Overseeing the Orchard

Managing an iOS Fleet

1/2-day Class

KRvW Associates, LLC

Your Instructor – Ken van Wyk
ken@krvw.com

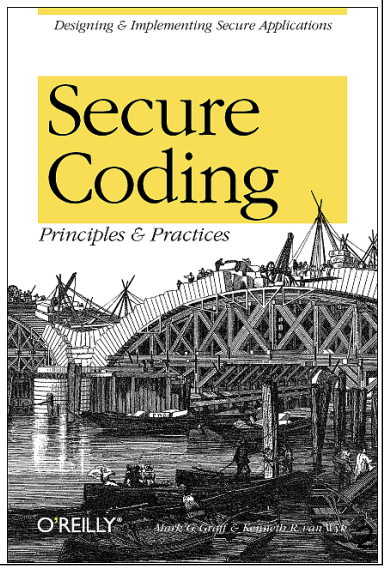
Work Experience
- 20+ years in Information Security
  - CMU CERT/CC Founder
  - DoD CERT
  - SAIC, Para-Protect
  - President and Founder, KRvW Associates, LLC

Security Work
- Technical lead on hundreds of commercial engagements since 1996, including
  - Application security assessments
  - Enterprise risk assessments
  - Secure network architecture
  - Security testing of enterprises and applications
- Author of two popular O'Reilly and Associates books
  - Incident Response: Planning and Management
  - Secure Coding: Principles and Practices

Credentials
- BS Lehigh University 1985
  - Mechanical Engineering

Personal Interests
- Travel, world cuisine, wine, mountain biking, zymurgy

Family (http://www.vanwyk.org/ken)
- Wife, two spectacularly spoiled basset hounds



Introductions

Please tell us a little about your

- Software dev background
- iOS and Xcode experience
- Mobile platform experience

Any specific questions you want answered in this class?

Understanding the problem

Just how bad is it, and why?

KRvW Associates, LLC

Mobile platforms

How secure are today's mobile platforms?

- Lots of similarities to web applications but...

Gold rush mentality

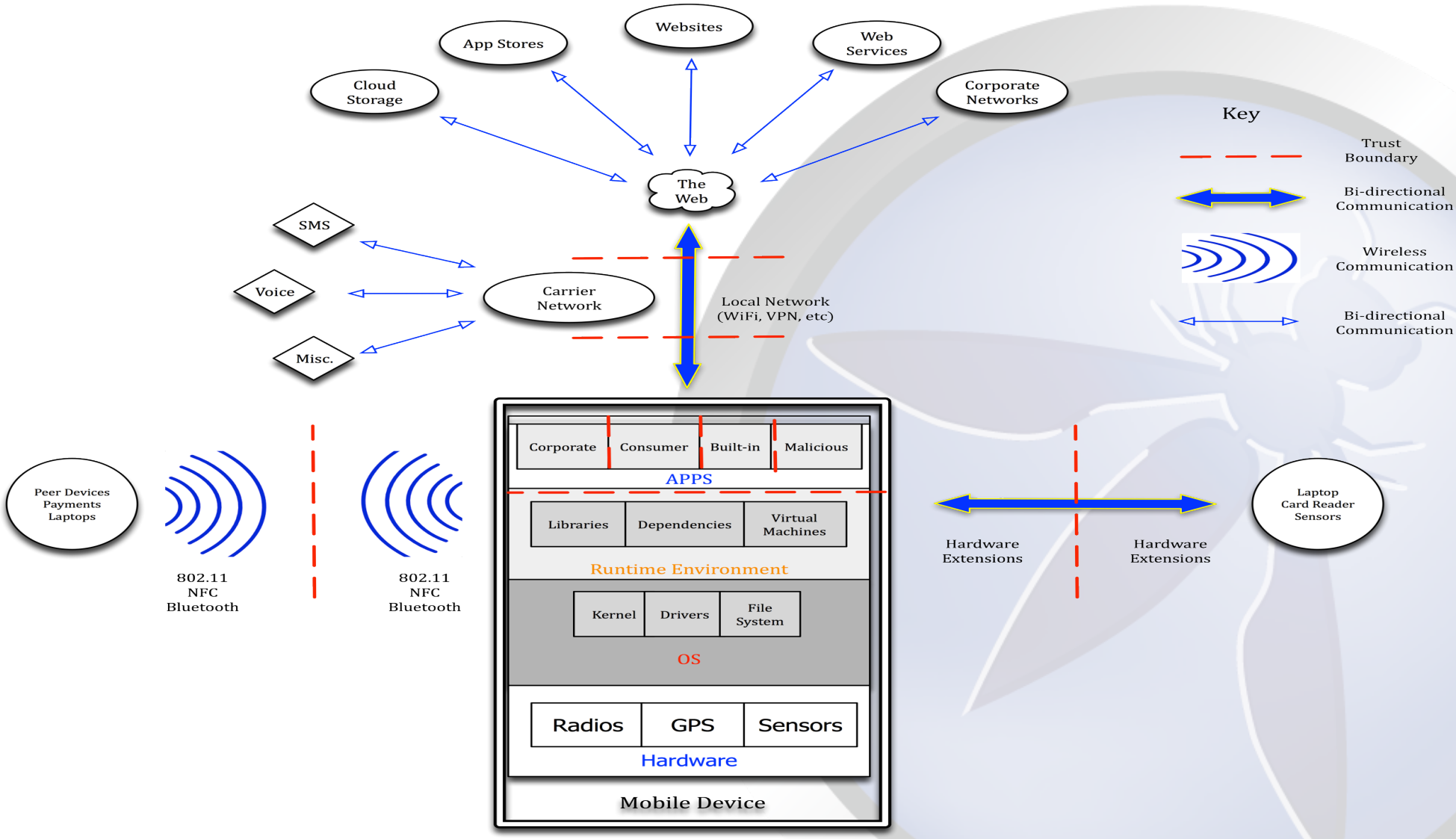
- Developers are on a death march to produce apps
- Unprecedented rate
- Security often suffers...

Mobile app threat model

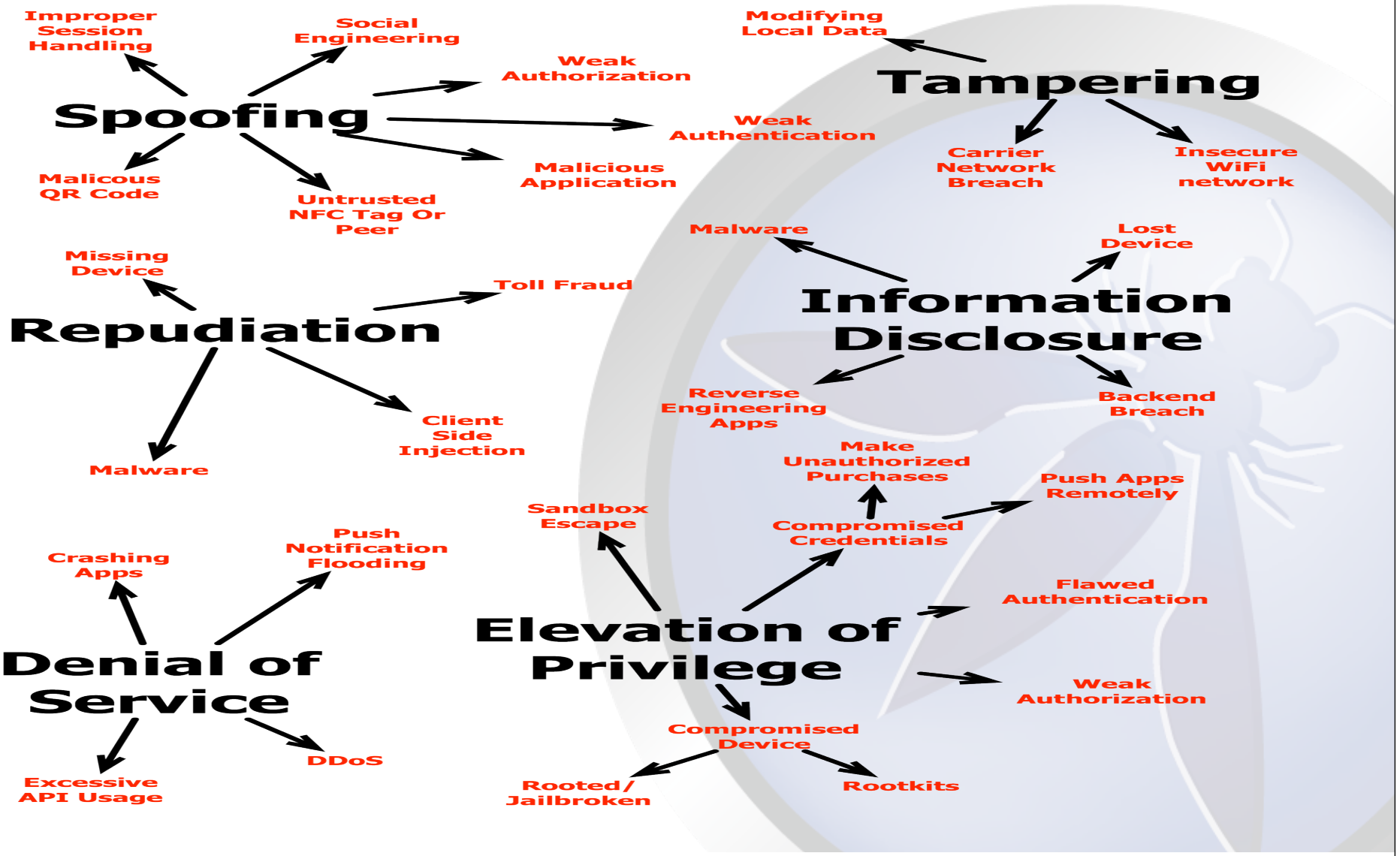
Many considerations

- Platforms vary substantially
- Similar but still very different than traditional web app--even when heavy with client-side code
- It's more than just apps
  - Cloud/network integration
  - Device platform considerations

