BUILDING SECURE MOBILE APPS

(YOU DON'T HAVE TO LEARN IT THE HARD WAY!)

SVEN SCHLEIER AND CARLOS HOGUERA

OWASP STAMMTISCH HAMBURG JAN 2021

$ /USR/BIN/WHOAMI

Hi everyone, my name is Sven!

- Previous roles: Unix Admin, Penetration Tester, Security Architect for Web and Mobile Apps during SDLC
- Now Security Architect in ☀️ Singapore
- Project leader together with Carlos Holguera of:
  - OWASP Mobile Security Testing Guide (MSTG) and
  - OWASP Mobile AppSec Verification Standard (MASVS)
- Blogging on http://bsddaemonorg.wordpress.com/

$ /USR/BIN/WHOAMI

Hola, my name is Carlos!

- Security Engineer & Technical Lead in Berlin:
  - Mobile & Automotive Security Testing
  - Security Testing Automation
- Project leader together with Sven Schleier of:
  - OWASP Mobile Security Testing Guide (MSTG) and
  - OWASP Mobile AppSec Verification Standard (MASVS)

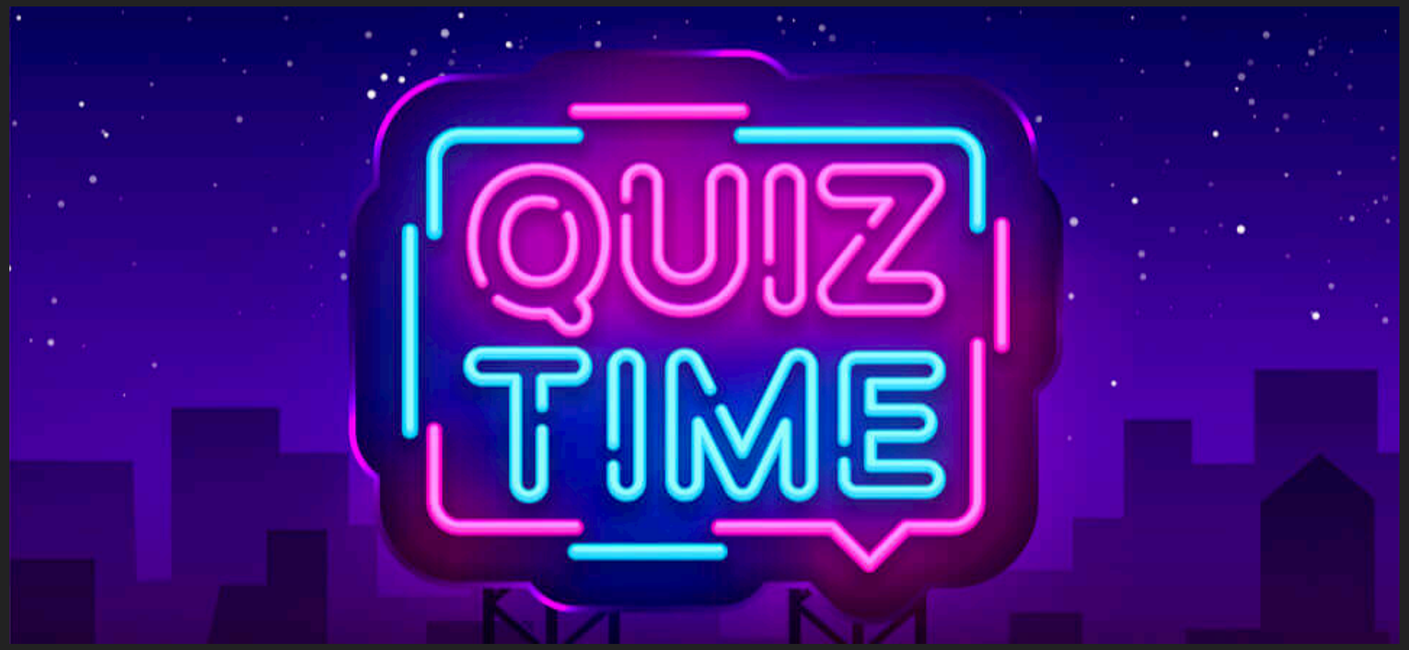
AGENDA

OWASP MOBILE APPSEC VERIFICATION STANDARD (MASVS)

OWASP MOBILE SECURITY TESTING GUIDE (MSTG)

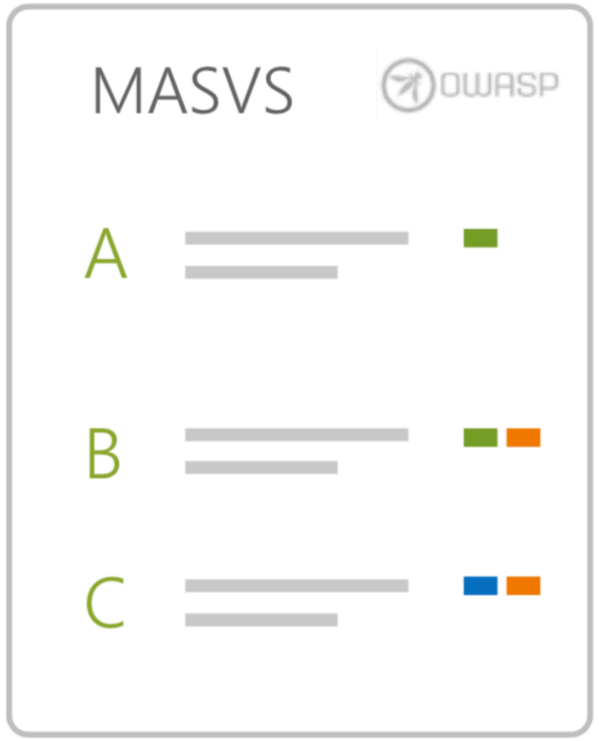
DEMOS

LET ME ASK YOU SOME QUESTIONS FIRST!



Go to **www.menti.com** and use the code **51 92 84 2**

MOBILE APPSEC VERIFICATION STANDARD (MASVS)

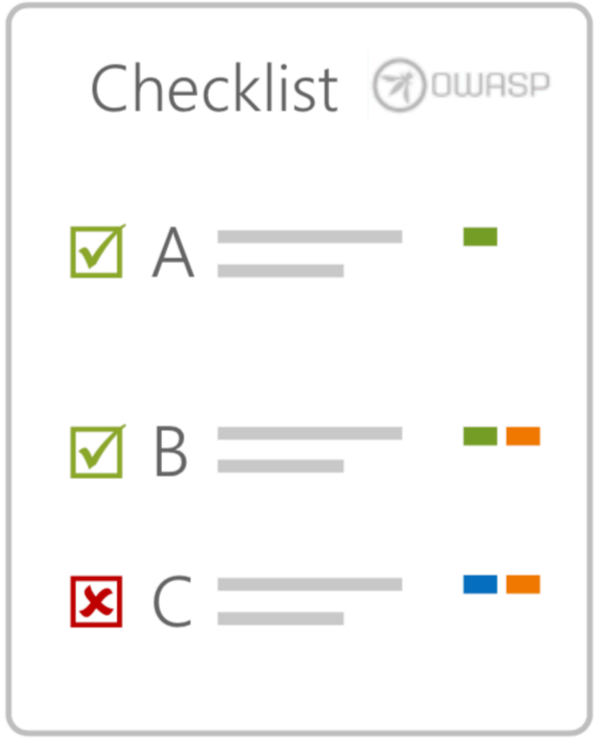


https://github.com/OWASP/owasp-masvs/releases

MOBILE SECURITY TESTING GUIDE (MSTG)

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

MOBILE APPSEC CHECKLIST

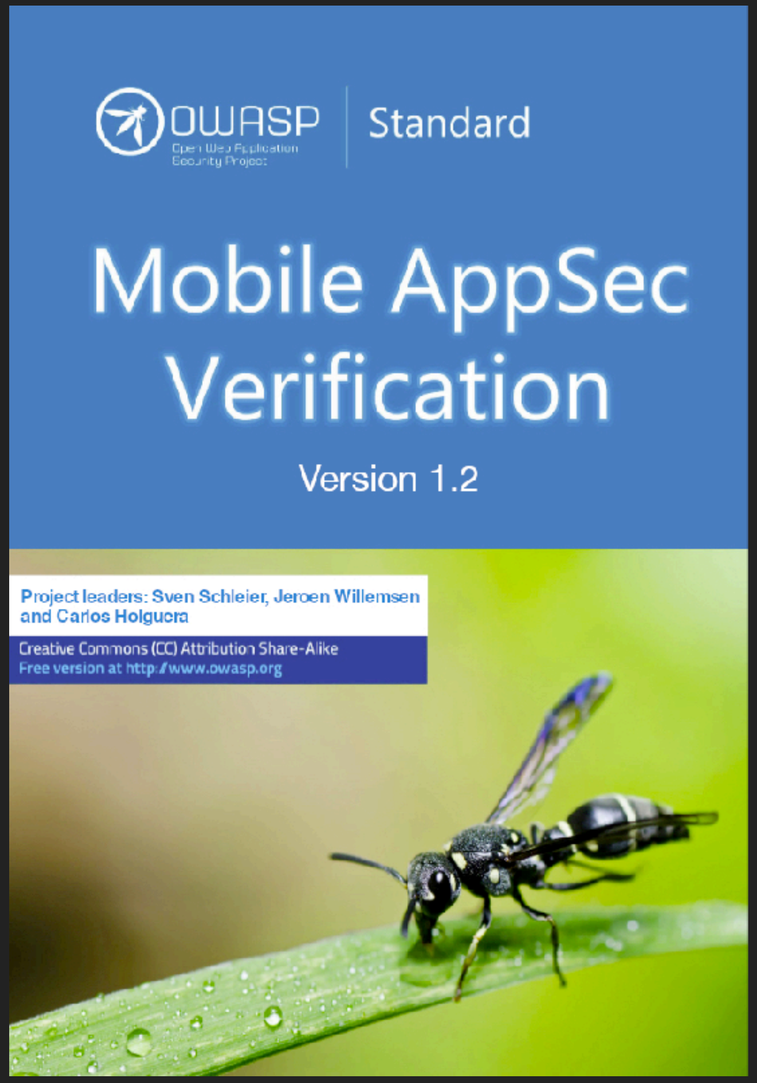


https://github.com/OWASP/owasp-mstg/tree/master/Checklists

OWASP MOBILE APPSEC VERIFICATION STANDARD (MASVS)

OWASP MOBILE SECURITY TESTING GUIDE (MSTG)

HANDS-ON



THE MASVS IS A STANDARD THAT DEFINES THE SECURITY REQUIREMENTS APPLICABLE FOR MOBILE APPS AND IS OS AGNOSTIC.

