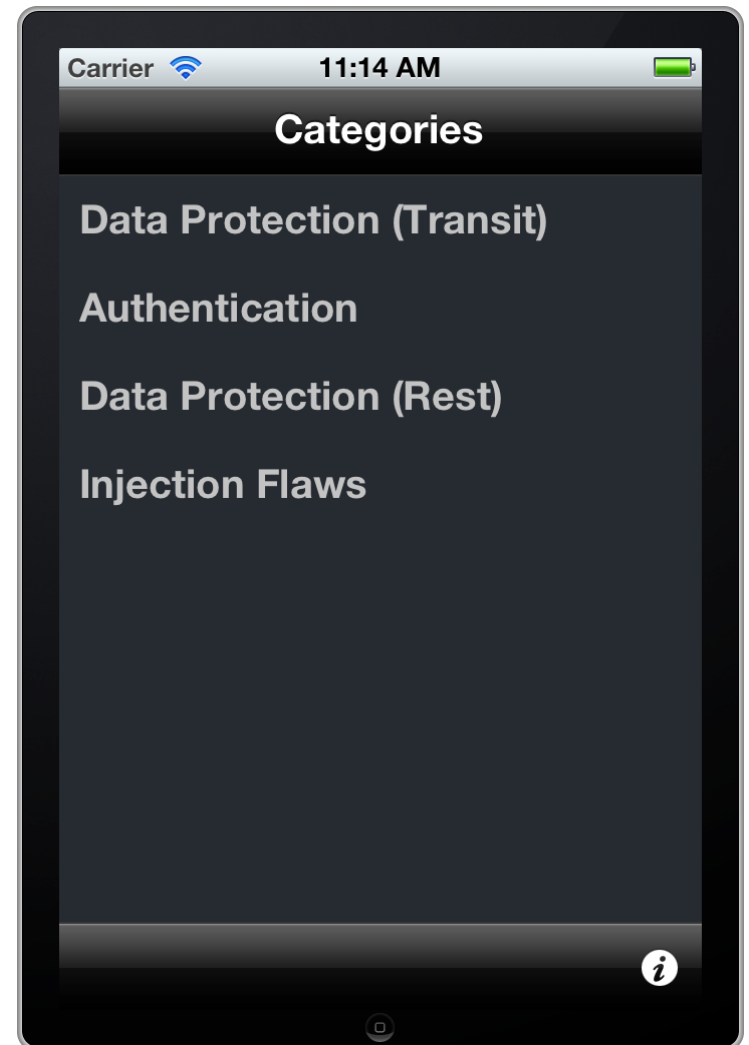
Exercise categories

Data protection (transit)

Authentication

Data protection (rest)

