iOS Security: Deep Dive

Vadim Drobinin | @valzevul

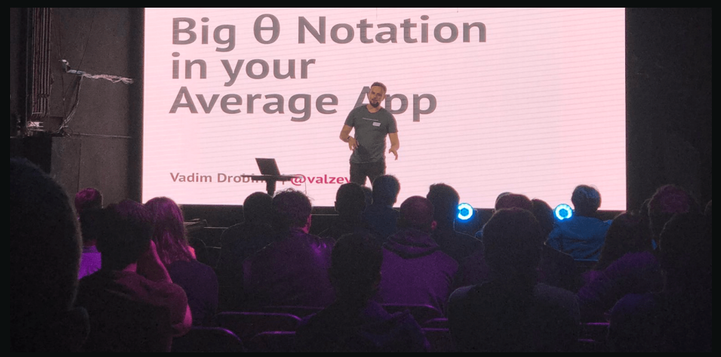
About me

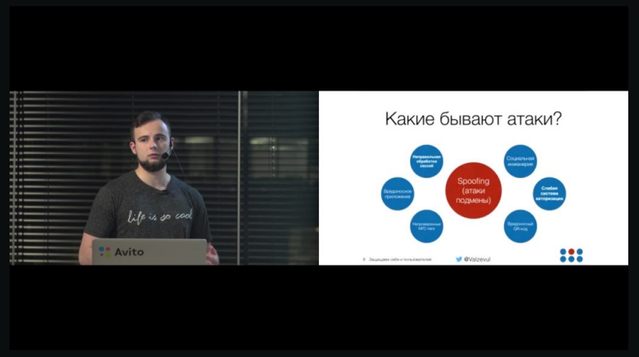
CocoaHeads Russia Декабрь 2016
Как задаётся токен
Кредитная карта
PAN, CVV, Exp. Date = X
Пользователь
Apple Pay
Apple Pay Servers
Банк
Token, Token-key, cvv-key = Y
Secure Element
Token Service Provider

"SEATBELT"

like a Pro
Vadim Drobinin | drobinin.com

NLP
...Grammar induction...
...PoC-tagging...
IF


Big θ Notation
in your
Average App


Какие бывают атаки?
Spoofing (атаки подмены)
Avito

Agenda (part I: Introduction)

- ~~Why bother?~~ ✅
- ~~Jailbreak or not?~~ ✅
- ~~Tools Overview~~ ✅
- ~~Platform Overview~~ ✅
- ~~Security 101:~~ ✅
  - ~~OWASP~~ ✅
  - ~~Data protection~~ ✅

Agenda (part II: Penetration testing)

- Local Authentication
- Network API
- Universal Links
- WebViews
- Unusual attack vectors
- Jailbreak detection
- Where to go from here