Translations available:

- Chinese (Traditional and Simplified)
- Farsi (Persian)
- French
- German
- Hindi
- Japanese
- Korean
- Portugese (inca. Brazilian Portugese)
- Russian
- Spanish

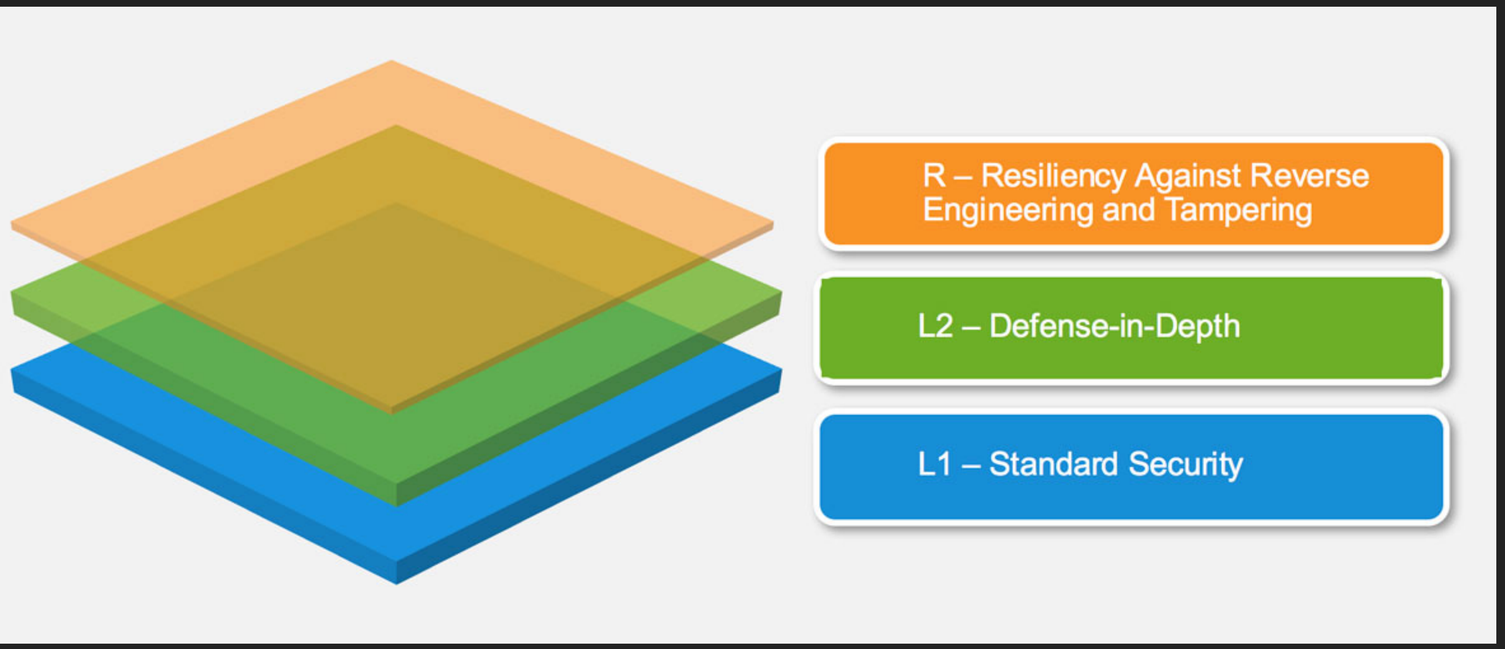
https://github.com/OWASP/owasp-masvs#getting-the-masvs

V5: NETWORK COMMUNICATION REQUIREMENTS

#	MSTG-ID	Description	L1	L2
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 practices, or as close as possible if the mobile operating system does not support the recommended standards.	✓	✓
5.3	MSTG-NETWORK-3	The app verifies the X.509 certificate of the remote endpoint when the secure channel is established. Only certificates signed by a trusted CA are accepted.	✓	✓
5.4	MSTG-NETWORK-4	The app either uses its own certificate store, or pins the endpoint certificate or public key, and subsequently does not establish connections with endpoints that offer a different certificate or key, 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.		✓

https://github.com/OWASP/owasp-masvs/blob/master/Document/0x10-V5-Network_communication_requirements.md

KEEPING THINGS FLEXIBLE: REQUIREMENT "LEVELS"



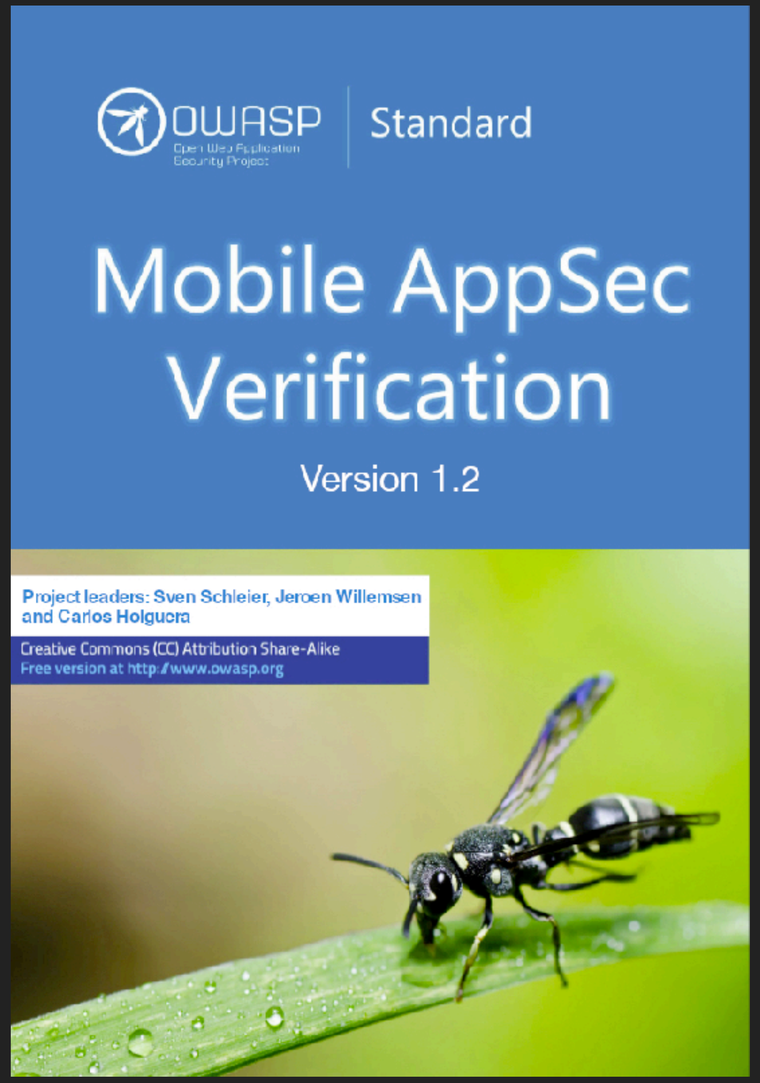
L2 + R	e.g. Banking Apps
L1 + R	e.g. Game Apps
L2	e.g. Health Apps
L1	All mobile apps

https://github.com/OWASP/owasp-masvs/blob/master/Document/0x03-Using_the_MASVS.md#verification-levels-in-detail

HOW TO USE THE MASVS?

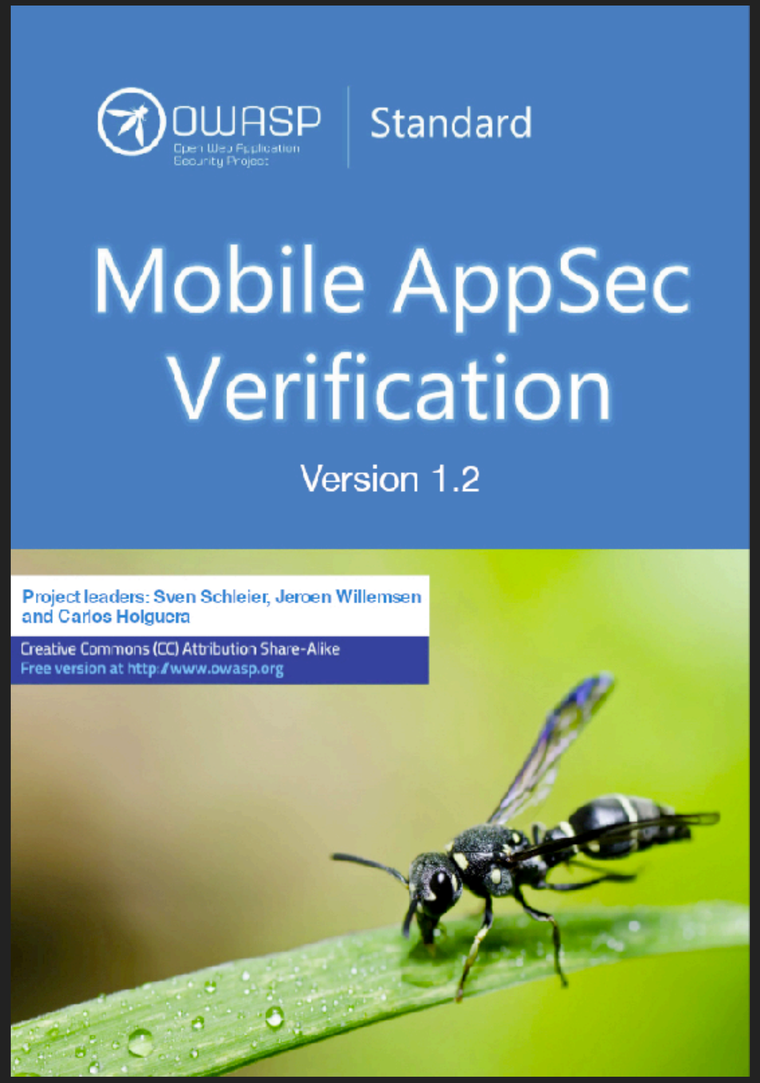
- The levels and it's requirements are a baseline that need to be tailored to your needs.
- Don't blindly follow the requirements!
  - Requirements might be missing (e.g. regulations in your country/industry)
  - Requirements might not be applicable (or you may want to accept the risk)
- Usage ensures consistency of mobile app security when developing / testing an app
- Can be part of your threat model to select the requirements that address your gaps!

WHERE CAN I GET IT?



- Github - http://bit.ly/2uMFDiY
- Gitbook - http://bit.ly/30kZPnW
- Releases - http://bit.ly/2NqspPc

- Download it
- Read it
- Use it
- Give Feedback and create an issue!



- A new release (Version 1.3) is in the making and will be published soon!