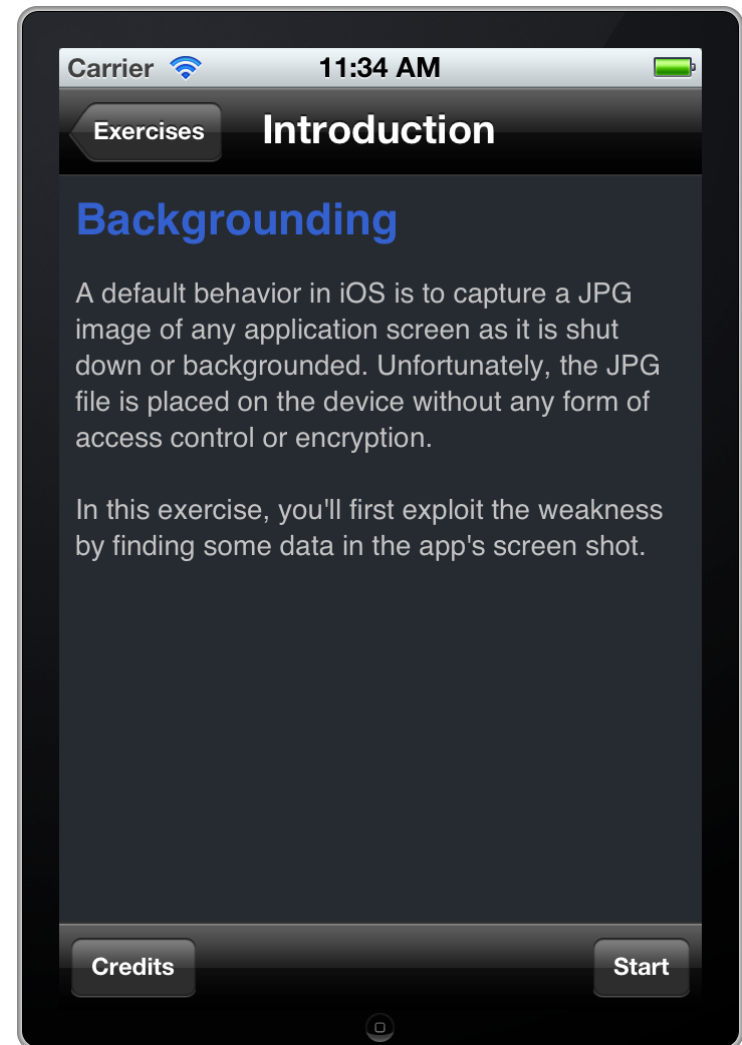
Injection



Exercise example - Backgrounding

Intro describes the nature of the issue


Credits page too, so others can contribute with due credit



Exercise example - Main screen

This screen is the main view of the exercise

Enter data, etc., depending on the exercise

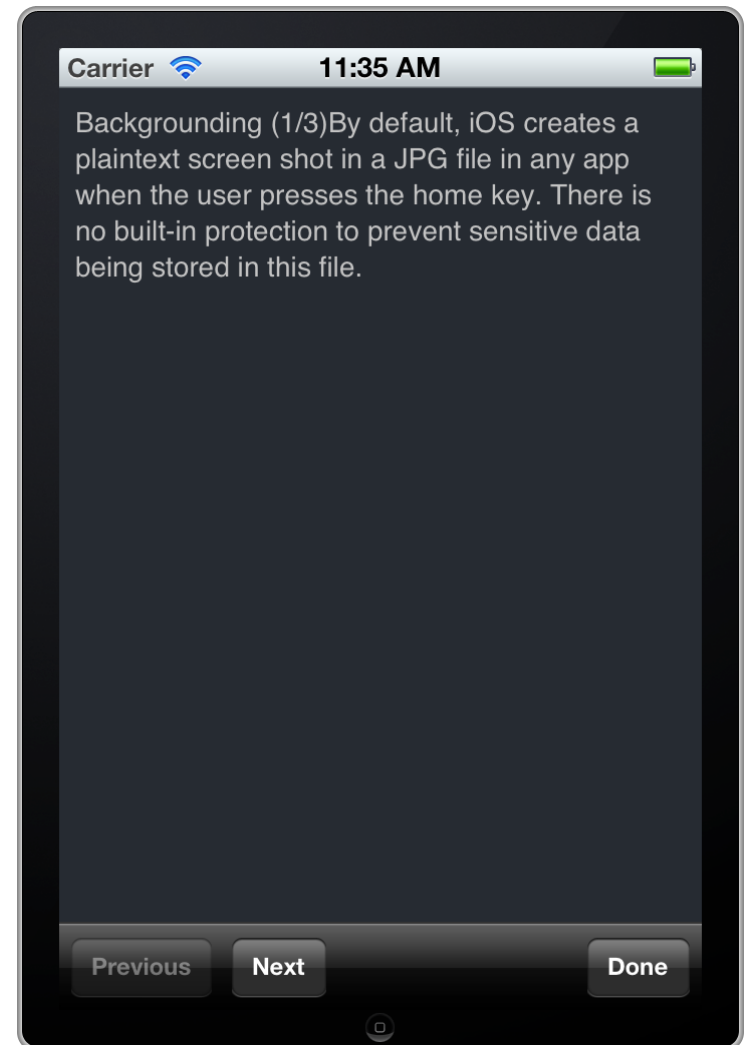


The screenshot shows an iPad app interface. At the top, the status bar displays 'Carrier', a Wi-Fi signal icon, '11:35 AM', and a battery icon. Below the status bar is a dark navigation bar with two tabs: 'Introduction' and 'Backgrounding', with 'Backgrounding' being the active tab. The main content area has a light blue background with vertical stripes. It features the title 'Goat Hills Financial' in blue, followed by the text 'Password reset...'. There are two text input fields: the first is labeled 'In what city were you born?' and the second is labeled 'What is your favorite color?'. A 'Submit' button is located at the bottom right of the main content area. At the very bottom of the screen is a dark navigation bar with three buttons: 'Hints', 'Solution', and 'Restart'.

