



Evaluating iOS Applications

Manchester OWASP

Feb 2012



Introduction

Company and Speaker Overview



- I'm a co-founder & director of MDSec 1999
 - Apple fanboy?
 - CVE-2011-0204: Apple ImageIO TIFF Heap Overflow 2004
 - CVE-2011-0194: Apple ImageIO TIFF Image Integer Overflow
 - CVE-2010-1845: Apple ImageIO PSD Memory Corruption
 - Perspective is that of a **Pen tester**, not a developer 2007
 - MDSec:
 - Web App Hacker's Handbook 1st & 2nd Edition
 - Worldwide training
 - Online training
 - Burp Suite 2011
- 2013



Evaluating iOS Applications

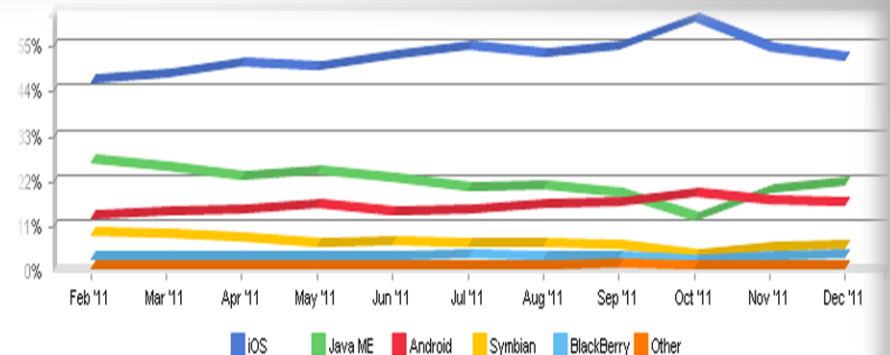
Overview

- Introduction
- Overview of iOS & Apps
- Blackbox Assessment
- Transport Security
- Data Storage
- Keychain
- Protocol Handlers
- UIWebViews
- Injection Attacks
- Filesystem Interaction
- Geolocation
- Logging
- Memory Corruption



Why iOS Apps?

- Apple have a 52% market share [1]
- Over half a million apps in App Store



Mobile Security

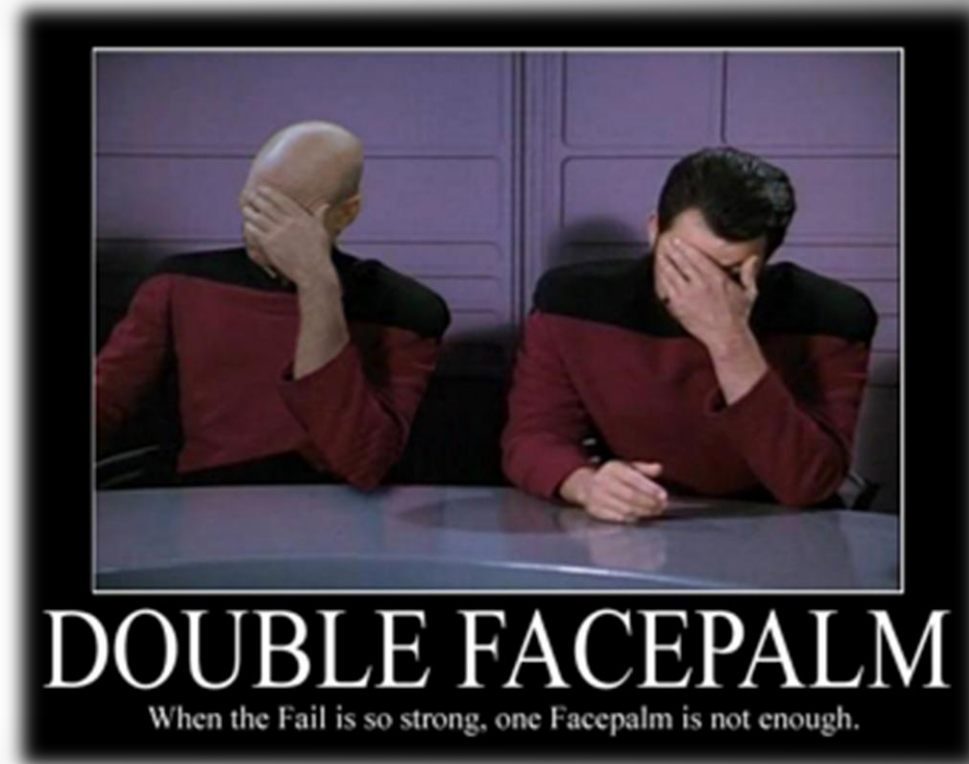
- In focus over last few years
- Steady increase in requests for mobile app assessments
- Public app problems:
 - Citigroup data storage
 - Skype XSS & Protocol Handler vulnerabilities
- Often hold personal data
 - Online banking, social networking etc...