OWASP MOBILE APPSEC VERIFICATION STANDARD (MASVS)

OWASP MOBILE SECURITY TESTING GUIDE (MSTG)

HANDS-ON

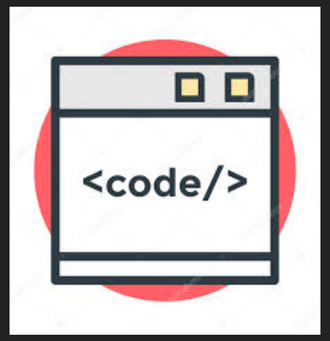
THE MSTG IS A COMPREHENSIVE MANUAL FOR MOBILE APP SECURITY TESTING AND REVERSE ENGINEERING.

IT DESCRIBES TECHNICAL PROCESSES FOR VERIFYING THE CONTROLS LISTED IN THE MASVS.

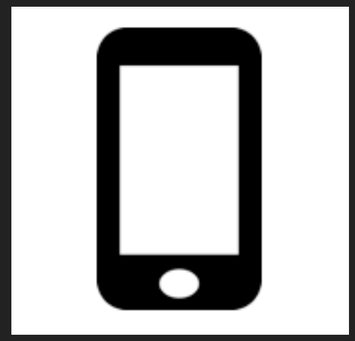
https://github.com/OWASP/owasp-mstg/#reading-the-mobile-security-testing-guide

STRUCTURE OF A TEST CASE IN THE MSTG

Overview



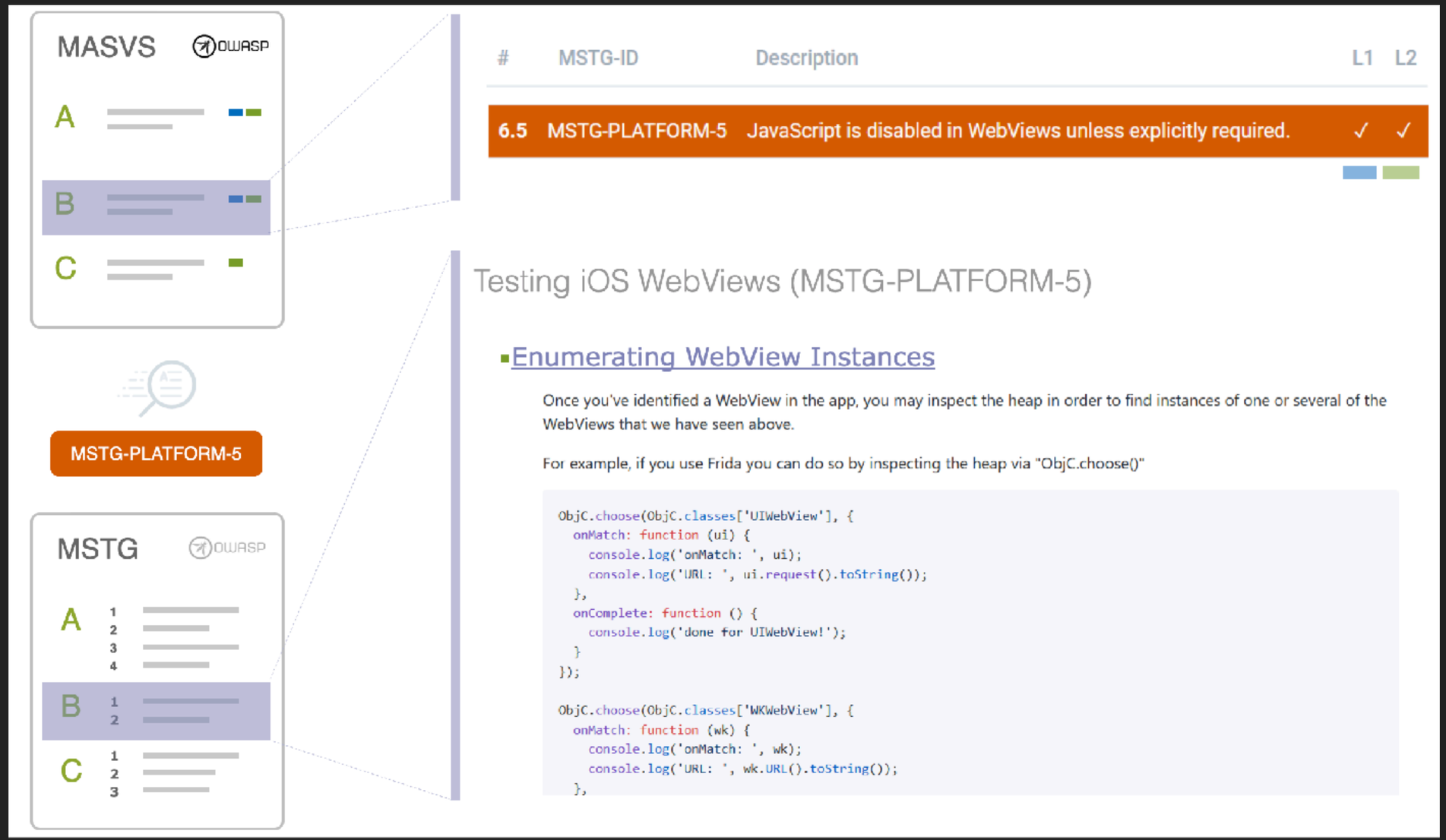
Static Analysis (here you will also find the best practice)



Dynamic Analysis

Example: Testing iOS WebViews - http://bit.ly/3cjH4sX

EXAMPLE: MSTG-PLATFORM-5



WHERE CAN I GET IT?

- Github - http://bit.ly/381ZRn9
- Gitbook - http://bit.ly/36Qr2Rz
- Releases - http://bit.ly/2Rdef57

- Download it
- Read it
- Use it
- Give Feedback and create an issue!

OWASP MOBILE APPSEC VERIFICATION STANDARD (MASVS)

OWASP MOBILE SECURITY TESTING GUIDE (MSTG)

HANDS-ON

HOW DOES A PENETRATION TESTER EXECUTE A TEST FOR AN IOS APP?

Jailbroken Device

- Cydia App Store
- Full Root Access

Dynamic instrumentation

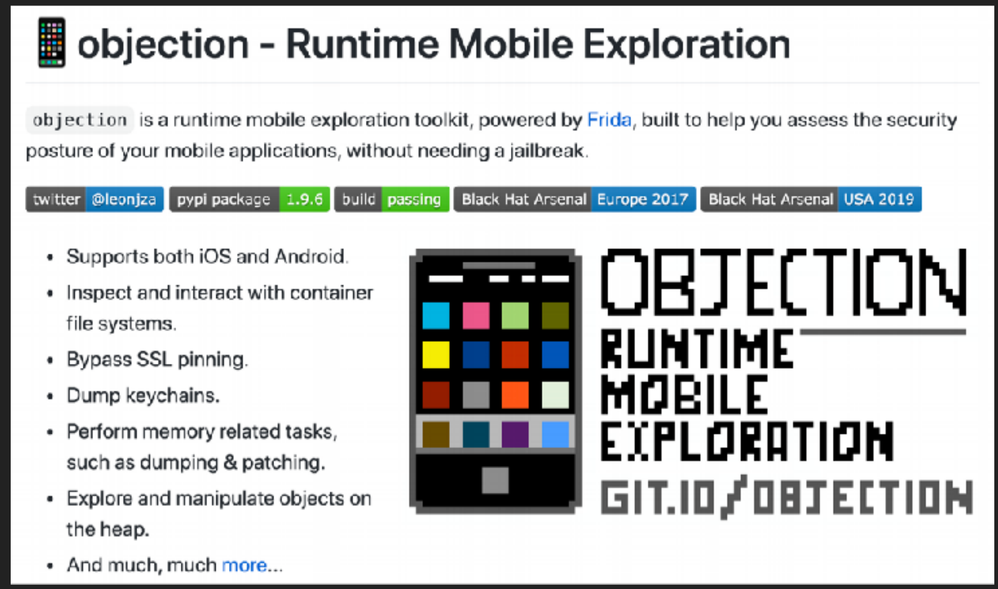
- Works on (non-)jailbroken devices
- Manipulate runtime behaviour of an app through Frida

FRIDA

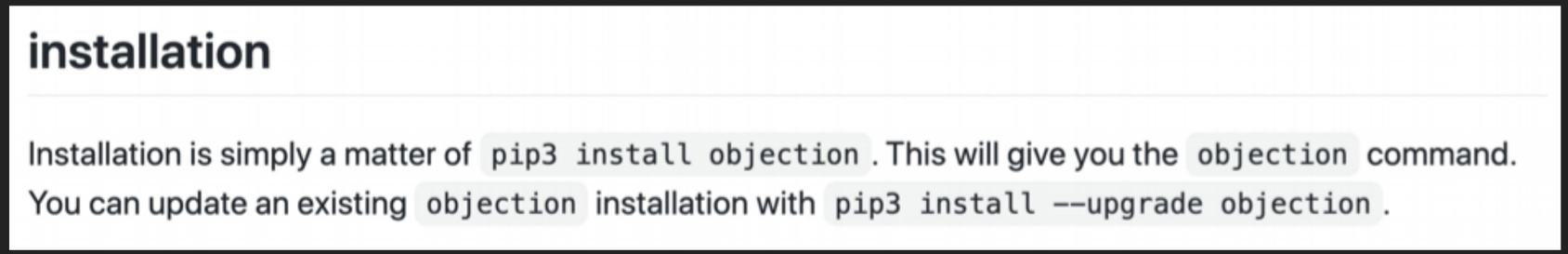
See also:

- Frida iOS: https://www.frida.re/docs/ios/
- iOS Basic Security Testing: https://bit.ly/2lHdGoj
- iOS Dynamic Testing on non jailbroken device: https://bit.ly/2lG7Kf7