Exercise - Hints

Each exercise contains a series of hints to help the user

Like in WebGoat, they are meant to help, but not quite solve the problem



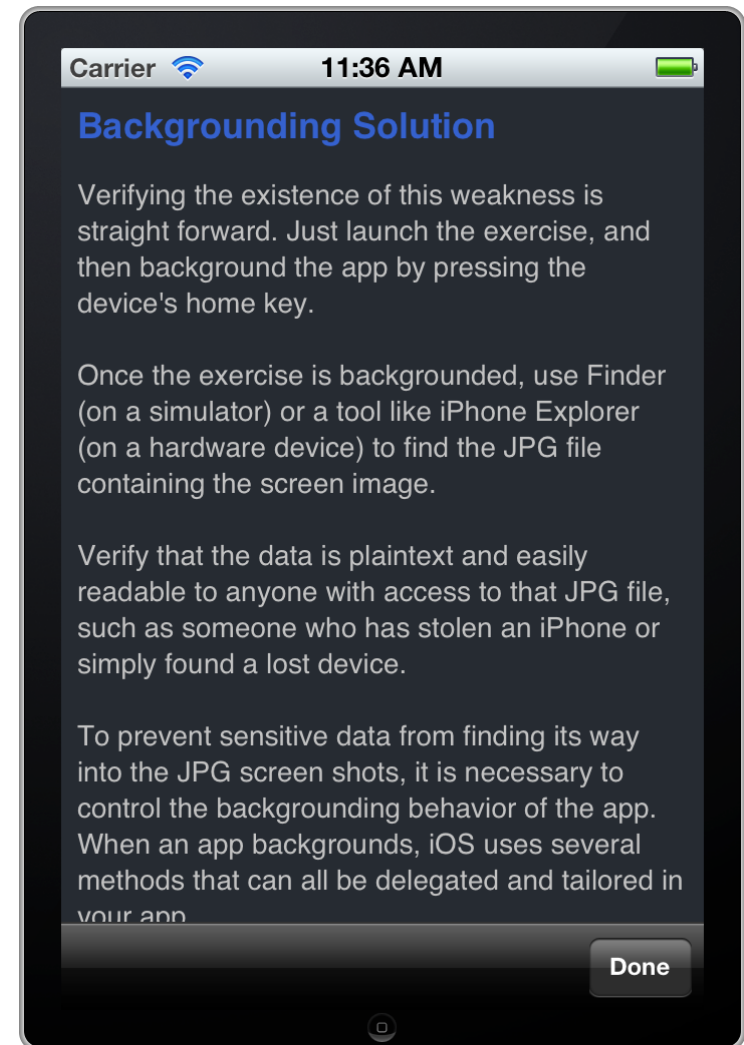
Exercise - Solution

Then there's a solution page for each exercise

This describes how the exercise can be solved

No source code remediations yet

That comes in the next step



Now let's try one

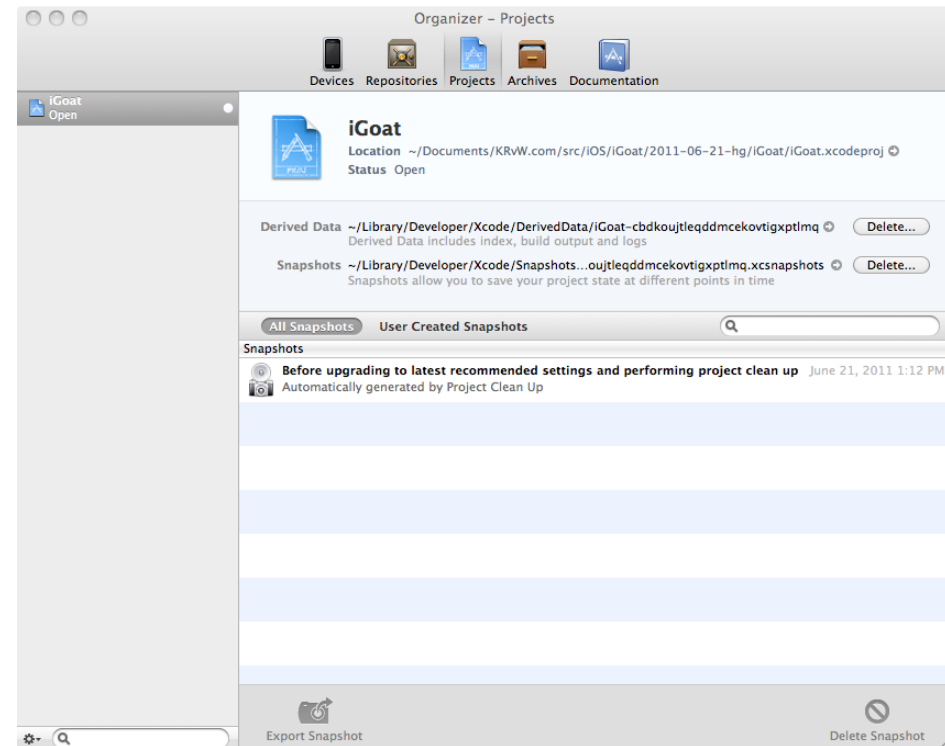
You're welcome to follow along on your Macs

You'll need

Xcode SDK for iOS

iGoat distribution

- Download tarball and unpack



iGoat URLs

Project Home:

https://www.owasp.org/index.php/OWASP_iGoat_Project

Source Home:

<http://code.google.com/p/owasp-igoat/>

Kenneth R. van Wyk
KRvW Associates, LLC

Ken@KRvW.com

<http://www.KRvW.com>

