Secure Development on iOS
Advice for developers and penetration testers

David Thiel

iSEC Partners
Outline

1. **Intro to iOS**
   - Basics

2. **Objective-C Primer**
   - Testing Setup

3. **Security-Relevant APIs**
   - TLS and Networking
   - Data Storage
   - The Keychain
   - Backgrounding

4. **IPC**
   - App URLs
   - Copy/Paste

5. **Common Attack Scenarios**
   - Platform-Specific Quirks

6. **Secure coding checklist**
I consult for iSEC.

My perspective is that of a penetration tester (not developer)

Info here is ideally of use to both testers and developers

Assumes little to no iOS knowledge

Focus is app security, not OS security

Takeaways: be able audit your own or others’ iOS apps
Intro to iPad

Padtastic
Intro to iOS

It’s an OS, but with an i

- High-level API, “Cocoa Touch”
- Development in XCode
  - So yes, you need a Mac
- iOS Simulator (not emulator)
  - Compiles iOS apps to native code to run locally
- Applications written primarily in Objective-C
Objective-C

How to spot it from a very long way away

- C + Smalltalk...ish
- Uses “infix” notation:
  - `[Object messagePassedToObject:argument];`
- It is not to everyone’s tastes
- But I have very refined tastes
@interface Classname : NSParentObject {
  SomeType aThing;  // instance variables
}
+(type)classMethod:(vartype)myVariable;
-(type)instanceMethod:(vartype)myVariable;
@end

These go in .h files, and define the structure of objects (like C structs).
Ok, 2 slides
Alternative interface declaration

#import "NSParentClass.h"

@interface Classname : NSParentClass {
    @public NSURL *blorg;
    @private NSString *gurgle;
}

@property (readonly) NSURL *blorg;
@property (copy) NSString *gurgle;

This is the “2.0” way to declare interfaces.
3, whatever
Infix and dot notation

```objective-c
@implementation Classname
@synthesize blorg; // generates set/get methods
@synthesize gurgle;

Instance *myInstance = [[Instance alloc] init];

[myInstance setGurgle:@"eep"]; // infix notation
myInstance.gurgle = @"eep"; // dot notation
```

This is the “implementation”, stored in .m files. “Synthesize” creates getter/setter methods for properties.
Memory Management
Retain/Release

- No garbage collection in iOS
- Must track with “retain” and “release” methods

```objective-c
Classname *myClass = [[Classname alloc] init]; // Retain count: 1
// Can be shortened to
// [Classname new];
...
[myClass release];
```
XCode

@implementation ReachabilityAppDelegate

-(void)configureTextField: (UITextField*) textField imageView: (UIImage*) imageView {
    NetworkStatus netStatus = [curReach currentReachabilityStatus];
    BOOL connectionRequired= [curReach connectionRequired];
    NSString* statusString = @"";
    switch (netStatus) {
    case NotReachable: {
        statusString = @"Access Not Available";
        imageView.image = [UIImage imageNamed: @"stop-32.png"];
        //Minor interface detail- connectionRequired may return yes, connectionRequired= NO;
        break;
    }
Testing Setup
Intercepting secure communications

- Standard proxy intercept won’t work
- Cert errors are a hard failure
- Options:
  - Change source to use HTTP
  - Use device + cert for proxy
  - Use simulator with → proxy → real site
Stunnel config

; SSL client mode
client = yes

; service-level configuration

[https]
accept  = 127.0.0.1:80
connect = 10.10.1.50:443
TIMEOUTclose = 0
Proxy Config

To add a new listener, complete the relevant details and click “add”.

- **Local listener port**: 8080
- **Listen on loopback interface only**
- **Support invisible proxying for non-proxy-aware clients**
- **Redirect to host**: 127.0.0.1
- **Redirect to port**: 80

**Server SSL certificate**:
- Use a self-signed certificate
- Generate CA-signed per-host certificates
- Generate a CA-signed certificate with a specific hostname:
Executing Unsigned Code
Or executing signed code without checking signature

On jailbroken device:
- tar local app bundle
- scp to root@dev.ice.i.p
- SSH to device, untar bundle into Applications
- Restart Springboard (or reboot)
The Sandbox Mechanism