<http://www.netmarketshare.com/operating-system-market-share.aspx?qprid=9&qpcustomb=1>

Overview

Why Mobile Security?

“In a letter, the US banking giant said the Citi Mobile app saved user information in a hidden file that could be used by attackers to gain unauthorized access to online accounts. Personal information stored in the file could include account numbers, bill payments and security access codes...”.



http://www.theregister.co.uk/2010/07/27/citi_iphone_app_weakness/

- Code Signing
 - Prevents unauthorised apps running
 - Validates app signatures at runtime
- Sandboxing
 - Apps run in a self-contained environment
 - Third party apps assigned “container” seatbelt profile
 - Allows some access to address book, media & outbound network
- ASLR
 - Randomises where data & code is mapped in an address space
 - Apps can have partial or full ASLR (compiled with PIE)
- Encryption
 - Hardware based encryption; “data is encrypted at rest”
 - Provides Data Protection API for protecting individual items



Overview

iOS Apps

- Developed in Objective C
 - Superset of C
- Xcode for development
 - I can haz Apple?

[Object method:argument]

```
NSAutoreleasePool *pool = [[NSAutoreleasePool alloc] init];  
NSLog (@"Hello, World!");  
[pool drain];
```



- Previous work:
 - “Auditing iPhone and iPad Applications” by Ilja van Sprundel
 - “Secure Development on iOS” by David Thiel
 - “Apple iOS 4 Security Evaluation” by Dino Dai Zovi

Blackbox Assessment

Intercepting Communications

- Configure the device for a proxy
- Install a self-signed certificate on the device to capture HTTPS



<http://carnal0wnage.attackresearch.com/2010/11/iphone-burp.html>

Blackbox Assessment

Position Independent Executable



- Use a jailbroken phone to SSH to the device and extract the app
- Otool is your friend
 - With PIE:

```
test-iphone:/Applications/MobileSafari.app root# otool -hv MobileSafari
MobileSafari:
Mach header
      magic cputype cpusubtype  caps   filetype ncmds sizeofcmds      flags
      MH_MAGIC   ARM           9   0x00   EXECUTE   43      5004   NOUNDEFS DYLDLINK TWOLEVEL PIE
test-iphone:/Applications/MobileSafari.app root#
```

- Without PIE:

```
test-iphone:/var/mobile/Applications/16B00BFC-A271-4C63-B603-8A33A135F4C4/Facebook.app root# ls -al Facebook
-rwxr-xr-x 1 mobile mobile 6218848 Jan 19 12:17 Facebook*
test-iphone:/var/mobile/Applications/16B00BFC-A271-4C63-B603-8A33A135F4C4/Facebook.app root# otool -hv Facebook
Facebook:
Mach header
      magic cputype cpusubtype  caps   filetype ncmds sizeofcmds      flags
      MH_MAGIC   ARM           V6   0x00   EXECUTE   41      4716   NOUNDEFS DYLDLINK TWOLEVEL
test-iphone:/var/mobile/Applications/16B00BFC-A271-4C63-B603-8A33A135F4C4/Facebook.app root#
```

Blackbox Assessment

Reverse Engineering



- Apps are stored as an IPA in iTunes Library
 - IPA is just ZIP
- App Store binaries are encrypted
 - Manual decryption
 - Use debugger, breakpoint EP, let loader decrypt, dump decrypted image
 - <http://dvlabs.tippingpoint.com/blog/2009/03/06/reverse-engineering-iphone-appstore-binaries>
 - Automated
 - Crackulous & AppCrack

Transport Security

Introduction



- Mobile devices may often use untrusted networks
 - Imperative that data is sent securely
- Apple provides a couple of ways to do HTTPS
 - NSURLConnection
 - CFNetwork
- Developers sometimes pass on to third party code
 - CyaSSL
 - Matrix SSL
 - OpenSSL

- Different TLS handshake depending on SDK
- Version 4.3 of SDK uses TLS 1.0 with 29 suites, some weak:

