



Mobile Security Testing Guide - Hands-on

Presented by:
Sven Schleier and Ryan Teoh

Table of Contents

iOS Labs

Introduction	1.1
Lab - Install IPA and Basic Dynamic Analysis	1.2
Lab - Frida 101	1.3
Lab - Local Storage, testing for Sensitive Data	1.4
Lab - Biometric Authentication	1.5
Lab - Bypassing SSL Pinning	1.6
Lab - Stateless Authentication	1.7
Lab - Bypassing Jailbreak Detection	1.8
Lab - Bypass Piracy Detection	1.9

Appendix

Lab - Memory	2.1
Testing Tools	2.2
Suggested Reading	2.3

MSTG Hands-on - iOS

Even though modern mobile operating systems like iOS and Android offer great APIs for secure data storage and communication, those APIs have to be used correctly in order to be effective. Data storage, inter-app communication, proper usage of cryptographic APIs and secure network communication are only some of the aspects that require careful consideration. The OWASP Mobile Security Testing Guide (MSTG) is a comprehensive manual for testing the security of mobile apps. It describes processes and techniques for verifying the requirements listed in the Mobile Application Security Verification Standard (MASVS), and provides a baseline for complete and consistent security tests.

This training is based on the Mobile Security Testing Guide (MSTG) and will offer hands-on exercises in the form of different iOS and Android Apps. They will demonstrate bad practices and current security best practices to avoid vulnerabilities and flaws within mobile Apps.

The goal of this course is to learn the technical skills to execute a penetration test against iOS and Android mobile applications and utilise the Mobile Security Testing Guide (MSTG) as a baseline and comprehensive methodology during mobile security assessments.

Enjoy the ride!

Lab - Install IPA and Basic Dynamic Analysis

Training Objectives

1. Install an IPA
2. Use Xcode for Log Analysis
3. Analyse network traffic

Time for completing this lab: 15 min

Exercise difficulty: easy

Tools used in this section

- ios-deploy
- Xcode
- Wireshark

App

Use the following app for this exercise:

- MSTG-JWT.ipa

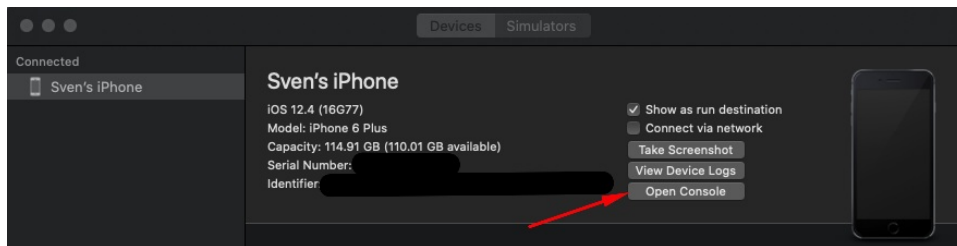
You can find the app on your macOS in the directory `ios-preparation-pack/Apps` , wherever you unzipped the file `ios-preparation-pack.zip` .

Exercise - Log Analysis

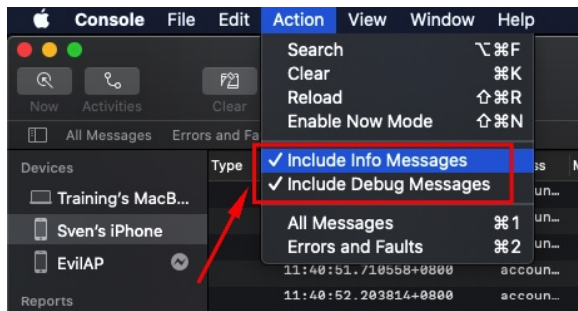
- Connect your iOS device to your MacBook via USB cable and unlock the device, so you can see the home screen.
- Open Xcode on your MacBook. Go to "Window / Devices and Simulators":



- In the new window that opened, your device will show up on the left side. Select your device and click on "Open Console":



- Ensure that in "Action" the Info and Debug messages are included:



- The console is a tool on macOS that offers you a central place for all log files of the host, but also all connected devices. This allows us to see now all log data of the iOS device. You should already see a lot of log items in the console.
- Open a terminal and go to the iOS Apps directory `ios-preparation-pack/Apps` and execute the following command. This will install (via side-loading) an app to your iOS device:

```
$ ios-deploy -W -b MSTG-JWT.ipa
...
[ 65%] InstallingEmbeddedProfile
[ 70%] VerifyingApplication
[ 75%] CreatingContainer
[ 80%] InstallingApplication
[ 85%] PostflightingApplication
[ 90%] SandboxingApplication
[ 95%] GeneratingApplicationMap
[100%] Installed package MSTG-JWT.ipa
```

- On your iOS device you will see now a new app called "MSTG-JWT". Click on the icon and start the app.
- Put in some dummy credentials and click on login, while you are monitoring the Console with the log files.

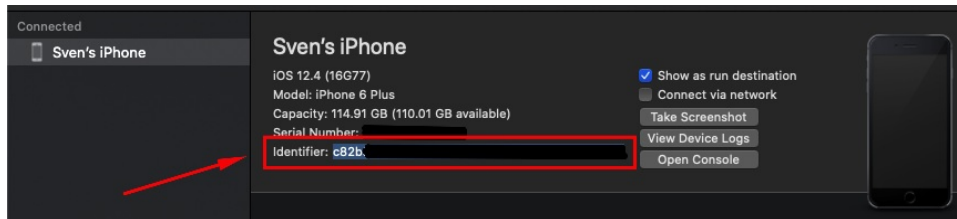
Try to find the credentials in the console! Use the search bar on the top right, to filter for app name or other keywords.

The console can be very useful during testing, to identify sensitive information leakage. Common items that you might find in the console are PII, authentication information, encryption keys or credentials that are being logged.

Exercise - Network Analysis

A very common way to analyse mobile apps is inspecting their network traffic. We will use a more targeted tool for HTTP traffic later, but for now you are going to create a `rvict1` (Remote Virtual Interface Tool). This tool is for capturing packets directly from your iOS Device and allows you to visualize all packets the device is sending and receiving.

You simply need to find your device UDID. There are various ways to do this, but as we already used Xcode we just stick to it. If the window is not open anymore, open again "Window / Devices and Simulators" and copy the identifier, which is the UDID.



An alternative on the command line to get the UDID, would be `instruments -s devices`.

The UDID is a feature of Apple's iOS devices. It is a unique identifier for a single device that is fetched from Apple servers when a user tries to activate the device using iCloud or the Setup app. This ID is also used by iTunes/Finder to detect the phone or to communicate with it while restoring the IPSW firmware.

Next we can use the command `rvictl` to create a virtual network interface, just paste your UDID:

```
$ sudo rvictl -s <UDID>
Password:
sudo: rvictl: command not found
```

In case you are getting the error above, try the following:

```
$ sudo /Library/Apple/usr/bin/rvictl -s <UDID>
Starting device <UDID> [SUCCEEDED] with interface rvi0
```

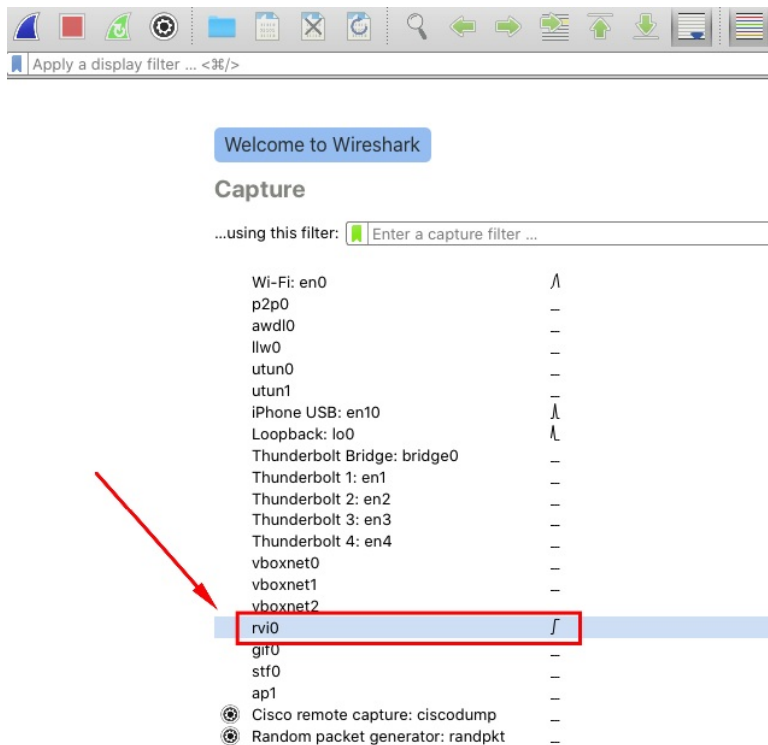
If the `rvictl` command is still not available install the following package (this might take 1-2 minutes) and try again the commands above:

```
$ open /Applications/Xcode.app/Contents/Resources/Packages/MobileDeviceDevelopment.pkg
```

You can verify the new interface `rvi0` with `ifconfig`:

```
$ ifconfig rvi0
rvi0: flags=3005<UP,DEBUG,LINK0,LINK1> mtu 0
```

Start Wireshark and select the `rvi0` interface.



