Securing iOS Applications

Dr. Bruce Sams, OPTIMAbit GmbH

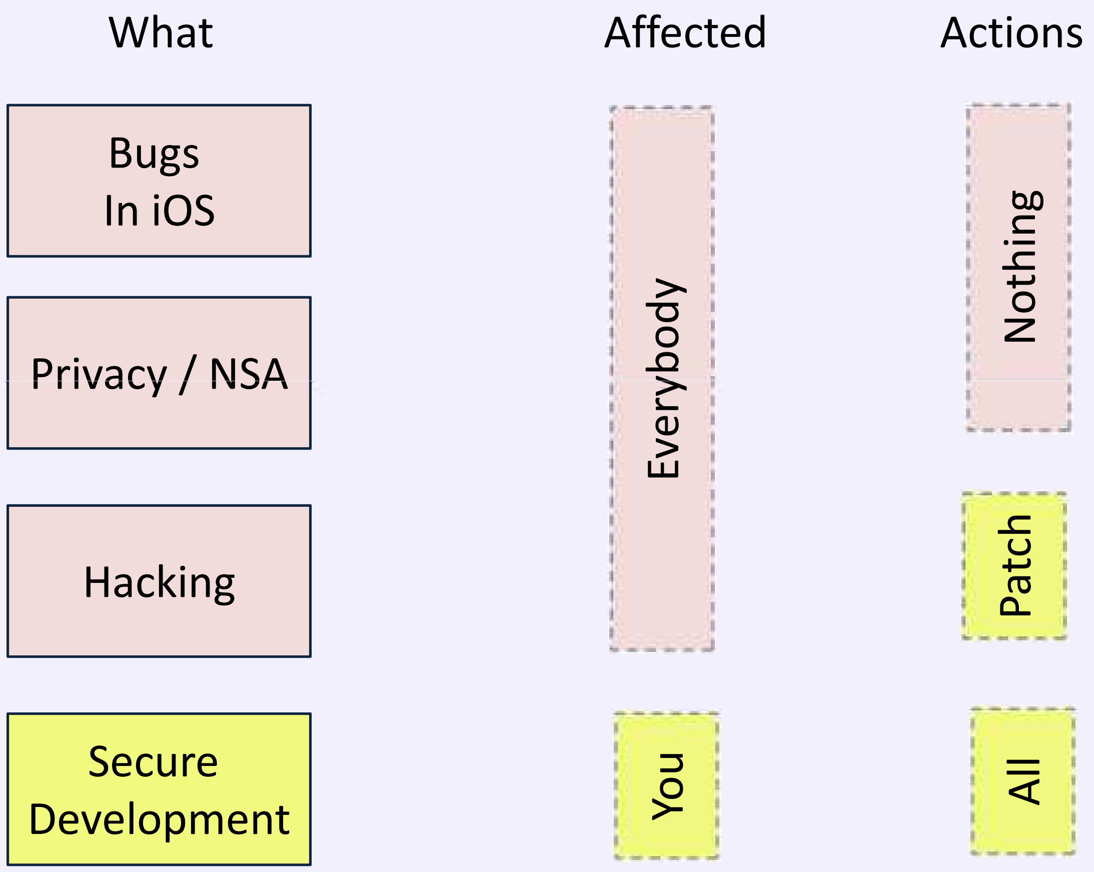
OWASP
The Open Web Application Security Project

About Me

- President of OPTIMAbit GmbH
- Responsible for > 200 Pentests per Year
- Ca 50 iOS Pentests and code reviews in the last two years.