Mobile Threat Model



Mobile Threat Model



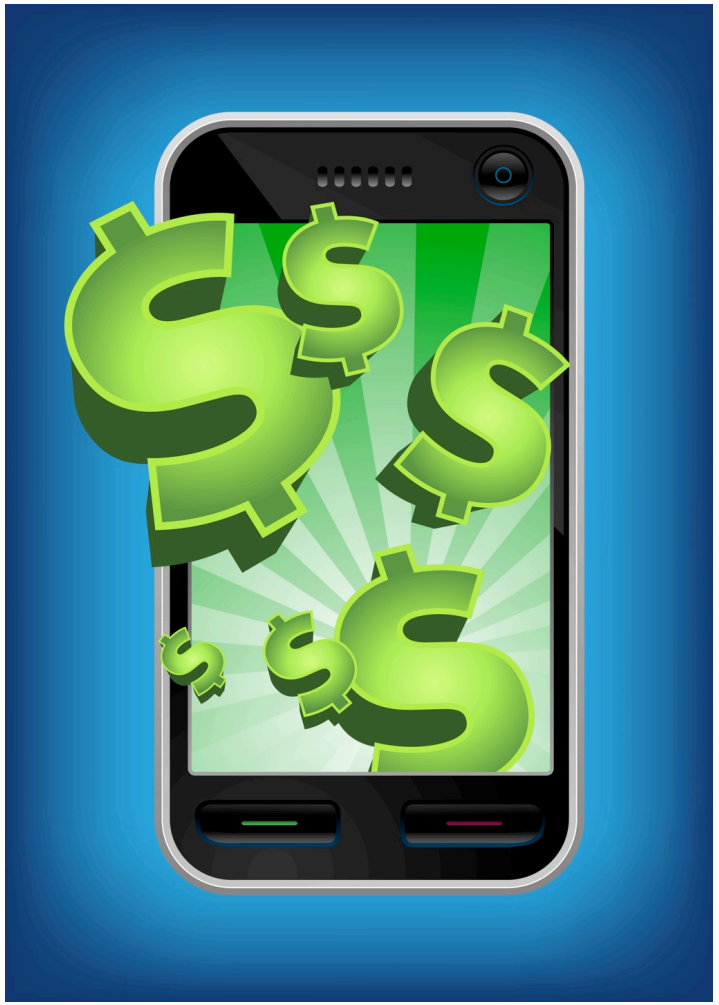
Biggest issue: lost/stolen device

Anyone with physical access to your device can get to a wealth of data

- PIN is not effective
- App data
- Keychains
- Properties

Disk encryption helps, but we can't count on users using it

See forensics results



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

Typical mobile app

Most mobile apps are basically web apps

- Clients issue web services request
  - SOAP or RESTful
- Servers respond with XML data stream

But with more client "smarts"

Almost all web weaknesses are relevant, and more

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

A lot to consider

That's a lot of mistakes to avoid (and there are more)

- What are the key differences between the web list and the mobile list?
- What assumptions must we then make in our apps?
- What assumptions are *unsafe*?

Security Principles and Pitfalls

Including hands-on exercises

KRvW Associates, LLC

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

Hands-on examples

- Topic discussion
- Hands-on examples to really understand
  - Optional, but recommended
- Instructor will demo as well

Some tools we'll be using

We'll also later use a couple others

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

Introducing OWASP's iGoat

A new OWASP project

- iGoat
- Developer tool for learning major security issues on iOS platform
- Inspired by OWASP's WebGoat tool for web apps

A word of warning on ethics

You will see, learn, and perform real attacks against a web and/or mobile application today

You may only do this on applications where you are authorized

Violating this is a breach of law in most countries

Do not do this on real apps without explicit authorization from the owner

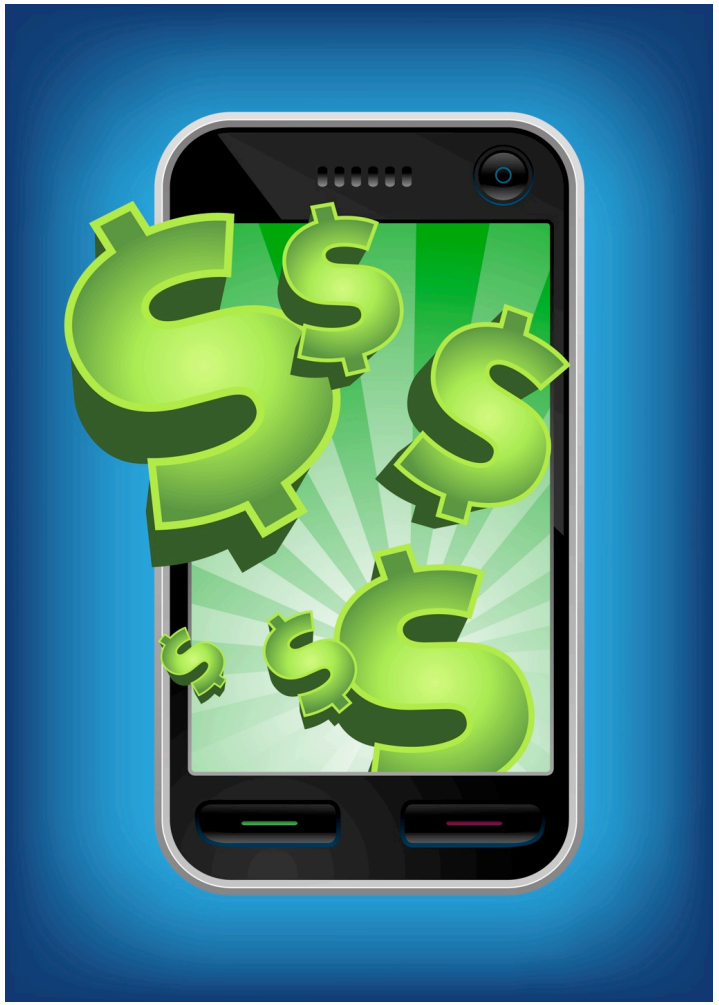
Attack vector: lost/stolen device

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



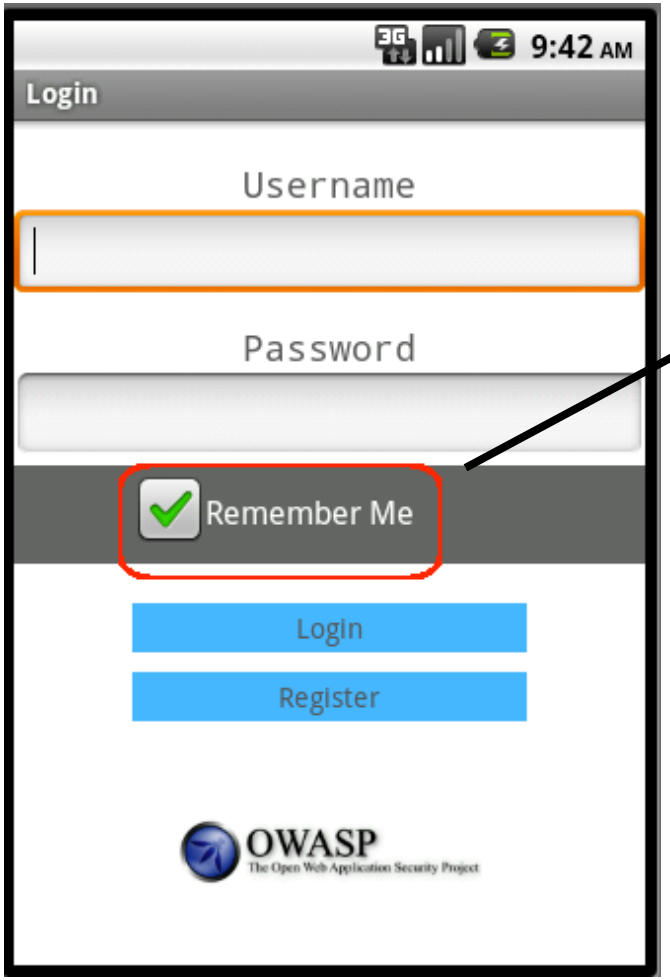
M1- Insecure Data Storage

- 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

M1- Insecure Data Storage




```
public void saveCredentials(String userName, String password) {

    SharedPreferences credentials = this.getSharedPreferences(
            "credentials", MODE_WORLD_READABLE);
    SharedPreferences.Editor editor = credentials.edit();
    editor.putString("username", userName);
    editor.putString("password", password);
    editor.putBoolean("remember", true);
    editor.commit();
}
```

— Very Bad

— Convenient!

M1- Insecure Data Storage
Prevention Tips

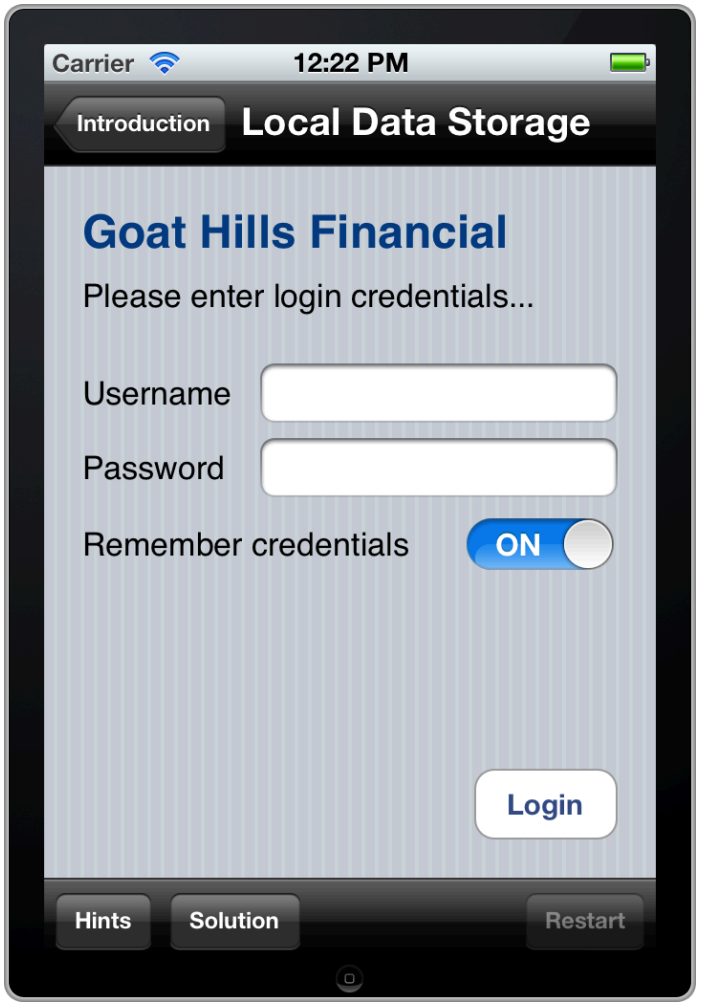
- Store ONLY what is absolutely required
- Never use public storage areas (ie- SD card)
- Leverage secure containers and platform provided file encryption APIs
- Do not grant files world readable or world writeable permissions

Control #	Description
1.1-1.14	Identify and protect sensitive data on the mobile device
2.1, 2.2, 2.5	Handle password credentials securely on the device

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



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?