Once selected you will see network packets appearing in Wireshark. Click again on the "Login" button, in the iOS app you installed. There might be a lot of packets as everything will be captured (DNS, DHCP, HTTP etc.) and it will become very fast very confusing.

Few tips when using Wireshark:

- The upper window is showing all network packets. When you select one packet you will see the details of it in the window below.
- Close all apps that are not in scope for the analysis.
- You can apply display filters. Filter for protocols you are interested in, like HTTP.
- Identify the IP address of server the app is communicating to.
- Filter traffic according to the IP (filter: `ip.addr == 192.168.0.102` views traffic with a source or destination address of 192.168.0.102).

Try to find the HTTP request that was sent to the server for the login. Can you spot the username and password?

If you ever need to capture network traffic from an iOS device on Linux, that is also possible. Instead of the `rvictl` command, you can use the following Python script: https://github.com/gh2o/rvi_capture

References

- Wireshark Display Filters - <https://wiki.wireshark.org/DisplayFilters>
- Console User Guide - <https://support.apple.com/en-sg/guide/console/welcome/mac>
- Capturing mobile phone traffic on Wireshark - <https://stackoverflow.com/questions/9555403/capturing-mobile-phone-traffic-on-wireshark/33175819#33175819>

Lab - Frida 101 (Frida-Gadget)

Training Objectives

1. Learn how to use the Frida Gadget and different Frida tools
2. Use a Frida script

Time for completing this lab: 10 min

Exercise difficulty: easy

Tools used in this section

- Frida - <https://www.frida.re/>

App

Use the following app for this exercise:

- MSTG-JWT-frida.ipa

You can find the app on your macOS in the directory `ios-preparation-pack/Apps` , wherever you unzipped the file `ios-preparation-pack.zip` .

Description

Frida supports interaction with iOS apps. You'll be able to hook and call functions inside the process and its native libraries. Your JavaScript snippets have full access to memory, e.g. to read and/or write any structured data.

In this exercise we will be using the following Frida tools:

- Frida CLI
- frida-ps

We will demonstrate you basic usage of Frida and how to load scripts.

Exercise - Frida Usage with Frida Gadget

Preparation

In case you're app is not running anymore, please execute the following steps, otherwise continue with "Frida Gadget and frida-ps".

- Connect your iOS device to your laptop via USB cable and unlock the device, so you can see the home screen.
- Go to the iOS Apps directory `ios-preparation-pack/Apps` and execute the following command in the terminal. The MSTG-JWT app was already installed, so we just need to start the app in debug mode by using the flag `-m` instead `-f -d` :

```
$ ios-deploy --bundle 'Payload/MSTG-JWT.app' -W -v -m
```

```
...
(1ldb)      command script add -s asynchronous -f fruitstrap_316f01bd160932d2bf2f95f1f142bc29b1c62dbc.autoexit_c
ommand autoexit
(1ldb)      command script add -s asynchronous -f fruitstrap_316f01bd160932d2bf2f95f1f142bc29b1c62dbc.safequit_c
ommand safequit
(1ldb)      connect
(1ldb)      run
success
2020-02-06 05:46:56.770971+0800 MSTG-JWT[16127:5985600] Frida: Listening on 127.0.0.1 TCP port 27042
(1ldb)
```

Check that you can see the last line with `Frida: Listening on 127.0.0.1 TCP port 27042`.

Frida Gadget and frida-ps

Open a new terminal and verify if Frida is up and running by using `frida-ps`:

```
$ frida-ps -U
  PID  Name
-----
16431  AppPredictionWidget
 8681  AppleCredentialManagerDaemon
16538  AssetCacheLocatorService
16501  AutoFillHelper
...
```

You should see now all running processes of the iOS device. Frida-ps is needed to identify either the Process ID (PID) or name of the process we want to inject our code in. The following flags are useful when executing `frida-ps`:

```
# Connect Frida to a device over USB and list running processes
$ frida-ps -U

# List running applications
$ frida-ps -Ua

# List installed applications
$ frida-ps -Uai
```

Use the command `frida-ps -ua`. You should be able to get the identifier (bundle ID), name and process ID (PID) of the App.

Frida CLI

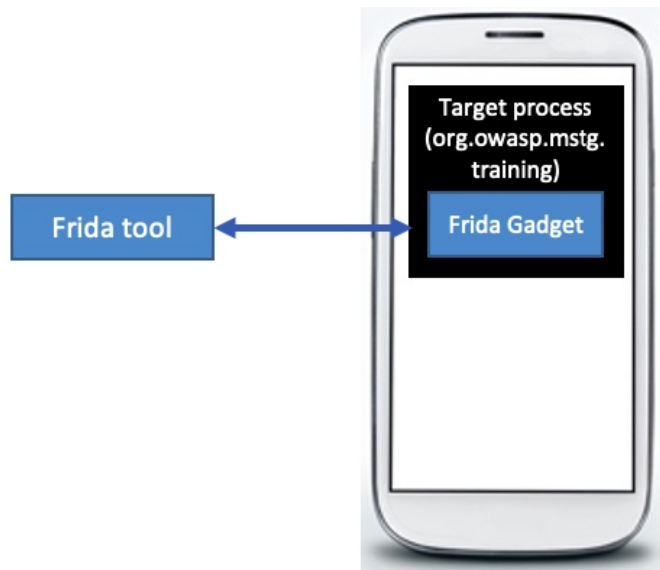
With the command `frida` the command-line interface can be started. Use the following command to attach to the running app:

```
$ frida -U -n MSTG-JWT

  ____
 / _ |  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]->
```

Frida is now connecting via USB (`-u`) to the Frida-Gadget running inside the iOS app and was injecting the Frida-Agent into the specified app name (`-n MSTG-JWT`).



So what can we do now with the interactive Frida shell? You can write commands to Frida using it's APIs. Just press the "Tab" key in the CLI to see the available commands. For example you can execute `objc.available`, which returns a boolean value specifying whether the current process has an Objective-C runtime loaded.

```
[iPhone::MSTG-JWT]-> objc.available
true
```

We can query for information of the app (e.g. class and method names) and the Frida CLI allows you to do rapid prototyping and also debugging. With the following command we can query for the bundle-id of the app:

```
[iPhone::MSTG-JWT]-> objc.classes.NSBundle mainBundle().bundleIdentifier().toString()
"org.owasp.MSTG.training"
```

The Frida CLI is case-sensitive, so always double check lower- and uppercase letters!

For now we leave the Frida CLI:

```
[iPhone::MSTG-JWT]-> exit
Thank you for using Frida!
```

Frida CLI + Scripts

Besides using the CLI of `frida`, we can also load scripts with the flag `-l`. This will load pre-defined JavaScript commands as script that Frida will execute in the context of the targeted app.

```
$ cd iOS-preparation-pack/Frida-Scripts
$ frida -U -n MSTG-JWT -l frida101.js

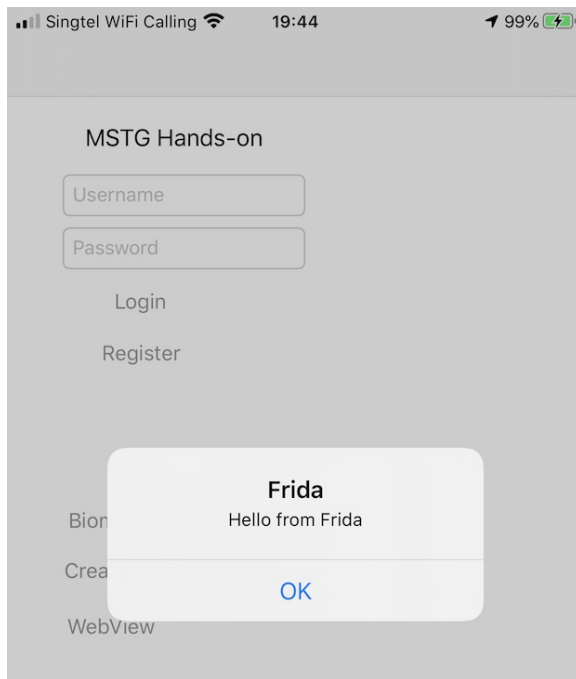
/ _ |  Frida 12.8.10 - A world-class dynamic instrumentation toolkit
| ( _ |
> _ |  Commands:
/_ |_ |    help      -> Displays the help system
. . . .    object?   -> Display information about 'object'
. . . .    exit/quit -> Exit
. . . .
. . . .    More info at https://www.frida.re/docs/home/
```

```
[iPhone::MSTG-JWT]->
```

Let's call the function `helloWorld()` :

```
[iPhone::MSTG-JWT]-> helloWorld()
```

This function is implemented in the Frida script that we just loaded. You will not see any output in the Frida console, but look at your phone!