Local Authentication on iOS

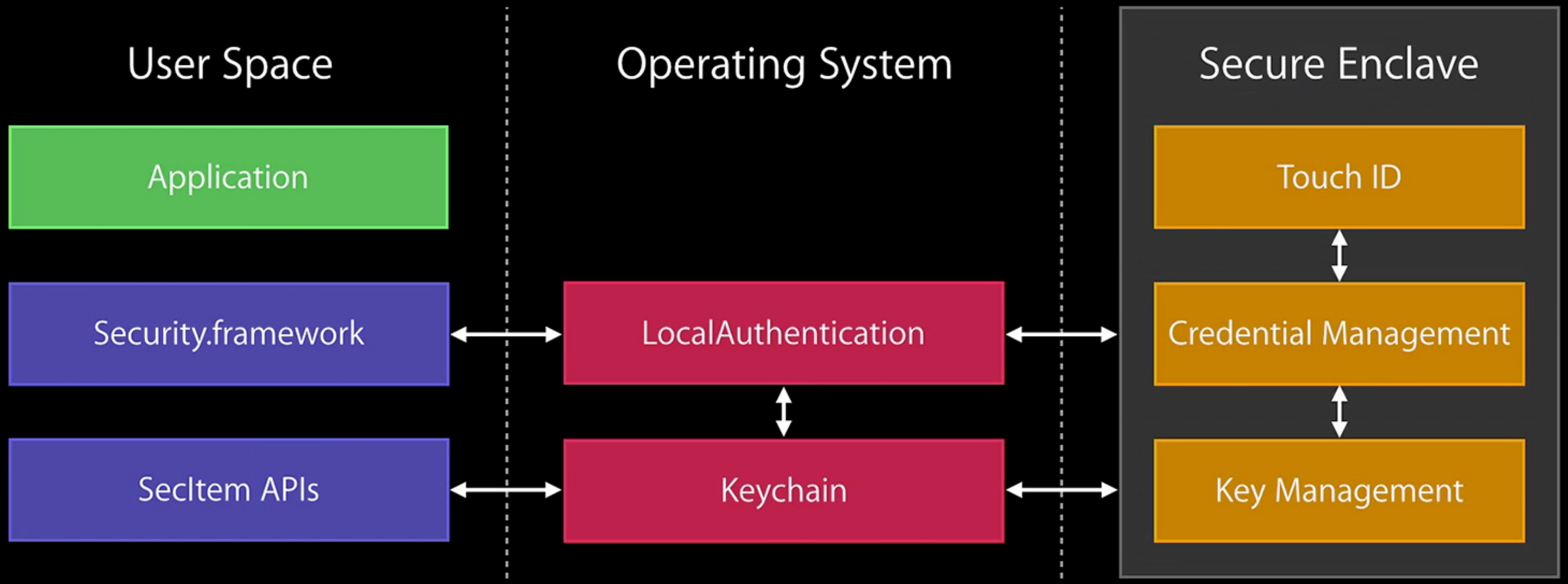
During local authentication, an app authenticates the user against credentials stored locally on the device.

- **LocalAuthentication.framework**
  high-level API for TouchID/FaceID
- **Security.framework**
  low-level API for Keychain Services

It’s secure, right?

It’s secure, right?

Nope.


User Space
Operating System
Secure Enclave
Application
Security.framework
SecItem APIs
LocalAuthentication
Keychain
Touch ID
Credential Management
Key Management

Local Authentication

- deviceOwnerAuthentication
- deviceOwnerAuthenticationWithBiometrics

Local Authentication

```
LAContext().evaluatePolicy(.deviceOwnerAuthentication, localizedReason: "…") {
    success, evaluationError in
    if success {
        // Now you can trust the user
    }
}
```

- See **Don't touch me that way**[2] for a bypassing auth demo

[2] https://www.youtube.com/watch?v=XhXIHVGCFFM by David Lidner et al

iOS Network API

App Transport Security (ATS):

- *NSURLConnection*, *NSURLSession* and *CFURL*
- Public hostnames (not IP addresses, unqualified domain names or TLD of *.local*)
- No HTTP connections
- Transport Layer Security (TLS) version *>=1.2*.
- Some more requirements to keys exchange

How to protect

- ATS should be configured according to best practices by Apple and only be deactivated under certain circumstances.
- Don’t forget about SSL pinning; never hardcode the password though.

How to protect

- If the application opens third party web sites in web views, *NSAllowsArbitraryLoadsInWebContent* can be used to disable ATS restrictions for the content loaded in web views.
- If the app connects to a defined number of domains under your control, configure the servers to support the ATS requirements and opt-in for the ATS requirements within the app.

How to protect

```
<key>NSAppTransportSecurity</key>
<dict>
    <key>NSAllowsArbitraryLoads</key>
    <true/>
    <key>NSExceptionDomains</key>
    <dict>
        <key>example.com</key>
        <dict>
            <key>NSIncludesSubdomains</key>
            <true/>
            <key>NSExceptionMinimumTLSVersion</key>
            <string>TLSv1.2</string>
            <key>NSExceptionAllowsInsecureHTTPLoads</key>
            <false/>
            <key>NSExceptionRequiresForwardSecrecy</key>
            <true/>
        </dict>
    </dict>
</dict>
```

iOS Platform APIs

- All apps run under non-privileged `mobile` user
- Each app has a unique home directory and is sandboxed
- Access to protected resources or data (*capabilities*) is possible, but it's strictly controlled via special permissions (*entitlements*).