- TLS_RSA_WITH_DES_CBC_SHA
- TLS_RSA_EXPORT_WITH_RC4_MD5
- TLS_RSA_EXPORT_WITH_DES40_CBC_SHA
- TLS_DHE_RSA_WITH_DES_CBC_SHA
- TLS_DHE_RSA_EXPORT_WITH_DES40_CBC_SHA

▼ Cipher Suites (29 suites)

```
Cipher Suite: TLS_ECDHE_ECDSA_WITH_AES_256_CBC_SHA (0xc00a)
Cipher Suite: TLS_ECDHE_ECDSA_WITH_AES_128_CBC_SHA (0xc009)
Cipher Suite: TLS_ECDHE_ECDSA_WITH_RC4_128_SHA (0xc007)
Cipher Suite: TLS_ECDHE_ECDSA_WITH_3DES_EDE_CBC_SHA (0xc008)
Cipher Suite: TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA (0xc013)
Cipher Suite: TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA (0xc014)
Cipher Suite: TLS_ECDHE_RSA_WITH_RC4_128_SHA (0xc011)
Cipher Suite: TLS_ECDHE_RSA_WITH_3DES_EDE_CBC_SHA (0xc012)
Cipher Suite: TLS_ECDH_ECDSA_WITH_AES_128_CBC_SHA (0xc004)
Cipher Suite: TLS_ECDH_ECDSA_WITH_AES_256_CBC_SHA (0xc005)
Cipher Suite: TLS_ECDH_ECDSA_WITH_RC4_128_SHA (0xc002)
Cipher Suite: TLS_ECDH_ECDSA_WITH_3DES_EDE_CBC_SHA (0xc003)
Cipher Suite: TLS_ECDH_RSA_WITH_AES_128_CBC_SHA (0xc00e)
Cipher Suite: TLS_ECDH_RSA_WITH_AES_256_CBC_SHA (0xc00f)
Cipher Suite: TLS_ECDH_RSA_WITH_RC4_128_SHA (0xc00c)
Cipher Suite: TLS_ECDH_RSA_WITH_3DES_EDE_CBC_SHA (0xc00d)
Cipher Suite: TLS_RSA_WITH_AES_128_CBC_SHA (0x002f)
Cipher Suite: TLS_RSA_WITH_RC4_128_SHA (0x0005)
Cipher Suite: TLS_RSA_WITH_RC4_128_MD5 (0x0004)
Cipher Suite: TLS_RSA_WITH_AES_256_CBC_SHA (0x0035)
Cipher Suite: TLS_RSA_WITH_3DES_EDE_CBC_SHA (0x000a)
Cipher Suite: TLS_RSA_WITH_DES_CBC_SHA (0x0009)
Cipher Suite: TLS_RSA_EXPORT_WITH_RC4_40_MD5 (0x0003)
Cipher Suite: TLS_RSA_EXPORT_WITH_DES40_CBC_SHA (0x0008)
Cipher Suite: TLS_DHE_RSA_WITH_AES_128_CBC_SHA (0x0033)
Cipher Suite: TLS_DHE_RSA_WITH_AES_256_CBC_SHA (0x0039)
Cipher Suite: TLS_DHE_RSA_WITH_3DES_EDE_CBC_SHA (0x0016)
Cipher Suite: TLS_DHE_RSA_WITH_DES_CBC_SHA (0x0015)
Cipher Suite: TLS_DHE_RSA_EXPORT_WITH_DES40_CBC_SHA (0x0014)
```

- Version 5.0 uses TLS 1.2 with 37 suites, none weak
- API provides no way to configure cipher suites AFAIK

- Developers often allow self-signed certs
- NSURLRequest:
 - Default behaviour is to reject cert and throw *NSURLErrorDomain*
 - Developers override with *allowsAnyHTTPSCertificateForHost*
 - Private delegate method
- NSURLConnection:
 - Alternate approach using *didReceiveAuthenticationChallenge* delegate
 - Ignore cert using *continueWithoutCredentialForAuthenticationChallenge* selector

Transport Security

CFNetwork

- Alternate implementation
 - More granular than NSURLConnection
- Developers have more control over certs
 - Allow expired certs:
 - *kCFStreamSSLAllowsExpiredCertificates*
 - Allow expired roots:
 - *kCFStreamSSLAllowsExpiredRoots*
 - Allow any root:
 - *kCFStreamSSLAllowsAnyRoot*
 - No validation at all????
 - *kCFStreamSSLValidatesCertificateChain*