Exercise - static analysis of an app

Explore folders

- ./Documents
- ./Library/Caches/*
- ./Library/Cookies
- ./Library/Preferences

App bundle

- Hexdump of binary
- plist file

What else?

Tools to use

- Mac tools
  - Finder
  - iPhone Explorer
  - 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

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

<u>At a bare minimum, your app *needs* to be able to withstand a coffee shop attack</u>

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

M3- Insufficient Transport Layer Protection Prevention Tips

- Ensure that all sensitive data leaving the device is encrypted
- This includes data over carrier networks, WiFi, and even NFC
- When security exceptions are thrown, it's generally for a reason…DO NOT ignore them!

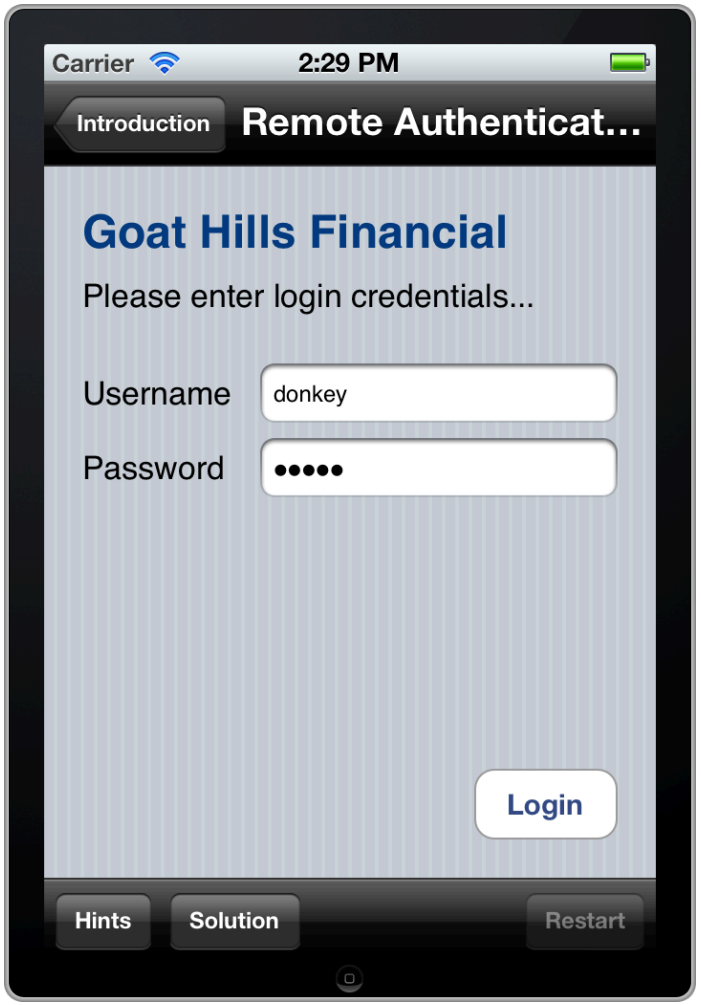
Control #	Description
3.1.3.6	Ensure sensitive data is protected in transit

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



Protecting users' secrets in transit

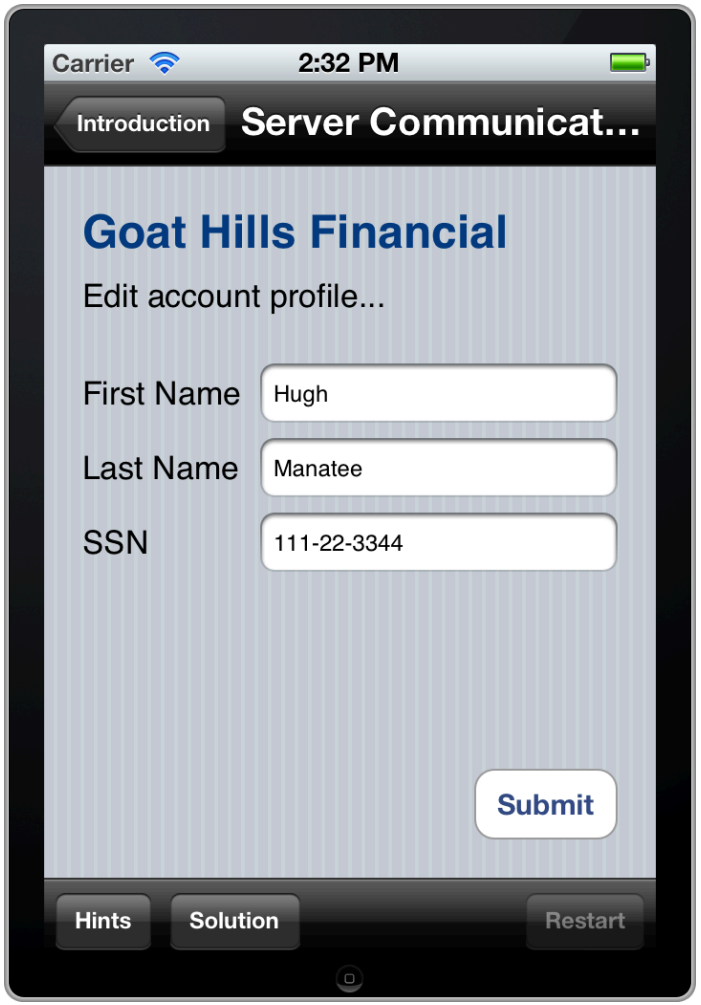
Always consider the coffee shop attack as lowest common denominator

We place a lot of faith in SSL

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

Passing secrets

In this simple example, we'll send customer data to a proxy server and intercept via a simulated coffee shop attack



How bad is it?

Neglecting SSL on network comms is common

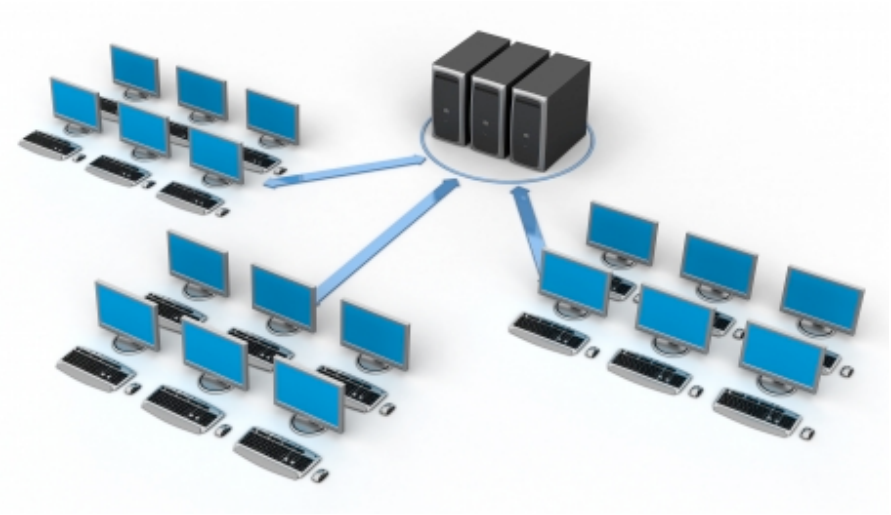
- Consider the exposures
  - Login credentials
  - Session credentials
  - Sensitive user data

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?

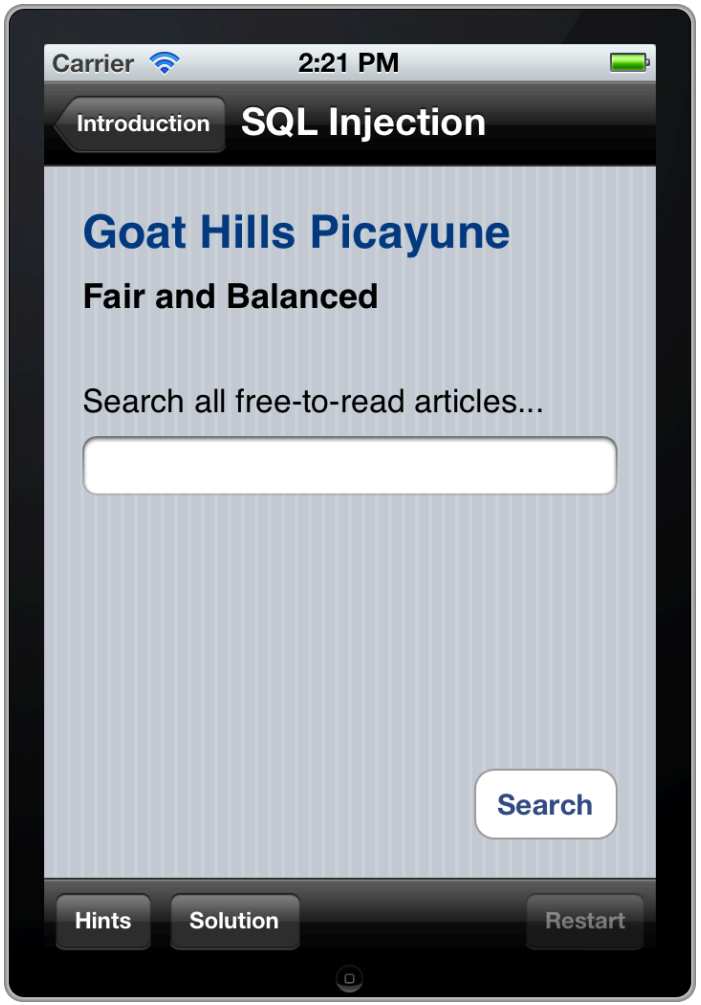
SQL injection exercise - client side

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?