Don’t ask for more permissions than you actually need at that very moment.

What might go wrong?[3]

- Camera access
  - record users at any time the app is in the foreground
  - run real-time face recognition to detect facial features or expressions
  - upload the pictures/videos it takes immediately
- Photos
  - Track all users' movements based on their photos' meta
  - Track all their devices
  - Use facial recognition to find out who the user hangs out with

[3] Felix Krause, https://krausefx.com/privacy

What might go wrong?

- MitM-attack to change the 3d-party framework
- Fake iCloud password alerts
- Inject anything into web views (if the app doesn't use SFSafariViewController)
- Inject anything into a Share Extension °_O
- Screenshot typing password in app's secured fields

°_O https://github.com/valzevul/ElevationOfPrivilegeHack

Inter Process Communication

- Universal Links
- UIPasteboard
- Custom URL Schemes

Universal Links

- tg://resolve?domain=valzevul is a custom URL scheme and uses the tg:// scheme.
- https://telegram.me/valzevul is a universal link and uses the https:// scheme.
- Unique
- Secure
- Flexible
- Private

What to test

- Check the Associated Domains entitlement
- Retrieve the Apple App Site Association file
- Check the link receiver method
- Check the data handler method
- Check if the app is calling other app's universal links

…do not allow universal links to directly delete content or access sensitive information about the user.

— Apple Documentation

UIPasteboard

- Users cannot grant or deny permission for apps to read the pasteboard.
- Apple warns about persistent named pasteboards and discourages their use. Instead, shared containers should be used.
- Universal Clipboard is enabled by default and allows the general pasteboard contents to automatically transfer between devices.

Custom URL Schemes

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 Documentation

- *canOpenURL* will always return *false* for undeclared schemes
- though *openURL* will still open it even if *LSApplicationQueriesSchemes* is set
- List of URL scheme names[4]

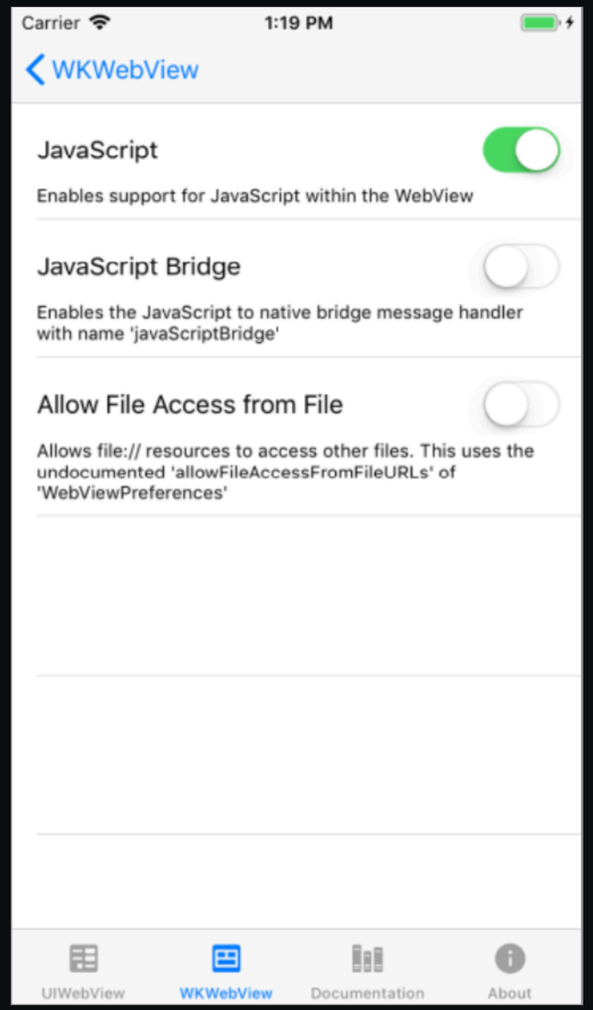
[4] https://ios.gadgethacks.com/news/always-updated-list-ios-app-url-scheme-names-0184033/

Web views

- UIWebView (deprecated + impossible to turn off JS)
- SFSafariViewController (impossible to turn off JS)
- WKWebView:
  - use *javaScriptEnabled = false*
  - use *hasOnlySecureContent = true*
  - out-of-process rendering → no memory corruption bugs

Web Views

- Native methods could be exposed through web views.
- Custom web views could ~~steal~~ store passwords, sessions, keys, etc.
- AutoFill data is available only for SFSafariViewController.
- Try **WhereIsMyBrowser**[5], an intentionally insecure app for training.



[5] https://github.com/authenticationfailure/WheresMyBrowser.iOS

iOS Anti-Reversing Defenses

- Jailbreak detection
- Device binding
- Anti-debugging checks

Jailbreak detection

- File-based checks
- File-permissions checks
- Protocol handlers (eg *cydia://*)
- Calling System APIs

Device binding

- ❌ MAC addresses, UDID, unsafe bindings
- ✅ UIDevice.current.identifierForVendor
- ✅ Keychain + kSecAttrAccessibleWhenUnlockedThisDeviceOnly
- ✅ Google and its Instance ID for iOS

Anti-debugging checks

`PT_DENY_ATTACH`: it allows a process that is not currently being traced to deny future traces by its parent. All other arguments are ignored.

— developer.apple.com

Anti-debugging checks

~~PT_DENY_ATTACH: it allows a process that is not currently being traced to deny future traces by its parent. All other arguments are ignored.~~
~~— developer.apple.com~~

- Does it work? Not really, not always [1]
- Can we make it more complicated? Yes. [2]

[1] https://knight.sc/debugging/2019/06/03/debugging-apple-binaries-that-use-pt-deny-attach.html

[2] https://stackoverflow.com/a/55575155

Anti-debugging checks

- Let's do it in Assembly
- Let's obfuscate it

Create a new `C` File (e.g `anti_debug.c`):

```
#include "anti_debug.h"

void m6d64657663616d70() {
  __asm (
    "mov r0, #31\n"
    "mov r1, #0\n"
    "mov r2, #0\n"
    "mov r3, #0\n"
    "mov ip, #26\n"
    "svc #0x80\n"
  );
}
```

Create a new `Header` (e.g `anti_debug.h`) [NB]

```
#ifndef anti_debug_h
#define anti_debug_h

#include <stdio.h>

void m6d64657663616d70(void) __attribute__((always_inline));

#endif
```

[NB] Add #include "anti_debug.h" to your bridging header and a bridging header if your project does not already have one.

Add it to your project

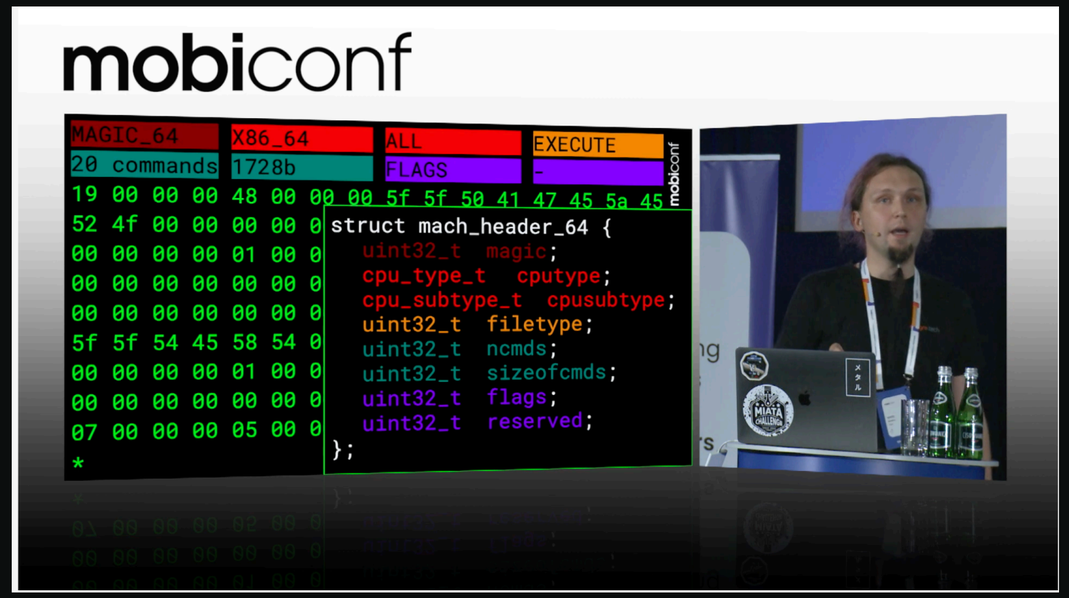
```
// main.swift

m6d64657663616d70()

UIApplicationMain(
    CommandLine.argc,
    CommandLine.unsafeArgv,
    nil,
    NSStringFromClass(AppDelegate.self)
)
```

github.com/valzevul/anti-debugging-poc

Obfuscate everything



0xFEEDFACE

by @kam800

Unusual vectors

Shortcuts

- Untrusted shortcuts
- Used to have access to files protected via Sandbox
- Protected schemes access
- Prone to DoS attacks
- Base64 obfuscation

Handoff

Sign-in with Apple

- JWT tokens at fault
- User impersonation
- JS-related risks on other platforms

HomeKit

- Look for access to unauthorised control
- Hardware issues are present too

HealthKit

- Look out for interactions (e.g whether the app stores something temporary)

ARKit

- ARKit might use the second camera in background
- The data might be stored / cached by the app

Lost Mode, remote wipe, and remote lock

- Installed certificates might have various access to the device

Screen Time

When a user sees a list of apps in Screen Time usage information, the app icons are pulled directly from the App Store, **which doesn't retain any data from these requests**.

— Apple Platform Security Guide

Caring about users

- Informing users on their private information:
  - The right to be forgotten
  - The right to correct data
  - The right to access user data
- Apple's best practices (Accessibility, Localization, etc)

Testing

- Preparation
- Intelligence Gathering
- Mapping the Application
- Exploitation
- Reporting

Not enough?

- iOS Security Guide by Apple[6]
- How SSL Kill Switch works on iOS 12[7]
- From checkra1n to Frida: iOS App Pentesting Quickstart on iOS 13[8]
- Apple Pay: Delve into the details[9]
- OWASP Mobile Security Testing Guide [10]

---

[6] https://manuals.info.apple.com/MANUALS/1000/MA1902/en_US/apple-platform-security-guide.pdf

[7] https://nabla-c0d3.github.io/blog/2019/05/18/ssl-kill-switch-for-ios12/

[8] https://spaceraccoon.dev/from-checkra1n-to-frida-ios-app-pentesting-quickstart-on-ios-13

[9] https://drobinin.com/talks/2017/apple-pay-delve-into-the-details/

[10] https://github.com/OWASP/owasp-mstg

More deep dive talks?

10 lines of encryption, 1500 lines of key management
by *@vixentael*
https://youtu.be/cp06I49Cv50

How to Frida good
by *@kevcody*
https://slideslive.com/38916544/how-to-frida-good

Don't stop at security testing. Write your own apps, compile your own kernels, dissect mobile malware, ***learn how things tick***.

— OWASP

Thank you

drobinin.com | @valzevul