Overview of the Talk

OWASP
The Open Web Application Security Project



Think Globally, Act Locally

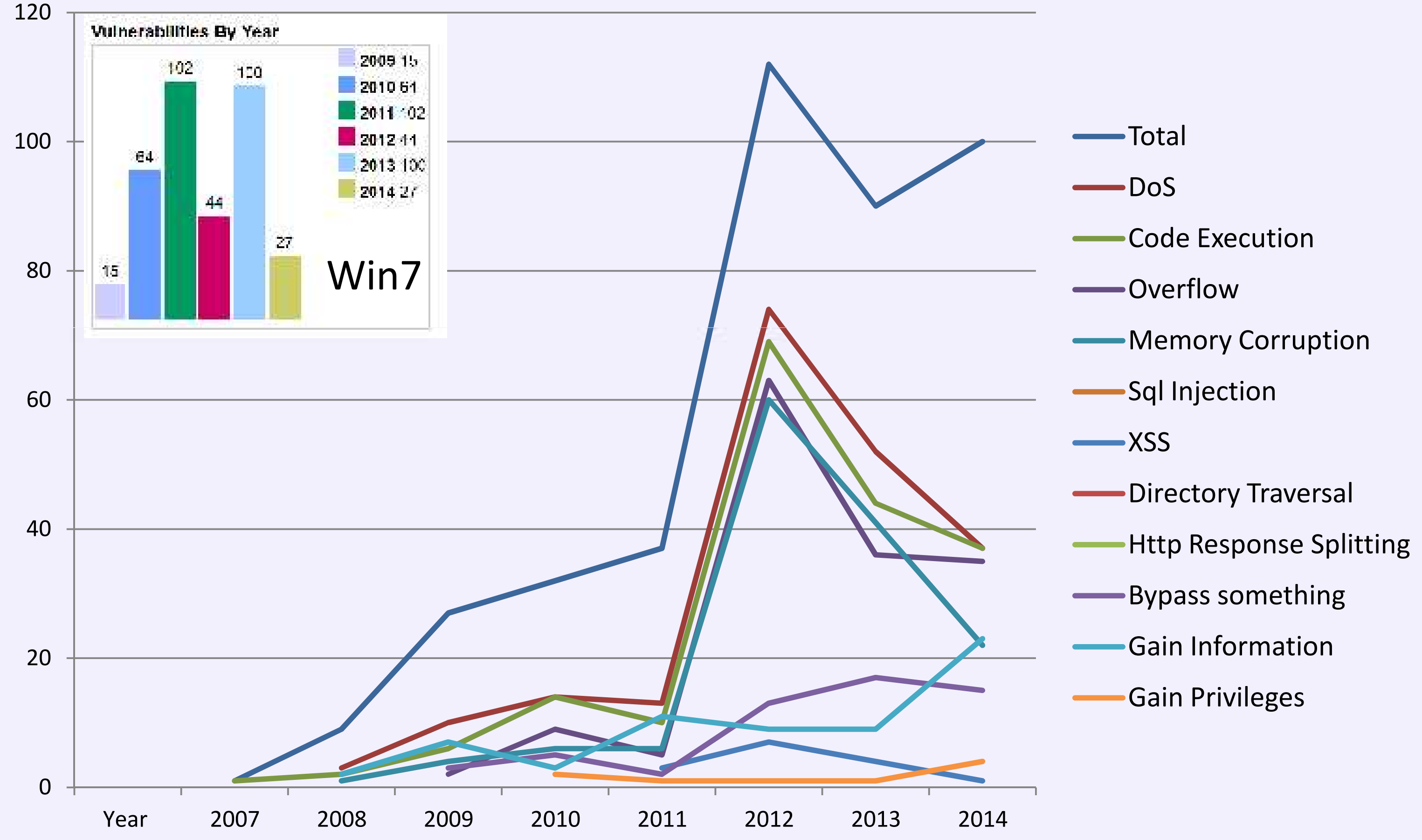
Security Goals of iOS

OWASP
The Open Web Application Security Project

- ✓ Sophisticated security built into the hardware.
- ✓ Strong encryption protects data at rest
- ✓ Mandatory code signing
- ✓ No possibility to revert to an older system
- ✓ Native language exploit mitigation (ASLR)
- ✓ App Sandboxing at Runtime
- ✘ Make jailbreaks impossible
- ? Secure Coding (not even on the list!)

iOS Vulnerability History (CERT)

