MSTG Hands-on – iOS

OWASP New Zealand Day 2020

Sven Schleier

/usr/bin/whoami

- Hi everyone, my name is Sven
- Unix Admin (previous life)
- Pentester (previous life)
- Security Architect for Web and Mobile Apps during SDLC (previous life)
- Principal Cyber Security Consultant in ☀️ Singapore
- One of the project leaders for:
  - OWASP Mobile Security Testing Guide (MSTG) and
  - OWASP Mobile AppSec Verification Standard (MASVS)

iOS Preparation Pack

Please download the zip file to your MacBook:

- http://bit.ly/2HrVqXv
- or on the USB flash drive / hard drive

This contains the slides, apps and other material we use with your iOS device today.

Agenda

- Introduction
- iOS Platform and Security Mechanisms
- iOS Device Setup and Security Testing Basics
- iOS Application Structure
- iOS Testing with Jailbroken and non-Jailbroken devices

Testing with a hardware device (Jailbroken or non-Jailbroken)

- Frida
- Analyze Local Data Storage
- Biometric Authentication (Face ID / Touch ID)

Testing with Corellium (Jailbroken)

- Intercepting Network Communication / SSL Pinning
- Stateful and Stateless Authentication
- Static Analysis of an IPA
- Jailbreak Detection

Additional Exercises

Rules

<u>There are no stupid questions!</u>

You are highly encouraged:

- To ask during the training
- Come to us before or after class
- Come to us during the breaks
- Or contact us via Twitter/Linkedin/OWASP Slack/$foo)

The intention of this course is to be interactive and this depends on your participation!

Do not hack / take over the instances of your new friends in the classroom ;-)

Disclaimer

The content for today is based on the OWASP Mobile Security Testing Guide and our experience during mobile pentests.

Feel free to use it after the training, but **please don't share the training content!**

All views and opinions in this training slides and that are mentioned during the training are our own and not the view of our employers.

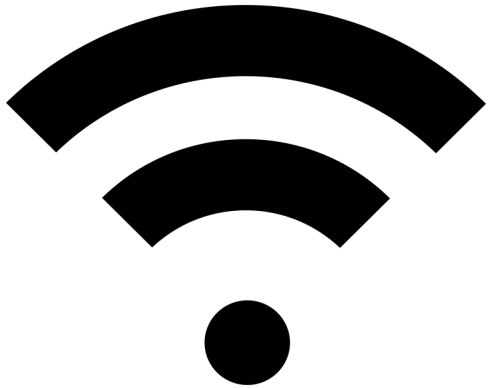
Setup for Training

iOS Device

Please bring your iOS device to me.

I would need to install one dummy app and register the device to my apple developer account!

Wifi



Please connect your **MacBook** and **iOS device** to the Wifi.

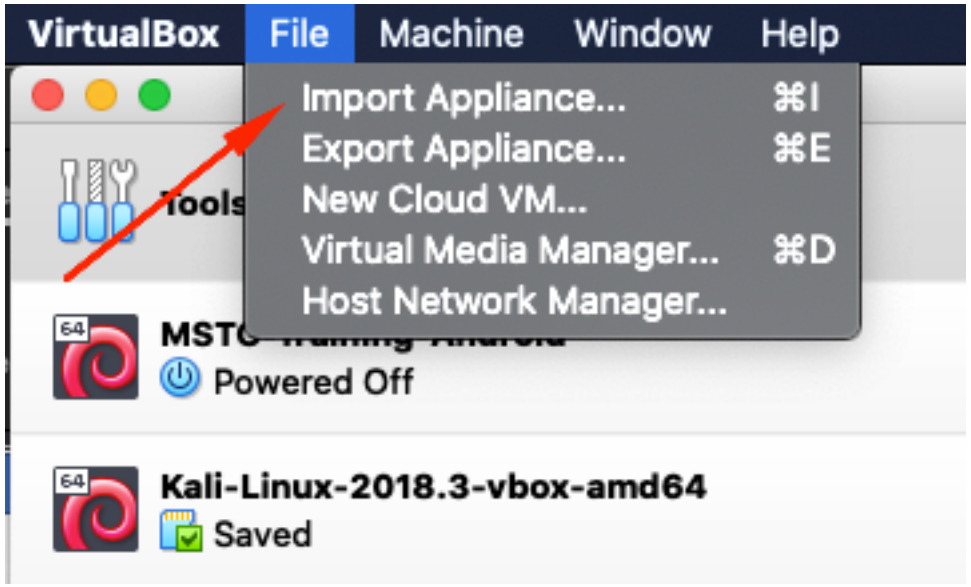
Virtual Machine

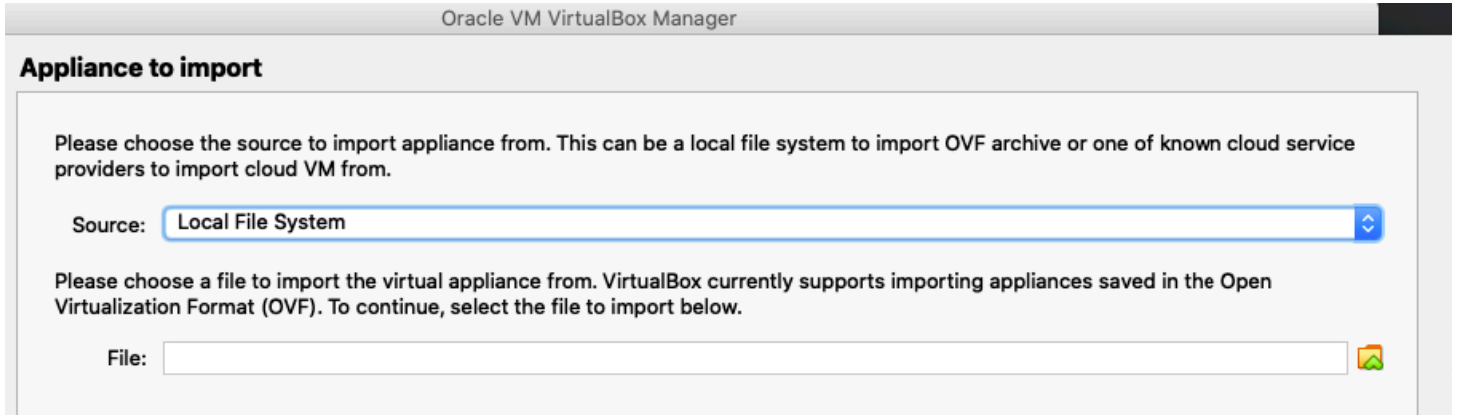
If you don’t have the VM yet, please copy it via the USB flash drive / hard drive to your laptop.

Virtual Machine

Please start VirtualBox!

Setup the VM by clicking on ”Import Appliance” and select the ova file from your local file system.

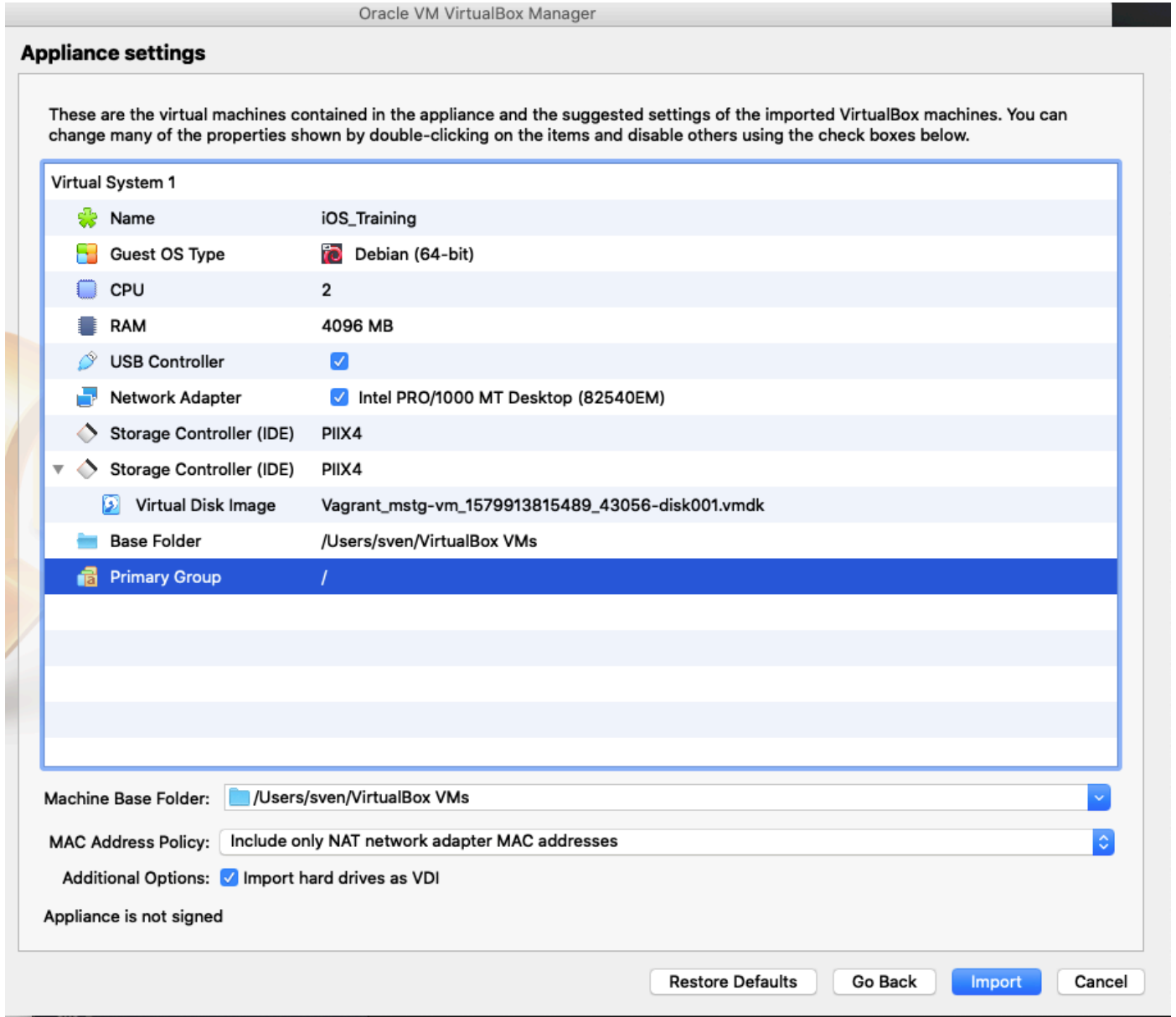




Virtual Machine

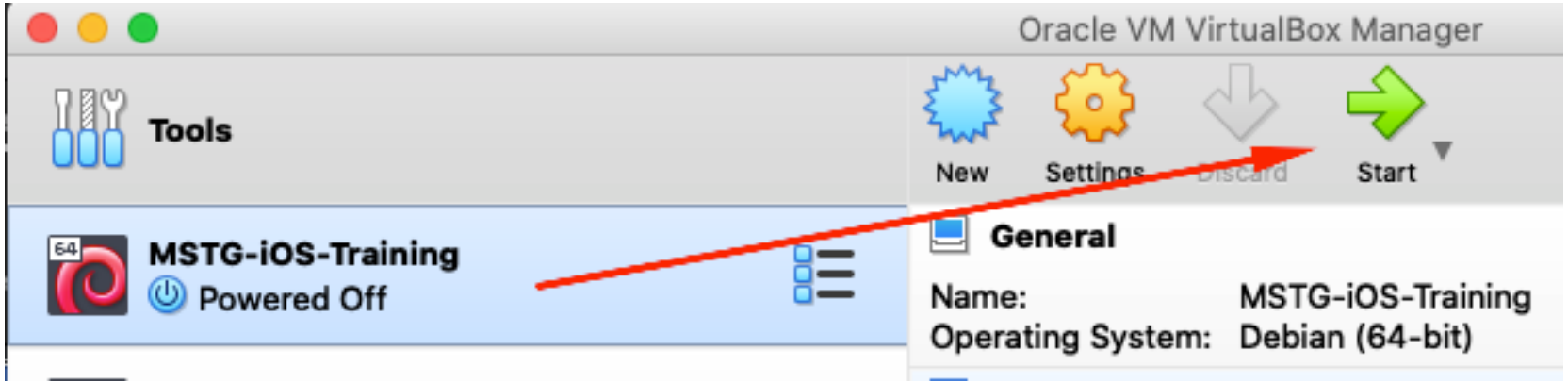
The default settings will work for all modern laptops. Ideally you give the VM 2 CPUs and 4096MB of RAM.

Click on “import” (this might take 10-20 minutes).



Virtual Machine

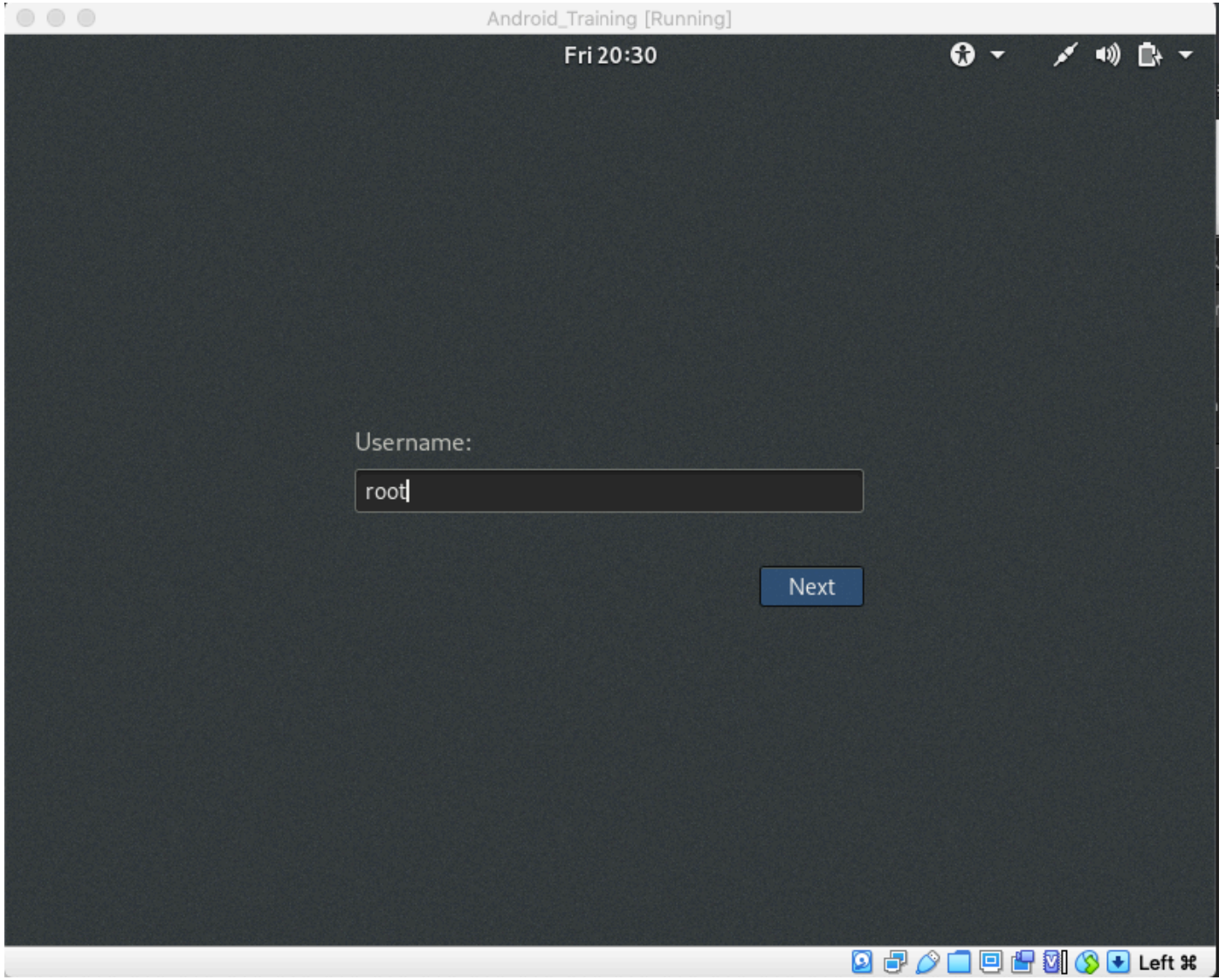
After importing, start the Virtual Machine.



Virtual Machine

Once you see the splash screen swipe up with your mouse. Then you can login with the following credentials:

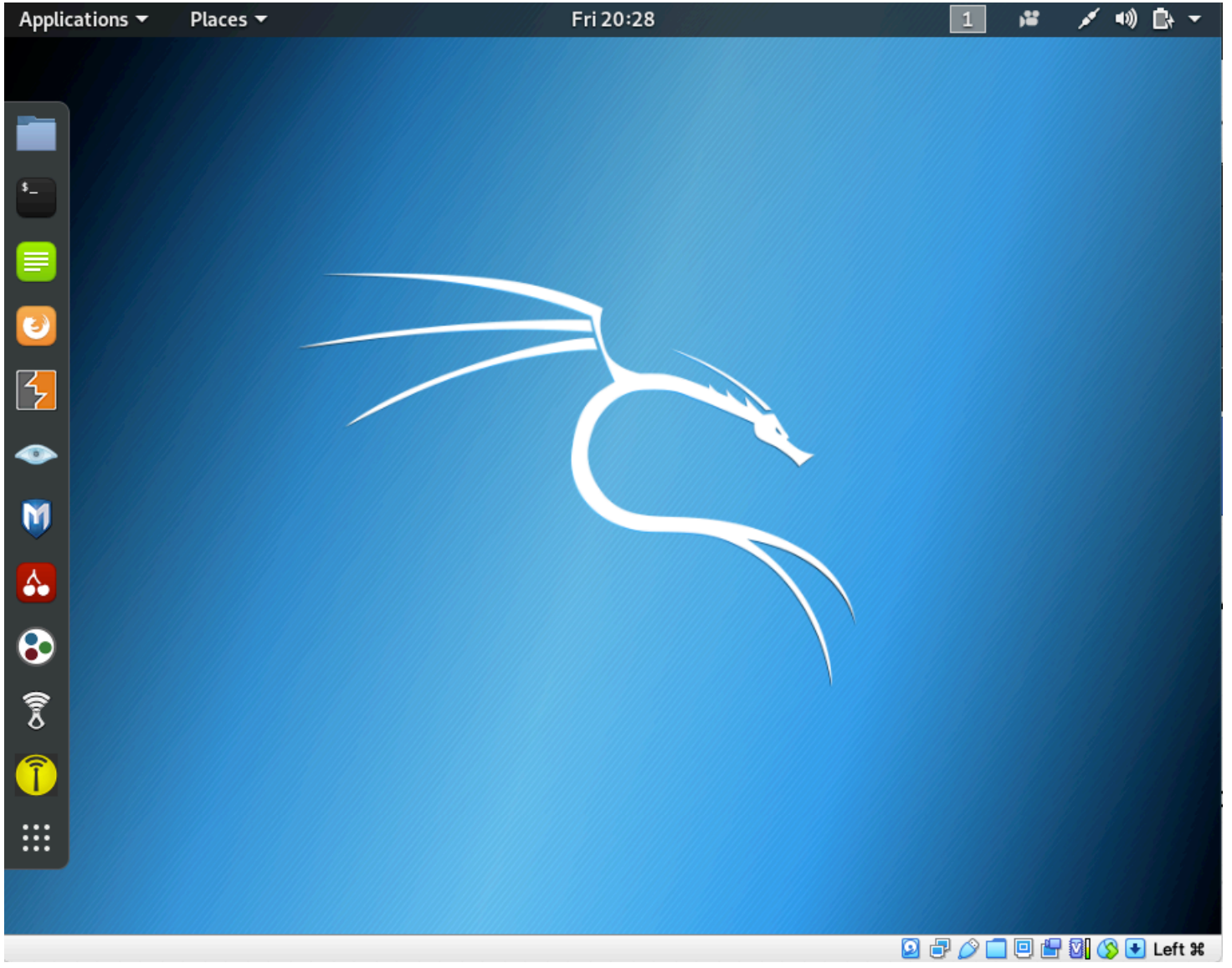
User: root
Password: toor



Virtual Machine

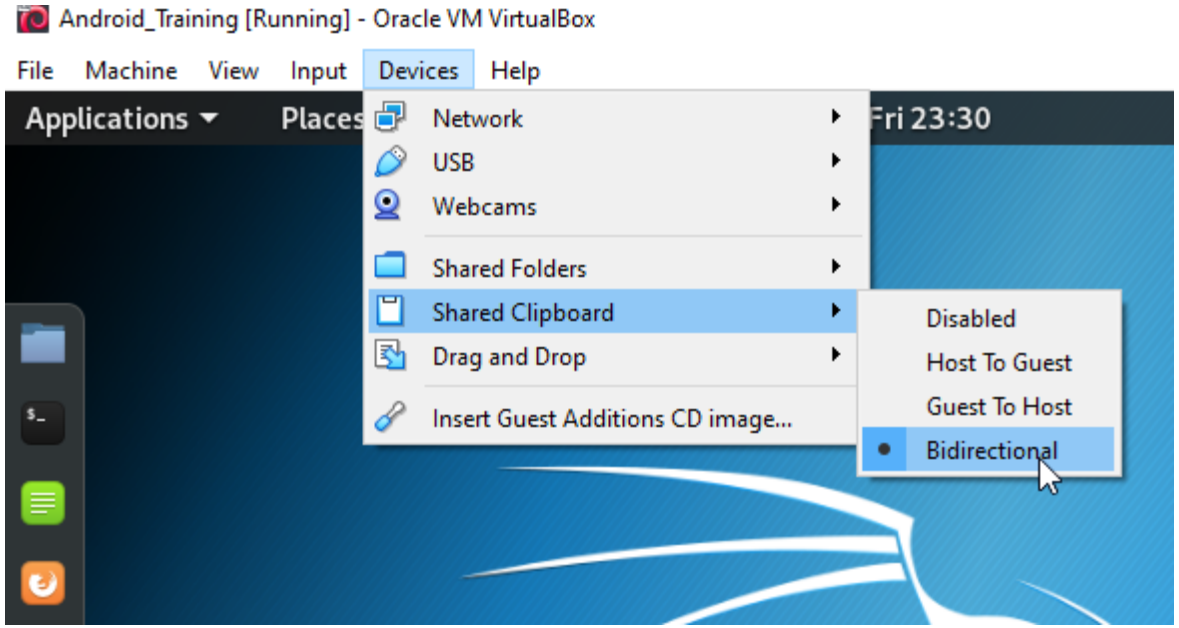
Your Training VM is up and running!

It’s a Kali Linux VM and has already all tools we need for the training pre-installed and configured



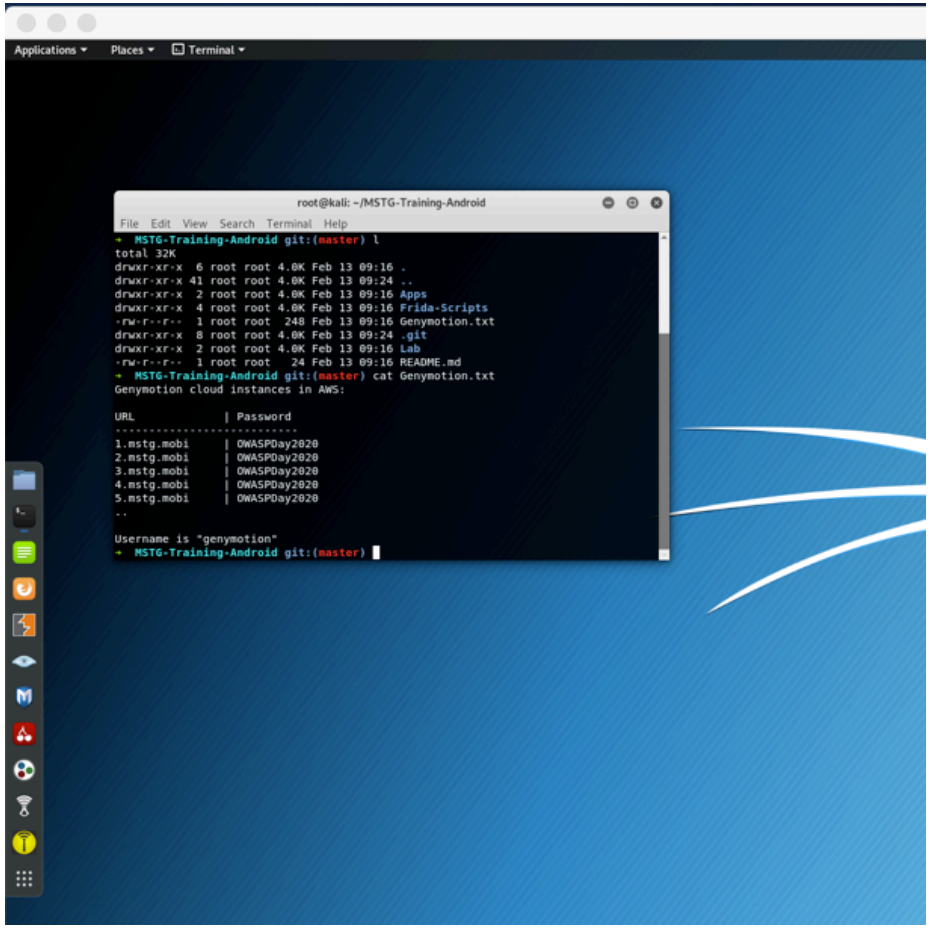
Virtual Machine

Check in “Devices/Shared Clipboard” if the clipboard is activated bidirectional.



Virtual Machine

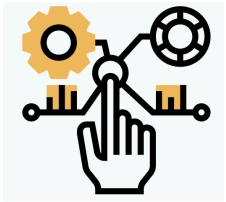
In case the resolution is way too big, or way too small, go to “View / Virtual Screen 1” and set it either to 100% (if too big) or 200% (if too small).



Why Mobile Application Security? Is it really that different from Web App Testing?

Why Mobile Application Security?

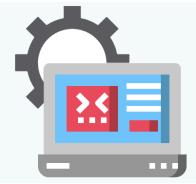
Mobile Applications are different compared to Web Apps:



Interaction with the OS through APIs or other apps through IPC

Data being stored on the device

Local Authentication (scanning fingerprint / face / iris)



Reverse Engineering (e.g. IP)

Why Mobile Application Security?

Do you think it’s possible to bypass Touch ID / Face ID?

a) Yes
b) No

Why Mobile Application Security?

Do you think you can do a fully fledged penetration test on a non-jailbroken device?

a) Yes
b) No

Why Mobile Application Security?

Do you think it’s possible to make an app immune to Reverse Engineering attacks?

a) Yes, through obfuscation
b) Yes, through encryption
c) No, by combing obfuscation and encryption
d) No, the reverse engineer will always win!

Why Mobile Application Security?

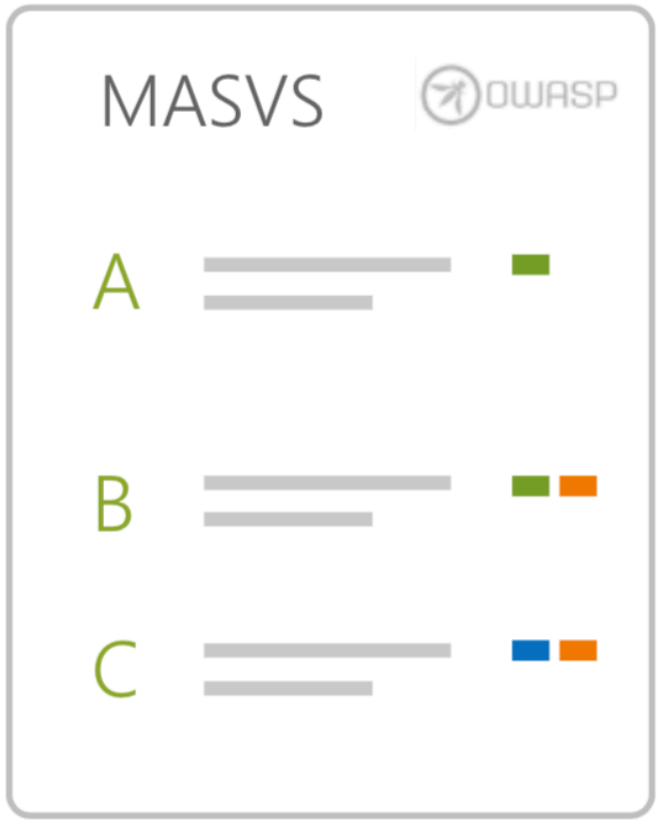
Common server side vulnerabilities are now also on the client-side:

https://twitter.com/natashenka/status/941737682803159040

OWASP Mobile Security Testing Guide
(From the Standard to the Guide)

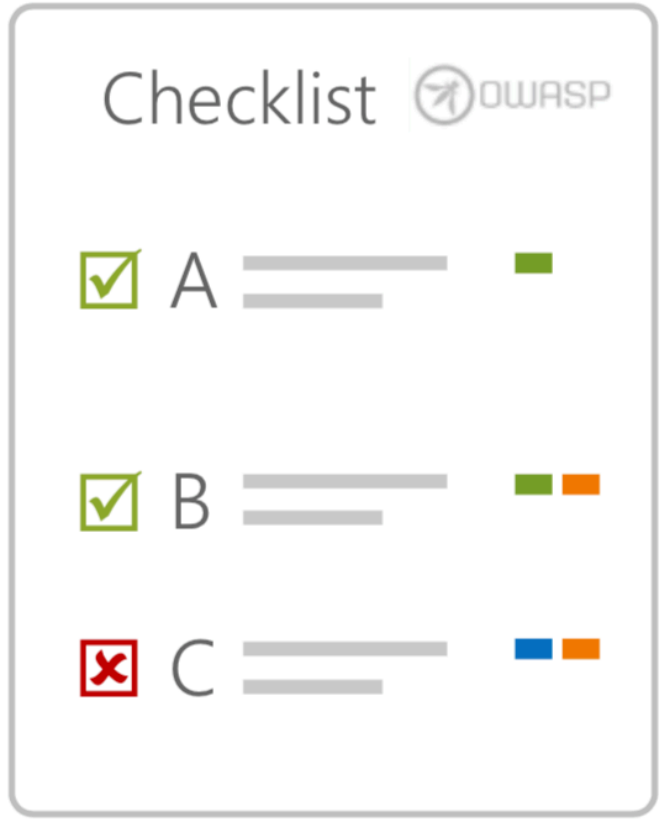
From the Standard to the Guide

Mobile AppSec Verification Standard
https://github.com/OWASP/owasp-masvs/releases



Mobile Security Testing Guide
https://github.com/OWASP/owasp-mstg/

Mobile AppSec Checklist
https://github.com/OWASP/owasp-mstg/tree/master/Checklists



Mobile AppSec Verification Standard (MASVS)

The MASVS is a standard that defines the security requirements for mobile app security and is OS agnostic.

Languages available:

- Chinese (Simplified)
- Chinese (Traditional)
- English
- French
- German
- Japanese
- Korean
- Russian
- Spanish

https://github.com/OWASP/owasp-masvs#getting-the-masvs

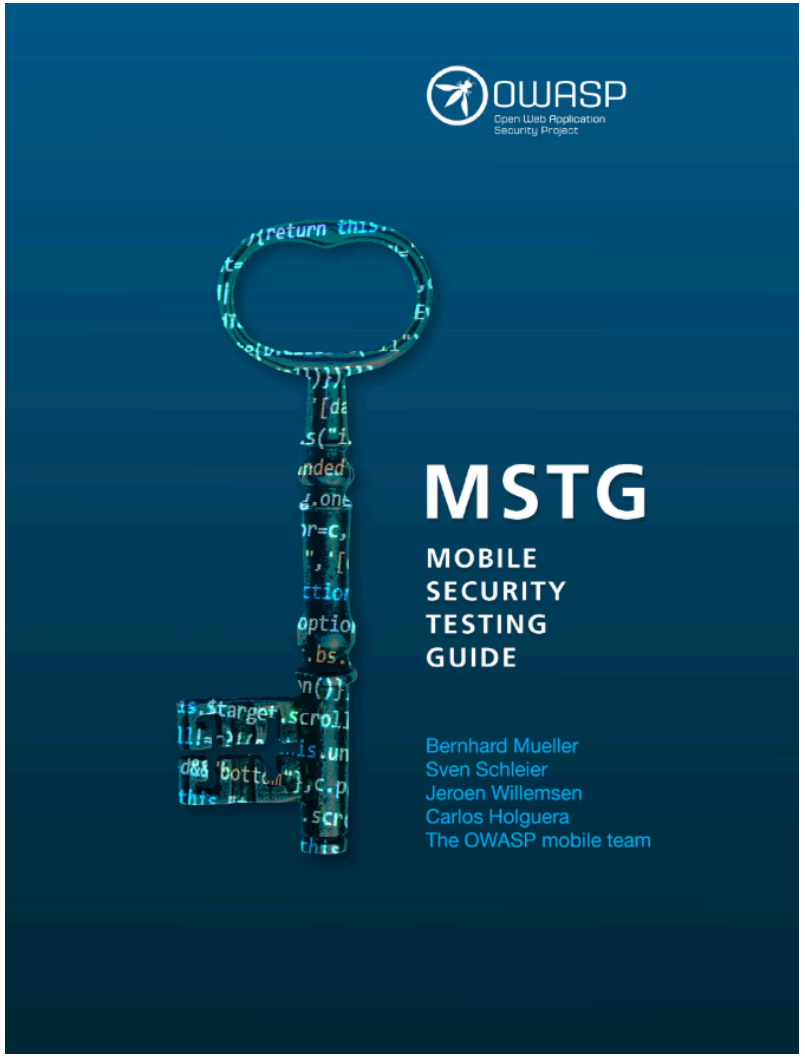
Mobile AppSec Verification Standard (MASVS)

Where to get it:
Github - https://goo.gl/YMCC8B
Gitbook - https://goo.gl/cLqTQE
ePub - https://goo.gl/P7b9Lm
Export as Doc - https://goo.gl/ySSbLJ

- ✓ Download it
- ✓ Read it
- ✓ Use it
- ✓ Give Feedback! Create an issue:

https://github.com/OWASP/owasp-masvs/issues

Mobile Security Testing Guide (MSTG)



The MSTG is a comprehensive manual for mobile app security testing and reverse engineering.
It describes technical processes for verifying the controls listed in the MASVS.

https://github.com/OWASP/owasp-mstg/#reading-the-mobile-security-testing-guide

Mobile Security Testing Guide (MSTG)

MSTG-IDs map requirements from the MASVS to test cases in the MSTG

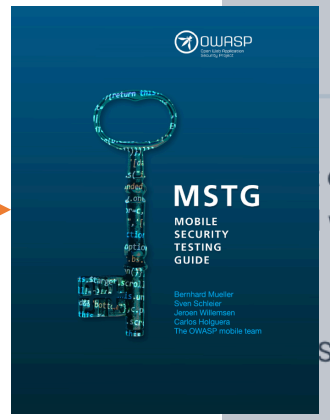
Security Verification Requirements

#	MSTG-ID	Description
5.1	MSTG-NETWORK-1	Data is encrypted on the network using TLS. The secure channel is used consistently throughout the app.
5.2	MSTG-NETWORK-2	The TLS settings are in line with current best pra as possible if the mobile operating system does recommended standards.
5.3	MSTG-NETWORK-3	The app verifies the X.509 certificate of the rem the secure channel is established. Only certifica trusted CA are accepted.
5.4	MSTG-NETWORK-4	The app either uses its own certificate store, or certificate or public key, and subsequently does connections with endpoints that offer a differen even if signed by a trusted CA.
5.5	MSTG-NETWORK-5	The app doesn't rely on a single insecure communication channel (email or SMS) for critical operations, such as enrollments and account recovery.
5.6	MSTG-NETWORK-6	The app only depends on up-to-date connectivity and security libraries.

GitBook

MSTG-Network-3

GitBook



mstg-network-3

Android Network APIs

Testing Endpoint Identify Verification (MSTG-NETW...

Using TLS to transport sensitive information over the network is essential for security. However, encrypting communication between a mobile application...

OWASP Mobile Top 10 2016

...mobile operating system does not support the recommended standards." **MSTG-NETWORK-3**: "The app verifies the X.509 certificate of the remote endpoint...

iOS Network APIs

Testing Custom Certificate Stores and Certificate Pi...

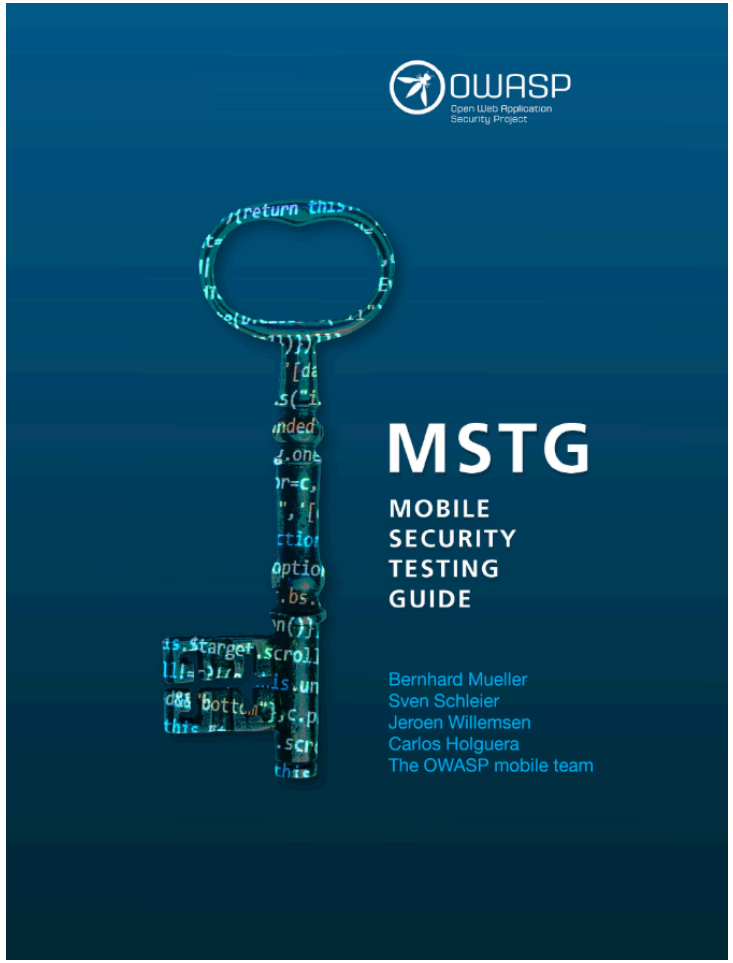
Overview Certificate Authorities are an integral part of a secure client server communication and they are predefined in the trust...

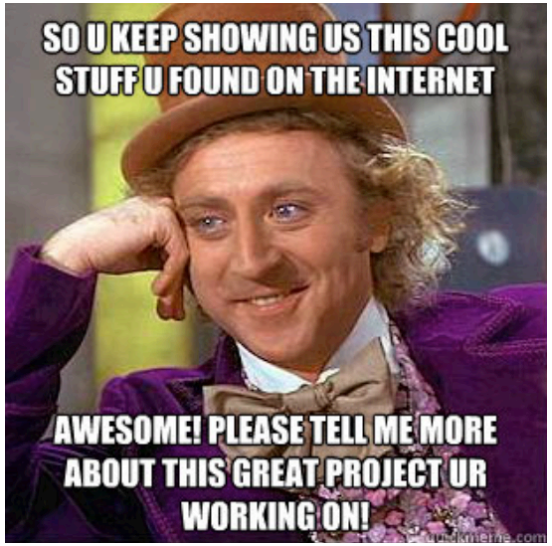
Mobile Security Testing Guide (MSTG)

Where to get it:
Github - https://goo.gl/k5z9Fs
Gitbook - https://goo.gl/SH6bK3
ePub - https://goo.gl/oNCFCJ
Export as Doc - https://goo.gl/FvTftn

- ✓ Download it
- ✓ Read it
- ✓ Use it
- ✓ Give Feedback! Create an issue:

https://github.com/OWASP/owasp-mstg/issues



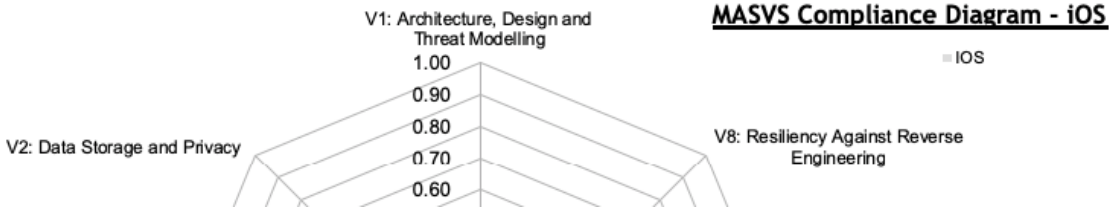


Checklist

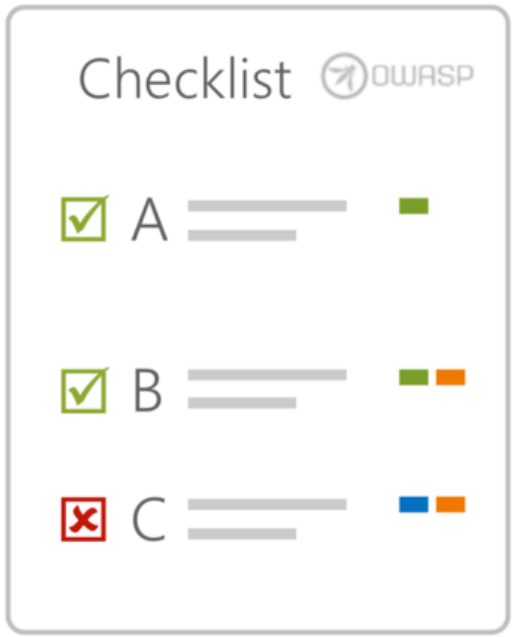
Mobile Application Security Requirements - Android

ID	MSTG-ID	Detailed Verification Requirement	Level 1	Level 2	Status	
V1		**Architecture, design and threat modelling**				
1.1	MSTG-ARCH-1	All app components are identified and known to be needed.	✓	✓		Architectural Information
1.2	MSTG-ARCH-2	Security controls are never enforced only on the client side, but on the respective remote endpoints.	✓	✓		Injection Flaws (MSTG-ARCH-2 and MSTG-PLATFORM-2)
1.3	MSTG-ARCH-3	A high-level architecture for the mobile app and all connected remote services has been defined and security has been addressed in that architecture.	✓	✓		Architectural Information
1.4	MSTG-ARCH-4	Data considered sensitive in the context of the mobile app is clearly identified.	✓	✓		Identifying Sensitive Data
1.5	MSTG-ARCH-5	All app components are defined in terms of the business functions and/or security functions they provide.		✓	N/A	Environmental Information
1.6	MSTG-ARCH-6	A threat model for the mobile app and the associated remote services has been produced that identifies potential threats and countermeasures.		✓	N/A	Mapping the Application
1.7	MSTG-ARCH-7	All security controls have a centralized implementation.		✓	N/A	Testing for insecure Configuration of Instant Apps (MSTG-
1.8	MSTG-ARCH-8	There is an explicit policy for how cryptographic keys (if any) are managed, and the lifecycle of cryptographic keys is enforced. Ideally, follow a key management standard such as NIST SP 800-57.		✓	N/A	Cryptographic policy
1.9	MSTG-ARCH-9	A mechanism for enforcing updates of the mobile app exists.		✓	N/A	Testing enforced updating (MSTG-ARCH-9)
1.10	MSTG-ARCH-10	Security is addressed within all parts of the software development lifecycle.		✓	N/A	Security Testing and the SDLC
V2		**Data Storage and Privacy**				
2.1	MSTG-STORAGE-1	System credential storage facilities are used appropriately to store sensitive data, such as PII, user credentials or cryptographic keys.	✓	✓		Testing Local Storage for Sensitive Data (MSTG-STORAGE
2.2	MSTG-STORAGE-2	No sensitive data should be stored outside of the app container or system credential storage facilities.				Testing Local Storage for Sensitive Data (MSTG-STORAGE
2.3	MSTG-STORAGE-3	No sensitive data is written to application logs.	✓	✓		Testing Logs for Sensitive Data (MSTG-STORAGE-3)
2.4	MSTG-STORAGE-4	No sensitive data is shared with third parties unless it is a necessary part of the architecture.	✓	✓		Determining Whether Sensitive Data is Sent to Third Parti
2.5	MSTG-STORAGE-5	The keyboard cache is disabled on text inputs that process sensitive data.	✓	✓		Determining Whether the Keyboard Cache Is Disabled for
2.6	MSTG-STORAGE-6	No sensitive data is exposed via IPC mechanisms.	✓	✓		Determining Whether Sensitive Stored Data Has Been Exp
2.7	MSTG-STORAGE-7	No sensitive data, such as passwords or pins, is exposed through the user interface.	✓	✓		Checking for Sensitive Data Disclosure Through the User I
2.8	MSTG-STORAGE-8	No sensitive data is included in backups generated by the mobile operating system.		✓	N/A	Testing Backups for Sensitive Data (MSTG-STORAGE-8)
2.9	MSTG-STORAGE-9	The app removes sensitive data from views when moved to the background.		✓	N/A	Finding Sensitive Information in Auto-Generated Screensh
2.10	MSTG-STORAGE-10	The app does not hold sensitive data in memory longer than necessary, and memory is cleared explicitly after use.		✓	N/A	Checking Memory for Sensitive Data (MSTG-STORAGE-10

Security Requirements - Android | Anti-RE - Android | Security Requirements - iOS | Anti-RE - iOS | Version history | +



The Checklist is used in security assessments. It contains links to the MSTG test cases for each requirement.

