Pentesting iOS Apps
Runtime Analysis and Manipulation

Andreas Kurtz
About

• PhD candidate at the Security Research Group, Department of Computer Science, University of Erlangen-Nuremberg
  – Security of mobile devices & mobile Apps
  – Dynamic analysis of iOS Apps

• Co-Founder of NESO Security Labs GmbH
  – Software security
  – Penetration testing, static code analysis
Pentesting iOS Apps

• Status quo: Focus on backend services
  – Well-known methodologies and techniques
  – Numerous tools available

• So far only little information on mobile App assessments

• Lack of tools
What this talk is about

• Introduction to the Objective-C Runtime
  – Backgrounds, techniques and tools for manipulating iOS Apps at runtime

• Use cases and impacts
  – Pentesters should be able to explore the attack surface of iOS Apps more efficiently
  – Developers might prefer to avoid client-side logic and security measures in the future
Objective-C Runtime

INTRODUCTION
Objective-C

• Provides a set of extensions to the C programming language

• Additions are mostly based on Smalltalk
  – Object-oriented
  – Messaging
  – Dynamic typing
  – Reflection

These concepts make Objective-C quite attractive from a hacking perspective
Objective-C

• Sample Code:

```objective-c
HelloWorld *hello = [[HelloWorld alloc] init];
[hello sayHello:@"DeepSec"];  

- (void) sayHello: (NSString *) string {
  printf("Hello %s!", [string UTF8String]);
}
```
Objective-C Runtime

- Apps are linked to *libobjc.A.dylib*

```bash
# otool -L HelloWorld

HelloWorld:
/System/Library/Frameworks/Foundation.framework/Foundation (compatibility version 300.0.0, current version 890.1.0)
/usr/lib/libobjc.A.dylib (compatibility version 1.0.0, current version 228.0.0)
[..]
```

This library provides all runtime functionalities of the Objective-C Runtime
Objective-C Runtime

• Most important function: `objc_msgSend`

• Example

```c
Class class = objc_getClass("HelloWorld");
id receiver = [[class alloc] init];
SEL selector = NSSelectorFromString(@"sayHello:");
objc_msgSend(theReceiver, theSelector, @"DeepSec");
```

Pointer to an instance of the class, whose method we want to call
Objective-C Runtime

• Most important function: `objc_msgSend`

• Example

```c
Class class = objc_getClass("HelloWorld");
id receiver = [[class alloc] init];
SEL selector = NSSelectorFromString(@"sayHello:");
objc_msgSend(theReceiver, theSelector, @"DeepSec");
```

The selector of the method that handles the message
Objective-C Runtime

• Most important function: `objc_msgSend`

• Example

```objective-c
Class class = objc_getClass("HelloWorld");
id receiver = [[class alloc] init];
SEL selector = NSSelectorFromString(@"sayHello:");

objc_msgSend(theReceiver, theSelector, @"DeepSec");
```

A variable argument list containing the arguments to the method
Static vs. Dynamic Analysis

• During static analysis, control flow is lost when `objc_msgSend` is called

• Characteristics of the Objective-C Runtime enables comprehensive dynamic analysis

<table>
<thead>
<tr>
<th>Technique</th>
<th>Usage</th>
</tr>
</thead>
<tbody>
<tr>
<td>Intercept messages</td>
<td>Trace internal control flow</td>
</tr>
<tr>
<td>Send arbitrary messages to existing objects</td>
<td>Manipulate internal state and processing logic of an iOS App</td>
</tr>
<tr>
<td>Rewrite implementations of arbitrary methods</td>
<td></td>
</tr>
</tbody>
</table>
RUNTIME MANIPULATION

Backgrounds & Techniques
Starting Point

• Goal: Black box analysis of an arbitrary iOS App
  – Enterprise or AppStore App
  – Binary format (no source code available)

• Approach: Examine the iOS App on a jailbroken device
  – Removes the limitations imposed by Apple
  – Provides root access to the operating system
  – Enables the installation of additional software
  – Enables access to the Objective-C Runtime!
Runtime Manipulation

• Objective-C Runtime [1] offers a wide range of opportunities to manipulate existing iOS Apps

• Two different approaches
  – Injecting a static library with new functionalities
  – Injecting an interpreter for on-the-fly manipulations
Dynamic Library Injection

• Advise the dynamic linker to load a dynamic shared library (*DYLD_INSERT_LIBRARIES*) [2]
Runtime Patching

- Replace existing methods and reroute program control during library initialization
Hooking in Practice

• MobileSubstrate [3]
  – *MobileLoader* loads 3rd-party patching code into the running application
  – *MobileHooker* is used to hook and replace system methods and functions

    IMP MSHookMessage(Class class, SEL selector, IMP replacement, const char* prefix);

    void MSHookFunction(void* function, void* replacement, void** p_original);

• Recommendation: Theos suite eases the development of MobileSubstrate extensions (Tweaks) [4]
Example: Fake Device Information

```c
#include "substrate.h"
#import <Foundation/Foundation.h>

NSString *replaced_UIDevice_uniqueIdentifier() {
    return @"DeepSec";
}

__attribute__((constructor))
static void initialize() {
    MSHookMessage(objc_getClass("UIDevice"),
                  @selector(uniqueIdentifier),
                  (IMP)replaced_UIDevice_uniqueIdentifier,
                  NULL);
}
```
Runtime Manipulation

• Objective-C Runtime [1] offers a wide range of opportunities to manipulate existing iOS Apps

• Two different approaches
  – Injecting a static library with new functionalities
  – Injecting an interpreter for on-the-fly manipulations

✓
Cycrypt: Objective-JavaScript [5]

“"A programming language designed to blend the barrier between Objective-C and JavaScript.””

• Injects a JavaScript interpreter into a running App
  – Based on MobileSubstrate

• Enables runtime manipulations in a flexible way [6], [7]
Example: Fake Device Information

• **Step 1:** Attach to the App process

  # cycript -p <PID>

• **Step 2:** Determine the current UDID

  cy# [[UIDevice currentDevice] uniqueIdentifier];
  @"768f0c93a69276d190b6..."
Example: Fake Device Information

• **Step 3:** Replace the implementation of the API method

```javascript
cy# UIDevice.messages['uniqueIdentifier'] = function() { return @"DeepSec"; }
```

• **Step 4:** Query the UDID again

```javascript
cy# [[UIDevice currentDevice] uniqueIdentifier]; @"DeepSec"
```
Example: Fake Device Information

<table>
<thead>
<tr>
<th>Name</th>
<th>iPhone-FAU</th>
</tr>
</thead>
<tbody>
<tr>
<td>UDID</td>
<td>DeepSec</td>
</tr>
<tr>
<td>Model</td>
<td>iPhone</td>
</tr>
<tr>
<td>System Type</td>
<td>iPhone OS</td>
</tr>
<tr>
<td>Version</td>
<td>5.1.1 (Build 9B208)</td>
</tr>
<tr>
<td>System Uptime</td>
<td>17275 sec</td>
</tr>
<tr>
<td>Hostname</td>
<td>iphone-fau.local</td>
</tr>
</tbody>
</table>
Example: Fake Device Information

• Example demonstrates the diverse possibilities of iOS runtime injection

• This might be useful in different scenarios
  – Apps that rely on hardware identifier for authentication
  – Apps that use binary or any proprietary protocols

• Easier to manipulate the App endpoint, compared to modifications at protocol-level
Advantages of Runtime Manipulation

• By using these techniques, running Apps can be extended with additional debugging and runtime tracing capabilities

• This assists security assessments of iOS Apps
  – Eases the discovery of vulnerabilities
  – Simplifies bypassing client-side limitations and restrictions
Evaluate Encryption Schemes

• Typical question: Which App methods are called after the “Login” button is pressed?

• Idea: Make use of dynamic analysis to reconstruct the control flow of an App
  – Use the results to navigate through static code

• Solution: Log all messages to `objc_msgSend`
The gdb way

(gdb) exec-file /var/mobile/Applications/<APP-EXECUTABLE>
Reading symbols for shared libraries . done
(gdb) attach <PID>
Attaching to program: `/private/var/mobile/Applications/...', process PID.
Reading symbols for shared libraries . done
Reading symbols for shared libraries ................................ done
Reading symbols for shared libraries + done
0x364d7004 in mach_msg_trap ()
(gdb) break objc_msgSend
Breakpoint 1 at 0x32ce2f68
(gdb) commands
Type commands for when breakpoint 1 is hit, one per line.
End with a line saying just "end".
>printf "-%[s %s]\n", (char *)class_getName($r0),$r1
>c
>end
(gdb) c
Continuing.
The gdb way

Breakpoint 1, 0x32ce2f68 in objc_msgSend ()
-[[UIStatusBarServer _receivedStatusBarData:actions:]]

Breakpoint 1, 0x32ce2f68 in objc_msgSend ()
-[[UIStatusBar statusBarServer:didReceiveStatusBarData:withActions:]]

Breakpoint 1, 0x32ce2f68 in objc_msgSend ()
-[[UIStatusBar _currentComposedData]]

Breakpoint 1, 0x32ce2f68 in objc_msgSend ()
-[[UIStatusBar _currentComposedDataForStyle]]

Breakpoint 1, 0x32ce2f68 in objc_msgSend ()
-[[UIStatusBarComposedData alloc]

[..]

Very noisy! All background activities of the runtime are shown as well.
App Tracing

• Preferred approach: Intercept messages to `objc_msgSend` within the runtime

• Apply filters with different granularity
  – Enumerate registered App classes and methods using the Objective-C Runtime API (`objc_getClassList`, `class_copyMethodList`, etc.)
  – Output a trace of only matching items

• Inspired by Aspective-C [8] and Subjective-C [9]
App Tracing

• Tricky part is to handle all parameters and to continue normal execution
  – Logging itself modifies CPU registers and the stack

• Current execution state has to be preserved
  – Allocate an alternate stack within heap memory
  – Backup $r0$ - $r3$ and $lr$ registers to alternate stack
  – Do the logging and filtering
  – Restore $r0$ - $r3$ and $lr$
  – Continue execution
Encryption scheme is based on a hardcoded key within the App
Sample Output

+ [SyncManager sharedSyncManager]
- [SyncManager init]
- [SyncManager setSynDocumentOpen:], args: 0
+ [DataModel setSynchManager:], args: <0x1102ce30>
+ [DataModel initFromFile]
+ [DataModel securityModelFilePath]
+ [DataModel securityModelFilePath]
+ [PBKDF2 getKeyForPassphrase:], args: <__NSCFConstantString 0x15e2e4: >
+ [CryptoUtils decrypt]
+ [DataModel sharedModel]
+ [CryptoUtils md5:], args: <__NSCFConstantString 0x15dea4: >
+ [DataModel sharedModel]

```c
int getRandomNumber()
{
    return 4; // chosen by fair dice roll.
    // guaranteed to be random.
}
```
Advantages of Runtime Manipulation

• The ability to manipulate Apps at runtime strikes out new paths
  – Discover weak/missing encryption
  – Bypassing client-side restrictions
  – Execution of hidden functionality, which was not supposed to be accessible
  – Unlock additional features and premium content
  – Dump copyright-protected content
  – Etc.
Lack of Tools

“Security will not get better until tools for practical exploration of the attack surface are made available”

- Josh Wright
Closing the Gap

- Retrofitting existing apps with debugging and runtime tracing capabilities
Introducing *Snoop-it*

- A tool to assist security assessments and dynamic analysis of iOS Apps
# Features

<table>
<thead>
<tr>
<th>Monitoring</th>
<th>File system access (print data protection classes)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Keychain access</td>
</tr>
<tr>
<td></td>
<td>HTTP(S) connections</td>
</tr>
<tr>
<td></td>
<td>Access to sensitive API (address book, photos etc.)</td>
</tr>
<tr>
<td></td>
<td>Debug outputs</td>
</tr>
<tr>
<td></td>
<td>Tracing App internals (objc_msgSend)</td>
</tr>
</tbody>
</table>
## Features

<table>
<thead>
<tr>
<th>Analysis / Manipulation</th>
<th>Features</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Fake hardware identifier (UDID, Wireless MAC, etc.)</td>
</tr>
<tr>
<td></td>
<td>Fake location/GPS data</td>
</tr>
<tr>
<td></td>
<td>Explore and force display of available ViewControllers</td>
</tr>
<tr>
<td></td>
<td>List custom URL schemes</td>
</tr>
<tr>
<td></td>
<td>List available Objective-C classes, objects and methods</td>
</tr>
<tr>
<td></td>
<td>Invoke and replace arbitrary methods at runtime</td>
</tr>
</tbody>
</table>
# Features

<table>
<thead>
<tr>
<th>Other</th>
</tr>
</thead>
<tbody>
<tr>
<td>Simple installation and configuration</td>
</tr>
<tr>
<td>Easy to use graphical user interface</td>
</tr>
<tr>
<td>Plenty of filter and search options</td>
</tr>
<tr>
<td>Detailed description of the XML-RPC web service interface</td>
</tr>
<tr>
<td>Freely available at the end of this year</td>
</tr>
</tbody>
</table>
Getting Started

• There’s an App for That!™
Getting Started

• There’s an App for That!™

1. Open the *Snoop-it Configuration* App
Getting Started

• There’s an App for That!™

1. Open the *Snoop-it Configuration* App

2. Select Apps (System/Cydia/AppStore) to analyze
Getting Started

• There’s an App for That!™

① Open the **Snoop-it Configuration** App

② Select Apps (System/Cydia/AppStore) to analyze

③ Adjust settings (GUI, Authentication, ...)

![Snoop-it Configuration App](image)
Getting Started

• There’s an App for That!™

① Open the *Snoop-it Configuration* App

② Select Apps (System/Cydia/AppStore) to analyze

③ Adjust settings (GUI, Authentication, ...)

④ Run App & point your browser to the *Snoop-it* web interface
Please follow me on Twitter (@aykay) to stay up-to-date with the latest news on Snoop-it.
# Filesystem Monitor

A screenshot of the Filesystem Monitor tool from NESO Security Labs. The tool appears to be running on a computer with the IP address `192.168.0.12`. The tool shows a list of files and their paths from various directories. The columns in the table include `ID`, `Filepath`, and `NSFile Protection Class`. The screenshot includes various folders and files, highlighting the detailed monitoring capabilities of the tool.
Location Faker
App Tracing
Keychain Monitor
Runtime Manipulation
Jailbreak Detection

• Purpose: Verification of platform integrity

• Common checks
  – Suspicious files and directories
  – File system permissions
  – Mount options
  – Symbolic links
  – Dynamic shared libraries
  – SSH Loopback
  – Sandbox integrity (fork)
Jailbreak Detection

MOV    R0, (eApplications - 0x17CCE) ; "/Applications"
STMIA.W R4, {R1-R3}
ADD    R1, SP, #0x130+var_R0
ADD    R0, PC ; "/Applications"
ADDS   R5, R1, #1
ADD    R4, SP, #0x130+var_94
B      loc_17CD6

loc_17CD6
MOV.W  R10, #0
CBZ    R0, loc_17CF4

MOV    R1, R4 ; struct stat *
BLX    _lstat
CMP    R0, #0
BNE    loc_17CD2

LDRE.W R0, [SP,#0x1B0+var_94.st_ino+1]
MOV.W  R10, #1
TST.W  R0, #0x00
BEQ    loc_17CD2

loc_17CD2
LDR.W  R0, [R5],#4

loc_17CF4
MOV    R0, (etc/fsTAB - 0x17D08)
MOV    R1, (aR_0 - 0x17D0A) ; "z"
ADD    R0, PC ; "/etc/fsTAB"
ADD    R1, PC ; "+"  
BLX    _open
MOV    R11, R0
MOVS   R6, #0
CMP.W  R11, #0
BEQ    loc_17D7E

MOV    R0, R11 ; FILE
MOVS   R1, #0 ; _int12
MOVS   R2, #2 ; int
MOVS   R6, #0
NLX    _close
CBNZ   R0, loc_17D78
Jailbreak Detection

• In order to assess the security of an iOS App, at first the jailbreak detection mechanisms have to be bypassed

  – Binary / Run-time patching to remove all checks (specific, time-consuming)

    Delegate.messages['isJailbroken'] = function() { return NO; }

  – Intercept system calls to simulate an unmodified execution environment (generic)
Jailbreak Detection Bypass

• *Snoop-it* supports generic bypass of the most common jailbreak detection mechanisms
  – Simple configuration switch in the Configuration App
Bypassing Jailbreak Detection

DEMO
Securing the Runtime

• Minimum of data/logic on the client-side

• Preferred use of C, at least for security-critical implementations
  – Inline Functions
  – Obfuscation

• Advanced Jailbreak Detection
• Runtime Integrity Checks \((\text{dladdr()}[10])\)
Summary

• Runtime Analysis and Manipulation facilitates both, dynamic and static analysis of iOS Apps

• Attack surface of iOS Apps can be explored more efficiently
Acknowledgements

• Thanks to

– Markus Troßbach  (University of Heidelberg)
– Sebastian Stocker  (University of Heidelberg)
– Christoph Settgast  (University of Erlangen)
– Andreas Weinlein  (University of Erlangen)
– Francesca Serpi  (University of Milan)
References

[1] Objective C Runtime Reference

[2] dyld - the dynamic link editor (DYLD_INSERT_LIBRARIES)

http://iphonedevwiki.net/index.php/MobileSubstrate

http://iphonedevwiki.net/index.php/Theos

[5] Cycript
http://www.cycript.org
References

[6] Cyccript Overview
http://iphonedevwiki.net/index.php/Cycript

[7] Cyccript Tips
http://iphonedevwiki.net/index.php/Cycript_Tricks

[8] Aspective-C by saurik
http://svn.saurik.com/repos/menes/trunk/aspectivec/AspectiveC.mm

[9] Subjective-C by KennyTM~

[10] dladdr - find the image containing a given address