WAYS TO ANALYSE LOCAL STORAGE – OBJECTION



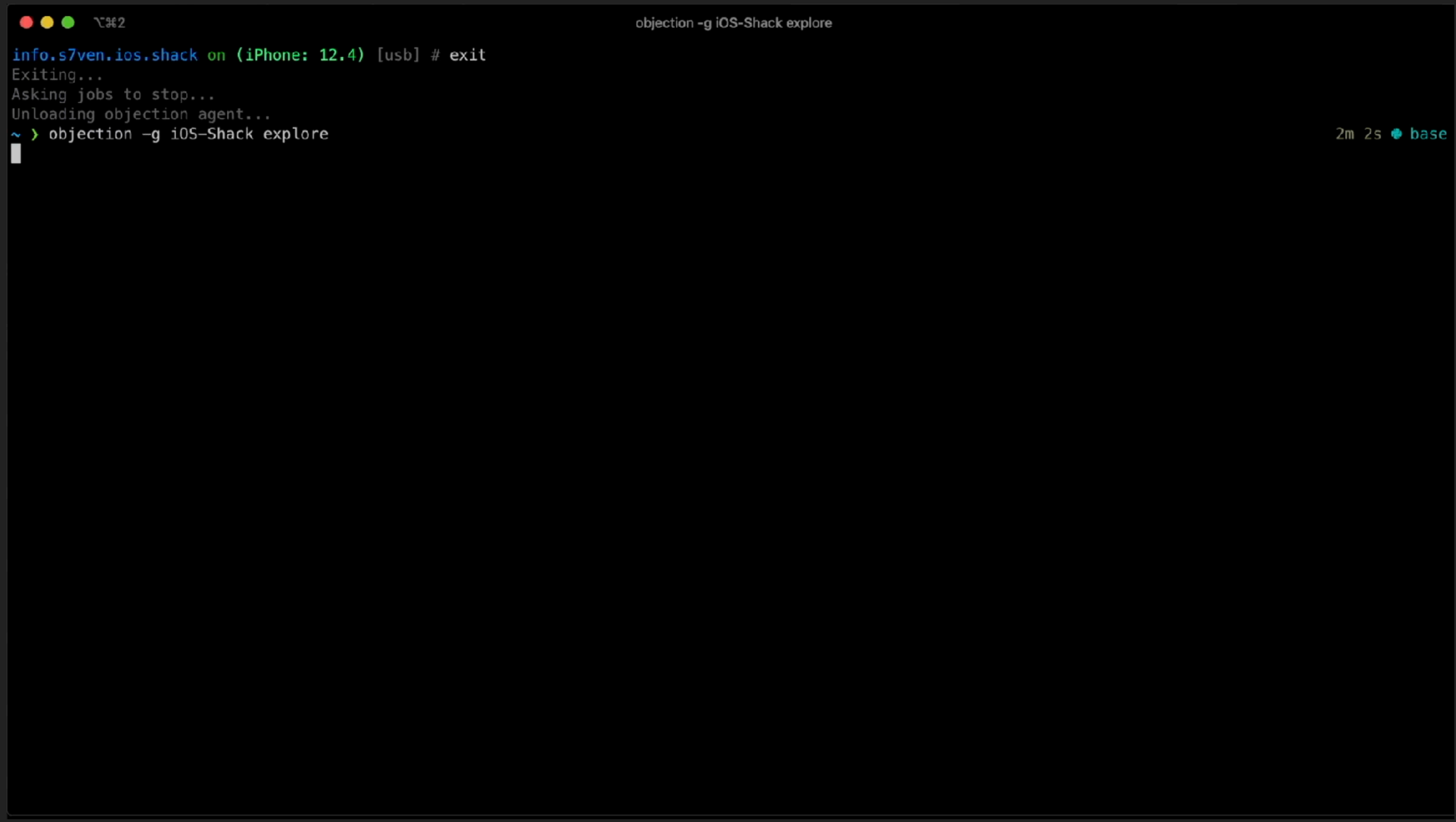
- https://github.com/sensepost/objection
- Python based
- Can be installed and upgraded by using pip3



- For more detailed update and installation instructions, please refer to the wiki page:

https://github.com/sensepost/objection/wiki/Installation

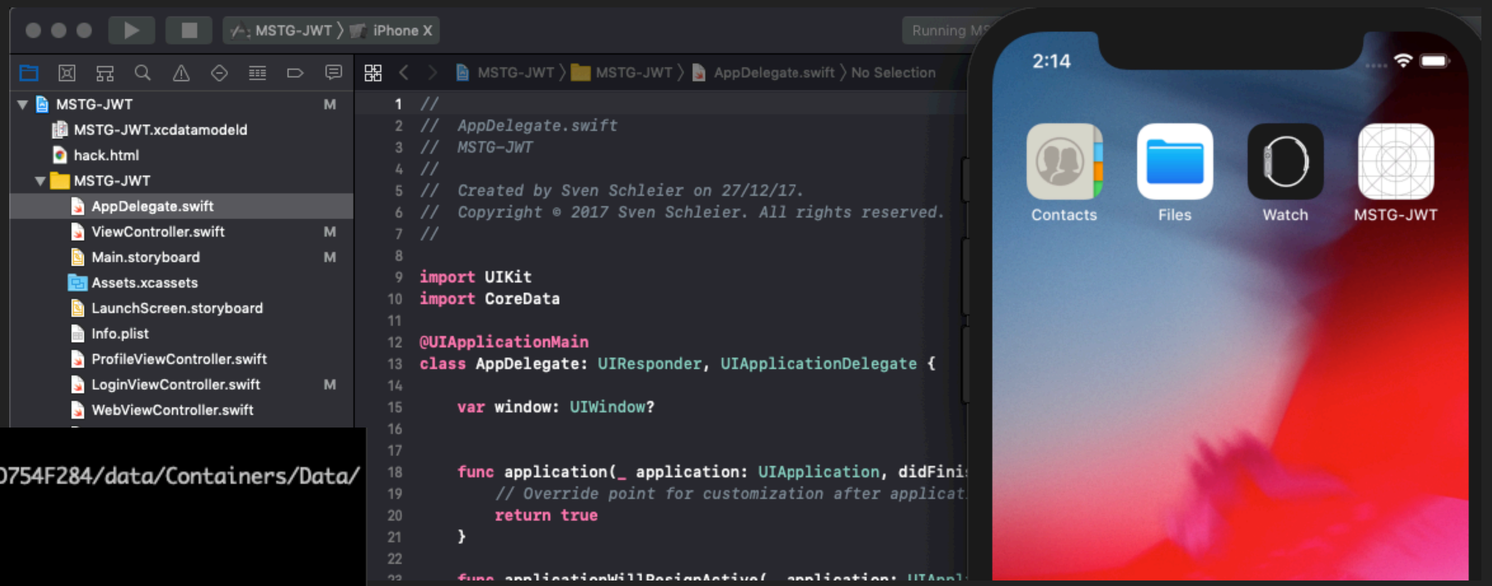
How to analyse local storage of an iOS App (Penetration Tester)



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?

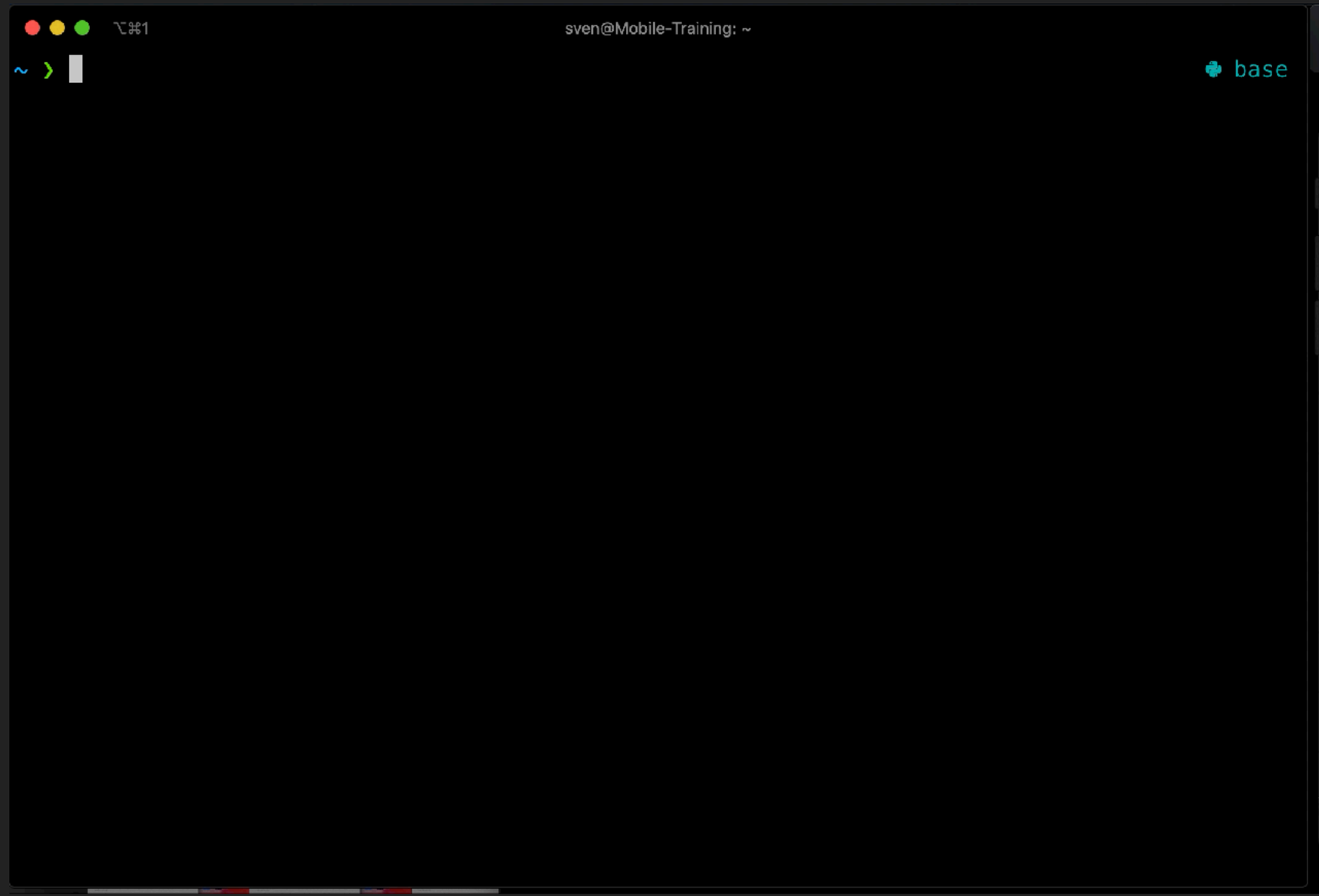
Use the tools you already have: Xcode and iOS Simulator

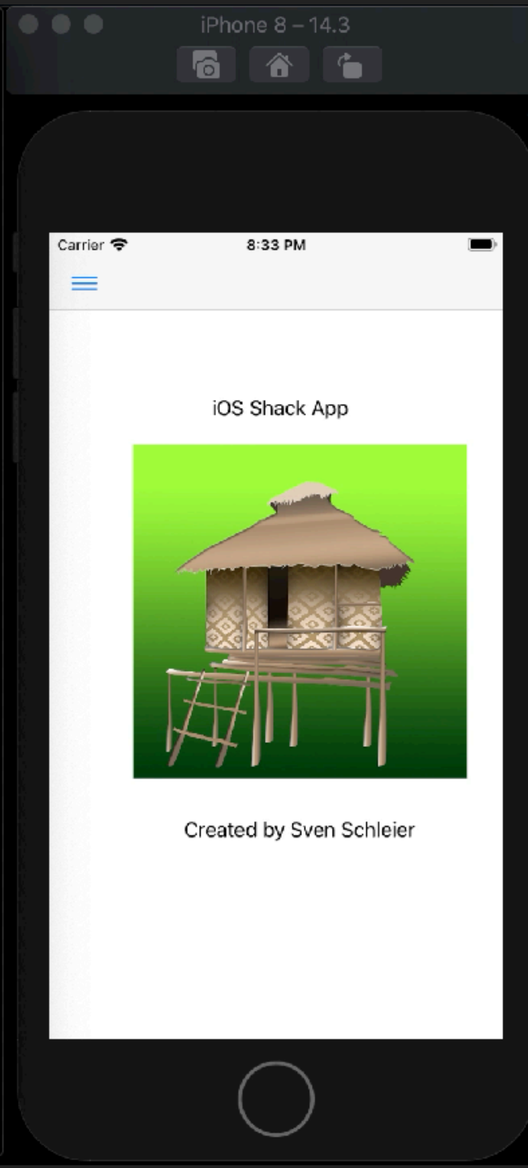
- You have full access to the file system of the iOS Simulator




```
➜ 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
```

How to analyse local storage of an iOS App (Developer Perspective)





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?