OWASP
The Open Web Application Security Project



Privacy Improvement in iOS8

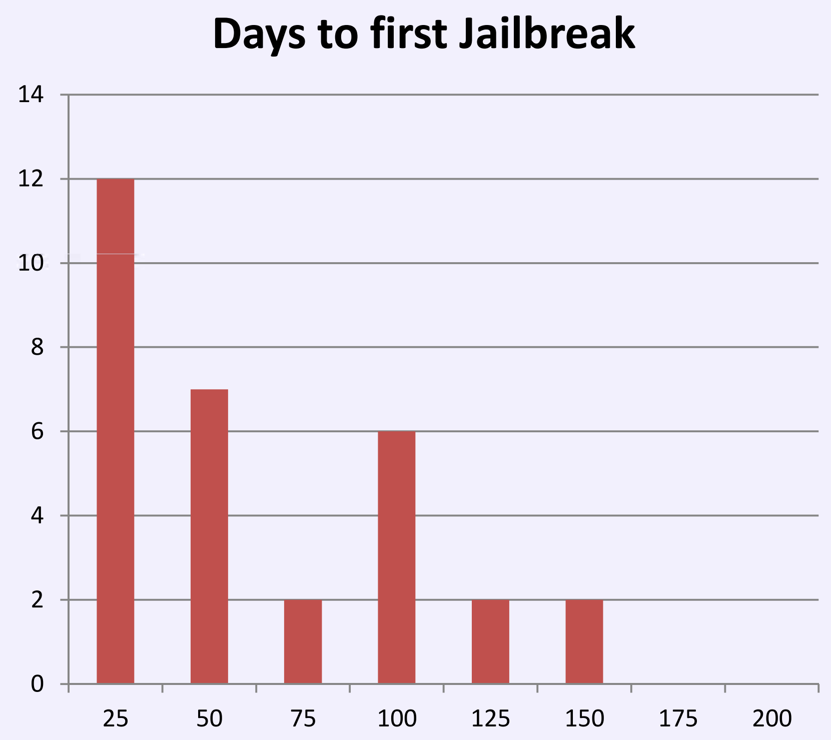
OWASP
The Open Web Application Security Project

- THREAT: An attacker could access services such as the Apple File Connection service (afc), the File Relay service and the Packet Sniffer over WiFi.
- Dump masses of third party application data using WiFi.
- Examples -- complete photo album, SMS messages, address book, typing cache, geolocation cache, application screenshots.
- FIX: Connections to many services are now restricted as “usb only”.

iOS Jailbreaks

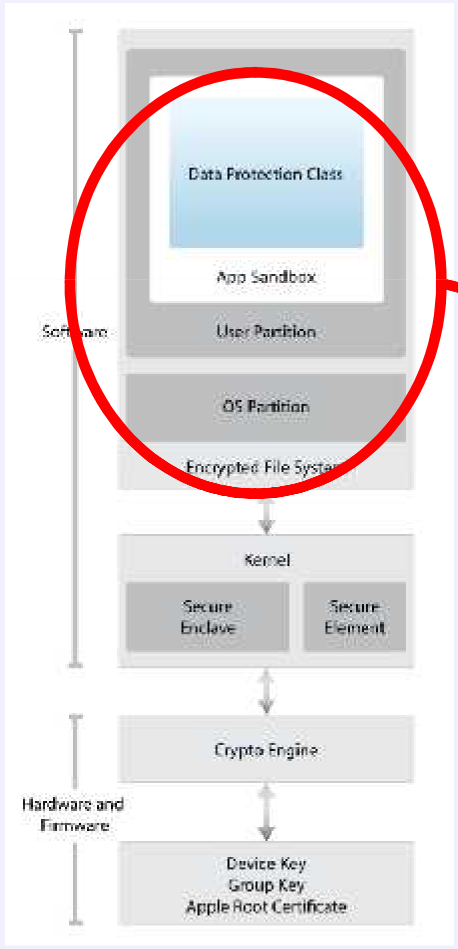
OWASP
The Open Web Application Security Project