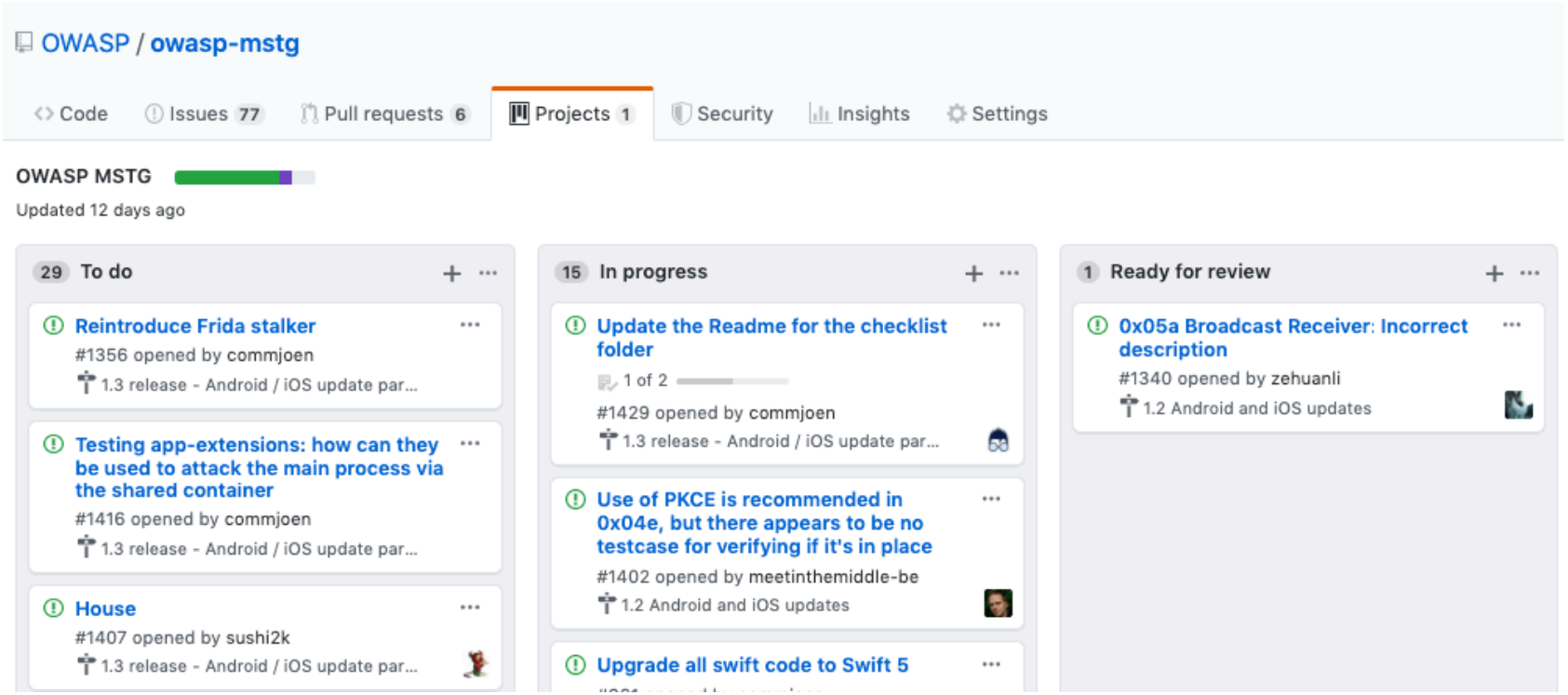


Available in:

- English
- French
- Japanese
- Spanish

https://github.com/OWASP/owasp-mstg/tree/master/Checklists

Contributing to the MSTG / MASVS



Contact us on Github:

cpholguera cpholguera

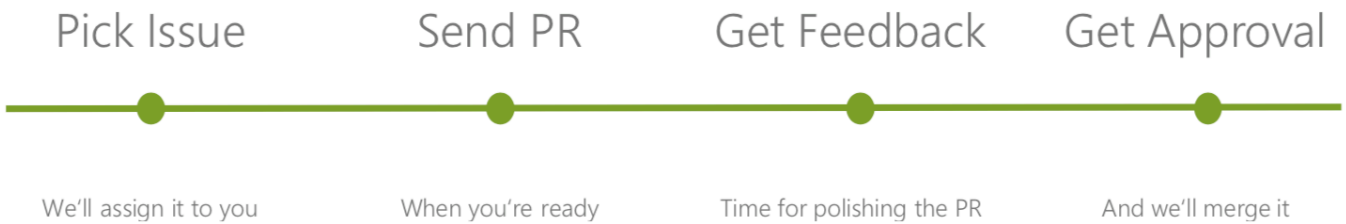
Jeroen Willemsen commjoen

Sven sushi2k

Singapore

Or via OWASP Slack:

#project-mobile_omtg



https://github.com/OWASP/owasp-mstg#contributions-feature-requests-and-feedback

Talks and slides about the MSTG and MASVS

You can find Videos and Slides about the MSTG and MASVS in details on our OWASP project site:

https://owasp.org/www-project-mobile-security-testing-guide/

Presentations

Below you can find a list of upcoming and previous talks:

- OWASP New Zealand Day, February 2020 - Building Secure Mobile Apps (you don't have to learn it the hard way!)
- iOS Conf Singapore, January 2020 - Building Secure iOS Apps (you don't have to learn it the hard way!)
- OWASP AppSec Day Melbourne, October 2019 - Fixing Mobile AppSec
- OWASP Global AppSec Amsterdam, September 2019 - Fast Forwarding mobile security with the OWASP Mobile Security Testing Guide
- r2con in Barcelona, September 2019 - radare2 and Frida in the OWASP Mobile Security Testing Guide
- Open Security summit 2019 - Outcomes
- OWASP Kyiv, April 2019 - OWASP MSTG in real life
- AppDevcon (Amsterdam), March 2019 - Securing your mobile app with the OWASP Mobile Security Testing Guide
- OWASP BeNeLux days 2018 - Fast forwarding mobile security with the MSTG, November 2018
- OWASP Germany days 2018 - Introduction to Mobile Security Testing, November 2018
- DBS AppSecCon (Singapore) - Fixing Mobile AppSec, October 2018
- OWASP Bay Area Chapter - Mobile Testing Workshop, October 2018
- OWASP AppSec USA - Fixing Mobile AppSec, October 2018
- CSC 2018 - A Perspective on Mobile Security in IoT and how OWASP can Help
- OWASP North Sweden Umea - Mobile Security Essentials
- OWASP Gotentburg - Mobile Security Essentials Introduction into OMTG and All about the keying material
- OWASP Day Indonesia 2017 - Fixing Mobile AppSec
- Confidence (Krakow, Poland) - Pawel Rzepa - Testing Mobile Applications
- OWASP AppSec EU 2017 - Fixing Mobile AppSec - Slides, Video

iOS Platform and Security Mechanisms

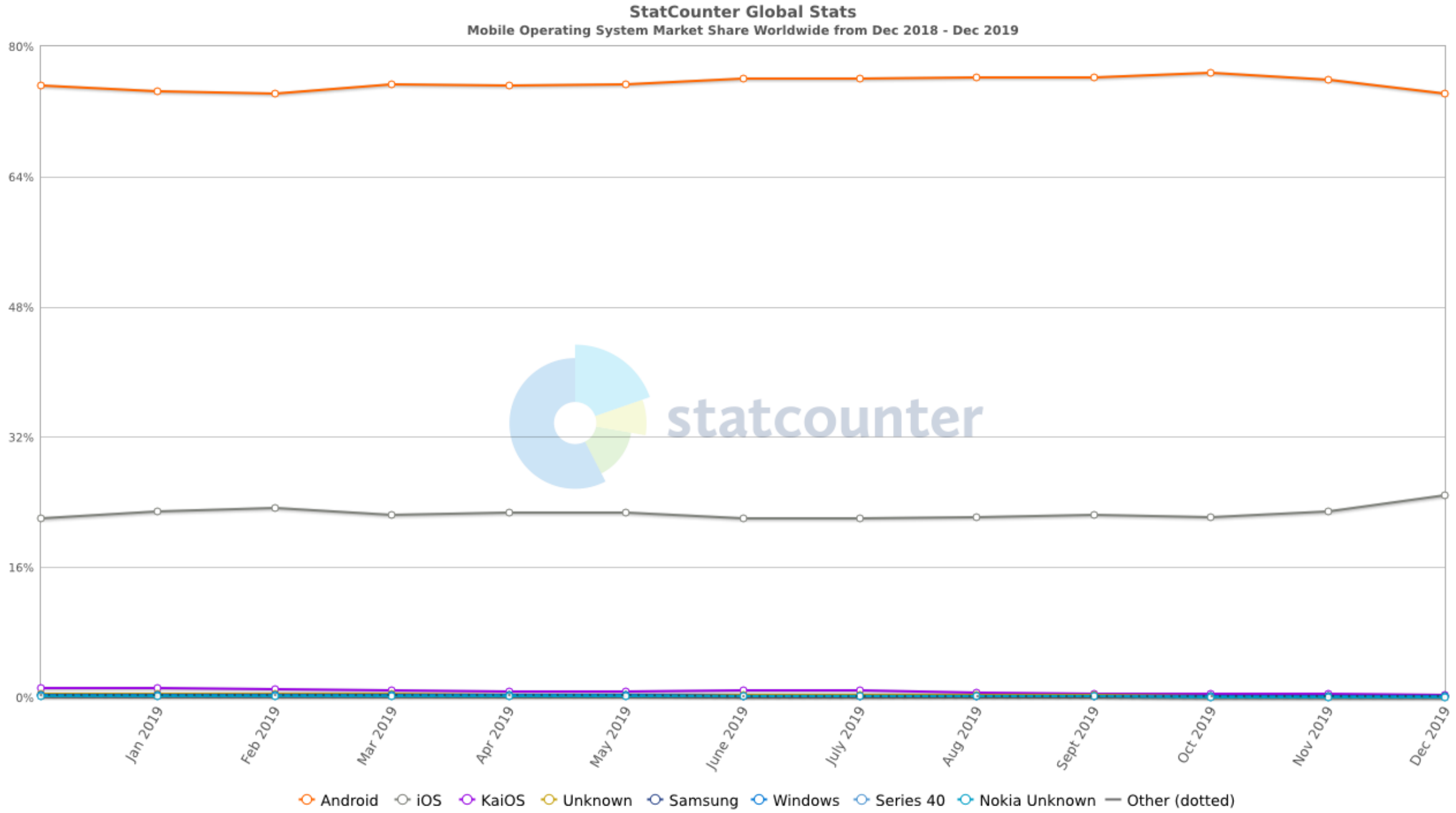
iOS Platform and Security Mechanisms

iOS is a mobile operating system from Apple that is exclusively for its own hardware:

- iPhone
- iPad
- iPod Touch
- Apple Watch
- Apple TV

There are various forks of iOS:

- WatchOS
- tvOS
- iPadOS

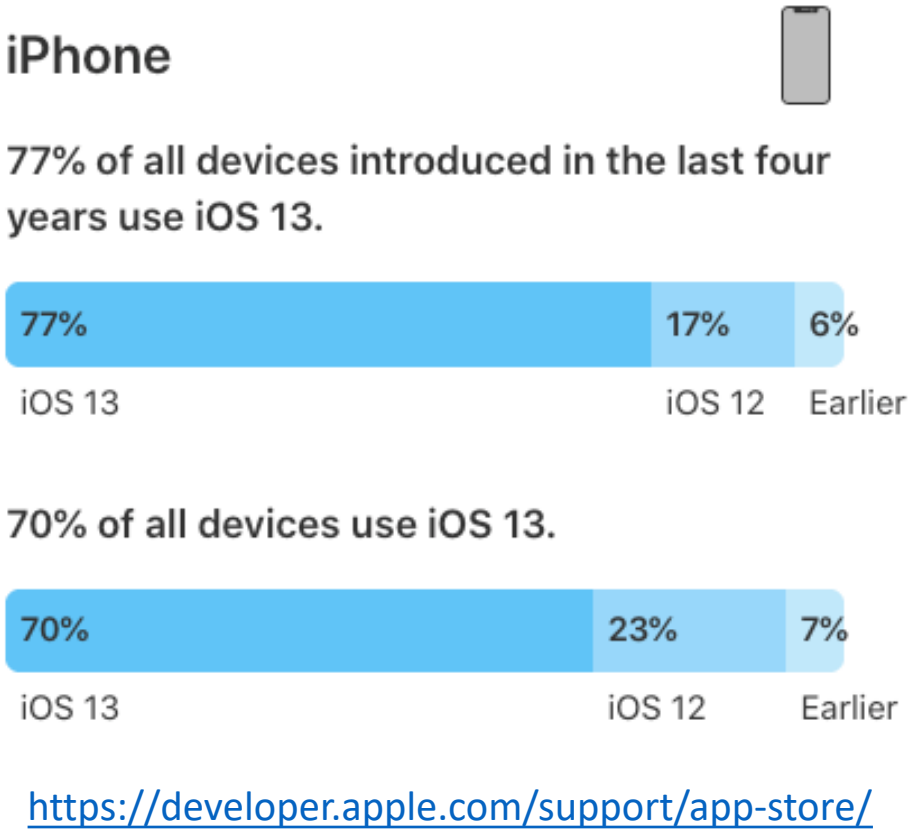


http://gs.statcounter.com/os-market-share/mobile/worldwide

iOS Platform and Security Mechanisms

Compared to Android, the iOS platform is much more homogenous:

- 93% of **all iPhones** are running on iOS 13 or 12
- Developers can assume that all offered hardware and software security features are available for their users:
  - Touch ID
  - Secure Enclave
  - KeyChain

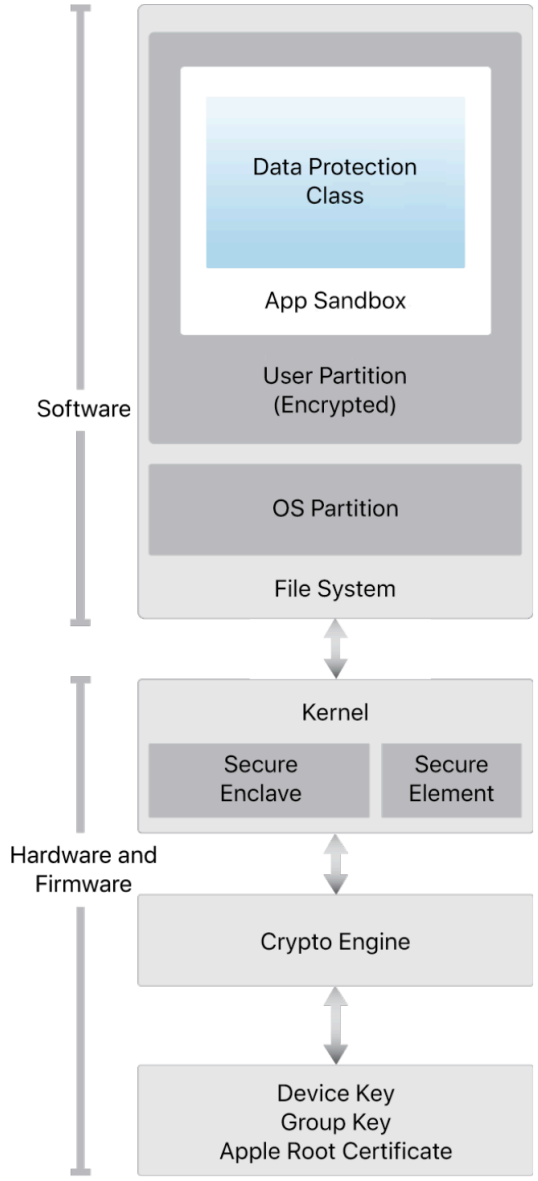


https://developer.apple.com/support/app-store/

iOS Platform and Security Mechanisms

The iOS security architecture consists of six core features:

- Hardware Security (Secure Enclave)
- Secure Boot
- Code Signing
- Sandbox
- Encryption and Data Protection
- General Exploit Mitigations



https://www.apple.com/business/docs/iOS_Security_Guide.pdf
https://www.blackhat.com/docs/us-16/materials/us-16-Krstic.pdf

iOS Platform and Security Mechanisms

Code Signing

- Only Apple approved code is allowed to be executed on an iOS device.
- Enforces to stay “up-to-date” (you cannot downgrade to earlier iOS versions)
- Before an app can be installed on a device, or be submitted to the App Store, it must be signed with a certificate issued by Apple (Xcode is taking care of that).
- Need to be enrolled in the Apple Developer Program or Enterprise Program (but you can also use a free Apple account to sign apps for ~7 days)

iOS Platform and Security Mechanisms

Sandboxing

iOS apps are isolated from each other at the file system level (Sandboxing)
Apple's sandbox, historically referred to as Seatbelt, is a mandatory access control (MAC) mechanism based on FreeBSD's TrustedBSD framework

But why is it the same user and group (mobile) for all installed apps?

```
Svens-iPhone:/var/mobile/Containers/Data/Application root# ls -lha
total 0
drwxr-xr-x 83 mobile mobile 2.6K Oct 27 20:08 ./
drwxr-xr-x  8 root   mobile  256 Aug 20 06:13 ../
drwxr-xr-x  7 mobile mobile  224 Aug 20 06:14 06B5C33C-3CFF-4F85-820B-B1ABDC7733FD/
drwxr-xr-x  7 mobile mobile  224 Aug 20 06:14 07DE3D66-0608-49F8-BB7F-14ECA50B43C9/
drwxr-xr-x  7 mobile mobile  224 Aug 20 06:53 093D3E24-BEE1-4741-B3FE-B21B2841AF70/
drwxr-xr-x  7 mobile mobile  224 Aug 20 06:14 0B62F15E-0C53-4AE5-AF23-8DC2BB1628E0/
drwxr-xr-x  7 mobile mobile  224 Aug 20 06:15 0CCBFC93-623C-4FF3-B513-608FC3DD1660/
drwxr-xr-x  7 mobile mobile  224 Aug 20 06:16 0FD9B1EF-E80D-4CFD-B5DF-9E446A839001/
drwxr-xr-x  7 mobile mobile  224 Aug 20 06:14 0FE2389A-0B82-4D92-9462-E90447C7D3A3/
drwxr-xr-x  7 mobile mobile  224 Aug 20 06:14 12280751-FBC8-4467-91DE-72D132349A55/
```

iOS Platform and Security Mechanisms

Sandboxing

App entitlements are used to manage access from apps in iOS

https://developer.apple.com/documentation/security/app_sandbox_entitlements
https://developer.apple.com/app-sandboxing/

iOS Platform and Security Mechanisms

Encryption and Data Protection

- Every app downloaded from the App Store has “FairPlay Code Encryption”
- FairPlay-encrypted Apps can only be decrypted on devices associated with your Apple user account (public/private key)
- The Apple App store is the only official application distribution platform

iOS Platform and Security Mechanisms

iOS Apps can only be developed on macOS, due to the requirement of Xcode.

Programming languages to develop native iOS Apps are:

- Objective-C and

- Swift Version [1 | 1.1 | 1.2 | 2 | 3 | 4.0 | 4.1 | 4.2 | 5.0 | 5.1 | 5.2]

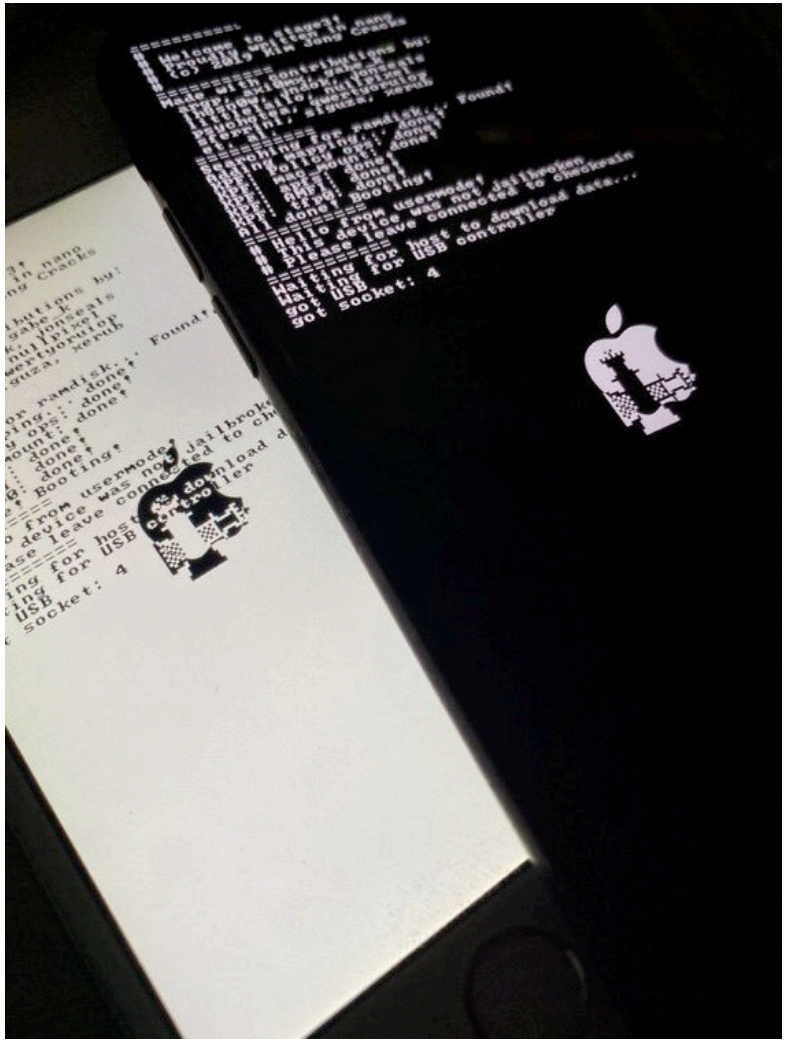
iOS Platform and Security Mechanisms (are 0wned)

CheckM8 - bootrom exploit, Apple cannot patch it!

- Exploit (not a Jailbreak) working for any iPhone device since 2010 (iPhone 4S to iPhone X)
- Downgrade to any iOS version
- USB Access needed (no remote exploit!)

https://github.com/axi0mX/ipwndfu

http://pangu8.com/jailbreak/checkra1n/



iOS Platform and Security Mechanisms (are 0wned)

https://twitter.com/axi0mx/status/1177542201670168576

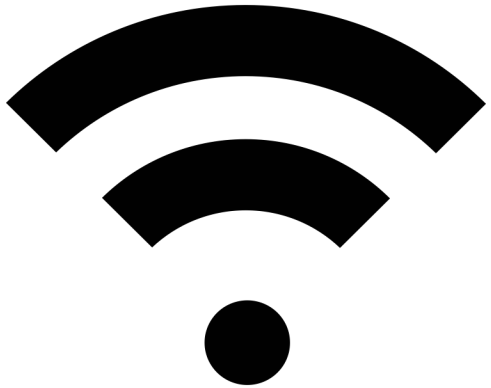

iOS Device

iOS Device

Please bring your iOS device to me.

I would need to install one dummy app and register the device to my apple developer account!

Wifi

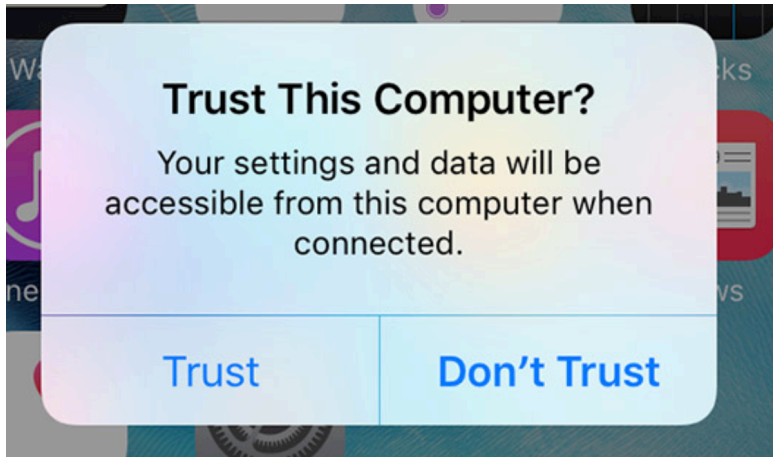


Please connect your iOS device to the Wifi.

iOS Device

Please connect your iOS device via the USB cable to your Mabook.

If you connect the device the first time, check your iOS device for a message. You should be getting a prompt to “Trust this computer” and you need to allow the access!

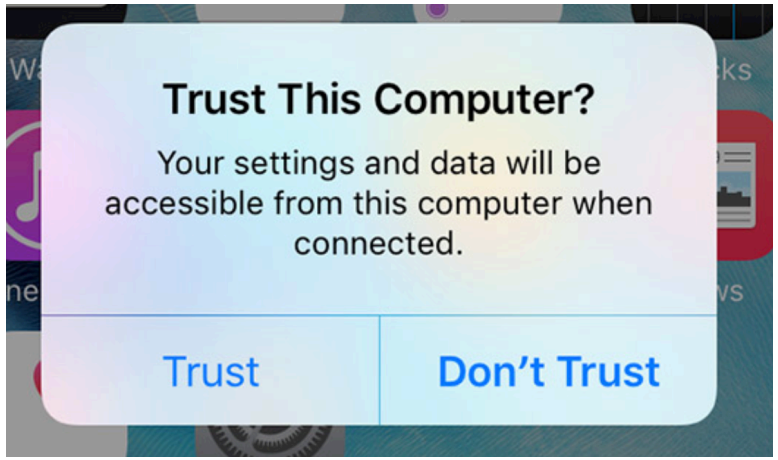


Don’t’ connect the iOS device to the VM!
Just use it with macOS!

iOS Device

If you are having issues connecting to it, execute the following command, then the “Trust this computer” dialog will pop-up:

```
$ instruments -s devices
```



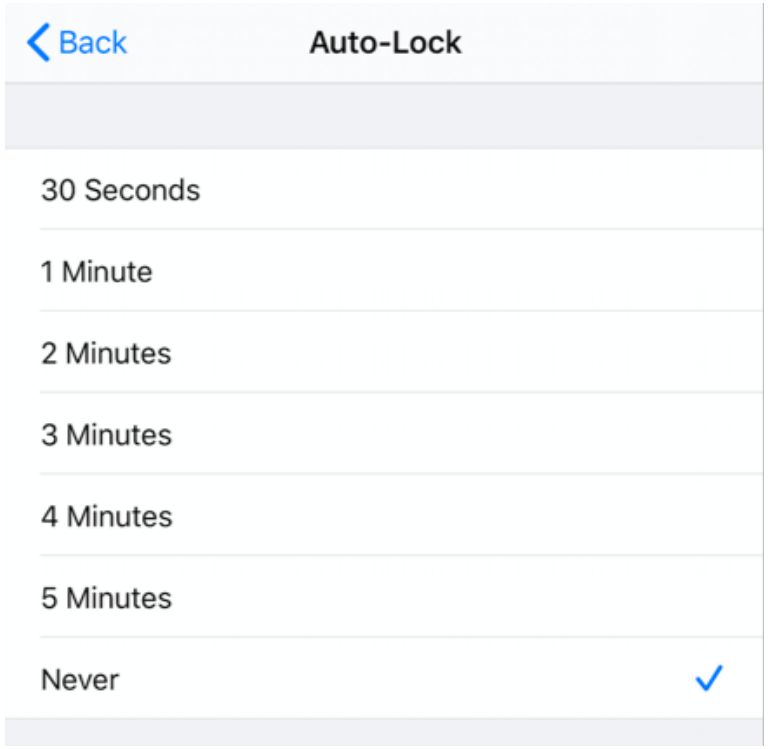
Don’t’ connect the iOS device to the VM!
Just use it with macOS!

iOS Device

Increase the time for the Auto-Lock!

Go to Settings / Display & Brightness / Auto-lock and set it to either 5 minutes or Never

<u>If the device is getting locked while you are analyzing an app, tools might break, or you get timeouts!</u>



iOS Security Testing Basics

iOS Security Testing Basics – Real device vs. Simulator

Simulators

- Simulator (part of Xcode)
- Corellium (https://corellium.com/) – (sued by Apple ☹)

There are several downsides to using an emulator. You may not be able to test an app properly in an emulator if the app relies on a specific mobile network or uses Bluetooth.

Testing within an emulator is also usually slower, and the testing itself may cause issues.

There are also limitations to SMS and Push notifications, but it's also possible to simulate.

iOS Security Testing Basics – Real device vs. Simulator

Real Device

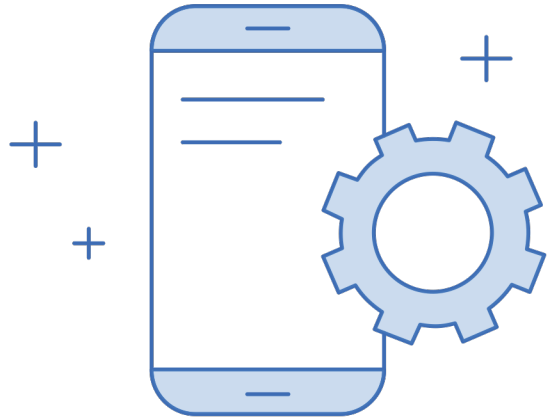
- Testing on a real device makes for a smoother process and a more realistic environment.
- iOS devices can be “jailbroken” (modifying the OS so that you can run commands as the root user)
- For an overview of available jailbreaks, see here https://canijailbreak.com/

For this training you will be using your iOS device and Corellium!

iOS Security Testing Basics – Install an IPA

Installation of iOS Apps via sideloading*:

- Cydia Impactor (macOS, Windows, Linux)
- Xcode (macOS only)
- ios-deploy (macOS only)
- ideviceinstaller (Linux only)



=> Since version 12.7.0 it’s not possible to install IPAs via iTunes anymore (option removed!)

**Sideloading: Installing an IPA to an iOS device without the Apple App Store.*

iOS Security Testing Basics – Install an IPA

Cydia Impactor (macOS, Windows, Linux)

Cydia Impactor

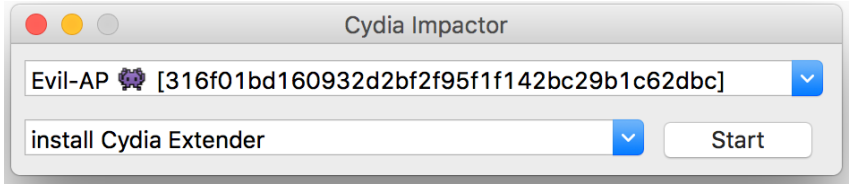
Cydia Impactor is a GUI tool for working with mobile devices. It has features already, but is still very much a work-in-progress. It is developed by saurik (Twitter and website).

You can use this tool to install IPA files on iOS and APK files on Android. It also can help you exploit the series of Android "Master Key" vulnerabilities.

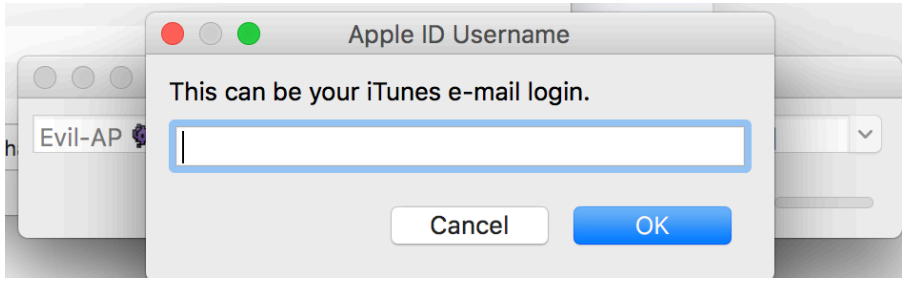
http://www.cydiaimpactor.com/

Make sure to use an app password for Cydia Impactor!
https://support.apple.com/en-ke/HT204397

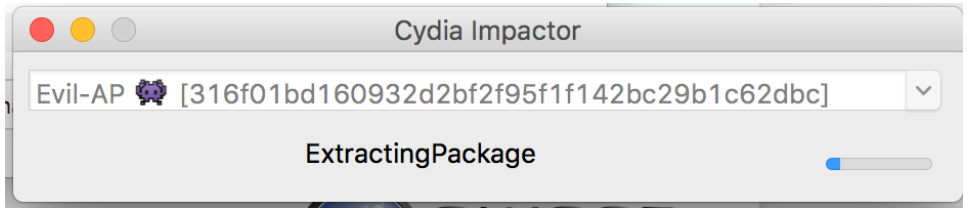
Start Cydia Impactor and drag n drop the IPA



Create a dummy iTunes Account that you can use with Cydia Impactor

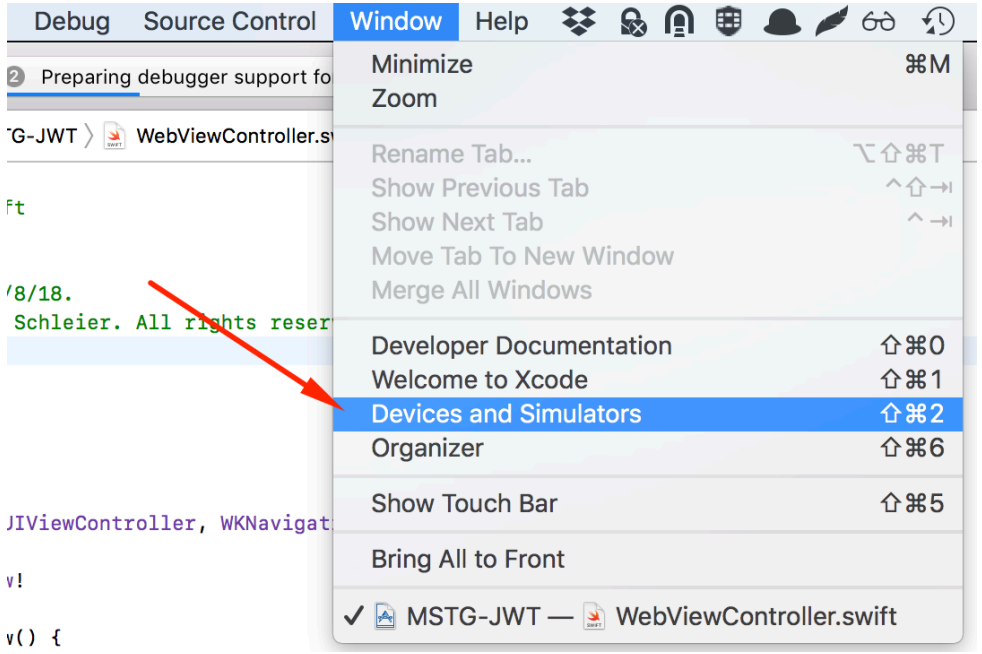


IPA is getting installed

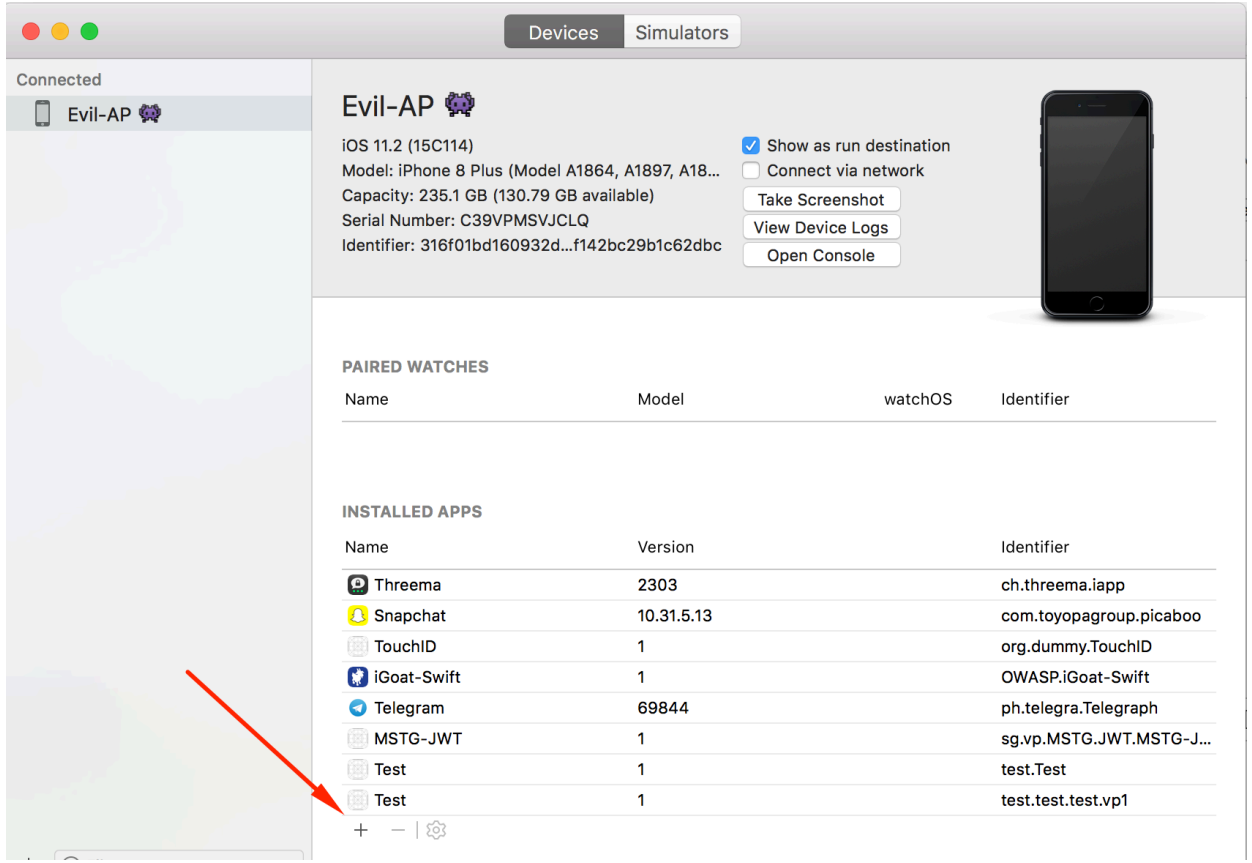


iOS Security Testing Basics – Install an IPA

Xcode (macOS only)



Select Window / Devices and Simulators



Click on the + sign and select the IPA that you want to install

iOS Security Testing Basics – Install an IPA

ios-deploy (macOS only) – "Install and debug iPhone apps from the command line, without using Xcode"

- Preferred way of side-loading apps on macOS
- Install apps with "ios-deploy -b Test.ipa"
- For debugging apps, unzip the app and execute ios-deploy with the following command:

```
➜ MSTG-JWT 2018-08-29 05-42-27 git:(master) ✗ ios-deploy --bundle Payload/MSTG-JWT.app -W -d
[....] Waiting for iOS device to be connected
[....] Using 316f01bd160932d2bf2f95f1f142bc29b1c62dbc (D211AP, D211AP, uknownos, unkarch) a.k.a.
'Evil-AP 👾.
------ Install phase ------
[  0%] Found 316f01bd160932d2bf2f95f1f142bc29b1c62dbc (D211AP, D211AP, uknownos, unkarch) a.k.a.
'Evil-AP 👾 connected through USB, beginning install
[  5%] Copying /Users/sven/VantagePoint/Conferences/OWASP-AppSec-USA-2018/Training/git/MSTG_AppSe
c_USA/Apps/iOS/MSTG-JWT/MSTG-JWT 2018-08-29 05-42-27/Payload/MSTG-JWT.app/META-INF/ to device
[  5%] Copying /Users/sven/VantagePoint/Conferences/OWASP-AppSec-USA-2018/Training/git/MSTG_AppSe
c_USA/Apps/iOS/MSTG-JWT/MSTG-JWT 2018-08-29 05-42-27/Payload/MSTG-JWT.app/META-INF/com.apple.ZipM
etadata.plist to device
```

https://github.com/ios-control/ios-deploy

iOS Security Testing Basics – Install an IPA

Ideviceinstaller (Linux and macOS)

Once you installed the IPA with **ideviceinstaller**, you can start the app with **idevicedebug**, by specifying the bundle-id of the app:

```
$ ideviceinstaller -i Test.ipa

$ idevicedebug -d run org.dummy.mstg.Test
```

The bundle-id will be shown in the output, when installing the app via **ideviceinstaller**. Otherwise you can find the bundle-id (org.dummy.mstg.Test) in the Info.plist after unzipping the IPA.

Install IPA and Basic Analysis

Let's do our first Lab!

Open the file Lab-iOS.pdf:

```
$ evince ~/MSTG-Handson/Lab/Lab-iOS.pdf
```

Section: Lab – Install IPA and Basic Dynamic Analysis

Please type the commands and do not copy & paste!

1. You will remember them easier
2. Some PDF readers do not copy properly and miss whitespaces etc.

Install IPA and Basic Analysis

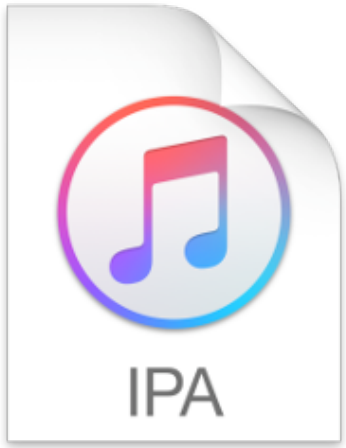
Uninstall the MSTG-JWT app from your iOS device that we just installed!

iOS Application Structure

iOS – Application Structure

What is an IPA?

- iOS apps are distributed as IPA (iOS App Store Package) archives.
- The IPA file is a ZIP-compressed archive that contains all the code and resources required to execute the app.



iOS – Application Structure

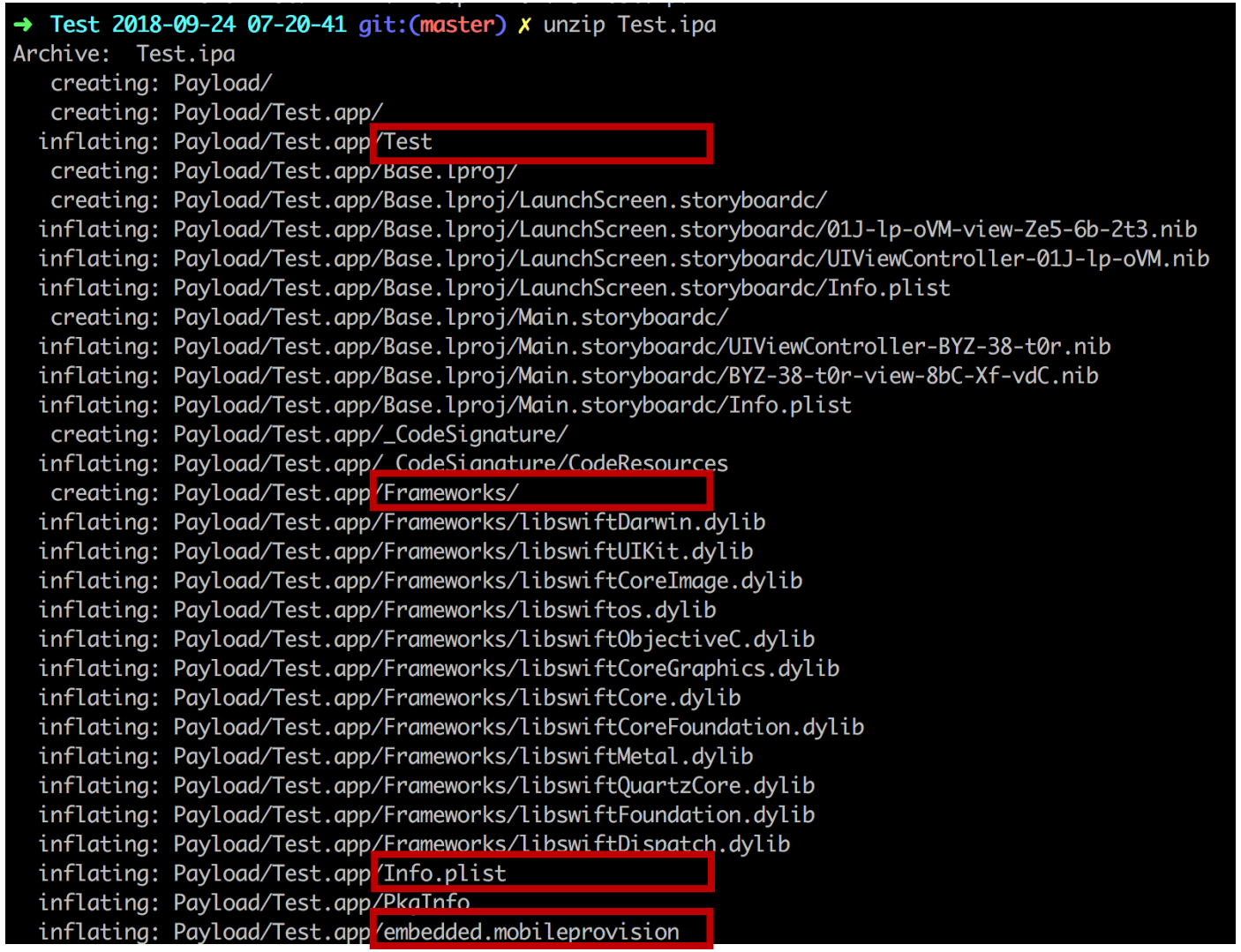
You can unzip the IPA using the standard tool unzip or any other ZIP utility. Inside you'll find a **Payload** folder containing the so-called Application Bundle (<app-name>.app). The following is an example :

```
➜  Test 2018-09-24 07-20-41 git:(master) ✗ unzip Test.ipa
Archive:  Test.ipa
   creating: Payload/
   creating: Payload/Test.app/
  inflating: Payload/Test.app/Test
   creating: Payload/Test.app/Base.lproj/
   creating: Payload/Test.app/Base.lproj/LaunchScreen.storyboardc/
  inflating: Payload/Test.app/Base.lproj/LaunchScreen.storyboardc/01J-lp-oVM-view-Ze5-6b-2t3.nib
  inflating: Payload/Test.app/Base.lproj/LaunchScreen.storyboardc/UIViewController-01J-lp-oVM.nib
  inflating: Payload/Test.app/Base.lproj/LaunchScreen.storyboardc/Info.plist
   creating: Payload/Test.app/Base.lproj/Main.storyboardc/
  inflating: Payload/Test.app/Base.lproj/Main.storyboardc/UIViewController-BYZ-38-t0r.nib
  inflating: Payload/Test.app/Base.lproj/Main.storyboardc/BYZ-38-t0r-view-8bC-Xf-vdC.nib
  inflating: Payload/Test.app/Base.lproj/Main.storyboardc/Info.plist
   creating: Payload/Test.app/_CodeSignature/
  inflating: Payload/Test.app/_CodeSignature/CodeResources
   creating: Payload/Test.app/Frameworks/
  inflating: Payload/Test.app/Frameworks/libswiftDarwin.dylib
  inflating: Payload/Test.app/Frameworks/libswiftUIKit.dylib
  inflating: Payload/Test.app/Frameworks/libswiftCoreImage.dylib
  inflating: Payload/Test.app/Frameworks/libswiftos.dylib
  inflating: Payload/Test.app/Frameworks/libswiftObjectiveC.dylib
  inflating: Payload/Test.app/Frameworks/libswiftCoreGraphics.dylib
  inflating: Payload/Test.app/Frameworks/libswiftCore.dylib
  inflating: Payload/Test.app/Frameworks/libswiftCoreFoundation.dylib
  inflating: Payload/Test.app/Frameworks/libswiftMetal.dylib
  inflating: Payload/Test.app/Frameworks/libswiftQuartzCore.dylib
  inflating: Payload/Test.app/Frameworks/libswiftFoundation.dylib
  inflating: Payload/Test.app/Frameworks/libswiftDispatch.dylib
  inflating: Payload/Test.app/Info.plist
  inflating: Payload/Test.app/PkgInfo
  inflating: Payload/Test.app/embedded.mobileprovision
```

iOS – Application Structure – IPA

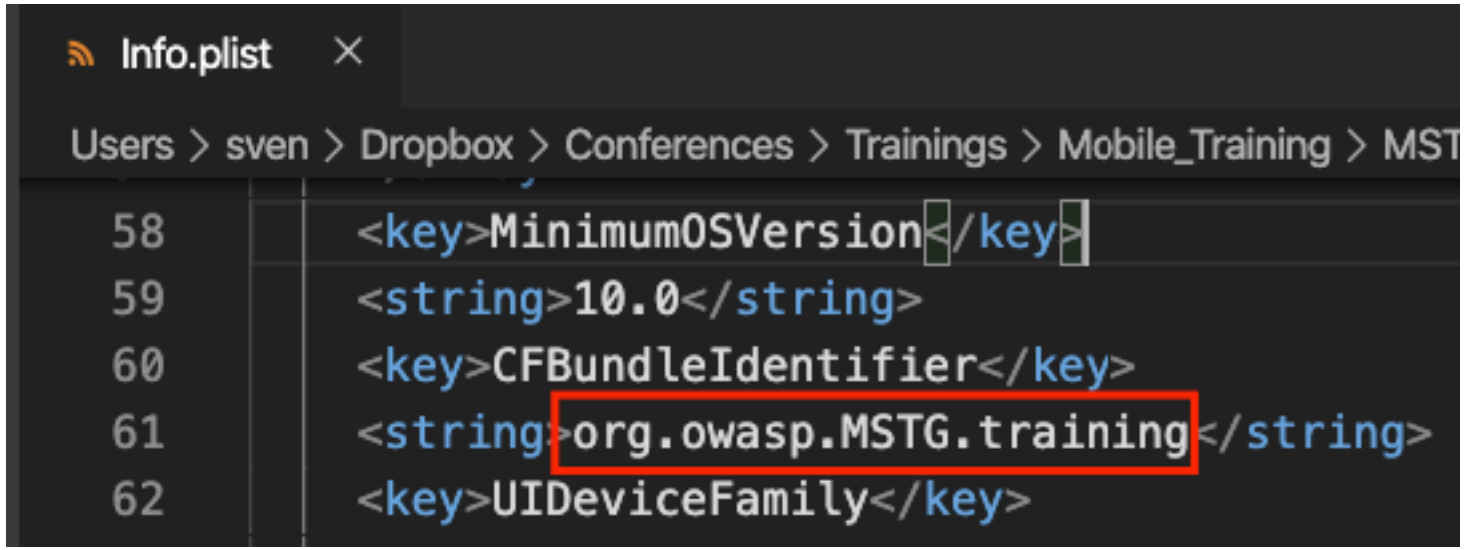
File Structure of an IPA:

- App binary (Test): The executable file containing the compiled application binary
- Frameworks: Dynamic libraries
- Info.plist: Application metadata
- Embedded.mobileprovsion: Certificates and entitlement group



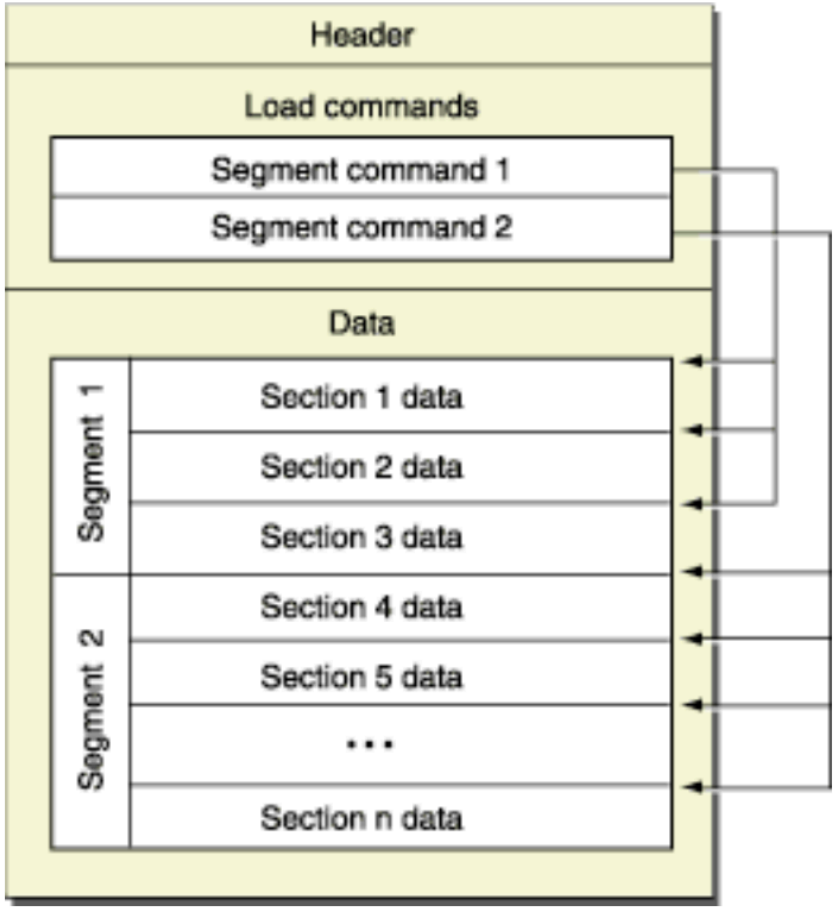
iOS – Application Structure – IPA

Bundle Identifier can be
found in Info.plist



iOS – Mach-o file structure

- Application binaries are in Mach-o file format.
- Mach-O, short for Mach object file format, is a file format for executables, object code, shared libraries and dynamically-loaded code.
- Main sections are Header, Load and Data
- Once installed, apps are stored inside the folder /var/mobile/Applications/{GUID}



If you want to know more details, you can read up about mach-o file structure here:
https://github.com/aidansteele/osx-abi-macho-file-format-reference

iOS Testing with Jailbroken and Non-Jailbroken device

iOS – Security Testing with and without Jailbreak

How can a penetration tester test iOS Apps?