- aka “Seatbelt”
- Based upon TrustedBSD MAC framework
- Unlike Android’s UID-based segregation, apps run as one user
- Seatbelt policies provide needed segregation. Probably.
- Policy file can be found on the device in /usr/share/sandbox/SandboxTemplate.sb
The Sandbox Mechanism

Jailbreaking

- On jailbroken devices, sandbox no longer applies
- However, devs for sideloaded apps can voluntarily hop into one
- Documented profiles for OSX:

  kSBXProfileNoNetwork (= "nonet")
  kSBXProfileNoInternet (= "nointernet")
  kSBXProfilePureComputation (= "pure-computation")
  kSBXProfileNoWriteExceptTemporary (= "write-tmp-only")
  kSBXProfileNoWrite (= "nowrite")

---

1 http://iphonedevwiki.net/index.php/Seatbelt

David Thiel (iSEC Partners)
Binary Analysis
XCode & Clang

- Useful for black box testing or self-testing
- Disassembly of Mach-O binary format quite clean
- Several useful tools: otool, otx, class-dump
- Use for reversing other applications, or finding what info would be available to a third party
- Obfuscation is generally pretty futile, but especially in ObjC
Binary Analysis

otool

```bash
otool -toV /Applications/iCal.app/Contents/MacOS/iCal/Applications/iCal.app/
                     Contents/MacOS/iCal

Objective-C segment
Module 0x22b52c

... Class Definitions
defs[0] 0x00204360

    isa 0x0020a560
    super_class 0x001a5f44 CALCanvasItem
    name 0x001c6574 CALCanvasAttributedString

    ... ivars 0x00224300
    ivar_count 13
    ivar_name 0x001a54e2 _text
    ivar_type 0x001a53d0 @”NSMutableAttributedString”
    ivar_offset 0x0000012c
    ivar_name 0x001a54e8
```

David Thiel (iSEC Partners)
Binary Analysis

http://otx.osxninja.com/

-(BOOL)[NSString(NSStringExtras) isFeedURLString]
+0 00003488 55 pushl %ebp
+1 00003489 89e5 movl %esp,%ebp
+3 0000348b 53 pushl %ebx
+4 0000348c 83ec14 subl $0x14,%esp
+7 0000348f 8b5d08 movl 0x08(%ebp),%ebx
+10 00003492 c744240844430700 movl $0x00074344,0x08(%esp)

feed:
+18 0000349a a180a00700 movl 0x0007a080,%eax

_web_hasCaseInsensitivePrefix:
+23 0000349f 89442404 movl %eax,0x04(%esp)
+27 000034a3 891c24 movl %ebx,(%esp)
+30 000034a6 e850420800 calll 0x000876fb
-[(%esp,1) _web_hasCaseInsensitivePrefix:]
Binary Analysis

class-dump

http://iphone.freecoder.org/classdump_en.html

class-dump-x /Developer/Platforms/iPhoneSimulator.platform/Developer/SDKs/
  iPhoneSimulator3.0.sdk/Applications/MobileSafari.app
  < snip >
  @protocol CALCanvasTextProtocol
  - (id)attributes;
  - (id)foregroundColor;
  - (float)fontSize;
  @end

  @protocol CALDetachmentDelegate
  - (int) decideDetachmentFor:(id)fp8 withOccurrence:(id)fp12;
  @end
Static Analysis

- Clang analyzer merged into XCode
- “Build & Analyze” option
- Identifies memory leakage, use-after-free, etc.
- Note: in some recent XCode versions, Analyzer results only show for device SDK builds. Meh
### Static Analysis

#### Output

```objective-c
NetworkStatus netStatus = [curReach currentReachabilityStatus];
BOOL connectionRequired = [curReach connectionRequired];
NSString* statusString = @"";
switch (netStatus)
{
  case NotReachable:
    statusString = @"Access Not Available";
    imageView.image = [UIImage imageNamed: @"stop-32.png"];
    // Minor interface detail—connectionRequired may return yes, even when the host is unreachable
    [myString release];
    [myString release];
    break;
  case ReachableViaWWAN:
    statusString = @"Reachable WWAN";
    imageView.image = [UIImage imageNamed: @"WWAN5.png"];
    break;
  case ReachableViaWiFi:
    statusString = @"Reachable WiFi";
    imageView.image = [UIImage imageNamed: @"Airport.png"];
    break;
}
```
Examining local storage