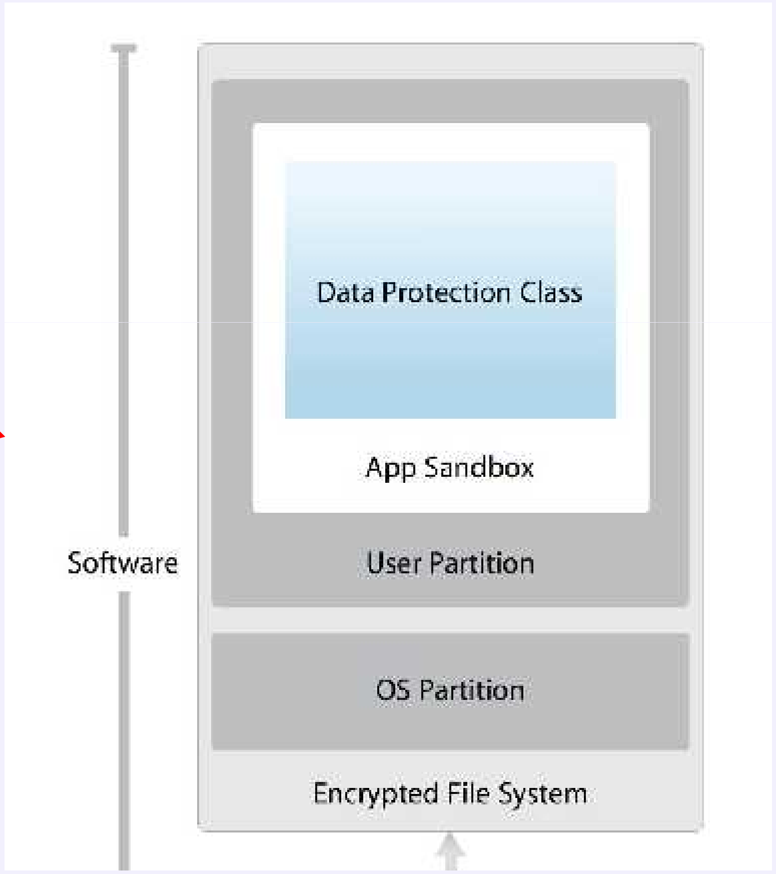
- Jailbreaking iOS is not easy, but it gets done regularly (mostly under 25 days)



OWASP
The Open Web Application Security Project

iOS Security for Developers

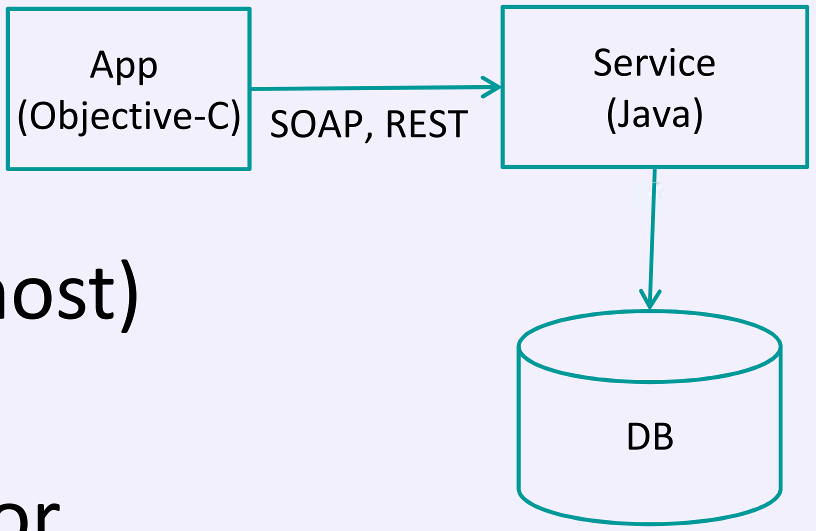




OWASP
The Open Web Application Security Project

Hybrid Apps

- Many enterprise apps are moving to the hybrid model.
- The app is only a presentation layer. (Almost) no data on the client.
- Security is easier than for „Rich Clients“.



Input Validation

- iOS apps are not really any different than other applications, when it comes to validation.
- Consequences of poor input validation
  - Buffer overflows (like C)
  - Format string vulnerabilities (like C)
  - URL commands
  - Code insertion
  - Cross Site Scripting
  - SQL Injection
  - Etc

Dangerous Functions

- There is a very long list of „Format String“ functions in the core libraries and in Cocoa.
- Examples:
  - `CFStringCreateWithFormat`
  - `CFStringCreateWithFormatAndArguments`
  - `NSAlert:alertWithMessageText`
  - `NSMutableString:initWithFormat`

SSL/TLS

- iOS has multiple APIs for managing secure network connections
  - NSURL (easy, high level, wraps CF Network)
  - CF Network (middle, c-based library)
  - Secure Transport (complex, low level)