Through Frida we are in full control of the app and can modify the app during runtime.

Modify a Frida Script

Open the file `iOS-preparation-pack/Frida-Scripts/frida101.js` in a text editor of your choice. You can see in the script the function `helloWorld()` that we just called.

Open the file `extension.js` also in an editor and copy both functions into the `frida101.js` file, before the `helloWorld()` function.

This Frida script offers us now 3 different functions:

- `appInfo()` - Dump key app paths and metadata
- `infoLookup()` - Helper function for `appInfo()`
- `helloWorld()` - Print a Hello World message into an iOS app

Once you save the file, the script will automatically be reloaded in the Frida CLI, so you can call now `appInfo()` . Give it a try!

```
$ [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"  
}
```

This will print a lot of useful meta-information about the app installation.

Key Takeaways:

- `frida-ps` allows you to identify PID, name and bundle-id of running and installed apps.
- Frida CLI allows you to do rapid prototyping and also debugging of scripts.
- The flag `-i` in the Frida command can be used to load pre-defined scripts.

References

- Install Frida on iOS - <https://www.frida.re/docs/ios/>
- Frida JavaScript API (Objective-C) - <https://frida.re/docs/javascript-api/#objc>
- Frida - <https://frida.re>
 - `frida-ps` - <https://www.frida.re/docs/frida-ps>
 - `frida` - <https://www.frida.re/docs/frida-cli>

Lab - Sensitive Data in Local Storage

Training Objectives

1. Usage of objection
2. Identify sensitive information stored in the iOS App

Time for completing this lab: 15 min

Exercise difficulty: easy

Tools used in this section

- objection - <https://github.com/sensepost/objection>

App

Use the following app for this exercise:

- MSTG-JWT-frida.ipa

You can find the app on your macOS in the directory `ios-preparation-pack/Apps` , wherever you unzipped the file `ios-preparation-pack.zip` .

Description

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:

- NSData
- NSUserDefaults
- Core Data

All of these storage techniques do not protect the confidentiality of data when used in the default settings.

In the following exercise we will analyse the various ways and formats of how data can be stored in iOS.

Exercise

Preparation

In case you're app is not running anymore, please execute the following steps, otherwise continue with the next section.

- Connect your iOS device to your laptop via USB cable and unlock the device, so you can see the home screen.
- Go to the iOS Apps directory `ios-preparation-pack/Apps` and execute the following command in the terminal. The MSTG-JWT app was already installed, so we just need to start the app in debug mode by using the flag `-m` instead `-f -d` :

```
$ ios-deploy --bundle 'Payload/MSTG-JWT.app' -W -v -m
...
(lldb)      command script add -s asynchronous -f fruitstrap_316f01bd160932d2bf2f95f1f142bc29b1c62dbc.autoexit_c
ommand autoexit
(lldb)      command script add -s asynchronous -f fruitstrap_316f01bd160932d2bf2f95f1f142bc29b1c62dbc.safequit_c
ommand safequit
(lldb)      connect
(lldb)      run
success
2020-02-06 05:46:56.770971+0800 MSTG-JWT[16127:5985600] Frida: Listening on 127.0.0.1 TCP port 27042
(lldb)
```

Check that you can see the last line with `Frida: Listening on 127.0.0.1 TCP port 27042`. The app is now installed on the iOS device and running in debug mode with the Frida gadget started.

The flags we are using tells ios-deploy to:

```
- '--bundle': install the app bundle from the specified path
- '-W': ignore Wi-Fi
- '-m': no installation
- '-v': enable verbose output
```

After the app is installed you can skip `-d` and use the flag `-m` in ios-deploy, then the app will not be installed again but directly started and debugged.

Identify Sensitive Data

- When you were just starting the app, it's in the so called "early instrumentation mode" and you will only see a white screen as the Frida Gadget is waiting for a connection. Open another terminal and start objection to connect to the Frida Gadget:

```
$ objection explore
Using USB device `iPhone`
Agent injected and responds ok!

  _ _ _ _ _
 | | | | | | | | | |
 | . . | | ^-| -| | . | |
 |__|__| |__|__| |__|__|
    |__(object)inject(ion) v1.8.4

Runtime Mobile Exploration
by: @leonjza from @sensepost

[tab] for command suggestions
org.owasp.MSTG.training on (iPhone: 13.3) [usb] #
```

If you see the prompt above you can verify that it's working as objection could connect to the Frida-Gadget and can read the bundle name (org.owasp.MSTG.training) and also the iOS version on your iOS device (13.3 in the example above).

- Go back to the app and click on the button "Create Sensitive Data". This has created now several different files within the app package.
- Also log into the app by using the following credentials:
 - User: `foo`
 - Password: `123`

Your goal is to find all data created by the app and identify if they contain sensitive data or not.

Using objection

- In objection type in the command `env` to get a listing of the directories you want to explore:

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

Name	Path
BundlePath	/private/var/containers/Bundle/Application/0D2DC95D-9E0B-40D0-B589-D8C58E69B934/MSTG-JWT.app
CachesDirectory	/var/mobile/Containers/Data/Application/A098A850-823D-43FA-A36F-E6A0E1F11C34/Library/Caches
DocumentDirectory	/var/mobile/Containers/Data/Application/A098A850-823D-43FA-A36F-E6A0E1F11C34/Documents
LibraryDirectory	/var/mobile/Containers/Data/Application/A098A850-823D-43FA-A36F-E6A0E1F11C34/Library

- Try to find the directory where an app usually stores data and navigate to it in objection:

```
org.owasp.MSTG.training on (iPhone: 13.3) [usb] $ cd <path>
```

- Once you identified a plist file you can execute `cat` in objection to print it's content:

```
org.owasp.MSTG.training on (iPhone: 13.3) [usb] $ ios plist cat <plist file name>
```

- Once you identified a sqlite database you can connect to it directly in objection:

```
org.owasp.MSTG.training on (iPhone: 13.3) [usb] $ sqlite connect <name>.sqlite
Caching local copy of database file...
Downloading /var/mobile/Containers/Data/Application/A098A850-823D-43FA-A36F-E6A0E1F11C34/Documents/Database.sqlite to /var/folders/nd/6h7d2qjd7pz8lhkzndwh54sm0000gp/T/tmpu43loskh.sqlite
Streaming file from device...
Writing bytes to destination...
Successfully downloaded /var/mobile/Containers/Data/Application/A098A850-823D-43FA-A36F-E6A0E1F11C34/Documents/Database.sqlite to /var/folders/nd/6h7d2qjd7pz8lhkzndwh54sm0000gp/T/tmpu43loskh.sqlite
Validating SQLite database format
Connected to SQLite database at: Database.sqlite
SQLite @ Database.sqlite >
```

- You can print the schema and tables of the database:

```
SQLite @ Database.sqlite > .schema
SQLite @ Database.sqlite > .tables
```

Write the `select` statement to dump the content of the table you found in the sqlite database.

What sensitive information could you find?

Check also the objection documentation to verify if you have used all options offered by objection to analyse stored data on iOS: <https://github.com/sensepost/objection/wiki/Screenshots>

References

- Using objection - <https://github.com/sensepost/objection/wiki/Using-objection>
- SQLite CLI - <https://sqlite.org/cli.html>

Lab - iOS Biometric Authentication

Training Objectives

1. Usage of objection
2. Bypass Touch ID / Face ID to get access to the protected page

Time for completing this lab: 15 min

Exercise difficulty: medium

Tools used in this section

- objection

App

Use the following apps for this exercise:

- MSTG-JWT-frida.ipa
- KeychainAuthenticator-frida.ipa

You can find the app on your macOS in the directory `MSTG-Handson/Apps` , wherever you unzipped the file `MSTG-Handson.zip` .

Description

In this exercise you will be exploring how to bypass biometric authentication on iOS.

The two different apps in this exercise will demonstrate you the difference and which one can be bypassed.

Preparation

Verify if Touch ID / Face ID is configured on your device. You will also need to set a passcode!

- Go to Settings
- Click on "Touch ID & Passcode" / "Face ID & Passcode"
- Add your fingerprint / facial data
- If no passcode has been set, you need to set one now. If a previous fingerprint / facial data has been stored it will be deleted!

Exercise 1

Preparation

In case your app is not running anymore, please execute the following steps, otherwise continue with the next section.

- Connect your iOS device to your laptop via USB cable and unlock the device, so you can see the home screen.
- Go to the iOS Apps directory `ios-preparation-pack/Apps` and execute the following command in the terminal. The MSTG-JWT app was already installed, so we just need to start the app in debug mode by using the flag `-m` instead of `-d`:

```
$ ios-deploy --bundle 'Payload/MSTG-JWT.app' -W -v -m
...
(lldb) command script add -s asynchronous -f fruitstrap_316f01bd160932d2bf2f95f1f142bc29b1c62dbc.autoexit_c
ommand autoexit
(lldb) command script add -s asynchronous -f fruitstrap_316f01bd160932d2bf2f95f1f142bc29b1c62dbc.safequit_c
ommand safequit
(lldb) connect
(lldb) run
success
2020-02-06 05:46:56.770971+0800 MSTG-JWT[16127:5985600] Frida: Listening on 127.0.0.1 TCP port 27042
(lldb)
```

Check that you can see the last line with `Frida: Listening on 127.0.0.1 TCP port 27042`. The app is now installed on the iOS device and running in debug mode with the Frida gadget started.