- Examine local store on OSX:
  - `/Library/Application Support/iPhone Simulator/Applications/(appID)`

- Hitlist:
  - plist files
  - Cookies
  - SQL databases
  - Preferences
  - Cache data
  - Keyboard cache
Keyboard Caching

- Partial keyboard cache used with form autocompletion
- Already disabled for password fields
- Should be disabled for any potentially sensitive fields
- Set `UITextField` property `autocorrectionType` = `UITextAutocorrectionNo`
Anatomy of an App

./Documents → properties, logs
./Library/Caches → cachey things
./Library/Caches/Snapshots → screenshots of your app
./Library/Cookies → cookie plists
./Library/Preferences → various preference plists
./Appname.app → app resources: binary, graphics, nibs, Info.plist
./tmp → tmp
Networking
TLS and NSURL Handling

- Standard method for working with URLs
- SSL/TLS handled properly! Bypassing failed verification not allowed by default.
- So, of course, people turn it off
Networking
TLS and NSURL Handling

- Check for `NSURLRequest` verification bypass via `setAllowsAnyHTTPSCertificate`
- SSL verification bypass via `NSURLConnection` delegation
  - Search for `continueWithoutCredentialForAuthenticationChallenge`²
- Extra bonus stupid: Define category method to slip by Apple’s private API checks³

³ http://stackoverflow.com/questions/2001565/alternative-method-for-nsurlrequests-private-setallowsanyhttpscertificateforho
Networking
NSStreams

- Good for non-HTTP traffic or going slightly lower-level

```swift
// First we define the host to be contacted
NSHost *myhost = [NSHost hostWithName:@"www.conglomco.com"];

// Then we create
[NSStream getStreamsToHost:myhost
    port:443
    inputStream:&MyInputStream
    outputStream:&MyOutputStream];

[MyInputStream setProperty:NSStreamSocketSecurityLevelTLSv1 // Note
    forKey:NSStreamSocketSecurityLevelKey];
```
Networking

CFStreams

- Slightly lower-level still
- Security defined by `kCFStreamPropertySSLSettings`
- Has sad set of constants 🙁

```c
const CFStringRef kCFStreamSSLLevel;
const CFStringRef kCFStreamSSLAllowsExpiredCertificates;
const CFStringRef kCFStreamSSLAllowsExpiredRoots;
const CFStringRef kCFStreamSSLAllowsAnyRoot;
const CFStringRef kCFStreamSSLValidatesCertificateChain;
const CFStringRef kCFStreamSSLPeerName;
```
Local Data Storage
The Various Mechanisms

A few ways data is stored (and potentially exposed):

- SQLite
- Core Data
  - Internally, SQLite
- Cookie management
- Caches
- plists
Cookies
Probably gluten-free

- Manipulated by the URL loading system
- Can alter cookieAcceptPolicy to:
  - NSHTTPCookieAcceptPolicyNever
  - NSHTTPCookieAcceptPolicyOnlyFromMainDocumentDomain

- Note that this may affect other running applications
  - In OS X, cookies and cookie policy are shared among apps
  - In iOS, only cookie policy is shared
SQLite and SQL injection
Dynamic SQL

```objective-c
NSString *uid = [myHTTPConnection getUID];
NSString *statement = [NSString stringWithFormat:@"SELECT username FROM users
    where uid = '%@'", uid];
const char *sql = [statement UTF8String];
```
SQLite and SQL injection

Parameterized SQL

```c
const char *sql = "SELECT username FROM users where uid = ?";
sqlite3_prepare_v2(db, sql, -1, &selectUid, NULL);
sqlite3_bind_int(selectUid, 1, uid);
int status = sqlite3_step(selectUid);
```
Security-Relevant APIs

Data Storage

Caching

- HTTP & HTTPS requests cached by default
- Can be prevented by delegating `NSURLConnection`

```objectivec
-(NSCachedURLResponse *)connection:(NSURLConnection *)connection
    willCacheResponse:(NSCachedURLResponse *)cachedResponse
{
    NSCachedURLResponse *newCachedResponse = cachedResponse;
    if ([[[[cachedResponse response] URL] scheme] isEqual:@"https"])
    {
        newCachedResponse = nil;
    }
    return newCachedResponse;
}
```