- Only Secure Transport lets developers choose cipher suites!
  - `SSLSetEnabledCiphers`

XML Parsing

- iOS has its own XML Parser, NSXMLParser
  - Default: parses DTDs, but not nested entities.
  - Default: does not parse external entity references.
  - Option: „setShouldResolveExternalEntities“

- Alternate libxml2 parser
  - Higher perfromance
  - Parses external entities

IPC via Handlers

OWASP
The Open Web Application Security Project

- IPC Workaround: register custom protocol handlers via „`application:openURL`"
  - Register in the plist file, e.g. URL Identifier=xxx
  - Overwrite method `handleOpenURL()` to accept form of „`getdata`"
  - Pass data via xxx://getdata

- Attackers misuse the url, e.g. iframe with src
  `src=„xxx://getdata?http://evilurl"`

Cross Site Scripting

OWASP
The Open Web Application Security Project

- UIWebView is a complex component that parses and displays many content types (HTML, Excel, PDF, Keynote etc.)
  - Also runs JavaScript => XSS is possible

- Access to core functions via JavaScript – Obj-C Bridge
  - `#ifdef JSC_OBJC_API_ENABLED`
  - New feature, need research

Why Jailbreak Detection?

OWASP
The Open Web Application Security Project

- The theory
  - The internal protection methods of a mobile device should not be removed via jailbreak, otherwise the device is unsafe.
- The reality
  - A determined, sophisticated attacker can circumvent any jailbreak detection scheme.
- Conclusion
  - Jailbreak detection is recommended, but it cannot protect against all attackers.

Jailbreak Detection

OWASP
The Open Web Application Security Project

- App developers may want to ensure that their apps will not run on a jailbroken device.
- Four practical methods exist:
  - Check file paths
  - Check command shell execution
  - Check library locations
  - Check access rights

Example Tests for Jailbreak

OWASP
The Open Web Application Security Project

- Presence of file paths of some commonly used hacks, e.g.
    - /Applications/Cydia.app
    - /Library/MobileSubstrate/DynamicLibraries/Veency.plist
    - /usr/bin/sshd

- Test writing a file to a private directory

```
[stringToBeWritten writeToFile:@"/private/jailbreak.txt"
   atomically:YES
   encoding:NSUTF8StringEncoding error:&amp;error];
if(error==nil){ //Device is jailbroken
return YES;
}
```

Encryption and Data Protection

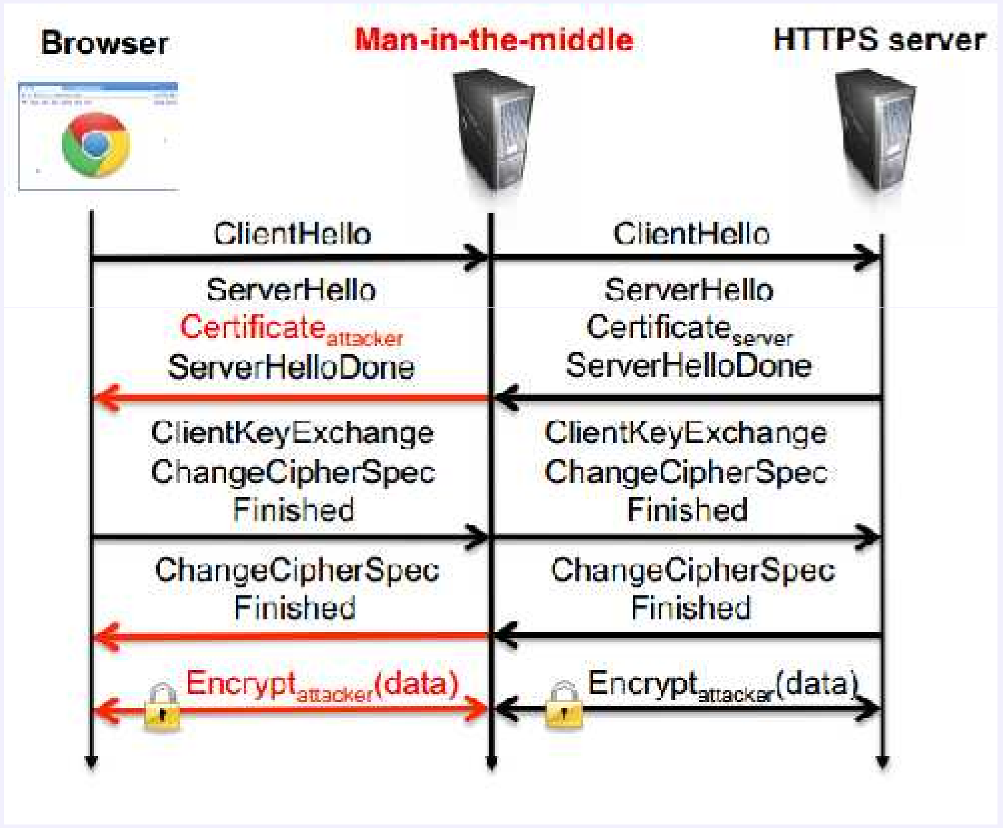
OWASP
The Open Web Application Security Project