The flags we are using tell `ios-deploy` to:

```
- '--bundle': install the app bundle from the specified path
- '-W': ignore Wi-Fi
- '-m': no installation
- '-v': enable verbose output
```

After the app is installed you can skip `-d` and use the flag `-m` in `ios-deploy`, then the app will not be installed again but directly started and debugged.

The last line should be about Frida and that it's listening on port 27042. The app is now installed on the iOS device and running in debug mode with the Frida gadget started.

Bypass Biometric Authentication

- When you were just starting the app, it's in the so called "early instrumentation mode" and you will only see a white screen as the Frida Gadget is waiting for a connection. Open another terminal and start objection to connect to the Frida Gadget:

```
$ objection explore
Using USB device `iPhone`
Agent injected and responds ok!

  _ _ _ _ _
 | | | | | | | | |
 | . | . | | - | _ | | . | |
 | _ | | | | | | | |
   | |(object)inject(ion) v1.8.4

Runtime Mobile Exploration
by: @leonjza from @sensepost

[tab] for command suggestions
org.owasp.MSTG.training on (iPhone: 13.3) [usb] $
```

If you see the prompt above you can verify that it's working as objection could connect to the Frida-Gadget and can read the bundle name (org.owasp.MSTG.training) and also the iOS version on your iOS device (13.3 in the example above).

Bypassing

- Validate that the function in the app is working properly on your phone first.
 - Click on "Biometric Authentication"
 - Click on "Get authentication Type". According to hardware feature of your phone it should either show Face ID or Touch ID.
 - Click on "Evaluate bioMetric" in the iOS app and authenticate with your fingerprint/face. You should see the following screen after successful authentication:

< Back

Biometric Test page

evaluating.....
successfully completed

Get authentication Type

Evaluate bioMetric

- When typing in `ios` and pressing space in objection you will see different sub commands that are available for iOS apps. Find the command in objection to apply the biometric bypass by using the objection shell.

```
org.owasp.MSTG.training on (iPhone: 13.3) [usb] # ios bundles
bundles      Work with iOS Bundles
cookies      Work with shared cookies
heap         Commands to work with the iOS heap
hooking      Commands used for hooking methods in iOS
```

- Once you applied the bypass do the authentication again and use a finger that wasn't enrolled, or ask your table neighbor to authenticate in Face ID for you :-)) and see what is happening.

Answer the following questions

- How and why is the bypass working? Have a look at <https://github.com/sensepost/objection/wiki/Understanding-the-iOS-Biometrics-Bypass>
- How could malware or an attacker exploit this in the real world?

Exercise 2

Preparation

- Install and start the App KeychainAuthenticator-frida.ipa (unzip and ios-deploy)
- Start objection
- This app can add a value in the keychain and read it again through biometric authentication. Test if it's working

properly with Touch/Face ID:

- Click on the button "Add"
- Click on "Read" button in the iOS app and authenticate with your fingerprint / face.

If it's working the stored credentials will show up on the bottom of the app.

Bypassing

- Try to bypass the biometric authentication by using objection like you did before.

Is it still working? If not, what do you think is this app doing differently? To understand it, consult the best practice from Apple:

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

Lab - SSL Pinning

Training Objectives

1. Usage of objection
2. Deactivate SSL Pinning in Snapchat

Time for completing this lab: 10 min

Exercise difficulty: easy

Tools used in this section

- objection - <https://github.com/sensepost/objection>
- Burp Suite - <https://portswigger.net/burp/communitydownload>

App

Use the following app for this exercise:

- Snapchat.ipa

You can find the app in the VM in the directory `~/MSTG-Handson/Apps`.

The app was already installed in your Corellium instance.

Frida Setup

Your iPhone instance has two interfaces. One for SSH starting with 10.11.1.X and one that we use for Frida which starts with 10.11.0.X. If you're IP address is 10.11.1.5 for SSH, it's 10.11.0.5 to use with Frida.

- Check if the frida server is running on your iPhone in Corellium, by executing the following command in the Kali VM:

Replace the IP with the IP of your corellium instance.

```
$ frida-ps -H 10.11.0.X
PID  Name
---  -
445  AppPredictionWidget
65   AppleCredentialManagerDaemon
...
```

- If you get PIDs and names of the running processes, please move on to the exercise section.
- If you don't get any output, login and verify if the frida process is running on your iPhone:

```
$ ssh root@10.11.1.X # Password is "alpine"
iPhone:~ root$ ps a | grep -i frida
```

If you don't get any output, frida was not started. Start the frida server with the flag `-i` and the IP address that you are getting for the en0 interface. In the example below the IP address is 10.11.0.2, but you need to use the IP address of your instance:

```
iPhone:~ root$ ifconfig en0
en0: flags=8863<UP,BROADCAST,SMART,RUNNING,SIMPLEX,MULTICAST> mtu 1500
    options=400<>
    ether 7c:c5:37:a7:02:6a
    inet6 fe80::1040:d3c2:9300:2726%en0 prefixlen 64 secured scopeid 0x9
    inet 10.11.0.2 netmask 0xfffffc00 broadcast 10.11.3.255
    ...
iPhone:~ root$ ./frida-server-12.8.9-ios-arm64 -i 10.11.0.X
```

- Check again in Kali VM if frida is now reachable:

```
$ frida-ps -H 10.11.0.X
PID  Name
---  ---
445  AppPredictionWidget
65   AppleCredentialManagerDaemon
...
```

If it's still not working, please let the trainer know.

Exercise - Snapchat

- Open Corellium (<https://sven.corellium.com/>) in a browser (Safari or Chrome) on your macOS and login.
- Make sure that the proxy in iOS is configured and that you can intercept HTTP traffic in Burp from your iOS device. For example open Safari and browse to wikipedia.org.
- Open your iOS device in Corellium and check that the app "Snapchat" is installed and start it.
- Open a terminal, to first identify the name of the app:

```
$ frida-ps -H 10.11.0.X | grep -i snap
1473  Snapchat
```

- Once you know the app name, start objection and attach to Snapchat (this might take a few seconds, as it's injected over network):

```
$ objection -N -h 10.11.0.X -g Snapchat explore
Using networked device @`10.11.0.X:27042`
Agent injected and responds ok!

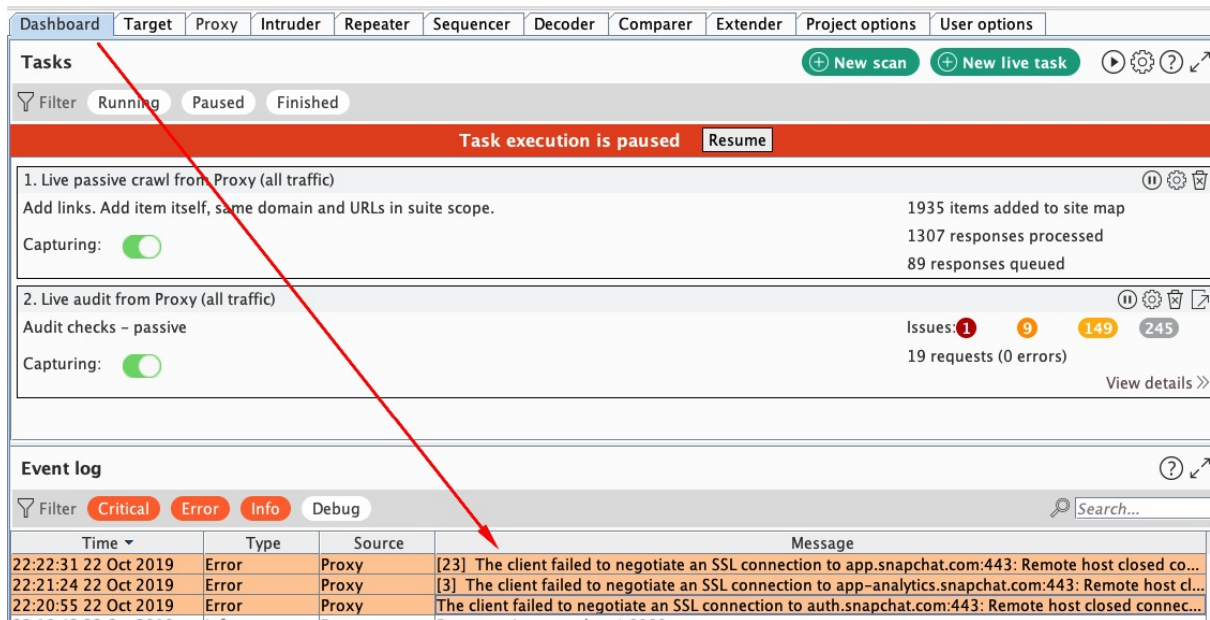
  _ _ _ _
  | | | | | | | | | |
  | . | . | | - | | | . | |
  | | | | | | | | | |
  | | | (object)inject(ion) v1.8.4

Runtime Mobile Exploration
by: @leonjza from @sensepost

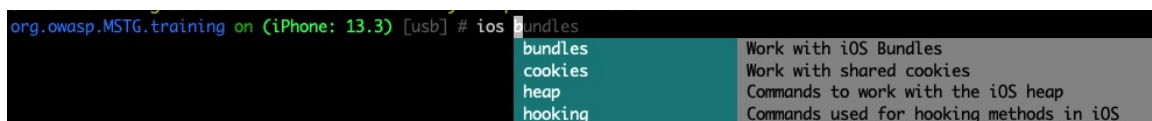
[tab] for command suggestions
com.toyopagroup.picaboo on (iPhone: 12.4.5) [net] $
```

If you see the prompt as above you can verify that is' working as objection could connect and can read the bundle name (`com.toyopagroup.picaboo`) and also the iOS version on your iOS device (12.4.5 in the example above)

- In Snapchat key in some dummy username and password and click on "Log in".
- The login request can not be intercepted in Burp, due to SSL Pinning and we will only see error messages in the event log.