Jailbroken device

- Cydia App Store
- Full Root Access

Dynamic instrumentation

- Works on (non-)jailbroken devices
- Manipulate runtime behaviour of an app through Frida

Objection is a runtime mobile exploration toolkit, powered by Frida.

See also:

Frida iOS: https://www.frida.re/docs/ios/

Objection: https://github.com/sensepost/objection

iOS Basic Security Testing: https://bit.ly/2lHdGoj

iOS Dynamic Testing on non jailbroken device: https://bit.ly/2lG7Kf7

Jailbreak

The purpose of jailbreaking is to disable iOS protections (Apple's code signing mechanisms in particular) so that arbitrary unsigned code can run on the device. The word "jailbreak" is a colloquial reference to all-in-one tools that automate the disabling process.

Different types:

- Tethered jailbreaks (connect to a computer during reboot)
- Semi-tethered jailbreaks (can also boot to non-jb)
- Semi-untethered jailbreaks (re-jailbreaking)
- Untethered jailbreaks (applied once)

See also:
https://mobile-security.gitbook.io/mobile-security-testing-guide/ios-testing-guide/0x06b-basic-security-testing#getting-privileged-access

Can I Jailbreak?

by IPSW Downloads

Home Get Jailbreaking Help

My iOS device is on **iOS 12.3 → 13.3**

✔ Jailbreak using checkra1n.

Currently in beta. Supports devices iPhone 5s to iPhone X. Semi-tethered. Windows version coming soon, some device support not fully tested yet.

iOS 12.4

✔ Jailbreak using unc0ver

Semi-Untethered Only. Follow the Cydia Impactor method of installing the Jailbreak.

iOS 12.1.3 → 12.2

✔ Jailbreak using Chimera

Semi-Untethered Only. Follow the Cydia Impactor method of installing the Jailbreak. A12 devices are not supported. Sileo installed by default (not Cydia).

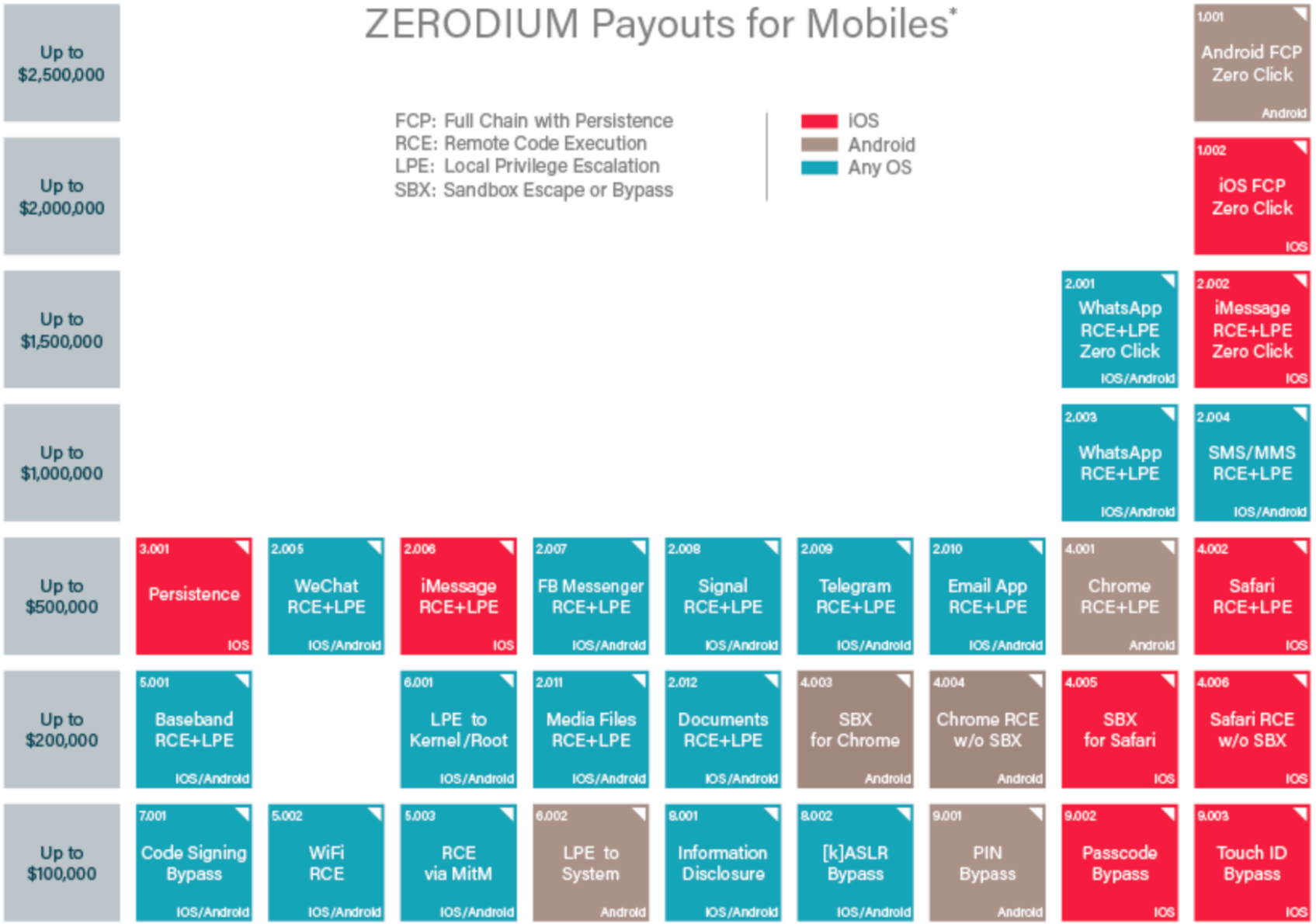
iOS 12.0 → 12.1.2

✔ Jailbreak using Chimera

Semi-Untethered Only. Follow the Cydia Impactor method of installing the Jailbreak. Sileo installed by default (not Cydia).

https://canijailbreak.com/

Cashout for Jailbreaks up to 2M USD



https://zerodium.com/program.html

Testing with Non-Jailbroken iOS device

Why test without jailbreak?

- Client gives you an IPA with Jailbreak detection (and might not be able or is not willing to provide a build without it)
- “Easy” way of testing as no jailbreak detection need to be bypassed
- We normally test apps that are coming directly from the developer, therefore no need to get rid of Fairplay encryption and it’s easy
- No need to have and maintain a jailbroken device
- Clients are sometimes not satisfied when testing on a jailbroken phone (Is this really a risk?)

DISCLAIMER:
It’s only “easy” if no other frameworks or detection mechanisms are in place that would otherwise detect static manipulation or dynamic instrumentation of the app.

Testing with Non-Jailbroken iOS device

Objection (https://github.com/sensepost/objection)

objection is a runtime mobile exploration toolkit, powered by Frida. It was built with the aim of helping assess mobile applications and their security posture without the need for a jailbroken or rooted mobile device.

Thanks to Leon Jacobs (@leonjza) who created **objection!**

Testing with Non-Jailbroken iOS device

How to test? Repackage the IPA!

1. Get a Apple Developer account and add it to your Xcode
2. Create a dummy project in Xcode and install the app on your iOS device (this is what I did when installing the test app on your iOS device)
3. Install dependencies for patching process
4. Install objection
5. Execute **objection patchipa -s test.ipa --code-signature <signature>**

Details:

Automated approach for repackaging Preparations for patching IPA (objection) and
Manual approach for repackaging Dynamic Analysis on non-jailbroken devices (MSTG)

DISCLAIMER:
Repackaging can only be done on macOS as Xcode is needed!

Testing with Non-Jailbroken iOS device

Patch the IPA - Only working on macOS!

Find your signing identities in Xcode:

```
➜  ~ security find-identity -v
  1) 4AF7F93E76[redacted] "Apple Development: Sven Schleier (SRGXYECTF2)"
     1 valid identities found
```

Testing with Non-Jailbroken iOS device

Patch the IPA - Only working on macOS!

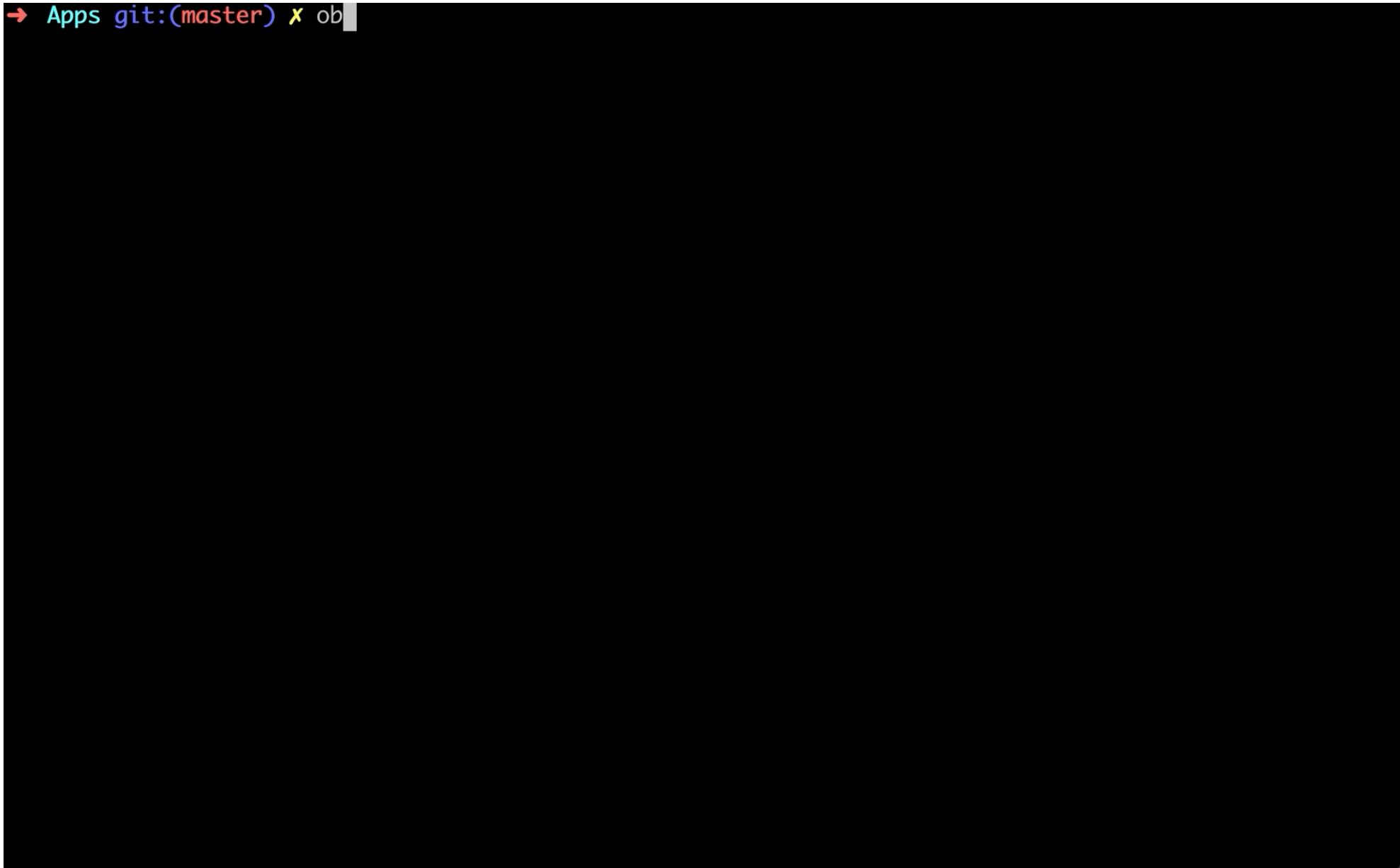
When executing the command “**objection patchipa**”, objection is:

- Unzipping the IPA
- Downloading the latest Frida-Gadget (we will come to this in the next module)
- Looking for a provisioning profile from Xcode
- Signing and patching the IPA

Source: https://github.com/sensepost/objection/wiki/Patching-iOS-Applications

Testing with Non-Jailbroken iOS device

Patch the IPA



Testing with Non-Jailbroken iOS device

If you haven’t done it yet:
Uninstall the MSTG-JWT app from your iOS device that we installed earlier!

Testing with Non-Jailbroken iOS device

Test it by yourself now! Install a repackaged app with the Frida Gadget on your jailbroken / non-jailbroken iOS device

Let’s install and run the app MSTG-JWT-frida.ipa.

Open a terminal and navigate to the directory where you unzipped iOS-preparation-pack.zip and go into the directory “iOS-preparation-pack/Apps”

```
$ cd iOS-preparation-pack/Apps
$ unzip MSTG-JWT-frida.ipa
$ ios-deploy --bundle 'Payload/MSTG-JWT.app' -W -v -d
```

If you see a blank white screen on your iOS device, you did everything right!

See also: Running Patched iOS Applications

Testing with Non-Jailbroken iOS device

With ios-deploy we can install and debug iPhone apps from the command line, without using Xcode - https://github.com/haxelime/ios-deploy

```
➜  Apps git:(master) ✗ ios-deploy --bundle 'Payload/MSTG-JWT.app' -W -v -d

[....] Waiting for iOS device to be connected
Handling device type: 1
Already found device? 0
Hardware Model: N56AP
Device Name: Sven's iPhone
Model Name: iPhone 6 Plus
SDK Name: iphoneos
```

```
(lldb)     command script add -s asynchronous -f fruitstrap_316f01bd160932d2bf2f95f1f142bc29b1c62dbc.safequit_command safequit
(lldb)     connect
(lldb)     run
success
2020-02-07 05:03:50.268566+0800 MSTG-JWT[17287:6379085] Frida: Listening on 127.0.0.1 TCP port 27042
```

After running ios-deploy you will see in one of the last lines “Frida Listening”

Testing with Non-Jailbroken iOS device

Open another terminal and start objection:

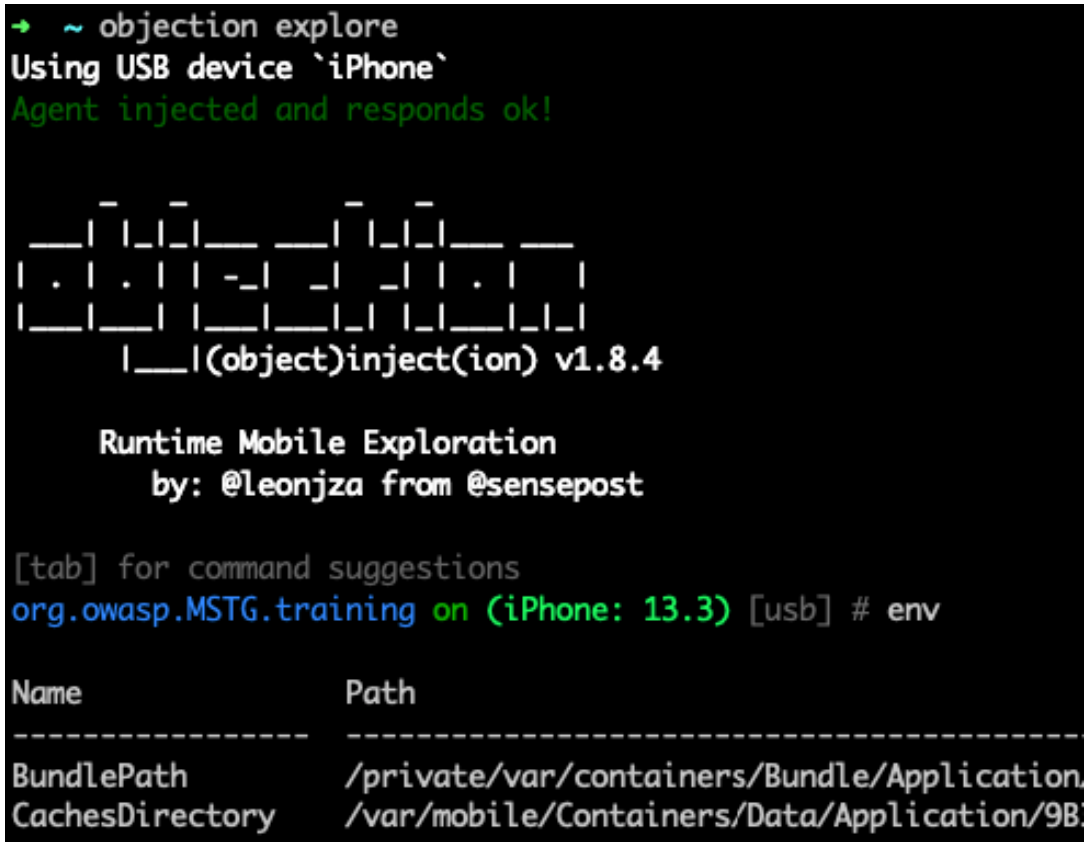
```
$ objection explore
```

Execute the **env** command that prints the environment variables:

```
$ env
```

Please type exit for now into the objection console!

See also: [Running Patched iOS Applications]



Congratulations, your setup is working!
You can see the bundle name in blue and the iOS version in green. This proofs that objection can connect to your iOS device

FRIDA

Frida

Frida Kahlo

FRIDA

OVERVIEW DOCS NEWS CODE CONTACT

Dynamic instrumentation toolkit for developers, reverse-engineers, and security researchers.

https://www.frida.re/

Frida

Dynamic instru… WHAT?

- Dynamic Binary Instrumentation (DBI)
- Modify and analyze a process / binary at runtime
- Do stuff you can do with a debugger without a debugger
- Script it!

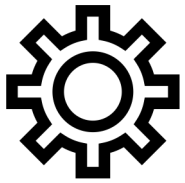
https://github.com/pspace/bsidesmuc/blob/master/Fun_With_Frida_Public.pdf

Frida

Frida is an open source runtime instrumentation framework that lets you inject JavaScript code or portions of your own library into processes on Windows, Linux, macOS and also native Android and iOS apps.

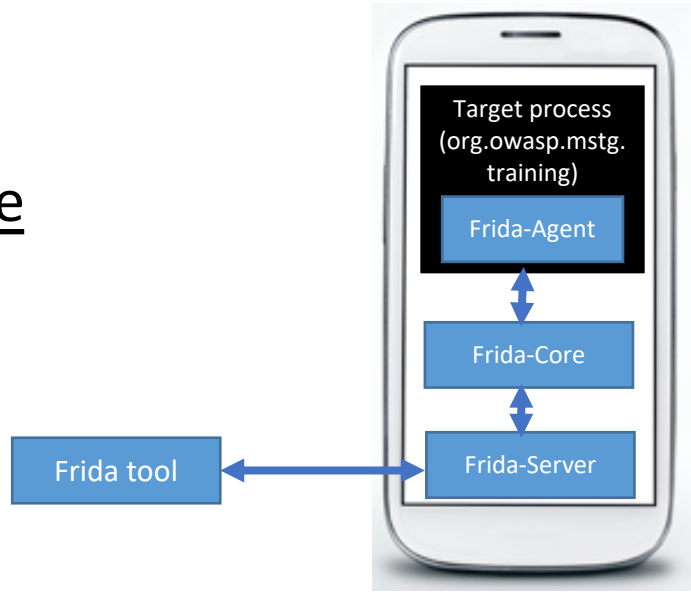
Frida injects a complete JavaScript runtime (Duktape or Google V8) into the process, along with a powerful API that provides a lot of useful functionality.

Frida - Different Modes of Operation



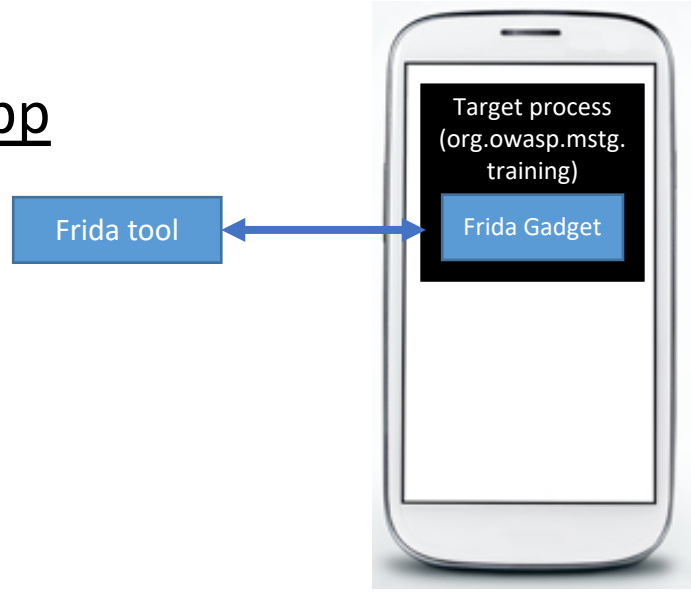
Injected into a process by running the Frida server on the device

- Only possible on jailbroken devices
- Frida handles the injection



Embedded as shared library (frida-gadget.so) into the mobile app

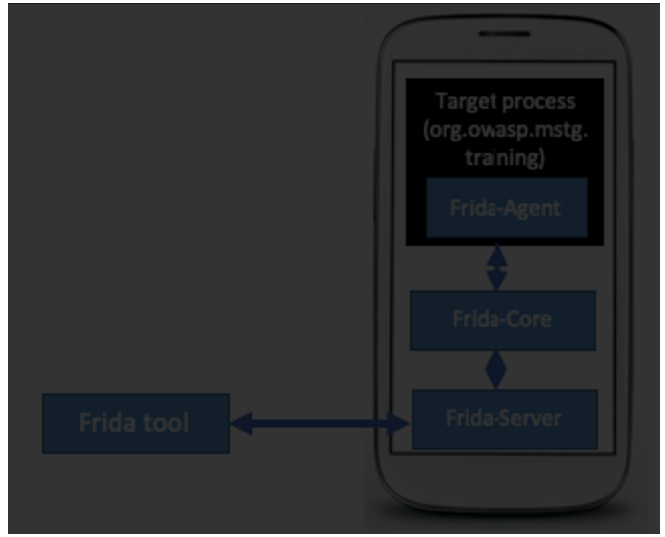
- Working on non-jailbroken devices
- Repackaging and resigning required



Frida - Different Modes of Operation

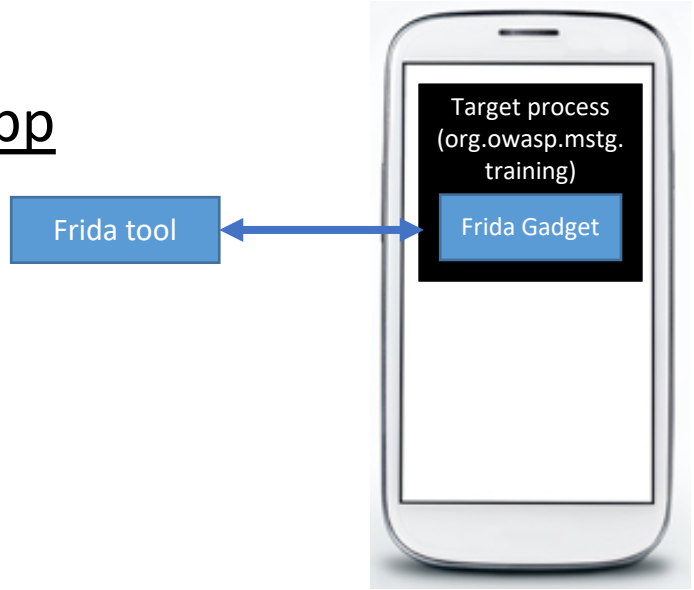
Injected into a process by running the Frida server on the device

- Only possible on jailbroken devices
- Frida handles the injection



Embedded as shared library (frida-gadget.so) into the mobile app

- Working on non-jailbroken devices
- Repackaging and resigning required



This is what we have done just now!

Frida - Basics

Frida Basics

Hook
- Similar to setting up a Software breakpoint on a function to inspect, instrument or manipulate.

Inspect
- Inspecting arguments being passed into the functions.

Instrument
- Injecting additional instructions that helps to debug/RE the application.

Manipulate
- Changing the flow of the function or completely overwriting it.

Frida - Basics

What is an overload?

Method overloading is having similar function names with different arguments. Frida API `.overload()` is commonly used when hooking functions that have more than one implementation.

Common error on Frida cli when the app is obfuscated:

```
{u'columnNumber': 1, u'description': u"Erroverloador: a(): has more than one overload, use .overload(<signature>) to choose from:\n\t.overload('java.lang.String')\n\t. ('int', 'int')",...
```

Frida - Basics

Frida CLI and "Frida scripts":

Two different way to use Frida. Frida CLI is am interactive environment (REPL) similar to Ipython. Frida scripts can be written in JS or wrapped with python and the Frida Python library.

```
→  Frida-Scripts git:(master) ✗ frida -U -n MSTG-JWT -l frida101.js
     ____
    / _  |   Frida 12.8.10 - A world-class dynamic instrumentation toolkit
   | (_| |
    > _  |   Commands:
   /_/ |_|       help      -> Displays the help system
   . . . .       object?   -> Display information about 'object'
   . . . .       exit/quit -> Exit
   . . . .
   . . . .   More info at https://www.frida.re/docs/home/

[iPhone::MSTG-JWT]-> helloWorld()
[iPhone::MSTG-JWT]-> appInfo()
{
    "Binary": "/private/var/containers/Bundle/Application/0F085A81-C97A-4E58-A8AC-9148A9123B50/MSTG-JWT.app/MSTG-JWT",
    "Bundle": "/private/var/containers/Bundle/Application/0F085A81-C97A-4E58-A8AC-9148A9123B50/MSTG-JWT.app",
    "Bundle ID": "org.owasp.MSTG.training",
    "Data": "/private/var/mobile/Containers/Data/Application/9B374977-05F3-4263-93E5-8C46C7E9CEF3",
    "Name": "MSTG-JWT",
    "Version": "1"
}
[iPhone::MSTG-JWT]->
```

Early instrumentation:

Loading Frida before initiating the application, like starting a process with a debugger. Needed to bypass security controls like ant-debugging or root detection that are verified during start-up.

Frida Lab

Let's do our next Lab!

Open the file Lab-iOS.pdf:

```
$ evince ~/MSTG-Handson/Lab/Lab-iOS.pdf
```

Section: Lab – Frida 101 (Frida-Server)

Please type the commands and do not copy & paste!

1. You will remember them easier
2. Some PDF readers do not copy properly and miss whitespaces etc.

Frida – Command Line Tools

Frida can help you to:

- Intercept network traffic, crypto calls, file access etc.
- Pentest and fuzz apps
- Overcome protection mechanisms
- Analyse unknown apps (Malware)
- Solve crackmes/CTFs
- Build your own analysis scripts, frameworks and tools

Frida - Resources

Useful Frida scripts for iOS:

https://codeshare.frida.re/:

- https://codeshare.frida.re/@ay-kay/ios-dataprotection/
- https://codeshare.frida.re/@dki/ios-app-info/
- https://codeshare.frida.re/@oleavr/ios-list-apps/
- https://codeshare.frida.re/@dki/find-ios-app-by-display-name/
- https://codeshare.frida.re/@andydavies/ios-tls-keylogger/
- https://codeshare.frida.re/@ChiChou/objective-c-class-hierarchy-dumper/
- https://codeshare.frida.re/@maltek/generate-c-header-for-swift-data-types/

https://github.com/dweinstein/awesome-frida
https://github.com/iddoeldor/frida-snippets
https://github.com/interference-security/frida-scripts/tree/master/iOS

These scripts are great to get an understanding of Frida and to use it as a starting point if you have a specific use case you want to implement.

Frida – Resources

Further reading to understand Frida better:

- Great introduction into Frida: https://github.com/pspace/bsidesmuc
- Detailed slide deck about Frida and how to use in mobile app testing: https://drive.google.com/file/d/1JccmMLi6YTnyRrp_rk6vzKrUX3oXK_Yw/view
- Read Code and go through existing Frida scripts (don't reinvent the wheel, leverage on what is out there 🤓): https://codeshare.frida.re
- Official Documentation: https://www.frida.re/docs/home/

iOS Testing Local Storage

iOS – Sensitive Data in Local Storage

Conventional wisdom suggests that as little sensitive data as possible should be stored on permanent local storage. In most practical scenarios, however, some type of user data must be stored, e.g.:

- Session Information (Cookies etc.)
- Access Tokens (JWT)
- Encryption keys
- Sensitive data for offline usage
- etc.

iOS – Sensitive Data in Local Storage

Disclosing sensitive information has several consequences. In general, an attacker (malware) may identify this information and use it for additional attacks, such as:

- social engineering (if PII has been disclosed) or
- account hijacking (if session information or an authentication token has been disclosed).

iOS – Sensitive Data in Local Storage

Storing Data is essential for many mobile apps. For example, some apps use data storage to keep track of user settings or user-provided data. Data can be stored persistently in several ways. The following list of storage techniques are widely used on the iOS platform:

- NSUserDefaults (Plist)
- NSData / NSMutableData
- CoreData

iOS – Sensitive Data in Local Storage

Plist (Property List) is a flexible and convenient format for storing application data. Since **plists** are actually XML files, you can use a simple text editor to open them.

NSUserDefaults - An interface to the user's defaults database, where you store key-value pairs persistently across launches of your app.
The defaults system allows an application to customize its behavior to match a user's preferences.

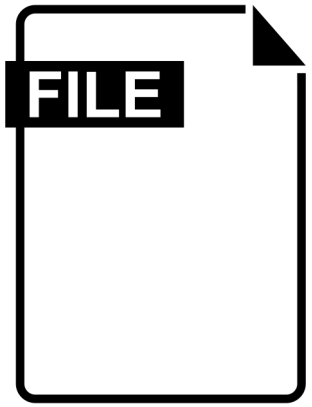
https://developer.apple.com/documentation/foundation/nsuserdefaults

iOS – Sensitive Data in Local Storage

NSData / NSMutableData

Both classes are used to read files and write data to files.

NSData creates static data objects, while NSMutableData creates dynamic data objects.



iOS – Sensitive Data in Local Storage

Core Data

Core Data is a framework, provided by Apple, to be used on Applications for iOS and macOS. As Apple states, it is not a database, but instead a persistence framework that commonly uses SQLite to store and retrieve structured data.

iOS – Sensitive Data in Local Storage

On iOS, system applications can be found in the /Applications directory while user-installed apps are available under /var/containers/Bundle/Application/.

However, finding the right folder just by navigating the file system is not a trivial task as every app gets a random 128-bit UUID (Universal Unique Identifier) assigned during installation for its directory names.

Besides the Frida script we executed earlier, using **ipainstaller** on a jailbroken device is another way that will show you all the directory information of the app.

```
iPhone:~ root# ipainstaller -l
com.highaltitudehacks.DVIAswiftv2
org.owasp.MSTG.training
iPhone:~ root# ipainstaller -i org.owasp.MSTG.training
Identifier: org.owasp.MSTG.training
Version: 1
Short Version: 1.0
Name: MSTG-JWT
Display Name: MSTG-JWT
Bundle: /private/var/containers/Bundle/Application/BE5BDD24-DC9E-450F-8FDB-B17FBD2AAA36
Application: /private/var/containers/Bundle/Application/BE5BDD24-DC9E-450F-8FDB-B17FBD2AAA36/MSTG-JWT.app
Data: /private/var/mobile/Containers/Data/Application/5850440D-C2ED-4B9C-ACF2-3ED0A23B7C8F
```

iOS – Sensitive Data in Local Storage

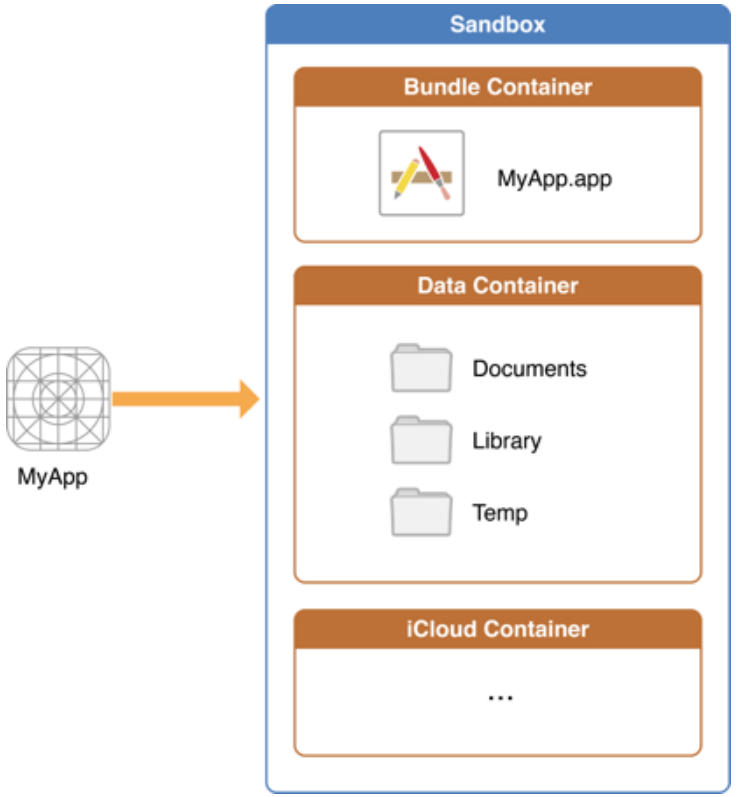
Where can I find files of an app in Local Storage?

/private/var/containers/Bundle/Application/[**UUID**]/[Application].app
Bundle directory includes all the static resources of the app

/private/var/containers/Bundle/Application/[**UUID**]/[Application].app/App
Path of the binary (executable)

/private/var/containers/Bundle/Application/[**UUID**]/[Application]/Info.plist
App metadata: configuration of the app (icon to display, supported document types, etc.)

UUID (Universally Unique Identifier): random 36 alphanumeric characters string unique to the app




```
org.owasp.MSTG.training on (iPhone: 13.3) [usb] # env

Name              Path
----------------  ------------------------------------------------------------------------------------
BundlePath        /private/var/containers/Bundle/Application/0F085A81-C97A-4E58-A8AC-9148A9123B50/MSTG-JWT.app
CachesDirectory   /var/mobile/Containers/Data/Application/9B374977-05F3-4263-93E5-8C46C7E9CEF3/Library/Caches
DocumentDirectory /var/mobile/Containers/Data/Application/9B374977-05F3-4263-93E5-8C46C7E9CEF3/Documents
LibraryDirectory  /var/mobile/Containers/Data/Application/9B374977-05F3-4263-93E5-8C46C7E9CEF3/Library
```

See also http://bit.ly/2OAJkzf

iOS – Sensitive Data in Local Storage

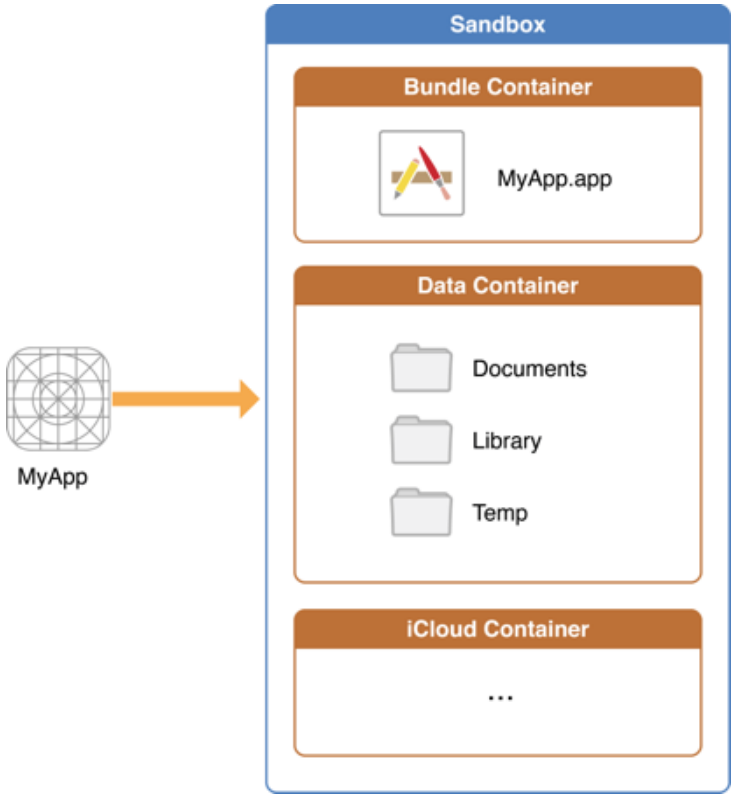
Where can I find files of an app in Local Storage?

/var/mobile/Containers/Data/Application/[**Data-UUID**]/Documents
Contains all the user-generated data.

/var/mobile/Containers/Data/Application/[**Data-UUID**]/Library
Contains all files that aren't user-specific, such as caches, preferences, cookies, and property list (plist) configuration files.

/var/mobile/Containers/Data/Application/[**Data-UUID**]/tmp
Contains temporary files which aren't needed between application launches.

Data-UUID: random 36 alphanumeric characters string unique to the app



See also http://bit.ly/2OAJkzf

```
org.owasp.MSTG.training on (iPhone: 13.3) [usb] # env

Name                Path
----------------    ------------------------------------------------------------------------------------------
BundlePath          /private/var/containers/Bundle/Application/0F085A81-C97A-4E58-A8AC-9148A9123B50/MSTG-JWT.app
CachesDirectory     /var/mobile/Containers/Data/Application/9B374977-05F3-4263-93E5-8C46C7E9CEF3/Library/Caches
DocumentDirectory   /var/mobile/Containers/Data/Application/9B374977-05F3-4263-93E5-8C46C7E9CEF3/Documents
LibraryDirectory    /var/mobile/Containers/Data/Application/9B374977-05F3-4263-93E5-8C46C7E9CEF3/Library
```

iOS – Sensitive Data in Local Storage

Let's do our next Lab!

Open the file Lab-iOS.pdf:

```
$ evince ~/MSTG-Handson/Lab/Lab-iOS.pdf
```

Section: Lab – Local Storage, testing for Sensitive Data

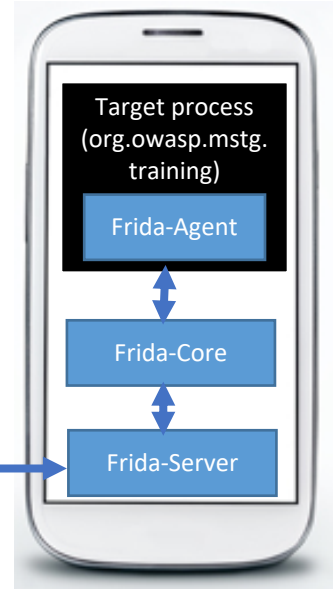
Please type the commands and do not copy & paste!

1. You will remember them easier
2. Some PDF readers do not copy properly and miss whitespaces etc.

Frida - Gadget

Injected into a process by running the Frida server on the device

- Only possible on jailbroken devices
- Frida handles the injection



Embedded as shared library (frida-gadget.so) into the mobile app

- Working on non-jailbroken devices
- Repackaging and resigning required

Repackage the app with the Frida Gadget: https://www.frida.re/docs/ios/#without-jailbreak

Frida - Gadget

Things to note when using the Frida Gadget and objection on a non jailbroken device:

- We are not bypassing any security controls in iOS!
- We are still in the sandbox!
- Through the usage of the Friday Gadget, objection can do whatever the app can do!

iOS – Sensitive Data in Local Storage

How can we do it better?

Keychain

The iOS Keychain can be used to securely store short, sensitive bits of data, such as encryption keys and session tokens. It is implemented as an SQLite database that can be accessed through the Keychain APIs only.

Can we access the Keychain with objection?

iOS – Sensitive Data in Local Storage

Of course we can! We can dump the Keychain, but we will only have access to the Keychain items the app has access to (entitlements)!

Type this in your objection console:

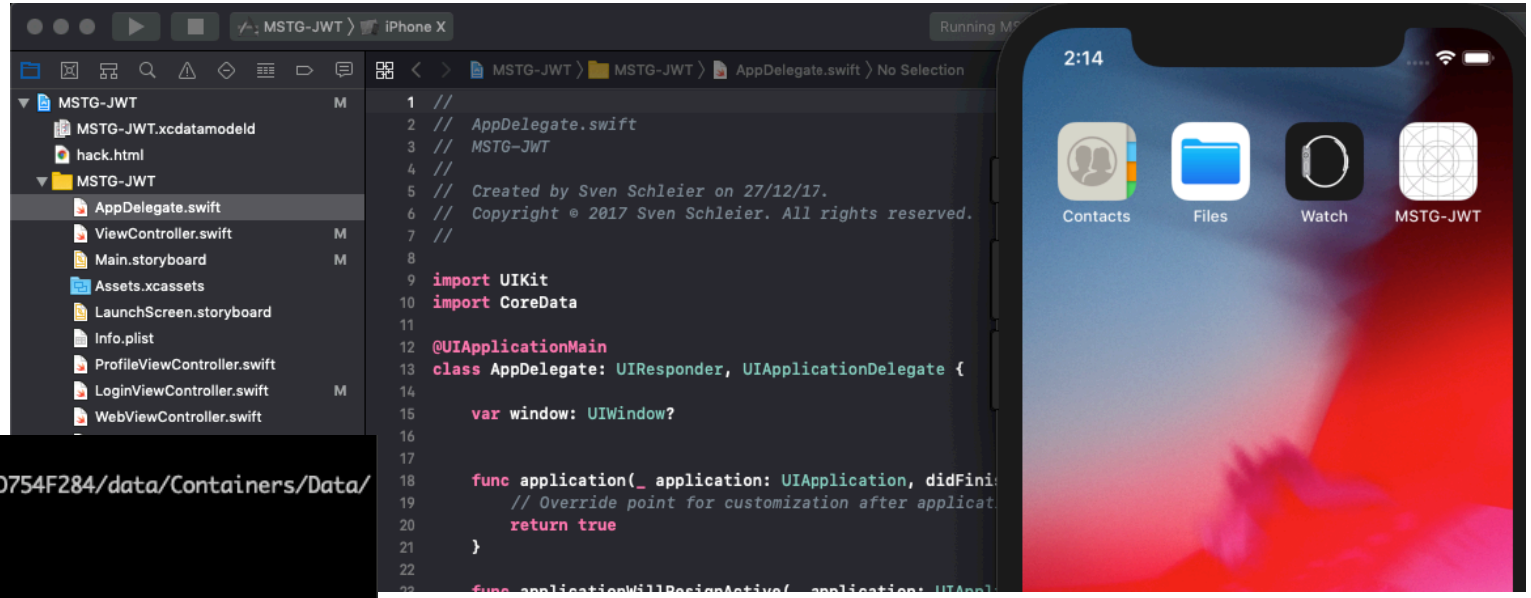
```
org.owasp.MSTG.training on (iPhone: 12.4) [usb] # ios keychain dump
Note: You may be asked to authenticate using the devices passcode or TouchID
Save the output by adding `--json keychain.json` to this command
Dumping the iOS keychain...
Created                  Accessible    ACL    Type      Account              Service    Data
-----------------------  ------------  -----  --------  -----------------    ---------  --------------------
2019-10-16 22:44:29 +0000  WhenUnlocked  None   Password  MSTG-JWT Passcode              masterAccessCode123
```

https://github.com/sensepost/objection/wiki/Notes-About-The-Keychain-Dumper

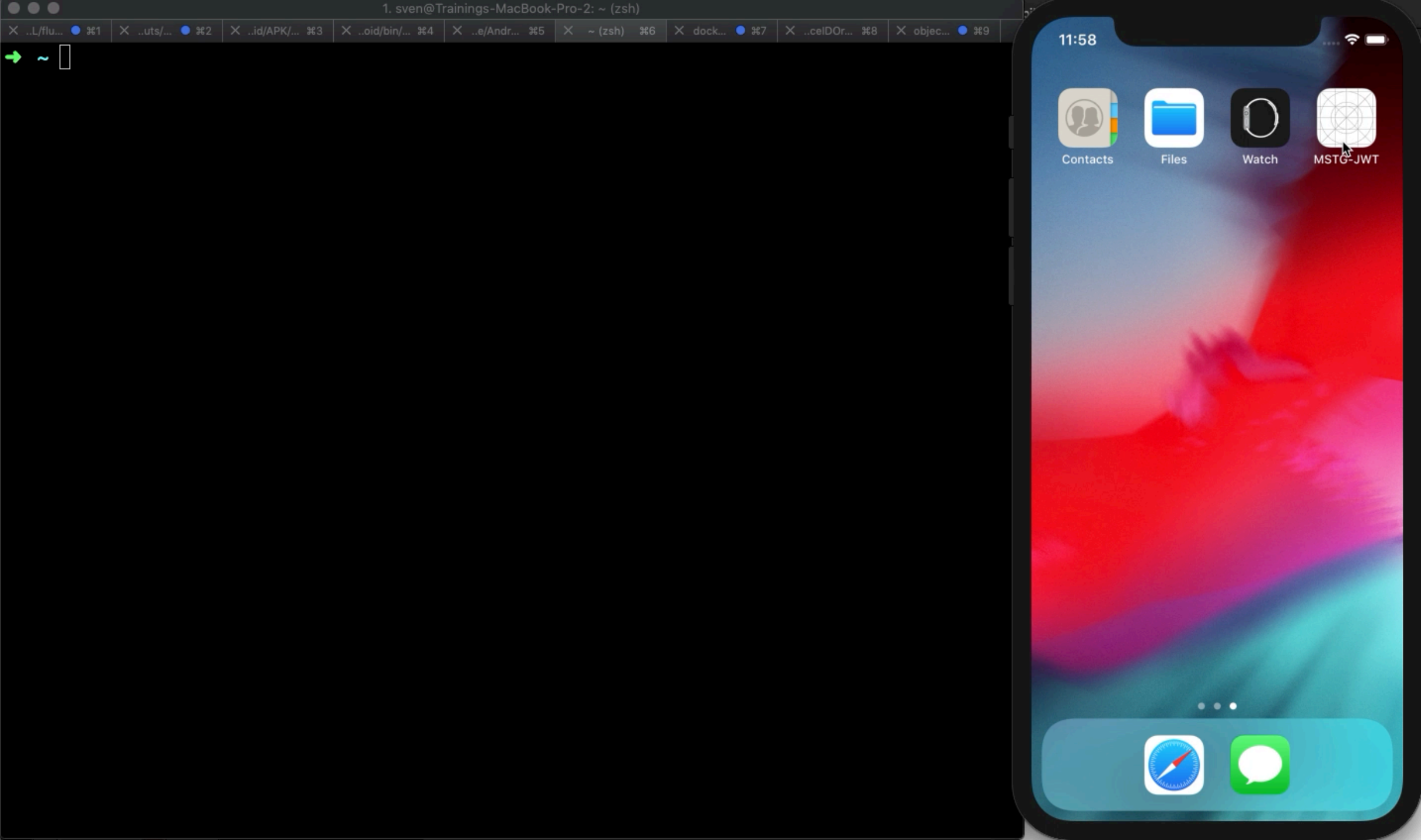
iOS – Sensitive Data in Local Storage

What about iOS devs? They don't usually have a jailbroken phone and Frida doesn't seem to fit for them. There should be a more easy way, right?