Every app and simulator gets a random 128-bit UUID (Universal Unique Identifier) assigned during installation for its directory names. When using the iOS Simulator the path is:

~/Library/Developer/CoreSimulator/Devices/<Device-UUID>/data/Containers/Data/Application/<App-UUID>

A very handy way to open the data directory of our app running in the current simulator in Finder is the following:

```
$ open `xcrun simctl get_app_container booted info.s7ven.ios.data data` -a Finder
```

The bundle name would need to be explicitly specified, which is **info.s7ven.ios.data** in this case.

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

HOW TO DO IT RIGHT?

First reflect: Is it really necessary to store sensitive information on the device? If so, use the following:

Keychain (small bits of data)

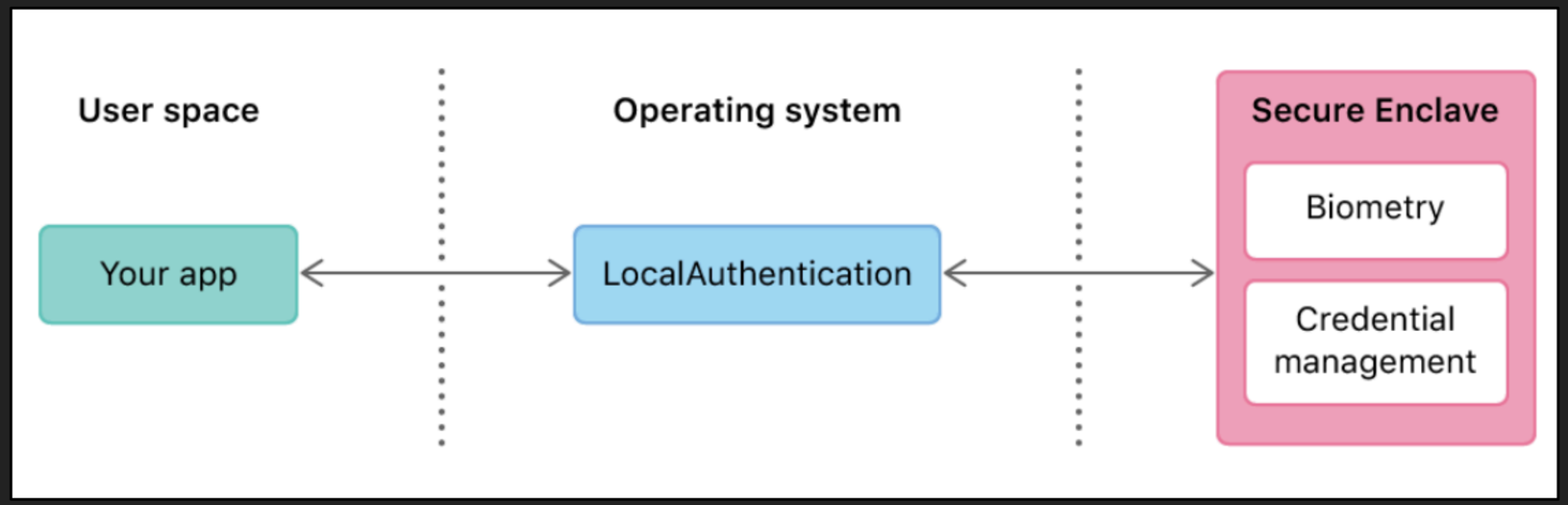
- The iOS Keychain can be used to securely store short, sensitive bits of data, such as encryption keys and session tokens. It is implemented as an SQLite database that can be accessed through the Keychain APIs only.

iOS Data Protection APIs

- App developers can leverage the iOS Data Protection APIs to implement fine-grained access control for user data stored on the device.

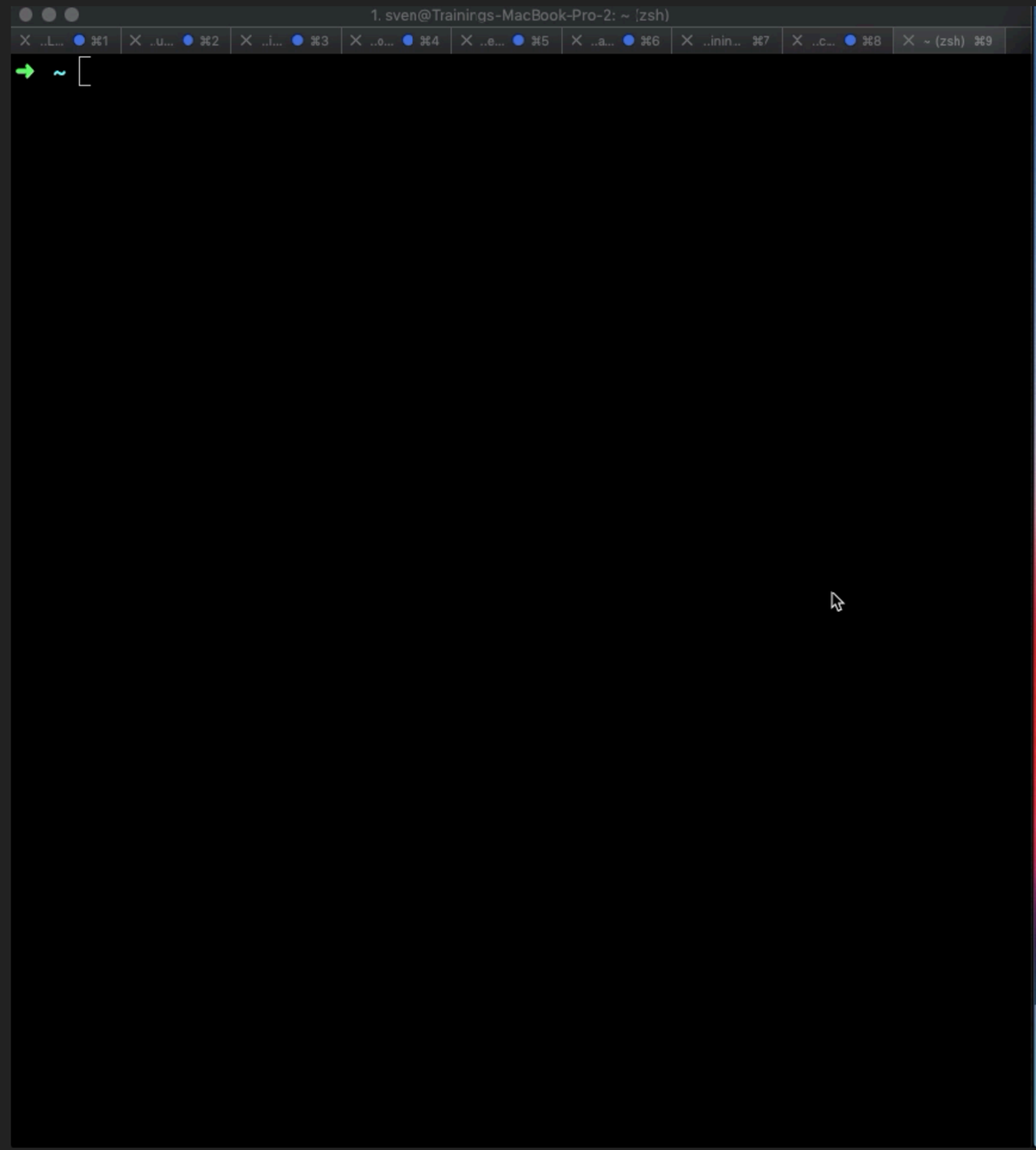
TOUCH ID / FACE ID

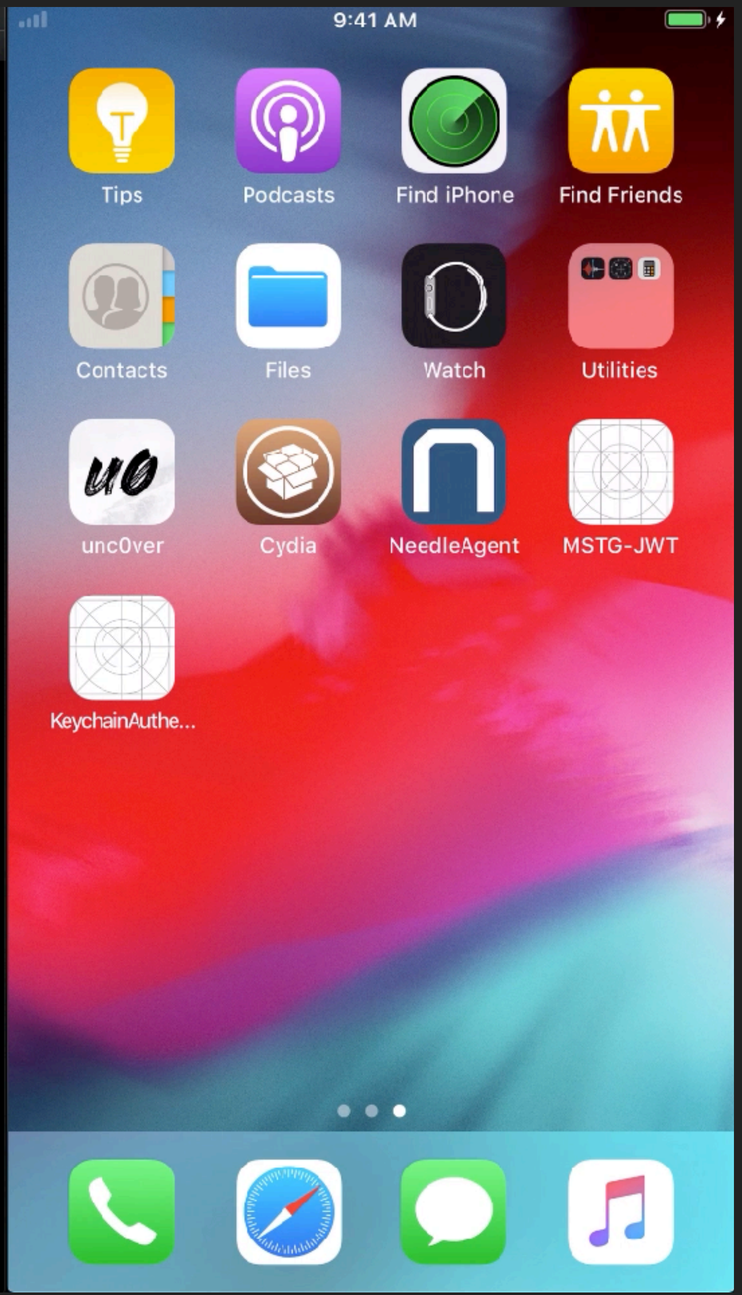
- Fingerprint / facial data is stored in the Secure Enclave which is part of the processor of an iOS device (during calibration).
- The provided data (fingerprint / facial data) is sent to the Secure Enclave and compared with the stored data to authenticate the user.
- An iOS app can confirm via the LocalAuthentication (LAContext) helper class to confirm the devices passphrase, Touch ID or Face ID.