Platform Architecture - iOS

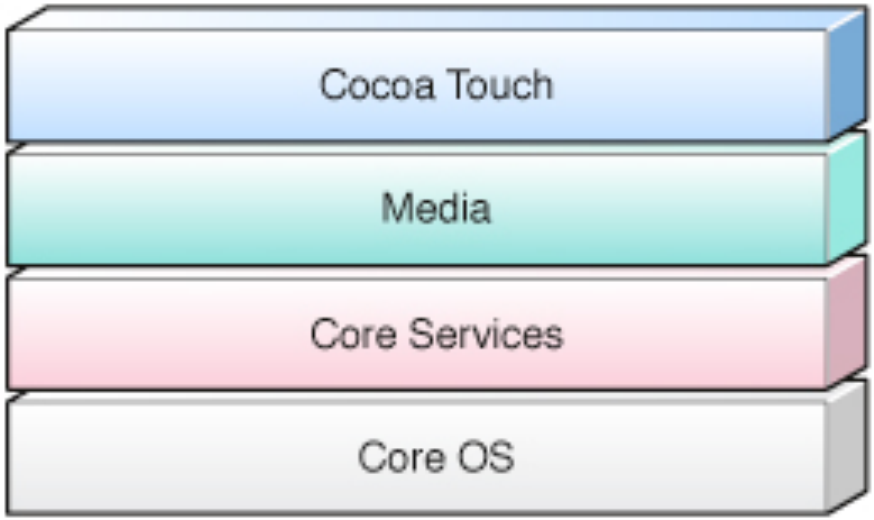
What the iOS / hardware platform offers us in the way of protection

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

- Application sandboxing
- App store protection
- Hardware encryption
- Keychains
- SSL and certificates

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
  - Apps are required to conform to Apple's rules
- 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"?

Hardware encryption

Each iOS device (as of 3S) has hardware crypto module

- Unique AES-256 key for every iOS device
- Sensitive data hardware encrypted

Sounds brilliant, right?

- Well...

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!
- Not so easy to implement, though...

And a few glitches...

- Keyboard data
- Screen snapshots
- Hardware encryption is flawed

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
- Because it looks pretty

WHAT?!

- It's a problem
- Requires local access to device, but still...

But the clincher

Hardware module protects unique key via device PIN

- PIN can trivially be disabled in many cases
- Jailbreak software

No more protection...

Note: Strong passcodes help

Discouraged?

If we build our apps using these protections only, we'll have problems

- But consider risk
- What is your app's "so what?" factor?
- What data are you protecting?
- From whom?
- Might be enough for some purposes

But for a serious enterprise...

The protections provided are simply not adequate to protect serious data

- Financial
- Privacy
- Credit cards

We need to further lock down

- But how much is enough?

Security Hardening

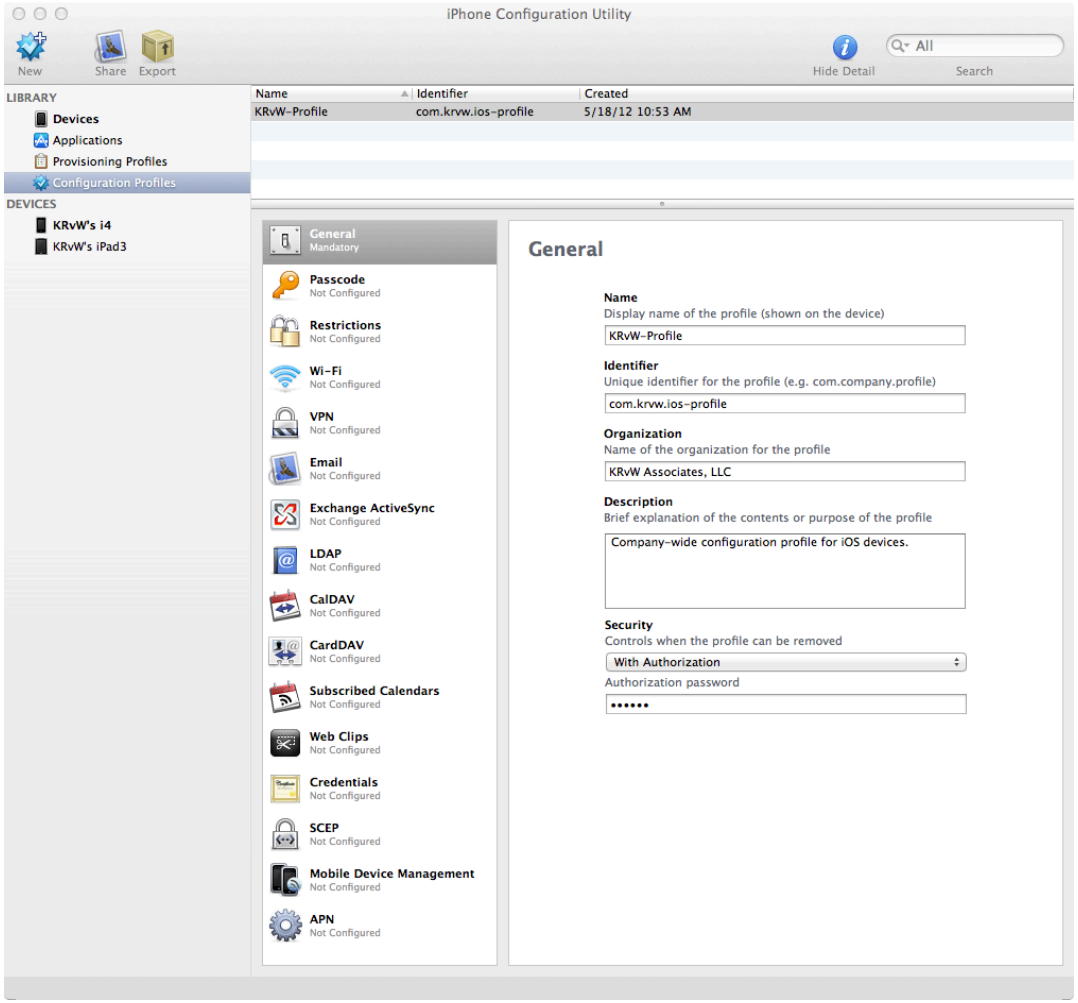
How can we increase the security profile of an iPad?

KRvW Associates, LLC

Configuration basics

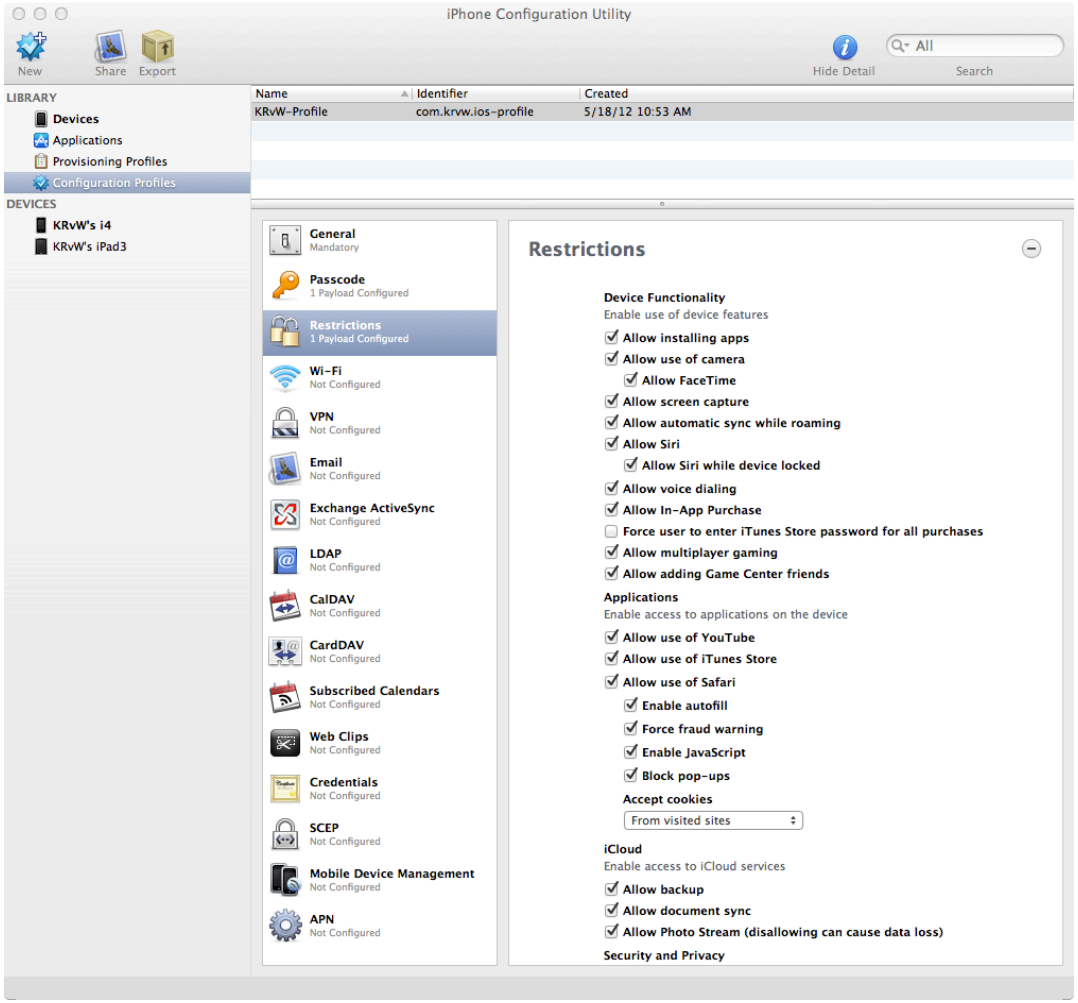
- Nomenclature
  - Configuration profiles
    - Configuration payloads
  - Provisioning profiles
  - Mobile device management
- Tools
  - Apple iPhone Configuration Utility
  - Dozens of commercial MDM products
    - Included with Lion Server too