Use the tools you already have:
Xcode and iOS Simulator
You have full access to the file system of the iOS Simulator




```
➜ Documents pwd
/Users/sven/Library/Developer/CoreSimulator/Devices/B13C39D7-F7F8-45D9-AB82-2A35D754F284/data/Containers/Data/
Application/8CE68F72-608B-44FB-AF8D-C31E13C2B406/Documents
➜ Documents ls
JWT.plist
➜ Documents cat JWT.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>token</key>
        <string>eyJhbGciOiJIUzI1NiJ9.eyJ1c2VyX2lkIjoxLCJlbWFpbCI6ImZvbyIsImV4cCI6MTU2ODI2MDc0MH0.288NMb4v5BFXA
i69apmVHTyVjLCHHXG8Y2PBt2K1Jpg</string>
</dict>
</plist>
➜ Documents
```


1. sven@Trainings-MacBook-Pro-2: ~ (zsh)
~ (zsh) ⌘6
11:58
Contacts
Files
Watch
MSTG-JWT

iOS – Local Data Storage

Execute the following command in terminal, which will bring you to the file system of the latest iOS simulator started

```
➜ cd $(ls -dlht ~/Library/Developer/CoreSimulator/Devices/*/ | \
head -n 1 |awk '{print $9}')/data/Containers/Data/Application
```

Grep for your app name or keyword, this will show you the UUID of the Application

```
➜ ~ cd $(ls -dlht ~/Library/Developer/CoreSimulator/Devices/*/ | \
head -n 1 |awk '{print $9}')/data/Containers/Data/Application

➜ Application grep -iRn MSTG .
Binary file ./8CE68F72-608B-44FB-AF8D-C31E13C2B406/.com.apple.mobile_container_manager.metadata.plist matches
Binary file ./8CE68F72-608B-44FB-AF8D-C31E13C2B406/Library/Caches/org.owasp.MSTG.training/Cache.db-wal matches
```

See also:
https://mobile-security.gitbook.io/mobile-security-testing-guide/ios-testing-guide/0x06d-testing-data-storage#dynamic-analysis-with-xcode-and-ios-simulator

Passionfruit

Another tool for iOS assessments:

- Web GUI
- Can download files
- Dump Keychain
- List Objective-C Classes
- etc.

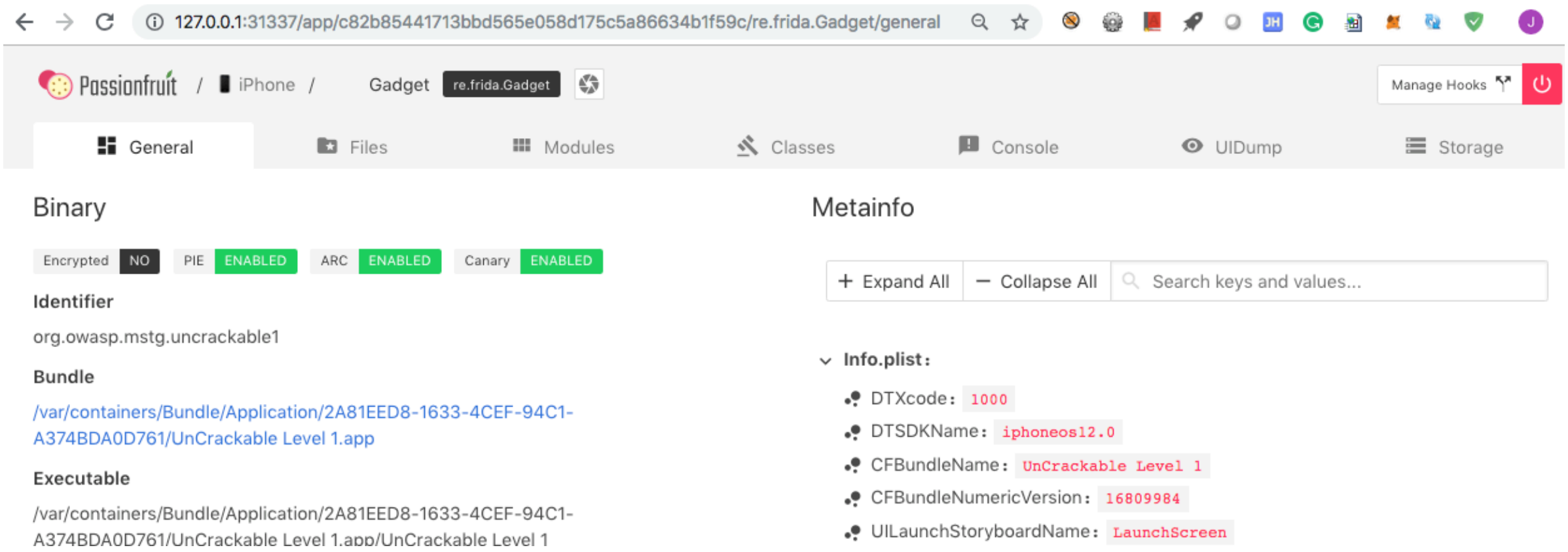
https://github.com/chaitin/passionfruit/issues

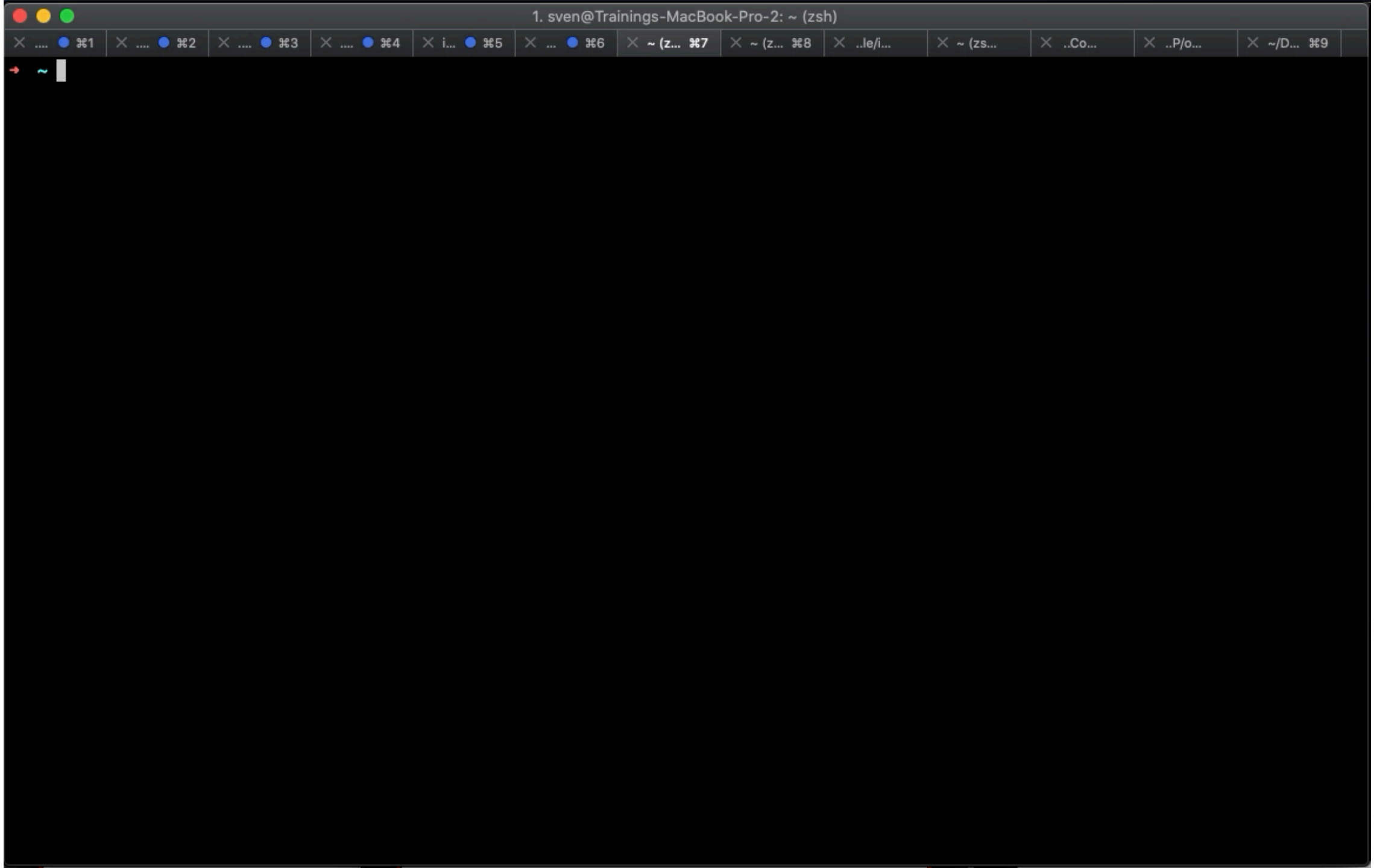
Simple iOS app blackbox assessment tool. Powered by frida.re and vuejs.

TL;DR

```
npm install -g passionfruit
passionfruit
```

Passionfruit



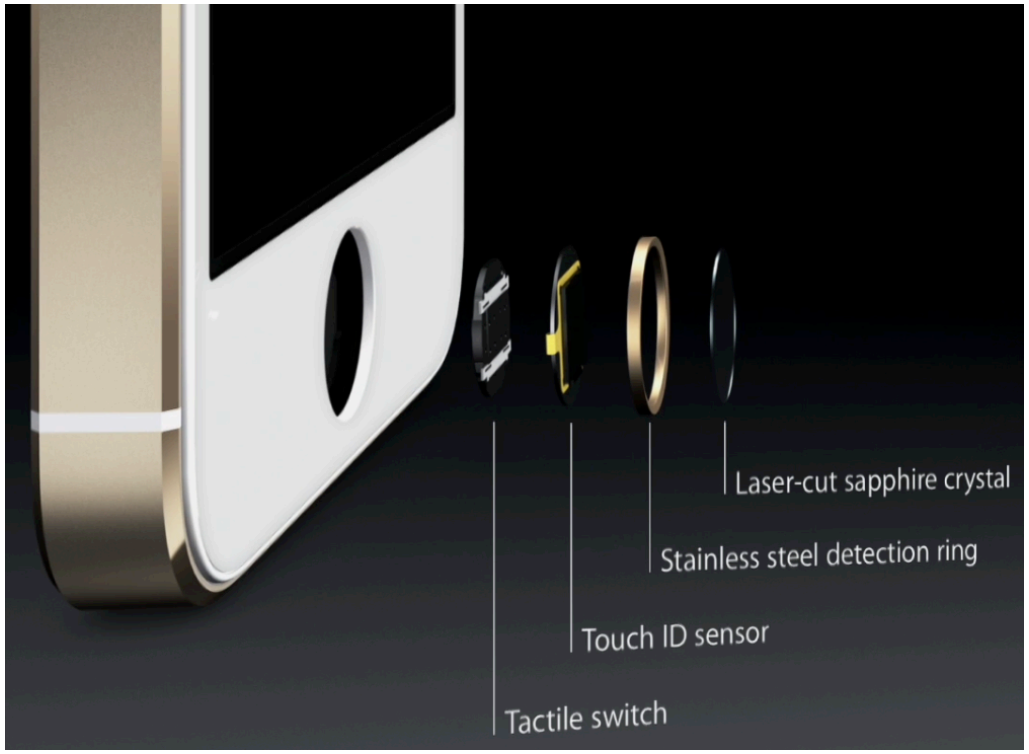

1. sven@Trainings-MacBook-Pro-2: ~ (zsh)
.... ⌘1
.... ⌘2
.... ⌘3
.... ⌘4
i... ⌘5
... ⌘6
~ (z... ⌘7
~ (z... ⌘8
..le/i...
~ (zs...
..Co...
..P/o...
~/D... ⌘9
~

Testing Biometric Authentication (Touch ID / Face ID)

iOS – Touch ID

Touch ID = Fingerprint Authentication

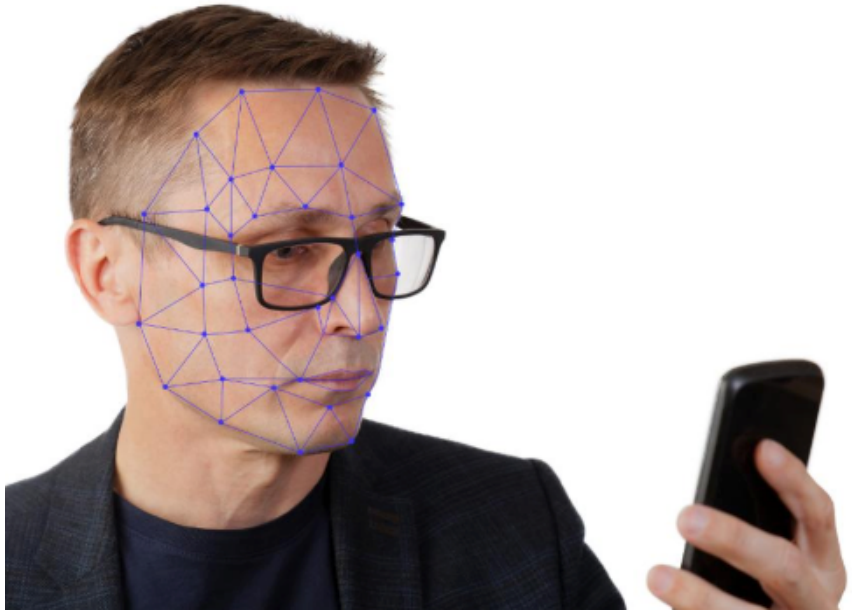
Introduced with the iPhone 5S, the user "unlocks" the iOS device or some inner layer of functionality in an app by providing a valid **fingerprint**.



iOS – Face ID

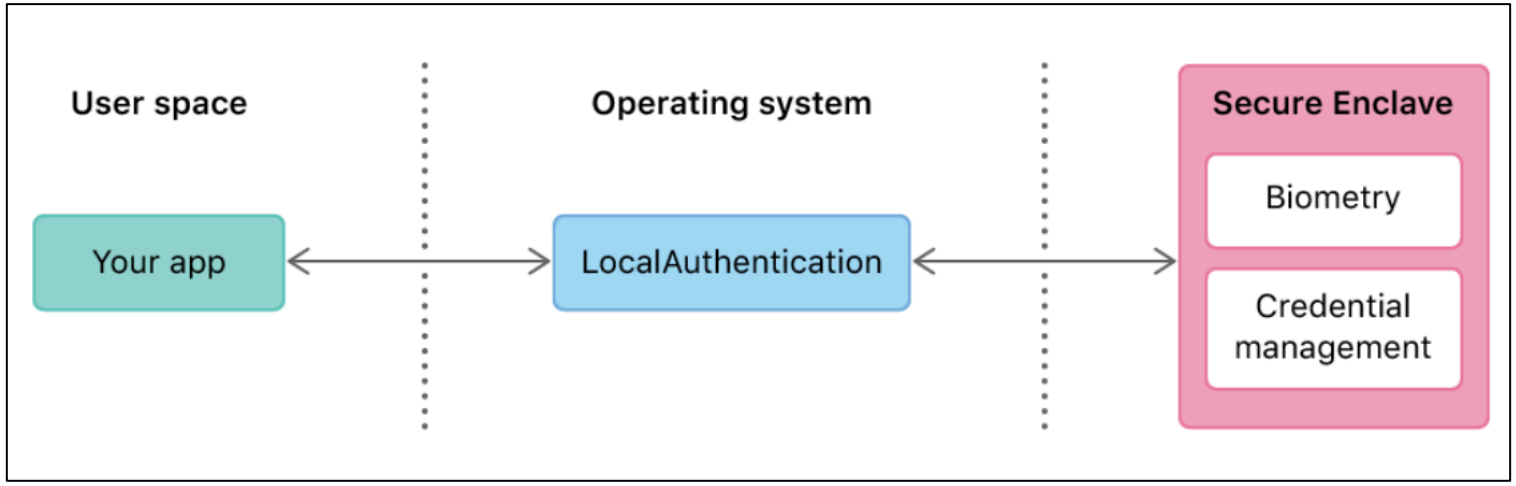
Face ID = Authentication via Facial Recognition

- Face ID works similar, but instead of a fingerprint sensor in the Home button it uses the new **TrueDepth camera** system on the front of iPhone X | XR | XS.
- Face ID accurately maps the **geometry of your face** and matching is performed within the Secure Enclave (SE).
- With the new property **biometryType** you can select either TouchID or FaceID in your app.



iOS – Biometric Authentication

- Fingerprint / facial data is **stored in the Secure Enclave** which is part of the processor of an iOS device (during calibration).
- The provided data (fingerprint / facial data) is sent to the Secure Enclave and compared with the stored data to authenticate the user.
- An iOS app can not confirm the devices passphrase, Touch ID or Face ID directly via code, but it is possible via the LocalAuthentication (LAContext) helper class.



https://developer.apple.com/documentation/localauthentication/logging_a_user_into_your_app_with_face_id_or_touch_id

iOS – Biometric Authentication

How is Face ID / Touch ID used in mobile apps?

- Users can conveniently resume an existing session with a remote service
- Step-up authentication to protect some critical function.

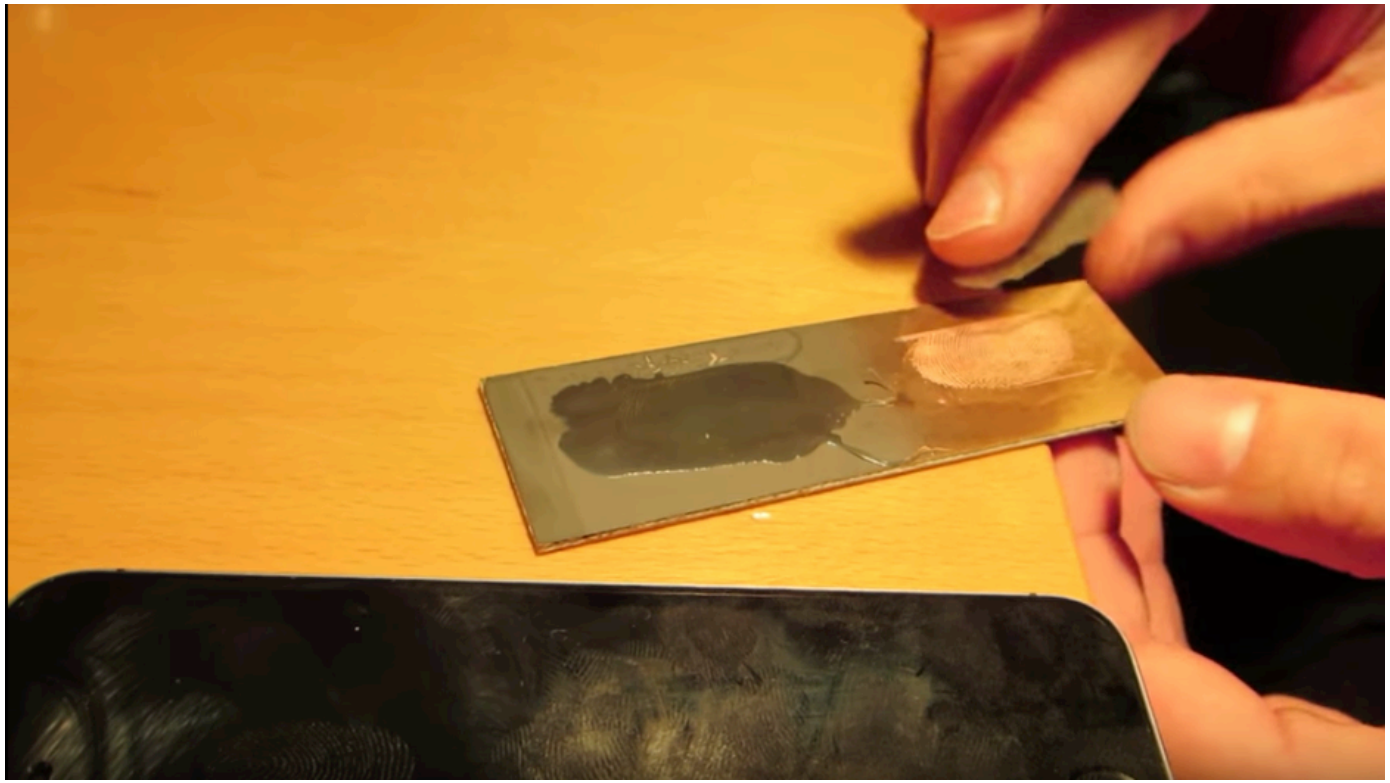
⇒ Passcode must be set when using Touch ID / Face ID (Passcode is used as fall-back when biometric authentication fails)

iOS – Touch ID

Bypassing Touch ID the easy way…

iOS – Touch ID

Bypassing Touch ID the hard way…



https://www.heise.de/video/artikel/iPhone-5s-Touch-ID-hack-in-detail-1966044.html
https://www.ccc.de/en/updates/2013/ccc-breaks-apple-touchid

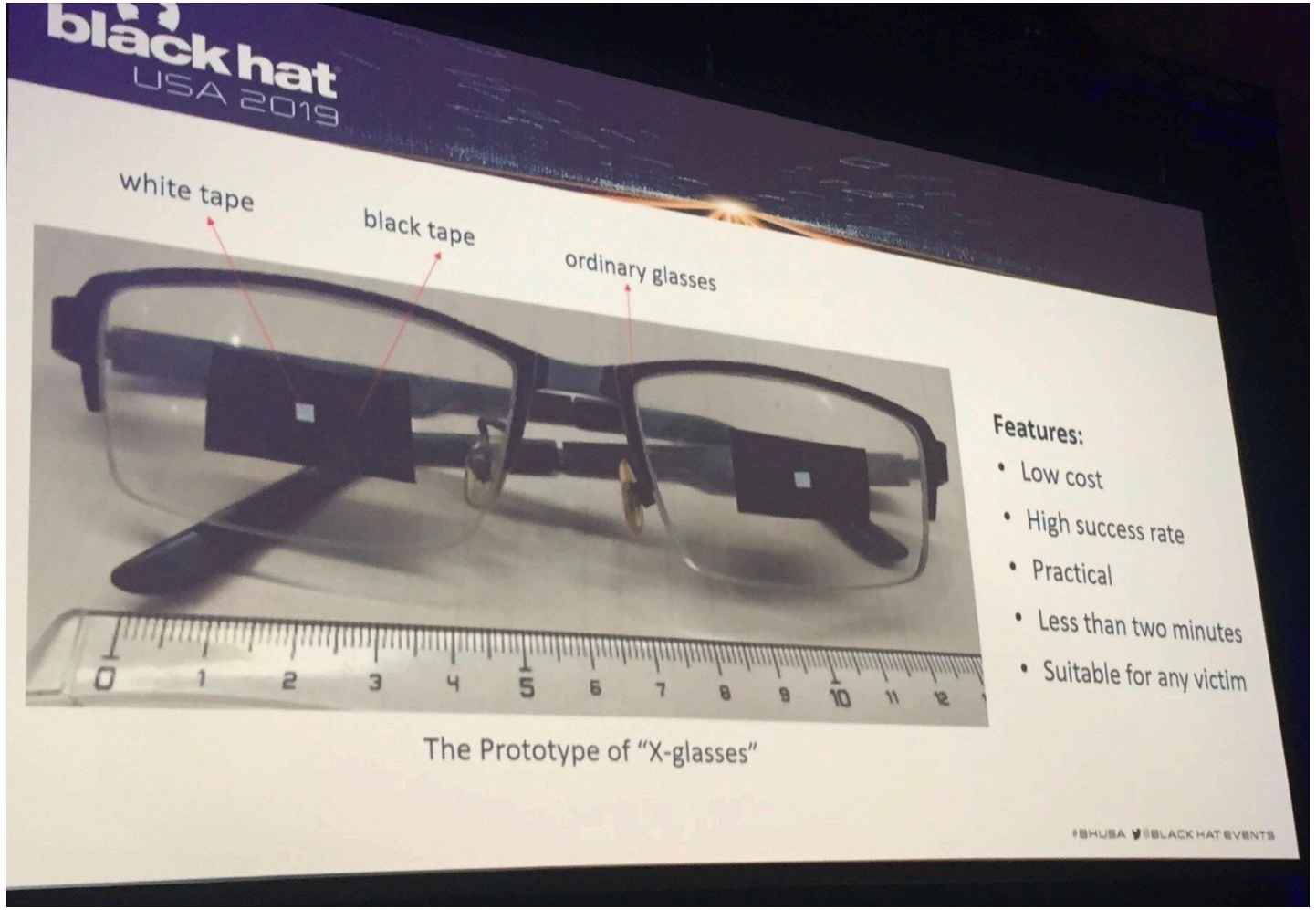
iOS – Face ID

Bypassing Face ID when you are in possession of the phone…

https://news.abs-cbn.com/business/11/14/17/vietnamese-researcher-shows-iphone-x-face-id-hack

iOS – Face ID

Bypassing Face ID when others sleeping…



My 6 year old daughter immediately tried it also and was drawing her X-glasses ;-)

https://9to5mac.com/2019/08/08/face-id-bypass-glasses-tape/

iOS – Biometric Authentication

Is Touch ID or Face ID more secure?

iOS – Biometric Authentication

Is Touch ID or Face ID more secure?

The probability that a random person in the population could unlock your iPhone is 1 in 50,000 with Touch ID or 1 in 1,000,000 with Face ID. This probability increases with multiple enrolled fingerprints (up to 1 in 10,000 with five fingerprints) or appearances (up to 1 in 500,000 with two appearances). For additional protection, both Touch ID and Face ID allow only five unsuccessful

There is always a way to bypass a security control, it's just a matter of how much time and money the attacker is willing to spend and the right opportunity!

Keep this in mind when threat modelling your own app.

iOS – Biometric Authentication

Bypassing Face ID / Touch ID on a jailbroken device

Needle (https://github.com/mwrlabs/needle)

needle

BlackHat Arsenal 2016 Black Hat Arsenal

Created by **MWR InfoSecurity** (@MWRLabs)

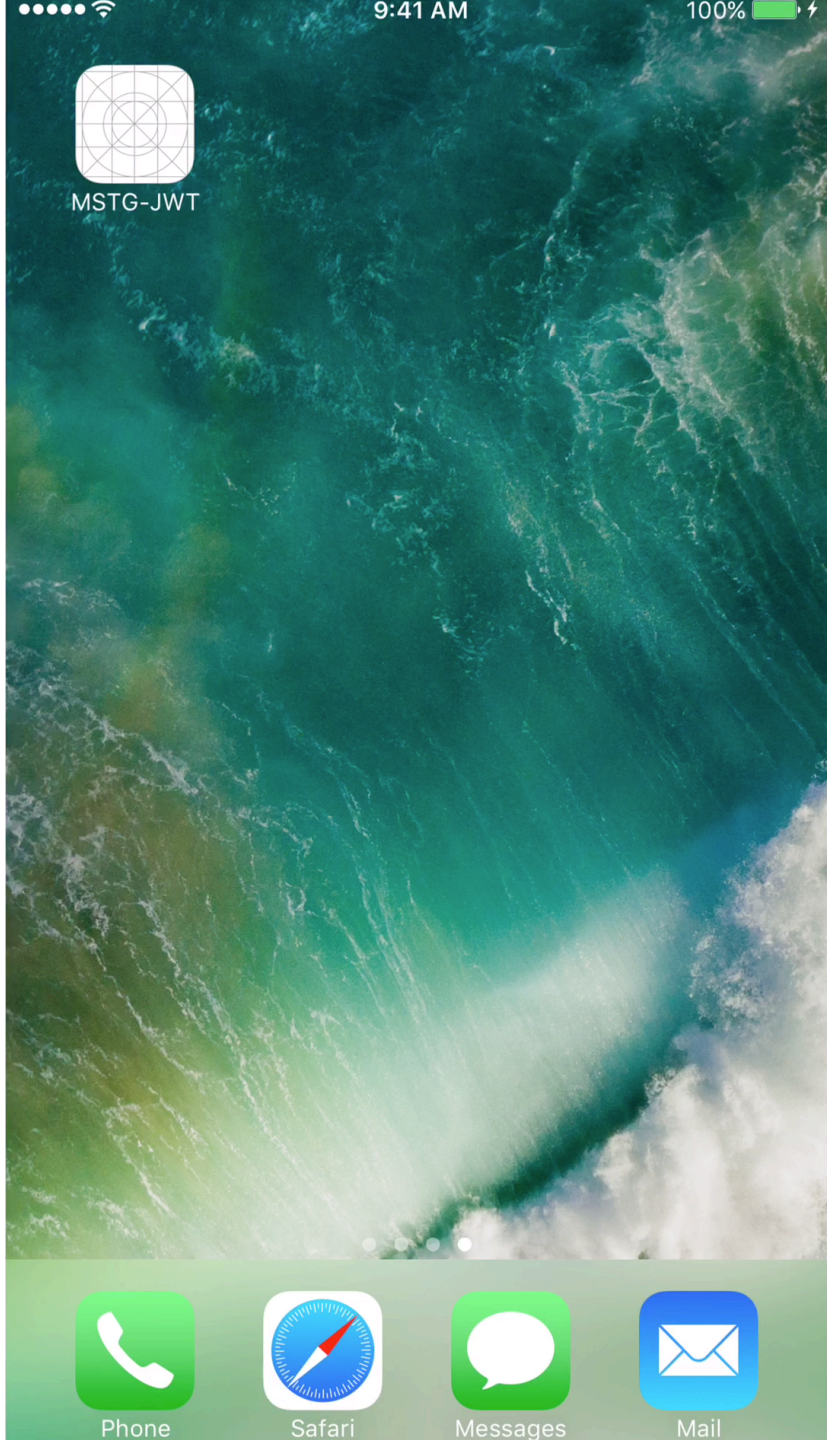
Description

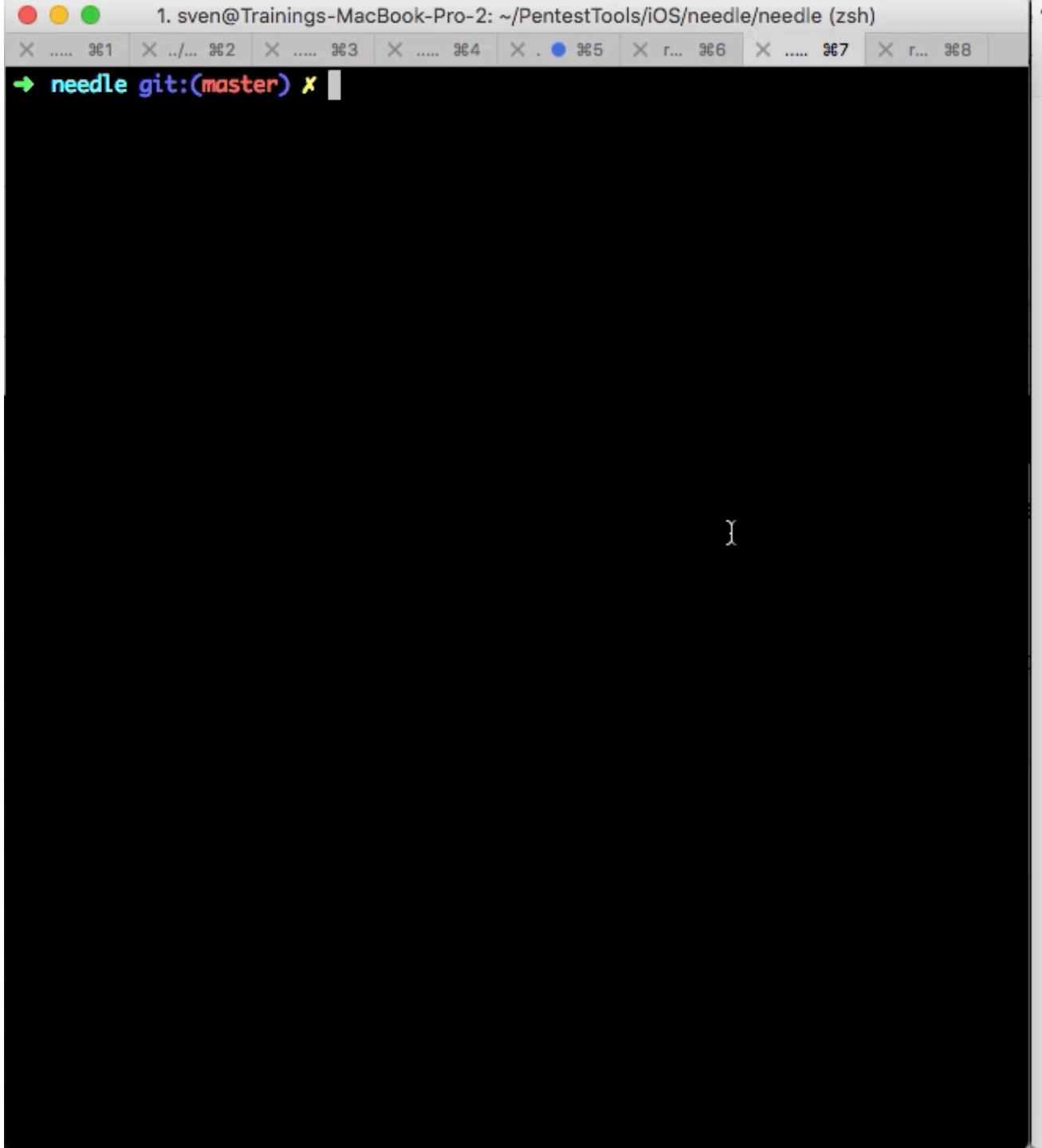
Assessing the security of an iOS application typically requires a plethora of tools, each developed for a specific need and all with different modes of operation and syntax. The Android ecosystem has tools like "drozer" that have solved this problem and aim to be a 'one stop shop' for the majority of use cases, however iOS does not have an equivalent.

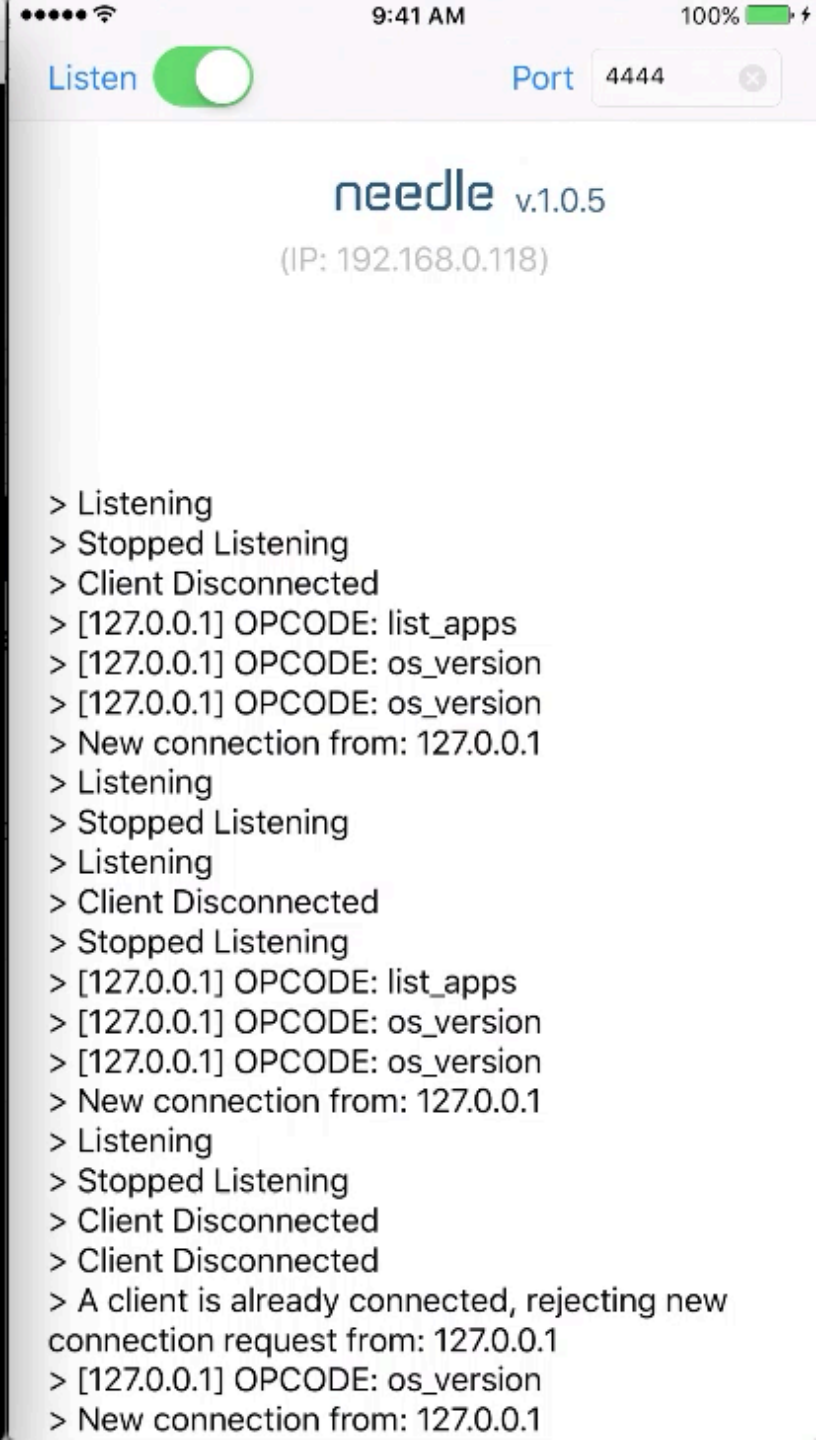
Needle is the MWR's iOS Security Testing Framework, released at Black Hat USA in August 2016. It is an open source modular framework which aims to streamline the entire process of conducting security assessments of iOS applications, and acts as a central point from which to do so. Needle is intended to be useful not only for security professionals, but also for developers looking to secure their code. A few examples of testing areas covered by Needle include: data storage, inter-process communication, network communications, static code analysis, hooking and binary protections. The only requirement in order to run Needle effectively is a jailbroken device.

The release of version 1.0.0 provided a major overhaul of its core and the introduction of a new native agent, written entirely in Objective-C. The new NeedleAgent is an open source iOS app complementary to Needle, that allows to programmatically perform tasks natively on the device, eliminating the need for third party tools.

Usage of Touch ID







Bypassing Touch ID on a jailbroken device with Needle

iOS – Biometric Authentication

Now it's your turn:
Bypass Touch ID / Face ID by yourself in our lab!

Open the file Lab-iOS.pdf:

```
$ evince ~/MSTG-Handson/Lab/Lab-iOS.pdf
```

Section: **Lab – Biometric Authentication**

Please type the commands and do not copy & paste!

1. You will remember them easier
2. Some PDF readers do not copy properly and miss whitespaces etc.

iOS – Biometric Authentication

So that was easy…

Why is it possible to bypass it?

the objection bypass

Within `objection`, when you run the `ios ui biometrics_bypass` command, a hook is executed that listens for invocations of the `-[LAContext evaluatePolicy:localizedReason:reply:]` selector. If the `evaluatePolicy` method is called, the hook will replace the `success` boolean to a `True` in the code block that is executed when a `reply` is received.

The verification of a successful Face ID / Touch ID Authentication when implemented via LAContext is event-based and done within the App and can therefore be manipulated!

https://github.com/sensepost/objection/wiki/Understanding-the-iOS-Biometrics-Bypass

iOS – Biometric Authentication

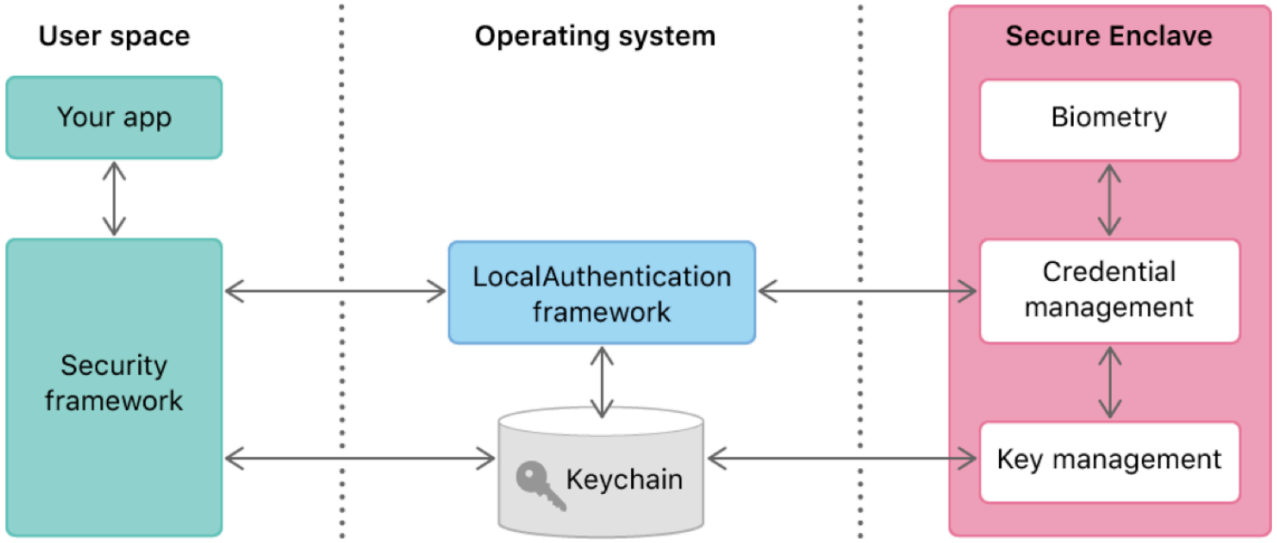
Vulnerable implementation of biometric authentication:

```
37          var authError: NSError?
38          let reasonString = "To access the secure data"
39
40          if localAuthenticationContext.canEvaluatePolicy(.deviceOwnerAuthenticationWithBiometrics, error: &authError) {
41
42              localAuthenticationContext.evaluatePolicy(.deviceOwnerAuthenticationWithBiometrics, localizedReason:
                    reasonString) { success, evaluateError in
43
44                  DispatchQueue.main.async{
45
46                  if success {
47
48                      //TODO: User authenticated successfully, take appropriate action
```

iOS – Biometric Authentication

2 different implementations are available:

- ~~Local Authentication Framework only (LAContext)~~
- LAContext together with KeyChain Services



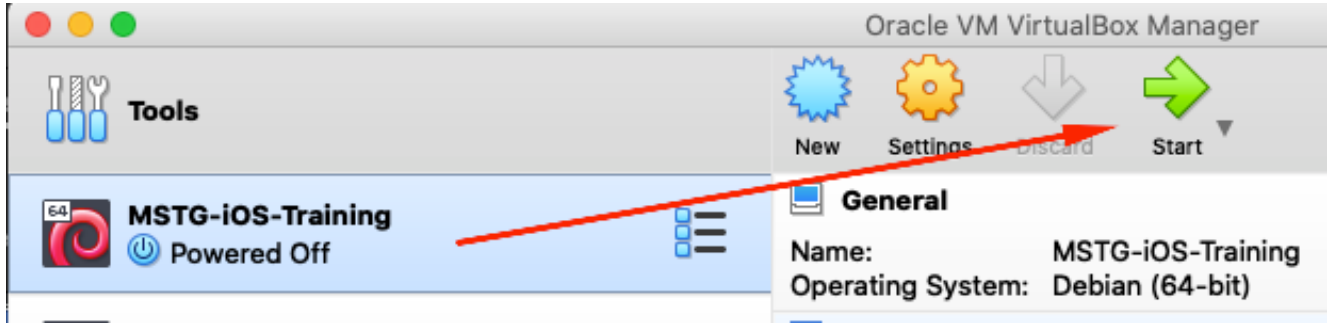
- App stores either a secret authentication token or another piece of secret data identifying the user in the Keychain.
- A valid set of biometrics must be presented before the key is released from the Secure Enclave to decrypt the keychain entry itself.
- This solution cannot be bypassed (even on jailbroken devices), as the verification is done within the Secure Enclave (SE).

See MSTG for sample implementations: http://bit.ly/2qCcIwq
See also https://github.com/sensepost/objection/issues/136 and https://apple.co/2KUscTr

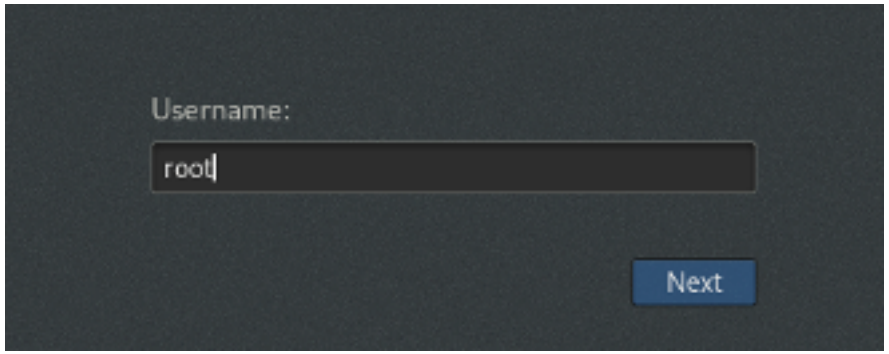
Corellium Setup

Virtual Machine

Start the VM



User: root
Password: toor



Virtual Machine

The following private GitHub repo was already cloned into the Linux VM

https://github.com/sushi2k/MSTG-Training-iOS

This contains all materials needed for the training and the Apps.

