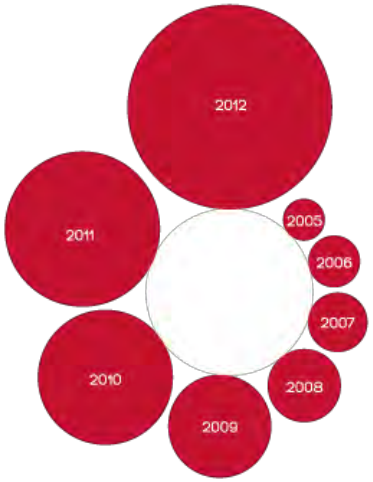
nccgroup
freedom from doubt

How to assess and secure iOS apps

An NCC Group workshop

About NCC Group



Eighth successive year of double digit growth

We protect 15,000 clients worldwide

We monitor over 16 million web pages every week

World's largest penetration testing team

Outline

- Introduction to iOS and Objective-C
- Platform security
- iOS apps
- Testing environment
- Black-box assessment
- Conclusion

Outline

- **Introduction to iOS and Objective-C**
- Platform security
- iOS apps
- Testing environment
- Black-box assessment
- Conclusion

Introduction to iOS

- iOS is derived from OS X, runs the same Darwin OS
- Apps written primarily in **Objective-C**
- Development in **Xcode**
  - Mac is needed
  - High-level API, "**Cocoa Touch**"
  - **iOS Simulator** – compile apps to native code to run locally

Introduction to Objective-C

- **Object-oriented** language inspired by Smalltalk
- Strict superset of **C**
  - Adds syntax for classes, methods, etc.
  - Adds concepts like **delegation**
- Methods are not called, **messages are passed** instead
- Libraries are referred to as **frameworks**

Objective-C – defining interfaces

```
@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)

Objective-C – more on interfaces

```
#import "NSParentClass.h"

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

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

- This is the new way of declaring interfaces

Message passing in Objective-C

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

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

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

- This is the "implementation", stored in `.m` files
- `@synthesize` creates getter and setter methods for properties
- At runtime this translates to

```
objc_msgSend(myInstance, setGurgle, @"foo")
```

Memory management

- **No garbage collection** in iOS
- In the past, object references tracked with retain and release methods
  - **MRR** – Manual Retain-Release
- iOS 5 SDK adds **Automatic Reference Counting** (ARC)
  - Compiler decides where to insert retain/release methods

Outline

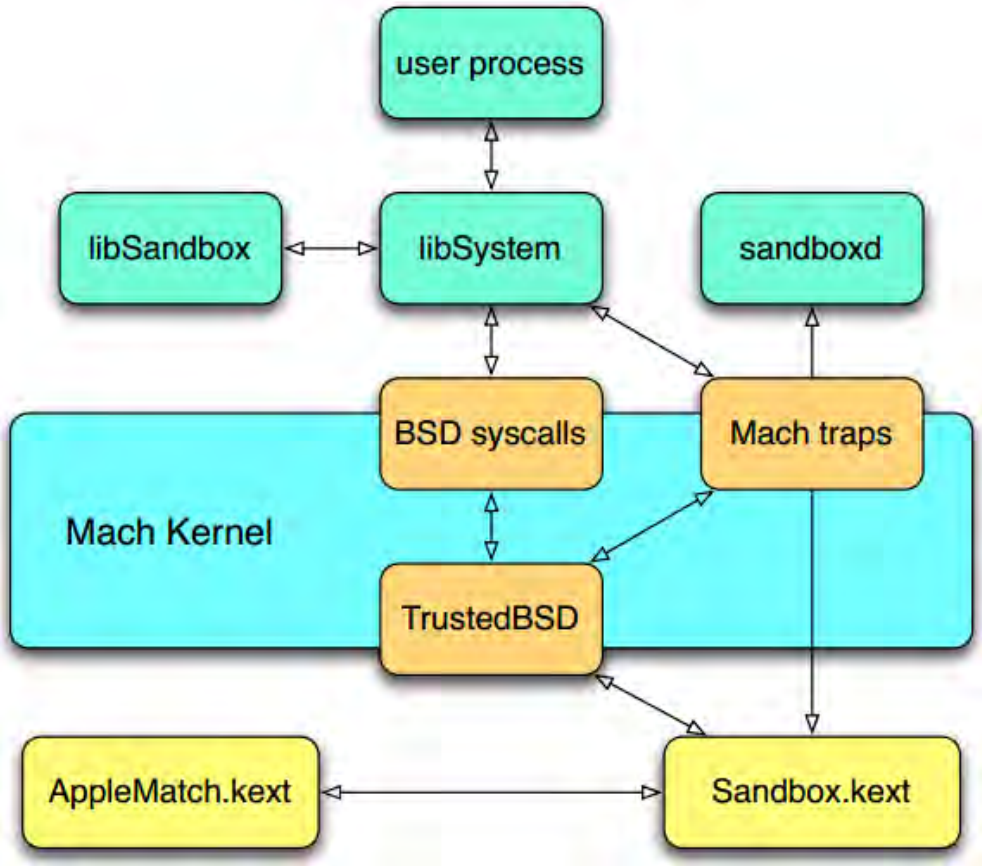
- Introduction to iOS and Objective-C
- **Platform security**
- iOS apps
- Testing environment
- Black-box assessment
- Conclusion

Secure boot chain

- First layer of defence for the platform security
- Each step of the boot-up is cryptographically signed by Apple
- Each step ensures the next step is signed by Apple

App sandbox

- App sandbox is called **seatbelt**
- Based upon TrustedBSD **MAC** framework
- **Entitlements** control access to user information and system-wide features
- Apps run under the same standard user `mobile`



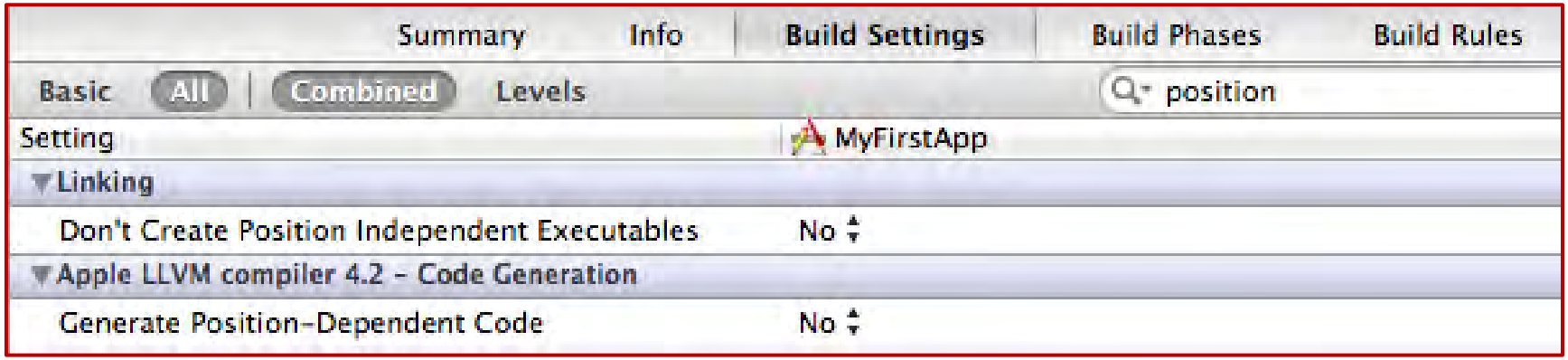
App code signing

- Runtime security feature
- Only apps that are signed by Apple issued certificates can be executed
- Prevents running of unauthorized applications on the device by validating the app signature upon execution

Kernel, runtime protection, etc.

- **XN bit** (eXecute Never) available for quite a while
  - No RWX allowed, only **R-X** or **RW-** pages

- Since iOS 4.3, **ASLR** protection
  - Developer corner: ensure PIE is enabled for your Xcode project in the `Project Build Settings`



- Syscalls `SYS_setreuid` and `SYS_setreguid` are removed at kernel level

Storage encryption

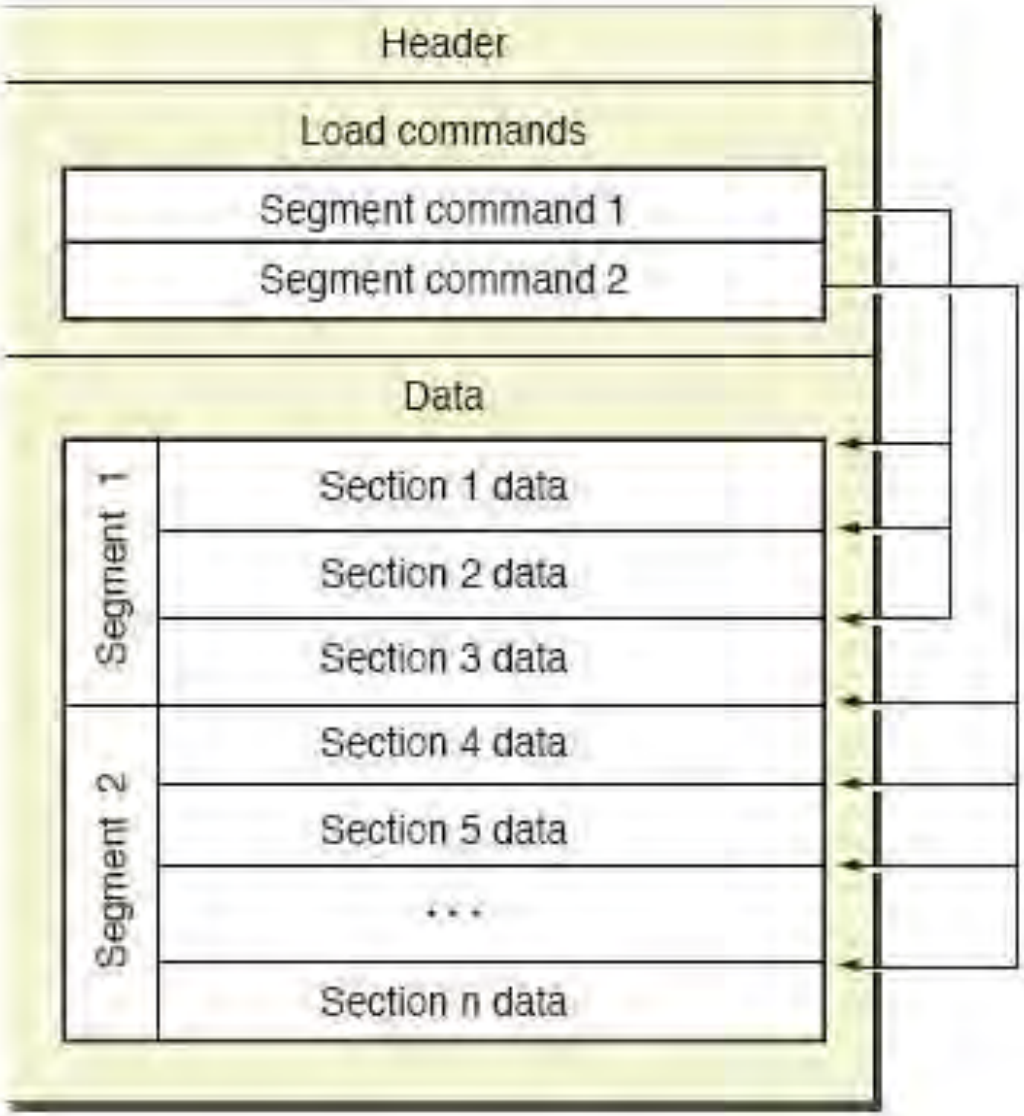
- iOS devices have full disk crypto
  - Each file encrypted with its own key, which is in turn encrypted by the file-system key
- This protects against someone disassembling the device and analyzing it directly with a specialized reader
- Does not protect against
  - **Jailbreaking** and extracting data off the drive
  - **Backing** up device data with iTunes
  - A software **exploit** being able to read your stuff
  - **Cracking** the PIN and using the device