iPCU - General settings

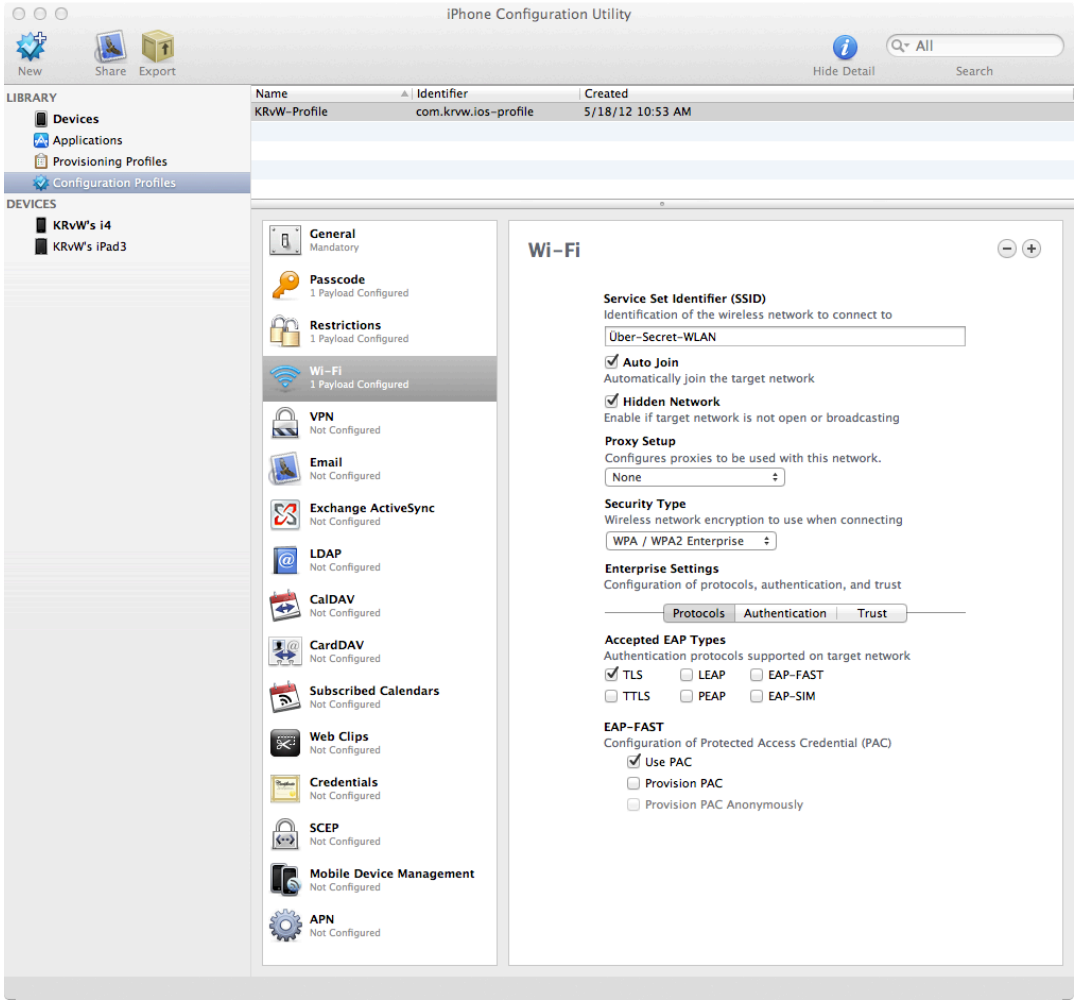


iPCU - Passcode settings

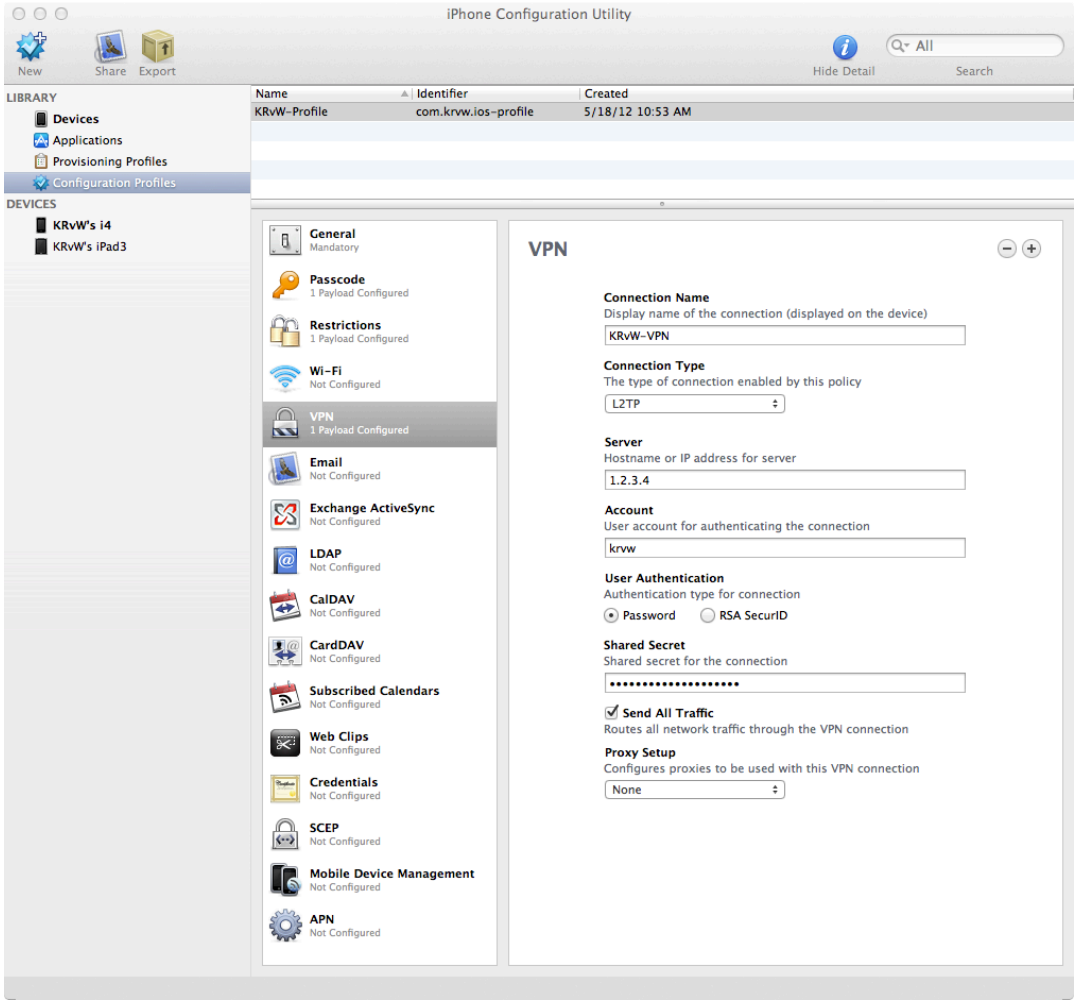
iPCU - Restriction settings



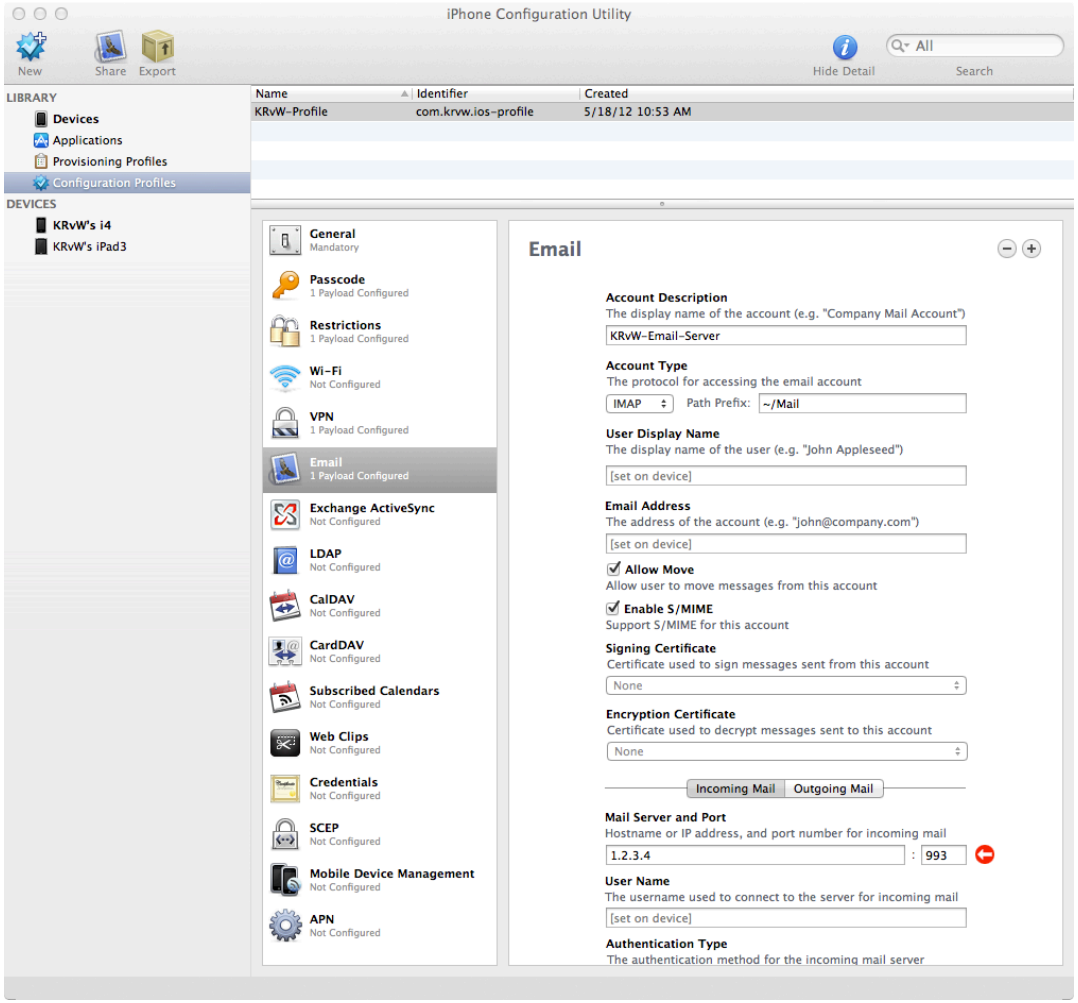
iPCU - Wi-Fi settings



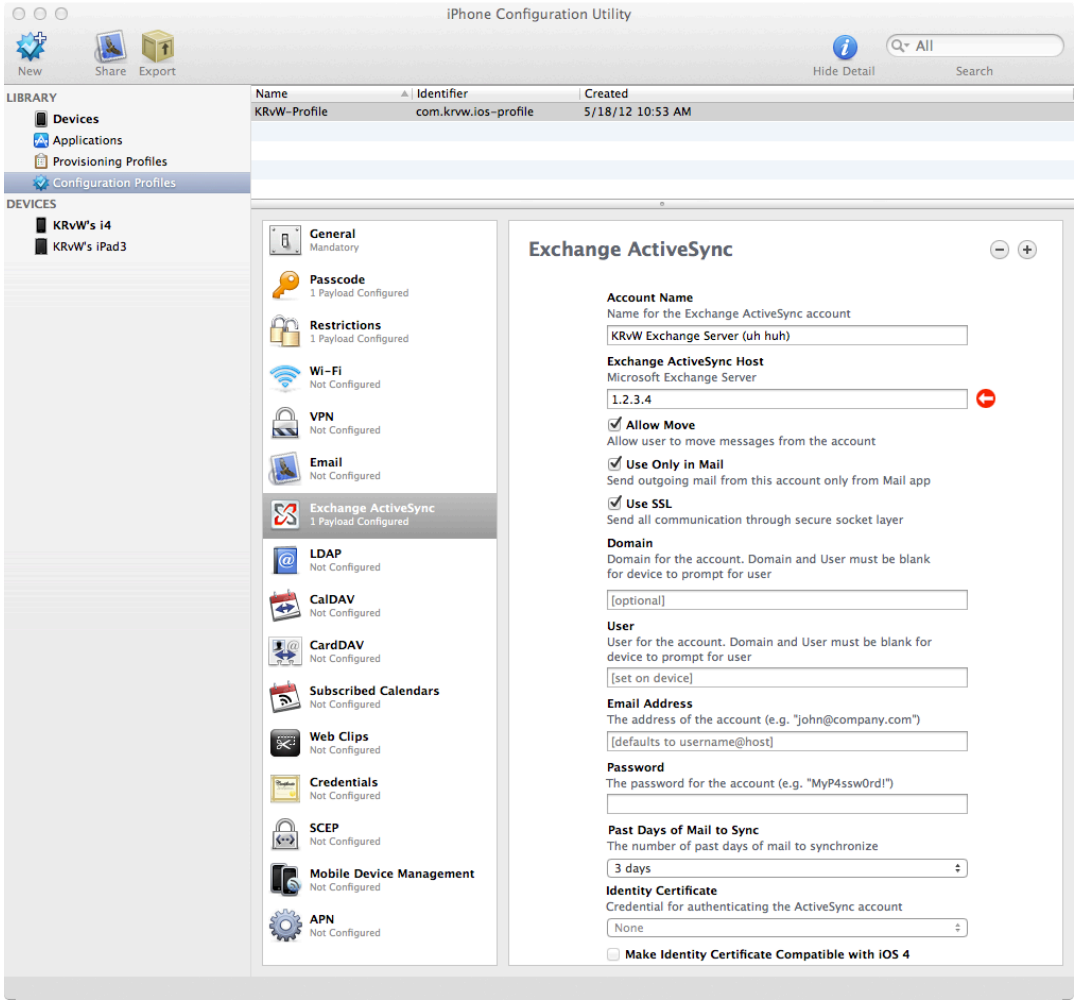
iPCU - VPN settings



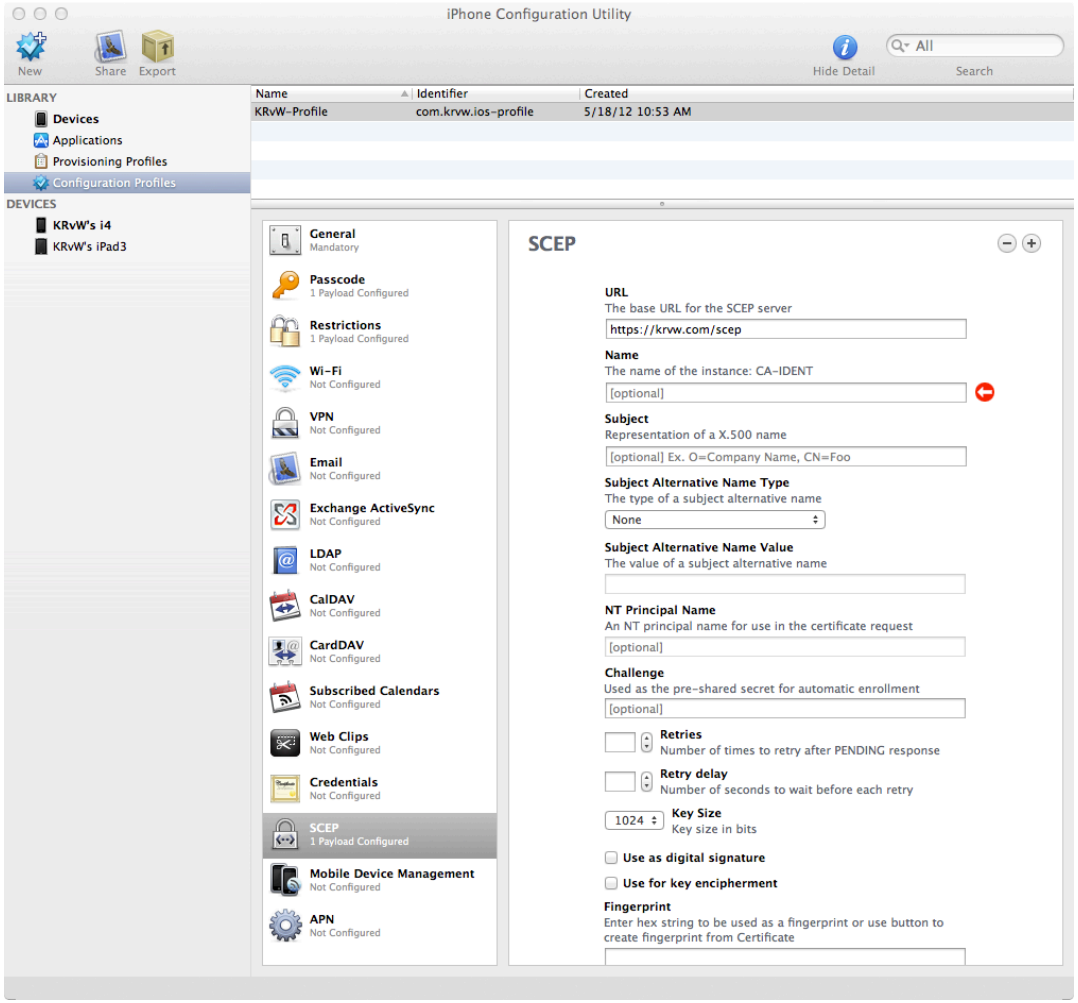
iPCU - Email settings



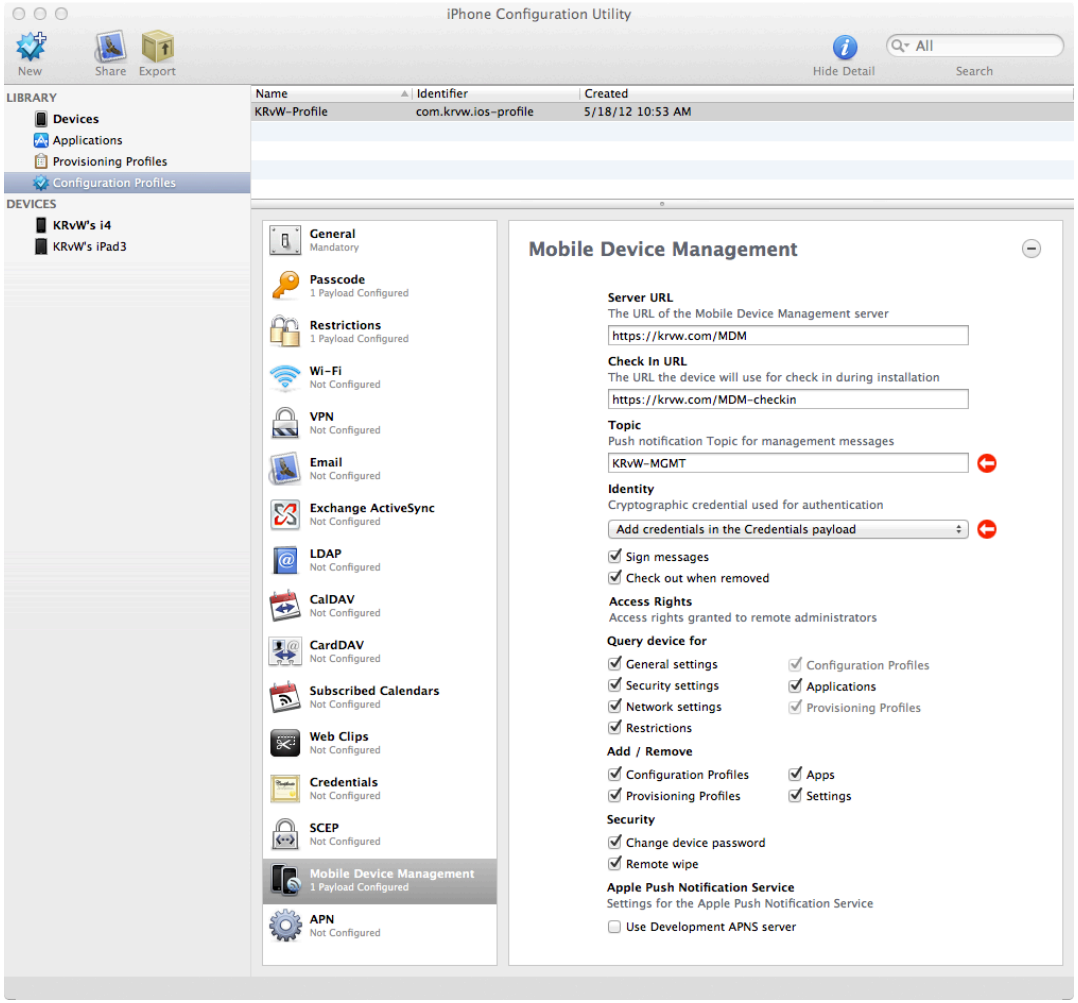
iPCU - Exchange ActiveSync



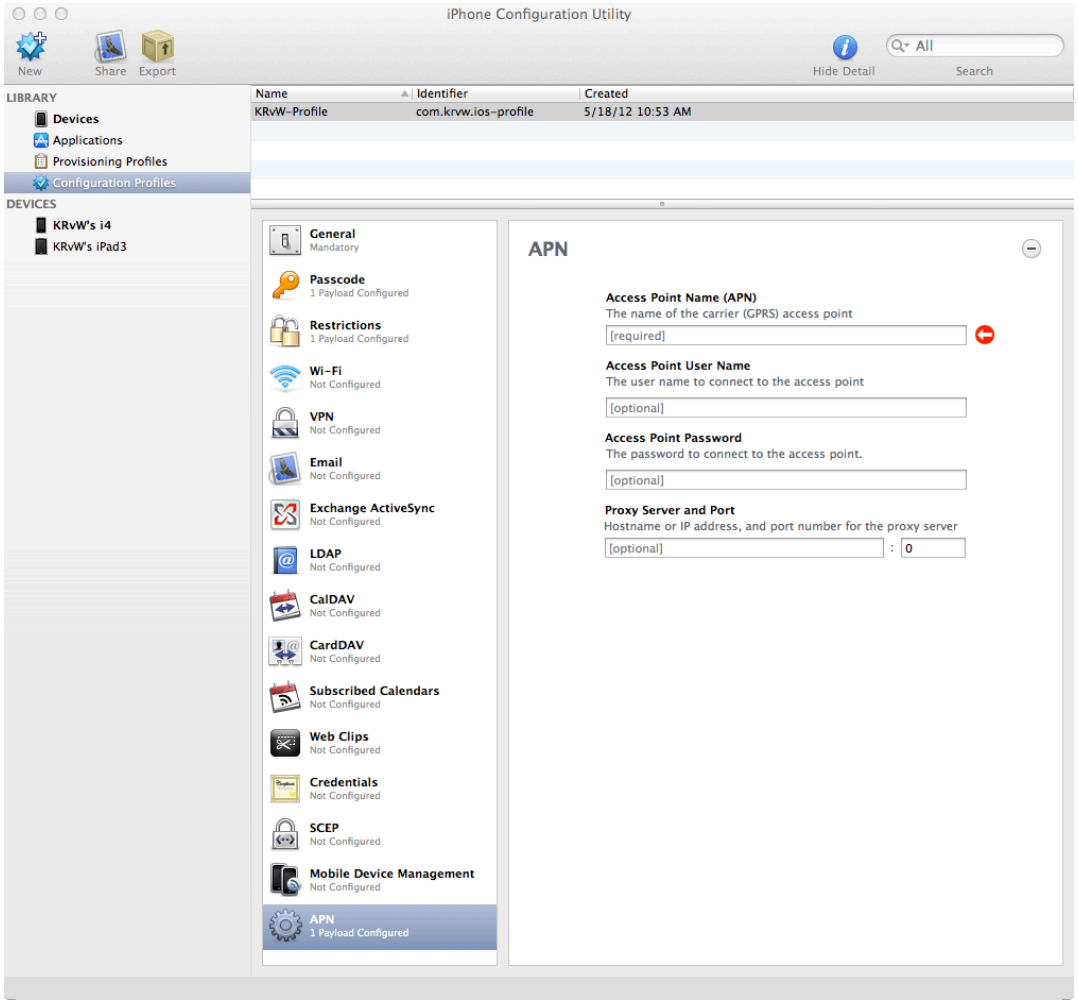