Please start a terminal (black icon on the left side) and execute the following command now, to get the latest updates (if none, you will only get “Already up to date”):

```
$ cd ~/MSTG-Handson
$ git pull
```

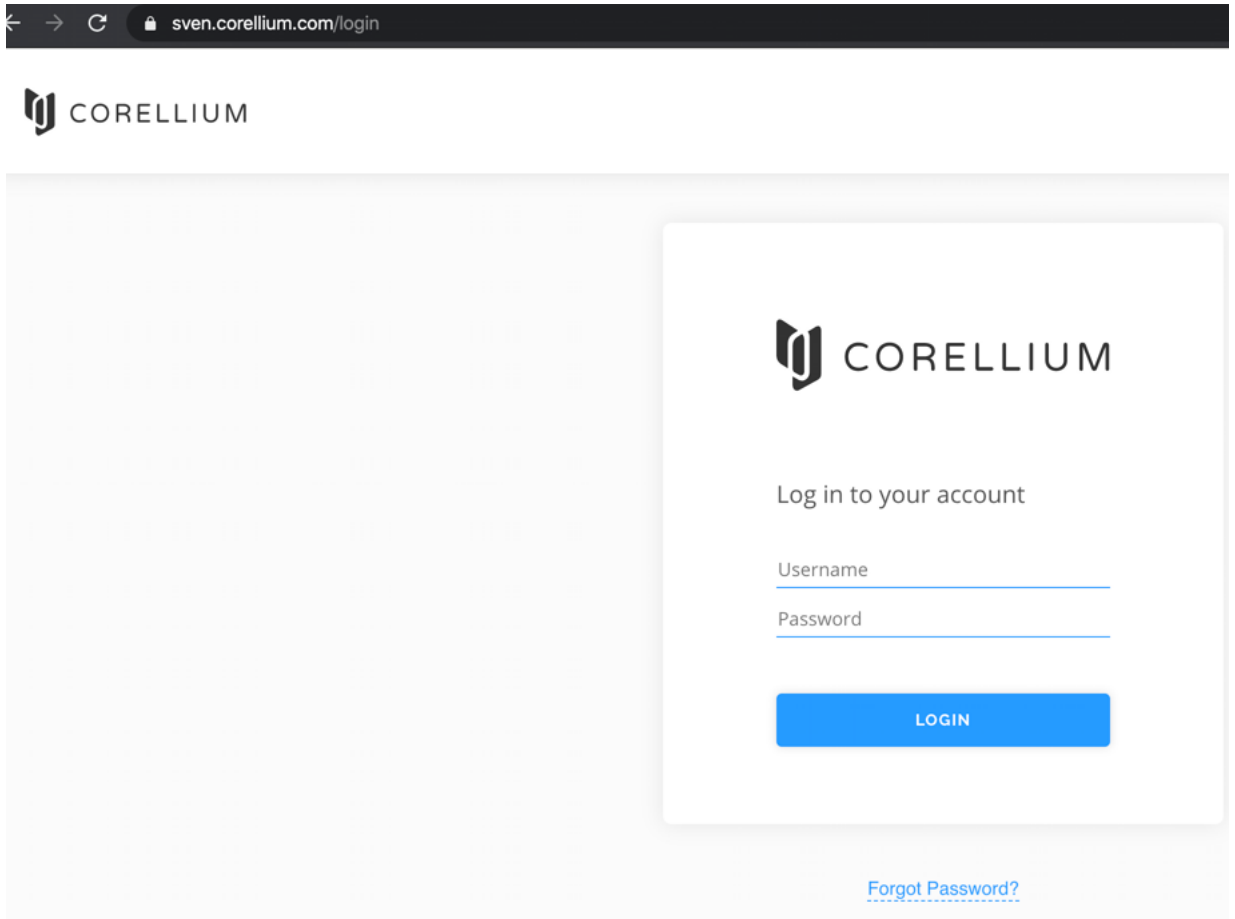
Use Corellium

Everybody is getting one jailbroken iOS instance, please stick to it!

ID	Username	Password
2	2@s7ven.info	OWASPDay2020
3	3@s7ven.info	OWASPDay2020
4	4@s7ven.info	OWASPDay2020
…	…	

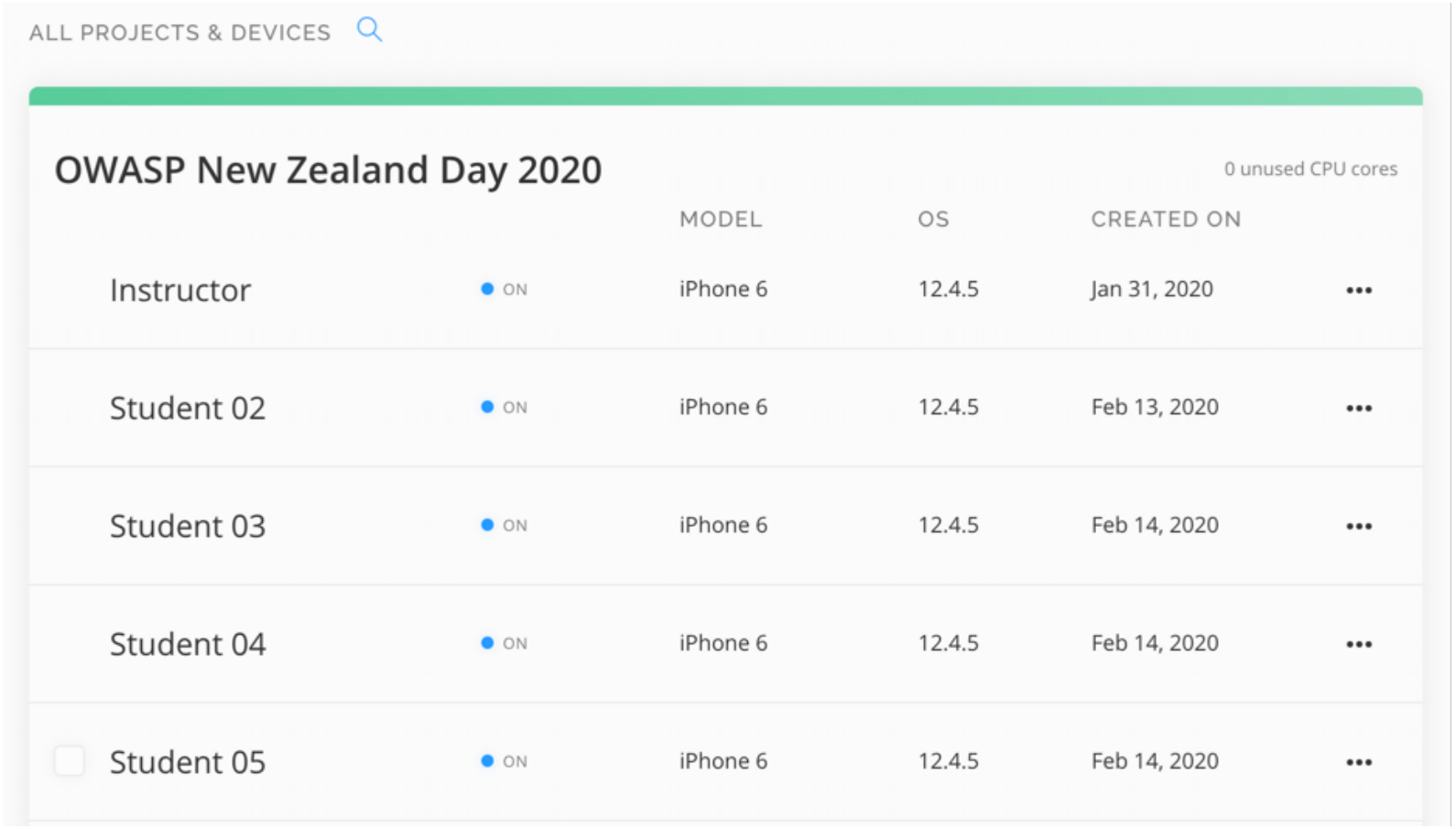
Use Corellium

- Open the URL on your macOS (Safari, Chrome)! It’s not working in the VM
- Go to https://sven.corellium.com
- Login with your credentials



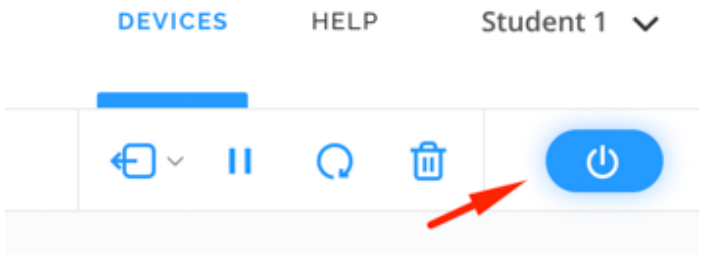
Use Corellium

Select your ID in the device list and stick to it!

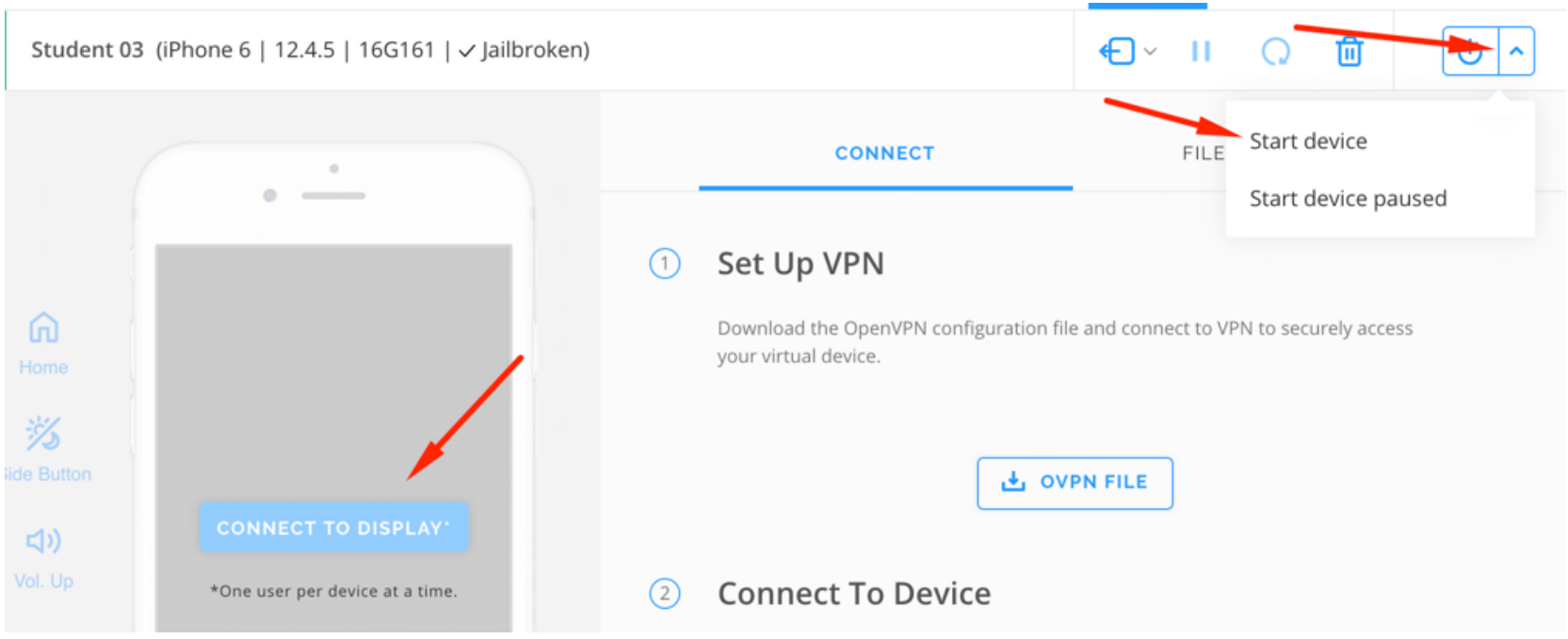


Use Corellium

If it looks like this, your device is already started. Click on “Connect to Display”



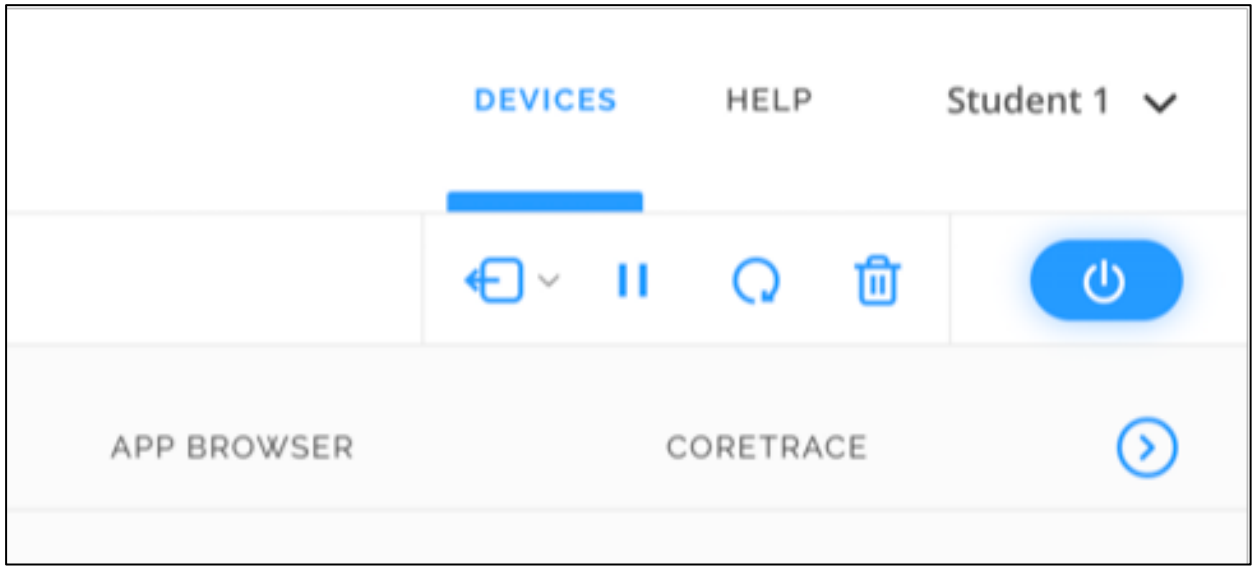
Otherwise, start the device first (top right corner) and then “Connect to Display”



Use Corellium

You have a fully working iOS device with iOS 12.4.5 which is jailbroken.

Use Corellium



You are administrator of all instances!
If you have problems with yours, raise your hand and don’t’ change any settings. This might affect other students.

Use Corellium

Go to the MSTG-Handson folder and execute the command **openvpn** with the ovpn file:

```
→  ~ cd MSTG-Handson
→  MSTG-Handson git:(master) ✗ openvpn Corellium\ VPN\ -\ OWASP\ New\ Zealand\ Day\ 2020.ovpn
Sun Feb 16 14:04:31 2020 OpenVPN 2.4.7 x86_64-pc-linux-gnu [SSL (OpenSSL)] [LZO] [LZ4] [EPOLL]
[PKCS11] [MH/PKTINFO] [AEAD] built on Feb 20 2019
Sun Feb 16 14:04:31 2020 library versions: OpenSSL 1.1.1d  10 Sep 2019, LZO 2.10
Sun Feb 16 14:04:31 2020 TCP/UDP: Preserving recently used remote address: [AF_INET]96.9.221.43
:49764
Sun Feb 16 14:04:31 2020 UDP link local: (not bound)
Sun Feb 16 14:04:31 2020 UDP link remote: [AF_INET]96.9.221.43:49764
Sun Feb 16 14:04:32 2020 [openvpn-e177b738-f735-41d8-a423-c5744cc549ce] Peer Connection Initiat
ed with [AF_INET]96.9.221.43:49764
Sun Feb 16 14:04:33 2020 TUN/TAP device tap0 opened
Sun Feb 16 14:04:33 2020 /sbin/ip link set dev tap0 up mtu 1500
Sun Feb 16 14:04:33 2020 /sbin/ip addr add dev tap0 10.11.3.2/22 broadcast 10.11.3.255
Sun Feb 16 14:04:33 2020 WARNING: this configuration may cache passwords in memory -- use the a
uth-nocache option to prevent this
Sun Feb 16 14:04:33 2020 Initialization Sequence Completed
```

Use Corellium

<u>Find the IP of your device in Corellium!</u>

Open another terminal window and ssh into **your** jailbroken device (this might take sometimes a minute)

The password is **alpine**

```
→  MSTG-Handson git:(master) ✗ ssh root@10.11.1.1
root@10.11.1.1's password:
Last login: Sun Feb 16 02:49:27 2020 from 127.0.0.1
iPhone:~ root# id
uid=0(root) gid=0(wheel) groups=0(wheel),1(daemon),2(kmem)
),9(procmod),20(staff),29(certusers),80(admin)
iPhone:~ root#
```

Executing the command **id** will show you that you are root on the device.

Use Corellium

The IP Address 10.11.0.X is used later for Frida

The IP Address 10.11.1.X is used for SSH

Use Corellium

You can use this instance for the next 2 days!

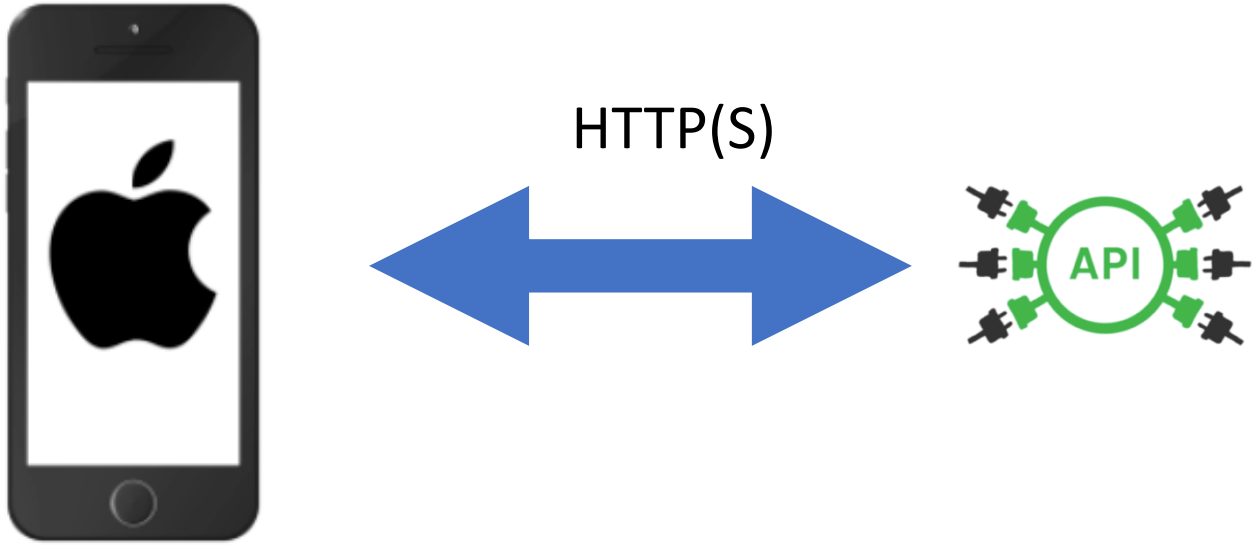
In case you are interested in Corellium, please reach out with your business mail address to:

Steve Dyer - steve@corellium.com

Intercepting App Communication

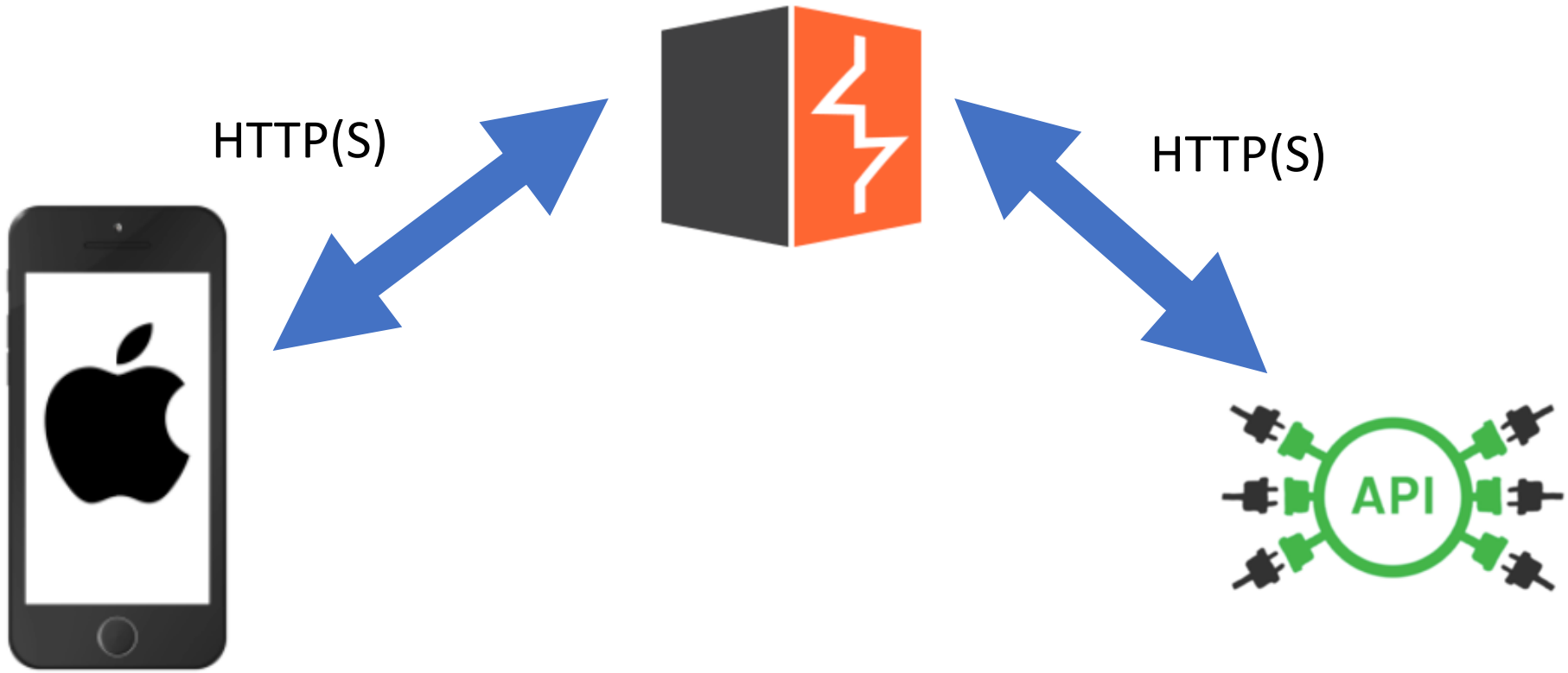
iOS Security Testing Basics – Intercepting Communication

Majority of mobile apps uses HTTP(S), to send data to / receive data from remote endpoints.



iOS Security Testing Basics – Intercepting Communication

To intercept we need to become
Man-in-the-middle (MITM)



iOS Security Testing Basics – Intercepting Communication

- HTTP(S) traffic is redirected to an **interception proxy** running on your VM
- **Monitoring the requests** between the mobile app and API
- Additionally, you can **replay and manipulate requests** to test for server-side bugs

Several free and commercial proxy tools are available. Here are two of the most popular:

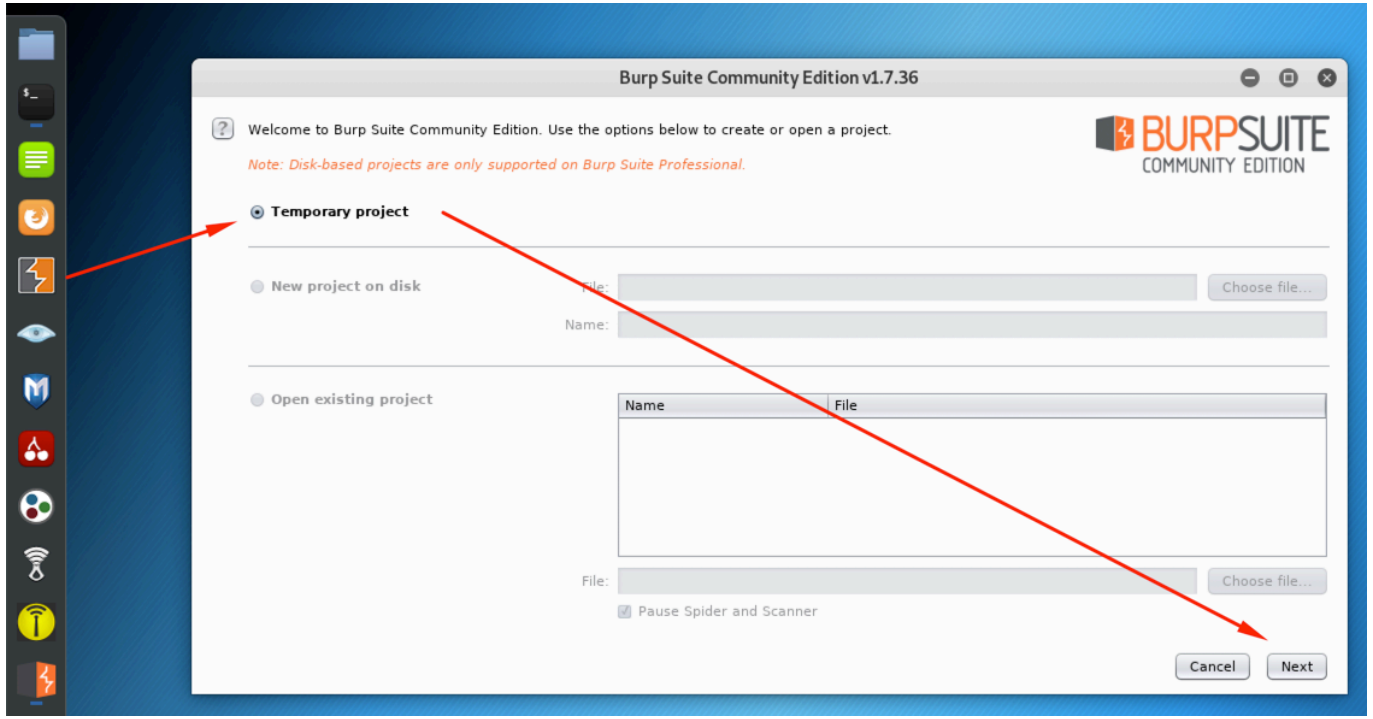
Burp Suite

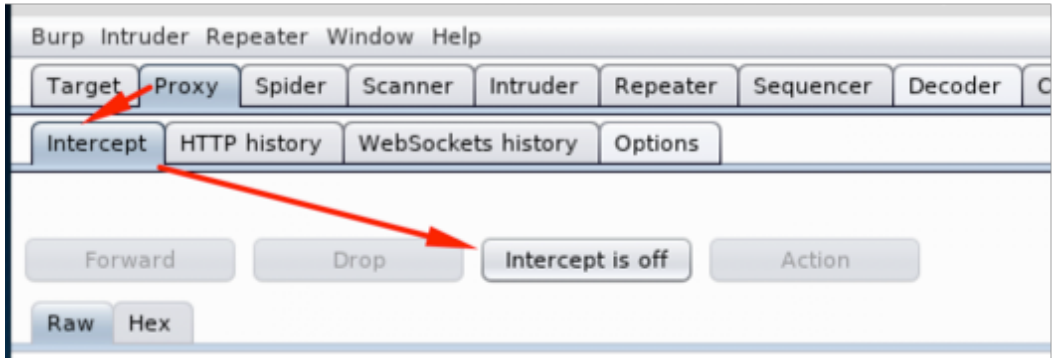
OWASP ZAP

iOS Security Testing Basics – Intercepting Communication

Let's start:

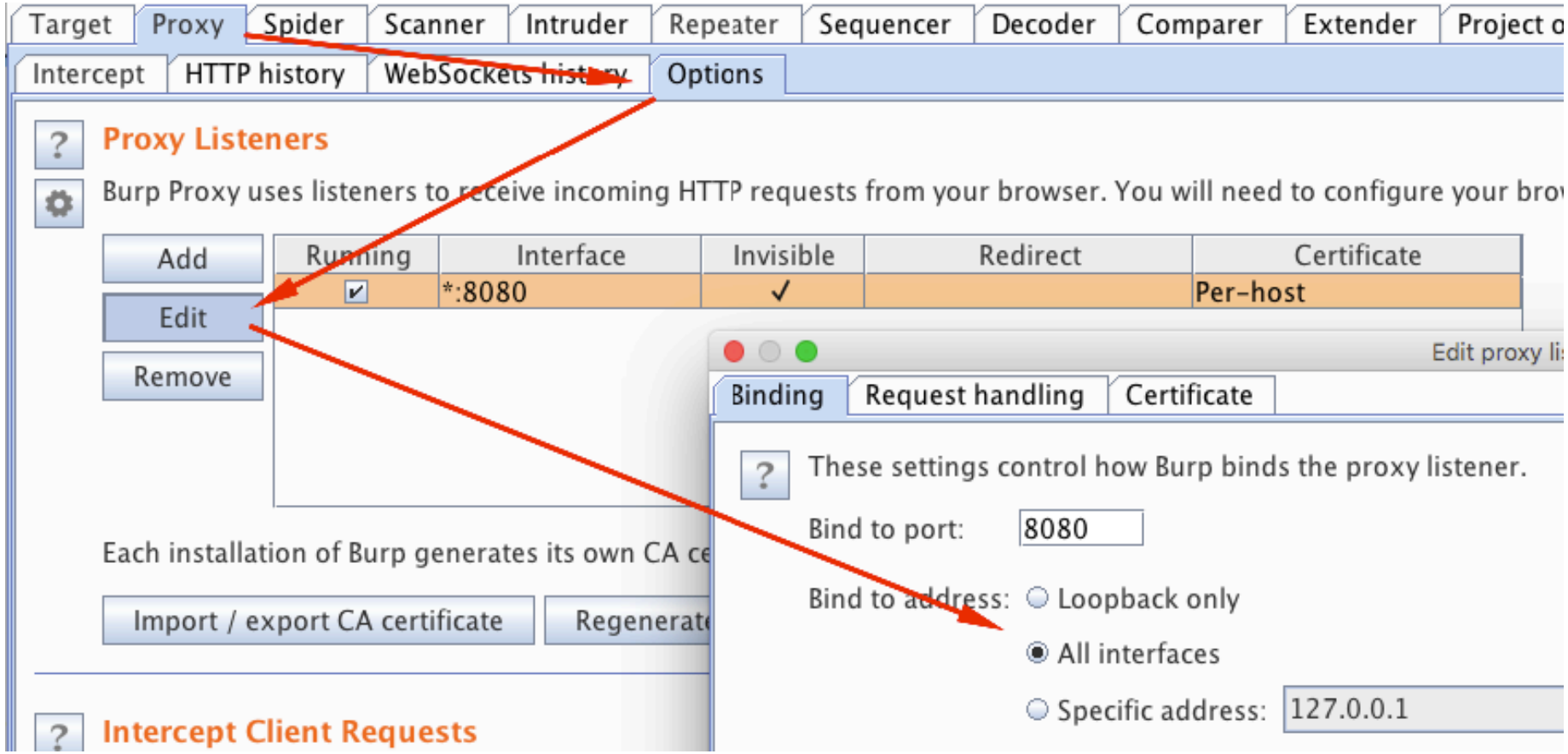
- Start Burp Community Edition
- Select Temporary Project
- Use Burp Defaults and "Start Burp"
- In the "Proxy Tab / Intercept", set "Intercept is off"





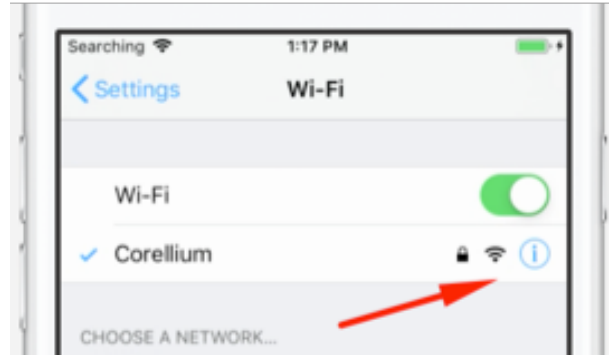
iOS Security Testing Basics – Intercepting Communication

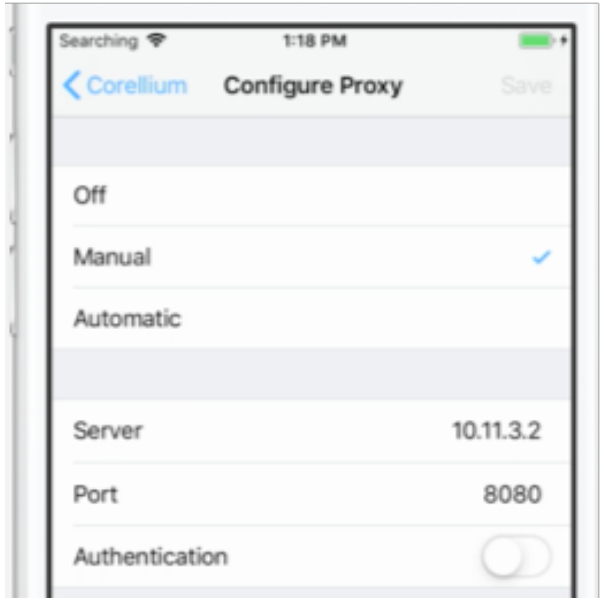
Burp need to listen on all interfaces.



Confirm the warning with ”Yes”.

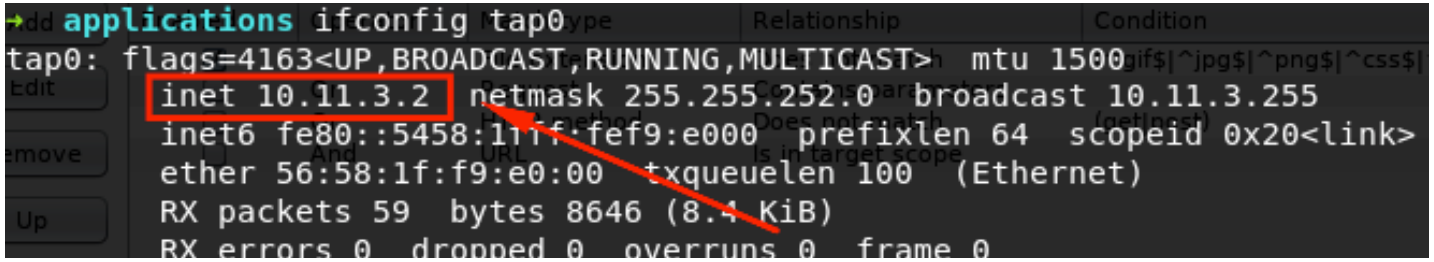
iOS Security Testing Basics – Intercepting Communication





Configure the Proxy settings on the iOS Phone

1. Go to Settings
2. Click on Wi-Fi
3. Select the “Corellim” Wi-Fi Network and press on the blue circle
4. Click on ”Configure Proxy” and select “Manual”
5. Key in the IP address of your VPN connection and port 8080 (default port of Burp)
6. To find out the IP address of your VPN connection (interface tap0) go to your VM :

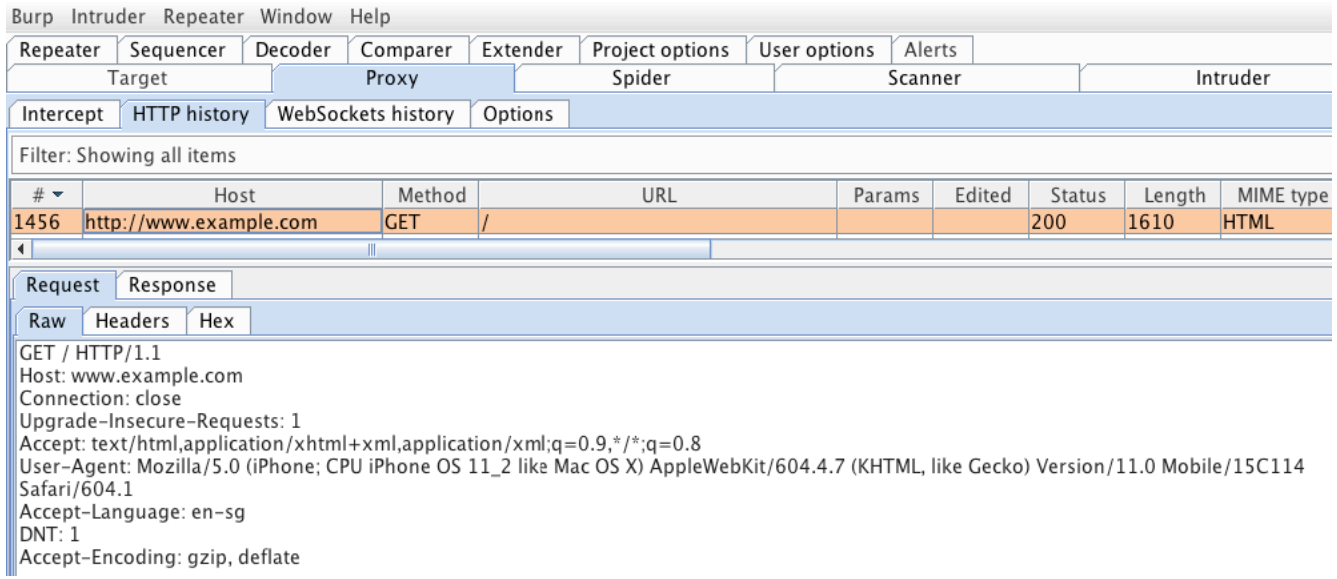


See also: https://support.portswigger.net/customer/portal/articles/1841108-configuring-an-ios-device-to-work-with-burp

iOS Security Testing Basics – Intercepting Communication

Configure the Proxy settings on the iOS Phone

7. Open Safari and go to http://www.example.com



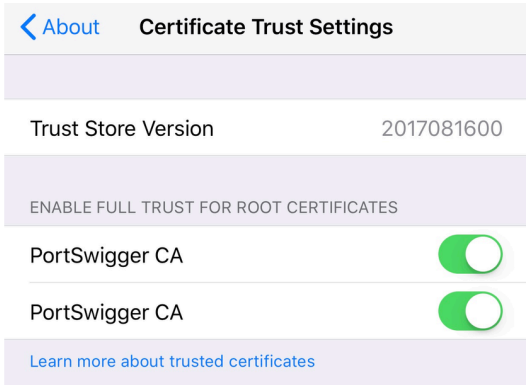
8. Now go to https://www.example.com

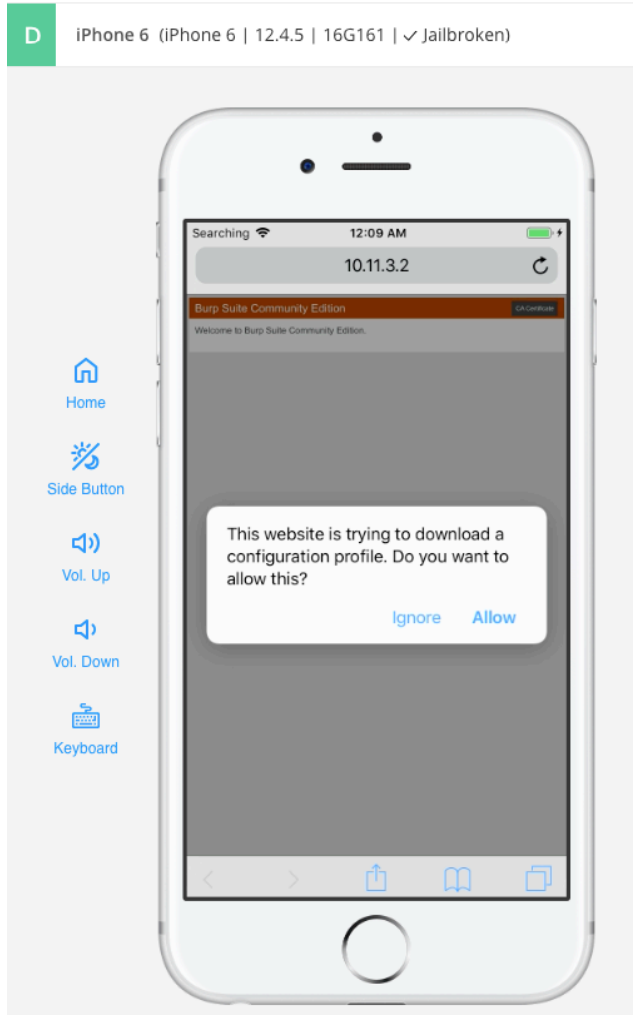
Bonus question: Why are we not getting a certificate error? We are intercepting network communication between a mobile device and a server.

iOS Security Testing Basics – Intercepting Communication

~~Next step is to add the CA of Burp to your iOS device~~ OPTIONAL (ALREADY DONE FOR YOU)

1. Open Safari on your iOS device and type the IP and port 8080 of Burp in the browser bar, 10.11.3.2:8080
2. Click on “CA Certificate” and allow to open the configuration profile
3. The Portswigger CA configuration profile is shown. Click on “Allow”
4. Go to ”Settings”, “General”, “Profiles” and finish the installation of “Portswigger CA”.
5. Finally you need to trust your Burp CA (this setting was introduced in iOS 10):
   - Go to Settings
   - General
   - About
   - Certificate Trust Settings (at the very end of the menu)
   - Enable the Certificate Trust Settings and click on “Continue”





Go to https://www.example.com again. You should see now the traffic in Burp

See also: https://support.portswigger.net/customer/portal/articles/1841109-installing-burp-s-ca-certificate-in-an-ios-device

iOS Security Testing Basics – Intercepting Communication

Dynamic analysis by using an interception proxy can be straight forward. But there are several cases where we run into problems:

Wi-Fi with client isolation

Mobile App frameworks like Google Flutter or Xamarin

XMPP or other non-HTTP protocols are used

~~HTTP~~

Let’s go through it one by one

Intercepting Communication

What if we face the following:

- If XMPP or other non-HTTP protocols are used.

In these cases you need to monitor and analyze the network traffic first in order to decide what to do next.

We did this already earlier today with Wireshark and the remote virtual interface in macOS!

Intercepting Communication – Analyzing Network Traffic

Using Wireshark to identify non-HTTP traffic:

- Close all apps in the background
- Identify IP address of server the app is communicating to
- Filter traffic according to the IP (filter: ip.addr == 192.168.0.102)
- Use the app and inspect the traffic in Wireshark:
  - Which protocols are used?
  - Is the traffic encrypted?
  - Follow TCP stream (http://bit.ly/3683BDj)

Intercepting Communication – Analyzing Network Traffic

So if I know that the traffic is not HTTP, what do I do next?

=> There is no easy answer for this.

Interception proxies such as Burp and OWASP ZAP won't show this traffic because they can only decode HTTP by default.