Data Storage

Introduction

- Mobile apps can often hold sensitive data
 - High risk of device being lost or stolen
 - Imperative data is protected in these scenarios

- Client-side data takes a number of forms
 - Custom created documents
 - Logs
 - Cookie stores
 - Plists
 - Data caches
 - Databases



- Stored in `/var/mobile/Applications/<GUID>`

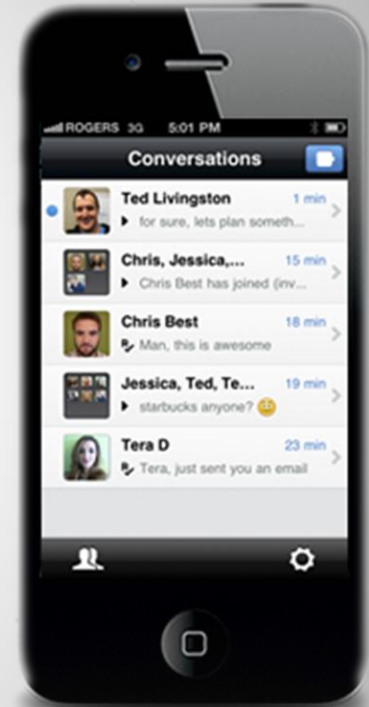
- Apple API for using the hardware crypto
- Encrypted using a key derived from passcode
- Developers must “mark” files to protect
- 4 levels of protection
 - No protection:
 - *NSDataWritingFileProtectionNone* / *NSFileProtectionNone*
 - Complete protection:
 - *NSDataWritingFileProtectionComplete* / *NSFileProtectionComplete*
 - Complete unless open:
 - *NSDataWritingFileProtectionCompleteUnlessOpen* / *NSFileProtectionCompleteUnlessOpen*
 - Complete until first authentication:
 - *NSDataWritingFileProtectionCompleteUntilFirstUserAuthentication* / *NSFileProtectionCompleteUntilFirstUserAuthentication*

Data Storage

Real World Example



- Kik Messenger
 - Send IM through data
 - Over 1 million users
 - Users sign up for a Kik account
 - <http://kik.com/>