- iOS provides many “Protection Classes” for encryption.
  - Fine grain control of when data is encrypted
  - Default Protection Class `ProtectionCompleteUntilFirstUserAuthentication` (protect against attacks that require a reboot)
  - plists are not encrypted!
  - Developers get into trouble when they try to be “clever” with the encryption

Attacking Data in Transit

OWASP
The Open Web Application Security Project

- SSL and certificates: MyCert is signed by IntermediateCert which is signed by RootCert. The iOS device trusts RootCert.
- If an attacker can perform a MITM attack, he can modify the data in transit.
- One interesting scenario: a mobile user intercepts HIS OWN data, in order to modify it.



Solution: Certificate Pinning

- The iOS app does not trust the RootCert, but instead ONLY the Cert that is installed on the server.
- The app is „pinned“ to the server, since it won‘t communicate with any other servers.
- For mobile apps that communicate only with a limited number of servers, this adds a substantial level of protection.
- Disadvantage: Certificate Pinning cannot be configured; it must be implemented in code.

Cert Pinning HowTo

OWASP
The Open Web Application Security Project

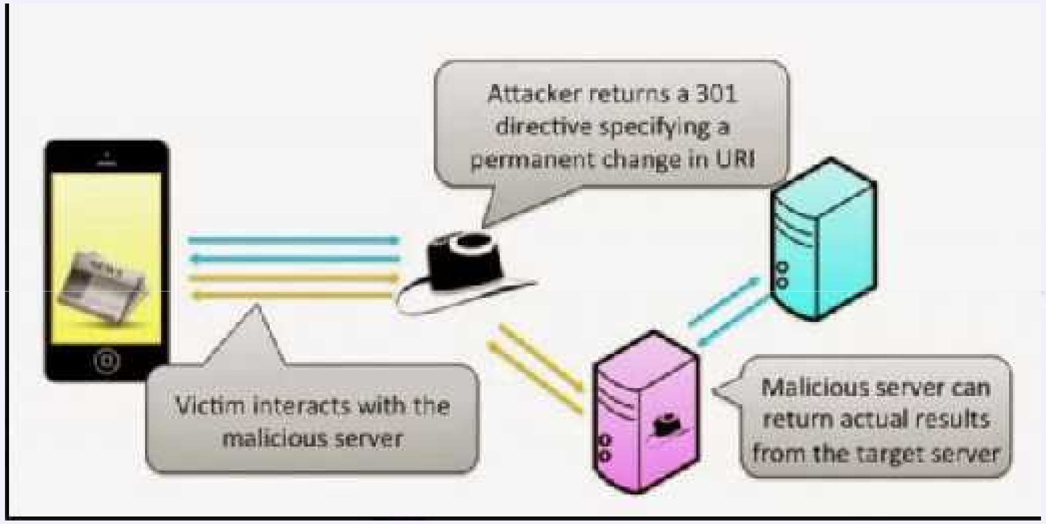
- iOS pinning is performed through a NSURLConnectionDelegate that implements `connection:didReceiveAuthenticationChallenge:`

- Example code available on the OWASP website: https://www.owasp.org/index.php/Certificate_and_Public_Key_Pinning

- MobileApp Pinning may be reverse engineered/defeated. Exploits already exist, e.g. iOS SSL Kill Switch (Github) works on jailbroken devices.