David Thiel (iSEC Partners)
Geolocation

Best Practices

- Use least degree of accuracy necessary
- If you don’t want to handle subpoenas from divorce lawyers:
  - Don’t log locally
  - Anonymize server-side data
  - Prune logs
Geolocation
Accuracy Settings

Several accuracy constants:

extern const CLLocationAccuracy kCLLocationAccuracyBestForNavigation;
extern const CLLocationAccuracy kCLLocationAccuracyBest;
extern const CLLocationAccuracy kCLLocationAccuracyNearestTenMeters;
extern const CLLocationAccuracy kCLLocationAccuracyHundredMeters;
extern const CLLocationAccuracy kCLLocationAccuracyKilometer;
extern const CLLocationAccuracy kCLLocationAccuracyThreeKilometers;
The Keychain

- Keychain is where secret stuff goes
  - Argh! Do not store this data in Preferences!

- Encrypted with device-specific key
  - Apps can’t read, not included in backups

- Simpler API than OS X: `SecItemAdd`, `SecItemUpdate`, `SecItemCopyMatching`

- Not available in simulator
  - This changes a bit with 4.0. I’ll update this soon.
The Keychain

Shared keychains

- For using the same keychain among different apps
- Used by setting `kSecAttrAccessGroup` on init
- Apps must have same `keychain-access-groups`
- Apps can only have one access group
- On jailbroken phone...all bets off

---

4http://useyourloaf.com/blog/2010/4/3/keychain-group-access.html
The Keychain

Certificates

- On device, can be installed via e-mail, Safari or iTunes sync
- On simulator, no such luck
- Certs still verified, but no way to install new ones
  - Since they’re stored in the Keychain
- Stubs necessary for detecting simulator vs. device
Data Protection
Improving file and keychain protection

- By default, data encrypted with “hardware” key
- In iOS 4, “hardware” key can be encrypted with PIN/password
- Developers can also mark files as “protected”
- Files encrypted, unreadable while device is locked
Data Protection
Usage

- 2 methods for enabling
- Pass `NSDataWritingFileProtectionComplete` to `writeToFile` method of `NSData` object
- Set `NSFileProtectionKey` to `NSFileProtectionComplete` on `NSFileManager` object
- Again, data not accessible when device is locked
  - Check for data availability before use

Entropy
How does it work?

- Using Cocoa, not /dev/random
- Gathered via `SecRandomCopyBytes`
  - Again, does not work in simulator
- Obviously, `rand()`, `random()`, `arc4random()` are all dealbreakers

```c
int result = SecRandomCopyBytes(kSecRandomDefault, sizeof(int), (uint8_t*)&randomResult);
```
**Backgrounding**

Initiating Background Tasks

- **Probably most security-relevant API in iOS 4.0**
- **Use** `beginBackgroundTaskWithExpirationHandler` **method to initiate background tasks**
  - **Needs matching** `endBackgroundTask` **method**
- **Remaining task time stored in** `backgroundTimeRemaining` **property**
Backgrounding

Concerns

- Note: app is snapshotted upon backgrounding
- Prior to this, application should remove any sensitive data from view
- Or, prevent backgrounding with UIApplicationExitsOnSuspend
Backgrounding

State Transitions

- Detect state transitions
- Key state transition methods:

```objective-c
application:didFinishLaunchingWithOptions:
applicationDidBecomeActive:
applicationWillResignActive:
applicationDidEnterBackground:
applicationWillEnterForeground:
applicationWillTerminate:
```
Apps can register their own URL handlers — added by editing the plist, usually from XCode

Called just like any URL, with multiple parameters, e.g.

```swift
openURL: [NSURL URLWithString:@"myapp://?foo=urb&blerg=gah"];  
```

Can be called by app or web page

Params accessible to receiving app via

```swift
- (BOOL)application:(UIApplication *)application handleOpenURL:(NSURL *)url
```

Obviously, sanitization is key here, especially given...
What happens if two apps use the same handler?

- If an Apple app uses it: Apple app launches
- Third-party apps: “Undefined”

“If your URL type includes a scheme that is identical to one defined by Apple, the Apple-provided application that handles a URL with that scheme (for example, “mailto”) is launched instead of your application. If a URL type registered by your application includes a scheme that conflicts with a scheme registered by another third-party application, the application that launches for a URL with that scheme is undefined.”