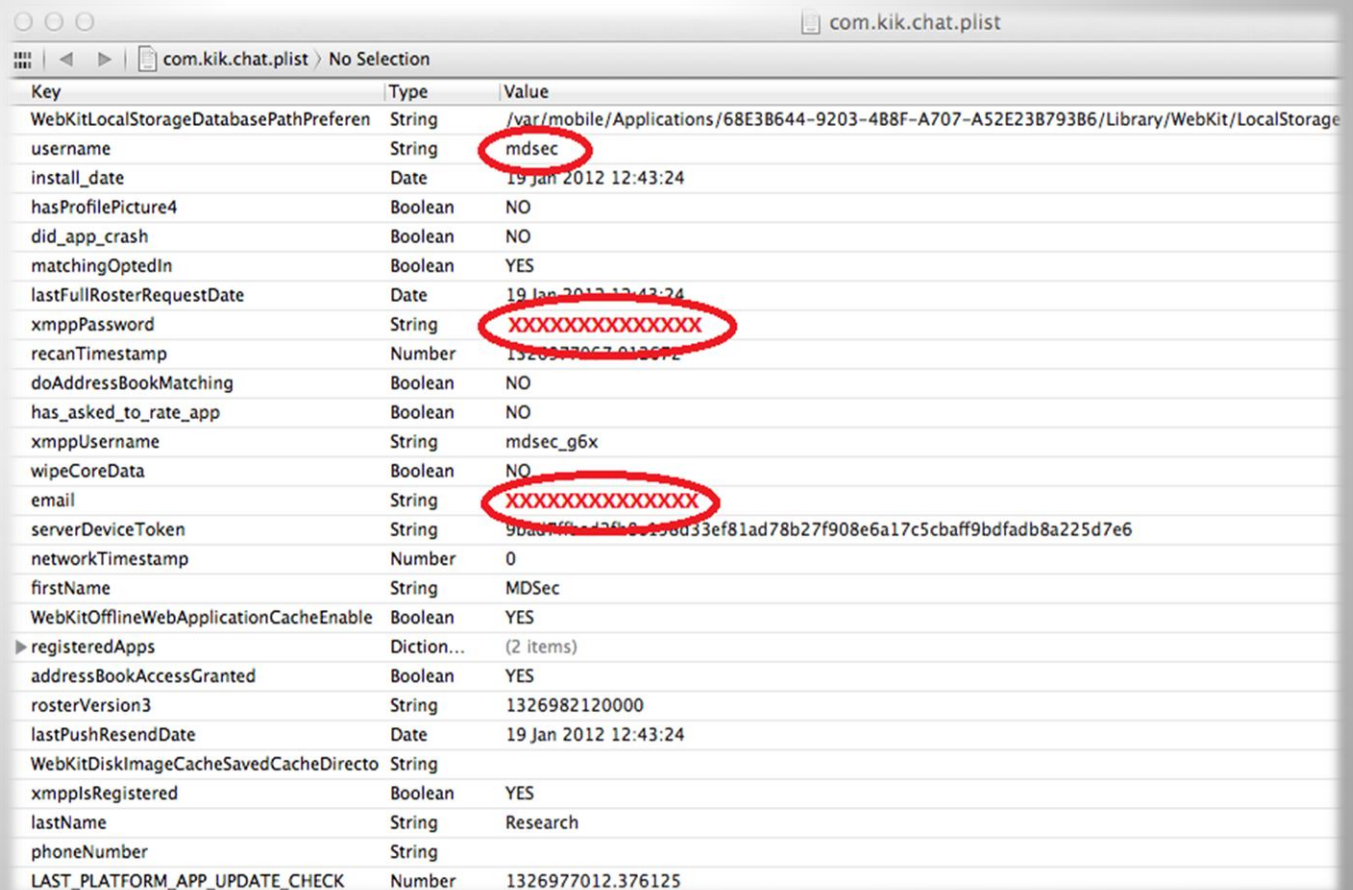


Data Storage

Kik Messenger

- Library/Preferences/com.kik.chat.plist:

- Username
- Password
- Email



Key	Type	Value
WebKitLocalStorageDatabasePathPreferen	String	/var/mobile/Applications/68E3B644-9203-4B8F-A707-A52E23B79386/Library/WebKit/LocalStorage
username	String	mdsec
install_date	Date	19 Jan 2012 12:43:24
hasProfilePicture4	Boolean	NO
did_app_crash	Boolean	NO
matchingOptedIn	Boolean	YES
lastFullRosterRequestDate	Date	19 Jan 2012 12:43:24
xmppPassword	String	XXXXXXXXXXXXXXXX
recanTimestamp	Number	1326977067.013672
doAddressBookMatching	Boolean	NO
has_asking_to_rate_app	Boolean	NO
xmppUsername	String	mdsec_g6x
wipeCoreData	Boolean	NO
email	String	XXXXXXXXXXXXXXXX
serverDeviceToken	String	90ad3ff8-1268-425dd33ef81ad78b27f908e6a17c5cbaff9bdfadb8a225d7e6
networkTimestamp	Number	0
firstName	String	MDSec
WebKitOfflineWebApplicationCacheEnable	Boolean	YES
registeredApps	Diction...	(2 items)
addressBookAccessGranted	Boolean	YES
rosterVersion3	String	1326982120000
lastPushResendDate	Date	19 Jan 2012 12:43:24
WebKitDiskImageCacheSavedCacheDirecto	String	
xmppsRegistered	Boolean	YES
lastName	String	Research
phoneNumber	String	
LAST_PLATFORM_APP_UPDATE_CHECK	Number	1326977012.376125

Data Storage

Kik Messenger

- Documents/kik.sqlite:
 - Chat history

Table: ZKIKMESSAGE 🔍

New Record Delete Record

	Z_PK	Z_ENT	Z_OPT	ZINTERNALID	ZSTATE	ZSYSTEMSTATE	ZTYPE	ZDRAFTM	ZI	ZLAS	ZREF	ZBODY	ZSTANZAID
1	1	3	2	0	16	0	1	1	1	1	342	Welcome to Kik! To find your friends, go to Contacts and	08431543-57
2	2	3	1	0	0	0	4				3761	MDSec has been added to your contacts	d9f57a4-79c
3	3	3	1	1	0	0	4		2		1198	MDSec has added you as a contact	77a2a207-ab
4	4	3	2	0	0	0	4				3917	Test has added you as a contact	a52fef4-d20
5	5	3	3	1	16	12	1				5113	Hi this is a test	23a909e-82
6	6	3	2	3	0	0	4				721	Test has been added to your contacts	909df251-c9
7	7	3	1	2	30	0	2				4112	Hello test	4eaa1982-c5
8	8	3	2	4	16	12	1	3			7503		3d6144c3-09

Data Storage

Kik Messenger

- Documents/fileAttachments:

mbp:Documents \$ file fileAttachments/057a8fc9-0daf-4750-b356-5b28755f4ec4
fileAttachments/057a8fc9-0daf-4750-b356-5b28755f4ec4: JPEG image data, JFIF
standard 1.01 mbp:Documents \$



D'oh
I messed again!