Outline

- Introduction to iOS and Objective-C
- Platform security
- **iOS apps**
- Testing environment
- Black-box assessment
- Conclusion

iOS apps

- Application binaries are in the Mach-O file format
- Three parts:
  - Header
  - Load
  - Data
- Each app gets unique identifier (GUID) with corresponding home directory, inside **/var/mobile/Applications/**



iOS apps distribution and location

- Apps are distributed as `.ipa` archives – **ZIP** file format
- Apps are stored inside the folder `/var/mobile/Applications/`
- Apps installed natively by Apple are stored inside the folder `/Applications/`
- Applications within the iOS Simulator are stored inside `/Users/<username>/Library/Application Support/iPhone Simulator/<v>/Applications/`
- Locate all installed apps

```
iPhone:~ root# less \
/private/var/mobile/Library/Caches/com.apple.mobile.installation.plist
iPhone:~ root# find / -name "*.app"
```

iOS apps bundle

File	Description
`AppName.app/` `  **AppName** ` Info.plist`	App resources: graphics, nibs, binary, Info.plist, etc. App binary App configuration file, includes bundle GUID, display name, version number, etc.
`Documents/` `  Inbox/`	User specific data. Backed up in iTunes Data other apps have asked the app to open

`Library/`
`  Application Support/`
`  Caches/`
`  Snapshots/`
`  Cookies/`
`  Preferences/`
`  WebKit/`
	App specific files. Backed up in iTunes. Not shared with user
	App generated files, templates, etc.
	Data to persist across subsequent executions (eg. db caches)
	Display screenshots
	Cookies
	User's preferences – `NSUserDefaults`
	WebKit local storage

`tmp/`	Temporary files
`iTunesMetadata.plist`	Dictionary file (containing some sensitive information about the purchaser)

Outline

- Introduction to iOS and Objective-C
- Platform security
- iOS apps
- **Testing environment**
- Black-box assessment
- Conclusion

Jailbreak your iDevice

- Allow running any app, control iDevice and access arbitrarily the file system access
- Exploit a vulnerability to install own tools and maintain access
  - Disables sandbox and diminishes code signing requirement
- Two types of jailbreaks, depending on the tool and underlying exploitation technique
  - **Untethered** – once restarted upon jailbreak, you can still use the phone and it remains jailbroken
  - **Tethered** – once restarted upon jailbreak you will need to connect to the PC to boot into jailbroken mode
- Currently, no public jailbreak for iOS 7.0 – latest is `evasi0n` for iOS 6.0-6.1.2

Cydia

- Application platform for jailbroken iDevices
  - Typically installed during the jailbreak process
- Cydia **repositories** host iOS tweaks and apps not allowed in the App Store
- Cydia **apps** are packaged as `.deb` files – Debian's dpkg package format
- Install `APT 0.6 Transitional` for `apt-get` to install apps from command line
- Install `OpenSSH` for SSH server on iDevice

Access the device

- Login over SSH to the device after OpenSSH is installed
  - Default credentials: `root` / `alpine` (change it with `passwd` command)

- Alternatively, tunnel a network connectivity through USB multiplexer
  - With usbmuxd (OS X)

```
$ git clone http://cgit.sukimashita.com/usbmuxd.git/
$ cd usbmuxd/python-client/
$ python tcprelay.py -t 22:2222
$ ssh -p 2222 root@127.0.0.1      # in another shell
```

  - With iTunnelMux (Windows and OS X)

```
C:\>itunnel_mux.exe --lport 2222
```

Cydia repositories

- Public Cydia repositories list – http://www.ijailbreak.com/cydia-repositories/

- Well-known repositories with tools useful to assess apps

```
iPhone:~ root# cat << EOF > /etc/apt/sources.list.d/repos.list
deb http://apt.modmyi.com/ stable main
deb http://apt.saurik.com/ ios/793.00 main
deb http://apt.thebigboss.org/repofiles/cydia/ stable main
deb http://cydia.zodttd.com/repo/cydia/ stable main
deb http://coredev.nl/cydia iphone main
deb http://nix.howett.net/theos ./
deb http://repo.insanelyi.com/ ./
deb http://repo.nesolabs.de/ ./
EOF

iPhone:~ root# apt-get update
iPhone:~ root# apt-get upgrade
```

Instrumentation

- Apple ships the iPhone without a usable shell
  - A lot of common/useful utilities are not installed by default
- Install tools manually after jail-breaking
- One-liner for tools within public Cydia repositories

```
iPhone:~ root# apt-get install adv-cmds com.sull.clutchpatched
curl cycript odcctools developer-cmds dpkg
com.ericasadun.utilities file file-cmds findutils gawk git grep
inetutils com.autopear.installipa ldid less lsof mobilesubstrate
com.saurik.substrate.safemode mobileterminal-applesdk nano netcat
network-cmds python sed shell-cmds sqlite3 syslogd system-cmds
tcpdump top uikittools unrar unzip vim wget whois zip
```

Build target app from source code

- Disable code signing for Xcode – `SDKSettings.plist` file

SDKSettings.plist

SDKSettings.plist › No Selection

Key	Type	Value
Root	Dictionary	(13 items)
AlternateSDK	String	iphonesimulator6.1
CanonicalName	String	iphoneos6.1
CustomProperties	Dictionary	(0 items)
DefaultProperties	Dictionary	(10 items)
AD_HOC_CODE_SIGNING_ALLOWED	String	NO
CODE_SIGNING_REQUIRED	String	NO
CODE_SIGN_ENTITLEMENTS	String	
CODE_SIGN_IDENTITY	String	iPhone Developer

- Edit `Build Settings` for your project

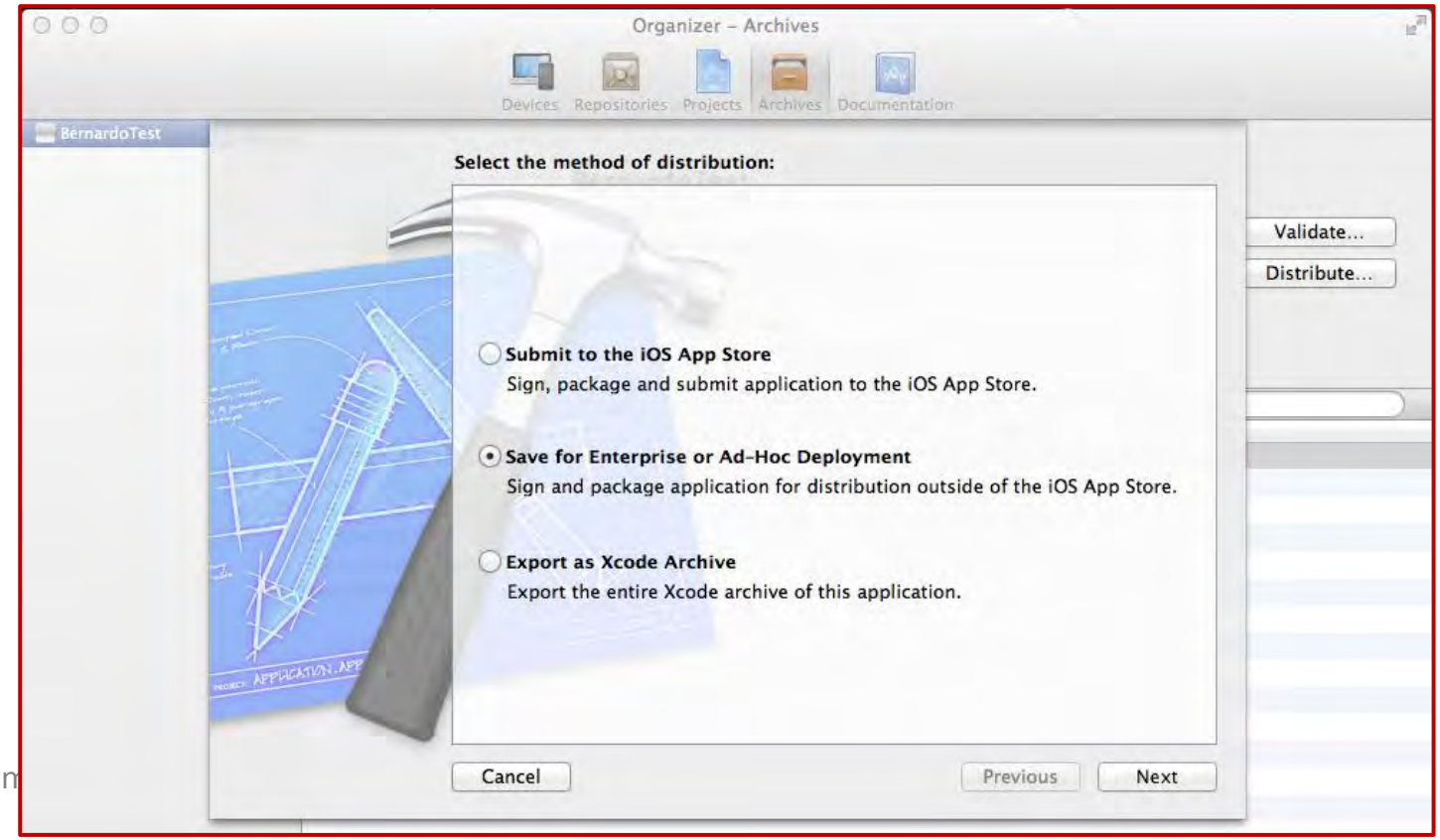
Info | Build Settings

Basic | All | Combined | Levels | code sign

Setting	BernardoTest
Code Signing	
Code Signing Entitlements	
Code Signing Identity	Don't Code Sign
Debug	Don't Code Sign
Any SDK	Don't Code Sign
Release	Don't Code Sign
Any SDK	Don't Code Sign
Code Signing Resource Rules Path	
Other Code Signing Flags	

Pack target app

- Pack the app to an `.ipa` file
  - In Xcode click on `Product` → `Archive`
  - Click on `Distribute…`
  - Select `Save for Enterprise or Ad-Hoc Deployment`



Deploy target app to iDevice

- Upload the packed app to the iDevice (with `scp`)

- Use the IPA Installer Console to install the app – example

```
iPhone:~ root# ipainstaller -c TargetApp.ipa
Clean installation enabled.
Will not restore any saved documents and other resources.

Analyzing TargetApp.ipa...
Installing TargetApp (v1.0)...
Installed TargetApp (v1.0) successfully.
Cleaning old contents of TargetApp...
```

- Alternatively, use iPhone Configuration Utility by adding the app to the library and then installing it to the plugged-in iDevice

Outline

- Introduction to iOS and Objective-C
- Platform security
- iOS apps
- Testing environment
- **Black-box assessment**
- Conclusion

Black-box assessment

- **Application traffic analysis**
- Client / server assessment
- Local data storage
- Keychain
- Logs
- Cache
- Inter-protocol communication (IPC)
- Binary analysis
- Runtime analysis

