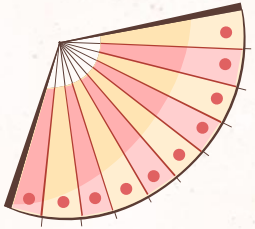
Keys To Improve Your iOS App Testing Skills

Whoa!

Laura García - Managing Security Consultant @ NCC Group

lain7z

laincode



iOS App Testing

- Methodology for iOS App Code Review
- OWASP MASVS and MASTG
- Key Areas in Mobile App Security
- SAST tools

Key Areas

- Platform Interaction
- Network
- Cryptography
- Local Authentication
- Data Storage

01.
Platform Interaction

XSS in WebViews

Check how WebViews are handling strings to identify XSS attacks.

UIWebView

- Insecure and deprecated (still allowed in updates of apps).
- JavaScript cannot be disabled.
- *loadHTMLString(_:baseURL:)*
- *file://*
- Can leak local files such as cookies.

WKWebView

- Prevents JavaScript injections.
- Check:
  - *javaScriptEnabled*
  - *JavaScriptCanOpenWindowsAutomatically*
  - *hasOnlySecureContent*
  - *AllowUniversalAccessFromFileURLs*
  - *AllowFileAccessFromFileURLs*
- Attack vectors:
  - URL scheme
  - User opening links
  - User-generated content

Native to JS - JS to Native

- WKWebview Native code to invoke JS code:
  - Avoid untrusted external data
  - Sanitize the string before passing it to *evaluateJavaScript:completionHandler:*

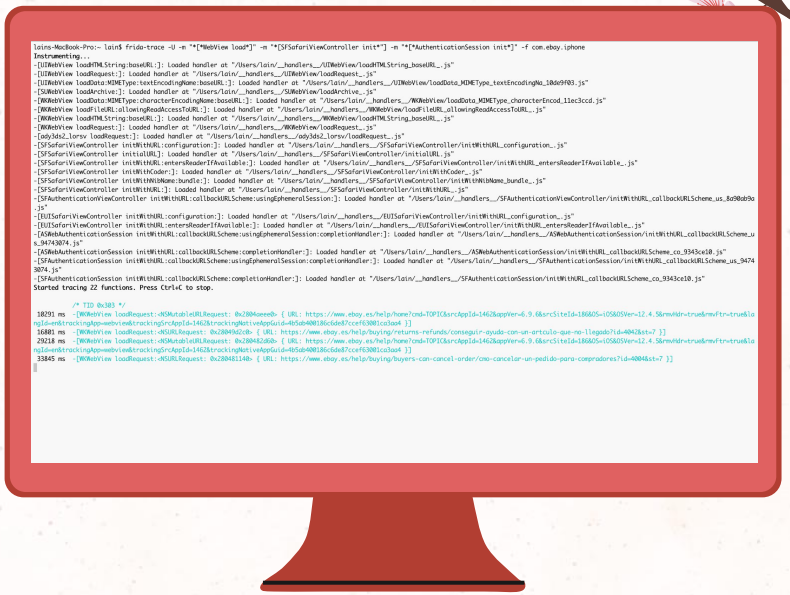
```
func evaluateColor(color:String) {
   var webview: WKWebview
   let script = “document.body.style.backgroundColor = `\(color)`;”
   webview.evaluateJavaScript(script, completionHandler:  nil)
}
```

- WKWebView JS code to invoke Native functionality (bridge):
  - Search for *WKScriptMessageHandler*
  - Identify all exposed methods

```
class JavaScriptBridgeMessageHandler: NSObject, WKScriptMessageHandler {
   func invokeNativeOperation() {
      let javaScriptCallBack = "javascriptBridgeCallBack('\(functionFromJS)','\(result)')"
      // bad practice JavaScript to Native bridge
      message.webView.evaluateJavaScript(javaScriptCallBack, completionHandler: nil)
   } }
```

Tracing Webviews

Type used
Configured
URLs loaded
Native code communication



View

String Interpolation

- Available in Swift.
- Combines variables and constants inside a string.
- Avoid the risk of a format string attacks.

```
var name = "Tim McGraw"
"Your name is \(name)"
```

```
var name = "Tim McGraw"
"Your name is " + name
```

Secure Deserialization

NSCoding

- The object is often already constructed and inserted before you can evaluate the class-type.
- This protocol does not verify the type of object upon deserialization.