HTTP Request Hijacking

- In HRH, the attacker can persistently alter the server URL from which the app loads its data.
- Example: instead of loading the data from real.com the attack makes the app persistently load the data from attacker.com


Attacker returns a 301 directive specifying a permanent change in URI
Victim interacts with the malicious server
Malicious server can return actual results from the target server

It‘s not a bug, it‘s a feature!

OWASP
The Open Web Application Security Project

- The problem is based on the HTTP 301 Response specification.
- **10.3.2 301 Moved Permanently** The requested resource has been assigned a new permanent URI and *any future references to this resource SHOULD use one of the returned URIs*. Clients with link editing capabilities ought to automatically re-link references to the Request-URI to one or more of the new references returned by the server, where possible. *This response is cacheable unless indicated otherwise*.

The attack can only work if a MITM exists. Since ca 75% of all users connect to insecure WLANs in airports and cafes, this attack is easy to setup.

In the corporate environment, could use a spoofed WLAN access point.

Two HRH Solutions

1) Connect only using SSL (simple and effective, but perhaps not practical everywhere).
2) Write a derived Class to eliminate caching of the 301 Response. See OWASP for a full example.

```
@interface HRHResistantURLCache : URLCache
@end
@implementation HRHResistantURLCache
-(void) storeCachedResponse:(NSCachedURLResponse *)cachedResponse
              forRequest:(NSURLRequest *)request
 if (301 == [(NSHTTPURLResponse *)cachedResponse.response statuscode]) {
 return;
}
[super storeCachedResponse:cachedResponse forRequest:request];
@end
```

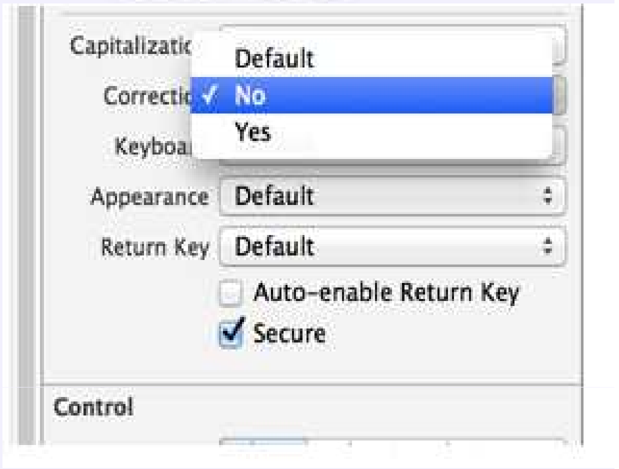
Secure Local Storage

OWASP
The Open Web Application Security Project

- Confidential data such as passwords, session ID's etc should never be stored locally on the device. If there is no other option, it should be stored on the keychain, not in NSUserDefaults.
- NSUserDefaults stores information in an unencrypted format in a plist file under Library -> Preferences -> $AppBundleId.plist
- Use helpful libraries, e.g. PDKeychainBindingsController (Github).
- Core Data files are also stored as unencrypted database files in your application bundle.

Handling Confidential Fields

- iOS usually caches all entries in text fields
  - confidential fields should be marked as “secure”.
  - disable *AutoCorrection* for those text fields.


Capitalizatic Default
Correctic [✓ No] Yes
Keyboa
Appearance Default
Return Key Default
Auto-enable Return Key
[✓ Secure]
Control

- Clear the Pasteboard once the application enters background.
  - [UIPasteboard generalPasteboard].items = nil;

Have the Goals Been Reached?

OWASP
The Open Web Application Security Project

- Security on mobile devices is a very hard problem to solve:
  - Generally excellent work from Apple
  - The APIs/Secure Coding can be improved

- Developers must do their part:
  - Proper use of encryption & data storage
  - Avoid dangerous constructions
  - “Special” options, e.g Certificate Pinning

- DO A CODE AUDIT!!

Exeunts

Thanks for listening!

OPTIMAbit GmbH

Marktplatz 2

85375 Neufahrn

Tel.:	+49 8165/65095
Fax	+49 8165/65096

bruce.sams@optimabit.com

www.optimabit.com