- When typing in `ios` in objection and pressing space in objection you will see different sub commands that are available for iOS apps. Find the command in objection to apply the SSL Pinning bypass by using the objection shell.



- Once the bypass is applied, click again on "Log In" in the Snapchat App and verify if you can see the HTTP login request in Burp.

What is happening in the objection console and the mobile app when you try to login now?

If you are interested, you can explore the code in objection that is applying the SSL Pinning Bypass:

<https://github.com/sensepost/objection/blob/master/agent/src/ios/pinning.ts>

Lab - Stateless Authentication

Training Objectives

1. Analysing JWT for vulnerabilities
2. Cracking the Signing Key
3. Resigning JWT

Time for completing this lab: 15 min

Exercise difficulty: easy

Tools used in this section

- jwtcrack - <https://github.com/Sjord/jwtcrack>
- Burp Suite - <https://portswigger.net/burp/communitydownload>

App

Use the following app for this exercise:

- MSTG-JWT.ipa

You can find the app in the VM in the directory `~/MSTG-Handson/Apps`.

This app was already installed in your Corellium instance.

Description

In the following exercise we will analyse a JWT, crack the signing key and resign it to access data of other users.

Exercise - Stateless Authentication

- Open Corellium (<https://sven.corellium.com/>) in a browser (Safari or Chrome) on your macOS and login.
- Open your iOS device in Corellium and check that the app "MSTG-JWT" is installed and start it.
- Ensure that Burp is configured as proxy on your iOS device and that you can see the HTTP requests in Burp on the Kali VM. You can test it by opening Safari and opening any website. You should see the traffic in Burp on your Kali VM.
- Go back to the iOS App MSTG-JWT and click on "Register" and create your own account. The Email you type in, is your username to login later.
- Go back, and login. After login click on the button "Create To-Do" and create a few to-dos.
- Go back and click now on the button "Query for ToDos". You should get a listing of the to-dos you just created.
- Open Burp and identify the requests that were sent to add the ToDos and that were querying for it. You should see the `Authorization` header with the `Bearer` schema. The JWT is everything after "Bearer" and is Base-64 encoded:

#	Host	Method	URL	Params	Edited	Status
58	http://3.0.101.99:8080	GET	/todos			200
57	http://3.0.101.99:8080	POST	/todos	✓		201
56	http://3.0.101.99:8080	GET	/todos			200

RequestResponse


RawHeadersHex

GET /todos HTTP/1.1
Host: 3.0.101.99:8080
Connection: close
Accept: application/json
User-Agent: MSTG-JWT/1.0 (org.owasp.MSTG.training; build:1; iOS 12.4.0) Alamofire/4.5.1
Authorization: Bearer
eyJhbGciOiJIUzI1NiJ9.eyJ1c2VyX2lkIjo1LCJlbWVpbCI6InN2ZW5AZ29vZ2xlbWVpbC5jb20iLCJleHAiOiJlNzE0NDc5MDF9.H_3rR-Ro5Az_wxEgNoIP2M9ZQaAx8HNhnsLTYnql6M
Accept-Language: en-SG;q=1.0, zh-Hans-SG;q=0.9
Accept-Encoding: gzip, deflate

- Let's try to crack the signing key of the JWT by using the `crackjwt` Python script in Kali Linux. Execute the following command and paste in the JWT we found in Burp:

```
$ cd ~/Tools/jwtcrack
$ python3 crackjwt.py <JWT Token> /usr/share/wordlists/metasploit/unix_passwords.txt
```

- You should have successfully cracked the key by now!
- Now it's time to resign the JWT and get access to the ToDos of another user account!
- Go to jwt.io in your browser and paste the JWT from your account in. You will see the decoded payload and claims:


Debugger Libraries Introduction Ask Get a T-shirt!
Crafted by Auth0

Encoded
PASTE A TOKEN HERE

Decoded
EDIT THE PAYLOAD AND SECRET

```
eyJhbGciOiJIUzI1NiJ9.eyJ1c2VyX2lkIjo1LCJlbWVpbCI6InN2ZW5AZ29vZ2xlbWVpbC5jb20iLCJleHAiOiJlNzE0NDc5MDF9.H_3rR-Ro5Az_wxEgNoIP2M9ZQaAx8HNhnsLTYnql6M
```

HEADER: ALGORITHM & TOKEN TYPE

```
{
  "alg": "HS256"
}
```

PAYLOAD: DATA

```
{
  "user_id": 5,
  "email": "sven@googlemail.com",
  "exp": 1571447901
}
```

VERIFY SIGNATURE

```
HMACSHA256(
  base64UrlEncode(header) + "." +
  base64UrlEncode(payload),
  your-256-bit-secret
) ☐ secret base64 encoded
```

- Change the value of `user_id` to `2` and put in the secret key you cracked. This will automatically create a new encoded JWT. Copy this new encoded JWT into the clipboard.

The image shows the JWT.io website interface. On the left, under 'Encoded', a JWT token is pasted: `eyJhbGciOiJIUzI1NiJ9.eyJ1c2VyX2lkIjoyLCJlbWp6bCI6InN2ZW5AZ29vZ2xlbWp6b20iLCJleHAiOiE1NzE0NDc5MDc5LjFmdsFKQeuG7pt9HvJuxzDX2_q1JAjPABqE2IY16YKY`. On the right, under 'Decoded', the token is broken down into three parts: Header, Payload, and Signature. The Header is `{ "alg": "HS256" }`. The Payload is `{ "user_id": 2, "email": "jven@googlemail.com", "exp": 157447981 }`. The Signature is `HMACSHA256(base64UrlEncode(header) + "." + base64UrlEncode(payload), secretkey)`. A red arrow points from the 'secretkey' field in the signature section to the 'Signature Verified' status on the left. A 'SHARE JWT' button is at the bottom right.

- Go back to Burp and find the request you were querying the To-Dos and sent it to the Repeater.

The image shows the Burp Suite interface. The 'HTTP history' tab is active, showing a list of requests. The selected request is #58, a GET request to `http://3.0.101.99:8080/todos`. A right-click context menu is open over the selected request, with a red arrow pointing to the 'Send to Repeater' option. The menu includes options like 'Scan', 'Send to Intruder', 'Send to Repeater', 'Send to Sequencer', 'Send to Comparer', 'Send to Decoder', 'Show response in browser', 'Request in browser', 'Engagement tools', 'Copy URL', 'Copy as curl command', 'Copy to file', 'Save item', 'Convert selection', 'Cut', 'Copy', 'Paste', 'Message editor documentation', and 'Proxy history documentation'.

- Send the request in Repeater and you should see in the response the ToDo you keyed in earlier, as this is the data tied to your user id.



- What can you do to protect yourself against such attacks?
- What is the root cause of it?
- Look at the decoded JWT, are there more vulnerabilities?

Lab - Bypass Jailbreak Detection

Training Objectives

1. Usage of objection
2. Bypass Jailbreak Detection Mechanisms

Time for completing this lab: 10 min

Exercise difficulty: easy

Tools used in this section

- Objection
- Frida

App

The following app is used for this exercise:

- DVIA-v2-swift.ipa

You can find the app in the VM in the directory `~/MSTG-Handson/Apps`.

This app was already installed in your Corellium instance.

Frida Setup

Your iPhone instance has two interfaces. One for SSH starting with 10.11.1.X and one that we use for Frida which starts with 10.11.0.X. If you're IP address is 10.11.1.5 for SSH, it's 10.11.0.5 to use with Frida.

- Check if the frida server is running on your iPhone in Corellium, by executing the following command in the Kali VM:

Replace the IP with the IP of your corellium instance.

```
$ frida-ps -H 10.11.0.X
PID  Name
---  -
445  AppPredictionWidget
65   AppleCredentialManagerDaemon
...
```

- If you get PIDs and names of the running processes, please move on to the exercise section.