NSSecureCoding

- Good practice.
- It enables encoding and decoding in a manner that is robust against object substitution attacks.
- *decodeObjectOfClass:forKey:* throw an exception if the deserialized object's class doesn't match the expected class.

```
supportsSecureCoding returns TRUE
let obj = decoder.decodeObject(of:MyClass.self, forKey: "myKey")
```

Universal Links

Apple recommends using Universal Links for linking to content within your app securely in iOS.

- Prevent URL scheme hijacking attacks.
- Are verified, this occurs at installation time.
- iOS will **refuse** to open those links if the **verification did not succeed**.
- iOS retrieves **Apple App Site Association file** for the declared domains (applinks) in the entitlements file.

```
<key>com.apple.developer.associated-domains</key>
<array>
   <string>applinks:telegram.me</string>
   <string>applinks:t.me</string>
</array>
```

https://branch.io/resources/aasa-validator/

View

Universal Links Potential Attack Vector

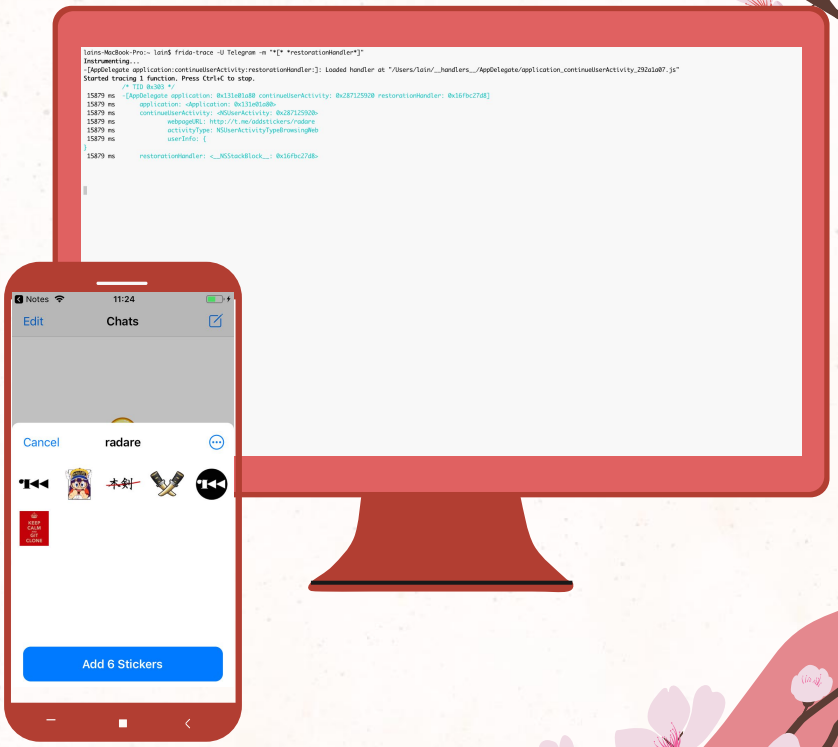
- Validate all URL parameters and discard any malformed URLs

```
application:continueUserActivity:restorationHandler:
```

- Make sure your **test cases** include improperly formatted URLs
- Do not allow universal links to directly delete content or access **sensitive information** about the user
- Verify that the **actions triggered** by the URL not leaking sensitive information

Tracing Universal Links

View



Insecure pasteboard

The UIPasteboard enables sharing data within an app, and from an app to other apps.