BYPASSING TOUCH ID THE EASY WAY…

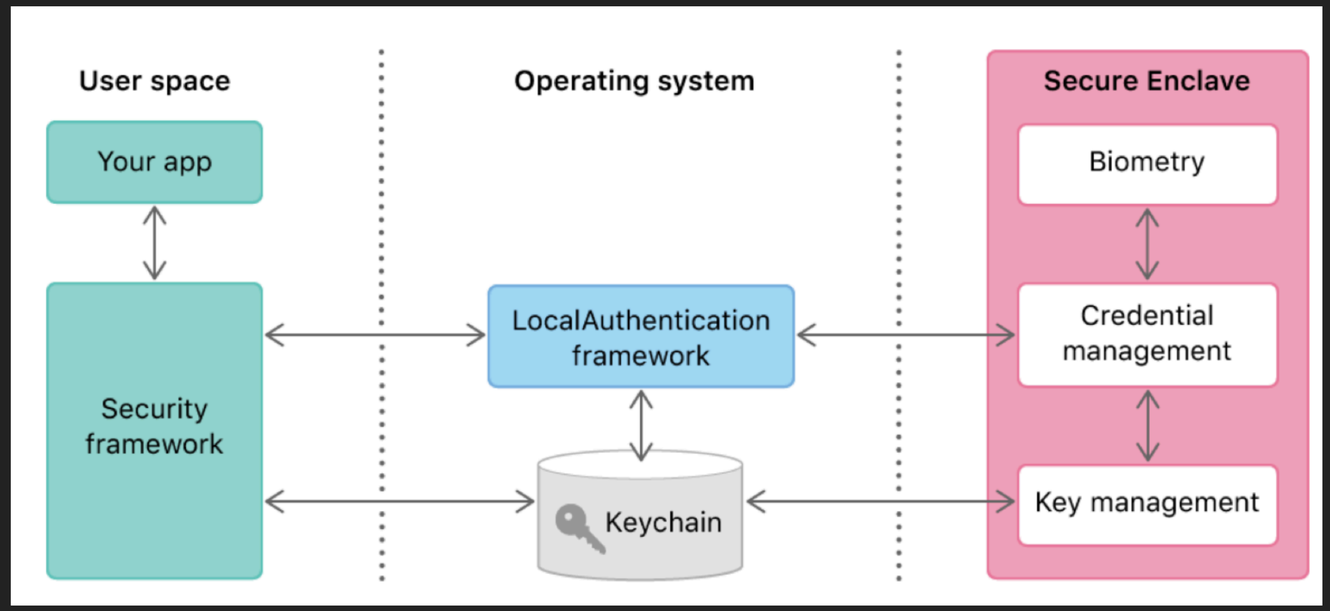
BYPASSING TOUCH ID





HOW TO DO IT RIGHT?

- 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:

- http://bit.ly/2KVNVKv
- https://apple.co/2KUscTr

HOW CAN WE MAKE SUCH ATTACKS HARDER?

- Jailbreak detection
- Detection of Dynamic Instrumentation (Frida)
- Anti Tampering
- Obfuscation
- …

Client Side Security Controls are always a cat and mouse game!

JAILBREAK DETECTION

What does Jailbreak Detection mean?

- File-based Checks
- Checking File Permissions
- Checking Protocol Handlers (cydia://)
- Calling System APIs
- …

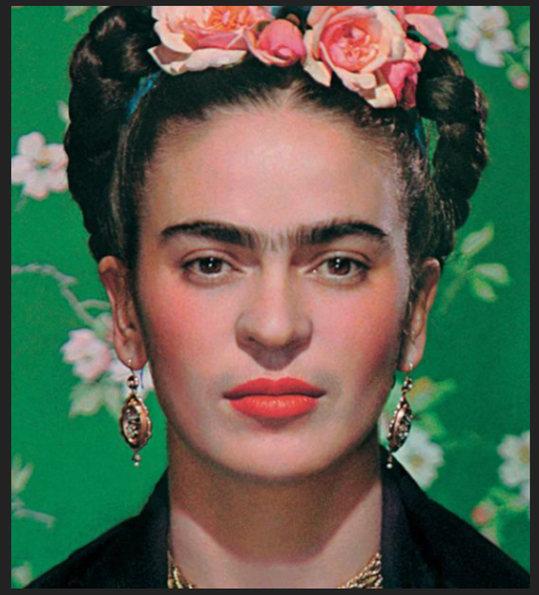
See also: http://bit.ly/33oEvgR

DYNAMIC BINARY INSTRUMENTATION DETECTION (FRIDA)

How can Frida be detected?

- Checking the App Signature
- Checking For Open TCP Ports
- Scanning Process Memory
- …

See also: bit.ly/2MfkXJx



Frida Kahlo

DYNAMIC BINARY INSTRUMENTATION DETECTION (FRIDA)

Where there's a detection, there is a bypass.

- Detection: https://github.com/securing/IOSSecuritySuite/blob/master/IOSSecuritySuite/JailbreakChecker.swift
- Bypass: https://github.com/as0ler/frida-scripts/blob/master/hooks/_jailbreak_detection.disabled

Frida Kahlo

And all in GitHub :)

DETECTION BYPASS THROUGH BINARY PATCHING

- Patch the executable binary file (disassemble or just edit the raw file)
- The bypass might be as easy as *making true (0x0) to false (0x1)* or *replacing some logic with a NOP*!
- Repackage and re-run the app

```
                    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
```

DETECTION BYPASS THROUGH DYNAMIC BINARY INSTRUMENTATION

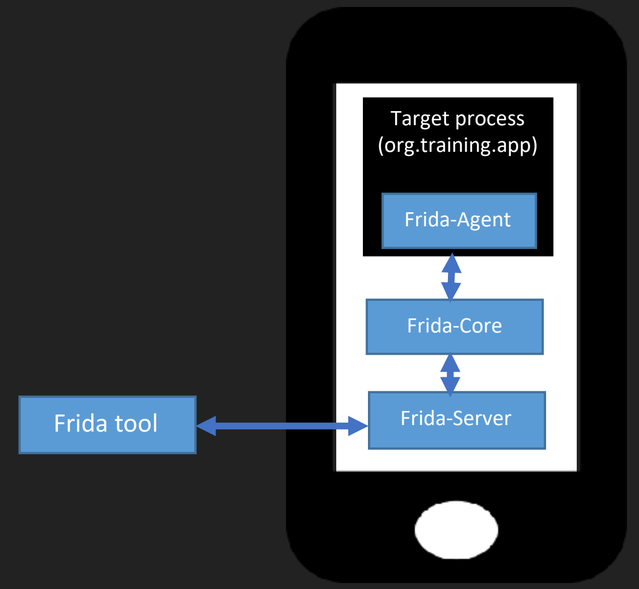
- Inject code to the running app
- Reverse engineering can help finding out which code to inject
- The bypass might be as easy as *forcing a function to return true or false*!

```
setTimeout(function(){
        Java.perform(function (){
                console.log("[*] Script loaded")

                var MainActivity = Java.use("org.owasp.mstg.antifrida.MainActivity")

                MainActivity.checkMemory.overload().implementation = function() {
                        console.log("[*] checkMemory function invoked")
                        return false
                }

                MainActivity.PortScanFrida.overload('java.lang.String').implementation = function() {
                        console.log("[*] PortScanFrida function invoked")
                        return false
                }

                MainActivity.getSignature.overload().implementation = function() {
                        console.log("[*] getSignature function invoked")
                        return "99sL2NrjIHWOtn7nBqgQ3Qwvlyc="
                }

        });

});
```

FRIDA – MODES OF OPERATION

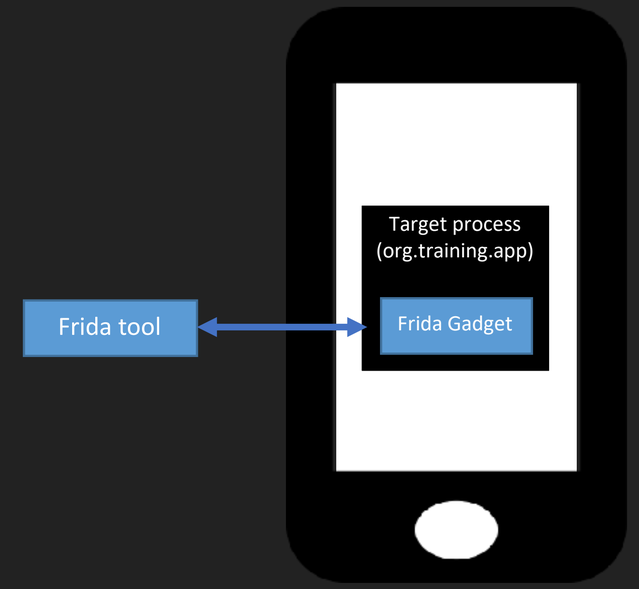
Injected into a process by running the Frida server on the device