Passive network traffic monitoring

- Numerous apps act as clients to a server

- Passive traffic interception

```
iPhone:~ root# tcpdump -vv -i en0 -s 0 -n -U -w /dump.pcap
```

- Piping tcpdump with netcat

```
Linux:~$ sudo wireshark -k -i <(nc -l 7777)

iPhone:~ root# tcpdump -vv -i en0 -s 0 -n -U -w - \
"not port 7777" | nc <Linux IP> 7777
```

Passive network traffic monitoring

- Piping tcpdump with named pipe

```
Linux~$ sudo mkfifo /tmp/pipe
Linux~$ ssh root@<iDevice IP> "tcpdump -vv -i en0 -s 0 -n
-U -w - "not port 22" > /tmp/pipe
```

- Execute Wireshark to read the above named pipe

```
Linux~$ sudo wireshark -k -i /tmp/pipe
```

- Similarly, on Windows you can use ADVsock2pipe

Be a gateway to the iDevice

- You can monitor the traffic on a network gateway – your laptop
- Set the `Router` of your iDevice to be your laptop while connected to a wireless network
  - `Settings → Wi-Fi → Network name → IP Address → Static`
- Enable IP forwarding on your laptop – Linux

```
Linux~$ sudo sysctl -w net.ipv4.ip_forward=1
```

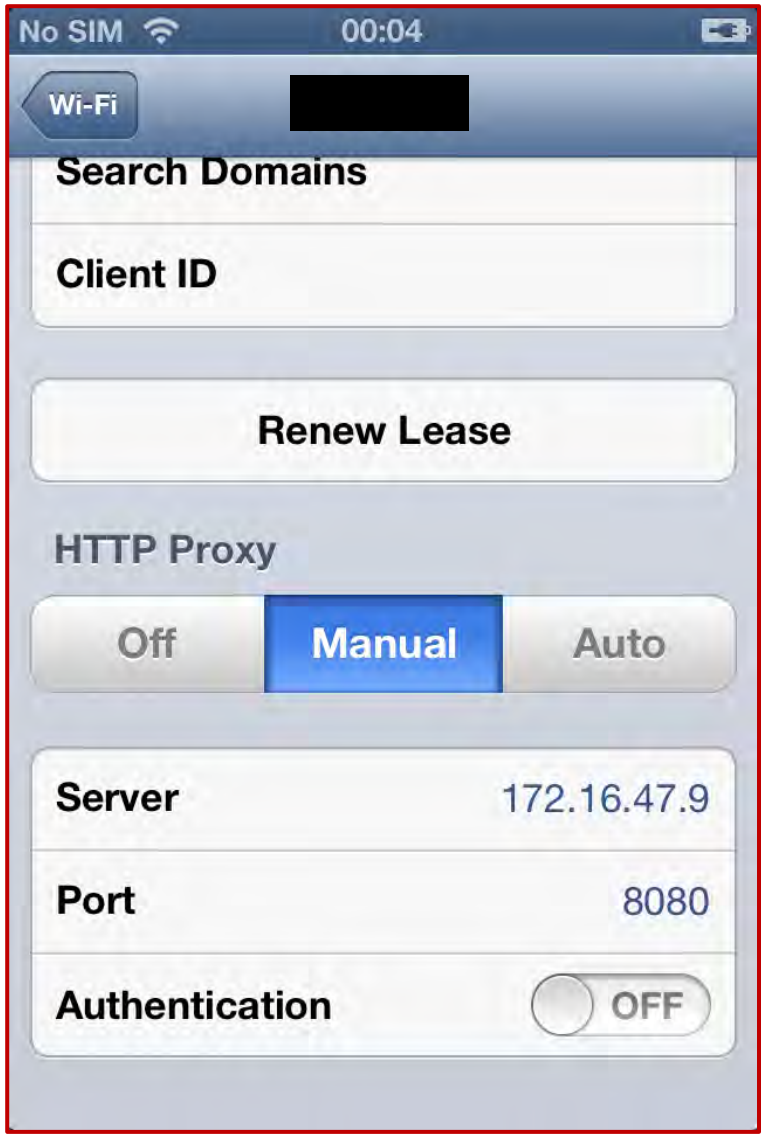
- On Windows set the following registry key to `1` – reboot is required

```
HKLM\SYSTEM\CurrentControlSet\Services\Tcpip\Parameters\IP
EnableRouter
```

- Ensure that your laptop **firewall is disabled**

Intercept HTTP traffic

- Run your real-time web proxy software of choice on your laptop
  - Burp Suite
- When connected to a **wireless** network
  - `Settings` → `Wi-Fi` → `Network name` → **`HTTP Proxy`** → `Manual` → enter `Server` and `Port`
- When connected to a **mobile** network
  - Use iPhone Configuration Utility to create a **Configuration Profile** with a proxy server and port for `APN`



Deal with HTTPS connections

- Import PortSwigger CA (Burp Suite) to the iDevice
  - Navigate to http://burp/cert with Safari
  - Install the CA to establish trust
- Alternatively, export the CA to a `.crt` file, send it by email as an attachment, open it in `Mail` app and install it
- You can also use iPhone Configuration Utility to create a **Configuration Profile** adding the CA under `Credentials`

Network traffic local redirection

- `CFStreams` and `NSStreams` do not honour HTTP proxy settings unless their traffic is routed via `CFNetworkCopySystemProxySettings()` – non-default
- Set your laptop to be the gateway for the iDevice
- On your laptop, redirect all HTTP(S) traffic to your local instance of Burp

```
Linux~$ sudo iptables -t nat -A PREROUTING -i eth0 -p \
tcp -m tcp --dport 80 -j REDIRECT --to-ports 8080

Linux~$ sudo iptables -t nat -A PREROUTING -i eth0 -p \
tcp -m tcp --dport 443 -j REDIRECT --to-ports 8080
```

- Ensure that Burp listener supports **invisible proxying**

Non-HTTP(S) network monitoring

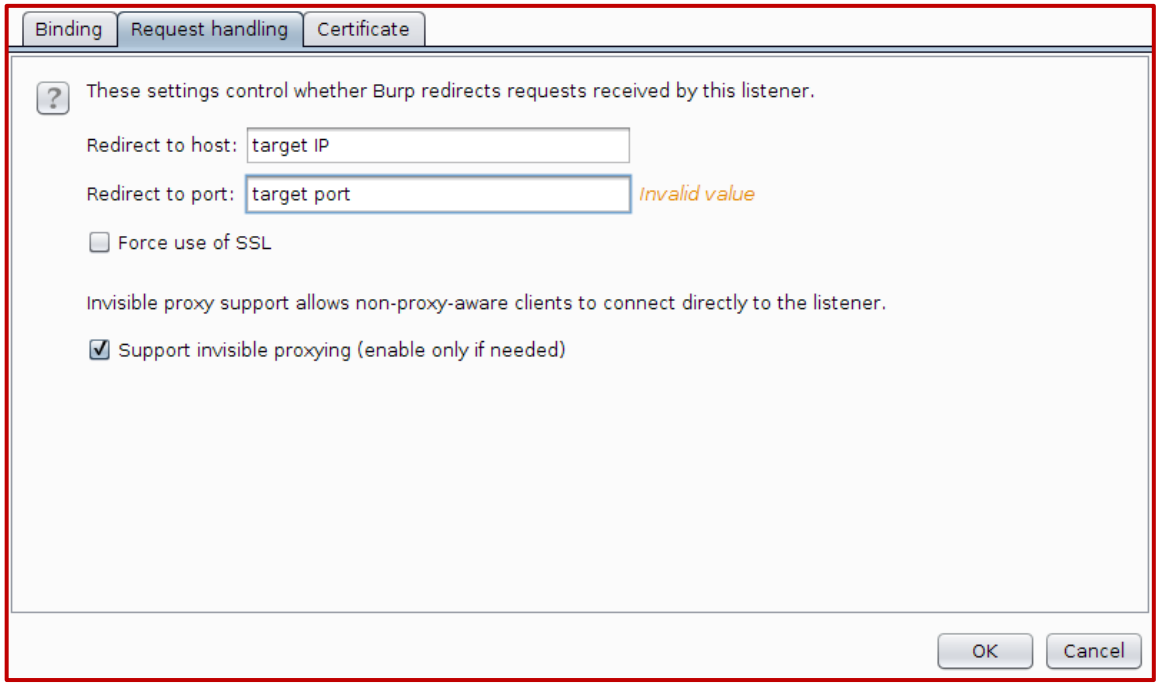
- Sometimes apps generate non-HTTP(S) traffic – DNS, SNMP, FTP, etc.
- Device proxy settings only route (most) HTTP traffic
- Setup a custom DNS server to resolve the target domain to your laptop IP address and use it from your iDevice
  - dnsRedir (by iSEC)
  - dnsChef
  - Metasploit's FakeDNS

Non-HTTP(S) network monitoring

- Run a suitable TCP/UDP proxy on your laptop
  - tcpprox (SSL trickery and IPv6 support, by iSEC)

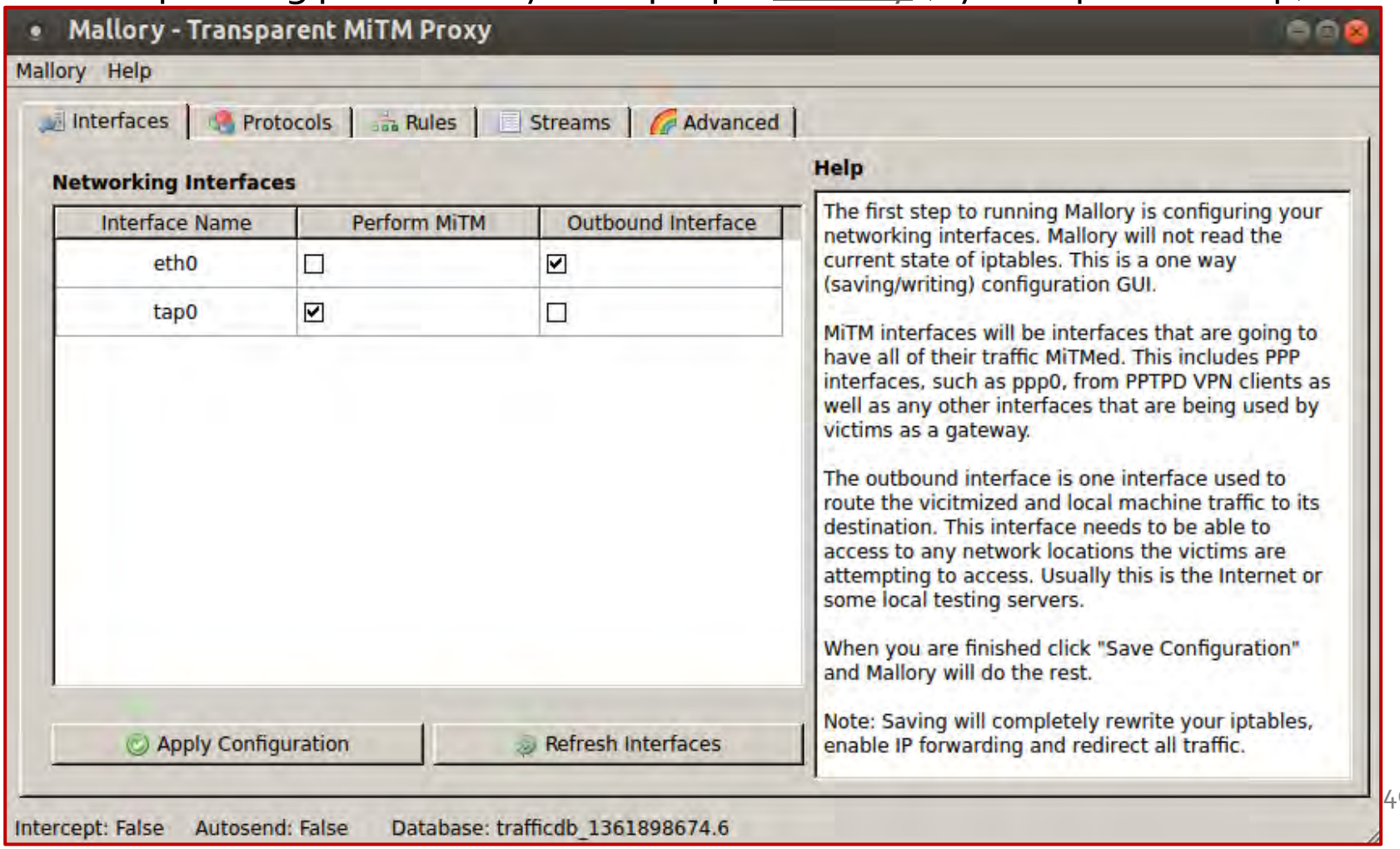
```
Linux~$ python prox.py -L 8888 <target IP> 25
```