General (Persistent)

generalPasteboard

Custom/Named (Non-persistent)

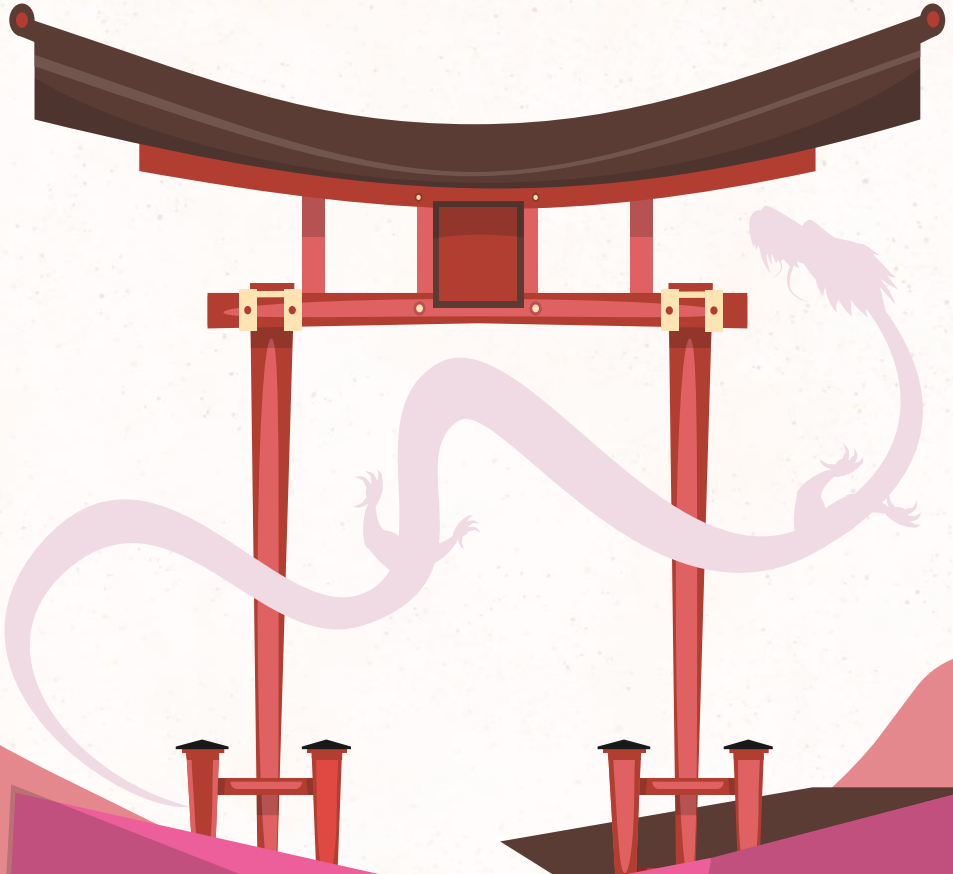
pasteboardWithName:create
pasteboardWithUniqueName

- Check that **sensitive information** is not being saved inside the general pasteboard.
- Universal Clipboard (Handoff). It automatically transfers its contents to nearby devices that use the same iCloud account.
  - *UIPasteboardOptionExpirationDate & UIPasteboardOptionLocalOnly*
- **Clear the pasteboard** items when going to background or when terminating.
- Check if pasteboards are being **removed**.
  - *removePasteboardWithName*: (no effect for the general pasteboard)

Memory for sensitive data

- Applications should **sanitise in-memory** buffers after use, where possible, if the data is no longer required for operation (for example a temporary password or PIN buffer).
- Apple Secure Coding Guide suggests **zeroing** sensitive data after usage.
- Consider using NSMutableData for storing secrets on Swift/Objective-C and use *resetBytes(in:)* method for zeroing.

```
// NSDictionary <passwordUniqueIdentifier, passwordKey>
@property (nonatomic, strong) NSMutableDictionary <NSString *,
NSMutableData *> *passwordsCache;
...
// clean up values
[self.passwordsCache enumerateKeysAndObjectsUsingBlock:^(NSString *
_Nonnull key, NSMutableData * _Nonnull data, BOOL * _Nonnull stop) {
        [data resetBytesInRange:NSMakeRange(0, [data length])];
}];
[self.passwordsCache removeAllObjects];
```



02.
Network

App Transport Security (ATS)

- Check that no exceptions are done in the Info.plist.
- ATS only be deactivated under certain circumstances.

```
<key>NSAppTransportSecurity</key>
 <dict>
 <key>NSAllowsArbitraryLoadsInWebContent</key>
 <true/>
 <key>NSExceptionDomains</key>
 <dict>
        <key>example.com</key>
        <dict>
        <key>NSIncludesSubdomains</key><true/>
        <key>NSExceptionMinimumTLSVersion</key>
        <string>TLSv1.1</string>
        </dict> </dict> </dict>
```

Change the server side configuration to make the server-side more secure, instead of weakening the configuration in ATS on the client

API Keys

API keys may provide access to third-party services like AWS storage, SMS gateway, payments API, or analytics.