If you don't get any output, frida was not started. Start the frida server with the flag `-i` and the IP address that you are getting for the en0 interface. In the example below the IP address is 10.11.0.2, but you need to use the IP address of your instance:

```
iPhone:~ root$ ifconfig en0
en0: flags=8863<UP,BROADCAST,SMART,RUNNING,SIMPLEX,MULTICAST> mtu 1500
options=400<>
```

```
ether 7c:c5:37:a7:02:6a
inet6 fe80::1040:d3c2:9300:2726%en0 prefixlen 64 secured scopeid 0x9
inet 10.11.0.2 netmask 0xfffffc00 broadcast 10.11.3.255
...
iPhone:~ root$ ./frida-server-12.8.9-ios-arm64 -l 10.11.0.X
```

- Check again in Kali VM if frida is now reachable:

```
$ frida-ps -H 10.11.0.X
PID  Name
---  ---
445  AppPredictionWidget
65   AppleCredentialManagerDaemon
...
```

If it's still not working, please let the trainer know.

Exercise - DVIA-v2

- Open Corellium (<https://sven.corellium.com/>) in a browser (Safari or Chrome) on your macOS and login.
- Open your iOS device in Corellium and check that the app "DVIA-v2" (grey symbol) is installed and start it. Keep the app in the foreground, otherwise you will get an error when attaching later with objection.
- Open a terminal, to first identify the bundle-id. We need the bundle-id for objection to hook into the app (this might take a few seconds as it's done over the network):

```
$ frida-ps -H 10.11.0.X | grep -i dvia
557  DVIA-v2
$ objection -N -h 10.11.0.X -g DVIA-v2 explore
Using networked device @`10.11.0.X:27042`
Agent injected and responds ok!

  _ _ _ _ _
 | | | | | | | | | |
 | . . . | - | - | . |
 | | | | | | | | | |
   |__(object)inject(ion) v1.8.4

Runtime Mobile Exploration
by: @leonjza from @sensepost

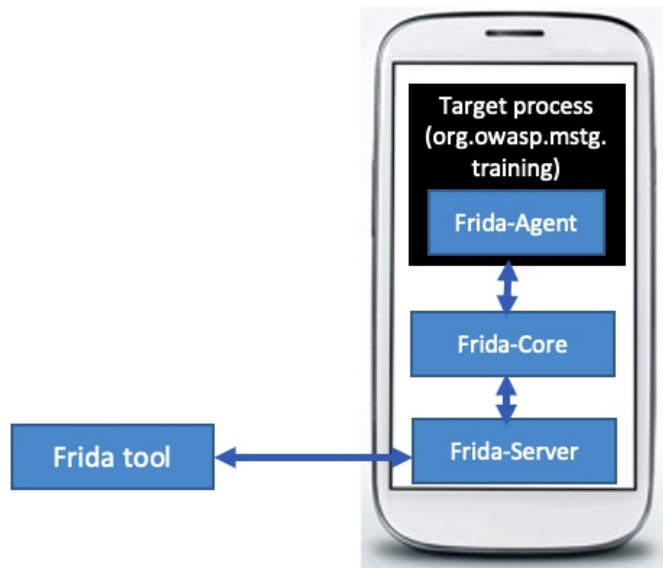
[tab] for command suggestions
....highaltitudehacks.DVIAswiftv2 on (iPhone: 12.4.5) [net] #
```

- If you see the prompt as above you can verify that its working as objection could connect and can read the bundle name and also the iOS version on your iOS device (12.4.5 in the example above).

The flags we are using tells objection to:

```
- `N`: Connect using a network connection instead of USB.
- `h`: Specify the host or IP.
- `g`: Name of the Frida Gadget/Process to connect to.
```

Objection is now connecting to the running Frida server and injecting the Frida Agent into the DVIA-v2 app.



- In the DVIAv2 App, click on the top left corner for the menu and select **Jailbreak Detection**.
- There are five different tests. Execute "Jailbreak Test 1" and "Jailbreak Test 2". Both tests will show "Device is Jailbroken".
- When typing in `ios` in objection and pressing space in objection you will see different sub commands that are available for iOS apps. Find the command in objection to apply the Jailbreak bypass by using the objection shell.

```

org.owasp.MSTG.training on (iPhone: 13.3) [usb] # ios bundles
bundles      Work with iOS Bundles
cookies      Work with shared cookies
heap         Commands to work with the iOS heap
hooking      Commands used for hooking methods in iOS

```

- Click again "Jailbreak Test 1" and "Jailbreak Test 2". Both Jailbreak tests still work. Look into the objection console and you can see that objection could only detect a few, but not all of jailbreak checks:

```

....highaltitudehacks.DVIAswiftv2 on (iPhone: 12.4.5) [net] # (agent) [k89mm6t7z1l] fileExistsAtPath: check for
/Applications/Cydia.app was successful with: 0x1, marking it as failed.
(agent) [k89mm6t7z1l] fileExistsAtPath: check for /bin/bash was successful with: 0x1, marking it as failed.
(agent) [k89mm6t7z1l] fileExistsAtPath: check for /usr/sbin/sshd was successful with: 0x1, marking it as failed
.
(agent) [k89mm6t7z1l] fileExistsAtPath: check for /etc/apt was successful with: 0x1, marking it as failed.
(agent) [k89mm6t7z1l] fileExistsAtPath: check for /Applications/Cydia.app was successful with: 0x1, marking it
as failed.
(agent) [k89mm6t7z1l] fileExistsAtPath: check for /bin/bash was successful with: 0x1, marking it as failed.
(agent) [k89mm6t7z1l] fileExistsAtPath: check for /usr/sbin/sshd was successful with: 0x1, marking it as failed
.
(agent) [k89mm6t7z1l] fileExistsAtPath: check for /etc/apt was successful with: 0x1, marking it as failed.

```

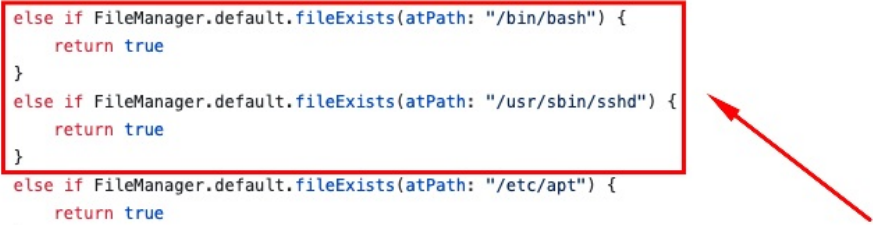
- Let's have a look at the source code and analyse "Jailbreak Test 1", which is available on Github:

<https://github.com/prateek147/DVIA-v2/blob/master/DVIA-v2/DVIA-v2/Vulnerabilities/Jailbreak%20Detection/Controller/JailbreakDetectionViewController.swift>

```

176
177 func isJailbroken() -> Bool {
178     #if !SIMULATOR
179         if FileManager.default.fileExists(atPath: "/Applications/Cydia.app") {
180             return true
181         }
182         else if FileManager.default.fileExists(atPath: "/Library/MobileSubstrate/MobileSubstrate.dylib") {
183             return true
184         }
185         else if FileManager.default.fileExists(atPath: "/bin/bash") {
186             return true
187         }
188         else if FileManager.default.fileExists(atPath: "/usr/sbin/sshd") {
189             return true
190         }
191         else if FileManager.default.fileExists(atPath: "/etc/apt") {
192             return true
193         }
194
195         var error: Error?
196         let stringToBeWritten = "This is a test."
197         do {
198             try stringToBeWritten.write(toFile: "/private/jailbreak.txt", atomically: true, encoding: .utf8)
199         } catch let err {
200             error = err
201         }
202
203         if error == nil {
204             //Device is jailbroken
205             return true
206         }
207         else {
208             try? FileManager.default.removeItem(atPath: "/private/jailbreak.txt")
209         }
210         if UIApplication.shared.canOpenURL(URL(string: "cydia://package/com.example.package")) {
211             //Device is jailbroken
212             return true
213         }
214     #endif
215     //All checks have failed. Most probably, the device is not jailbroken
216     return false
217 }

```



- The function `isJailbroken()` is checking for five different files if they are present and objection was only able to bypass three of them. The files `/bin/bash` and `/usr/sbin/sshd` were detected by the app and concluded the device is jailbroken.
- Type `exit` into the objection console and start Frida with a generic Jailbreak bypass script:

```

....highaltitudehacks.DVIAswiftv2 on (iPhone: 12.4.5) [usb] $ exit
$ cd ~/MSTG-Handson/Frida-Scripts
$ frida -H 10.11.X.X -l bypass-jb-dviav2-generic.js DVIA-v2

```

- Click again "Jailbreak Test 1" and "Jailbreak Test 2". This time the message will show you "Device is not Jailbroken"!
- Look at the Frida console and open the Frida script:

```

$ cd ~/MSTG-Handson/Frida-Scripts
$ sublime_text bypass-jb-dviav2-generic.js

```

- What does the script actually do and what is being "bypassed"? How many different jailbreak detection mechanisms that are being bypassed can you find?