iPCU - SCEP certificate settings



iPCU - MDM settings



iPCU - APN settings



Lots of settings to remember...

Really aren't any shortcuts here

- Set up a test bed and step through all the settings
  - Test test test, and then test some more

Develop and test (!) one or more standard configurations

- Sensitivities
- Job roles

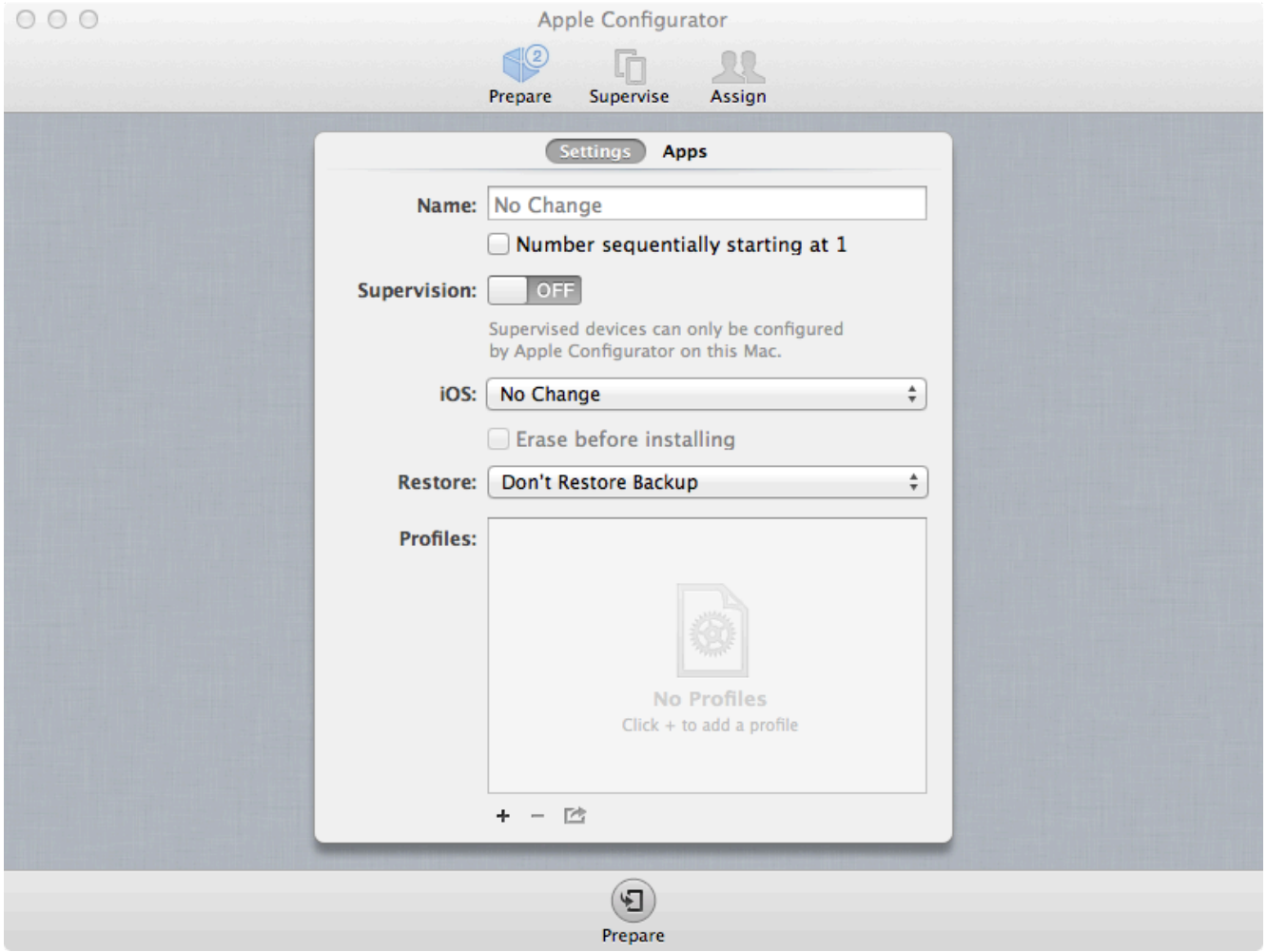
DSD guide is a good starting point

Deploying en masse

You've built your configs, now how do you manage them?