- Search for API Keys stored in the app code.
- If you must have an API key or a secret to access some resource from your app, build an **orchestration layer between your app and the resource**.

03.

Cryptography

Cryptographic Standard Algorithms

Apple provides libraries that include implementations of most common cryptographic algorithms.

- CryptoKit (released with iOS 13): Secure algorithms for hashing, symmetric-key and public-key cryptography.
- CommonCrypto: lacks a few types of operations in its public APIs
- SecKey: for asymmetric operations.
- Third-party: CryptoSwift (Github). It is important to verify the effective implementation of a function.

Key Management

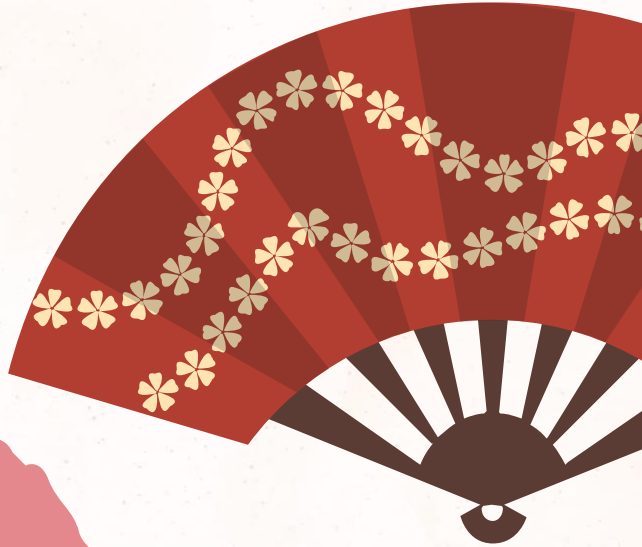
- Keys are not hardcoded.
- Keys are not stored without additional protection.
- Keys are not synchronized over devices if it is used to protect high-risk data.
- Keys are not hidden by use of lower level languages (C/C++) or derived from stable features of the device.
- Use the Keychain (NOT kSecAttrAccessibleAlways protection class).

Generate cryptographically-strong random numbers

- Apple provides a Randomization Services API, which generates cryptographically secure random numbers.

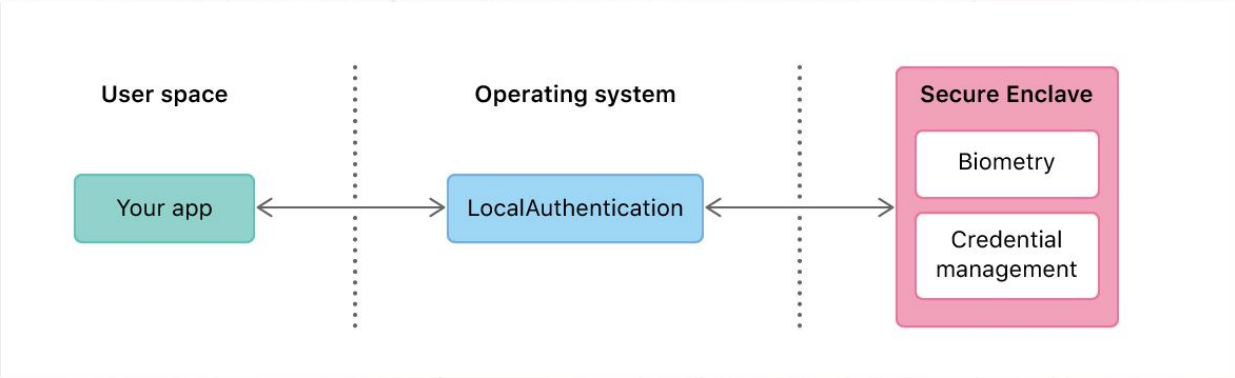
```
int r = SecRandomCopyBytes(kSecRandomDefault, sizeof(int), (uint8_t*) &res);
```

04.
Local Authentication



During local authentication, an app authenticates the user against credentials stored locally on the device, by providing a valid PIN, password or biometric characteristics such as face or fingerprint.