- Note aside, Burp can also automatically redirect HTTP(S) traffic to a target IP and port



Intercept arbitrary TCP and UDP traffic

- Run a transparent TCP/UDP proxy capable of intercepting and manipulating packets on your laptop – Mallory (by Intrepidus Group)



Intercept arbitrary TCP and UDP traffic

- Set your laptop to be the gateway for the iDevice
  - Preferably by acting as a wireless access point and setting the iDevice to associate to it

- On your laptop, redirect all TCP and UDP traffic from the iDevice to your local instance of Mallory which will in turn forward it to the target server

```
Linux~$ sudo iptables -t nat -A PREROUTING -i at0 -p tcp \
-m tcp -j REDIRECT --to-ports 20755
Linux~$ sudo iptables -t nat -A PREROUTING -i at0 -p udp \
-m udp -j REDIRECT --to-ports 20755
```

  - `at0` is the virtual network interface on your laptop acting as wireless access point to the iDevice
  - `20755` is the TCP and UDP port where Mallory is listening on

All-in-one network traffic interception

- Firstly, set your iDevice to associate to your laptop's wireless access point

```
# echo 1 > /proc/sys/net/ipv4/ip_forward
# iptables -F
# iptables -F -t nat
# iptables --delete-chain
# iptables -t nat -A POSTROUTING -o eth0 -j MASQUERADE
# iptables -t nat -A PREROUTING -i at0 -p tcp -m tcp \
--dport 80 -j REDIRECT --to-ports 8080
# iptables -t nat -A PREROUTING -i at0 -p tcp -m tcp \
--dport 443 -j REDIRECT --to-ports 8080
# iptables -t nat -A PREROUTING -i at0 -p tcp -m tcp \
-j REDIRECT --to-ports 20755
# iptables -t nat -A PREROUTING -i at0 -p udp -m udp \
-j REDIRECT --to-ports 20755
```

Implement certificate pinning

- The security of the app's network communications can be improved through SSL pinning
  - Solves the rogue CA issue in a MiTM scenario

- Library developed by iSEC can be used – SSL Conservatory's `SSLCertificatePinning` class

```
NSData *certData = [TestSSLCertificatePinning
loadCertificateFromFile:@"www.targetapp.com.der"];
    if (certData == nil) {
        NSLog(@"Failed to load the certificates");
        return;
    }
    [domainsToPin setObject:certData forKey:@"www.targetapp.com"];
```

- OWASP also released a certificate pinning library

Circumvent certificate pinning

- Successful proxying SSL traffic can be difficult if the application performs **certificate pinning**

- Disable SSL certificate validation and accept all certificate chains
  - trustme – it replaces `SecTrustEvaluate` by Intrepidus Group
  - ios-ssl-killswitch – it hooks functions within the Secure Transport API by iSEC
    - It adds a new menu in the device's `Settings` where you can enable the extension

```
Linux~$ scp ios-ssl-kill-switch.deb root@<iDevice IP>
Linux~$ ssh root@<iDevice IP>

iPhone:~ root# dpkg -i ios-ssl-kill-switch.deb
iPhone:~ root# killall -HUP SpringBoard
```

Black-box assessment

- Application traffic analysis
- **Client / server assessment**
- Local data storage
- Keychain
- Logs
- Cache
- Inter-protocol communication (IPC)
- Binary analysis
- Runtime analysis

SSL certificate mismatch

- `NSURLConnection` by default, rejects the use of self-signed certificates
- Warning message `Cannot Verify Server Identity` is displayed when the SSL certificate provided is invalid
  - Developer corner: app needs to handle this case gracefully with NSURLConnection's `connection:didFailWithError`
- Sometimes developers override this built-in check
  - Developer corner: make sure the app does not do
- Check for calls to
  - NSURLRequest's private method `setAllowsAnyHTTPSCertificate`
  - NSURLConnection's delegation `continueWithoutCredentialForAuthenticationChallenge`

HTTP(S) responses caching

- HTTP and HTTPS requests are cached by default

- Developer corner: can be prevented using `NSURLConnection` delegate

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

HTTP cookies

- Manipulated by the URL loading system
- Developer can alter `cookieAcceptPolicy` to
  - `NSHTTPCookieAcceptPolicyNever`
  - `NSHTTPCookieAcceptPolicyOnlyFromMainDocumentDomain`
- Check if the target app set persistent cookies
  - Inspect HTTP(S) responses' `Set-Cookie` header in Burp Suite Proxy
  - Decode the locally stored persistent cookies

Persistent HTTP cookies decoding

- Persistent cookies are stored in a file called `Cookies.binarycookies` under
  - `/private/var/mobile/Library/`
  - `/private/var/mobile/<App GUID>/Library/Cookies`
- Use BinaryCookieReader.py script to decode cookies from this file
  - Verify the expiry date of the cookies
- Developer corner: avoid the use of persistent cookies

CFStreams sockets

- Lower-level network sockets
- Security defined by `kCFStreamPropertySSLSettings`
- Configurable, but hopefully you never see this

```
CFStringRef kCFStreamSSLLevel;
CFStringRef kCFStreamSSLAllowsExpiredCertificates;
CFStringRef kCFStreamSSLAllowsExpiredRoots;
CFStringRef kCFStreamSSLAllowsAnyRoot;
CFStringRef kCFStreamSSLValidatesCertificateChain;
CFStringRef kCFStreamSSLPeerName;
```

- Developer corner: only `kCFStreamSSLLevel` should be set to `kCFStreamSocketSecurityLevelTLSv1`

UIWebViews

- `UIWebView`'s are used to embed web content into an application
- An `UIWebView` object is created and the URL is passed to it
- The object will render the HTML with the ability to execute JavaScript
  - With `UIWebView::stringByEvaluatingJavaScriptFromString`
  - Potential for cross-site scripting (XSS)
- Developer corner: sanitization should be performed before passing data to the object. Avoid evaluating JavaScript if unnecessary

UDID and personal info leaking

- **U**nique **D**evice **Id**entifier derived from hardware information
- UDID cannot be removed or changed
  - It can be spooked with UDIDFaker tool
- UDID is exposed through API – deprecated with iOS 5
  - Check for calls to `[[UIDevice currentDevice] uniqueIdentifier]`
- Developer corner: do not rely on UDID for anything, ever
  - Use `[UIDevice currentDevice] identifierForVendor]`
  - Instance of NSUUID class
  - Guaranteed to return the same value when called by apps signed by the same developer certificate
  - Will change for a vendor if all of their apps are uninstalled

Address book leaking

- A number of apps have sent the entire address book of a user to a remote server in the past
- Developer corner: avoid the use of `ABAddressBrookCopyArrayOfAllPeople`
- You can use AdiOS to find all the apps that do call this function

Geolocation

- Developer corner: use the least degree of accuracy necessary and shouldn't log locally

- Ensure graceful handling of `locationServicesEnabled` and `authorizationStatus` method responses

- Several accuracy constants

```
CLLocationAccuracy kCLLocationAccuracyBestForNavigation;
CLLocationAccuracy kCLLocationAccuracyBest;
CLLocationAccuracy kCLLocationAccuracyNearestTenMeters;
CLLocationAccuracy kCLLocationAccuracyHundredMeters;
CLLocationAccuracy kCLLocationAccuracyKilometer;
CLLocationAccuracy kCLLocationAccuracyThreeKilometers;
```

XML injections

- Common to see XML snippets back and forth in the network traffic
- Fuzzing server responses may yield XML parsing vulnerabilities in the app
  - Check for calls to `NSXMLParser`
  - Handles DTD by default (XXE vulnerability)
- Based on `shouldResolveExternalEntities`, e.g.

```
NSURL *testURL = [NSURL URLWithString:@"http://target.com"];
NSXMLParser *testParser = [[NSXMLParser alloc] initWithContentsOfURL: testURL];
[testParser setShouldResolveExternalEntities: YES];
```

- Developer corner: ensure `setShouldResolveExternalEntities` is set to `NO`

Black-box assessment

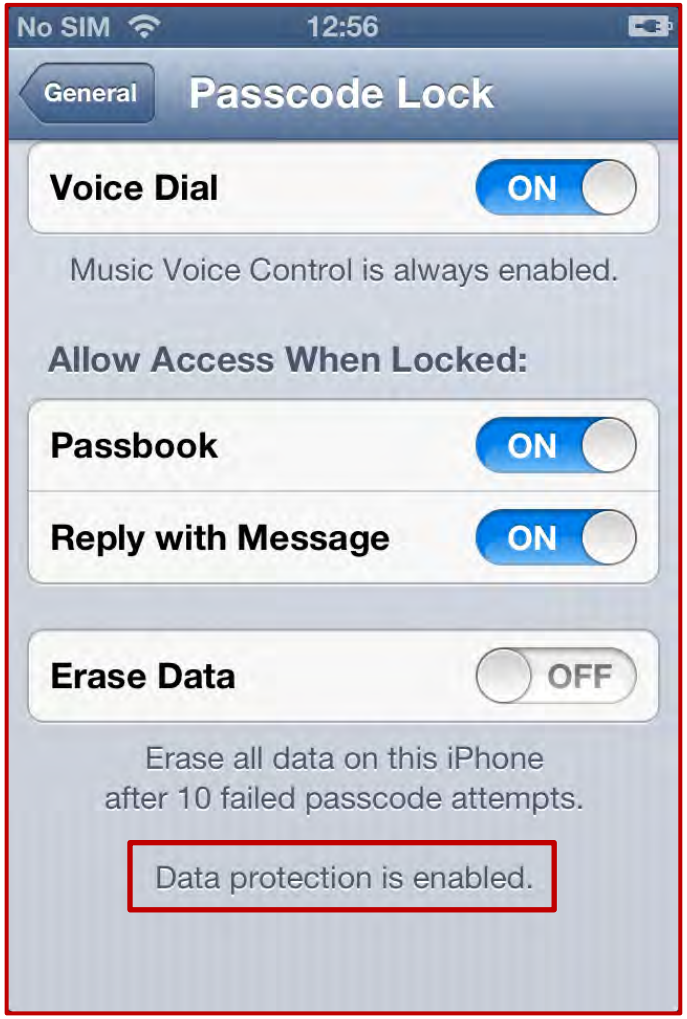
- Application traffic analysis
- Client / server assessment
- **Local data storage**
- Keychain
- Logs
- Cache
- Inter-protocol communication (IPC)
- Binary analysis
- Runtime analysis