- One at a time via USB
- iPCU can export signed/ encrypted profiles
  - Email, web URL, etc.
  - What about updates?
- If you have a lot of devices to manage...

Apple Configurator



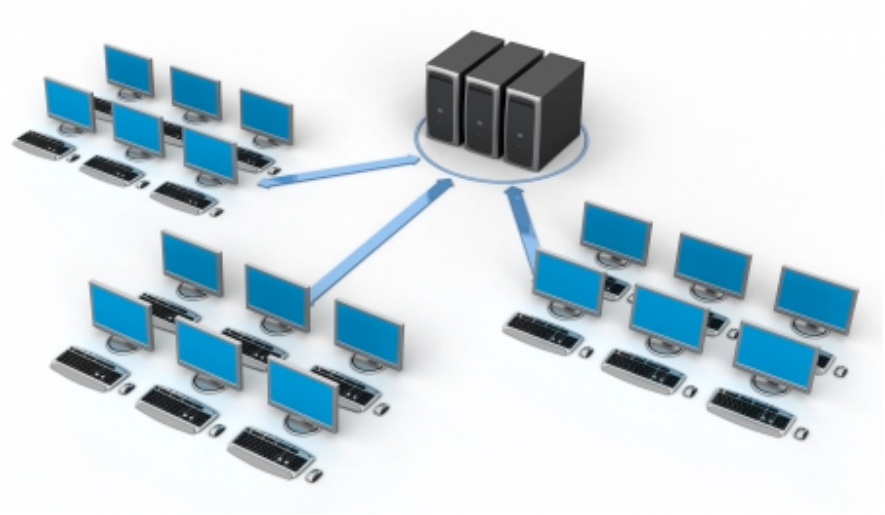
Consider an MDM product

Many to choose from, but

- Ability to manage hundreds of devices
- Update devices over the air
  - Uses push notifications (APNS)
- Can locate/wipe errant devices
  - If they're on the net...