LocalAuthentication.framework



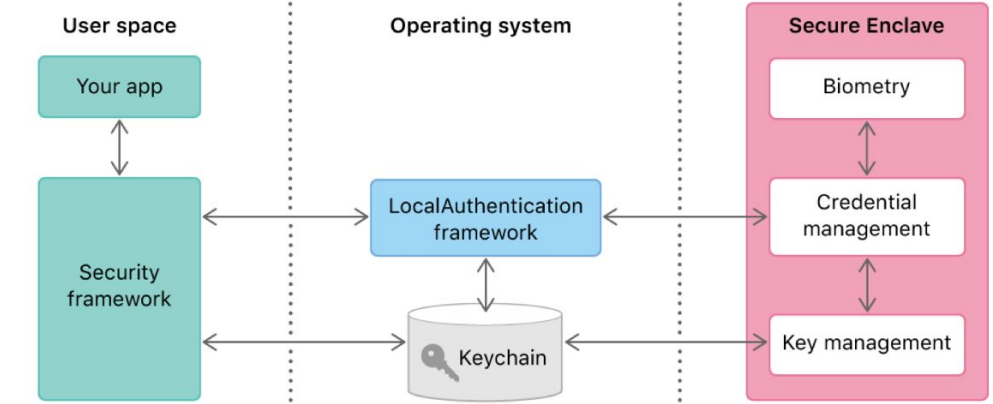
- High-level API used to authenticate the user via Touch ID.
- **evaluatePolicy** function returns a boolean value indicating whether the user has authenticated successfully.

How to

Bypassing your apps' biometric checks on iOS

View

Protect a keychain item with biometric authentication



- Lower-level API to access keychain services.
- App stores a secret authentication token in the keychain.
- User unlocks the keychain using their passphrase or fingerprint.

- *SecAccessControlCreateFlags.biometryCurrentSet*
- *kSecAttrAccessibleWhenPasscodeSetThisDeviceOnly*

05.
Data Storage

KeyChain

Used to securely store sensitive data, such as encryption keys and session tokens.

Consider if sensitive data stored by the application should be marked with the “do not backup” attribute.

- **kSecAttrAccessibleWhenPasscodeSetThisDeviceOnly** Items can be accessed only when the device is unlocked. Passcode set on the device. The data won't be included in an iCloud or local backup.
- **kSecAttrAccessibleWhenUnlocked** Items with this attribute migrate to a new device when using encrypted backups.

When an app is first launched after installation, wipe all Keychain data associated with the application.

Make sure that the Keychain data is wiped as part of account logout.

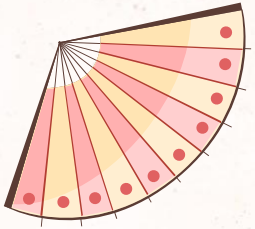
TouchID / FaceID

Access control flags to protect an item with biometry:

- **kSecAccessControlBiometryAny:** The Keychain entry will survive any re-enrolling of new fingerprints or facial representation.
- **kSecAccessControlBiometryCurrentSet:** Item in the Keychain is invalidated if fingers are added or removed for Touch ID, or if the user re-enrolls for Face ID.
- **kSecAccessControlUserPresence:** Allow the user to authenticate through a passcode if the biometric authentication no longer works.

It is also possible to prevent a physical attacker from enrolling their own fingerprint on the device.

- **evaluatedPolicyDomainState**: Determinates enrollment changes have occurred since last usage.




SAST tools

Semgrep

- Open-source **static analysis tool** used to **search for patterns** in code.
- Customized and combined to create powerful rules for code analysis.
- Integrated into a development workflow (CI/CD) pipeline.
- Supports many programming languages: Swift (experimental).