Keychain

Overview



- Encrypted container for storing sensitive information
- Apps can only access their keychain items unless part of a keychain access group:
 - Set by entitlements from provisioning profile
 - Jailbroken – apps to dump keychain
- 6 levels of protection:
 - kSecAttrAccessibleAlways
 - kSecAttrAccessibleWhenUnlocked
 - kSecAttrAccessibleAfterFirstUnlock
 - kSecAttrAccessibleAlwaysThisDeviceOnly
 - kSecAttrAccessibleWhenUnlockedThisDeviceOnly
 - kSecAttrAccessibleAfterFirstUnlockThisDeviceOnly

Protocol Handlers

Overview



- No real Inter-Process Communication
- Apps prohibited from sharing because of sandbox
- Apps sometimes need to share data
- Apps can register a custom protocol handler

Protocol Handlers Implementation



- Two methods for implementing protocol handlers
- `handleOpenURL`
 - Now deprecated
- `openURL`
 - Provides bundle identifier
 - Allows developer to validate source app
- Example found during an app assessment
 - `app://setConfiguration?landingpage=` - Set the landing page for an app

Protocol Handlers

Skype Vulnerability

- Skype registers the “skype:” protocol handler
- Malicious web site could make calls
- Skype app did not prompt or validate before call



```
<iframe src="skype://123456789?call"></iframe>
```

https://media.blackhat.com/bh-eu-11/Nitesh_Dhanjani/BlackHat_EU_2011_Dhanjani_Attacks_Against_Apples_iOS-WP.pdf

UIWebViews

Overview



- iOS rendering engine for displaying text, supports a number of formats:
 - HTML
 - PDF
 - RTF
 - Office Documents (XLS, PPT, DOC)
 - iWork Documents (Pages, Numbers, Keynote)
- Built upon WebKit and uses the same core frameworks as Safari
- Supports java-script, cannot be disabled
 - Unescaped input leads to Cross-Site Scripting

UIWebView

Cross-Site Scripting



- Similar attacks to standard XSS
 - Session theft etc
- Can occur whenever user controlled Objective C variables populated in to WebView
 - *stringByEvaluatingJavaScriptFromString*

```
NSString *javascript = [[NSString alloc] initWithFormat:@"var myvar=\"%@\";",  
username];
```

```
[mywebView stringByEvaluatingJavaScriptFromString:javascript];
```

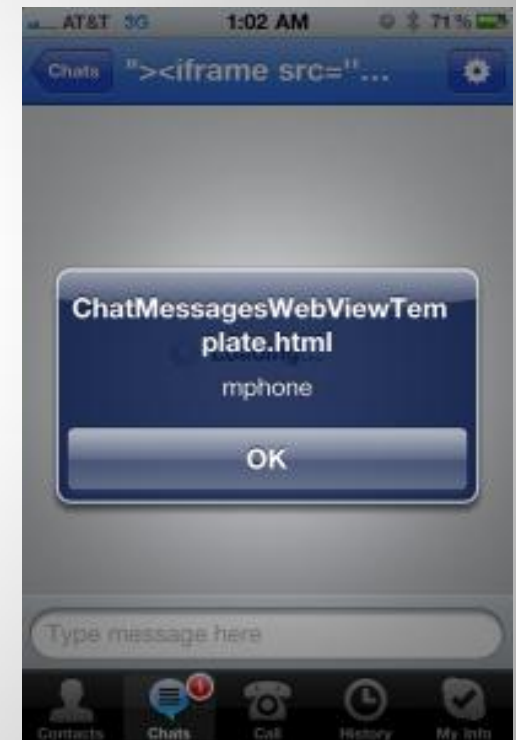
- No native JS to Objective C bridge
 - Developers will often implement one
 - Examples:
 - Using camera from JS
 - Sending e-mails from JS
 - Sending SMS from JS
- Bridge implemented using WebView specific URL handler:
 - `shouldStartLoadWithRequest`
- Bridge can often expose Objective C methods
 - Serialize/Unserialize methods & parameters
 - *`performSelector:NSSelectorFromString(method)`*