Vendors

- See http://www.apple.com/ipad/business/integration/mdm/
- And now Apple Lion Server!



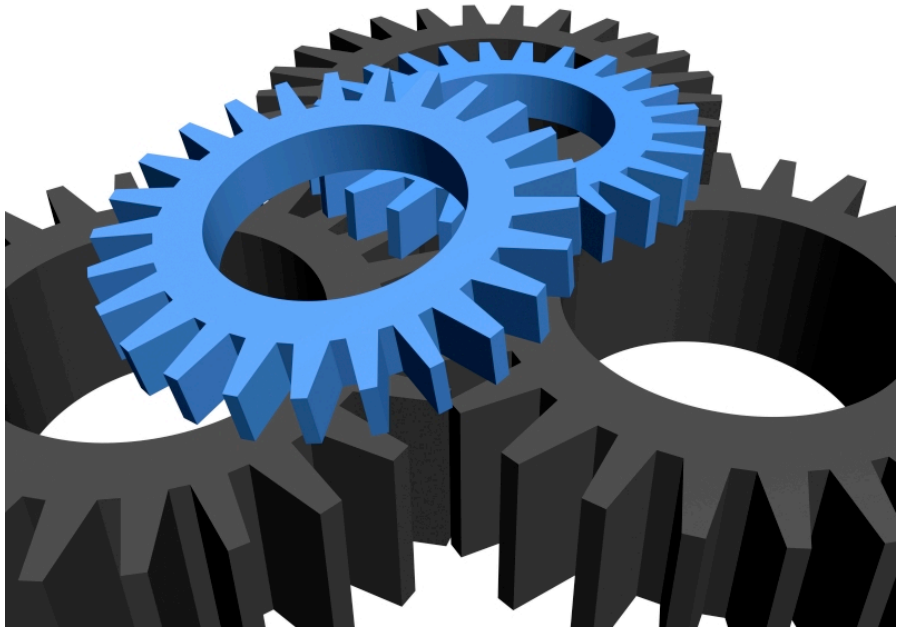
Internal app controls

What can developers do (briefly)

KRvW Associates, LLC

Common mechanisms

- Input validation
- Output escaping
- Authentication
- Session handling
- Protecting secrets
  - At rest
  - In transit
- SQL connections



Input validation

- Positive vs negative validation
  - Dangerous until proven safe
  - Don't just block the bad
- Consider the failures of desktop anti-virus tools
  - Signatures of known viruses

Input validation architecture

We have several choices

- Some good, some bad

Positive validation is our aim

Consider tiers of security in an enterprise app

- Tier 1: block the bad
- Tier 2: block and log
- Tier 3: block, log, and take evasive action to protect

Output encoding

Principle is to ensure data output does no harm in output context

- Output escaping of control chars
  - How do you drop a “<“ into an XML file?
- Consider all the possible output contexts

Authentication

This next example is for authenticating an app user to a server securely

- Server takes POST request, just like a web app

Mutual authentication

We may also want to use x.509 certificates and SSL to do strong mutual authentication

More complicated, but stronger

Certificate framework in NSURL is complex and tough to use

Session handling

- Normally controlled on the server for client-server apps
- Varies tremendously from one tech and app container to another
- Basic session rules apply
- Testing does help, though

Access control (authorization)

On the iOS device itself, apps have access to everything in their sandbox

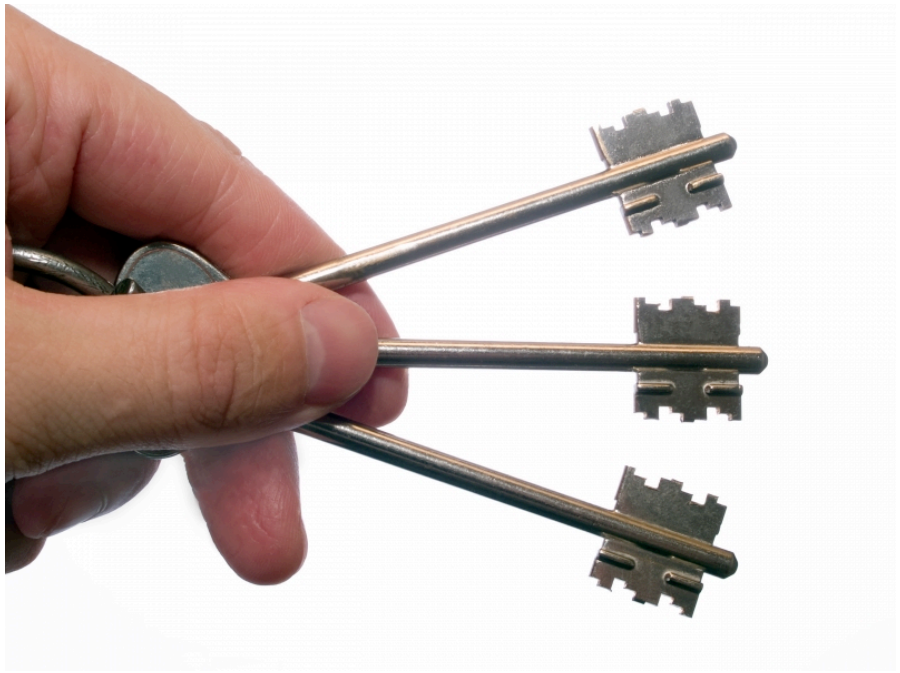
Server side must be designed and built in like any web app

Protecting secrets at rest

The biggest problem by far is key management

- How do you generate a strong key?
- Where do you store the key?
- What happens if the user loses his key?

Too strong and user support may be an issue



Built-in file protection (weak)

```
// API for writing to a file using writeToFile API

- (BOOL)writeToFile:(NSString *)path options:
(NSDataWritingOptions)mask error:(NSError
**)errorPtr

// To protect the file, include the
// NSDataWritingFileProtectionComplete option
```

Protecting secrets at rest (keychain)

```
// Write username/password combo to keychain.
BOOL writeSuccess = [SFHFKeychainUtils storeUsername:username
andPassword:password
    forServiceName:@"com.krvw.ios.KeychainStorage" updateExisting:YES
error:nil];
...

// Read password from keychain given username.
NSString *password = [SFHFKeychainUtils getPasswordForUsername:username
    andServiceName:@"com.krvw.ios.KeychainStorage" error:nil];
...

// Delete username/password combo from keychain.
BOOL deleteSuccess = [SFHFKeychainUtils deleteItemForUsername:username
    andServiceName:@"com.krvw.ios.KeychainStorage" error:nil];
...
```

Enter SQLcipher

Open source extension to SQLite

- Free
- Uses OpenSSL to AES-256 encrypt database
- Uses PBKDF2 for key expansion
- Generally accepted crypto standards

Available from

- http://sqlcipher.net

Protecting secrets at rest (SQLcipher)

```
sqlite3_stmt *compiledStmt;
// Unlock the database with the key (normally obtained via user input).
// This must be called before any other SQL operation.
sqlite3_exec(credentialsDB, "PRAGMA key = 'secretKey!'", NULL, NULL, NULL);
// Database now unlocked; perform normal SQLite queries/statments.
...
// Create creds database if it doesn't already exist.
const char *createStmt =
  "CREATE TABLE IF NOT EXISTS creds (id INTEGER PRIMARY KEY AUTOINCREMENT, username TEXT, password
TEXT)";
sqlite3_exec(credentialsDB, createStmt, NULL, NULL, NULL);
// Check to see if the user exists.
const char *queryStmt = "SELECT id FROM creds WHERE username=?";
int userID = -1;
if (sqlite3_prepare_v2(credentialsDB, queryStmt, -1, &compiledStmt, NULL) == SQLITE_OK) {
    sqlite3_bind_text(compiledStmt, 1, [username UTF8String], -1, SQLITE_TRANSIENT);
    while (sqlite3_step(compiledStmt) == SQLITE_ROW) {
        userID = sqlite3_column_int(compiledStmt, 0);
    }
}
if (userID >= 1) {
  // User exists in database.
  ...
}
```

Protecting secrets in transit

Key management still matters, but SSL largely takes care of that

- Basic SSL is pretty easy in NSURL
- Mutual certificates are stronger, but far more complicated
- NSURL is awkward, but it works
  - See previous example

SQL connections

Biggest security problem is using a mutable API

- Weak to SQL injection

Must use immutable API

- Similar to PreparedStatement in Java or C#

Other pitfalls

Format string issues from C

```
NSString outBuf = @”String to be appended”;
outBuf = [outBuf stringByAppendingFormat:[UtilityClass
  formatBuf: unformattedBuff.text]];

  vs.

NSString outBuf = @”String to be appended”;
outBuf = [outBuf stringByAppendingFormat:@”%@”,[UtilityClass
     formatBuf: unformattedBuff.text]];
```

Getting started

KRvW Associates, LLC

Where to begin?

Many of the issues you'll face are IT management "101" things

- Some advantages over what happened when PCs rolled in
  - Very little user interaction beyond UI
  - Modern management tools
  - Testbeds are cheap and easy

Enterprise suites

All-in-one enterprise comms suites

- Good For Enterprise
- All corporate data in one vault
  - Email
  - Calendars
  - etc
- Sound familiar?

Treat them like any other app and test thoroughly

Checklist

Here's a few things to start with

- Vet a list of approved apps
  - Static and dynamic analysis
- Develop and test config profiles
  - Use DSD guide for suggestions
  - Consider impact of policies
  - Test test test
- Pilot study
- For large orgs, test MDMs
  - Lion is probably cheapest
- Develop policies

Special cases and issues

Not everything fits cleanly into a box

- Privately owned iOS device
- User-purchased apps
- Personal emails
- Web browsing
- Public Wi-Fi hotspots

Choose carefully

- Putting whipped cream back in the can is tough

Kenneth R. van Wyk
KRvW Associates, LLC

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