- Working on only jailbroken devices
- Frida handles the injection



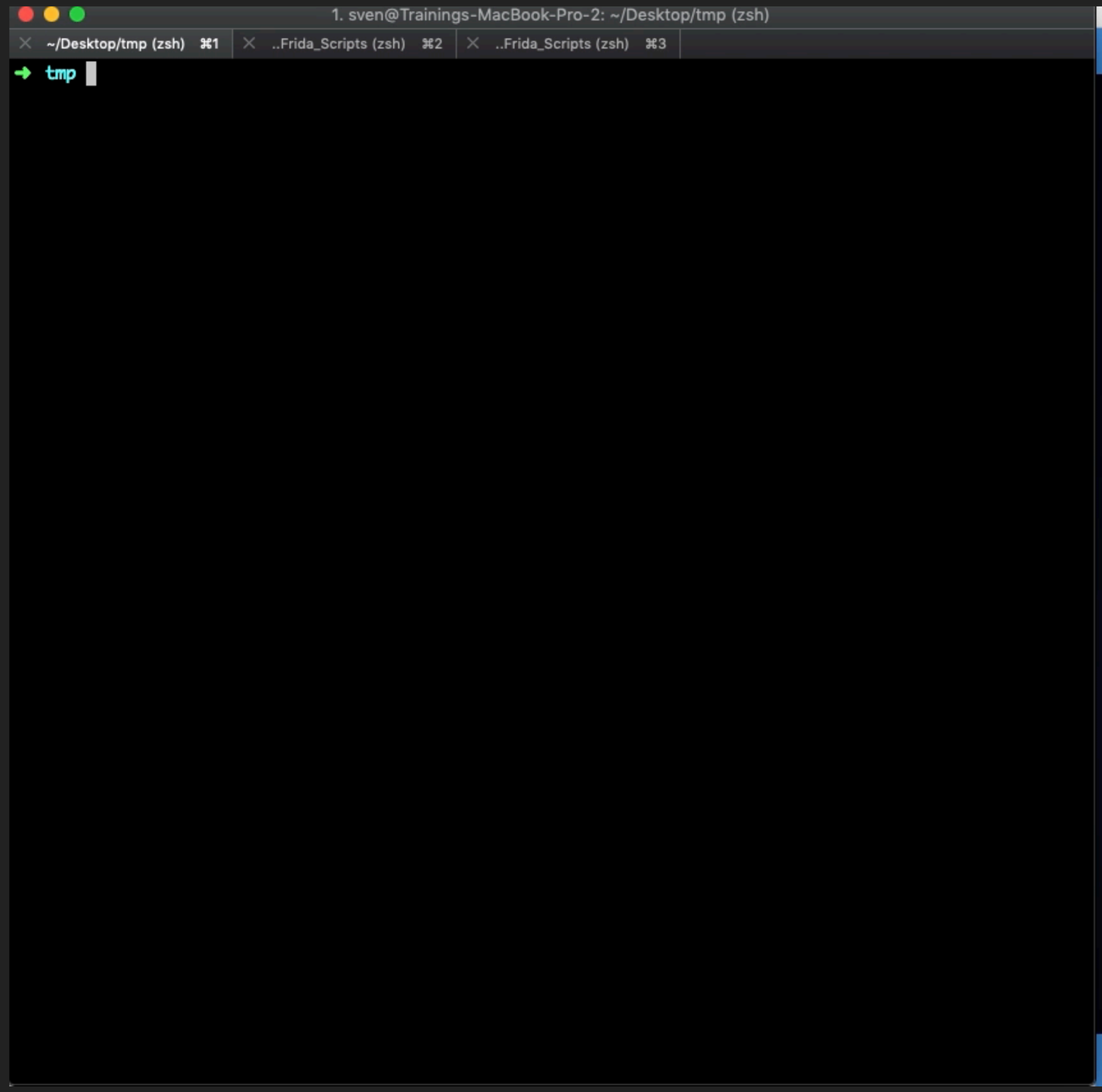
Embedded as shared library (frida-gadget.so) into the mobile app

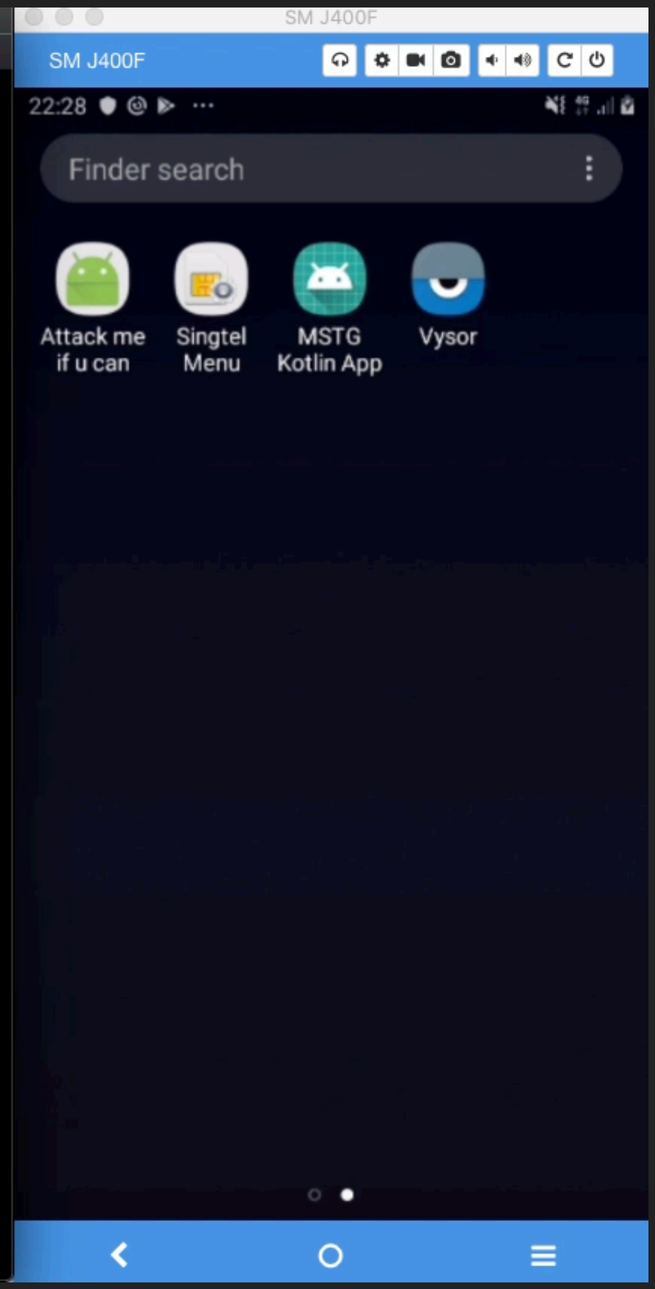
- Working on non-jailbroken devices
- Repackaging and resigning required