- May go to the last claiming app...ew.
- Hence: be wary of passing private data in app URLs
Copy/Paste
Pasteboards

- Obligatory dig at Apple re: copy/paste debacle
- 2 system UIPasteboard access methods:
  - UIPasteboardNameGeneral & UIPasteboardNameFind
Also “private” application pasteboards, which (in true Objective-C form) are not in any way “private”

Occasionally used as IPC hack

- Migrating data from free → paid app
- I saw one suggestion to transfer private keys with the pasteboard

Bottom line: avoid sensitive data here & clean up after yourself

- Clear pasteboard on `applicationWillTerminate`
- `pasteBoard.items = nil`
Copy/Paste
Disabling it

- Possible mitigation: For fields with sensitive data, disable copy/paste menu

```c
-(BOOL)canPerformAction:(SEL)action withSender:(id)sender {
    UIMenuController *menuController = [UIMenuController sharedMenuController];
    if (menuController) {
        [UIMenuController sharedMenuController].menuVisible = NO;
    }
    return NO;
}
```

- Can also disable menu items individually

```
```

How not to pasteboard: Twitter OAuth library

```swift
- (void) pasteboardChanged: (NSNotification *) note {
    UIPasteboard *pb = [UIPasteboard generalPasteboard];

    if ([note.userInfo objectForKey:UIPasteboardChangedTypesAddedKey] == nil)
        return;

    NSString *copied = pb.string;

    if (copied.length != 7 || !copied.oauth_twitter_isNumeric) return;

    [self gotPin:copied];
}
```

73rd-party library, not by Twitter
Classic C Attacks
Nothing new here

- Still has the same classic issues
- Buffer overflows
- Integer issues, especially with malloc()
  - Why are you malloc’ing? We are in the future here
- Double-frees
- Format strings
Object use after release

- Exploitable! Under some circumstances.[2]
- Procedure:
  - Release object
  - Release some other object
  - Allocate space of same size as first object
  - Write your code to the new buffer
  - ...
  - Send message or release to original object
Classic format string attack

In its simplest form

Don’t code like this.

```c
int main()
{
    char foo[512];
    gets(foo); // Whatever, it’s just an example
    printf(foo);
}
```

What’s in foo? `%x` to read straight from memory, or `%n` for memory corruption (and maybe code execution)
iOS & Format Strings

- withFormat/appendinFormat family
- %x works – %n does not 😞
- %n does still work with regular C code...
Format Strings

Format string confusion

- Found on pentest:

```objective-c
NSString  myStuff = @"Here is my stuff.";
myStuff = [myStuff stringByAppendingFormat:[UtilityClass formatStuff:
    unformattedStuff.text]];
```

- Bzzt. NSString objects aren’t magically safe.

```objective-c
NSString  myStuff = @"Here is my stuff.";
myStuff = [myStuff stringByAppendingFormat:@"%@", [UtilityClass
    formatStuff:unformattedStuff.text]];
```
Format Strings

Likely culprits

- `[NSString *WithFormat]`
- `[NSString stringByAppendingFormat]`
- `[NSMutableString appendFormat]`
- `[NSAlert alertWithMessageText]`
- `[NSException]`
- `[NSLog]`
Secure coding checklist
Or penetration tester’s hit list

- HTTPS used and correctly configured (i.e. not bypassed by delegation or `setAllowsAnyHTTPSCertificate`)
- All format strings properly declared
- General C issues (`malloc`, `str`*)
  - Any third-party C/C++ code is suspect
- Entropy gathered correctly
- Secure backgrounding
Secure coding checklist
Continued

- UIPasteBoards not leaking sensitive data
- Objects correctly released
- URL handler parameters sanitized
- Secure keychain usage
- No inappropriate data stored on local filesystem
- CFStream, NSStream, NSURL inputs sanitized/encoded
QUESTIONS?
HTTPS://WWW.ISECPARTNERS.COM
For Further Reading I

H. Dwivedi, C. Clark, D. Thiel
*Mobile Application Security.*
McGraw Hill, 2010

Neil Archibald
STOP!!! Objective-C Run-TIME.

Apple, Inc.
iOS Application Programming Guide
For Further Reading II

Other resources

http://culater.net/wiki/moin.cgi/CocoaReverseEngineering
http://www.musicalgeometry.com/archives/872