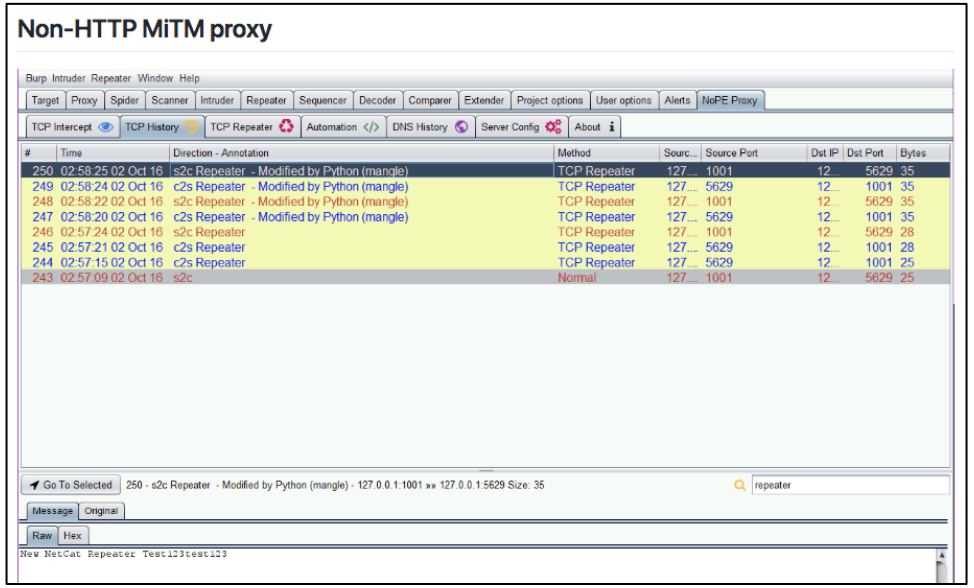
There are, however, Burp plugins that visualize non-HTTP traffic, such as:

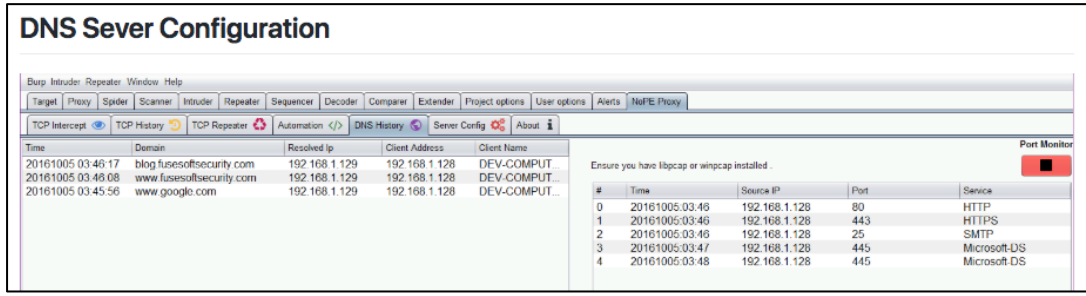
- Burp-non-HTTP-Extension - https://github.com/summitt/Burp-Non-HTTP-Extension
- Mitm-relay - https://github.com/jrmdev/mitm_relay

Intercepting Communication – Analyzing Network Traffic

Burp-non-HTTP-Extension (NoPE Proxy):
https://github.com/summitt/Burp-Non-HTTP-Extension

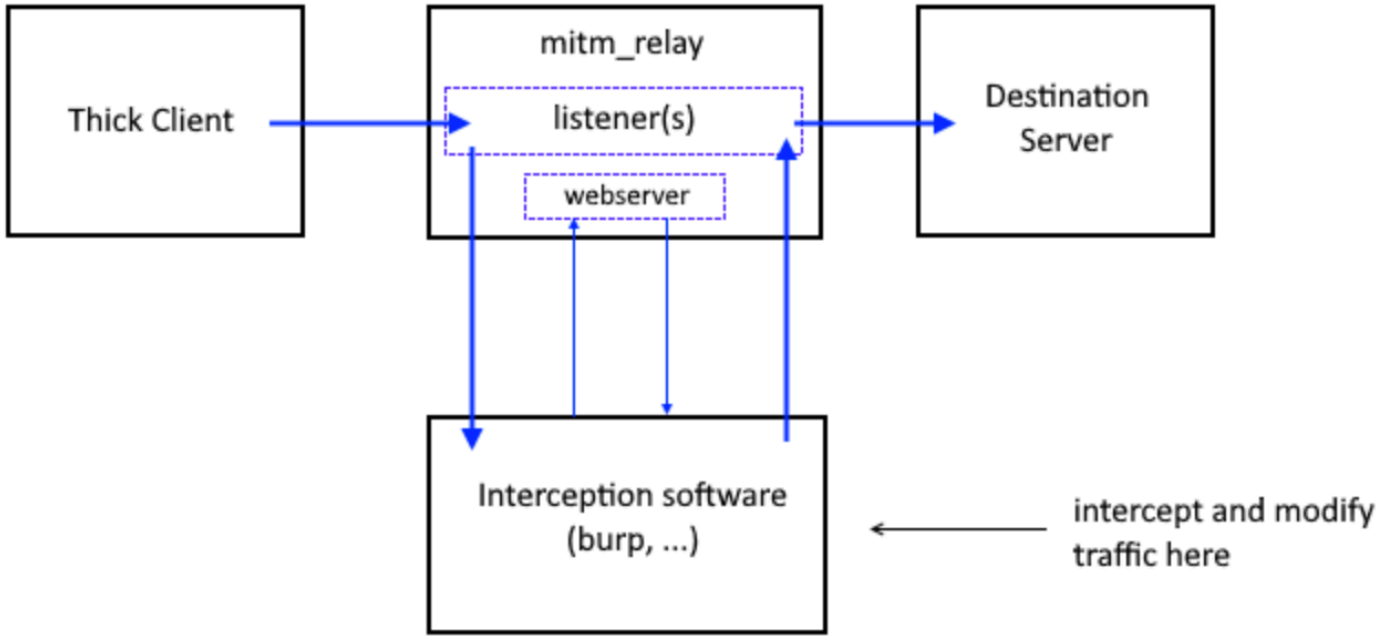
- Configure your laptop as DNS server on Android
- Burp Extension adds DNS Server
- Need to create invisible Proxy listeners
- Can then intercept / replay requests
- Extend the plugin with custom Python scripts





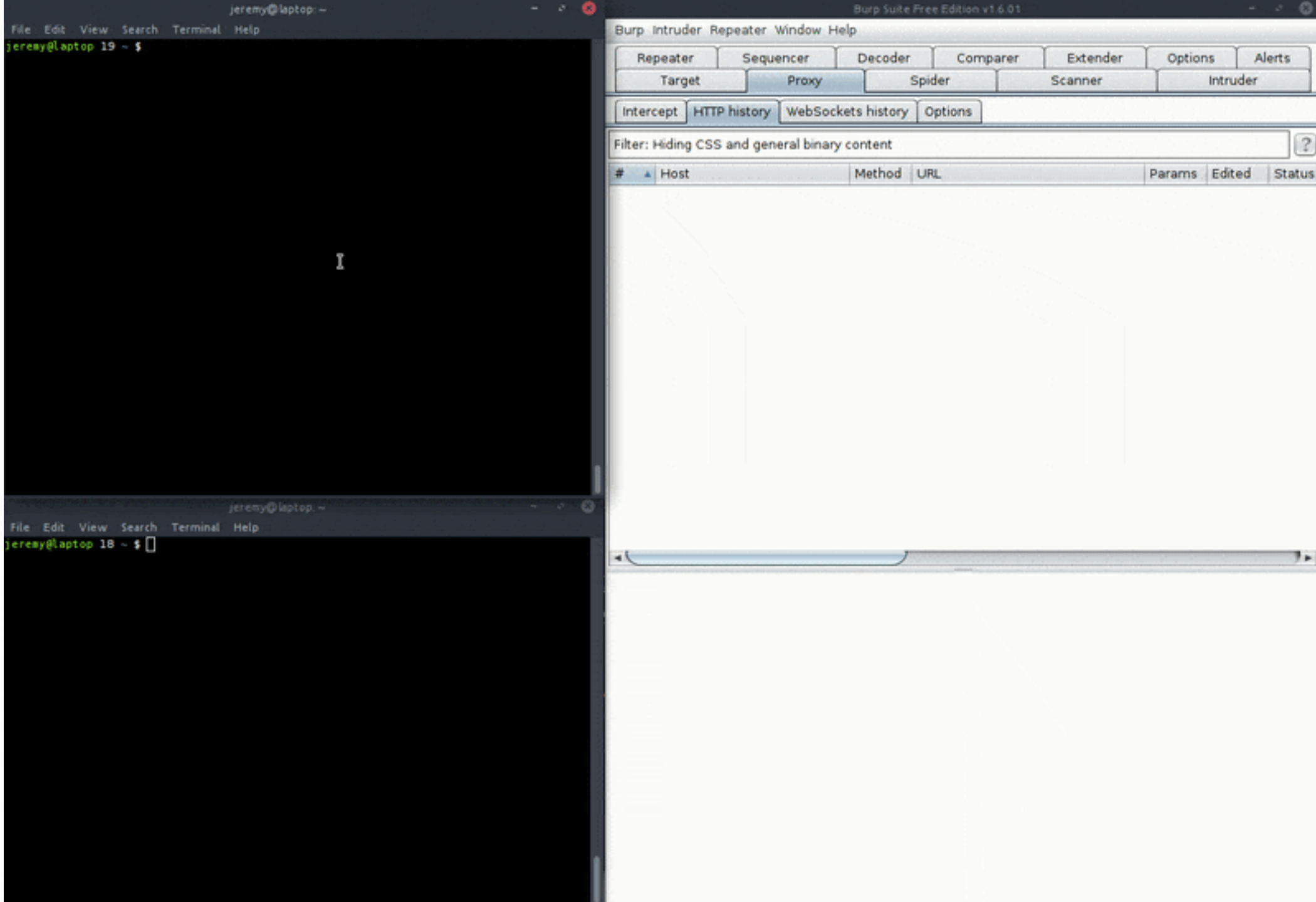
Intercepting Communication – Analyzing Network Traffic

Mitm-relay - https://github.com/jrmdev/mitm_relay



PoC for intercepting DNS with Burp:
https://camo.githubusercontent.com/8bef535888da6dfae15dfbb528d3ab8c7a1a678d/68747470733a2f2f692e696d6775722e636f6d2f304a57596443502e676966

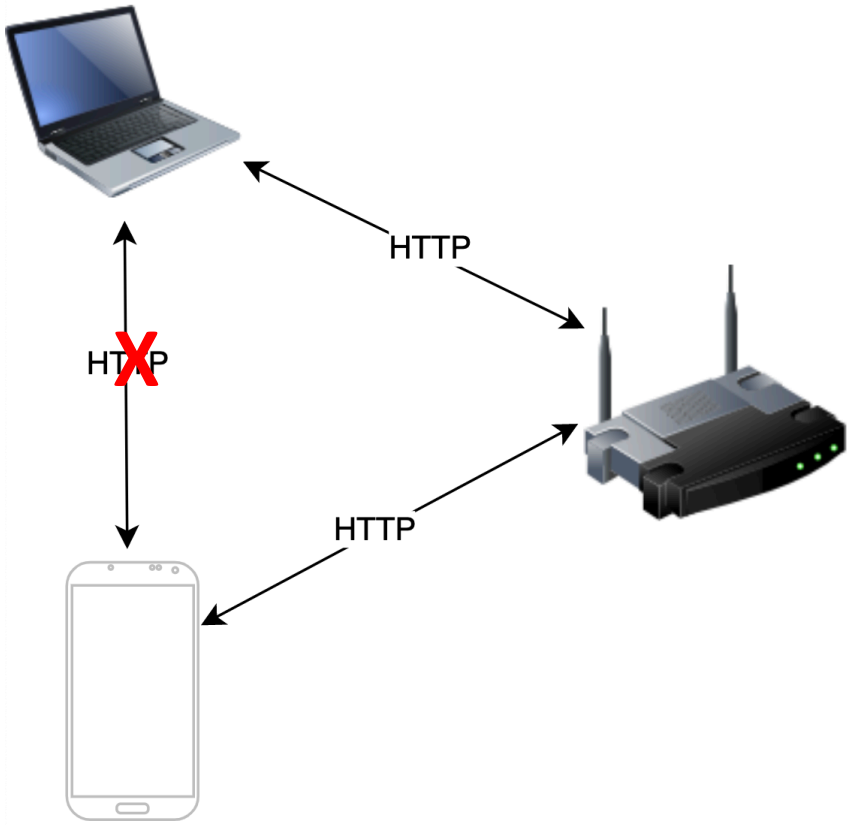
The following PoC shows how Burp was used to intercept DNS requests with mitm-relay



Intercepting Communication – Wireless Client Isolation

Wireless Client Isolation is a security feature that prevents wireless clients from communicating with one another.

What to do if the Wi-Fi we need for testing has client isolation?



We can use SSH Port Forwarding via USB!

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

Intercepting Communication – Wireless Client Isolation

Setup if you have a hardware iOS device connected via USB to your macOS device:
With iproxy we can use SSH via USB:

```
$ brew install libimobiledevice
$ iproxy 2222 22
```

Open another terminal!
The next step is to make a remote port forwarding of port 8080 (Burp) to the iOS device :

```
$ ssh -R 8080:localhost:8080 root@localhost -p 2222
```

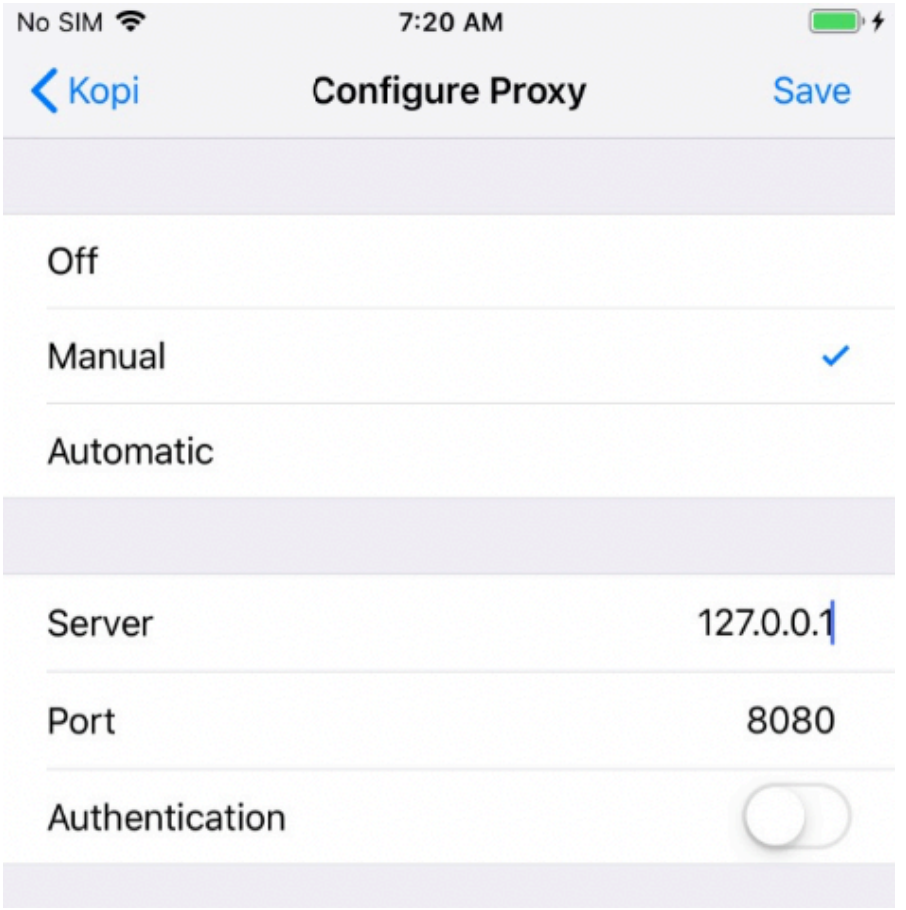
You should now be able to reach Burp on your iOS device. Open Safari on iOS and go to 127.0.0.1:8080 and you will see the Burp Suite Landing Page.

See also in the MSTG:
https://mobile-security.gitbook.io/mobile-security-testing-guide/ios-testing-guide/0x06b-basic-security-testing#using-burp-via-usb-on-a-jailbroken-device

Intercepting Communication – Wireless Client Isolation

Re-configure the Proxy settings:

1. Go to Settings
2. Click on Wi-Fi
3. Select the Wi-Fi Network you connected to and press on the blue circle
4. Click on ”Configure Proxy” and select “Manual”
5. Key in the IP address 127.0.0.1 and the port we just forwarded (port 8080)
6. Open Safari and browse to any page.

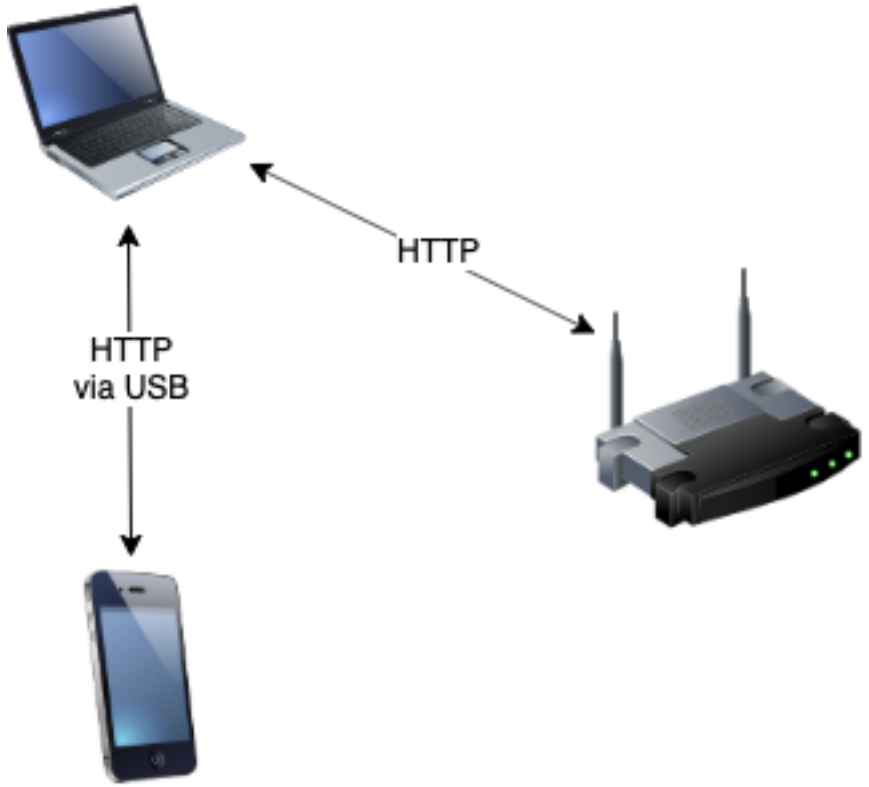


All your HTTP(S) traffic is now going via localhost and a port forwarding rule to your Burp!

Intercepting Communication – Wireless Client Isolation

All HTTP Communication is now going through the Proxy (127.0.0.1:8080) which forwards the traffic via SSH to Burp running on your laptop.

You can also now literally be connected to any Wifi, as you just need to set the proxy and all traffic is anyway forwarded via USB.

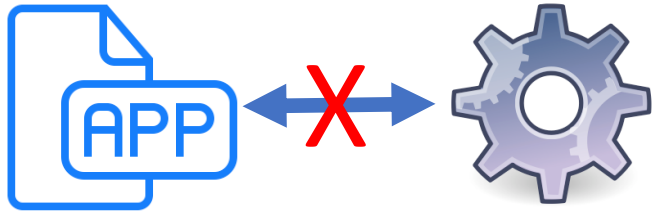


Intercepting Communication – Ignore System Proxy

Dynamic analysis by using an interception proxy can be straight forward, if standard libraries are used in the app and all communication is done via HTTP. But there are several cases where this is not working:

Mobile App frameworks like Google Flutter or Xamarin

Mobile Apps verify if the system proxy is used



System Proxy

Intercepting Communication – Ignore System Proxy

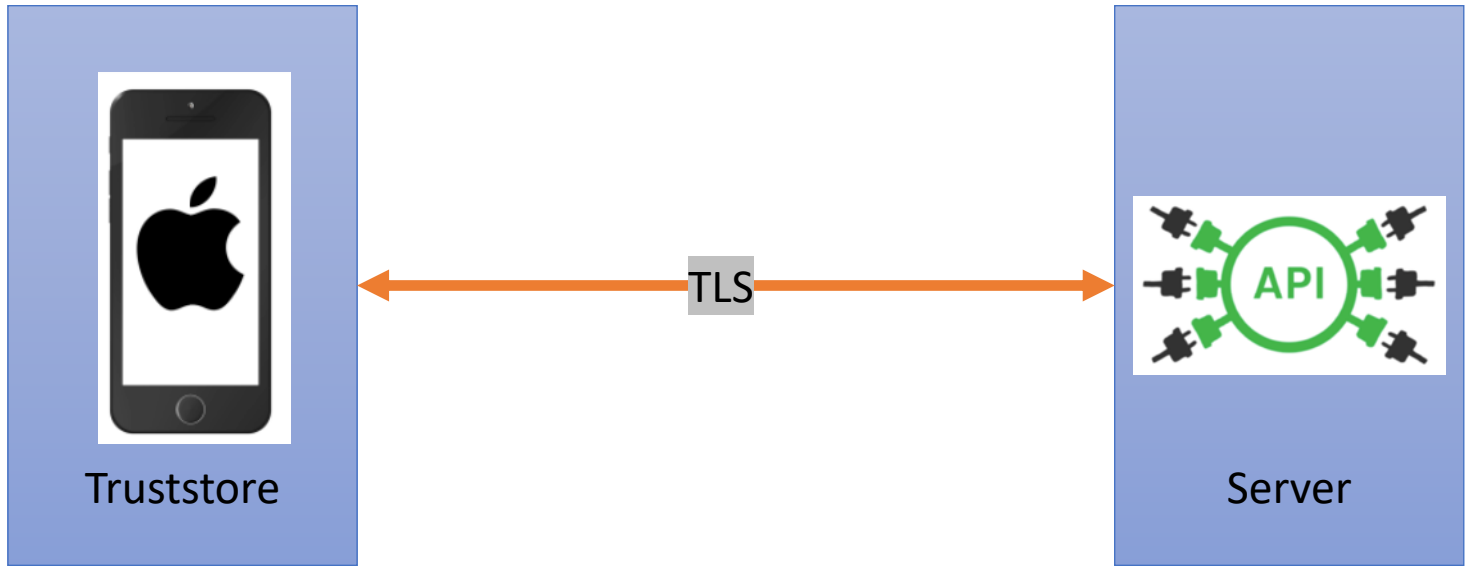
Dynamic analysis by using an interception proxy can be straight forward if standard libraries are used in the app and all communication is done via HTTP. But there are several cases where this is not working:

- If mobile application development platforms like Xamarin are used that ignore the system proxy settings;
- If mobile applications verify if the system proxy is used and refuse to send requests through a proxy;

We can attack on layer 2, by using (b)ettercap:
https://mobile-security.gitbook.io/mobile-security-testing-guide/general-mobile-app-testing-guide/0x04f-testing-network-communication#simulating-a-man-in-the-middle-attack

SSL Pinning on iOS

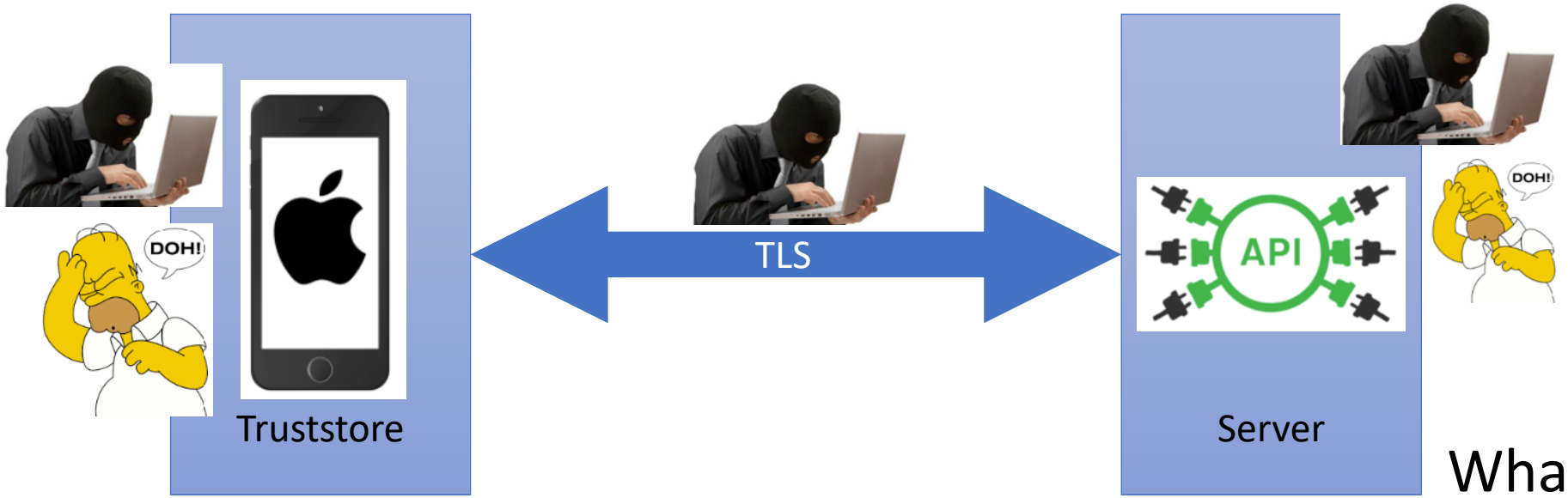
TLS recap



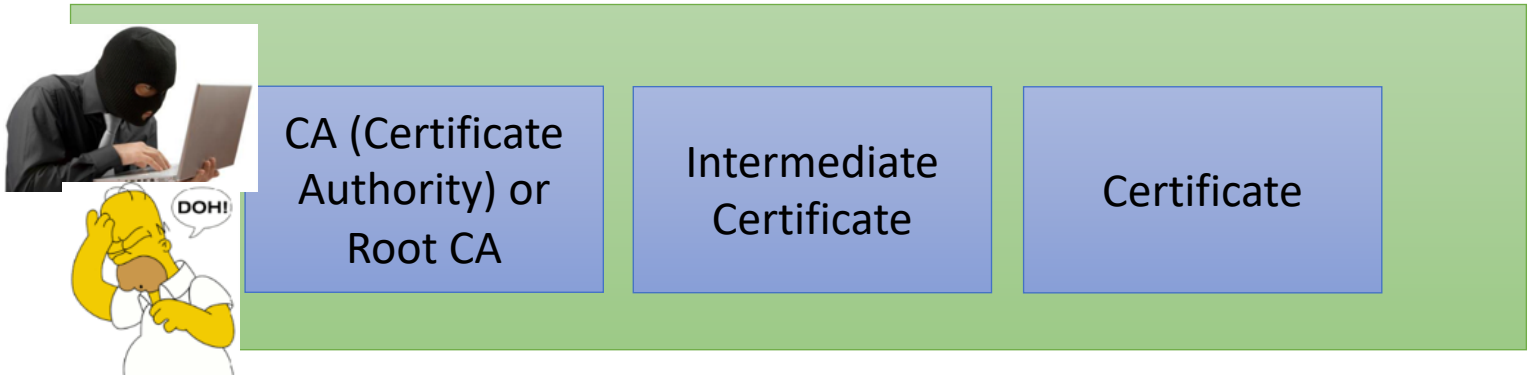
How does TLS protect the communication?

- **Authenticity**, as we can validate the issuer of the certificate.
- **Confidentiality,** as the communication is encrypted.
- **Integrity**, through calculating a message digest.

TLS recap



What can go wrong in TLS and what can be attacked?



Further details: [Testing SSL Pinning on iOS (MSTG)]

TLS recap

Attacks against CA's are real!

Issuance of fraudulent certificates [edit]

On July 10, 2011, an attacker with access to DigiNotar's systems issued a wildcard certificate for Google. This certificate was subsequently used by unknown persons in Iran to conduct a man-in-the-middle attack against Google services.[22][23] On August 28, 2011, certificate problems were observed on multiple Internet service providers in Iran.[24] The fraudulent certificate was posted on pastebin.[25] According to a subsequent news release by VASCO, DigiNotar had detected an intrusion into its certificate authority infrastructure on July 19, 2011.[26] DigiNotar did not publicly reveal the security breach at the time.

After this certificate was found, DigiNotar belatedly admitted dozens of fraudulent certificates had been created, including certificates for the domains of Yahoo!, Mozilla, WordPress and The Tor Project.[27]

DigiNotar was bankrupt in September 2011

https://www.cnet.com/news/fraudulent-google-certificate-points-to-internet-attack/

TLS recap

Man in the Middle attacks through CA's are real!

Kazakhstan man-in-the-middle attack

From Wikipedia, the free encyclopedia

In 2015, the government of Kazakhstan created a "national security certificate" which would have allowed a man-in-the-middle attack on HTTPS traffic from Internet users in Kazakhstan. Such an attack would involve requiring all Internet users to install a root certificate controlled by the Kazakh government into all their devices, allowing it to intercept, decrypt, and re-encrypt any traffic passing through systems it controlled.[1][2]

In July 2019, Kazakh ISPs started messaging their users that the certificate, now called the **Qaznet Trust Certificate**,[3] issued by the state certificate authority the **Qaznet Trust Network**, would now have to be installed by all users.[4][5][6]

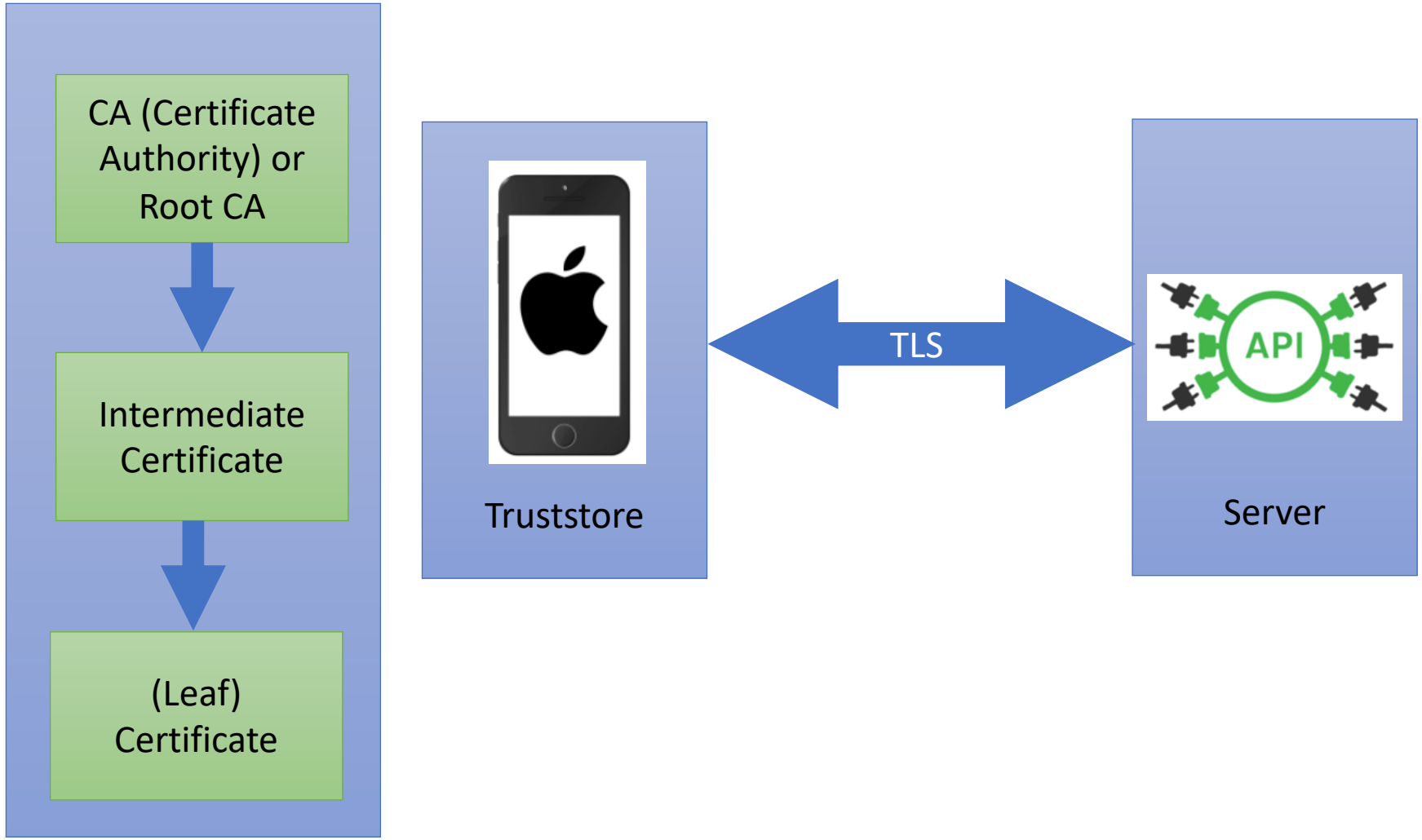
Sites operated by Google, Facebook and Twitter appear to be among the Kazakh government's initial targets.[7]

On August 21, 2019, Mozilla and Google simultaneously announced that their Firefox and Chrome web browsers would not accept the government-issued certificate, even if installed manually by users.[8][9] Apple also announced that they would make similar changes to their Safari browser.[7] As of August 2018, Microsoft has so far not made any changes to its browsers, but reiterated that the government-issued certificate was not in the trusted root store of any of its browsers, and would not have any effect unless a user manually installed it.[10]

https://en.wikipedia.org/wiki/Kazakhstan_man-in-the-middle_attack

SSL Pinning

What is SSL Pinning?



X.509 v3 Digital Certificate

Certificate Field
Version Number
Serial Number
Signature Algorithm Identifier
Issuer Name
Validity Period (Not Before / Not After)
Subject Name
Subject Public-Key Information • Public Key Algorithm • Subject Public Key
Issuer Unique Identifier
Subject Unique Identifier
Extensions
Certification Authority's Digital Signature

SSL Pinning

What should be pinned against?

	Certificate Pinning	SPKI* / Public Key Pinning
Ease of installation	Easy to implement	Easy to implement nowadays
Expiry	When certificate expires	When you stop using the public key
Challenges	• CAs might have multiple Certificates • Need to be updated more often	• How long is it secure to use the same public key? • Self-signed CA?

*SPKI = Subject Public Key Info (Fingerprint/Thumbprint of the certificate)

SSL Pinning

What are the dangers of SSL Pinning aka should I pin?

- Is the organization mature enough
- PKI with certificate lifecycle management need to be established
- Private key need to be protected!
- Fail hard / Fail soft?

SSL Pinning

Who should do SSL Pinning?

JPMorganChase

Bank of America

⇒ **The decision for (not) using SSL Pinning should be the result of a Threat Model. Meaning we also shouldn't blindly flag out missing SSL Pinning as a finding in penetration test reports.**

SSL Pinning

What can we do to disable SSL Pinning?

- Repackage the App and patch the SSL Pinning function
- SSL Kill Switch 2 (Jailbroken device)
- Burp Suite Mobile Assistant
- **Frida (Jailbroken / non-jailbroken device)**

SSL Pinning

SSL Pinning can be bypassed on a jailbroken device by using SSL Kill Switch 2

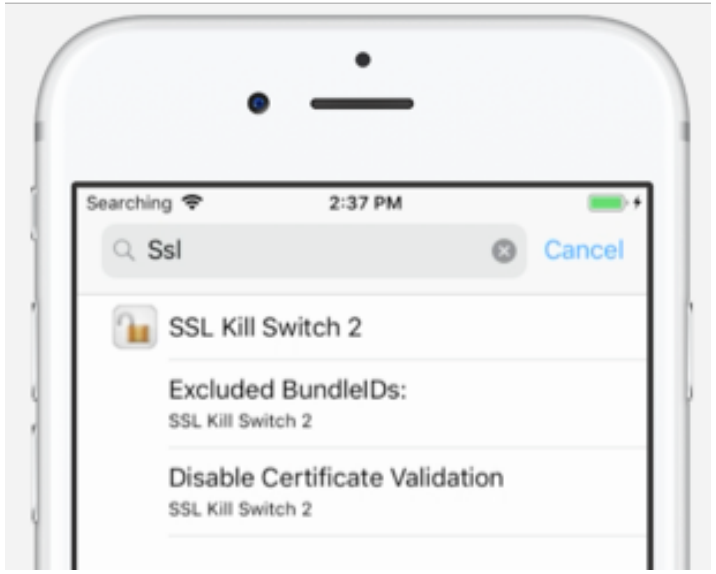
Description

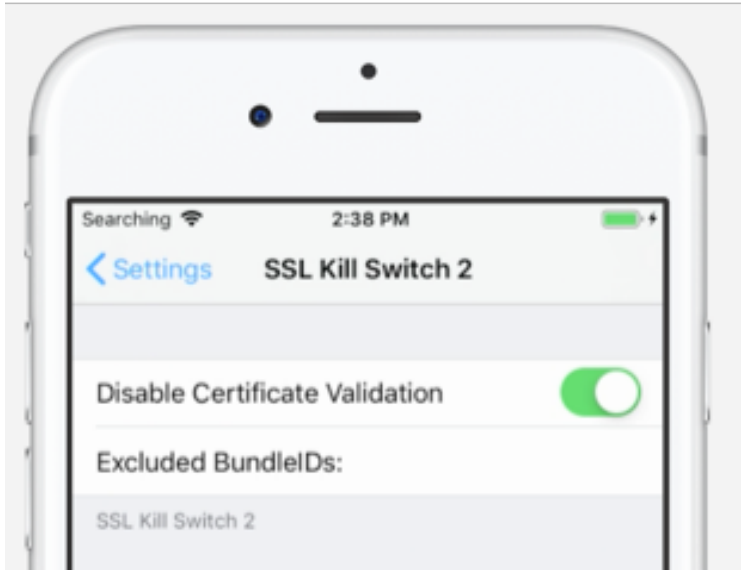
Once loaded into an iOS or OS X App, SSL Kill Switch 2 patches specific low-level SSL functions within the Secure Transport API in order to override, and disable the system's default certificate validation as well as any kind of custom certificate validation (such as certificate pinning).

https://github.com/nabla-c0d3/ssl-kill-switch2

SSL Pinning

When installed, SSL Kill Switch is an additional setting in iOS:





Once activated it will overwrite low level SSL classes and just accept any certificates and bypasses SSL systemwide.

SSL Pinning

SSL Pinning can be bypassed on a jailbroken device by using

Burp Suite Mobile Assistant

Burp Suite Mobile Assistant is a tool to facilitate testing of iOS apps with Burp Suite.

If you do not already have Mobile Assistant installed, please see the help on Installing Burp Suite Mobile Assistant.

Once installed, Burp Suite Mobile Assistant can be launched just like any other app on your device. Simply tap the app's icon to get started.

https://portswigger.net/burp/documentation/desktop/tools/mobile-assistant

SSL Pinning

Now it's your turn:
Bypass SSL Pinning in our lab!

Open the file Lab-iOS.pdf:

```
$  evince ~/MSTG-Handson/Lab/Lab-iOS.pdf
```

Section: **Lab – SSL Pinning**

Please type the commands and do not copy & paste!

1. You will remember them easier
2. Some PDF readers do not copy properly and miss whitespaces etc.

SSL Pinning - Disable SSL Pinning with Objection!

```
➜ node-applesign git:(master) ✗ objection -g Snapchat explore
Using USB device `iPhone`
Agent injected and responds ok!

     _  _         _  _
 ___| |_|_|___ ___| |_|_|___ ___
| . | . | | -_|  _|  _| | . |   |
|___|___| |___|___|_| |_|___|_|_|
      |___|(object)inject(ion) v1.8.0

     Runtime Mobile Exploration
       by: @leonjza from @sensepost

[tab] for command suggestions
com.toyopagroup.picaboo on (iPhone: 12.4) [usb] # ios sslpinning disable
(agent) Hooking common framework methods
(agent) Found NSURLSession based classes. Hooking known pinning methods.
(agent) Hooking lower level SSL methods
(agent) Hooking lower level TLS methods
(agent) Hooking BoringSSL methods
(agent) Registering job ll7y6nstkk9. Type: ios-sslpinning-disable
com.toyopagroup.picaboo on (iPhone: 12.4) [usb] # (agent) [ll7y6nstkk9] Called SSL_CTX_set_custom_verify(), setting custom
(agent) [ll7y6nstkk9] Called custom SSL context verify callback, returning SSL_VERIFY_NONE.
(agent) [ll7y6nstkk9] Called SSL_get_psk_identity(), returning "fakePSKidentity".
(agent) [ll7y6nstkk9] Called SSL_CTX_set_custom_verify(), setting custom callback.
(agent) [ll7y6nstkk9] Called custom SSL context verify callback, returning SSL_VERIFY_NONE.
(agent) [ll7y6nstkk9] Called nw_tls_create_peer_trust(), no working bypass implemented yet.
(agent) [ll7y6nstkk9] Called SSL_get_psk_identity(), returning "fakePSKidentity".
(agent) [ll7y6nstkk9] Called SSL_CTX_set_custom_verify(), setting custom callback.
(agent) [ll7y6nstkk9] Called SSL_CTX_set_custom_verify(), setting custom callback.
(agent) [ll7y6nstkk9] Called custom SSL context verify callback, returning SSL_VERIFY_NONE.
(agent) [ll7y6nstkk9] Called SSL_get_psk_identity(), returning "fakePSKidentity".
(agent) [ll7y6nstkk9] Called SSL_CTX_set_custom_verify(), setting custom callback.
(agent) [ll7y6nstkk9] Called custom SSL context verify callback, returning SSL_VERIFY_NONE.
(agent) [ll7y6nstkk9] Called nw_tls_create_peer_trust(), no working bypass implemented yet.
(agent) [ll7y6nstkk9] Called SSL_get_psk_identity(), returning "fakePSKidentity".
(agent) [ll7y6nstkk9] Called SSL_CTX_set_custom_verify(), setting custom callback.
(agent) [ll7y6nstkk9] Called SSL_CTX_set_custom_verify(), setting custom callback.
(agent) [ll7y6nstkk9] Called custom SSL context verify callback, returning SSL_VERIFY_NONE.
(agent) [ll7y6nstkk9] Called SSL_get_psk_identity(), returning "fakePSKidentity".
(agent) [ll7y6nstkk9] Called SSL_CTX_set_custom_verify(), setting custom callback.
(agent) [ll7y6nstkk9] Called custom SSL context verify callback, returning SSL_VERIFY_NONE.
(agent) [ll7y6nstkk9] Called nw_tls_create_peer_trust(), no working bypass implemented yet.
(agent) [ll7y6nstkk9] Called SSL_get_psk_identity(), returning "fakePSKidentity".
(agent) [ll7y6nstkk9] Called SSL_CTX_set_custom_verify(), setting custom callback.
```

#	Host	Method	URL
14	https://auth.snapchat.com	POST	/scauth/login

Request | Response

Raw | Params | Headers | Hex

v8:7B2AA00DB3705DC455321A2E6715BB20:1D86A957939CE0A59871F42BEE6888
7B290C70636D75B179510EC22E1AB199510094F0A2B1FA2CF14D6D057F0AC40E
84FDD67A71F06C07F4F75266FA727883A28A94F5C0A9CC473D96B440CEE2A0F7
5CF52F6474E504CC2163AFC21E419EBEB2BDE316476299A120BA59279EF5A2A3
1F678EAD5142169F751AB948271752AFB4F2C16CD0C28E43A7A278F05C761689
D646FEA84EE651DBF8C7945FE75D7A1F8639B024898B60818C27D5651D8B168A
9D642DE6CF870A00DD62757D694495BF3E0C970994C05A110504EE978E068407
2746516EF40A0CF445

Connection: close
X-Snapchat-UUID: B4F648E0-C4F5-4B28-A0A5-51EAF1E61589

device_check_token=DEVICE_CHECK_TOKEN_NOT_AVAILABLE_GTE_IOS11&dsig=098DADC79E1C89FD5095&dtoken1i=00001%3An7zHksq%2FS0GI7W2WhXUGT0MnJ9JzwWi%2BH6Yj%2FHMywiOxrrGyujNaLOFxl%2FPDCW6Z&fidelius_client_init=%7B%22new_fidelius_version%22%3A9%2C%22hashed_out_betas%22%3A[%22vIUF3nTHtm%2F%2FrJKwHhmLMfofLVMoYEmgF4MNJBvjoYw%3D%22]%2C%22new_hashed_out_beta%22%3A%22vIUF3nTHtm%2F%2FrJKwHhmLMfofLVMoYEmgF4MNJBvjoYw%3D%22%2C%22new_iwek%22%3A%22usQfCuQEy0TNqVXe%2BDp9AX2MCYh6Mz8PGFFFE7E9sRQ%3D%22%2C%22new_out_beta%22%3A%22MFkwEwYHKoZIzj0CAQYIKoZIzj0DAQcDQgAED9zNAa3%2BBC9shUoLPmWbPmc7N%2FUTB9kl645kjPt6qbNk0ugNDLoN8laQmQa43XpFVNMZYquznFIPCUtRjrcC7g%3D%3D%22%7D&from_deeplink=false&height=1136&odlv_pre_auth_token=&password=qawswdef&pre_auth_token=&reactivation_confirmed=false&remember_device=true&req_token=93057c5191a124086ec79de9faa4c8b4d9e34d8719aaba8199b4fb14dfc51cbb&screen_height_in=3.5&screen_height_px=568&screen_width_in=1.9&screen_width_px=320&timestamp=1526254602508&username=qqqq%40qqq.de&width=640

SSL Pinning

Seems SSL Pinning can be bypassed quite easily.

Anything we can suggest to developers to make it harder for an attacker to bypass?

⇒ **SSL Pinning is a client-side control and with enough time and skills it will always be possible to bypass for a reverse engineer!**

⇒ **Additional security controls might make it harder (Jailbreak detection, detection of dynamic instrumentation or manipulation of the IPA)**

SSL Pinning

Best Practice for implementing SSL Pinning on iOS:

- Use TrustKit: here you can pin by setting the public key hashes in your Info.plist or provide the hashes in a dictionary.

- Use AlamoFire: here you can define a ServerTrustPolicy per domain for which you can define the pinning method.

- Use AFNetworking: here you can set an AFSecurityPolicy to configure your pinning.

Static Analysis - iOS Apps

Static Analysis of iOS Apps

Manual

A common approach is by searching/"grepping" for certain APIs and keywords. Besides of the MSTG, another good source of secure coding and keywords in iOS can be found in:

- "iOS Application Security: The Definitive Guide for Hackers and Developers" by David Thiel
- Needle has a list of useful keywords: https://github.com/FSecureLABS/needle/blob/master/needle/modules/static/code_checks.py

Automatic

Automated analysis tools can be used to speed up the review process of Static Application Security Testing (SAST). They check the source code for compliance with a predefined set of rules or industry best practices.

In most cases, a hybrid of automatic and manual approach is used. Automatic scans catch the low-hanging fruits, and the human tester can explore the code base with specific usage contexts in mind.

Static Analysis of iOS Apps

MobSF: Open Source Mobile Security Framework

What can I analyse with MobSF?

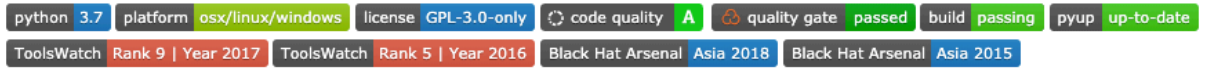
- Class-dump
- Info.plist
- App Transport Security (ATS)

Mobile Security Framework (MobSF)

Version: v3.0 beta

Mobile Security Framework (MobSF) is an automated, all-in-one mobile application (Android/iOS/Windows) pen-testing, malware analysis and security assessment framework capable of performing static and dynamic analysis. MobSF support mobile app binaries (APK, IPA & APPX) along with zipped source code and provides REST APIs for seamless integration with your CI/CD or DevSecOps pipeline.The Dynamic Analyzer helps you to perform runtime security assessment and interactive instrumented testing.

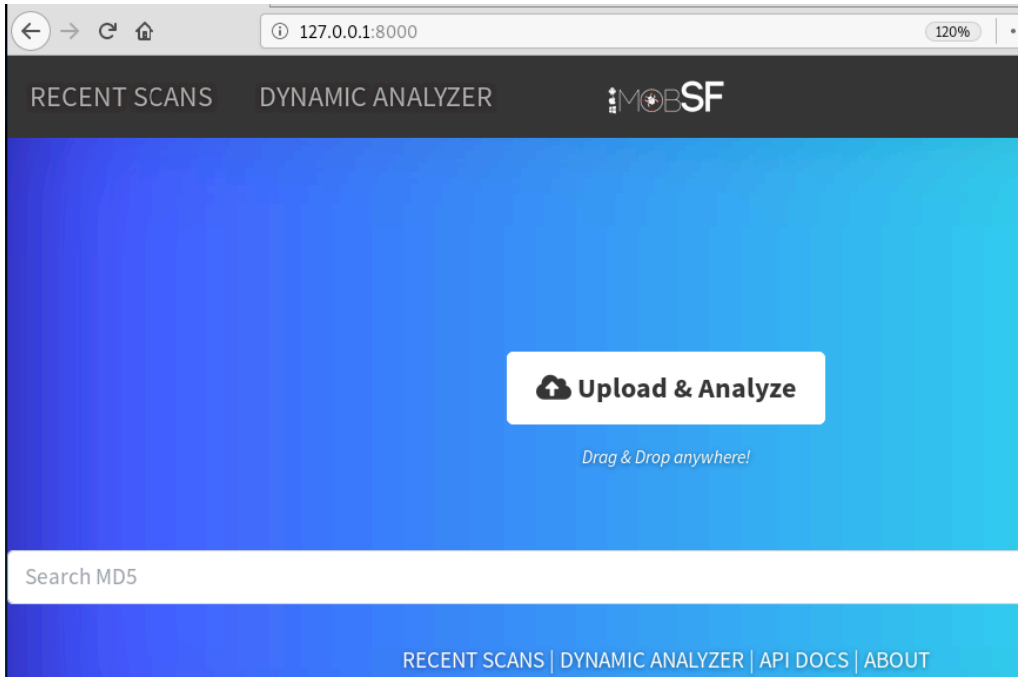
Made with ❤ in India



Static Analysis of iOS Apps

Start MobSF in the VM:

```
$ cd ~/Tools/Mobile-Security-Framework-MobSF/
$ ./run.sh
```



Open Firefox in your VM and go to 127.0.0.1:8000

Click on “Upload & Analyze” and upload the app **DVIA-v2-swift.ipa** in the directory ~/MSTG-Handson/App

This will take now a few minutes. Let’s continue while its scanning!

Static Analysis of iOS Apps

What is a class-dump?