Lab - Bypass "Piracy Detection"

Training Objectives

1. Use existing Frida scripts
2. Bypass a client side security check in an iOS app.

Time for completing this lab: 15 min

Exercise difficulty: medium

Tools used in this section

- Frida - <https://www.frida.re/>

App

Use the following app for this exercise:

- DamnVulnerableiOSApp.ipa

You can find the app in the VM in the directory `~/MSTG-Handson/Apps`.

This app was already installed in your Corellium instance.

Description

Frida supports interaction with iOS apps. You'll be able to hook and call functions inside the process and its native libraries. Your JavaScript snippets have full access to memory, e.g. to read and/or write any structured data.

In this exercise we will be using the following Frida tools:

- Frida CLI
- frida-ps

We will demonstrate you how to bypass a check in an iOS app, that is verifying if the app was pirated.

Frida Setup

- Check if the frida server is running on your iPhone in Corellium, by executing the following command in the Kali VM:

```
$ frida-ps -H 10.11.X.X
PID  Name
---  -
445  AppPredictionWidget
65   AppleCredentialManagerDaemon
...
```

- If you get PIDs and names of the running processes, please move on to the exercise section.

- If you don't get any output, login and verify if the frida process is running on your iPhone:

```
$ ssh root@10.11.X.X # Password is "alpine"
iPhone:~ root$ ps a | grep -i frida
```

If you don't get any output, frida was not started. Start the frida server with the flag `-l` :

```
iPhone:~ root$ ./frida-server-12.8.9-ios-arm64 -l 10.11.X.X
```

- Check again in Kali VM if frida is now reachable:

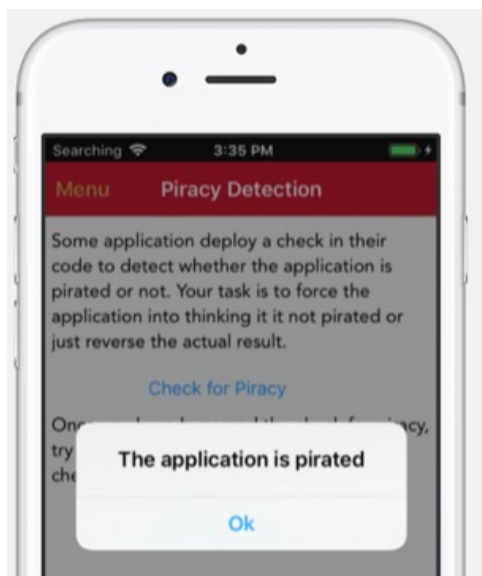
```
$ frida-ps -H 10.11.X.X
PID  Name
---  ---
445  AppPredictionWidget
65   AppleCredentialManagerDaemon
...
```

If it's still not working, please let the trainer know.

Exercise - DamnVulnerableiOSApp

Preparation

- Open Corellium (<https://sven.corellium.com/>) in a browser (Safari or Chrome) on your macOS and login.
- Open your iOS device in Corellium and check that the app "DVIA" (red icon) is installed and start it. Keep the app in the foreground, otherwise you will get an error when attaching later with objection.
- In the DVIA App, click on the top left corner for the menu and select **Piracy Detection** . Click on the upper button and you will get the following message.



Most likely this app is verifying if it's run on a jailbroken device and has maybe several other checks.

Our mission is to reverse this check!

Frida CLI + Scripts

We already know that we can use Frida scripts and load them with the flag `-l` to inject them into the target app. In the VM there are already plenty of pre-defined scripts that are ready for use.

Open the existing scripts in sublime, we need to edit a few of them:

```
$ cd ~/MSTG-Handson/Frida-Scripts/iOS
$ sublime_text .
```

First we need to identify the class name we want to hook into. We can use the script `find-classes.js` which will list all classes from the app:

```
$ frida -H 10.11.X.X DamnVulnerableIOSApp -l find-classes.js
...
[*] Classes found: 11248
[*] Completed: Find Classes
[Remote::DamnVulnerableIOSApp]-> exit
```

Afterwards exit the Frida console. In the output search for keywords we already know from using the app like "Piracy".

Once you have identified a class you want to investigate further, replace the placeholder "YOUR_CLASS_HERE" with your class in the script `show-all-methods-of-specific-class.js`.

Execute Frida again to get a listing of available methods for this class:

```
$ frida -H 10.11.X.X DamnVulnerableIOSApp -l show-all-methods-of-specific-class.js
...
[*] Functions Found:657
[*] Completed: Find All Methods of a Specific Class
[Remote::DamnVulnerableIOSApp]-> exit
```

Afterwards exit the Frida console. In the output search again for keywords like "Piracy" or "Pirated".

Note: In the output you can find plus and minus signs in front of the methods. (+) stands for class method and (-) for instance method.

Once you have identified a method, replace the placeholder "YOUR_CLASS_HERE" with your class and the placeholder "YOUR_METHOD_HERE" with the method in the `show-modify-method-return-value.js`.

```
$ frida -H 10.11.X.X DamnVulnerableIOSApp -l show-modify-method-return-value.js
```

Press the button "Check for Privacy" again. If you couldn't get the message "The application is not pirated" yet, you might have the wrong class or method name. Try again!

What is the script `show-modify-method-return-value.js` doing?

References

- Install Frida on iOS - <https://www.frida.re/docs/ios/>
- Frida JavaScript API (Objective-C) - <https://frida.re/docs/javascript-api/#objc>
- Frida - <https://frida.re>
 - `frida-ps` - <https://www.frida.re/docs/frida-ps>
 - `frida` - <https://www.frida.re/docs/frida-cli>

Lab - Analysing Memory on iOS

Training Objectives

1. Usage of objection
2. Identify sensitive information in the memory of an iOS App

Time for completing this lab: 10 min

Exercise difficulty: easy

Tools used in this section

- Objection
- strings

App

Use the following app for this exercise:

- DVIA-v2-swift-frida-codesigned.ipa

You can find the app in the VM in the directory `~/MSTG-Handson/Apps`.

Exercise - iOS Memory

- Connect your iOS device to your laptop via USB cable.
- Go to the iOS Apps directory `$ cd ~/MSTG-Handson/Apps` and execute the following commands on either your macOS device or the Linux VM:

macOS	Linux VM
Unpack the IPA: <pre>\$ unzip DVIA-v2-swift-frida-codesigned.ipa</pre>	Install the IPA on iOS: <pre>\$ ideviceinstaller -i DVIA-v2-swift-frida-codesigned.ipa</pre>
Install and run the app with ios-deploy: <pre>\$ ios-deploy --bundle 'Payload/DVIA-v2.app' -W -d</pre>	Execute and run the app through it's bundle name: <pre>\$ idevicedebug -d run com.hightitudehacks.DVIAswiftv2</pre>

If you want to run the app a 2nd time, use additionally the flag `-m` in ios-deploy, then the app will not be installed again but directly started.

- Once the app has started open a terminal and start objection

```
$ cd ~/tmp
$ objection -g Gadget 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
...highaltitudehacks.DVIAswiftv2 on (iPhone: 12.4) [usb] $

```

- If you see the prompt as above you can verify that is' working as objection could connect and can read the bundle name and also the iOS version on your iOS device (12.4 in the example above).
- In the DVIAv2 App, lick on the top left corner for the menu and select **Local Data Storage** . Select **Plist** and key in a username and password and click on **Save in a Plist file** .
- Now dump the memory with objection.

```

...highaltitudehacks.DVIAswiftv2 on (iPhone: 12.4) [usb] $ memory dump all DVIAv2-plist.dump

```

- This might take a few minutes to complete and the DVIAv2-plist.dump is now stored in the directory where you started objection (~/.tmp).
- Dumping memory is a fragile procedure and it might not be finishing completely. If th'ats the case and the app crashes or the dumping process has stoped, please close the app, and start the app again with ideviedebug/ios-deploy.
- Once you have the dump, open another terminal, go to the directory where the dump is stored and use **strings** .

```

$ cd ~/.tmp
$ strings DVIAv2-plist.dump > DVIAv2-plist.dump.txt

```

- Try to identify in the memory the username and password you keyed in earlier.

```

$ grep -i <username> DVIAv2-plist.dump.txt
$ grep -i <password> DVIAv2-plist.dump.txt

```

The same can also be done with the tool Fridump.

Resources

- Fridump - <https://github.com/Nightbringer21/fridump>

Testing Tools

To perform security testing different tools are available in order to be able to manipulate requests and responses, decompile Apps, investigate the behavior of running Apps and other test cases and automate them.