UIWebView

Cross-Site Scripting

- Real world example:
 - Skype (AGAIN!)
 - Displays “full name” from incoming call in a WebView
 - Used a local HTML template so loaded in local context
 - XSS in full name lead to addressbook theft

<https://superevr.com/blog/2011/xss-in-skype-for-ios/>



XML Processing Overview



- Widely used in mobile apps
- iOS offers 2 options for parsing XML with the SDK:
 - NSXMLParser
 - libXML2
- Lots of other third party implementations exist

- Not vulnerable to “billion laughs” attack by default
 - Parser raises a *NSXMLParserEntityRefLoopError* exception
- Not vulnerable to eXternal Xml Entity injection by default
- Developer must enable the *setShouldResolveExternalEntities* option
 - Not unthinkable, seen in practice on several occasions

```
NSXMLParser *addressParser = [[NSXMLParser alloc] initWithData:xmlData];  
[addressParser setShouldResolveExternalEntities:YES];
```

- Not vulnerable to “billion laughs” attack by default
 - Parser throws error: “Detected an entity reference loop”
- Vulnerable to eXternal XML Entity injection by default!

```
-(BOOL) parser:(NSString *)xml {  
  
    xmlDocPtr doc = xmlParseMemory([xml UTF8String], [xml  
lengthOfBytesUsingEncoding:NSUTF8StringEncoding]);  
  
    xmlNodePtr root = xmlDocGetRootElement(doc);  
}
```

SQL

Overview



- Apps may need to store data client-side
 - API supports SQLite
- Unsanitised user input in dynamic queries leads to SQL injection

```
NSString *sql = [NSString stringWithFormat:@"SELECT name FROM products  
WHERE id = '%@'", id];  
const char *query = [sql UTF8String];
```

- Used parameterised queries!

```
const char *sql = "SELECT name FROM products WHERE id = ?";  
sqlite3_prepare_v2(database, sql, -1, &sql_statement, NULL);  
sqlite3_bind_text(&sql_statement, 1, id, -1, SQLITE_TRANSIENT);
```

```
NSString *sql = [NSString stringWithFormat:@"INSERT INTO tweets VALUES('1',  
'%@','%@','%@')", tweet, user, displayname];  
const char *insert_stmt = [sql UTF8String];  
sqlite3_prepare_v2(database, insert_stmt, -1, &statement, NULL);  
if (sqlite3_step(statement) == SQLITE_DONE)
```

DEMO

Filesystem Interaction

Overview



- Objective C provides NSFileManager class for filesystem access:
 - Check if file exists
 - Compare file contents
 - Check file permissions
 - Move/Copy files
 - Read & write from/to files
- Can be affected by traditional file IO issues

Filesystem Interaction

Directory Traversal



- Vulnerable to vanilla traversals:
 - ../../../../../../

```
NSString *sourcePath = [[[NSBundle mainBundle] resourcePath]
stringByAppendingPathComponent:@file];
NSString *contents = [[NSString alloc] initWithData:[fm readContents:sourcePath]
encoding:NSUTF8StringEncoding];
```

```
- (NSData*) readContents:(NSString*)location {
    NSFileManager *filemgr;
    NSData *buffer;
    filemgr = [NSFileManager defaultManager];
    buffer = [filemgr contentsAtPath:location];
    return buffer;
}
```

Logging Overview



- API provides the NSLog() method
 - Will print to console
 - Visible in Xcode Organiser
- Some jailbreaks redirect console > syslog
- Some apps will use their own wrapper and log to app folder
- Don't store sensitive information there!
 - If used, ensure removed in release builds

```
NSLog(@"Account Number: %@, Sort code: %@", account, sortcode);
```

Geolocation Overview

- Provided by the Core Location framework
- Avoid being “too accurate”
- Don’t log location information on either client or server
 - If you MUST – make anonymous!