Local data storage

- Data stored by the app to retain information across executions
- Information such as app config settings, preferences, media files, etc.
- Not uncommon to identify credentials and server's authentication tokens being stored in insecure manner
  - File storage
  - Property list files (`.plist` files)
  - SQLite databases (`.sqlite`, `.db`, `.sqlite3` files)
  - Core data

Data protection

- Data protection "*protects the user's data when the device is lost or stolen*"
  - API introduced in iOS 4
- Apps can tie the user's **passcode** to the mechanism to encrypt
  - **Files**
  - **Keychain entries**
- By setting up a device passcode, the user enables data protection



Data protection – file storage

- Set the `NSFileProtectionKey` attribute with the `NSFileProtectionComplete` value on `NSFileManager`

```
[[NSFileManager defaultManager] createFileAtPath:[self filePath]
  contents:[@"file content" dataUsingEncoding:NSUTF8StringEncoding]
  attributes:[NSDictionary dictionaryWithObject:NSFileProtectionComplete
forKey:NSFileProtectionKey]];
```

Protection class	Description
`NSFileProtectionComplete`	File only accessible when device unlocked
`NSFileProtectionCompleteUnlessOpen`	File must be open when device unlocked but accessible when unlocked
`NSFileProtectionCompleteUntilFirstUserAuthentication`	File protected until the device is unlocked after reboot

Data protection – file storage

- Alternatively, passed to calls to `NSData::writeToFile:options:error`

```
enum {
   NSDataWritingAtomic = 1UL << 0,
   NSDataWritingWithoutOverwriting = 1UL << 1,
   NSDataWritingFileProtectionNone = 0x10000000,
   NSDataWritingFileProtectionComplete = 0x20000000,
   NSDataWritingFileProtectionCompleteUnlessOpen = 0x30000000,
   NSDataWritingFileProtectionCompleteUntilFirstUserAuthentication = 0x40000000,
   NSDataWritingFileProtectionMask = 0xf0000000,
};
typedef NSUInteger NSDataWritingOptions;
```

File storage

- Check file attributes – sample code

```
NSString *fileProtectionValue = [[[NSFileManager defaultManager]
attributesOfItemAtPath:@"/path/to/file" error:NULL] valueForKey:
NSFileProtectionKey];
NSLog(@"NSFileProtectionKey: %@", fileProtectionValue);
```

- You can also check file attributes and operations during runtime analysis

Monitor file system operations

nccgroup
freedom from doubt

- Use filemon to monitor file system operations in real-time

```
Tests-iPhone:~ root# filemon.iOS
GOT PID: 403 and rc: 0 -  iGoat
iGoat (PID:403) Created  /private/var/mobile/Applications/7ED82FBE-8D70-4214-894C-7AE31F8BC92A/Documents/credentials.sqlite
DEV: 1,3 INODE: 204282 MODE: 81a4 UID: 501 GID: 501 Arg64: 7436062915

GOT PID: 403 and rc: 0 -  iGoat
iGoat (PID:403) Created  /private/var/mobile/Applications/7ED82FBE-8D70-4214-894C-7AE31F8BC92A/Documents/credentials.sqlite-journal
DEV: 1,3 INODE: 204283 MODE: 81a4 UID: 501 GID: 501 Arg64: 7436102800

GOT PID: 403 and rc: 0 -  iGoat
iGoat (PID:403) Modified  /private/var/mobile/Applications/7ED82FBE-8D70-4214-894C-7AE31F8BC92A/Documents/credentials.sqlite-journal
DEV: 1,3 INODE: 204283 MODE: 81a4 UID: 501 GID: 501 Arg64: 7436201903

GOT PID: 403 and rc: 0 -  iGoat
iGoat (PID:403) Deleted  /private/var/mobile/Applications/7ED82FBE-8D70-4214-894C-7AE31F8BC92A/Documents/credentials.sqlite-journal
DEV: 1,3 INODE: 204283 MODE: 81a4 UID: 501 GID: 501 Arg64: 7436211673

GOT PID: 403 and rc: 0 -  iGoat
iGoat (PID:403) Created  /private/var/mobile/Applications/7ED82FBE-8D70-4214-894C-7AE31F8BC92A/Documents/credentials.sqlite-journal
DEV: 1,3 INODE: 204284 MODE: 81a4 UID: 501 GID: 501 Arg64: 7436268310

GOT PID: 403 and rc: 0 -  iGoat
iGoat (PID:403) Modified  /private/var/mobile/Applications/7ED82FBE-8D70-4214-894C-7AE31F8BC92A/Documents/credentials.sqlite-journal
DEV: 1,3 INODE: 204284 MODE: 81a4 UID: 501 GID: 501 Arg64: 7436409553

GOT PID: 403 and rc: 0 -  iGoat
iGoat (PID:403) Deleted  /private/var/mobile/Applications/7ED82FBE-8D70-4214-894C-7AE31F8BC92A/Documents/credentials.sqlite-journal
DEV: 1,3 INODE: 204284 MODE: 81a4 UID: 501 GID: 501 Arg64: 7436416721

GOT PID: 403 and rc: 0 -  iGoat
iGoat (PID:403) Modified  /private/var/mobile/Applications/7ED82FBE-8D70-4214-894C-7AE31F8BC92A/Documents/credentials.sqlite
DEV: 1,3 INODE: 204282 MODE: 81a4 UID: 501 GID: 501 Arg64: 7436425717
```

Property list files

- Used to store application configuration and user preferences
  - Sometimes used to store confidential information such as credentials
- Stored within the application's `Library/Preferences/` directory
- They are clear-text XML or structured serialized format
  - Use plutil `-convert xml1` to convert the serialized into human-readable format

```
Tests-iPhone:/private/var/mobile/Applications/7ED82FBE-8D70-4214-894C-7AE31F8BC92A/Library/Preferences root# cat com.krvw.iGoat.plist
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE plist PUBLIC "-//Apple//DTD PLIST 1.0//EN" "http://www.apple.com/DTDs/PropertyList-1.0.dtd">
<plist version="1.0">
<dict>
        <key>WebDatabaseDirectory</key>
        <string>/var/mobile/Applications/7ED82FBE-8D70-4214-894C-7AE31F8BC92A/Library/Caches</string>
        <key>WebKitDiskImageCacheSavedCacheDirectory</key>
        <string></string>
        <key>WebKitLocalStorageDatabasePathPreferenceKey</key>
        <string>/var/mobile/Applications/7ED82FBE-8D70-4214-894C-7AE31F8BC92A/Library/Caches</string>
        <key>WebKitOfflineWebApplicationCacheEnabled</key>
        <true/>
        <key>WebKitShrinksStandaloneImagesToFit</key>
        <true/>
        <key>password</key>
        <string>wwwww</string>
        <key>username</key>
        <string>donkeyw</string>
</dict>
</plist>
```

Property list files

- `NSUserDefaults` class provides methods to create, access and modify property files

- Typical insecure use of `NSUserDefaults`

```
NSString *creditCardNumber = [...];

[[NSUserDefaults standardUserDefaults] setObject:creditCardNumber
forKey:@"cc-number"];
[[NSUserDefaults standardUserDefaults] synchronize];
```

- Check all calls made to `NSUserDefaults` and content of property files to ensure no confidential information is stored onto the iDevice

- Developer corner: use **keychain** to store confidential information

SQLite databases

- SQLite is a file-based database
  - Sometimes used to store confidential information such as credentials
- Stored within the application's `Documents/` directory
- Use `sqlite3` tool to access, query and edit SQLite files

```
Tests-iPhone:/var/mobile/Applications/7ED82FBE-8D70-4214-894C-7AE31F8BC92A/Documents root# sqlite3 credentials.sqlite
SQLite version 3.7.13
Enter ".help" for instructions
sqlite> .tables
creds
sqlite> .schema
CREATE TABLE creds (id INTEGER PRIMARY KEY AUTOINCREMENT, username TEXT, password TEXT);
sqlite> .headers ON
sqlite> SELECT * FROM creds;
id|username|password
1|user|pass
2|foobar|secretpass
sqlite>
```

SQLite and SQL injection

- Check all calls made to `sqlite3_*` methods
- Avoid querying the SQLite database like this – bad dynamic statement

```
NSString *uid = [myHTTPConnection getUID];
NSString *statement = [NSString StringWithFormat:@"SELECT
     username FROM users where uid = '%@'", uid];
const char *sql = [statement UTF8String];
```

- One-liner to list the database scheme of all SQLite files

```
Tests-iPhone:~ root# find / -name "*.sqlite" -exec echo {} \; -
exec sqlite3 -batch {} ".schema" \;
```

- Developer corner: use parameterized queries

```
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);
```

Core data

- Way to store persistent data inside iOS without having to worry with the management
  - Easily accessible and manageable at the same time
  - In the form of SQLite files
  - Difference with others SQLite databases is that all tables start with `Z`
- Defective assumption: Apple takes care of the security
- Check calls to `NSManagedObjectContext`
- Developer corner: use **keychain** to store confidential information

Black-box assessment

- Application traffic analysis
- Client / server assessment
- Local data storage
- **Keychain**
- Logs
- Cache
- Inter-protocol communication (IPC)
- Binary analysis
- Runtime analysis

Keychain

- **Secure storage** of passwords, keys, certificates, and notes for one or more users
  - Encrypted with device and keybag specific keys

- In iOS, there is a **single keychain** that stores keychain items for all apps
  - Each app has only access to its own keychain items
  - `/private/var/Keychains/keychain-2.db`

Keychain table	Description
`genp`	Generic passwords – `kSecClassGenericPassword`
`inet`	Internet passwords – `kSecClassInternetPassword`
`cert` and `keys`	Certificates, keys and digital identity (cert+key) items – `kSecClassCertificates` and `kSecClassIdentity`

Data protection – keychain

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

- Key accessibility is defined by **data protection**
  - Pass appropriate `kSecAttrAccessible` value to `SecItemAdd` and `SecItemUpdate`