Mobile Application Security Testing Distributions

- [Androl4b](#) - A Virtual Machine For Assessing Android applications, Reverse Engineering and Malware Analysis
- [Android Tamer](#) - Android Tamer is a Debian-based Virtual/Live Platform for Android Security professionals.
- [AppUse](#) - AppUse is a Virtual Machine developed by AppSec Labs.
- [Santoku](#) - Santoku is an OS and can be run outside a VM as a standalone operating system.
- [Mobile Security Toolchain](#) - A project used to install many of the tools mentioned in this section both for Android and iOS at a machine running Mac OSX. The project installs the tools via Ansible

Static Source Code Analysis

- [Checkmarx](#) - Static Source Code Scanner that also scans source code for Android and iOS.
- [Fortify](#) - Static source code scanner that also scans source code for Android and iOS.
- [Veracode](#) - Static Analysis of iOS and Android binary

All-in-One Mobile Security Frameworks

- [Appmon](#) - AppMon is an automated framework for monitoring and tampering system API calls of native macOS, iOS and android apps.
- [Mobile Security Framework](#) - [MobSF](#) - Mobile Security Framework is an intelligent, all-in-one open source mobile application (Android/iOS) automated pen-testing framework capable of performing static and dynamic analysis.
- [Needle](#) - Needle is an open source, modular framework to streamline the process of conducting security assessments of iOS apps including Binary Analysis, Static Code Analysis, Runtime Manipulation using Cypcript and Frida hooking, and so on.
- [objection](#) - objection is a runtime mobile security assessment framework that does not require a jailbroken or rooted device for both iOS and Android, due to the usage of Frida.

Tools for iOS

Access Filesystem on iDevice

- [FileZilla](#) - It supports FTP, SFTP, and FTPS (FTP over SSL/TLS).
- [Cyberduck](#) - Libre FTP, SFTP, WebDAV, S3, Azure & OpenStack Swift browser for Mac and Windows.
- [itunnel](#) - Use to forward SSH via USB.
- [iFunbox](#) - The File and App Management Tool for iPhone, iPad & iPod Touch.

Reverse Engineering and Static Analysis

- [otool](#) - The otool command displays specified parts of object files or libraries.
- [Clutch](#) - Decrypted the application and dump specified bundleID into binary or .ipa file.
- [Dumpdecrypted](#) - Dumps decrypted mach-o files from encrypted iPhone applications from memory to disk. This tool is necessary for security researchers to be able to look under the hood of encryption.
- [class-dump](#) - A command-line utility for examining the Objective-C runtime information stored in Mach-O files.
- [Flex2](#) - Flex gives you the power to modify apps and change their behavior.
- [Weak Classdump](#) - A Cypcript script that generates a header file for the class passed to the function. Most useful

when you cannot classdump or dumpdecrypted , when binaries are encrypted etc.

- [IDA Pro](#) - IDA is a Windows, Linux or Mac OS X hosted multi-processor disassembler and debugger that offers so many features it is hard to describe them all.
- [HopperApp](#) - Hopper is a reverse engineering tool for OS X and Linux, that lets you disassemble, decompile and debug your 32/64bits Intel Mac, Linux, Windows and iOS executables.
- [Radare2](#) - Radare2 is a unix-like reverse engineering framework and command line tools.
- [iRET](#) - The iOS Reverse Engineering Toolkit is a toolkit designed to automate many of the common tasks associated with iOS penetration testing.
- [Plutil](#) - plutil is a program that can convert .plist files between a binary version and an XML version.

Dynamic and Runtime Analysis

- [Cycrypt](#) - Cycrypt allows developers to explore and modify running applications on either iOS or Mac OS X using a hybrid of Objective-C++ and JavaScript syntax through an interactive console that features syntax highlighting and tab completion.
- [iNalyzer](#) - AppSec Labs iNalyzer is a framework for manipulating iOS applications, tampering with parameters and method.
- [idb](#) - idb is a tool to simplify some common tasks for iOS pentesting and research.
- [snoop-it](#) - A tool to assist security assessments and dynamic analysis of iOS Apps.
- [Introspect-iOS](#) - Blackbox tool to help understand what an iOS application is doing at runtime and assist in the identification of potential security issues.
- [gdb](#) - A tool to perform runtime analysis of IOS applications.
- [lldb](#) - LLDB debugger by Apple's Xcode is used for debugging iOS applications.
- [keychaindumper](#) - A tool to check which keychain items are available to an attacker once an iOS device has been jailbroken.
- [BinaryCookieReader](#) - A tool to dump all the cookies from the binary Cookies.binarycookies file.
- [Burp Suite Mobile Assistant](#) - A tool to bypass certificate pinning and is able to inject into apps.

Bypassing Root Detection and SSL Pinning

- [SSL Kill Switch 2](#) - Blackbox tool to disable SSL certificate validation - including certificate pinning - within iOS and OS X Apps.
- [iOS TrustMe](#) - Disable certificate trust checks on iOS devices.
- [Xcon](#) - A tool for bypassing Jailbreak detection.
- [tsProtector](#) - Another tool for bypassing Jailbreak detection.

Tools for Network Interception and Monitoring

- [Tcpdump](#) - A command line packet capture utility.
- [Wireshark](#) - An open-source packet analyzer.
- [Canape](#) - A network testing tool for arbitrary protocols.
- [Mallory](#) - A Man in The Middle Tool (MiTM)) that is used to monitor and manipulate traffic on mobile devices and applications.

Interception Proxies

- [Burp Suite](#) - Burp Suite is an integrated platform for performing security testing of applications.
- [OWASP ZAP](#) - The OWASP Zed Attack Proxy (ZAP) is a free security tool which can help you automatically find security vulnerabilities in your web applications and web services.
- [Fiddler](#) - Fiddler is an HTTP debugging proxy server application which can capture HTTP and HTTPS traffic and logs it for the user to review. Fiddler can also be used to modify HTTP traffic for troubleshooting purposes as it is

being sent or received.

- [Charles Proxy](#) - HTTP proxy / HTTP monitor / Reverse Proxy that enables a developer to view all of the HTTP and SSL / HTTPS traffic between their machine and the Internet.

IDEs

- [Android Studio](#) - is the official integrated development environment (IDE) for Google's Android operating system, built on JetBrains' IntelliJ IDEA software and designed specifically for Android development.
- [IntelliJ](#) - IntelliJ IDEA is a Java integrated development environment (IDE) for developing computer software.
- [Eclipse](#) - Eclipse is an integrated development environment (IDE) used in computer programming, and is the most widely used Java IDE.
- [Xcode](#) - Xcode is an integrated development environment (IDE) available only for macOS to create apps for iOS, watchOS, tvOS and macOS.

Suggested Reading

Mobile App Security

Android

- Dominic Chell, Tyrone Erasmus, Shaun Colley, Ollie Whitehous (2015) *Mobile Application Hacker's Handbook*. Wiley. Available at: <http://www.wiley.com/WileyCDA/WileyTitle/productCd-1118958500.html>
- Joshua J. Drake, Zach Lanier, Collin Mulliner, Pau Oliva, Stephen A. Ridley, Georg Wicherski (2014) *Android Hacker's Handbook*. Wiley. Available at: <http://www.wiley.com/WileyCDA/WileyTitle/productCd-111860864X.html>
- Godfrey Nolan (2014) *Bulletproof Android*. Addison-Wesley Professional. Available at: <http://www.amazon.com/Bulletproof-Android-Practical-Building-Developers/dp/0133993329>

iOS

- Charlie Miller, Dionysus Blazakis, Dino Dai Zovi, Stefan Esser, Vincenzo Iozzo, Ralf-Philipp Weinmann (2012) *iOS Hacker's Handbook*. Wiley. Available at: <http://www.wiley.com/WileyCDA/WileyTitle/productCd-1118204123.html>
- David Thiel (2016) *iOS Application Security, The Definitive Guide for Hackers and Developers*. No Starch Press. Available at: <https://www.nostarch.com/iossecurity>
- Jonathan Levin (2017), *Mac OS X and iOS Internals*, Wiley. Available at: <http://newosxbook.com/index.php>

Misc

Reverse Engineering

- Bruce Dang, Alexandre Gazet, Elias Backaalany (2014) *Practical Reverse Engineering*. Wiley. Available at: <http://as.wiley.com/WileyCDA/WileyTitle/productCd-1118787315,subjectCd-CSJ0.html>
- Skakenunny, Hangcom *iOS App Reverse Engineering*. Online. Available at: <https://github.com/iosre/iOSAppReverseEngineering/>
- Bernhard Mueller (2016) *Hacking Soft Tokens - Advanced Reverse Engineering on Android*. HITB GSEC Singapore. Available at: <http://gsec.hitb.org/materials/sg2016/D1%20-%20Bernhard%20Mueller%20-%20Attacking%20Software%20Tokens.pdf>
- Dennis Yurichev (2016) *Reverse Engineering for Beginners*. Online. Available at: <https://github.com/dennis714/RE-for-beginners>
- Michael Hale Ligh, Andrew Case, Jamie Levy, Aaron Walters (2014) *The Art of Memory Forensics*. Wiley. Available at: <http://as.wiley.com/WileyCDA/WileyTitle/productCd-1118825098.html>