Geolocation Accuracy

- Can be set by one of the following constants:
 - kCLLocationAccuracyBestForNavigation;
 - kCLLocationAccuracyBest;
 - kCLLocationAccuracyNearestTenMeters;
 - kCLLocationAccuracyHundredMeters;
 - kCLLocationAccuracyKilometer;
 - kCLLocationAccuracyThreeKilometers;



STALKING
its not healthy

```
self.locationManager.desiredAccuracy = kCLLocationAccuracyBest;
```

Memory Corruption

Overview

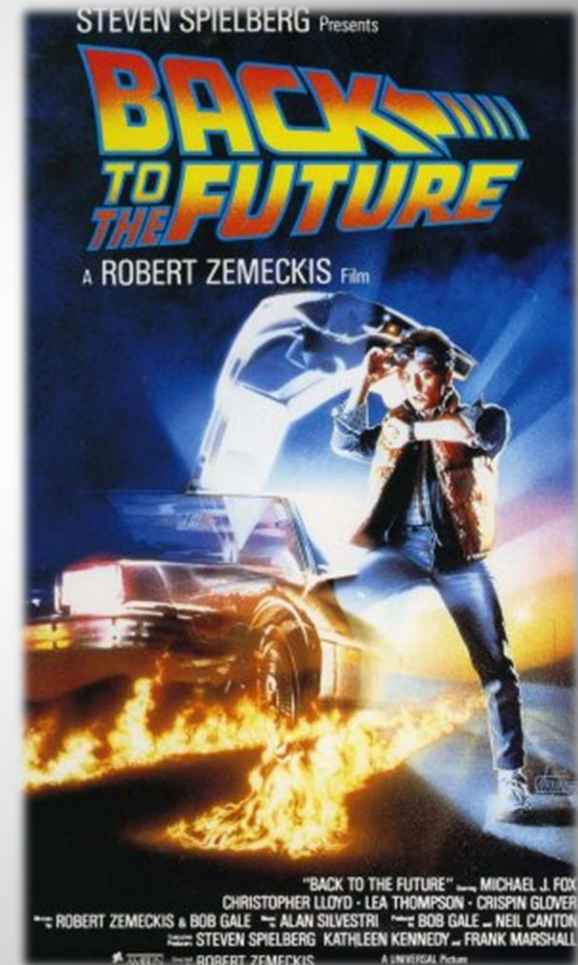


- As previously mentioned – superset of C
 - Developers often using straight C
 - Compiled to native code
 - Gives rise to the traditional issues
 - Overflows
 - Integer wraps
- Shouldn't need to allocate memory unless specific performance overhead
 - Stick to objective C allocators

Memory Corruption

Format Strings

- A number of API methods support format specifiers
- If used incorrectly, leads to classic format string bugs
- Vulnerable methods include:
 - NSLog()
 - [NSString stringWithFormat]
 - [NSString stringByAppendingFormat]
 - [NSString initWithFormat]
 - [NSMutableString appendFormat]
 - [NSAlert alertWithMessageText]
 - [NSException]



- Traditionally use %n to write to an arbitrary address address
 - Not available on iOS
- Apple provide %@ specifier for objects
 - Call an arbitrary function pointer!
 - Unfortunately rare to find data stored on stack ☹

Memory Corruption

Format Strings - Exploitation



- Example:

```
NSString *myURL=@"http://localhost/test";
NSURLRequest *theRequest = [NSURLRequest requestWithURL:[NSURL
URLWithString:myURL]];
NSURLResponse *resp = nil;
NSError *err = nil;
NSData *response = [NSURLConnection sendSynchronousRequest: theRequest
returningResponse: &resp error: &err];
NSString * theString = [[NSString alloc] initWithData:response
encoding:NSUTF8StringEncoding];
NSLog(theString);
```


Memory Corruption

Format Strings - Exploitation



- Example:

```
HTTP/1.1 200 OK  
Content-Length: 29
```

```
AAAA%08x.%08x.%08x.%08x.%08x.
```

- Output:

```
2012-01-31 17:46:41.780 fmtstr[2476:1207]  
AAAA93f9ea22.0030fc90.00000001.bffffbf8.00000000.
```

- Dumps stack memory

Memory Corruption

Object Use after Free



- Same concept as use-after-free bugs
- References to an object still exist after it has been freed
- Exploitable but unlikely in practice

- Transport security & data storage are probably two of the biggest issues for iOS apps
- Apps can be vulnerable to lots of API specific attacks
- Platform provides additional security features to mitigate against some attacks

- OWASP Mobile Security Project
 - https://www.owasp.org/index.php/OWASP_Mobile_Security_Project
- MDSec Research / Blog
 - <http://blog.mdsec.co.uk>
 - <http://www.mdsec.co.uk/research>

Q & A

That's all folks!

QUESTIONS?

- **Online:**
 - <http://www.mdsec.co.uk>
 - <http://blog.mdsec.co.uk>
- **E-Mail:**
 - dominic [at] mdsec [dot] co [dot] uk
- **Twitter:**
 - @deadbeefuk
 - @MDSecLabs