FRIDA DETECTION BYPASS THROUGH DYNAMIC BINARY INSTRUMENTATION

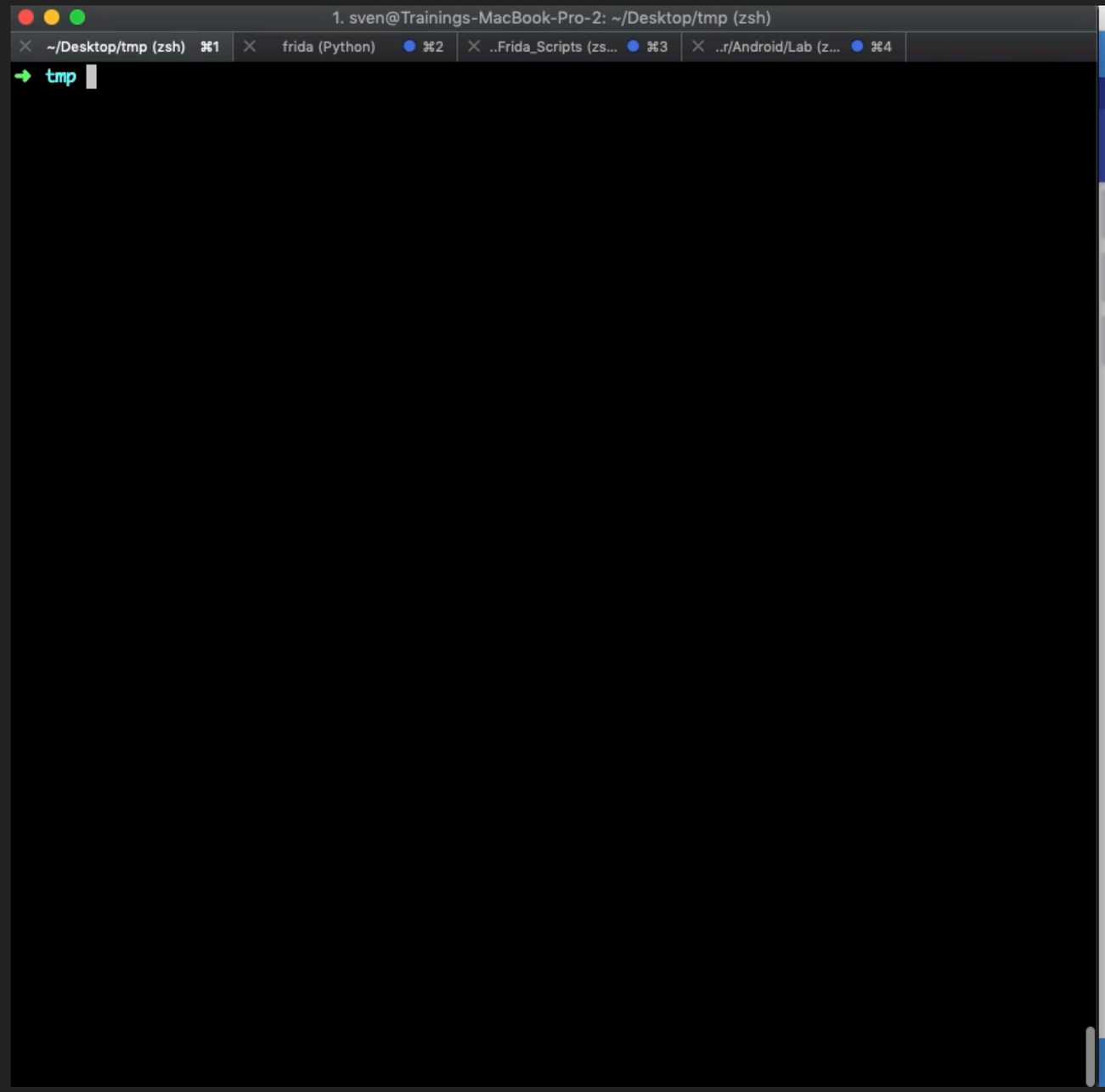
Inject and Attach

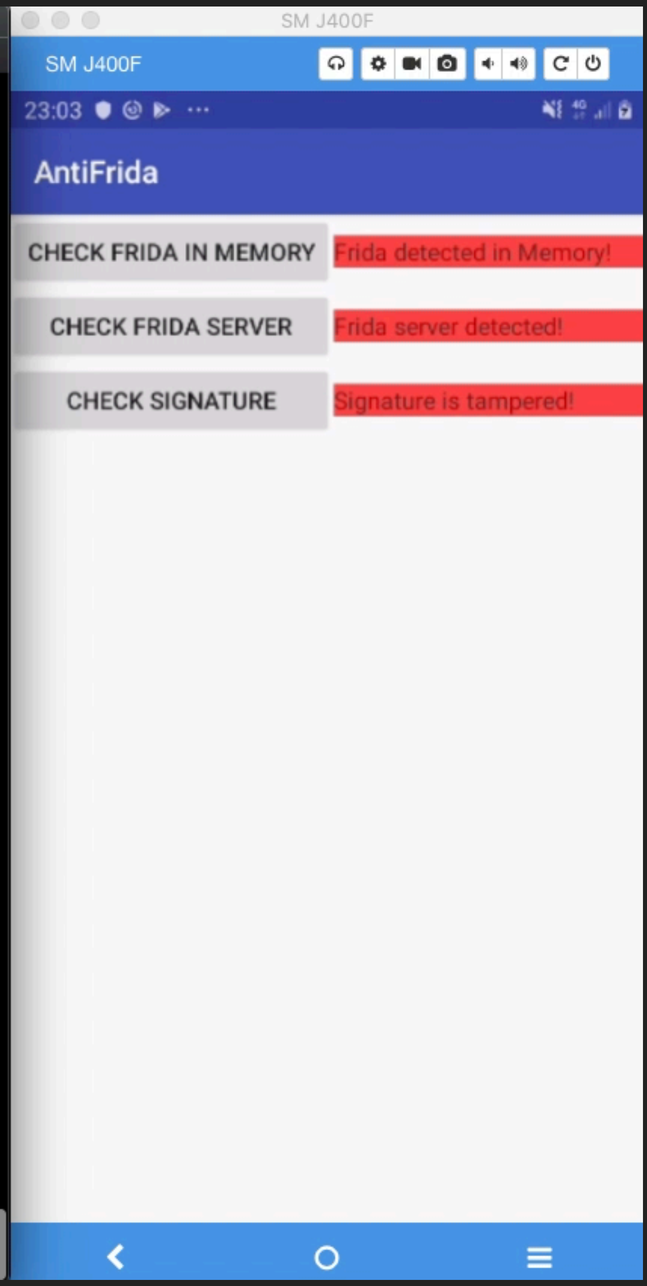




FRIDA DETECTION BYPASS THROUGH DYNAMIC BINARY INSTRUMENTATION

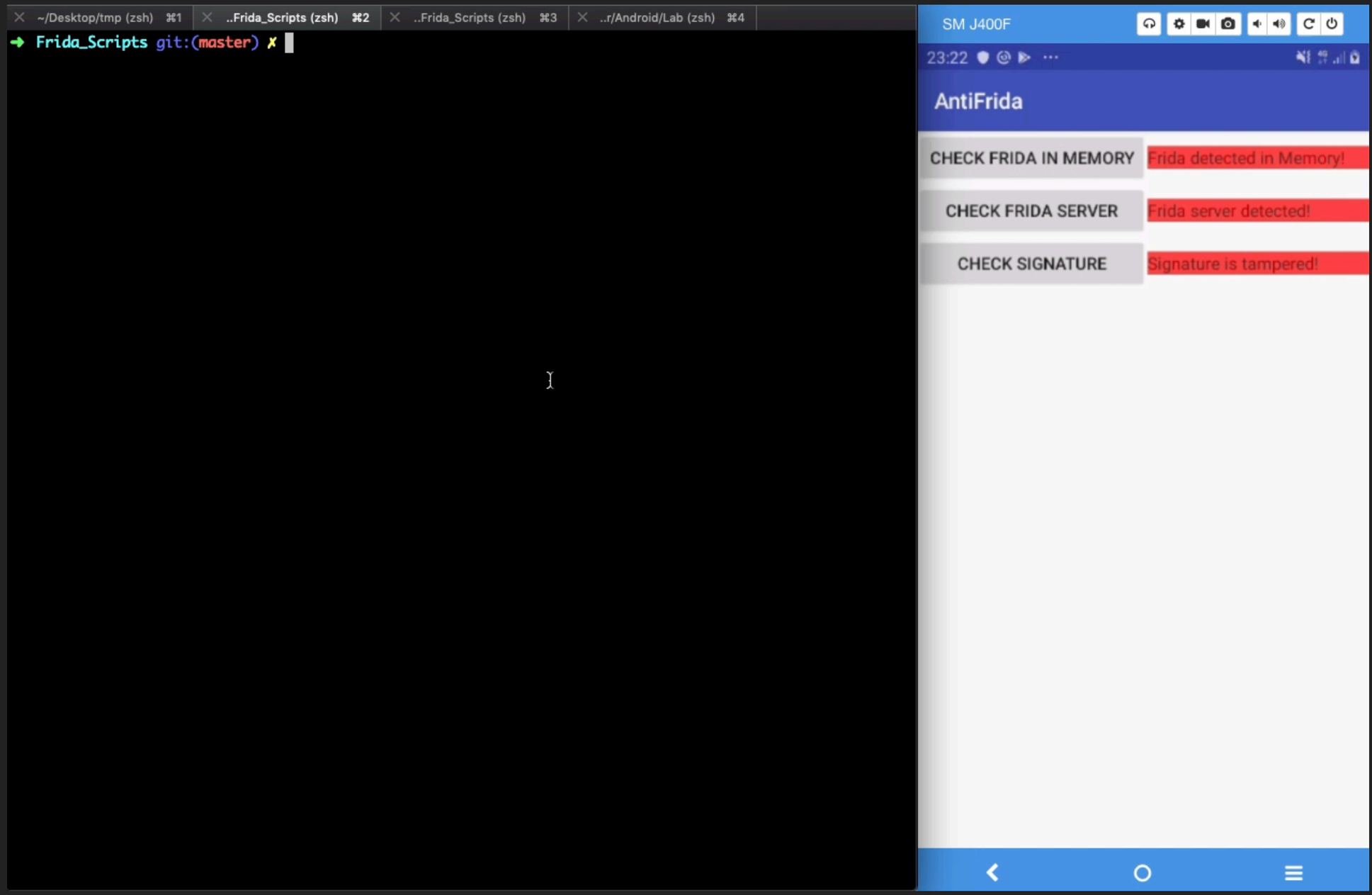
Reverse Engineer





FRIDA DETECTION BYPASS THROUGH DYNAMIC BINARY INSTRUMENTATION

Script and Bypass



WHAT ABOUT LAYERING CLIENT SIDE CONTROLS?

- 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
- Usage of Obfuscation
- etc.

Makes the effort more time consuming and can be used as part of a defence in depth strategy by raising the bar and putting obstacles in the attackers way.

Remember: Reverse Engineering is still possible and will always be a cat and mouse game!

V8: RESILIENCE REQUIREMENTS

Impede Dynamic Analysis and Tampering

#	MSTG-ID	Description	
8.1	MSTG-RESILIENCE-1	The app detects, and responds to, the p jailbroken device either by alerting the u	
8.2	MSTG-RESILIENCE-2	The app prevents debugging and/or de debugger being attached. All available covered.	
8.3	MSTG-RESILIENCE-3	The app detects, and responds to, tamp critical data within its own sandbox.	
8.4	MSTG-RESILIENCE-4	The app detects, and responds to, the p engineering tools and frameworks on t	
8.5	MSTG-RESILIENCE-5	The app detects, and responds to, being	
8.6	MSTG-RESILIENCE-6	The app detects, and responds to, tamp own memory space.	
8.7	MSTG-RESILIENCE-7	The app implements multiple mechanis (8.1 to 8.6). Note that resiliency scales the originality of the mechanisms used	
8.8	MSTG-RESILIENCE-8	The detection mechanisms trigger resp including delayed and stealthy respons	
8.9	MSTG-RESILIENCE-9	Obfuscation is applied to programmatic defenses, which in turn impede de-obfuscation via dynamic analysis.	✓

Device Binding

#	MSTG-ID	Description	R
		he app implements a 'device binding' functionality using a device ingerprint derived from multiple properties unique to the device.	✓

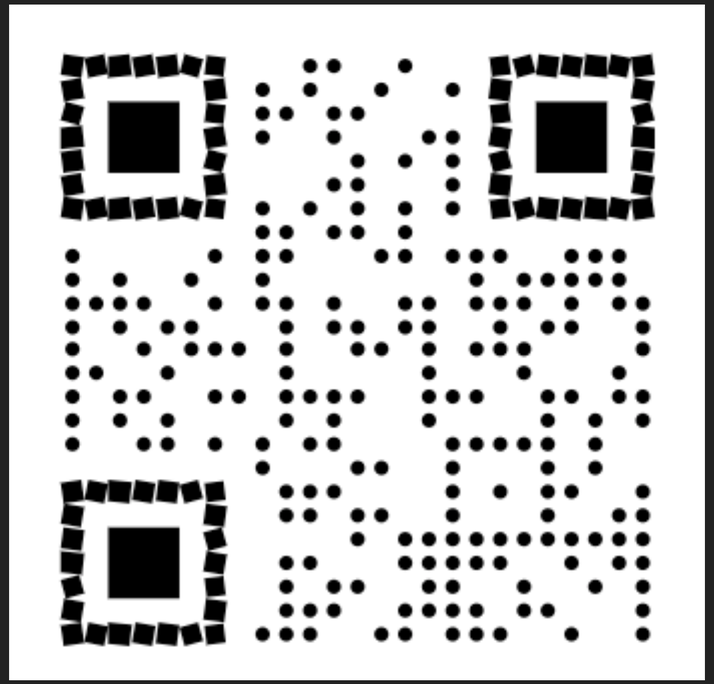
		Description	R
		All executable files and libraries belonging to the app are either encrypted on the file level and/or important code and data segments inside the executables are encrypted or packed. Trivial static analysis does not reveal important code or data.	✓
		f the goal of obfuscation is to protect sensitive computations, an obfuscation scheme is used that is both appropriate for the particular task and robust against manual and automated de-obfuscation methods, considering currently published research. The effectiveness of the obfuscation scheme must be verified through manual testing. Note that hardware-based isolation features are preferred over obfuscation whenever possible.	✓

		Description	R
8.13	MSTG-RESILIENCE-13	As a defense in depth, next to having solid hardening of the communicating parties, application level payload encryption can be applied to further impede eavesdropping.	✓

http://bit.ly/359PpZh

KEY TAKEAWAYS

- The MASVS defines mobile apps security requirements
- The MSTG outlines those requirements into technical test cases for Android and iOS
- Make a Threat Model of your app
- Get the basics right first (MASVS Level 1)
- Main Security belongs ALWAYS in the server. NEVER rely on client side security controls only.
- Reverse Engineering Controls NEVER go alone, layer them as a defence-in-depth strategy
- The Reverse Engineer will always win!



Download slide deck here:

http://bit.ly/2YkjQuA

Thank you!

sven.schleier@owasp.org

@bsd_daemon

Carlos.Holguera@owasp.org

@grepharder