- **Swift repo**: https://github.com/akabe1/akabe1-semgrep-rules
- **Rules**: Certificate Pinning, Biometric Authentication, XXE, SQL Injection, Crypto, Log Injection, NoSQL Injection, WebView, Insecure Storage, Keychain Settings issues and others ...

```
$ semgrep --config akabe1-semgrep-rules/<SUBFOLDER>/ /code/
```

Semgrep

Scanning iOS (Swift) app

View

Mobsfscan

- **Static analysis tool** that can find insecure **code patterns** in your Android and iOS source code.
- Supports: Swift and Objective-C Code.
- Mobsfscan uses MobSF static analysis rules and is powered by semgrep and libsast pattern matcher.
- Integrated into a development workflow (CI/CD) pipeline.

```
$ mobsfscan tests/assets/src/
```

Mobsfscan

Scanning iOS (Obj-C/Swift) app

SEVERITY	INFO

RULE ID	ios_custom_keyboard_disabled
CWE	CWE-919: Weaknesses in Mobile Applications
MASVS	MSTG-PLATFORM-11
OWASP-MOBILE	M1: Improper Platform Usage
REFERENCE	https://github.com/MobSF/owasp-mstg/blob/master/Document/0x06h-Testing-Platform-Interaction.md#app-extensions
DESCRIPTION	This app does not have custom keyboards disabled.
SEVERITY	INFO

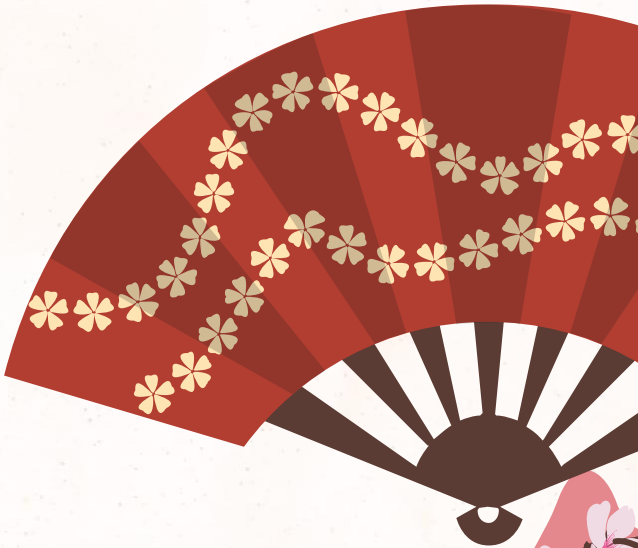
RULE ID	ios_cert_pinning
CWE	CWE-295: Improper Certificate Validation
MASVS	MSTG-NETWORK-4
OWASP-MOBILE	M3: Insecure Communication
REFERENCE	https://github.com/MobSF/owasp-mstg/blob/master/Document/0x05g-Testing-Network-Communication.md#testing-custom-certificate-stores-and-certificate-pinning-mstg-network-4
DESCRIPTION	This app does not have Certificate Pinning implemented in code.
SEVERITY	INFO

RULE ID	ios_keyboard_cache
CWE	CWE-919: Weaknesses in Mobile Applications
MASVS	MSTG-STORAGE-5
OWASP-MOBILE	M2: Insecure Data Storage
REFERENCE	https://github.com/MobSF/owasp-mstg/blob/master/Document/0x06d-Testing-Data-Storage.md#finding-sensitive-data-in-the-keyboard-cache-mstg-storage-5
DESCRIPTION	This app does not disable Keyboard cache. It must be disabled for all sensitive data inputs.
SEVERITY	INFO

View

References

- https://github.com/OWASP/owasp-mastg/
- https://github.com/felixgr/secure-ios-app-dev
- https://book.hacktricks.xyz/mobile-pentesting/ios-pentesting/
- https://developer.apple.com/documentation/localauthentication/accessing_keychain_items_with_face_id_or_touch_id
- https://github.com/MobSF/mobsfscan
- https://github.com/returntocorp/semgrep
- https://github.com/akabe1/akabe1-semgrep-rules
- https://github.com/ajinabraham/libsast
- https://github.com/sushi2k/SecMob-Training
- https://github.com/OWASP/MASTG-Hacking-Playground/tree/master/iOS/MSTG-JWT
- https://github.com/OWASP/iGoat-Swift
- https://github.com/prateek147/DVIA-v2
- https://github.com/authenticationfailure/WheresMyBrowser.iOS



Thanks!

/Rooted CON

Do you have any questions?

CREDITS: This presentation template was created by **Slidesgo**, including icons by **Flaticon**, infographics and images by **Freepik**