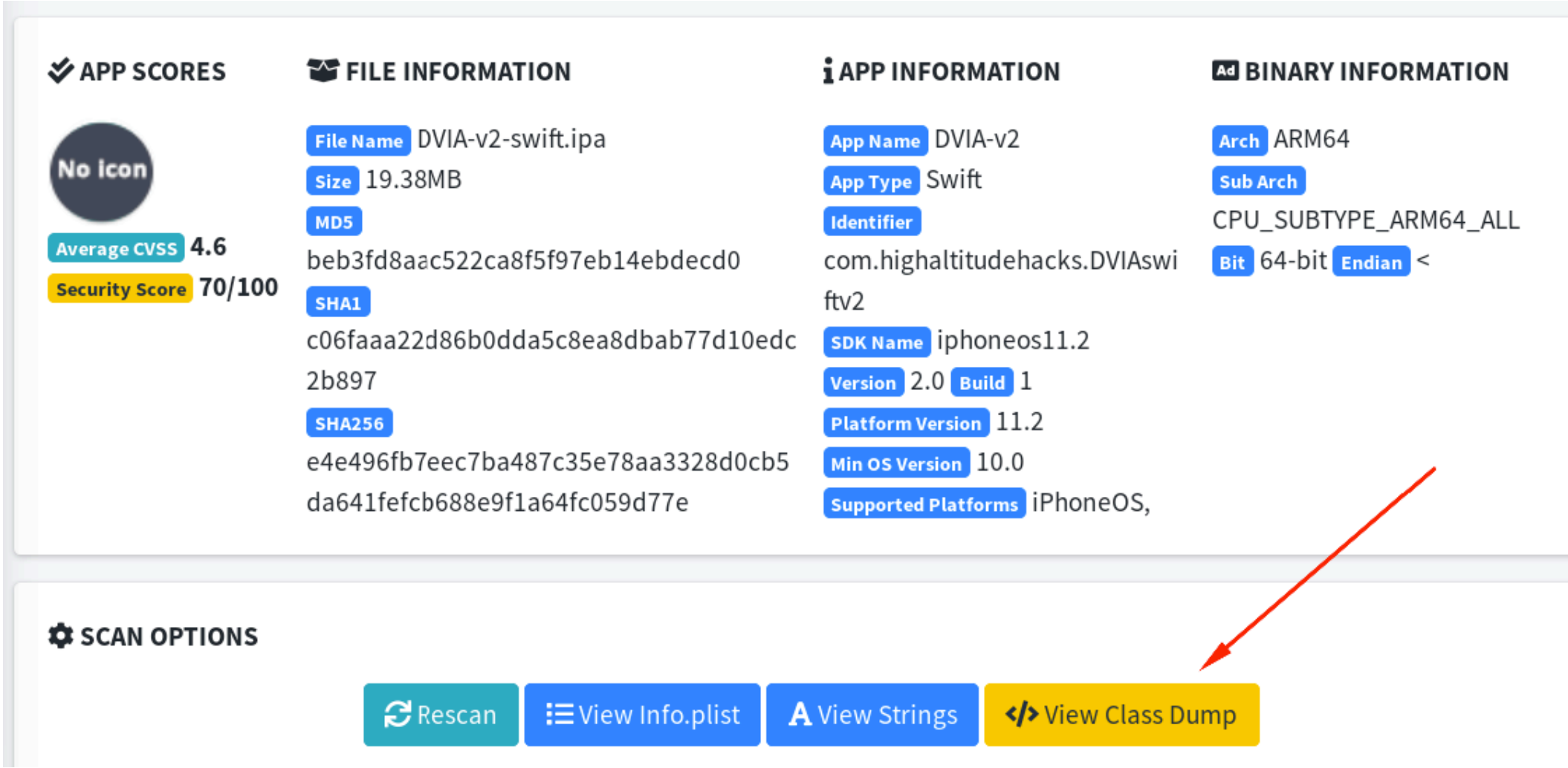
class-dump

class-dump is a command-line utility for examining the Objective-C segment of Mach-O files. It generates declarations for the classes, categories and protocols. This is the same information provided by using 'otool -ov', but presented as normal Objective-C declarations.

https://github.com/nygard/class-dump

Static Analysis of iOS Apps

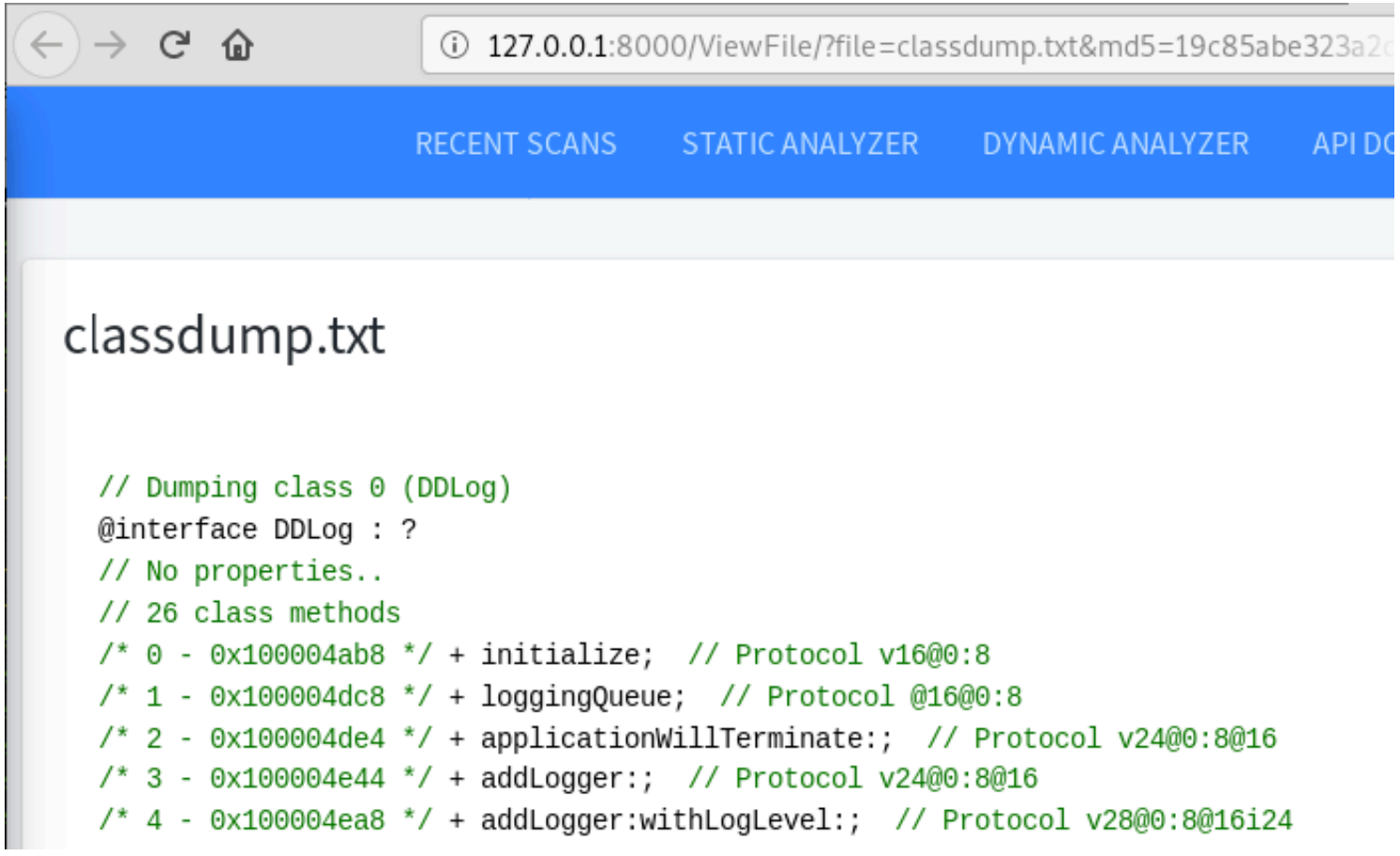
If iOS Apps are written in Objective-C or have at least parts of Objective-C a class-dump is available:



Static Analysis of iOS Apps

What can I use class-dump for?

If the code is not obfuscated, you can see class names, method names and variable names. This is the first step for Reverse Engineering an iOS app and getting basic information.



Search for Keywords like:

- **username**
- **password**
- **login**
- **sensitive**
- **…**

Static Analysis of iOS Apps

You might see classes that look like this:

```
/* 8 - 0x1001a9310 */ - applicationWillTerminate:;  // Protocol v24@0:8@16
/* 9 - 0x1001a9764 */ - init;  // Protocol @16@0:8
/* 10 - 0x1001a965c */ - .cxx_destruct;  // Protocol @?
@end
// Dumping class 129 (_TtC7DVIA_v234SensitiveInformationViewController)
@interface _TtC7DVIA_v234SensitiveInformationViewController : ?
// No properties..
// Adjusting num elems: 2
```

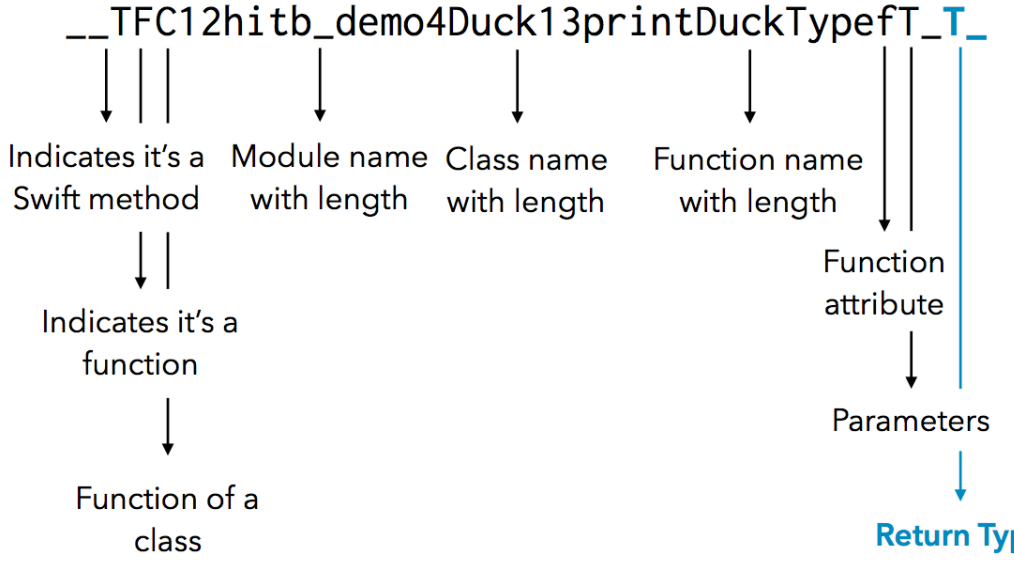
This is a disassembled Swift Class!

Static Analysis of iOS Apps

Objective-C and Swift are fundamentally different, the programming language in which the app is written affects the possibilities for reverse engineering it. For example, Objective-C allows method invocations to be changed at run time. This makes hooking into other app functions (a technique heavily used by Cycript and other reverse engineering tools) easy. This "method swizzling" is not implemented the same way in Swift, and the difference makes the technique harder to execute with Swift than with Objective-C.

```
1   -[ViewController showJailbreakAlert]
2   -[ViewController protectAgainstDebugger]
3   -[ViewController viewDidLoad]
4   -[ViewController handleButtonClick:]
5   -[ViewController didReceiveMemoryWarning]
6   -[ViewController theLabel]
7   -[ViewController setTheLabel:]
```

Objective-C



Swift

Trustwave SpiderLabs

Static Analysis of iOS Apps

In Objective-C, object methods are called via dynamic function pointers called "selectors," which are resolved by name during run time. The advantage of run-time name resolution is that these names need to stay intact in the final binary, making the disassembly more readable. But only a handful of disassembler are resolving the methods name which hugely aid the RE process.

```
UIAlertController *alert = [UIAlertController alertControllerWithTitle:@"Jailbreak Detected!”
message:@"TThis app is unavailable on jailbroken devices.” preferredStyle:UIAlertControllerStyleAlert];
```

Ghidra

```
(*(code *)_objc_msgSend)
          (PTR__OBJC_CLASS_$_UIAlertController_00010e4c,
           PTR_s_alertControllerWithTitle:message_00010d44,
&cf_JailbreakDetected!,
           &cf_TThisappisunavailableonjailbrokendevices.,1);
```

Hopper

```
var_10 = [[UIAlertController
alertControllerWithTitle:@"Jailbreak
Detected!" message:@"TThis app is
unavailable on jailbroken devices."
preferredStyle:0x1] retain];
```

Static Analysis of iOS Apps

Open **iGoat-Swift_v1.0.ipa** in MobSF.
The app can be found in **~/MSTG-Handson/Apps**

=> Check the console where you started MobSF. Any errors?

```
[INFO] 14/Oct/2019 22:37:38 - Running jtool against the binary for dumping classes
[ERROR] 14/Oct/2019 22:37:39 - class-dump-z/class-dump-swift failed on this binary
[INFO] 14/Oct/2019 22:37:39 - Running strings against the Binary
[INFO] 14/Oct/2019 22:37:40 - Connecting to DB
```

MobSF should be run on macOS, to get the most out of the tool!
Class-dump-z is not supported on Linux!

https://github.com/MobSF/Mobile-Security-Framework-MobSF/issues/996

Static Analysis of iOS Apps

A very good example for manual reverse engineering of iOS Apps can be found in the MSTG:

https://mobile-security.gitbook.io/mobile-security-testing-guide/ios-testing-guide/0x06c-reverse-engineering-and-tampering#manual-reversed-code-review

iOS – Static Analysis – App Transport Security

Check the Info.plist for ATS of the iGoat App in MobSF:

APP TRANSPORT SECURITY (ATS)

Search:

ISSUE	STATUS	DESCRIPTION
App Transport Security is allowed	insecure	App Transport Security restrictions are disabled for all network connections. Disabling ATS means that unsecured HTTP connections are allowed. HTTPS connections are also allowed, and are still subject to default server trust evaluation. However, extended security checks like requiring a minimum Transport Layer Security (TLS) protocol version—are disabled. This setting is not applicable to domains listed in NSExceptionDomains.

Showing 1 to 1 of 1 entries

Previous 1 Next

iOS – Static Analysis – App Transport Security

App Transport Security (ATS) is a set of security checks that the operating system enforces when making connections with:

- NSURLConnection
- NSURLSession
- CFURL

ATS is enabled by default for applications build on iOS SDK 9 and above.

ATS is enforced only when making connections to public hostnames. Therefore any connection made to an IP address, unqualified domain names or TLD of .local is not protected with ATS.

iOS – Static Analysis – App Transport Security

The following is a summarized list of App Transport Security Requirements

- No HTTP connections are allowed
- The X.509 Certificate has a SHA256 fingerprint and must be signed with at least a 2048-bit RSA key or a 256-bit Elliptic-Curve Cryptography (ECC) key.
- Transport Layer Security (TLS) version must be 1.2 or above and must support Perfect Forward Secrecy (PFS) through Elliptic Curve Diffie-Hellman Ephemeral (ECDHE) key exchange and AES-128 or AES-256 symmetric ciphers.
- Strong Ciphers need to be used (e.g. TLS_ECDHE_ECDSA_WITH_AES_256_GCM_SHA384)

https://developer.apple.com/library/content/documentation/General/Reference/InfoPlistKeyReference/Articles/CocoaKeys.html

iOS – Static Analysis – App Transport Security

Your first encounter with HTTPS server trust evaluation is likely to be an error like the following:

```
Domain=NSURLErrorDomain Code=-1202 "The certificate for this server is
invalid. You might be connecting to a server that is pretending to be
“example.com” which could put your confidential information at risk."
UserInfo=0x14a730 {NSErrorFailingURLStringKey=https://example.com/,
NSLocalizedRecoverySuggestion=Would you like to connect to the server anyway?,
NSErrorFailingURLKey=https://example.com/, NSLocalizedDescription=The
certificate for this server is invalid. You might be connecting to a server
that is pretending to be “example.com” which could put your confidential
information at risk., NSUnderlyingError=0x14a6c0 "The certificate for this
server is invalid. You might be connecting to a server that is pretending to
be “example.com” which could put your confidential information at risk.",
NSURLErrorFailingURLPeerTrustErrorKey=<SecTrustRef: 0x14ec00>}
```

In this case error -1202 in the `NSURLErrorDomain` domain is `NSURLErrorServerCertificateUntrusted`, which means that server trust evaluation has failed. You might also receive a variety of other errors; Appendix A: Common Server Trust Evaluation Errors lists the most common ones.

If you receive one of these errors and search the 'net for help, you might find advice like this:

```
To get around this problem simply disable the certificate checks.
```

Or, worse yet, like this:

```
To get around this problem disable the certificate checks by calling +
[NSURLRequest setAllowsAnyHTTPSCertificate:forHost:].
```

Source: https://developer.apple.com/library/archive/technotes/tn2232/_index.html

iOS – Static Analysis – App Transport Security

App Transport Security (ATS) can be “relaxed” with the following settings:

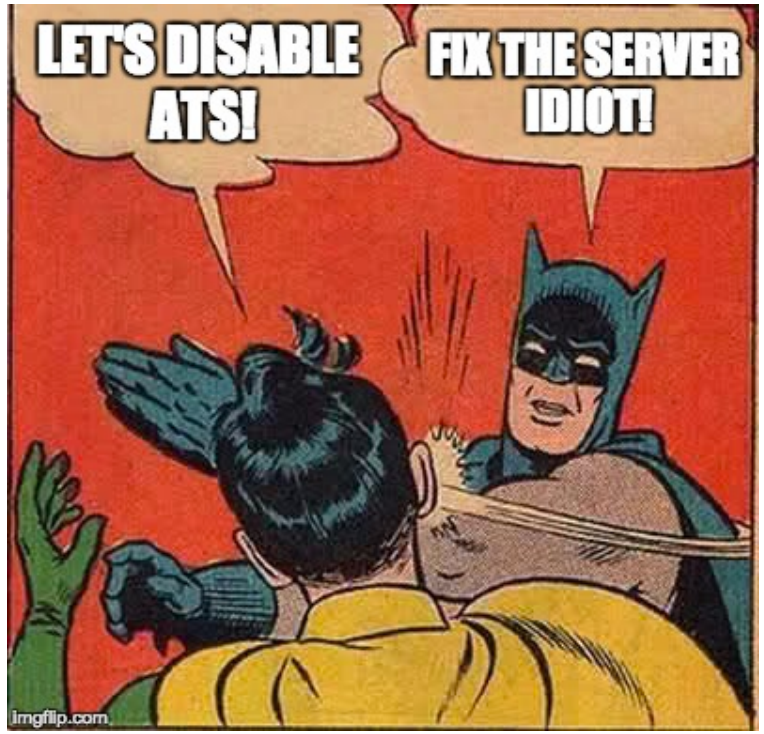
- NSAllowsArbitraryLoads
- NSAllowsArbitraryLoadsForMedia
- NSAllowsArbitraryLoadsInWebContent
- NSExceptionAllowsInsecureHTTPLoads
- NSExceptionMinimumTLSVersion

Not sure if you break your App when using ATS? Share this with the developers, the endpoint might be more mature than they think (only working on macOS):

```
$ /usr/bin/nscurl --ats-diagnostics --verbose https://example.com
```

A Guide to ATS for Security Analysts: https://goo.gl/X8iKix

iOS – Static Analysis – App Transport Security



Apple Review is also getting more restrictive for Apps that disable ATS!

Important

You must supply a justification during App Store review if you set the key's value to YES, as described in Provide Justification for Exceptions. Use this key with caution because it significantly reduces the security of your app. In most cases, it's better to upgrade your servers to meet the requirements imposed by ATS, or at least to use a narrower exception.

https://developer.apple.com/documentation/bundleresources/information_property_list/nsapptransportsecurity/nsallowsarbitraryloads

Static Analysis of iOS App - MobSF

What else is noteworthy?

- File Analysis
- View Info.plist
- Overview of Libraries identified
- IPA Binary Analysis
- Available as Docker Image
- REST APIs available (Integration in CI/CD possible)

MobSF is an easy and cheap (Open Source) way at the moment to get at lest some of the low hanging fruits without the need of having the source code.

Github Project Page: https://github.com/MobSF/Mobile-Security-Framework-MobSF

iOS – Static Analysis

SAST Enterprise Tools for iOS Apps

There are several tools available that execute SAST for iOS Apps.
Usually you face the following problems:

- Usually no Support for latest Swift versions
- Usually the test cases applied are quite limited
- False positive rate is quite high
- Depending on the SAST tool you would need to build the app (e.g. when using Coverity, Fortify), meaning you need the source code.

Alternatives are tools that execute the app dynamically (DAST scan, like NowSecure), but this has other pre-requisites that need to be fulfilled.

Stateful and Stateless Authentication

Stateful and Stateless Authentication

Stateful vs. Stateless Authentication

- HTTP doesn’t hold a user state; therefore a mechanism need to be available to allocate a request with a user.
- Can be done in two ways:

    - **Stateful authentication**: A unique session id is generated when the user logs in. In subsequent requests, this session ID serves as a reference to the user details stored on the server.

    - **Stateless authentication**: All user-identifying information is stored in a client-side token. The token can be passed to any server or micro service, eliminating the need to maintain session state on the server.

Stateful and Stateless Authentication

Stateful authentication

- Web applications commonly use stateful authentication with a random session ID that is stored in a client-side cookie.

```
Vary: Accept-Encoding
Set-Cookie: JSESSIONID=9GNa_wlpepSR8hIJa1xdHcZi_jEfTHyqV6kZ4aVZ.lxmbkusg01; path=/vtap;HttpOnly;secure
X-Permitted-Cross-Domain-Policies: none
```

Stateful and Stateless Authentication

Stateful authentication

If the mobile app is using stateful authentication, several test cases from the OWASP Testing Guide v4 are applicable and we won't cover this in this training.

Authentication:
https://wiki.owasp.org/index.php/Testing_for_authentication

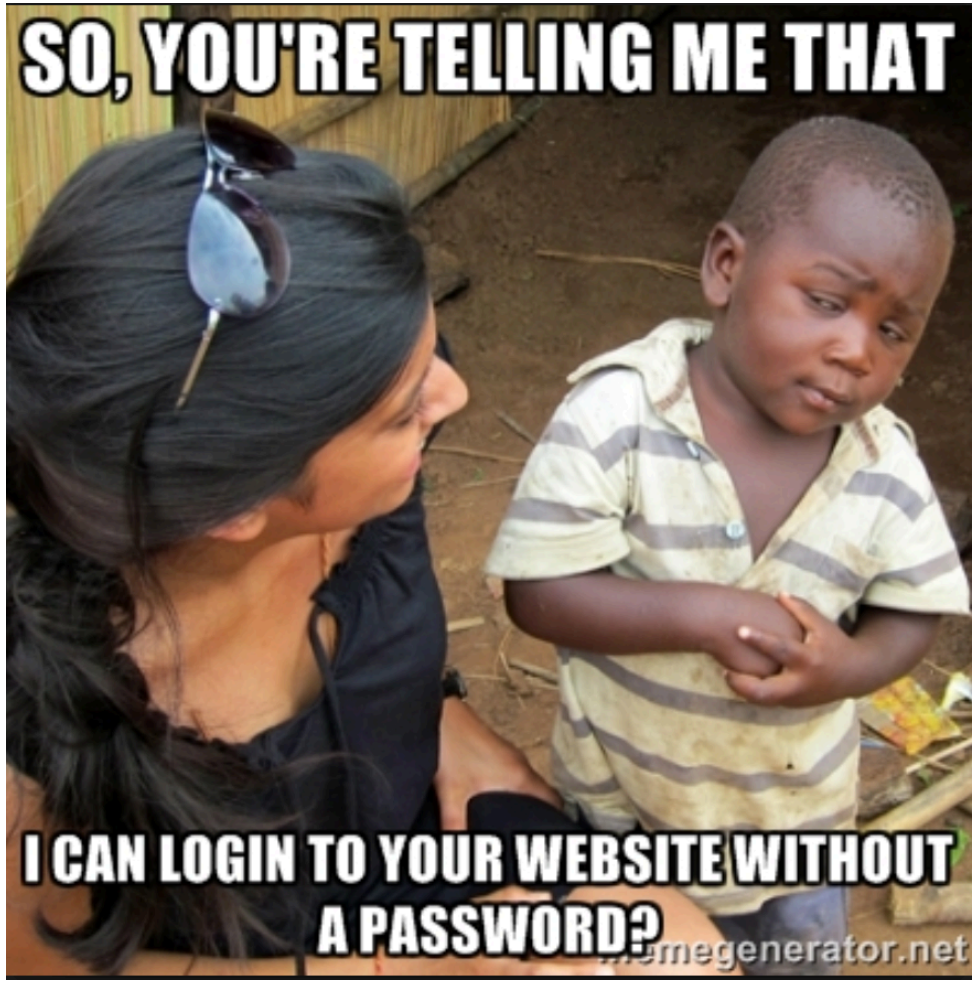
Session Management:
https://wiki.owasp.org/index.php/Testing_for_Session_Management

Stateful and Stateless Authentication

Stateless authentication

Stateless token-based approaches are becoming popular for a variety of reasons:

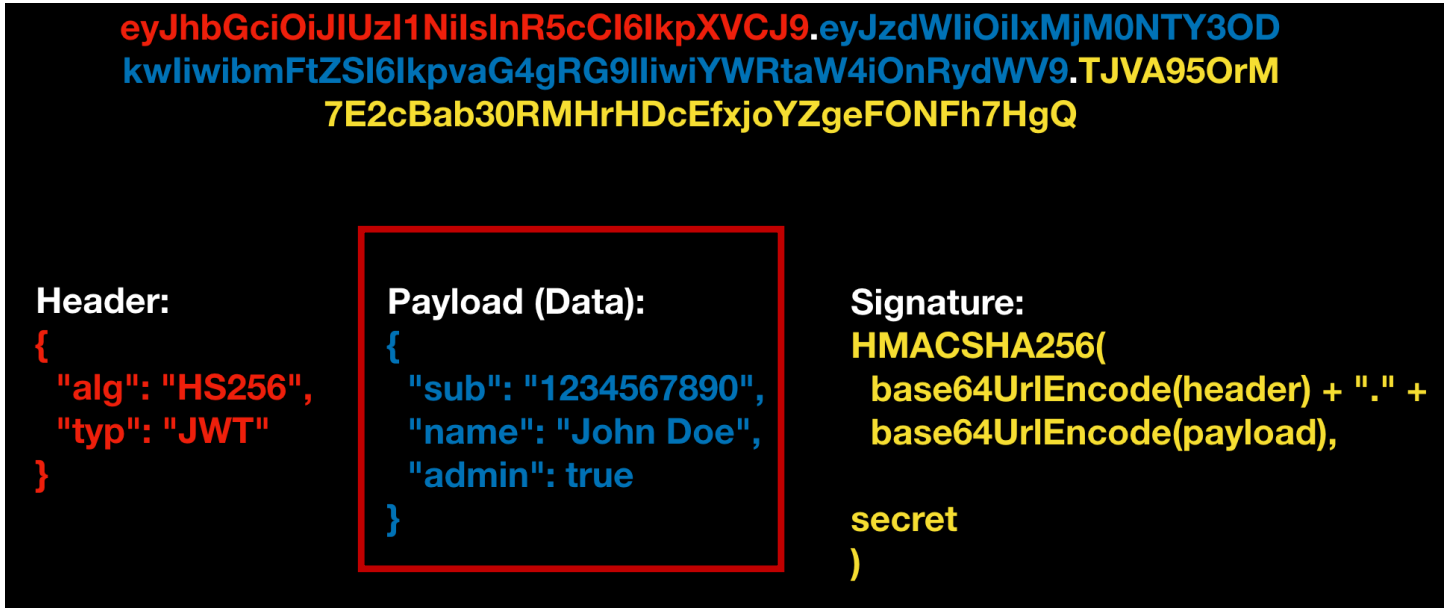
- They improve scalability and performance by eliminating the need to store session state on the server.
- Tokens enable developers to decouple authentication from the app.
- After first authentication with credentials, users don't need to re-enter their passwords for days or weeks



Stateful and Stateless Authentication

What is JSON Web Token (JWT)?

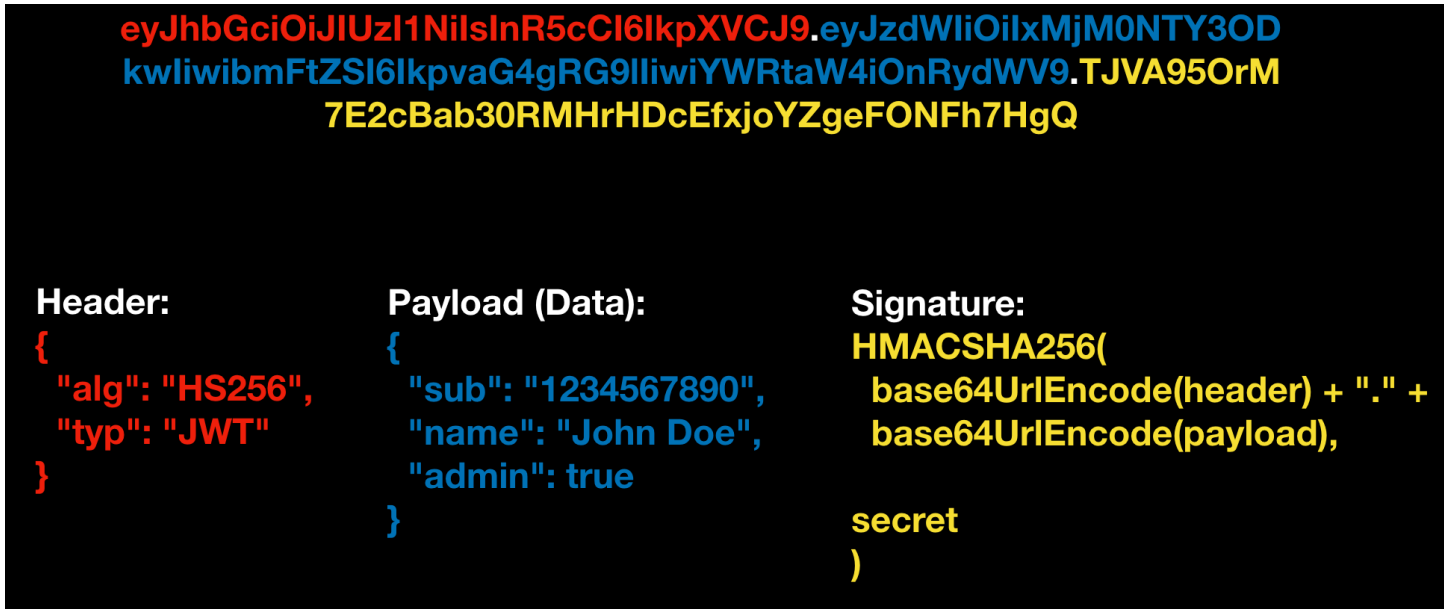
- Open Standard for creating access tokens (RFC 7519), pronounced /dʒɒt/
- Access token contains “Claims” that are part of the “Payload”
- For example, a server could generate a token that has the claim "logged in as admin" and provide that to a client.
- The client could then use that token to prove that it is logged in as admin.



Stateful and Stateless Authentication

What is JSON Web Token (JWT)?

- The header typically consists of two parts: the token type, which is JWT, and the hashing algorithm being used to compute the signature.
- The signature is created by applying the algorithm specified in the JWT header to the encoded header, encoded payload, and a secret value (Private Key).



Stateful and Stateless Authentication

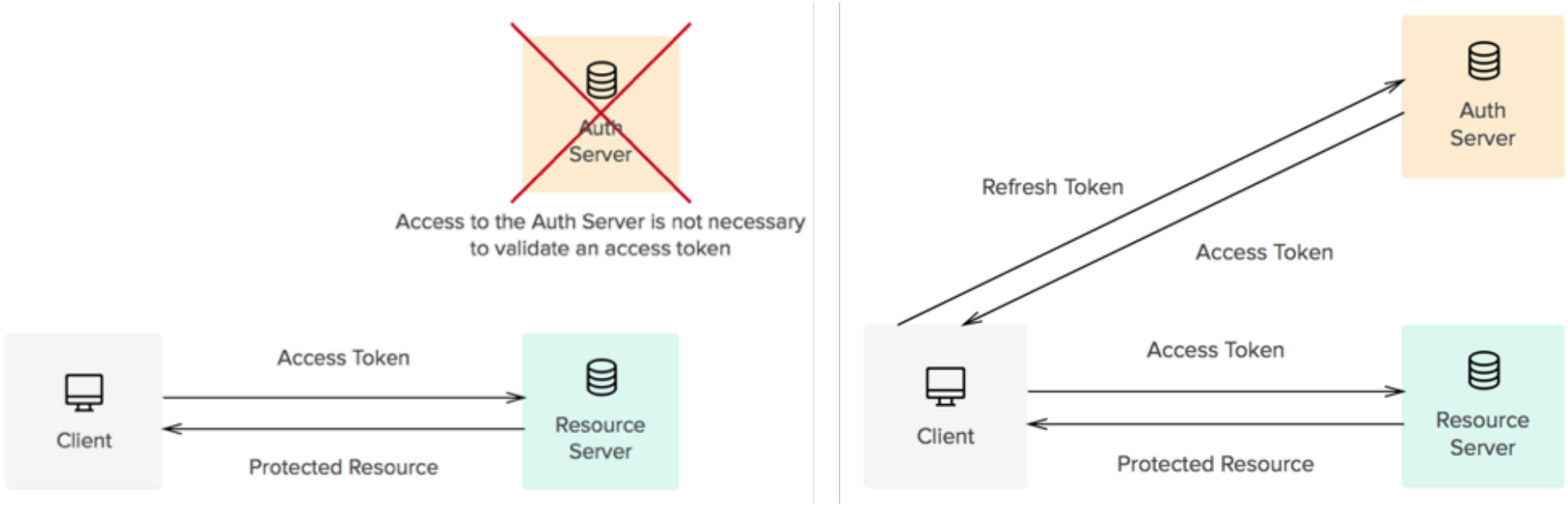
What is JSON Web Token (JWT)?

- Tokens are generated on the server and signed with a private key (JSON Web Signature - JWS)
- Server can validate that the token is legitimate due to the signature
- If JWT contains sensitive information it can be encrypted, which is JWE (JSON Web Encryption)

Stateful and Stateless Authentication

Access Token (short lived) and Refresh Token (long lived)

- After logging into an app you don't get a session ID you get an access token and refresh token instead
- Access tokens carry the necessary information to access a resource directly.
- Refresh tokens carry the information necessary to get a new access token.



https://auth0.com/blog/refresh-tokens-what-are-they-and-when-to-use-them/

Stateful and Stateless Authentication

Access Token (short lived):

- Usually in the range of minutes or hours

Refresh Token (long lived)

- Usually in the range of weeks or months

Once signed, a stateless authentication token is valid forever unless the signing key changes!

=> Therefore both should have the expiry attribute that defines when a token expires.

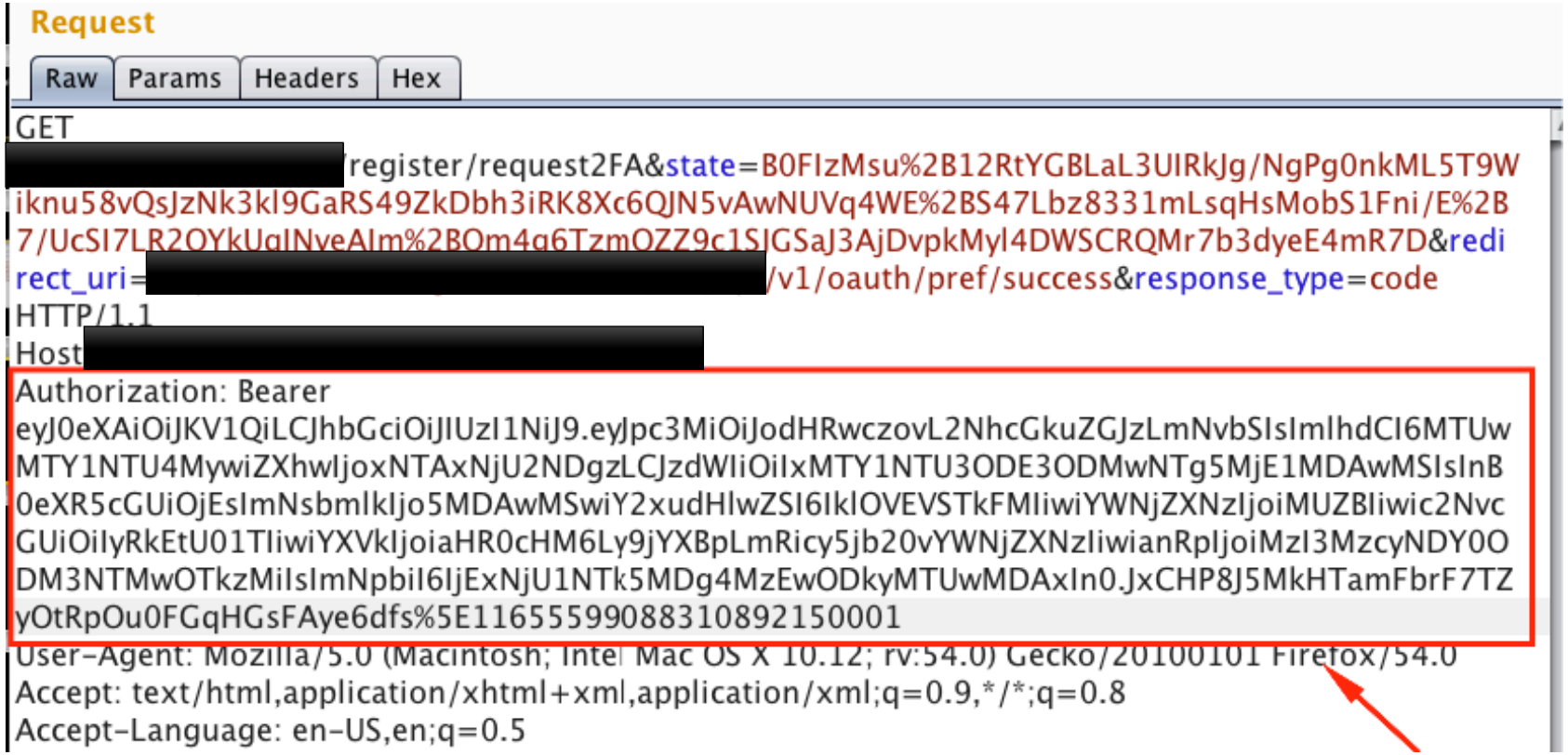
Stateful and Stateless Authentication

Wrong usage in projects – Example 1

JWT should be sent as part of the Authorization Header, not in the URL!

Stateful and Stateless Authentication

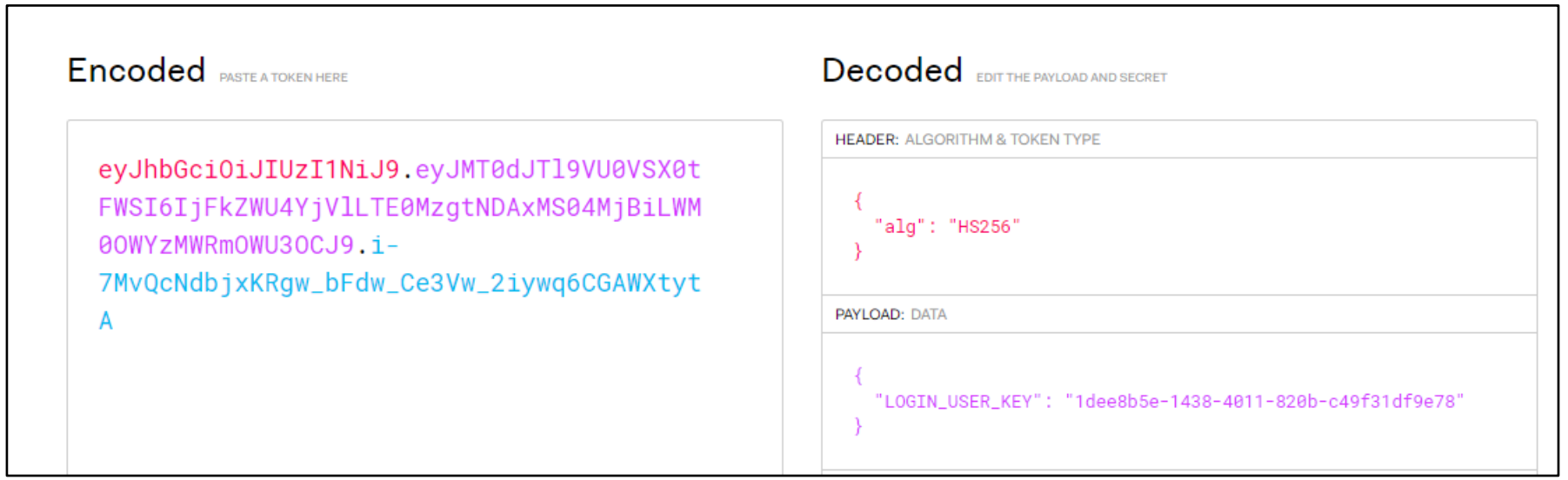
Correct Usage – Example 1



Authorization Header by using the Bearer schema is best practice

Stateful and Stateless Authentication

Wrong usage in projects – Example 2



There is no expiry claim. So this token is valid until the private key is changed.

Stateful and Stateless Authentication

Wrong usage in projects – Example 3

Encoded PASTE A TOKEN HERE

```
eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJKU0VTU0lPTklEIjoiMjNBNTJCRUIxNDAyQ0Y2QzU5M0U4OTExQTNEODk5RDIiLCJuYW1lIjoiSm9obiBEb2UifQ.0bBO-BxitNrLlozQMlJDFHHsXvoGmdffx7g-CeEhqfY
```

Decoded EDIT THE PAYLOAD AND SECRET

HEADER: ALGORITHM & TOKEN TYPE

```
{
  "alg": "HS256",
  "typ": "JWT"
}
```

PAYLOAD: DATA

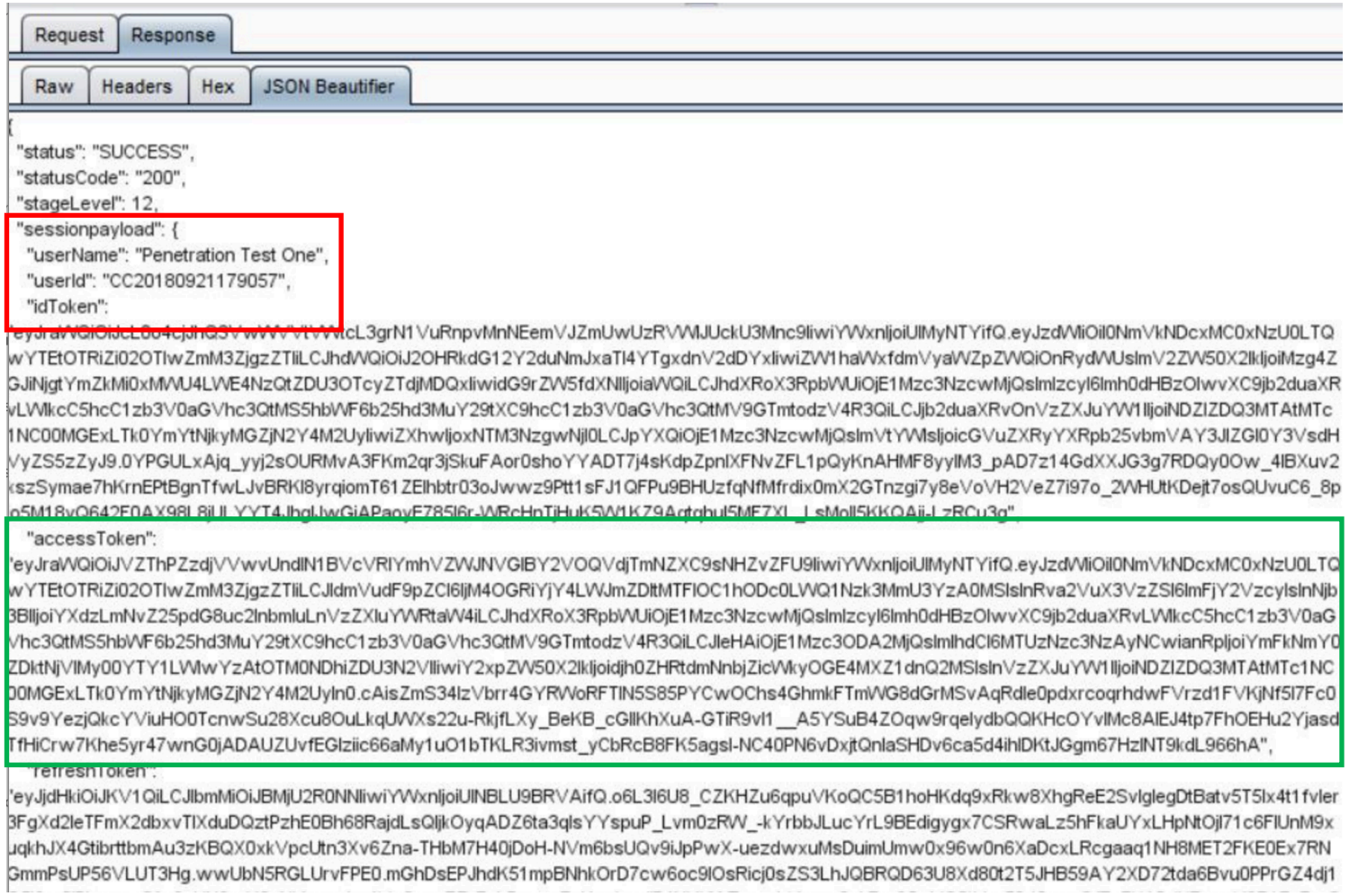
```
{
  "JSESSIONID": "23A52BEB1402CF6C593E8911A3D899D2",
  "name": "John Doe"
}
```

The Payload should be used for the Claims. This example contains a Session ID and mixes Stateful with Stateless to get the worst out of both worlds…

- When logging out of the session the JWT remains valid forever, likely to make errors in the business logic.
- After verification of the JWT a request to the database or redis/memcache is still done to verify the session (additional overhead)

Stateful and Stateless Authentication

Wrong usage in projects – Example 4



After authentication an access token (JWT) is sent back to the client that contains all the claims.

But the claims in the JWT are not verified on the server side. Every request will contain a userID parameter so you just need to change the userID to any user you want to take over…

Stateful and Stateless Authentication

Tool support when testing JWT

- jwt.io (https://jwt.io)
- **Burp Extension: JSON Web Tokens** (https://github.com/portswigger/json-web-tokens)
- Chrome Extension: JWT Debugger (https://chrome.google.com/webstore/detail/jwt-debugger/ppmmlchacdbknfphdeafcbmklcghghmd?hl=en)
- Cracking the signing key (https://github.com/Sjord/jwtcrack/blob/master/crackjwt.py)

Stateful and Stateless Authentication

What should be checked during a penetration test when JWT is used?

- Look for sensitive information in the JWT (Information Disclosure)
- Try if you can crack the signing key
- Verify where and how the JWT is stored on the device
- Verify the Token Expiration

The usual "Insecure Logout" finding (Session is not destroyed on server side after logout) is not a finding when JWT is used, if an expiry time is defined!

See also:
https://www.sjoerdlangkemper.nl/2016/09/28/attacking-jwt-authentication
https://www.owasp.org/index.php/JSON_Web_Token_(JWT)_Cheat_Sheet_for_Java

Stateful and Stateless Authentication

Now it's your turn:
Crack a JWT in our lab!

Open the file Lab-iOS.pdf:

```
$  evince ~/MSTG-Handson/Lab/Lab-iOS.pdf
```

Section: **Lab – Stateless Authentication**

Please type the commands and do not copy & paste!

1. You will remember them easier
2. Some PDF readers do not copy properly and miss whitespaces etc.

Stateful and Stateless Authentication

Known vulnerabilities in outdated libraries (was found in 2015) :

- In systems using HMAC signatures, verificationKey will be the server's secret signing key (since HMAC uses the same key for signing and verifying).
- In systems using an asymmetric algorithm, verificationKey will be the public key against which the token should be verified.

If a server is expecting a token signed with RSA, but receives a token signed with HMAC, **it will think the public key is an HMAC secret key**.