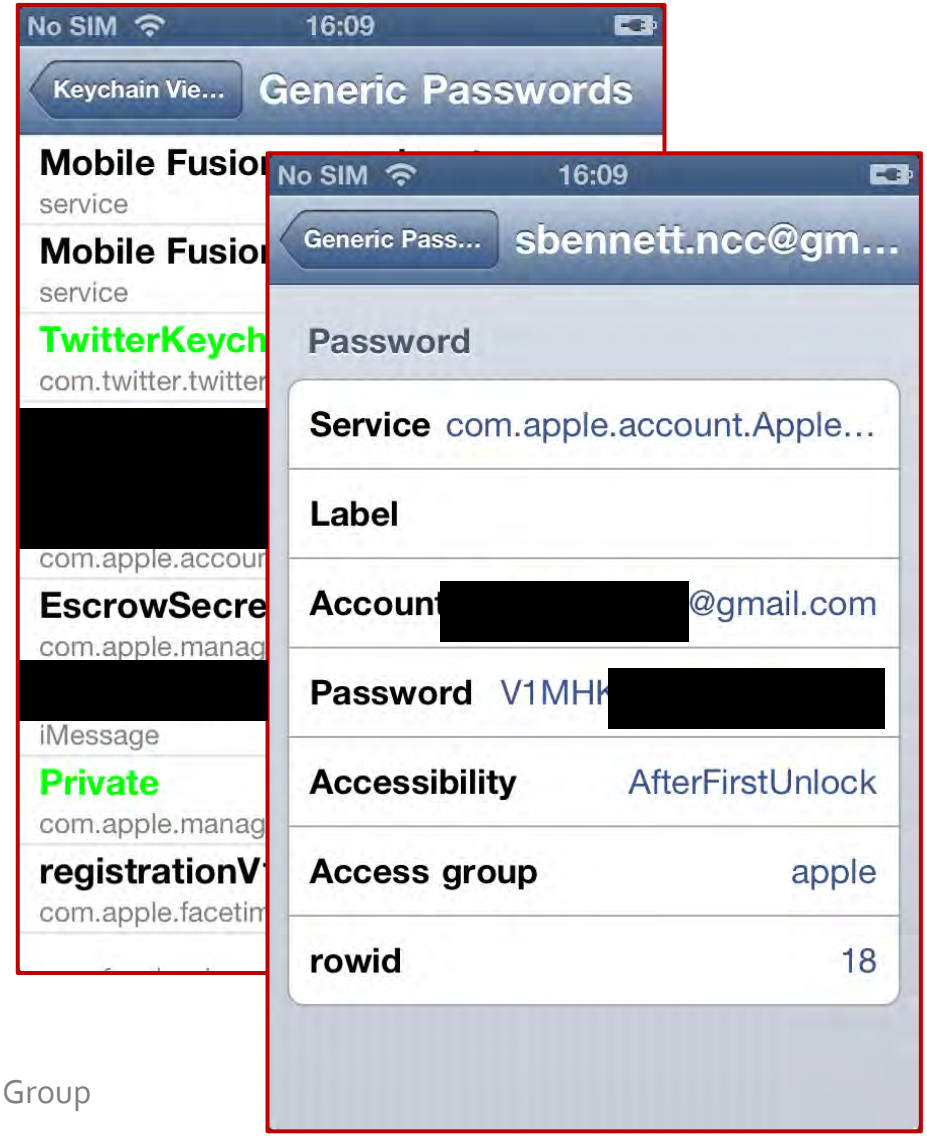
```
CFTypeRef kSecAttrAccessibleWhenUnlocked; // Only when the device is unlocked
CFTypeRef kSecAttrAccessibleAfterFirstUnlock; // After user enters passcode once
CFTypeRef kSecAttrAccessibleAlways; // Following boot
CFTypeRef kSecAttrAccessibleWhenUnlockedThisDeviceOnly; // Omitted from backups
CFTypeRef kSecAttrAccessibleAfterFirstUnlockThisDeviceOnly;
CFTypeRef kSecAttrAccessibleAlwaysThisDeviceOnly;
```

  - `ThisDeviceOnly` value denies iTunes backup to export the entry to other devices

Dump keychain items

- App signed with entitlement file with a wildcard or with `com.apple.keystore.access-keychain-keys` as keychain access group can access all keychain items

- You can use a number of tools to dump keychain items
  - keychain_dumper – command line tool to list keychain items
  - keychain_dump – command line tool to export to human-readable Plist files the keychain items
  - KeychainViewer – graphical app to list keychain items



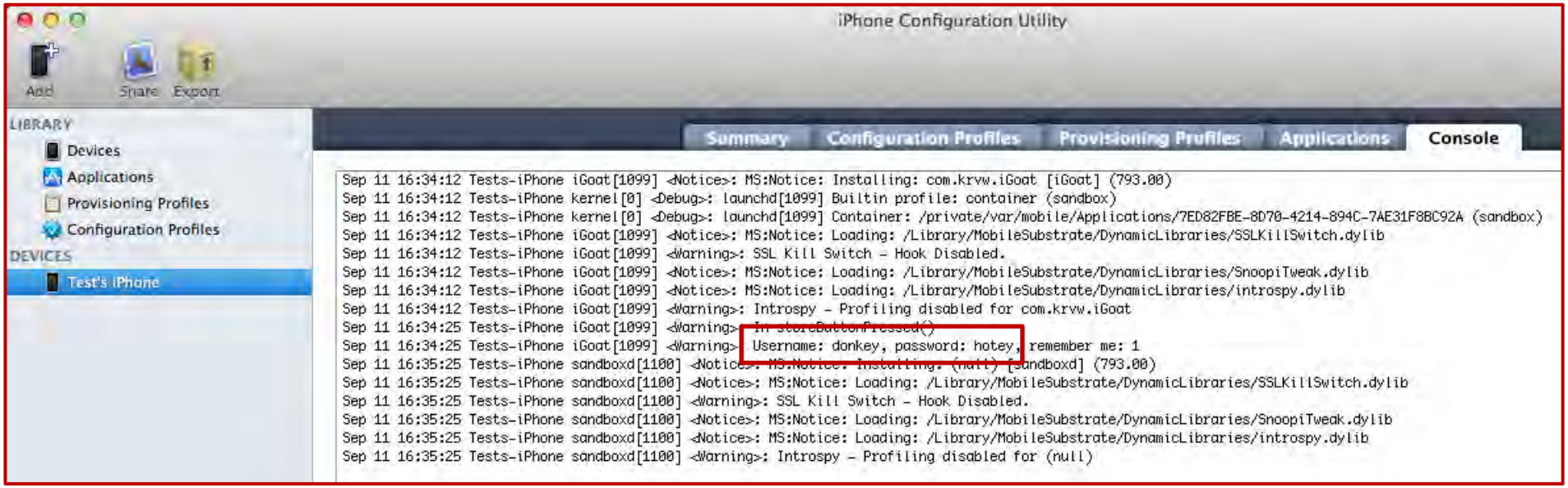
Black-box assessment

- Application traffic analysis
- Client / server assessment
- Local data storage
- Keychain
- **Logs**
- Cache
- Inter-protocol communication (IPC)
- Binary analysis
- Runtime analysis

Logs

- `Console` tab of the iPhone Configuration Utility
  - Sometimes developers forget debug `NSLog` calls disclosing sensitive information to the Apple system log

```
NSLog(@"Username: %@, password: %@, remember me: 1", myName, myPass);
```



- Install Cydia's `syslogd` tool to store logs to `/var/log/syslog` too

Logs

- App Store's `Console` and `System Console` apps allow to read and search logs too

- Developer corner: ensure that the app does not disclose any sensitive information into Apple system log or disable `NSLog` in non-debug builds

```
#ifdef DEBUG
# define NSLog (...) NSLog(_{VA_ARGS_ {);
#else
# define NSLog (...)
#endif
```

- iTunes sync retrieve logs, including **crash reports**
  - These may yield valuable information

```
C:\Users\<username>\AppData\Roaming\Apple
Computer\Logs\CrashReporter\MobileDevice\<DEVICE NAME>\

/Users/<username>/Library/Logs/CrashReporter/MobileDevice/<DEVICE NAME>/
```

Black-box assessment

- Application traffic analysis
- Client / server assessment
- Local data storage
- Keychain
- Logs
- **Cache**
- Inter-protocol communication (IPC)
- Binary analysis
- Runtime analysis

UIPasteboard

- Allows app to share data with other apps
  - `UIPasteboardNameGeneral`, `UIPasteboardNameFind`

- Developer corner
  - Limit the lifetime of data on the pasteboard
  - If app only needs to copy/paste internally, wipe the pasteboard on `applicationDidEnterBackground` and `applicationWillTerminate` by setting `pasteBoard.items = nil`
  - Disable copy/paste menu for sensitive data fields

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

Backgrounding – state transition

- Screenshot is taken right before the application is backgrounded
  - `<Application GUID>/Library/Caches/Snapshots/*/*.png`

- Developer corner: prior to this, app should remove any sensitive data from view

  - Set `windows.hidden` properties to `YES` of `UIWindow` in the `applicationDidEnterBackground` delegate and set `windows.hidden` properties to `NO` in the `applicationWillEnterForeground` delegate

  - Set `hidden` attribute on sensitive fields

Backgrounding – state preservation

- iOS 6 introduces the concept of **state preservation**
  - application:shouldSaveApplicationState
- Saves state of views and view controllers tagged with a `restorationIdentifier`
- Theoretically can be done safely using `willEncodeRestorableStateWithCoder` and `didDecodeRestorableStateWithCoder` delegates
- Developer corner: implement an `NSCoder` to perform cryptographic operations and store key in keychain

Keyboard cache

- iOS logs keystrokes to provide customized auto-correction, form completion and other features
  - Almost every non-numeric word is cached
- Stored in `/var/mobile/Library/Keyboard/en_GB-dynamic-text.dat`
- Cache content beyond the app developer realm
- Developer corner: for any sensitive `UITextField` and `UISearchBar` mark them as secure fields and disable auto correction

```
fieldName.secureTextEntry = YES
fieldName.autocorrectionType = UITextAutocorrectionTypeNo
```

Black-box assessment

- Application traffic analysis
- Client / server assessment
- Local data storage
- Keychain
- Logs
- Cache
- **Inter-protocol communication (IPC)**
- Binary analysis
- Runtime analysis

Inter-protocol communication (IPC)

- IPC is performed by registering URL schemes handled by `UIApplicationDelegate::openURL:(NSURL *)url sourceApplication:(NSString *)sourceApplication annotation:(id)annotation`
  - Allows for determining calling application, receives data in plist form
- Other applications make requests using those schemes in order to invoke the registering application

```
openURL:[NSURL URLWithString:@"myapp://?un=foo&pw=test"];
```

- Web pages also can call URL handlers, without confirmation

URL scheme conflict

- *"If more than one third-party app registers to handle the same URL scheme, there is currently no process for determining which app will be given that scheme"* – Apple

- Developer corner: be wary of passing private data in app URLs

Black-box assessment

- Application traffic analysis
- Client / server assessment
- Local data storage
- Keychain
- Logs
- Cache
- Inter-protocol communication (IPC)
- **Binary analysis**
- Runtime analysis

App binary encryption

- Apps from the App Store are encrypted
  - FairPlay DRM
  - Does not apply to Apple built-in apps under `/Applications/`
- To successfully analyse the binary, you need first to **decrypt** it
- Firstly, ensure that it is encrypted by inspecting the Mach-O binary header with `otool` utility

```
Tests-iPhone:/var/mobile/Applications/84593008-5B13-4CFC-BCEB-727DDC29DD50/Twitter.app root# otool -l Twitter | grep LC_ENCRYPTION_INFO -A 4
          cmd LC_ENCRYPTION_INFO
      cmdsize 20
 cryptoff  8192
 cryptsize 5476352
 cryptid   1
```

Encrypted

Decrypt app binary

- Locate the encrypted segment

```
Tests-iPhone:/var/mobile/Applications/DC9C5245-7304-4778-BD6F-AB502CD9FCCC/Twitter.app root# otool -l Twitter | grep LC_ENCRYPTION_INFO -A 4
          cmd LC_ENCRYPTION_INFO
      cmdsize 20
 cryptoff  8192
 cryptsize 5476352
 cryptid   1
```

- The encrypted segment starts at `0x2000` (8192)
- The segment is `0x539000` bytes (cryptsize `5476352`)

Decrypt app binary

- Run the app under a debugger (gdb) to dump the decrypted segments before the app runs

```
Tests-iPhone:/var/mobile/Applications/DC9C5245-7304-4778-BD6F-AB502CD9FCCC/Twitter.app root# gdb -q ./Twitter
Reading symbols for shared libraries . done
(gdb) rb doModInitFunctions
Breakpoint 1 at 0x2fe0d48a
<function, no debug info> __dyld__ZN16ImageLoaderMachO18doModInitFunctionsERKN11ImageLoader11LinkContextE;
(gdb) r
Starting program: /private/var/mobile/Applications/DC9C5245-7304-4778-BD6F-AB502CD9FCCC/Twitter.app/Twitter
Reading symbols for shared libraries + done
Removing symbols for unused shared libraries . done
Reading symbols for shared libraries ..................................................................
............................ done

Breakpoint 1, 0x2fe8748a in __dyld__ZN16ImageLoaderMachO18doModInitFunctionsERKN11ImageLoader11LinkContextE ()
(gdb) dump memory twitter-decrypted-segments.bin 0x2000 0x53b000
(gdb) quit
The program is running.  Exit anyway? (y or n) y
```

Decrypt app binary – repackaging

- Transpose the decrypted segments into a copy of the original binary

```
Tests-iPhone:/var/mobile/Applications/DC9C5245-7304-4778-BD6F-AB502CD9FCCC/Twitter.app root# dd bs=1 seek=8192 conv=notrunc if=twitter-decrypte
5476352+0 records in
5476352+0 records out
5476352 bytes (5.5 MB) copied, 149.099 s, 36.7 kB/s
```

- Patch the binary's `Load commands` header's `cryptid` value to `0`
  - You can use vbindiff from command line or MachOView from OSX

```
lqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqq Find Hex Bytesqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqk
x 21 00 00                                                                         x
mqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqj
```

```
Twitter
0000 0A24: 21 00 00 00 14 00 00 00  00 20 00 00 00 90 53 00  !....... . ....S.
0000 0A34: 00 00 00 00 0C 00 00 00  34 00 00 00 18 00 00 00  ........ 4.......
0000 0A44: 02 00 00 00 00 06 09 00  00 00 09 00 2F 75 73 72  ........ ..../usr
0000 0A54: 2F 6C 69 62 2F 6C 69 62  73 71 6C 69 74 65 33 2E  /lib/lib sqlite3.
0000 0A64: 64 79 6C 69 62 00 00 00  0C 00 00 00 30 00 00 00  dylib... ....0...
```

Decrypt app with ClutchPatched

- ClutchPatched (com.sull.clutchpatched)

```
Tests-iPhone:~ root# Clutch Twitter
Cracking Twitter...
Creating working directory...
Performing initial analysis...
Performing cracking preflight...
yolofat magic 4277009102
Application is a thin binary, cracking single architecture...
dumping binary: analyzing load commands
found vmaddr
found LC_ENCRYPTION
found LC_CODE_SIGNATURE
dumping binary: obtaining ptrace handle
dumping binary: forking to begin tracing
dumping binary: obtaining mach port
dumping binary: preparing code resign
dumping binary: preparing to dump
dumping binary: ASLR enabled, identifying dump location dynamically
dumping binary: performing dump
dumping binary: patched cryptid
dumping binary: writing new checksum
Packaging IPA file...
        /var/root/Documents/Cracked/Twitter-v5.10.1.ipa
```

Decrypt app with dumpdecrypted

- dumpdecrypted by Stefan Esser

```
Tests-iPhone:~ root# DYLD_INSERT_LIBRARIES=dumpdecrypted.dylib /var/mobile/Applications/3CDD2F16-6346-4E5A-8885-A3
mach-o decryption dumper

DISCLAIMER: This tool is only meant for security research purposes, not for application crackers.

[+] offset to cryptid found: @0xaa34(from 0xa000) = a34
[+] Found encrypted data at address 00002000 of length 5476352 bytes - type 1.
[+] Opening /private/var/mobile/Applications/3CDD2F16-6346-4E5A-8885-A32A48C64E40/Twitter.app/Twitter for reading.
[+] Reading header
[+] Detecting header type
[+] Executable is a plain MACH-O image
[+] Opening Twitter.decrypted for writing.
[+] Copying the not encrypted start of the file
[+] Dumping the decrypted data into the file
[+] Copying the not encrypted remainder of the file
[+] Setting the LC_ENCRYPTION_INFO->cryptid to 0 at offset a34
[+] Closing original file
[+] Closing dump file
Tests-iPhone:~ root#
```

Binary analysis – otool

- otool can be used to inspect the Objective-C segment (__OBJC)
- Reveals class names, method names, and instance variables

```
# otool -oV DecryptedApp
```

- Can also be used to disassemble the text segment

```
# otool -tV DecryptedApp
```

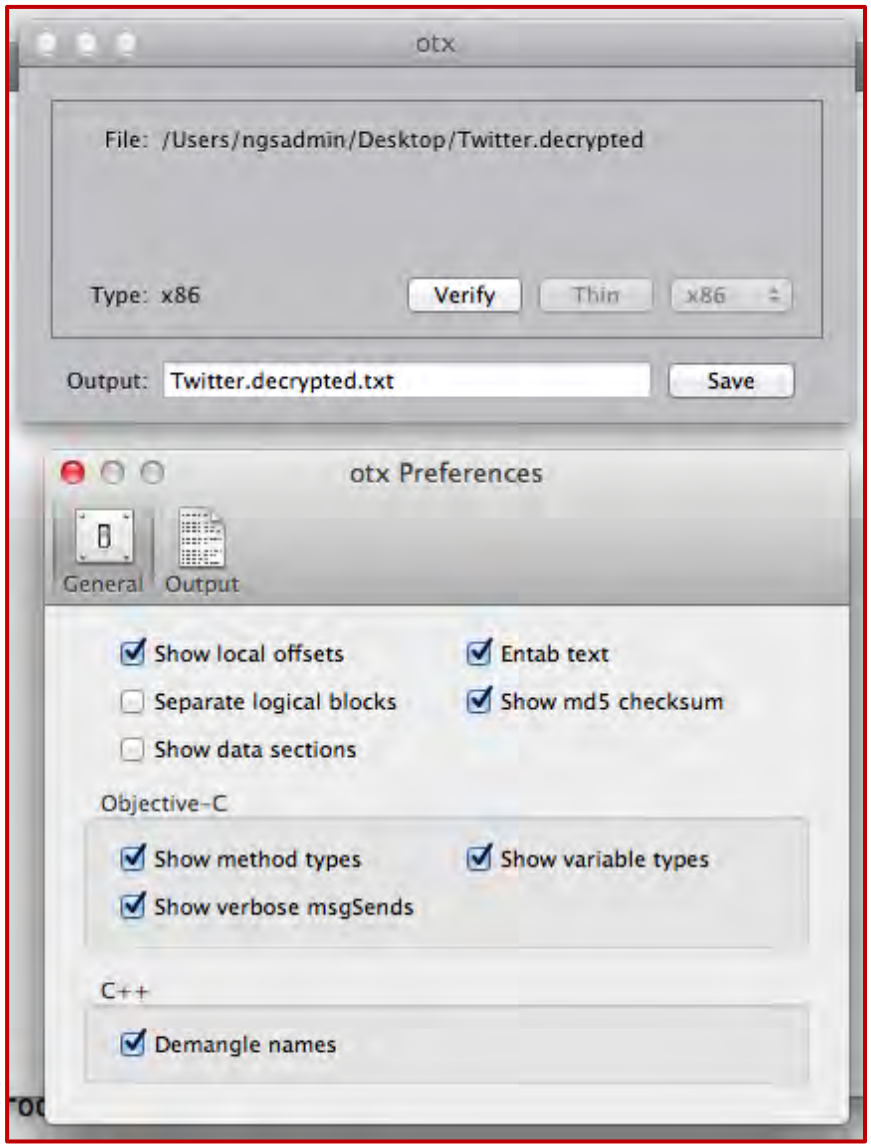
- Check also exported symbols

```
# otool -IV DecryptedApp
```

- Recommended using additional tools for disassembly
  - Hopper is a good (and cheap) alternative to IDA Pro

Binary analysis – otx

- otx – object tool extended
- Uses otool to disassemble a Mach-O executable file
- Enhances the disassembled output
  - Display names and data types of Obj-C methods even if symbols have been stripped
  - Descriptive comments to member variables
  - Etc.



Check for exploit mitigations

- Check for PIE (Position Independent) flag from mach header

```
Tests-iPhone:~ root# otool -hv DecryptedApp
DecryptedApp:
Mach header
      magic cputype cpusubtype  caps    filetype ncmds sizeofcmds
flags
   MH_MAGIC     ARM          9  0x00     EXECUTE    51       5500
NOUNDEFS DYLDLINK TWOLEVEL BINDS_TO_WEAK PIE
```

- Check for stack smashing protection ("stack canary") from symbol table

```
Tests-iPhone:~ root# otool -Iv DecryptedApp | grep __stack
0x0040804c 148284 ___stack_chk_fail
0x0053c048 148285 ___stack_chk_guard
0x0053d4d0 148284 ___stack_chk_fail
```

Check for exploit mitigations

- Check for Automatic Reference Counting flag from mach header

```
Tests-iPhone:~ root# otool -Iv DecryptedApp | grep _objc_release
0x00407cac 148535 _objc_release
0x0053d3e8 148535 _objc_release
```

- Symbols that prove the use of ARC
  - `__objc_retainAutoreleaseReturnValue`
  - `__objc_autoreleaseReturnValue`
  - `__objc_storeStrong`
  - `__objc_retain`
  - `__objc_release`
  - `__objc_retainAutoreleasedReturnValue`

Objective-C runtime information

- class-dump-z is used to examine the Objective-C runtime information stored in Mach-O binaries
- Generates similar output to `otool -oV`
- Output presented as normal Objective-C declerations
  - Easier to read and becomes handy when writing MobileSubstrate hooks

```
Tests-iPhone:~ root# class-dump-z DecryptedApp > DecryptedApp
Tests-iPhone:~ root# less DecryptedApp
[…]
@interface NSData (Keychain)
+ (void)deleteKeychainData:(id)arg1 account:(id)arg2;
+ (id)dataFromKeychain:(id)arg1 account:(id)arg2;
- (BOOL)storeInKeychain:(id)arg1 account:(id)arg2 accessibility:(void *)arg3;
@end
```

Black-box assessment

- Application traffic analysis
- Client / server assessment
- Local data storage
- Keychain
- Logs
- Cache
- Inter-protocol communication (IPC)
- Binary analysis
- **Runtime analysis**

Runtime analysis and manipulation

- Running apps can be extended with additional debugging and runtime tracing capabilities
  - Injecting a library with new functionality
  - Injecting an interpreter for on-the-fly manipulation

- MobileSubstrate: framework that allows third-party developers to develop runtime patches (extensions) to system and application functions
  - **MobileLoader:** loads third-party patching code into the running application
  - **MobileHooker:** hooks and replaces methods and functions

- Runtime manipulation can be used to
  - Bypass client-side controls
  - Execute hidden functionality
  - Unlock premium content
  - Etc.

Runtime manipulation example

- Example of MobileHooker to replace a function

```
static int (* orig_dladdr)(const void *, Dl_info *);
int replaced_dladdr(const void *addr , Dl_info *info) {
  IMP impl = class_getMethodImplementation(objc_getClass("JailBreakSecurity"),
    @selector(isFunctionValid:));
  return ((uintptr_t)addr == (uintptr_t)impl) ? 1 : 0;
}
MSHookFunction((void *)dladdr , (void *)replaced_dladdr, (void **)&orig_dladdr);
```

Runtime manipulation with theos

- Theos: set of tools for working with iOS apps outside of Xcode
  - Logos is a built-in preprocessor-based library of directives designed to make MobileSubstrate extension development easy

- Example

```
%group NSURLConnectionHooks
%hook NSURLConnection
- (id)initWithRequest:( NSURLRequest *)request delegate:(id <
     NSURLConnectionDelegate >)delegate {
        NSLog(@"Requesting: %@", [[request url] absoluteString]);
        return %orig(request, delegateProxy);
}
%end
%ctor {
        %init(NSURLConnectionHooks);
}
```

Runtime analysis with cycript

- JavaScript interpreter which understands Objective-C syntax
- Runtime injection and modification of control flow
- Hook into a running process

```
Tests-iPhone:~ root# ps -ef | grep -i iGoa
  501   907     1   0   0:00.00 ??         0:01.21 /var/mobile/Applications/7ED82FBE-8D70-4214-894C-7AE31F8BC92A/iGoat.app/
    0   909   312   0   0:00.00 ttys000    0:00.01 grep -i iGoa
Tests-iPhone:~ root# cycript -p 907
cy#
```

- Application instance (`UIApplication sharedApplication`) stored by default in `UIApp`
- Get the instance of the view controller for the key window run `UIApp.keyWindow.rootViewController`

Runtime analysis with cycript

- Use class-dump-z to dump class information, list the view controller instance name (e.g. `AppNameViewController`)

- Scroll down in the output and look for the properties of the active window – example

```
[...]
@property(assign, nonatomic) __weak UITextField* username;
@property(assign, nonatomic) __weak UITextField* password;
@property(assign, nonatomic) __weak UILabel* result;
[...]
```

- Use this info to access these properties – example

```
cy# UIApp.keyWindow.rootViewController.username
@"<UITextField: ...; text = 'foobar'; ...>"
```

Runtime analysis with cycript

- Modify property value – example

```
cy# UIApp.keyWindow.rootViewController.result.text = "Valid
credentials!"
```

- List instance variables (iVars) of a specific object

```
cy# function tryPrintIvars(a){ var x={}; for(i in *a){ try{
x[i] = (*a)[i]; } catch(e){} } return x; }
Example:
cy# tryPrintIvars(UIApp.keyWindow.rootViewController)
```

- Access directly the value of an iVar

```
cy# UIApp.keyWindow.rootViewController->userInput
```

Runtime analysis with cycript

- List class methods

```
function printMethods(className) {
  var count = new new Type("I");
  var methods =
class_copyMethodList(objc_getClass(className), count);
  var methodsArray = [];
  for(var i = 0; i < *count; i++) {
    var method = methods[i];
    methodsArray.push({selector:method_getName(method),
implementation:method_getImplementation(method)});
  }
  free(methods);
  free(count);
  return methodsArray;
}

cy# printMethods("AppNameViewController")
[{selector:@selector(setCredentials:),...},{selector:...}]
```

Runtime analysis with cycript

- Replace class methods (hook) – need to get to its metaclass

```
cy# UIApp.keyWindow.rootViewController-
>isa.messages['validateCredentials:'] = function() { return true; }
function() {return true;}
```

- Now you can either execute the functionality that calls the `validateCredentials` method from your iDevice screen or execute it directly from cycript – example

```
cy# [UIApp.keyWindow.rootViewController validateCredentials]
```

Introspy

- Introspy is an open-source security profiler for iOS by iSEC Partners
- Two separate components
- **iOS tracer**
  - On jailbroken iDevice
  - Hook security-sensitive APIs called by given app
  - Records details on a SQLite database
- **Analyzer**
  - Analyzes the details stored in the SQLite database, offline
  - Generates an HTML report displaying recorded calls
  - Lists potential vulnerabilities

Introspy menu – apps

Introspy menu – settings

Introspy – log to the console

Organizer – Devices

Devices | Repositories | Projects | Archives | Documentation

LIBRARY
- Provisioning Profiles
- Software Images
- Device Logs
- Screenshots

DEVICES
- My Mac 10.8.4 (12E55)
  - Provisioning Profiles
- iPad Mini 6.1 (10B141)
  - Provisioning Profiles
  - Applications
  - Console
  - Device Logs
  - Screenshots
- isec-test-0309 6.1 (10B141)
  - Device Logs
  - Screenshots

```
            alg = 0;
            cryptorRef = 803202968;
            data = <73336372 00549c02 32330000 00000000>;
            dataUsed = 12;
            iv = <49564956 49564956 00000000 00000000>;
            key = <4b657920 31323300 00000000 00000000>;
            op = 0;
            options = 1;
        };
        returnValue = 0;
    }
    ---------------
Aug 14 13:39:50 iPad-Mini IntrospyTestApp[12232] <Warning>:
    -----INTROSPY-----
    CALLED C CCCryptorCreate
    WITH:
    {
        arguments =     {
            alg = 0;
            cryptorRef = 803202968;
            iv = <49564956 49564956 00000000 00000000>;
            key = <4b657920 31323300 00000000 00000000>;
            op = 0;
            options = 1;
        };
        returnValue = 0;
    }
    ---------------
Aug 14 13:39:50 iPad-Mini IntrospyTestApp[12232] <Warning>:
    -----INTROSPY-----
    CALLED C CCCryptorUpdate
    WITH:
    {
        arguments =     {
            cryptorRef = 10386944;
            dataIn = <73336372 00549c02 32330000 00000000>;
            dataOut = <d06a0000 b46a0000 9ce7df2f 671f1539>;
        };
        returnValue = 4294962995;
    }
    ---------------
Aug 14 13:39:50 iPad-Mini IntrospyTestApp[12232] <Warning>:
    -----INTROSPY-----
    CALLED C CCCryptorFinal
    WITH:
    {
        arguments =     {
            cryptorRef = 10386944;
            dataOut = <ff370362 85e66c53 961313d0 134add0b>;
```

Clear | Save Log As...

Introspy analyzer

- Python script
- Recover the tracer's SQLite database from the iDevice

```
$ python introspy.py 172.16.47.8 -o iGoat
mobile@172.16.47.8's password:
0. ./Applications/7ED82FBE-8D70-4214-894C-7AE31F8BC92A/introspy-
com.krvw.iGoat.db
Select the database to analyze: 0
scp mobile@172.16.47.8:././Applications/7ED82FBE-8D70-4214-894C-
7AE31F8BC92A/introspy-com.krvw.iGoat.db ./
mobile@172.16.47.8's password:
```

- Process a SQLite database generated by the tracer
- Output various information about the traced calls
- Generate an HTML report

Introspy analyzer – report

Introspy HTML Report

file://localhost/Users/nabla/Documents/git/github/introspy/analyzer/test/testapp%202/report....

Traced Calls

Potential Findings

Show / Hide | Show All | Hide All | Data Storage | Crypto | Network | IPC | Misc

- User Preferences - 65
- Keychain - 24
- Filesystem - 81

83: C CCCrypt

Arguments:


```
{
  "alg": 0,
  "dataIn": "czNjcgCopwAyMwAAAAAAAA==",
  "iv": "IVIVIVIV\u0000\u0000\u0000\u0000\u0000\u0000\u0000\u0000",
  "dataOut": "/zcDYoXmbFOWExPQE0rdCwC+pwDT+ZlYmHEAANjWYDQ=",
  "dataOutAvailable": 16,
  "key": "Key 123\u0000\u0000\u0000\u0000\u0000\u0000\u0000\u0000\u0000",
  "options": 1,
  "op": 0
}
```


Return Value:

```
4294962995
```

84: C CCKeyDerivationPBKDF

85: NSURLConnection initWithRequest:delegate:

86: NSURLConnection initWithRequest:delegate:startImmediately:

Introspy analyzer – report

Introspy HTML Report

file://localhost/Users/nabla/Documents/git/github/introspy/analyzer/test/testapp%202/report....

Traced Calls

Potential Findings

Vulnerable XML Parser

Client Certificate Import

Severity

Informational

Description

The application imported a private key and a certificate from a PKCS12 file.

Relevant function calls

100: C SecPKCS12Import

Arguments:


```
{
  "pkcs12_data": "MIIHAQIBAzCCBscGCSqGSIb3DQEHAaCCBrgEgga0MIIGsDCCA68GCSqGSIb3DQEHBqCCA6A
wggOcAgEAMIIDlQYJKoZIhvcNAQcBMBwGCiqGSIb3DQEMAQYwDgQIEpitfp8eloICAggAgIIDaL/dYUnOlVFEAjSy
LaJHe6CZkYkOmZXdqo0xO0b8H/V9jKFjGxSL2LCuOyKQ98yif3njT/rjthKzaQoxgz47+1UO/ijKcdydHGGWy5AV5
xqPwf3OLV2+LrQBDvqwmAMOI469aJtZwV2sx9IjjFf9kKAVZ8Rknvi+Znv5uShAcoSwkJTeU2Kttu9qfISUq9i3zl
LJFcatRssk5U3MFPPGavpukFSvdzx7vgwiEr+tJGJUM3tK0df2z6yrEQd5UoK+NFo7+oyQgHN9PbAr5EDU3lnYy6G
qtDOxIkUX051QWXGEVGUAGRww6fahUrNQHtS0zxFvh3KJXFz9GXQmpOpHhGhJS2USQy9iWdtWsNTT0U4mWUeEaYzd
```

Outline

- Introduction to iOS and Objective-C
- Platform security
- iOS apps
- Testing environment
- Black-box assessment
- **Conclusion**

Conclusion for developers

- Learn about security-oriented Cocoa Touch API objects
  - Prefer use of C for security-critical implementations
- Least data and logic possible on the client-side
  - Security enforced server-side, not by the app's logic
  - As in JavaScript client-side controls for web applications

References

nccgroup
freedom from doubt

- July 11, 2013 - Pentesting iOS Apps - Runtime Analysis and Manipulation by Andreas Kurtz
- February 22, 2013 - End-to-End Mobile Security by NCC Group
- January 11, 2013 - Inspecting Class Information at runtime for Encrypted iOS Applications by Jason Haddix
- November 29, 2012 - Pentesting iOS Apps - Runtime Analysis and Manipulation by Andreas Kurtz
- October 2, 2012 - iOS Security by Apple
- August 2, 2012 - The Dark Art of iOS Application Hacking by Jonathan Zdziarski
- July 14, 2012 - When Security Gets in the Way: PenTesting Mobile Apps That Use Certificate Pinning by Justine Osborne and Alban Diquet (NCC Group)
- May 2, 2012 - iOS Application (In)Security by Dominic Chell
- March 28, 2012 - Debunking NSLog Misconceptions by Ron Gutierrez
- April 21, 2011 - Secure Development on iOS by David Thiel (NCC Group)
- March 12, 2011 - New Age Attacks Against Apple's iOS (and Countermeasures) by Nitesh Dhanjani

Thanks to

- Tom Daniels, David Thiel, Alban Diquet, Peter Oehlert, Joel Wallenstrom and Raphael Salas
  - iSEC Partners (NCC Group USA)
- Doug Ipperciel, Richard Warren, Ollie Whitehouse and Arjun Pednekar
  - NCC Group UK
- Steve, Adrian and the rest of the 44CON crew

Thank you! Questions?

nccgroup
freedom from doubt

UK Offices

Manchester - Head Office

Cheltenham

Edinburgh

Leatherhead

London

Thame

North American Offices

San Francisco

Atlanta

New York

Seattle

Austin

Chicago

Australian Offices

Sydney

European Offices

Amsterdam - Netherlands

Munich – Germany

Zurich - Switzerland

Contact us

training@nccgroup.com