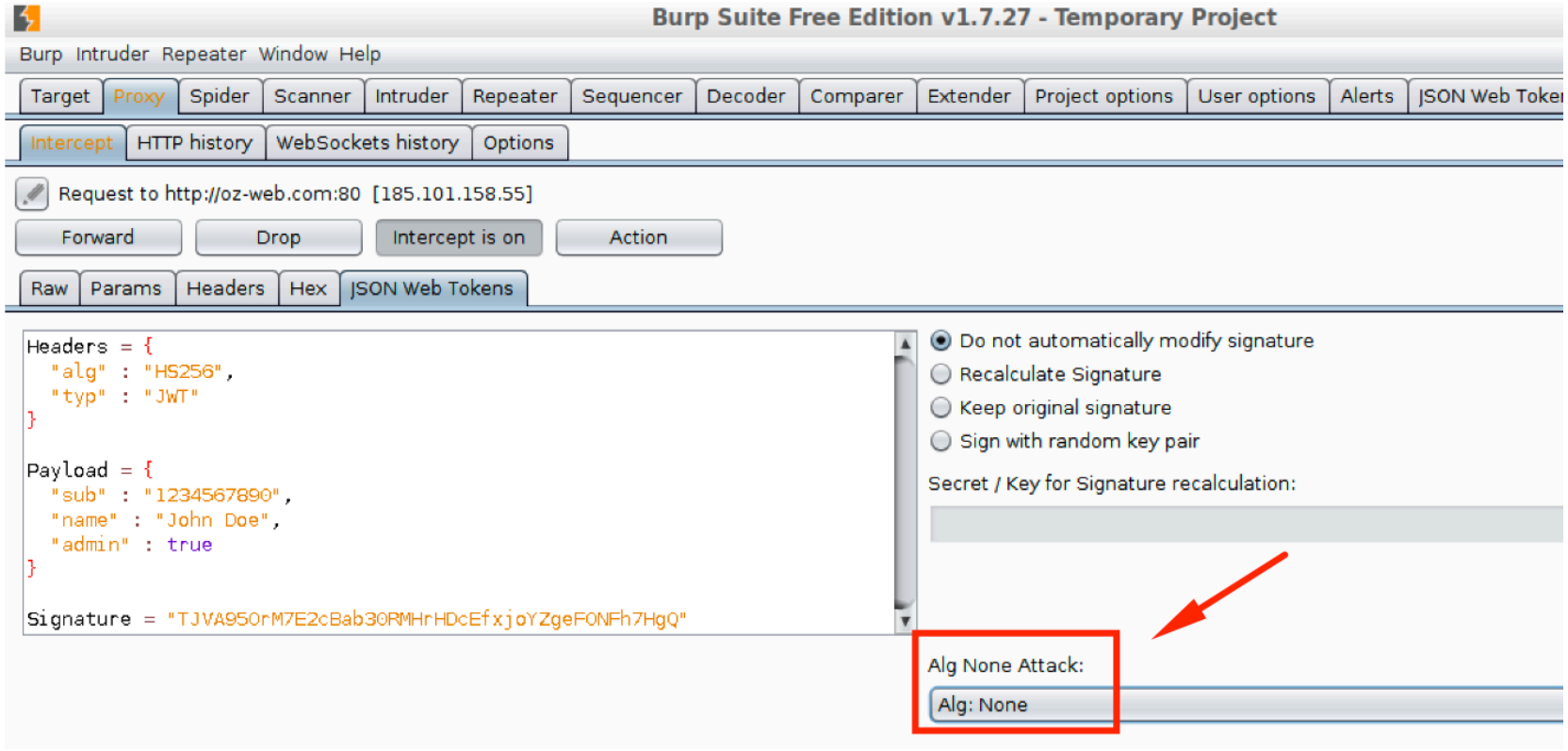
⇒ Everybody has access to the public key and can use this to forge a token that the server will accept.

For more details:
https://auth0.com/blog/critical-vulnerabilities-in-json-web-token-libraries

Stateful and Stateless Authentication

Known vulnerabilities in outdated libraries (was found in 2015) :

Usage of "none” as algorithm in the Header section of the JWT disables verification of the signing



For more details:

https://auth0.com/blog/critical-vulnerabilities-in-json-web-token-libraries

Stateful and Stateless Authentication

Lessons learned:

- Private key must have a high entropy and must be stored securely on server side
- For most (small/medium) projects ”old school” session management might work better and there is less benefit in using JWT
- In scenarios of decentralized authentication (like SSO / Federated Login / Microservices) JWT is useful, as the services can easily verify the signature with the public key without querying an authentication server

See also:
https://www.owasp.org/index.php/JSON_Web_Token_(JWT)_Cheat_Sheet_for_Java
https://x-team.com/blog/my-experience-with-json-web-tokens/
https://scotch.io/bar-talk/why-jwts-suck-as-session-tokens

Reverse Engineering on iOS

iOS App Dumping

- Dumping an IPA can be challenging.
- IPAs are protected by FairPlay DRM, and it can only be decrypted on devices associated with your Apple user account.
- Multiple scripts can be used to dump the application such as dumpdecrypted, Clutch, Frida-iOS-dump and so on.
- In hindsight, to dump a DRM-protected IPA on a device, the mentioned scripts would usually calculate the actual protected offset and size from LC_ENCRYPTION_INFO load command and dump the unencrypted app from the running process and patch the LC_ENCRYPTION_INFO as plain.

iOS – Reverse Engineering

	iOS
Dump Application for RE	*frida-ios-dump* *(jailbroken*)*
Protection	Binary encrypted via Fairplay (if installed via Apple Store)
Static analysis	Disassemble
Dynamic analysis	Frida

iOS – Reverse Engineering

Once you have the IPA you might face the next challenges:

- **Jailbreak detection**
- Anti-debugging
- Detection of Frida
- …

Jailbreak Detection

What is Jailbreak Detection?
The goal of jailbreak detection is to make running the app on a jailbroken device more difficult, which in turn blocks some of the tools and techniques reverse engineers like to use.

What do you think are common Jailbreak Detection Methods?

Jailbreak Detection

What is Jailbreak Detection?

The goal of jailbreak detection is to make running the app on a jailbroken device more difficult, which in turn blocks some of the tools and techniques reverse engineers like to use.

Common Jailbreak Detection Methods

- File existence checks (e.g. Cydia.app, bash, sshd)
- File permission checks (e.g. writing to /private)
- Checking Protocol Handlers (cydia://)
- Checking running processes
- Calling System APIs

See also: https://mobile-security.gitbook.io/mobile-security-testing-guide/ios-testing-guide/0x06j-testing-resiliency-against-reverse-engineering#jailbreak-detection-mstg-resilience-1

Jailbreak Detection

- Jailbreak detection mechanisms tries to prevent users from using a Jailbroken device.

- Usually two actions by an app, if it detects a jailbroken device:
  - Informs the user about the insecure environment (decision is up to the user to use it or not)
  - Termination of the app

- Especially popular in Mobile Device Management (MDM) solutions.

Jailbreak Detection

Let's do our next Lab!

Open the file Lab-iOS.pdf:

```
$ evince ~/MSTG-Handson/Lab/Lab-iOS.pdf
```

Section: Lab - Bypass Jailbreak Detection

Please type the commands and do not copy & paste!

1. You will remember them easier
2. Some PDF readers do not copy properly and miss whitespaces etc.

Jailbreak Detection

How was the Jailbreak detection bypassed?

```
➜ Frida-Scripts git:(master) ✗ frida -U -l bypass-jb-dviav2-generic.js Gadget
    ____
   / _  |   Frida 12.6.5 - A world-class dynamic instrumentation toolkit
  | (_| |
   > _  |   Commands:
  /_/ |_|       help      -> Displays the help system
  . . . .       object?   -> Display information about 'object'
  . . . .       exit/quit -> Exit
  . . . .
  . . . .   More info at http://www.frida.re/docs/home/

message: {'type': 'send', 'payload': 'Jailbreak Detection enabled'} data: None
message: {'type': 'send', 'payload': 'Found our target class : UIApplication'} data: None
message: {'type': 'send', 'payload': 'Found our target class : NSFileManager'} data: None
message: {'type': 'send', 'payload': 'Found our target class : NSString'} data: None
[iPhone::Gadget]-> message: {'type': 'send', 'payload': 'Jailbreak detection => Trying to read path: /Applications/Cydia.app'} data: None
message: {'type': 'send', 'payload': 'Jailbreak detection bypassed!'} data: None
message: {'type': 'send', 'payload': 'Jailbreak detection => Trying to read path: /Library/MobileSubstrate/MobileSubstrate.dylib'} data: None
message: {'type': 'send', 'payload': 'Jailbreak detection => Trying to read path: /Library/MobileSubstrate/MobileSubstrate.dylib'} data: None
message: {'type': 'send', 'payload': 'Jailbreak detection bypassed!'} data: None
message: {'type': 'send', 'payload': 'Jailbreak detection => Trying to read path: /bin/bash'} data: None
message: {'type': 'send', 'payload': 'Jailbreak detection bypassed!'} data: None
message: {'type': 'send', 'payload': 'Jailbreak detection => Trying to read path: /usr/sbin/sshd'} data: None
message: {'type': 'send', 'payload': 'Jailbreak detection bypassed!'} data: None
message: {'type': 'send', 'payload': 'Jailbreak detection => Trying to read path: /etc/apt'} data: None
message: {'type': 'send', 'payload': 'Jailbreak detection bypassed!'} data: None
message: {'type': 'send', 'payload': 'Jailbreak detection => Trying to write path: /private/jailbreak.txt'} data: None
message: {'type': 'send', 'payload': 'Jailbreak detection bypassed!'} data: None
message: {'type': 'send', 'payload': 'URL : cydia://package/com.example.package'} data: None
message: {'type': 'send', 'payload': 'Jailbreak detection => Trying to use Cydia URL Schema: cydia://package/com.example.package'} data: None
message: {'type': 'send', 'payload': 'Jailbreak detection bypassed!'} data: None
[iPhone::Gadget]->
```

Jailbreak Detection

By hooking into NSFileManager. Every time a file is detected, the Frida script is replacing the return value from true to false.

```
if(ObjC.available) {
    send("Jailbreak Detection enabled");
    for(var className in ObjC.classes) {
        if (ObjC.classes.hasOwnProperty(className)) {
            //Jailbreak detection via accessing special files
            if(className == "NSFileManager") {
                send("Found our target class : " + className);

                var hook = ObjC.classes.NSFileManager["- fileExistsAtPath:"];
                Interceptor.attach(hook.implementation, {
                onEnter: function (args) {
                    var  path = ObjC.Object(args[2]).toString(); // NSString

                    this.jailbreakCall = false;
                    var i = jailbreakPaths.length;
                    while (i--) {
                        if (jailbreakPaths[i] == path) {
                                send("Jailbreak detection => Trying to read path: "+path);
                                this.jailbreakCall = true;
                        }
                    }
                },
                    onLeave: function (retval) {
                        if(this.jailbreakCall) {
                            retval.replace(0x00);
                            send("Jailbreak detection bypassed!");
                        }
                    }
                });
            }
```

Frida script is from https://github.com/as0ler/frida-scripts/blob/master/hooks/_jailbreak_detection.disabled

Jailbreak Detection

How can we make such attacks harder?

Jailbreak Detection

How can we make such attacks harder?

- Checking if app was repackaged (Objection/Frida-Gadget)
- Checking if a debugger is used
- Checking if Reverse Engineering tools are used (Frida)
- Checking if device is jailbroken (multiple checks)
- Obfuscation of source code
- etc.

Jailbreak Detection

How can we make such attacks harder?

- Checking if app was repackaged (Objection/Frida-Gadget)
- Checking if a debugger is used
- Checking if Reverse Engineering tools are used (Frida)
- Checking if device is jailbroken (multiple checks)
- Obfuscation of source code
- etc.

Makes the effort more time consuming and can be used as part of a defence in depth strategy. Reverse Engineering is still possible and will always be a cat and mouse game!

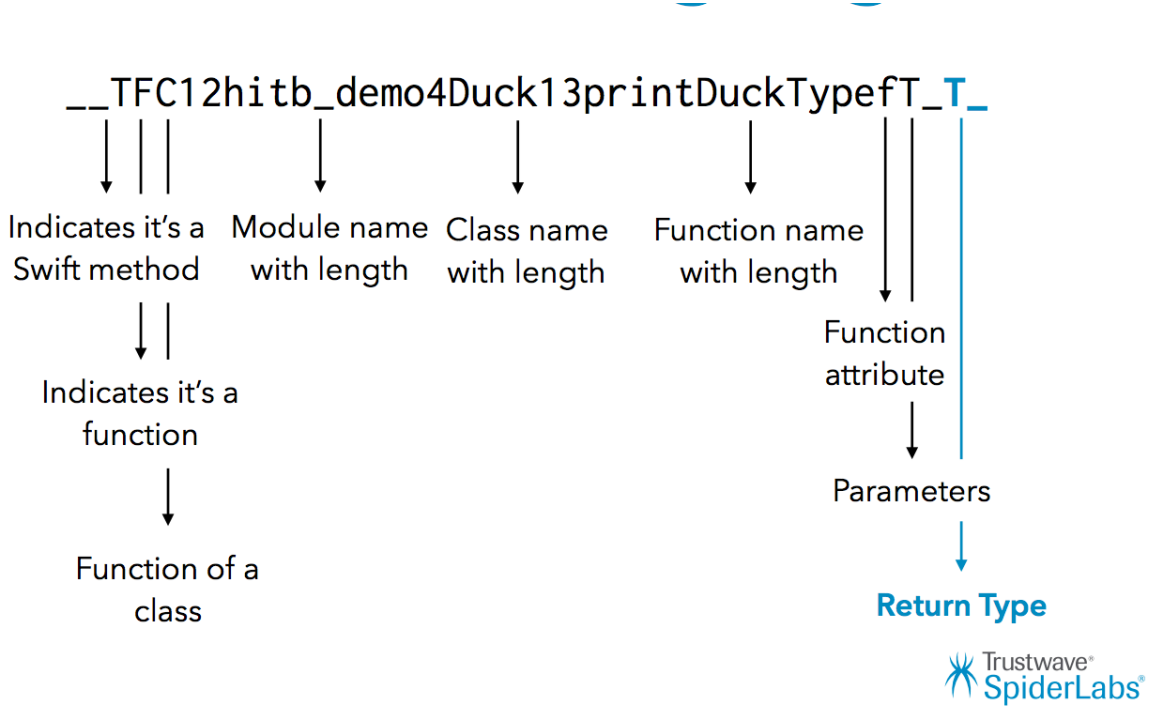
iOS Reverse Engineering – Binary Patching

Jailbreak detection Bypass through Binary Patching

- Disassembler such as Ghidra or Hopper can be used to modify executable (Mach-O)
- Bypass with Binary Patching of IPA and make true (0x0) to false (0x1) or replace logic with NOP!
- Repackage the IPA with the new Mach-O binary and the check is reversed (check will NOT trigger jailbreak detection on jailbroken device)

iOS Reverse Engineering – Binary Patching

Remember this:



Trustwave SpiderLabs

iOS Reverse Engineering – Binary Patching

Swift_demangler.py for Ghidra -
https://github.com/ghidraninja/ghidra_scripts#swift_demanglerpy


```
auVar3 = _$SSS21_builtinStringLiteral17utf8CodeUnitCount7isASCIISSBp_BwBi1_tcfC
                   ("Hello, world!",0xd,1);
*puVar2 = SUB168(auVar3,0);
puVar2[1] = SUB168(auVar3 >> 0x40,0);
auVar3 = _$Ss5print_9separator10terminatoryypd_S2StFfA0_();
auVar4 = _$Ss5print_9separator10terminatoryypd_S2StFfA1_();
_$Ss5print_9separator10terminatoryypd_S2StF
          (uVar1,SUB168(auVar3,0),SUB168(auVar3 >> 0x40,0),SUB168(auVar4,0),
           SUB168(auVar4 >> 0x40,0));
_swift_bridgeObjectRelease(SUB168(auVar4,0));
_swift_bridgeObjectRelease(SUB168(auVar3,0));
_swift_bridgeObjectRelease(uVar1);
_$S13swift_crackme6foobar3fooySi_tF(5);
_$S13swift_crackme7foobar2yyF();
```

swift_demangler.py

```
16  auVar3 = Swift.String.init("Hello, world!",0xd,1);
17  *puVar2 = SUB168(auVar3,0);
18  puVar2[1] = SUB168(auVar3 >> 0x40,0);
19  auVar3 = default_argument_1_of_print();
    auVar4 = default_argument_2_of_print();
    Swift.print(uVar1,SUB168(auVar3,0),SUB168(auVar3 >> 0x40,0),SUB168(auVar4,0),
                SUB168(auVar4 >> 0x40,0));
23  _swift_bridgeObjectRelease(SUB168(auVar4,0));
    _swift_bridgeObjectRelease(SUB168(auVar3,0));
    _swift_bridgeObjectRelease(uVar1);
26  swift_crackme.foobar(5);
27  swift_crackme.foobar2();
```

iOS Reverse Engineering – Binary Patching

```
                        loc_10000770c:
000000010000770c            movz        x25, #0x12                          ; CODE XREF=sub_10000769c+68, sub_10000769c+96
0000000100007710            movk        x25, #0xd000, lsl #48
0000000100007714            mov         x0, x22
0000000100007718            orr         w1, wzr, #0x40
000000010000771c            orr         w2, wzr, #0x7
0000000100007720            bl          imp___stubs__swift_allocObject      ; swift_allocObject
0000000100007724            mov         x23, x0
0000000100007728            nop
000000010000772c            ldr         q0, =0x1
0000000100007730            str         q0, [x0, #0x10]
0000000100007734            tbz         w24, 0x0, loc_100007808

0000000100007738            nop
000000010000773c            ldr         x8, #_$sSSN_100010010               ; _$sSSN
0000000100007740            str         x8, [x23, #0x38]
0000000100007744            add         x8, x25, #0xb
0000000100007748            adr         x9, #0x10000d0c0                    ; "This device is not jailbroken"
000000010000774c            nop
0000000100007750            sub         x9, x9, #0x20
0000000100007754            orr         x9, x9, #0x8000000000000000
```

Videos to demonstrate this can be found here: http://bit.ly/3bEnlkL

iOS Reverse Engineering – Binary Patching

Homework:

- Get your own apple developer account (free account is sufficient)
- Add account to Xcode
- Patch the app with Ghidra
- Repackage and resign the app with applesign (https://github.com/nowsecure/node-applesign)

```
$  bin/applesign.js -c -a -w -m embedded.mobileprovision <IPA>
```

Follow the instructions:
https://github.com/sensepost/objection/wiki/Patching-iOS-Applications

iOS Security Suite

If implemented in Objective-C, such checks can usually be bypassed through existing tools (objection) and Frida scripts.

An alternative is implementing it with iOS Security Suite (this library is implemented in Swift), this cannot easily be hooked and bypassed with Frida:

https://github.com/securing/IOSSecuritySuite

iOS Security Suite

SwiftSecurity

This is a PoC implementing the IOS Security Suite https://github.com/securing/IOSSecuritySuite

The following checks can be run:

- Jailbreak Detection
- Debugger Detection
- Emulator Detection
- Reverse Engineering Tools Detection

And the Deny Debugger function.

A precompiled IPA can be found in the root directory SwiftSecurity.ipa. You would need to resign it before running on your device.

Current look of the app:

https://github.com/sushi2k/SwiftSecurity

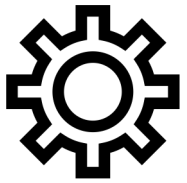
iOS Security Suite

Bypassing debugger check and Frida check in iOS Security Suite.

Demo

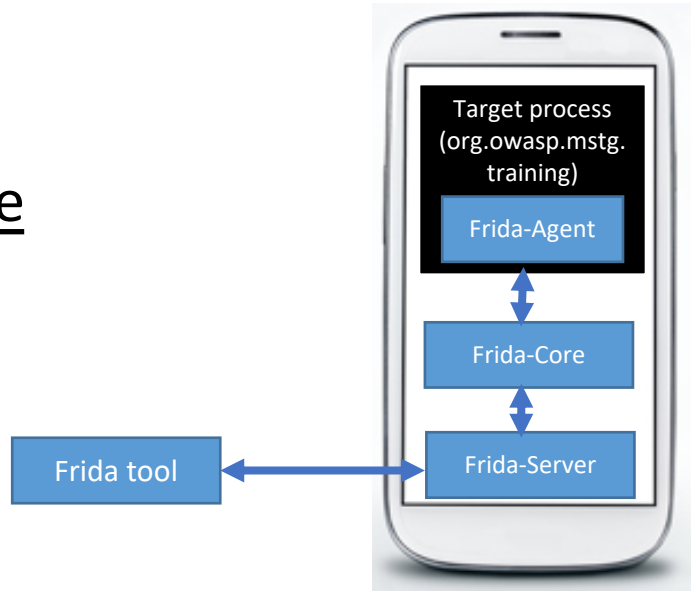
URL to download Video:
http://bit.ly/3bEnlkL

Frida - Different Modes of Operation



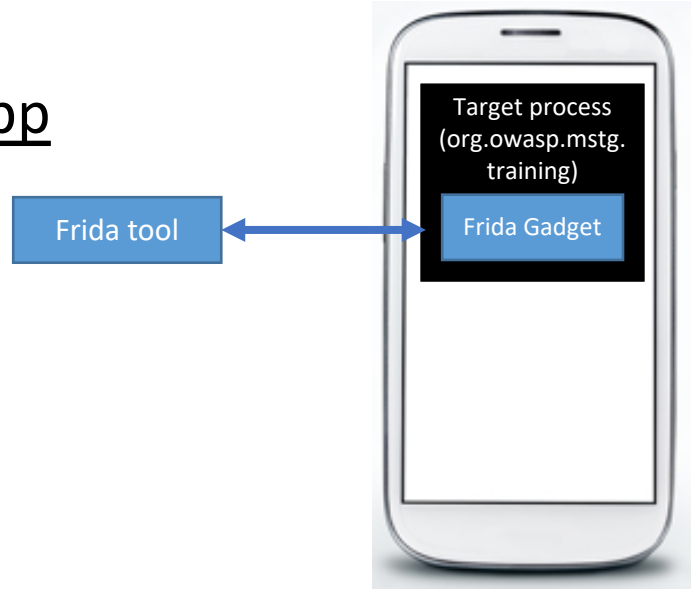
Injected into a process by running the Frida server on the device

- Only possible on jailbroken devices
- Frida handles the injection



Embedded as shared library (frida-gadget.so) into the mobile app

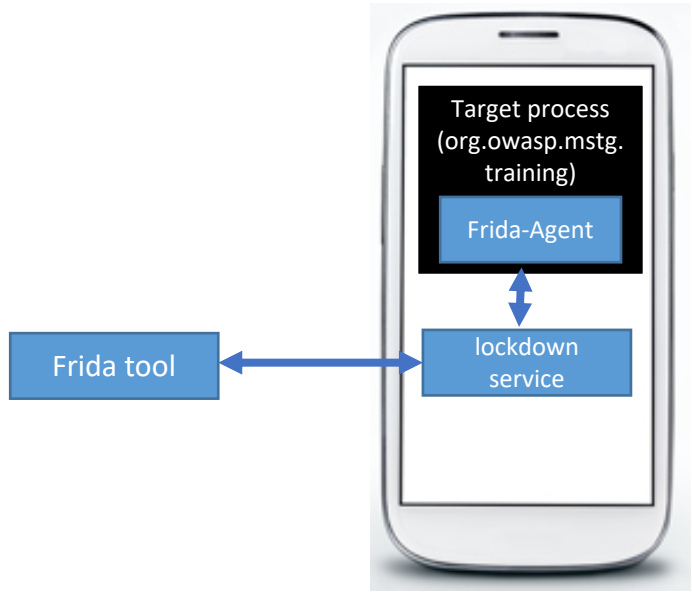
- Working on non-jailbroken devices
- Repackaging and resigning required



Frida - Different Modes of Operation

Injected into a process by using lockdown service

- Possible on non-jailbroken devices
- Frida handles the injection and is using the lockdown service on iOS



There is not much documentation from Apple about the lockdown service on iOS, but this article explains it in more depth: https://medium.com/@jon.gabilondo.angulo_7635/understanding-usbmux-and-the-ios-lockdown-service-7f2a1dfd07ae

Frida - Different Modes of Operation

This is a new Frida feature since Frida 12.7.12:

Changes in 12.7.12

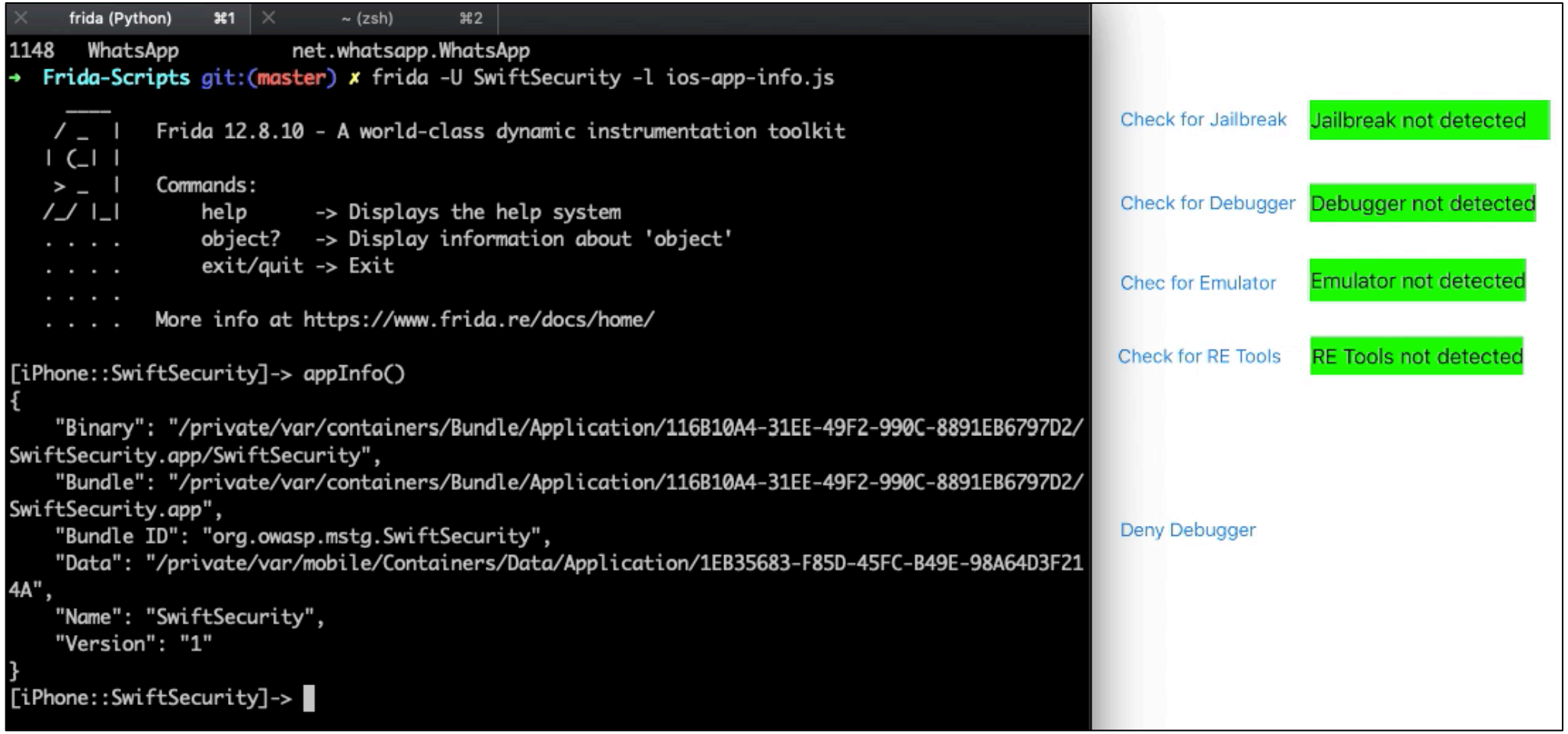
- Full-featured iOS lockdown integration and unified devices, so Frida-based tools don't need to worry as much about jailed vs jailbroken. When interacting with a jailed iOS device, Gadget is now injected automatically and there is no need to repackage the app, it only has to be debuggable.

https://frida.re/news/2019/09/18/frida-12-7-released/

Detailed explanation of how to use the new feature:
https://www.nowsecure.com/blog/2020/01/02/how-to-conduct-jailed-testing-with-frida/

Pre-requisites:
- iOS 13
- Frida on macOS with >= 12.7.12

iOS Security Suite



Bypass was already raised by me ☺ So use the latest version of the library

https://github.com/securing/IOSSecuritySuite/issues/8

Comparison

Overview of both methods we learned to bypass client-side security controls, such as Jailbreak Detection:

	Frida	Binary Patching
Deployment	ios-deploy + Connected macOS device	ios-deploy + Connected macOS device
Invasiveness	Modify Target Process	Change Compiled App
Granularity	Method	Instruction
Updates	Instantaneous	Recompilation

Additional Exercises

- You have 30 Minutes
- Work on your own and solve all challenges!

Challenges

App Name	Mission	Hint
DVIA	Try to bypass the Piracy Check	Go through the **Lab - Bypass Piracy Detection** in Lab-iOS.pdf
DVIA	Try to get your re-signing setup up and running with objection on your macOS and re-sign an app with objection.	This app need to be done on your own iOS device! Follow the steps to configure your macOS environment: https://github.com/sensepost/objection/wiki/Patching-iOS-Applications Steps: - Re-sign the app with your new development profile - Re-install DVIA (delete it from the iOS device and install and run your DVIA app with ios-deploy). - Connect objection to the DVIA app - Bypass Certificate Pinning with objection in the challenge “Transport Layer Security”

Team Exercise

Let’s start ☺

More apps to hack!

Hacking Playground

Collection of iOS (and Android) Apps that can be attacked for learning purpose. Your homework ☺

They have purposely vulnerabilities that you can exploit:

https://github.com/OWASP/MSTG-Hacking-Playground

iOS Reverse Engineering

Practical Reverse Engineering Challenges for iOS! More homework ☺



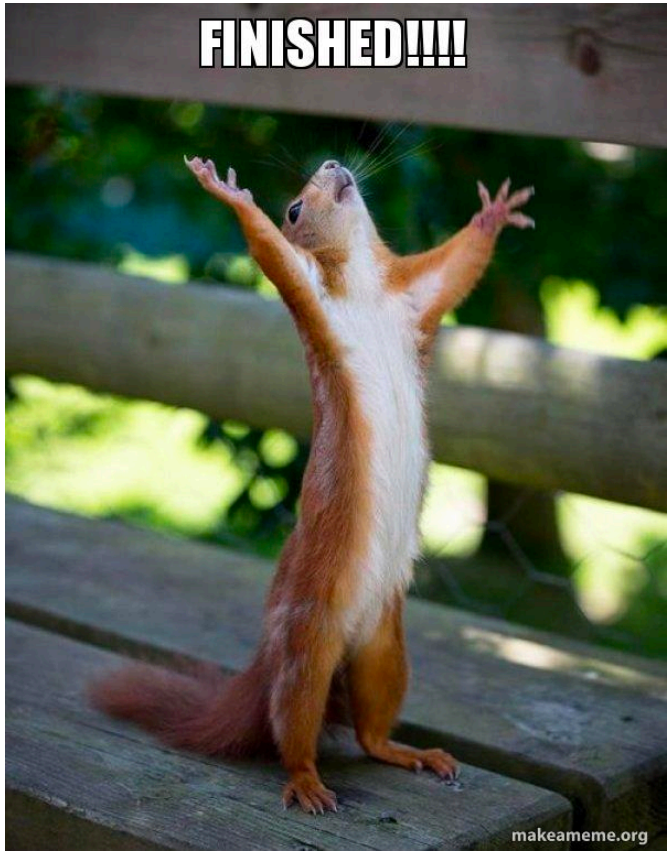
« UnCrackable Mobile Apps »
https://github.com/OWASP/owasp-mstg/tree/master/Crackmes

More apps that can be used to train your new skills:
https://mobile-security.gitbook.io/mobile-security-testing-guide/appendix/0x08-testing-tools#android

Feedback Form!

https://forms.gle/dVH6h8yKTnwrHvpH6

And we’re done… Thanks for your time!



sven.schleier@owasp.org
@bsd_daemon

Appendix

Memory Analysis on iOS

Memory Analysis on iOS

Analyzing the memory is done by developers usually to identify

- memory leaks or
- root cause of app crashes

A penetration tester or security researcher will also analyze the memory, but he wants to identify:

- Sensitive information, e.g. Passwords, Personal Identifiable Information (PII)
- Encryption keys

Memory Analysis on iOS

To investigate an application's memory, you must first create a **memory dump**.

Memory dumping is a very error-prone process in terms of verification because each dump contains the output of executed functions. You may miss executing critical scenarios. In addition, overlooking data during analysis is probable unless you know the data's footprint (either the exact value or the data format).

For example, if the app encrypts with a randomly generated symmetric key, you likely won't be able to spot it in memory unless you can recognize the key's value in another context.

Memory Analysis on iOS

Now it's your turn:
Dump memory of an iOS App!

Open the file Lab-iOS.pdf:

```
$ evince ~/MSTG-Handson/Lab/Lab-iOS.pdf
```

Section: **Lab – Analysing Memory on iOS**

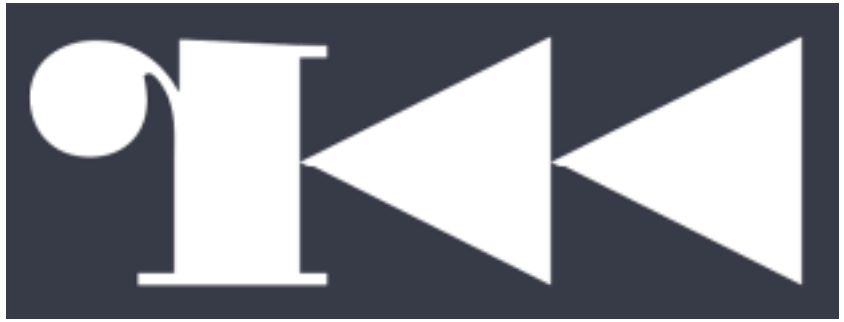
Please type the commands and do not copy & paste!

1. You will remember them easier
2. Some PDF readers do not copy properly and miss whitespaces etc.

DISCLAIMER: THIS LAB CAN BE A BIT BUGGY AND THE APP MIGHT CRASH

Memory Analysis on iOS

- Radare2 is a reverse engineering framework
- It aims to provide a full toolchain for reverse engineering



Github: https://github.com/radareorg/radare2

Memory Analysis on iOS

r2Frida combines Radare2 with Frida

Install r2FRida via r2pm:
https://github.com/nowsecure/r2frida#installation

Demonstration of memory dumping with r2 Frida

Memory Analysis on iOS

In this example you were retrieving data that you keyed in!

But think about private keys you want to protect. Even when stored in the KeyChain, they will still be exposed in memory when being used.

In order to store them securely they should be stored in the Secure Enclave, than they will not even be exposed in the memory.

The Secure Enclave is a hardware-based key manager that's isolated from the main processor to provide an extra layer of security. When you store a private key in the Secure Enclave, you never actually handle the key, making it difficult for the key to become compromised. Instead, you instruct the Secure Enclave to create the key, securely store it, and perform operations with it. You receive only the output of these operations, such as encrypted data or a cryptographic signature verification outcome.

See also:
https://developer.apple.com/documentation/security/certificate_key_and_trust_services/keys/storing_keys_in_the_secure_enclave

Memory Analysis on iOS

Memory Analysis might be more useful for Malware analysis on iOS than during a penetration test for an app.

Memory analysis in the mobile domain is not much developed, particularly in iOS.

See also
https://mobileapphacking.com/2018/01/15/ios-memory-dump/

https://files.sans.org/summit/DFIR_Summit_Prague_2016/PDFs/iOS-Forensics-Where-are-we-now-Mattia-Epifani-Pasquale-Stirparo.pdf

Uncrackable Challenges

iOS – Reverse Engineering - Uncrackable Level 1

Reverse Engineering hands-on:

- Uncrackable Level 1
  - Static analysis
  - Dynamic instrumentation

iOS – Reverse Engineering - Uncrackable Level 1

Go to the iOS Apps directory (for Linux):

```
$ cd ~/MSTG-Handson/Apps
```

Execute the following commands on either your macOS device or the Linux VM to install the app:

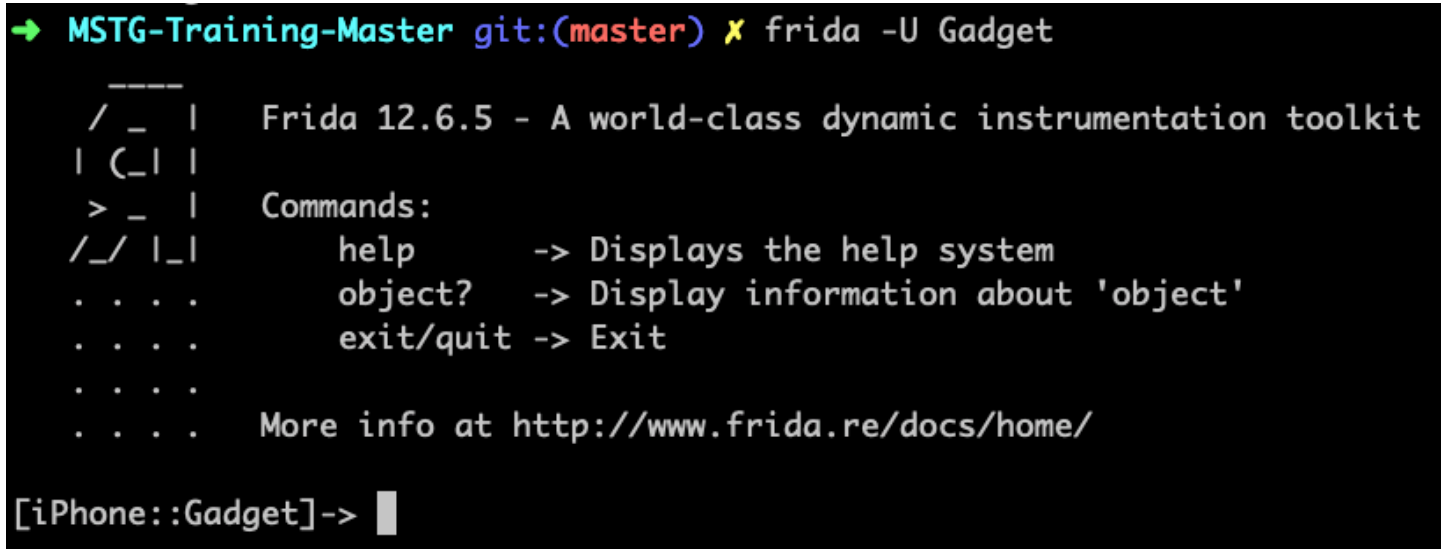
macOS	Linux VM
Unpack the IPA: $ unzip UnCrackable_Level1.ipa	Install the IPA on iOS: $ ideviceinstaller -i UnCrackable_Level2.ipa
Install and run the app with ios-deploy: $ ios-deploy --bundle 'Payload/UnCrackable Level 1.app/' -W -d -v	Execute and run the app through it's bundle name: $ idevicedebug -d run me.ryantzj.com

iOS – Reverse Engineering - Uncrackable Level 1

Testing connection with Frida Gadget

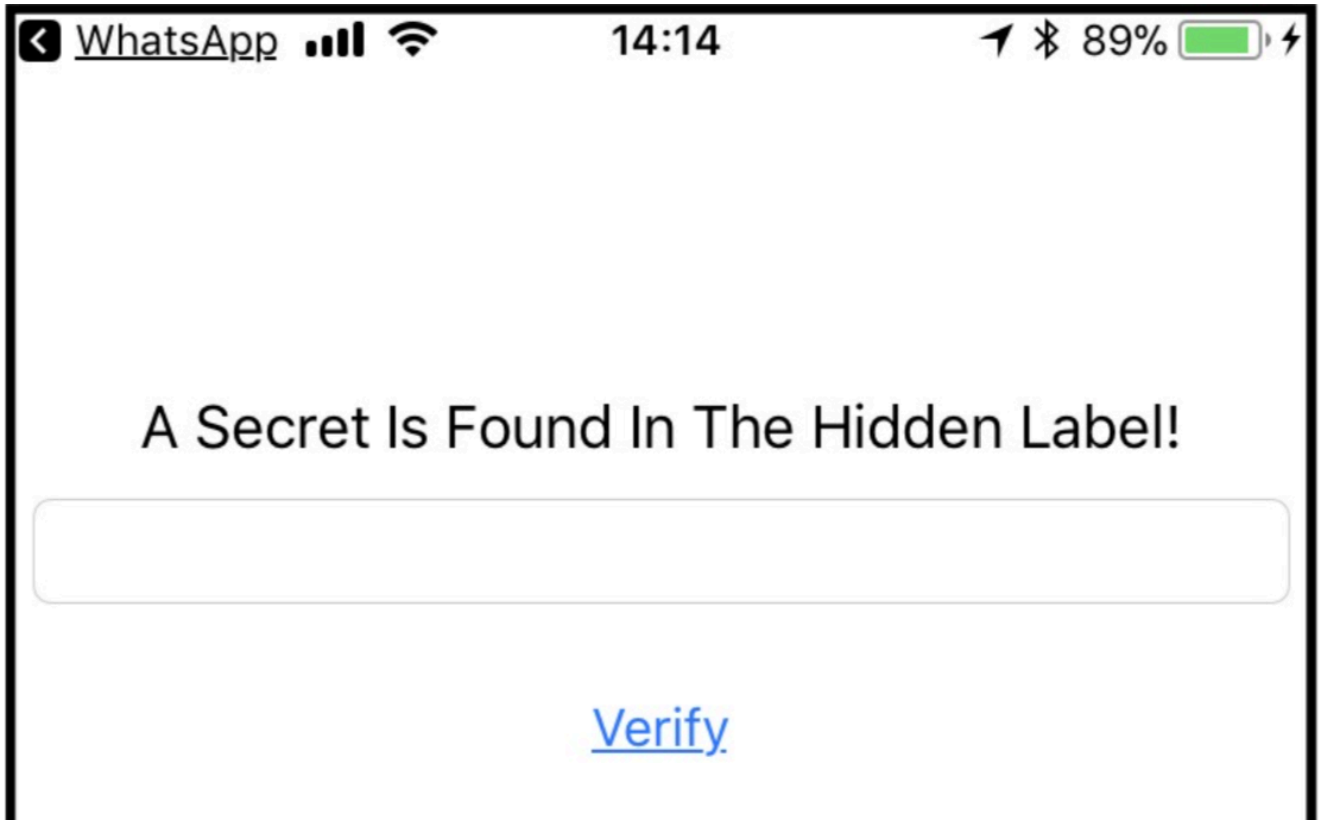
```
$ frida-ps –U
PID  Name
---  ------
528  Gadget
```

Hooking into the app with Frida



iOS – Reverse Engineering - Uncrackable Level 1

Objective: Discover the hidden label!



iOS – Reverse Engineering - Uncrackable Level 1

Dumping classes (only possible on macOS)

```
➜  Payload git:(master) ✗ class-dump -S -s -I -H UnCrackable\ Level\ 1.app -o uncrackable1.dump
➜  Payload git:(master) ✗ cd uncrackable1.dump
➜  uncrackable1.dump git:(master) ✗ l
total 48
drwxr-xr-x@ 7 sven  staff   224B Oct 24 16:52 .
drwxr-xr-x@ 5 sven  staff   160B Oct 24 16:52 ..
-rw-r--r--@ 1 sven  staff   936B Oct 24 16:52 AppDelegate.h
-rw-r--r--@ 1 sven  staff   366B Oct 24 16:52 CDStructures.h
-rw-r--r--@ 1 sven  staff   898B Oct 24 16:52 NSObject-Protocol.h
-rw-r--r--@ 1 sven  staff   5.6K Oct 24 16:52 UIApplicationDelegate-Protocol.h
-rw-r--r--@ 1 sven  staff   803B Oct 24 16:52 ViewController.h
➜  uncrackable1.dump git:(master) ✗
```

iOS – Reverse Engineering - Uncrackable Level 1

Open the File ViewController.h

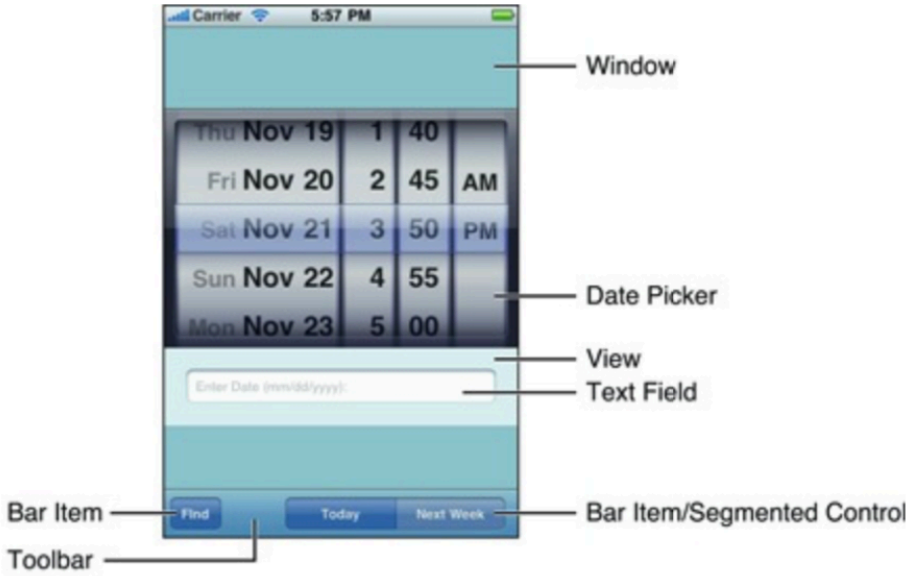
```
7   #import "UIViewController.h"
8
9   @class UIButton, UILabel, UITextField;
10
11  @interface ViewController : UIViewController
12  {
13      UILabel *_theLabel;
14      UILabel *_Hint;
15      UITextField *_theTextField;
16      UIButton *_bVerify;
17  }
18
19  - (void).cxx_destruct;
20  @property(nonatomic) __weak UILabel *Hint; // @synthesize Hint=_Hint;
21  @property(nonatomic) __weak UIButton *bVerify; // @synthesize bVerify=_bVerify;
22  - (void)buttonClick:(id)arg1;
23  - (void)didReceiveMemoryWarning;
24  @property(nonatomic) __weak UILabel *theLabel; // @synthesize theLabel=_theLabel;
25  @property(nonatomic) __weak UITextField *theTextField; // @synthesize theTextField=_theTextField;
26  - (void)viewDidLoad;
27
28  @end
```

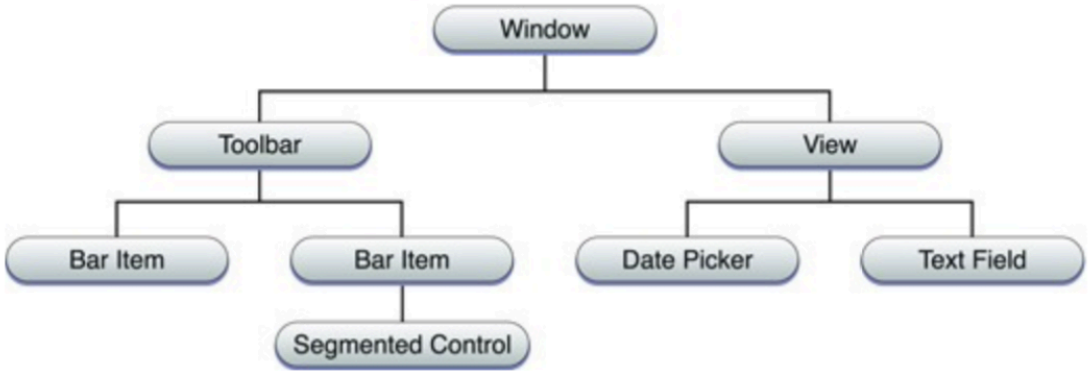
From the ViewController header file we can identify that there are 2 labels, 1 textfield and 1 Button declared, but we can see only 1 label on the app.
Let’s print the ViewHierarchy of the app!

iOS – Reverse Engineering - Uncrackable Level 1

A View Hierarchy defines the relationships of views in a window to each other

Example of View Hierarchy





iOS – Reverse Engineering - Uncrackable Level 1

Let’s continue in our Frida shell and create a JS Wrapper on the KeyWindow method and invoke a recursiveDescription function:

```
[iPhone::Gadget]-> var keyWindow = ObjC.classes.UIWindow.keyWindow();
undefined
[iPhone::Gadget]-> keyWindow.recursiveDescription()
{
    "handle": "0x280806fd0"
}
[iPhone::Gadget]->
```

Seems like it generates an object. Let’s try to confirm that using JS typeof function:

```
[iPhone::Gadget]-> var viewHierarchy = keyWindow.recursiveDescription()
undefined
[iPhone::Gadget]-> typeof viewHierarchy
"object"
[iPhone::Gadget]-> String(viewHierarchy)
```

With object we can simply cast it as string to print it. Can you spot the hidden label?

iOS – Reverse Engineering - Uncrackable Level 1

Type your solution in the app and verify the answer: