

The Mobile Playbook: Day 2



Sven Schleier - © Bai7 GmbH

Contents

Lab - Fork and setup Repo	2
Lab - Static Scanning with mobsfscan	4
Lab - Frida 101 (Frida-Server)	10
Lab - Sensitive Data in Local Storage	18
Lab - Bypass “Piracy Detection”	21
Lab - Bypass Jailbreak Detection	29

Lab - Fork and setup Repo

Time to finish lab	5 minutes
App used for this exercise	None

Training Objectives

In the following exercise we will fork a Github repository that we will use for some of the Android exercises.

Preparation

Github

- Fork the following repo: <https://github.com/bai7-at/mobile-playbook-dev-ios>

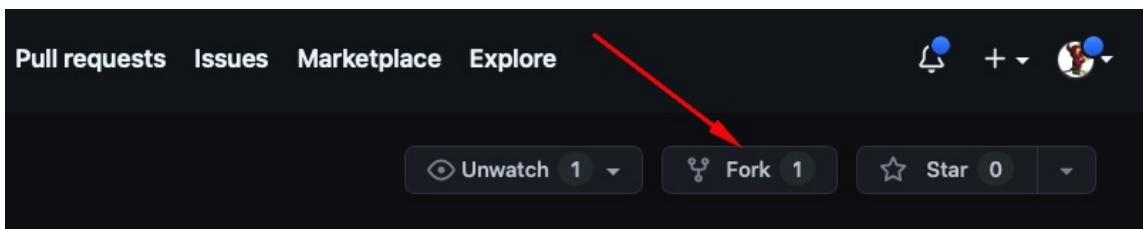


Figure 1: Fork repo

- In the next screen simply click “Create fork” and it will be forked to your Github account.
- Go to “Actions” in your newly forked repo and click on the green button to enable Github workflows.

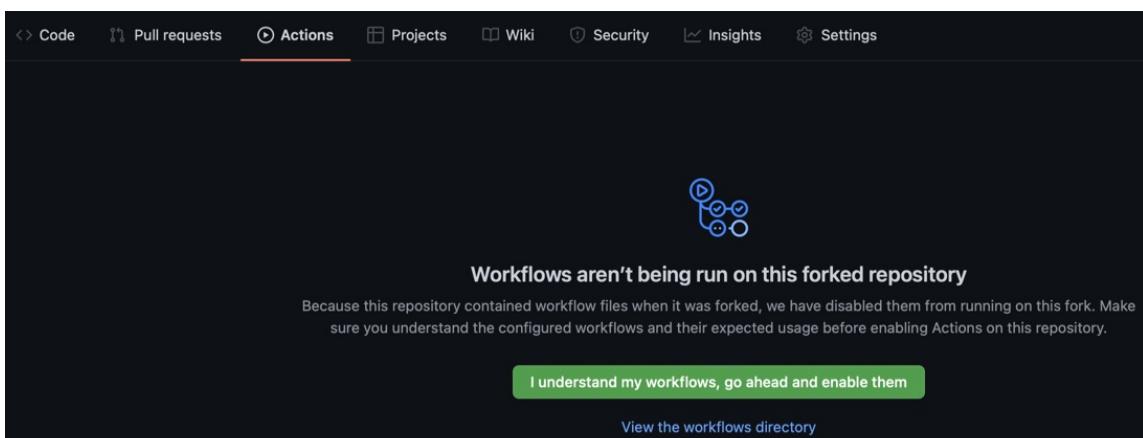


Figure 2: Enable Workflows

- Enable “Issues” in your forked repo. You can find it in the “Settings” tab under “Features”.

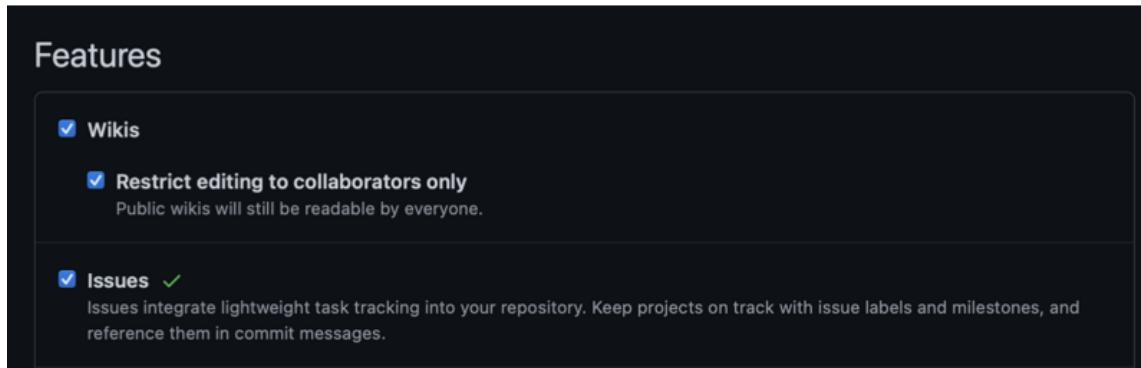


Figure 3: Enable Issues

Congratulations, you forked the repo! Later we will be activating some Github Actions to execute automated security checks.

Lab - Static Scanning with mobsfscan

Time to finish lab	20 minutes
--------------------	------------

Training Objectives

In this lab we will be scanning the source code of an iOS app for vulnerabilities and misconfiguration with [mobsfscan](#).

Tools used in this section

- [mobsfscan](#) - <https://github.com/MobSF/mobsfscan>

Preparation

- Go to your forked repo and to the main directory in your browser. Select the `.github/workflows` folder and the file `03-mobsfscan.yml`.

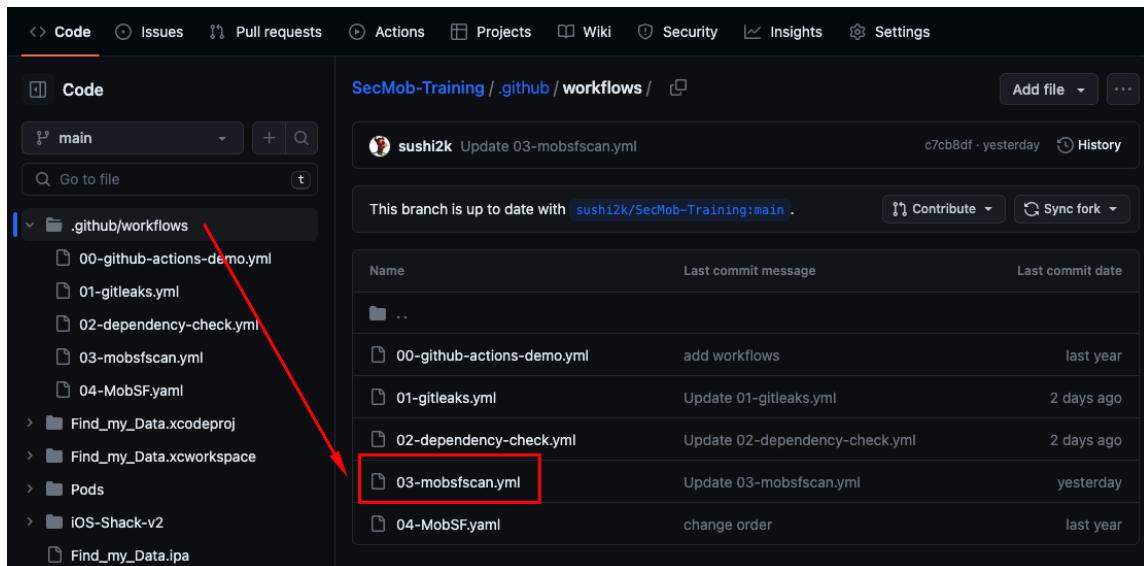


Figure 4: mobsfscan workflow

- In the next screen press either the key “e” on your keyboard or press the edit button top right. In the editor mode uncomment everything by selecting the whole text and pressing “/”.

The Mobile Playbook: Day 2

A screenshot of the GitHub interface showing the 'Code' tab for the file '03-mobsfscan.yml'. The file content is displayed in a code editor with syntax highlighting. A red arrow points to the 'Edit' button in the top right corner of the code editor.

```
# Documentation for Workflow Syntax: https://docs.github.com/en/actions/learn-github-action/configuring-a-workflow#example-workflow
# The name of the GitHub event that triggers the workflow.
name: mobsfscan

# on:
#   push:
#     branches: [ master, main ]
#   pull_request:
#     branches: [ master, main ]
jobs:
  # Name of Job
  - mobsfscan
```

Figure 5: Edit mobsfscan workflow

- Commit the changes directly into the main branch.

A screenshot of the GitHub interface showing the 'Code' tab for the file '03-mobsfscan.yml'. The file content is displayed in a code editor. A red arrow points to the 'Commit changes...' button in the top right corner of the code editor.

```
# Documentation for Workflow Syntax: https://docs.github.com/en/actions/learn-github-action/configuring-a-workflow#example-workflow
# The name of the GitHub event that triggers the workflow.
name: mobsfscan

# on:
#   push:
#     branches: [ master, main ]
#   pull_request:
#     branches: [ master, main ]
jobs:
  # Name of Job
  - mobsfscan
```

Figure 6: Commit changes

GitHub Action

- Click on “Actions” and you will see the workflows that were just triggered due to the commit. Click on the `mobsfscan` workflow run. The scan might still be running.

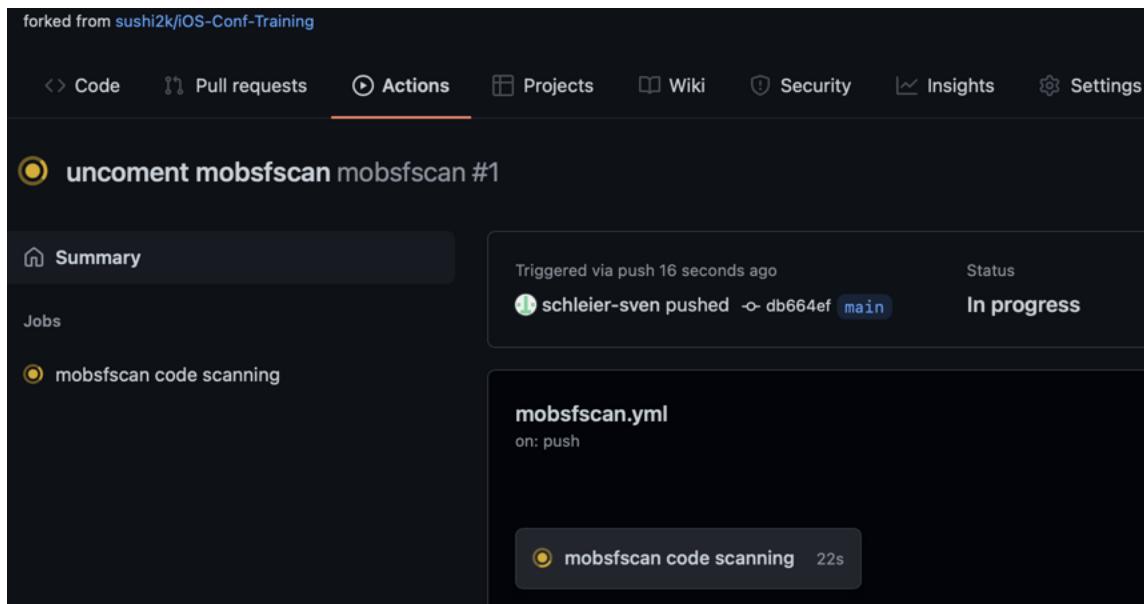


Figure 7: mobsfscan Workflow

- Once the mobsfscan job is done you will have identified new vulnerabilities that were added to the “Security” tab, click on it.

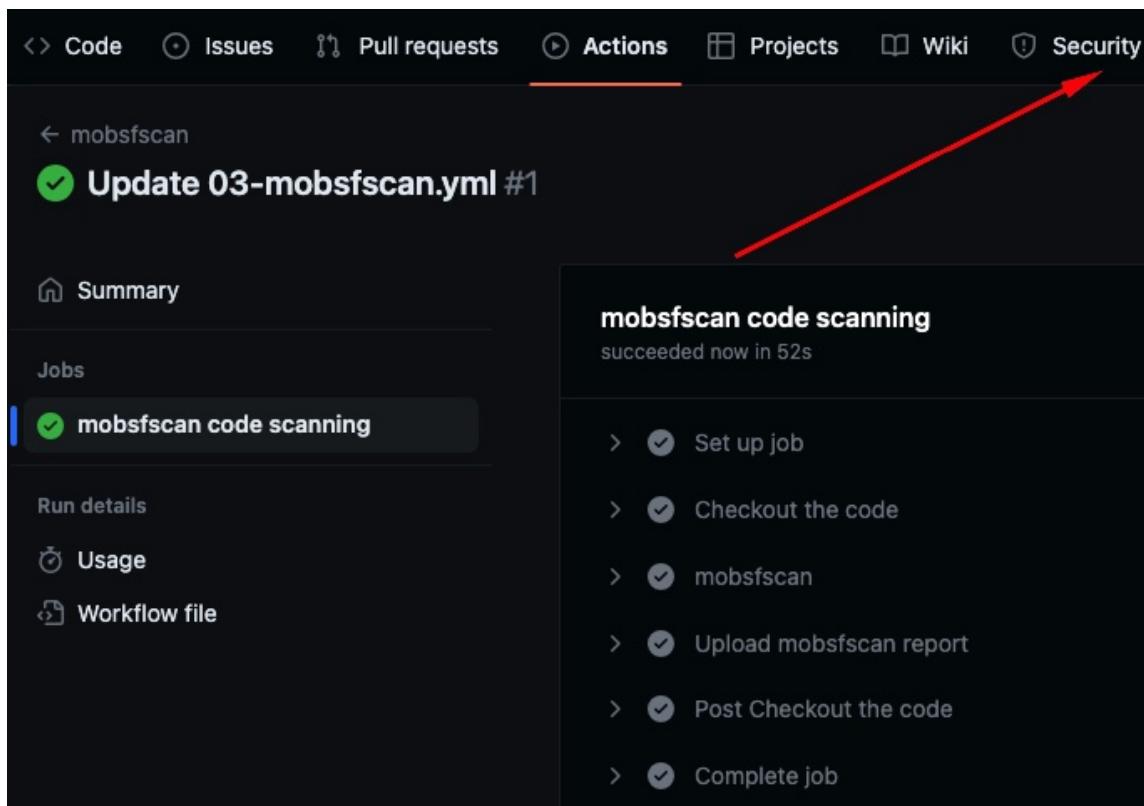


Figure 8: Security Tab

- The `mobsfscan` Github Action will now be executed every time we are doing a commit or pull request and will raise now an alert if sensitive information is being detected. But

detecting is only half the job, as we need to manage the amount of detected vulnerabilities in an efficient way. Luckily this was already being taken care of automatically through our Github action and the SARIF file that was being created and imported into Github Code Scanning. This is the relevant snippet from the `mobsfscan.yml` file:

```
# Upload the SARIF file to Github, so the findings show up in "Security / Code Scanning alerts"
- name: Upload mobsfscan report
  uses: github/codeql-action/upload-sarif@v1
  with:
    sarif_file: results.sarif
```

- Go to “Code Scanning Alerts”. This allows us to manage the identified vulnerabilities. Github takes care of de-duplication of findings automatically through fingerprinting, so Github will not flag out the same finding again in consecutive scans.

We have a large number of findings and to start with we should focus on specific rules from `mobsfscan` and start with `ios_hardcoded_secret`:

The screenshot shows the GitHub Code Scanning interface. At the top, there are navigation links: Code, Issues, Pull requests, Actions, Projects, Wiki, Security (with 131 findings), and Insights. A red arrow points from the 'Issues' link to the 'Code scanning' section. In the 'Code scanning' section, there's a green checkmark indicating 'All tools are working as expected'. Below it, a search bar shows 'ios open branch:main'. A red arrow points from the 'Tool' dropdown menu to the 'ios_hardcoded_secret' checkbox, which is highlighted with a red border. The main list of findings includes:

- 131 Open (0 Closed)
- App allows self signed or invalid SSL certificates. (#113 opened 7 minutes ago - Detected by mobsfscan in Find_my_ip)
- App allows self signed or invalid SSL certificates. (#112 opened 7 minutes ago - Detected by mobsfscan in Find_my_ip)
- Biometric authentication should be hardware and a boolean that can be bypassed by runtime instru...

Figure 9: Findings for specific rule

- Now the work starts! Your job is to triage the identified findings, go through them one by one and decide if they are either a:
 - valid finding (then create an issue),
 - false positive,
 - used in tests, or
 - if you won't fix it (and why).

Code scanning alerts / #8

Files may contain hardcoded sensitive information like usernames, passwords, keys etc.

The screenshot shows a GitHub code scanning alert for a file named `ViewController.swift`. The alert is triggered at line 37, where it finds hardcoded AWS access and secret access keys. The alert details show the tool used is `mobsfscan` and the rule ID is `ios_hardcoded_secret`. A modal window titled "Select a reason to dismiss" is open, listing three options: "False positive" (selected), "Used in tests", and "Won't fix". A red box highlights the "False positive" option, and a red arrow points from this box down to the "Dismiss alert" button at the bottom right of the modal.

Figure 10: Triage findings

You can exclude for now the rules `ios_log` and `ios_app_logging` as they have almost 90 findings.

Go through each finding and make a decision! Once done with the first rule, choose another one.

In case a finding is identified but no code is highlighted, this means the absence of code patterns might be a vulnerability.

Congratulations, you are now able to scan vulnerabilities with `mobsfscan` and detect and triage vulnerabilities and manage code scanning alerts with Github Code Scanning!

Appendix

Github Actions are defined in Yaml files and have a very simple structure and are self explanatory; you might want to revisit the Gitleaks yaml file after you've done the lab to understand the flow in more depth:

```
# Documentation for Workflow Syntax: https://docs.github.com/en/actions/learn-github-actions/workflow-syntax-for-github-actions
```

```
# The name of the GitHub event that triggers the workflow.
name: mobsfscan

on:
  push:
    branches: [ master, main ]
  pull_request:
    branches: [ master, main ]
jobs:
  # Name of Job
  mobsfscan:
    permissions:
      id-token: write
      contents: read
      security-events: write
    runs-on: ubuntu-latest
    name: mobsfscan code scanning
    steps:
      # This action checks-out your repository under
      # $GITHUB_WORKSPACE, so your workflow can access it.
      - name: Checkout the code
        uses: actions/checkout@v2
      # Execute the MobSF scan and create a SARIF file
      - name: mobsfscan
        uses: MobSF/mobsfscan@main
        with:
          args: 'Find_my_Data --sarif --output results.sarif || true'
      # Upload the SARIF file to Github, so the findings show up in "
      # Security / Code Scanning alerts"
      - name: Upload mobsfscan report
        uses: github/codeql-action/upload-sarif@v2
        with:
          sarif_file: results.sarif
```

Lab - Frida 101 (Frida-Server)

Time to finish lab	20 minutes
App used for this exercise	Find_my_Data.ipa

Training Objectives

1. Learn how to use the Frida Server
2. Use a Frida script

Tools used in this section

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

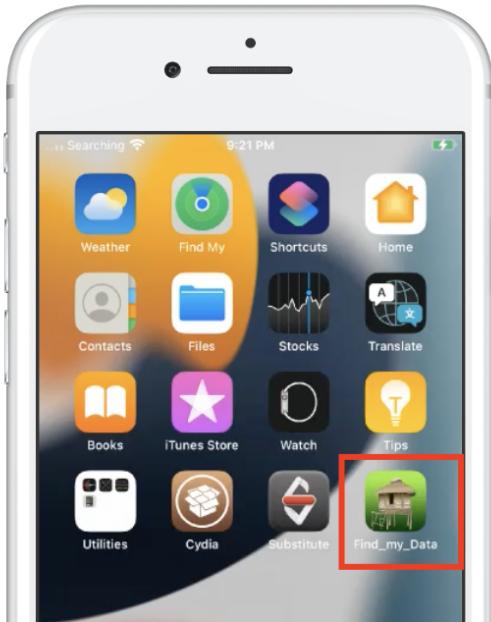
Exercise - Frida Usage

Preparation

- Go to “Apps” and install the app `Find_my_Data.ipa` by selecting “Install App”. You downloaded the IPA as part of the preparation pack today. This might take a minute to install.

The screenshot shows the ring-point mobile application interface. At the top, it displays "5 ring-point (iPhone 7 Plus | 15.7.6 | 19H349 | ✓ Jailbroken)". On the left is a sidebar with icons for Connect, Files, Apps (which is selected and highlighted in grey), Network, and CoreTrace. The main area is titled "Apps ⓘ" and subtitle "Manage apps and packages". It features a search bar labeled "Search". A red arrow points from the "Apps" icon in the sidebar to the "INSTALL APP" button at the top right of the main screen. Below the table, a red arrow points from the "Apps" icon in the sidebar to the "NAME" column header of the table. The table lists one app: "Calendar com.apple.mobi" with details: INSTALLED "7/11/23 11:41 AM", TYPE "System", and SIZE "1.16 MI".

- Run the app “Find_my_data”:



Frida

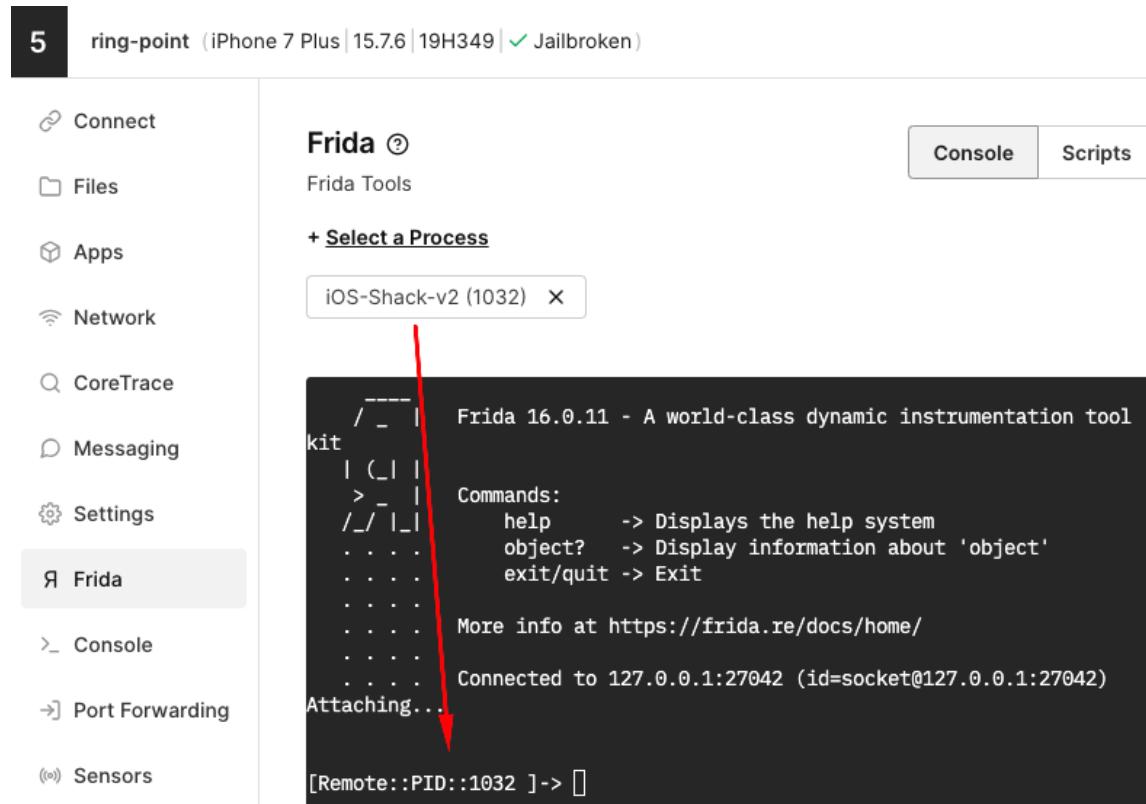
- Start the Frida Server by clicking on the Frida tab on the left in Corellium. Click on “Select a Process” and search for “find” and you will find the running app (your process id, the PID, will be different in your case):

The screenshot shows the Corellium interface with the Frida tab selected. On the left, there's a sidebar with options like Connect, Files, Apps, Network, CoreTrace, Messaging, Settings, and Frida. In the main area, under the Frida tab, there's a "Frida Tools" section with a button labeled "+ Select a Process". To the right, a modal window titled "Select a Process" is open. It has a search bar containing "find" and a list of processes. The list includes:

PID	Name
144	findmydeviced
933	FindMyWidgetPeople
944	FindMyWidgetItems
1174	Find_my_Data

A red arrow points from the "Select a Process" button on the left to the search bar in the modal. Another red arrow points from the search bar in the modal to the "Find_my_Data" entry in the list.

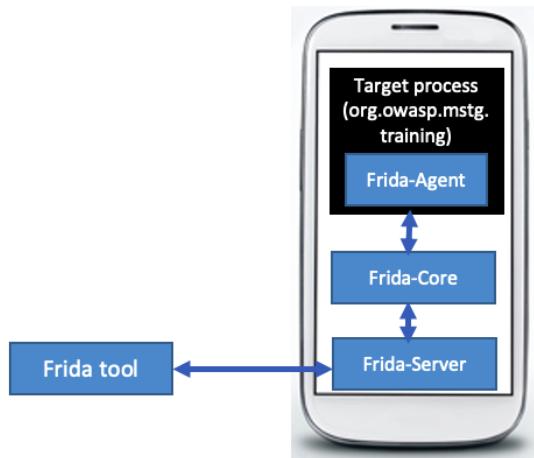
- Once selected, attach to the process and make sure that the app is running in the foreground on the iPhone.



- You can see now the Frida console.

```
/ _ _ |   Frida 16.0.11 - A world-class dynamic instrumentation
toolkit
| ( _ | |
> _ |   Commands:
/_/ |_|   help      -> Displays the help system
. . . .   object?   -> Display information about 'object'
. . . .   exit/quit -> Exit
. . . .
. . . .
. . . .
. . . .
More info at https://frida.re/docs/home/
. . . .
. . . .
Connected to 127.0.0.1:27042 (id=socket@127
.0.0.1:27042)
Attaching...
[Remote::PID::1032 ]->
```

Frida CLI (console) is now connecting to the Frida-Server running on the iOS instance and the Frida Core Framework is injecting the Frida-Agent into the specified identifier (`iOS-Shack-v2`).



So what can we do now with the interactive Frida console? You can write commands to Frida using its ObjC API, just press Tab 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.

```
[Remote:::PID::2194 ]-> 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:

```
[Remote:::Find_my_Data]-> ObjC.classes.NSBundle mainBundle()  
    .bundleIdentifier().toString()  
"info.s7ven.ios.data"
```

The Frida CLI is case-sensitive. So in case you are facing any errors this might be the issue.

Frida Scripts

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

You can find the `Frida-Scripts` directory in the directory where you unzipped the preparation file.

Click on the top right on “Scripts” and upload the script `frida101.js`:

The Mobile Playbook: Day 2



Execute the script:

The screenshot shows the Frida Tools interface with the 'Scripts' tab selected. In the center, there is a table listing scripts. The first row, 'frida101.js', has a red arrow pointing to its 'Execute' button. The second row, 'hook_native.js', also has an 'Execute' button.

	Name	Size	Last Modified	Actions
	frida101.js	2.39 KB	8/14/2024 6:06 AM	Execute ⋮
	hook_native.js	329 bytes	2/4/2021 1:41 PM	Execute ⋮

Go back to the console and type in [y](#) and confirm.

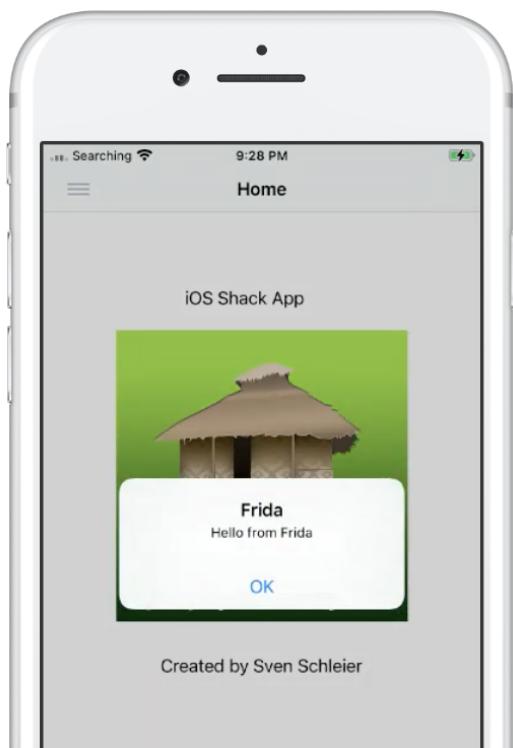
The screenshot shows the Frida console. It displays the Frida help menu, connection details, and a command history. The last command entered is '[Remote:::PID:::1174] -> %load /data/corellium/frida/scripts/frida101.js'. A red box highlights the confirmation prompt '[Are you sure you want to load a new script and discard all current state? [y/N]]' followed by the letter 'y'.

```
-----  
| _ |  Frida 16.2.2-dev.6 - A world-class dynamic instrumentation toolkit  
| (_| |  
> _ |  Commands:  
/_/_|    help      -> Displays the help system  
... . .  object?   -> Display information about 'object'  
... . .  exit/quit -> Exit  
... . .  
... . .  More info at https://frida.re/docs/home/  
... . .  
... . .  Connected to 127.0.0.1:27042 (id=socket@127.0.0.1:27042)  
  
[Remote:::PID:::1174 ]-> ObjC.classesNSBundle mainBundle().bundleIdentifier().toString()  
"info.s7ven.ios.data"  
[Remote:::PID:::1174 ]-> %load /data/corellium/frida/scripts/frida101.js  
[Are you sure you want to load a new script and discard all current state? [y/N]] y  
[Remote:::PID:::1174 ]->
```

Let's call the function [helloWorld\(\)](#):

```
[Remote:::PID:::1032]-> 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 in any way we want.

Modify a Frida Script

Edit the file `Frida-Scripts/frida101.js` in Corellium. You can see in the script the function `helloWorld()` that we just called.

The screenshot shows the Corellium Frida interface. It has a sidebar with "Frida Tools" and a main area where a process named "Find_my_Data (1174)" is selected. There is a "Upload" button. Below is a table of scripts:

	Name ^	Size	Last Modified	Actions
<input type="checkbox"/>	frida101.js	2.39 KB	8/14/2024 6:06 AM	<u>Execute</u> ⋮
<input type="checkbox"/>	hook_native.js	329 bytes	2/4/2021 1:41 PM	
<input type="checkbox"/>	hook_objc.js	810 bytes	2/4/2021 1:46 PM	
<input type="checkbox"/>	replace_native.js	733 bytes	2/4/2021 1:48 PM	

A context menu is open over the "frida101.js" row, with "Edit" highlighted and a red arrow pointing to it. Other options in the menu include "Execute", "Download", and "Delete".

In the Frida-Scripts folder of the files you downloaded today, open the file `extension.js` in an editor of your choice. You will see two functions. Copy both functions from `extension.js` into the `frida101.js` file, before the `helloWorld()` function in Line 5. Save the script.

```
1  /* https://frida.re/docs/examples/ios/
2   * Sent a Hello World to your app:
3   *     helloWorld()
4   */
5
6  function appInfo() {
7    var output = {};
8    output["Name"] = infoLookup("CFBundleName");
9    output["Bundle ID"] = ObjC.classesNSBundle.mainBundle().bundleIdentifier().toString();
10   output["Version"] = infoLookup("CFBundleVersion");
11   output["Bundle"] = ObjC.classesNSBundle.mainBundle().bundlePath().toString();
12   output["Data"] = ObjC.classes.NSProcessInfo.processInfo().environment().objectForKey_("HOME").toString(); output["Binary"]
13     = ObjC.classesNSBundle.mainBundle().executablePath().toString();
14   return output;
15 }
16
17 function infoLookup(key) {
18   if (ObjC.available && "NSBundle" in ObjC.classes) {
19     var info = ObjC.classesNSBundle.mainBundle().infoDictionary(); var value = info.objectForKey_(key);
20     if (value === null) {
21       return value;
22     } else if (value.class().toString() === "__NSArray") {
23       return arrayFromNSArray(value);
24     } else if (value.class().toString() === "__NSDictionary") {
25       return dictFromNSDictionary(value); } else {
26       return value.toString(); }
27     return null;
28   }
29 }
30 function helloWorld()
```

SAVE

CLOSE

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()`. Go back to the Frida console and give it a try!

```
[Remote:::PID:::1032 ]-> appInfo()
{
  "Binary": "/private/var/containers/Bundle/Application/D68ED06F
              -3746-4CC1-9F0C-F772CAAE16CE/Find_my_Data.app/Find_my_Data",
  "Bundle": "/private/var/containers/Bundle/Application/D68ED06F
              -3746-4CC1-9F0C-F772CAAE16CE/Find_my_Data.app",
  "Bundle ID": "info.s7ven.ios.data",
  "Data": "/private/var/mobile/Containers/Data/Application/
              C57CA8B9-D0CE-4DBC-AC25-2B77565391FB",
  "Name": "Find_my_Data",
  "Version": "1"
}
```

This will print a lot of useful meta-information for a penetration tester about the app, including the data and bundle paths.

Congratulations you are able to use the Frida server and can modify and load scripts!

References

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

Lab - Sensitive Data in Local Storage

Time to finish lab	20 minutes
App used for this exercise	Find_my_Data.ipa

Training Objectives

Identify sensitive information stored in the iOS App

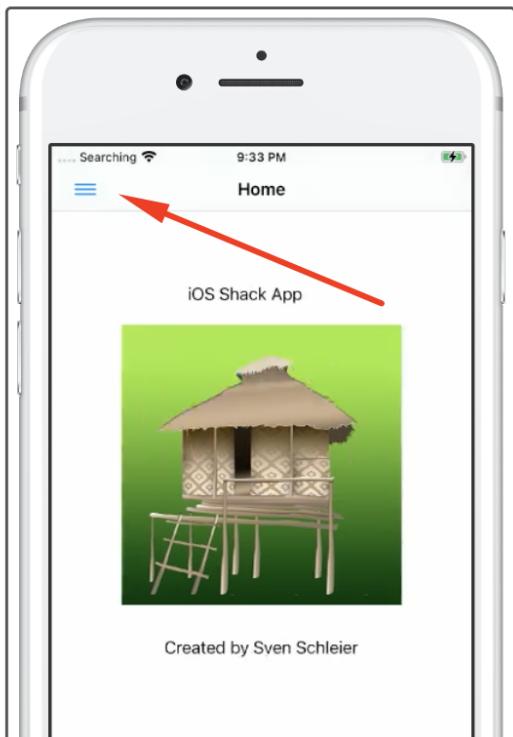
Tools used in this section

- Corellium

Exercise

Identify Sensitive Data

- Go to the app you installed in the previous lab and click on the menu button top left and on “Sensitive Data”. This will create files in the App’s sandbox.



- From the previous lab you have all the information you need to access the app's sandbox. Check the output of `appinfo()` and search for the “Data” key, the value will be the directory of this app on your device.

The Mobile Playbook: Day 2

The image shows two side-by-side screenshots. On the left is the Frida Tools interface, specifically the 'Console' tab, displaying the output of an appinfo() command for a process named 'Find_my_Data'. A red box highlights the 'Bundle ID' and 'Data' lines, which show the bundle identifier and the path to the app's data directory. On the right is a screenshot of a virtual iPhone displaying a 'Sensitive Data' screen with instructions to check local storage.

- Go to the files menu on the left in Corellium and navigate to the Data path of your app.

The image shows the Corellium interface with the 'Files' menu item highlighted by a red box and a red arrow pointing to the main file listing area. The listing shows the contents of the app's sandbox at the path '/private/var/mobile/Containers/Data/Application/C57CA8B9-D0CE-4DBC-AC25-2B77565391FB'. The table includes columns for Name, Size, UID, GID, Permissions, and Last Modified. Several files are listed, including '.com.apple.mobile_container_manager.metadata.plist', 'Documents', 'Library', 'SystemData', and 'tmp'.

Name	Size	UID	GID	Permissions	Last Modified
.com.apple.mobile_container_manager.metadata.plist	534 bytes	0	501	r--r--r--	8/14/2024 6:17 AM
Documents	160 bytes	501	501	rwxr-xr-x	8/14/2024 6:17 AM
Library	192 bytes	501	501	rwxr-xr-x	8/14/2024 6:17 AM
SystemData	64 bytes	501	501	rwxr-xr-x	8/14/2024 6:17 AM
tmp	64 bytes	501	501	rwxr-xr-x	8/14/2024 6:17 AM

- Once you identified files in the apps' sandbox, you can view the content when clicking on the menu on the right:

The Mobile Playbook: Day 2

Files ?
Browse the device filesystem

Search for files

/ private / var / mobile / Containers / Data / Application / C57CA8B9-D0CE-4DBC-AC25-2B77565391FB / Documents /

Name	Size	UID	GID	Permissions	Last Modified
Database.sqlite	12.3 KB	501	501	rw-r--r--	8/14/2024 6:17 AM
debug.plist	290 bytes	501	501	rw-r--r--	8/14/2024 6:17 AM
message.txt	12 bytes	501	501	rw-r--r--	8/14/2024 6:17 AM

View Contents

Download

Modify permissions

Copy filepath

Delete

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

If you find a sqlite database, download it and you can analyze it with Sqlitebrowser.

What sensitive information could you find? Do you need further protection mechanisms for the sensitive information? If so, how would you do it and implement it?

Lab - Bypass “Piracy Detection”

Time to finish lab	20 minutes
App used for this exercise	DamnVulnerableiOSApp.ipa

Training Objectives

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

Tools used in this section

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

Exercise

Preparation

- Go to “Apps” and install the app `DamnVulnerableiOSApp.ipa` by selecting “Install App”. You downloaded the IPA as part of the preparation pack today. This might take a minute.

The screenshot shows the 'ring-point' mobile application interface. At the top, it displays the device information: 'ring-point (iPhone 7 Plus | 15.7.6 | 19H349 | ✓ Jailbroken)'. On the left, there is a sidebar with icons for 'Connect', 'Files', 'Apps' (which is highlighted), 'Network', and 'CoreTrace'. The main area is titled 'Apps ⓘ' and contains the sub-instruction 'Manage apps and packages'. A red arrow points from the 'Apps' icon in the sidebar to the 'INSTALL APP' button. Another red arrow points from the 'Search' bar to the 'NAME' column header. Below the table, there is a small circular icon with a red dot and a grid pattern. The table itself has columns for NAME, INSTALLED, TYPE, and SIZE. One entry is visible: 'Calendar com.apple.mobi' was installed on '7/11/23 11:41 AM' and is a 'System' type file of size '1.16 Mi'.

NAME	INSTALLED	TYPE	SIZE
Calendar com.apple.mobi	7/11/23 11:41 AM	System	1.16 Mi

- Run the app “DVIA”:



Frida

- Start the Frida Server by clicking on the Frida tab on the left in Corellium. Click on “Select a Process” and search for “dvia” and you will find the running app (your process id, the PID, will be different in your case).

The screenshot shows the Corellium interface for an iPhone 7 Plus jailbroken device. On the left, there's a sidebar with options like Connect, Files, Apps, Network, CoreTrace, Messaging, Settings, and Frida. The Frida option is highlighted with a red arrow. In the main pane, under the Frida Tools section, there's a button labeled "+ Select a Process". To its right is a search bar with the placeholder "Q dvia". Below the search bar is a table with two columns: NAME and PID. The first row in the table has a red box around the "NAME" column value "DVIA" and another red arrow pointing to it. The "PID" column shows "646".

- Once selected, attach to the process and make sure that the app is running in the foreground on the iPhone.

The screenshot shows the Frida Tools interface. At the top, there's a header with the Frida logo and a question mark icon, followed by "Frida Tools". Below that is a button labeled "+ Select a Process". A process list box contains "DVIA (646)" with a close button ("X"). The main area is a dark terminal window displaying the Frida command-line interface. It shows the Frida version (16.0.11), some internal commands, and then connects to a remote target at 127.0.0.1:27042. The text ends with "[Remote:::PID::646]->".

```
----|  Frida 16.0.11 - A world-class dynamic instrumentation toolkit
| ( | |
> _ | Commands:
/_/ |_ help      -> Displays the help system
. . . . object?   -> Display information about 'object'
. . . . exit/quit -> Exit
. . . .
. . . . More info at https://frida.re/docs/home/
. . . .
. . . . Connected to 127.0.0.1:27042 (id=socket@127.0.0.1:27042)

[Remote:::PID::646 ]->
```

- You can see now the Frida console.

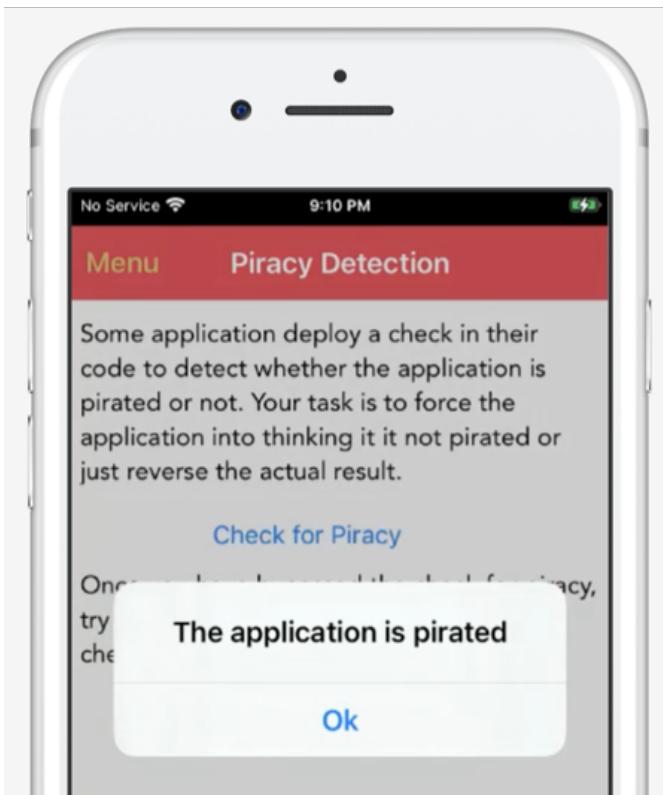
This screenshot is identical to the one above, showing the Frida command-line interface in the terminal window of the Frida Tools application. It displays the Frida version, commands, and a connection to a remote target at 127.0.0.1:27042. The text ends with "[Remote:::PID::646]->".

```
----|  Frida 16.0.11 - A world-class dynamic instrumentation
     toolkit
| ( | |
> _ | Commands:
/_/ |_ help      -> Displays the help system
. . . . object?   -> Display information about 'object'
. . . . exit/quit -> Exit
. . . .
. . . . More info at https://frida.re/docs/home/
. . . .
. . . . Connected to 127.0.0.1:27042 (id=socket@127
.0.0.1:27042)

[Remote:::PID::646 ]->
```

Damn Vulnerable iOS App (DVIA)

- In the DVIA App, click on the top left corner on the menu and select Piracy Detection. Click on the upper button and you will get the following message.



Our mission is to reverse this check!

Frida CLI + Scripts

We already know that we can use Frida scripts to inject them into the target app.

- You can list all classes that are used in the app with the script `find-classes.js` and search for interesting class names. Upload the script and execute it.

Frida ⓘ

Frida Tools

+ Select a Process

DVIA (646) X

UPLOAD

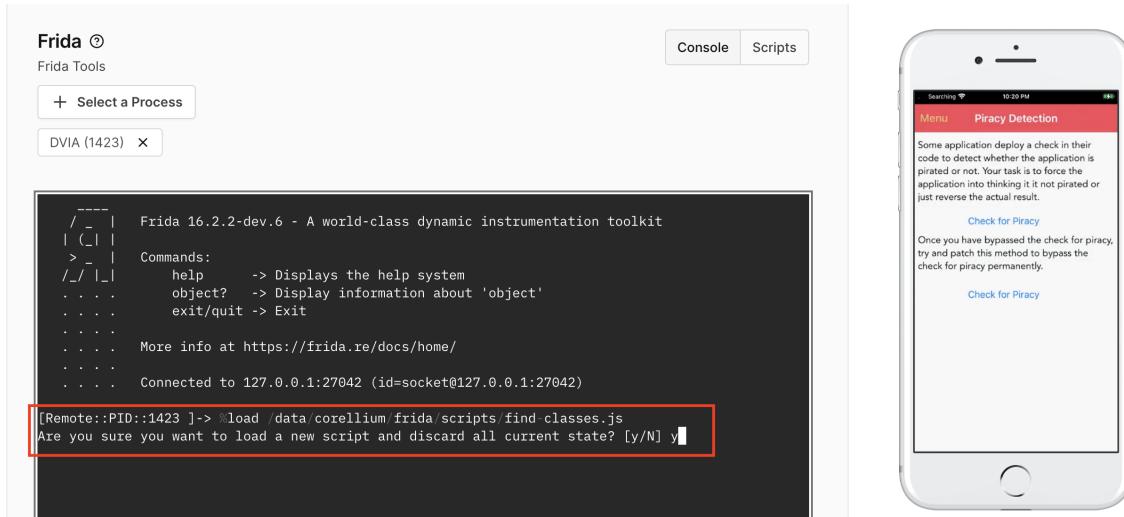
NAME SIZE LAST MODIFIED

<input type="checkbox"/> frida101.js	2.39 KB	7/11/2023 12:41 PM	EXECUTE	...
--------------------------------------	---------	--------------------	---------	-----

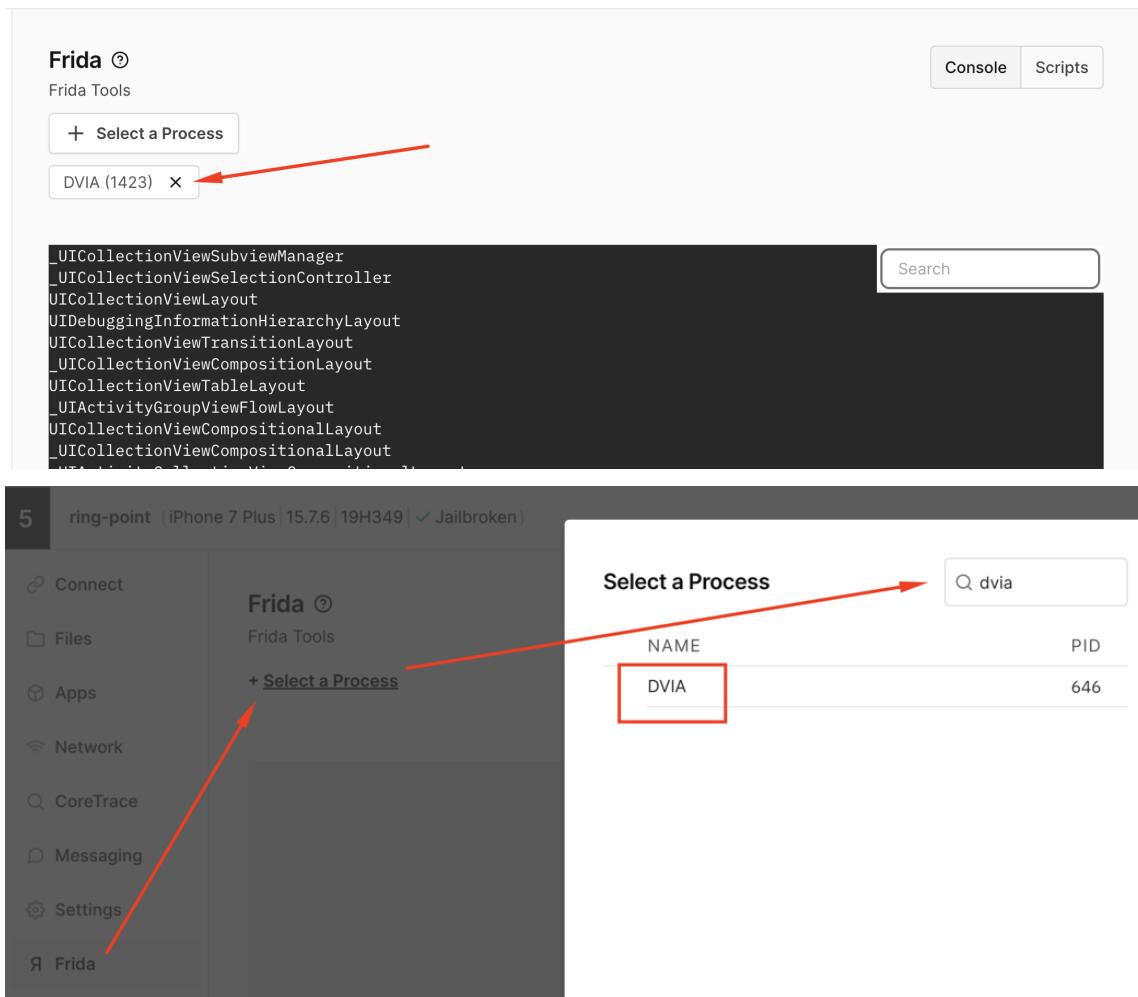
A red arrow points from the 'UPLOAD' button to the 'frida101.js' file entry in the list.

- Go back to the console and accept the execution. This script will dump now all loaded classes of the app and this might take a few seconds.

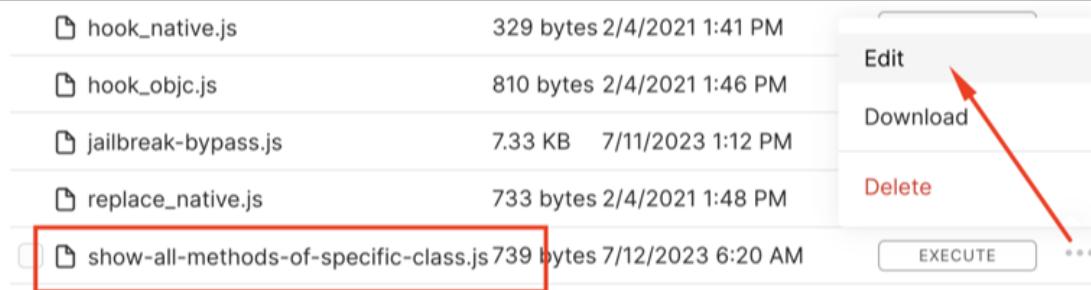
The Mobile Playbook: Day 2



- When it's finished search with “Ctrl+F” the output in the Frida console for the keyword “antipiracy”. You should find a class name that contains this keyword. Copy the class name into a text file in an editor of your choice to have it handy!
- To avoid that this script is now executed all the time, we will stop Frida on the DVIA app and re-attach it.



- Once you have identified the class you want to investigate further. Upload the script `show-all-methods-of-specific-class.js`. This will list all methods of a class. Replace the placeholder in line 7 “XXXXXXX” with the class name you identified earlier. The script will get all methods of that class.



```
1 console.log("[*] Started: Find All Methods of a Specific Class");
2 if (ObjC.available)
3 {
4     try
5     {
6         //Your class name here
7         var className = "XXXXXX";
8         var methods = eval('ObjC.classes.' + className + '.$methods');
9         for (var i = 0; i < methods.length; i++)
10        {
11            try
12            {
13                console.log("[-] " + methods[i]);
14            }
15            catch(err)
16            {
17                console.log("[!] Exception1: " + err.message);
18            }
19        }
20    }
21    catch(err)
22    {
23        console.log("[!] Exception2: " + err.message);
24    }
25 }
26 else
27 {
28     console.log("Objective-C Runtime is not available!");
29 }
30 console.log("[*] Completed: Find All Methods of a Specific Class");
```

SAVE

CLOSE

- Execute the script you just modified. Switch to the console, confirm the execution of the script with “y”, and look at the output and search for “pirated”.

Frida ②

Frida Tools

+ [Select a Process](#)

DVIA (646) X

```
_DKEventQuery
_DKPredictionQuery
[*] Completed: Find Classes
[*] Started: Find All Methods of a Specific Class
[-] + isTheApplicationCracked
[-] + isTheDeviceJailbroken
[-] + isTheApplicationTamperedWith
[-] + urlCheck
[-] + cydiaCheck
[-] + inaccessibleFilesCheck
[-] + plistCheck
[-] + processesCheck
[-] + fstabCheck
[-] + systemCheck
[-] + symbolicLinkCheck
[-] + filesExistCheck
[-] + isPirated
[-] + isJailbroken
```

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 the class and method upload the script [show-modify-function-return-value.js](#). Edit the script by replacing the placeholder “YOUR_CLASS_NAME_HERE” with your class and the placeholder “YOUR_EXACT_FUNC_NAME_HERE” with the method in line 21.
- The other Frida scripts will also be executed, so wait till the output of the other scripts was generated.

Press the button “Check for Privacy” again.

The Mobile Playbook: Day 2

The image shows two side-by-side screenshots. On the left is the Frida Tools interface, specifically the 'Console' tab. It displays a list of methods for the class 'SFAntiPiracy'. A red box highlights the method 'isPirated'. Below it, a message asks if the user wants to load a new script and discard current state. On the right is a screenshot of an iPhone displaying an application titled 'Piracy Detection'. The app's text says: 'Some application deploy a check in their code to detect whether the application is pirated or not. Your task is to force the application into thinking it is not pirated or just reverse the actual result.' A button labeled 'Check for Piracy' has been pressed, and a modal dialog box shows the text 'The application is not pirated' with an 'Ok' button. A red arrow points from the highlighted 'isPirated' method in the Frida console to the 'Ok' button in the mobile application's dialog.

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?

Lab - Bypass Jailbreak Detection

Time to finish lab	20 minutes
App used for this exercise	Jailbreak-1.ipa

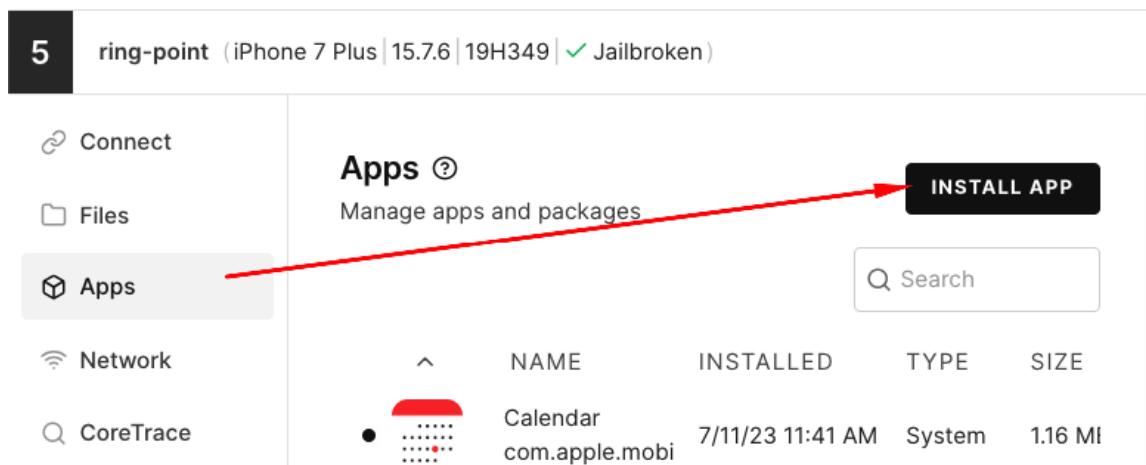
Training Objectives

Bypass Jailbreak Detection Mechanisms

Exercise

Preparation

- Go to “Apps” and install the app [Jailbreak-1.ipa](#) by selecting “Install App”. You downloaded the IPA as part of the preparation pack today. This might take a minute.



- Run the app “Jailbreak-1”:

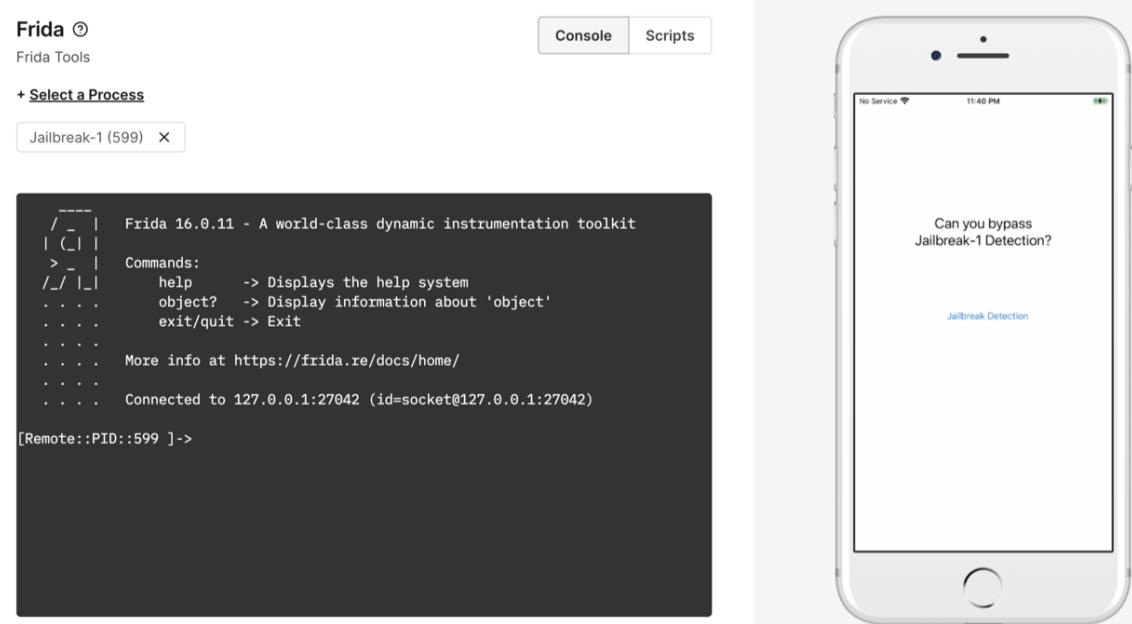


Jailbreak App

- Start the Frida Server by clicking on the Frida tab on the left in Corellium. Click on “Select a Process” and search for “jailbreak” and you will find the running app (your process id, the PID, will be different in your case):

A screenshot of the Corellium interface. On the left, there's a sidebar with options like Connect, Files, Apps, Network, CoreTrace, Messaging, Settings, and Frida. The Frida option is selected, indicated by a red arrow. In the main pane, there's a "Frida Tools" section with a "Select a Process" button. To its right is a "Select a Process" dialog box. This dialog box has a search bar at the top with the text "jail". Below the search bar is a table with two columns: "NAME" and "PID". A row in the table is highlighted with a red arrow and shows "Jailbreak-1" in the NAME column and "599" in the PID column.

- Once selected, attach to the process and make sure that the app is running in the foreground on the iPhone.

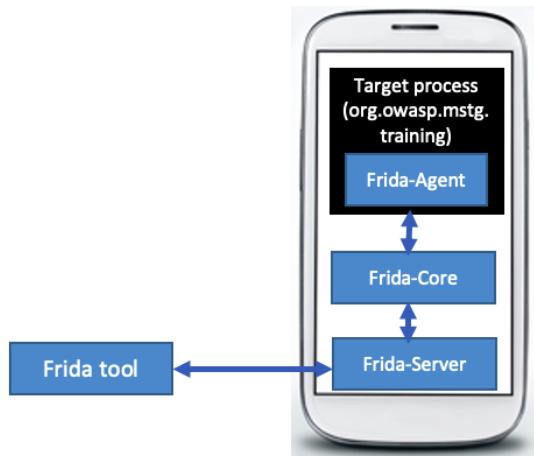


- Click on “Console” button on the top right. You can see now the Frida console.

```
----|  Frida 16.0.11 - A world-class dynamic instrumentation toolkit
| (.| |
> _ |  Commands:
/_/ |_|  help      -> Displays the help system
. . . .  object?   -> Display information about 'object'
. . . .  exit/quit -> Exit
. . . .  More info at https://frida.re/docs/home/
. . . .  Connected to 127.0.0.1:27042 (id=socket@127.0.0.1:27042)

[Remote::PID::599 ]->
```

Frida CLI is now connecting to the running Frida server and is injecting the Frida Agent into the **Jailbreak-1** app.



- In the App, click on the button to trigger the **Jailbreak Detection**. A pop-up will appear saying the device is jailbroken. This check we want to bypass.

Frida CLI + Scripts

We already know that we can use Frida scripts to inject them into the target app.

- Upload the script `jailbreak-bypass-1.js` and execute it and go back to the console and confirm the execution with “y”.

NAME	SIZE	LAST MODIFIED	EXECUTE	...
find-classes.js	336 bytes	7/12/2023 6:15 AM	EXECUTE	...
frida101.js	2.39 KB	7/11/2023 12:41 PM	EXECUTE	...
hook_native.js	329 bytes	2/4/2021 1:41 PM	EXECUTE	...
hook_objc.js	810 bytes	2/4/2021 1:46 PM	EXECUTE	...
jailbreak-bypass-1.js	2.70 KB	7/12/2023 9:06 AM	EXECUTE	...

Frida ⓘ

Frida Tools

+ [Select a Process](#)

Jailbreak-1 (599) X

```

| _ | Commands:
> - | help      -> Displays the help system
/_/ |_| object?   -> Display information about 'object'
. . . . exit/quit -> Exit
. . . .
. . . . More info at https://frida.re/docs/home/
. . . .
. . . . Connected to 127.0.0.1:27042 (id=socket@127.0.0.1:27042)

[Remote:::PID::599 ]-> %load /data/corellium/frida/scripts/jailbreak-bypass-1.js
Are you sure you want to load a new script and discard all current state? [y/N] y

[Remote:::PID::599 ]-> []

```

- Trigger again the jailbreak detection button. You can see in the console output that some checks were detected and hooked and therefore bypassed, but the app still detects the jailbreak.

Frida ⓘ

Frida Tools

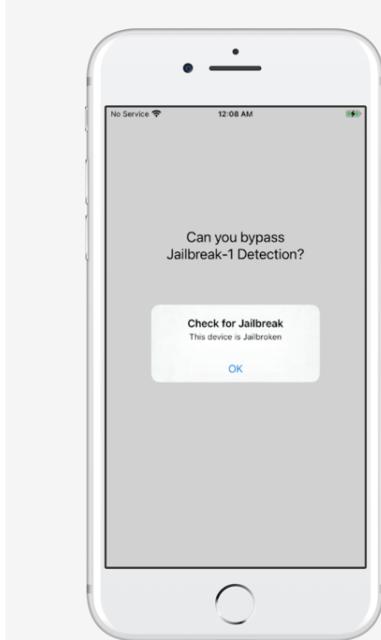
+ [Select a Process](#)

Jailbreak-1 (599) X

```

[Remote:::PID::599 ]-> Hooking native function stat64: /Applications/Cydia.app
Hooking native function stat: /Applications/Cydia.app
stat64 Bypass!!!
stat Bypass!!!
Hooking native function stat64: /Library/MobileSubstrate/MobileSubstrate.dylib
Hooking native function stat: /Library/MobileSubstrate/MobileSubstrate.dylib
stat64 Bypass!!!
stat Bypass!!!
Hooking native function stat64: /bin/bash
Hooking native function stat: /bin/bash
stat64 Bypass!!!
stat Bypass!!!
Hooking native function stat64: /usr/sbin/sshd
Hooking native function stat: /usr/sbin/sshd
stat64 Bypass!!!
stat Bypass!!!
Hooking native function stat64: /etc/apt
Hooking native function stat: /etc/apt
stat64 Bypass!!!
stat Bypass!!!

```



- This script was therefore not fully working and couldn't bypass some checks. Upload the script `jailbreak-bypass-2.js` and execute it and go back to the console and confirm the execution with “y”.
- Trigger again the jailbreak detection button. You can see in the console output that some checks were detected and hooked and therefore bypassed, and this time the app

is not detecting the jailbreak anymore.

The image shows two side-by-side screenshots. On the left is a screenshot of the Frida Tools interface, specifically the 'Console' tab. It displays a log of native function hooking attempts, many of which are failing due to bypass mechanisms. Key lines from the log include:

```
Hooking native function stat64: /usr/sbin/sshd
Hooking native function stat: /usr/sbin/sshd
stat64 Bypass!!!
stat Bypass!!!
fileExistsAtPath: try to check for /usr/sbin/sshd was failed
Hooking native function stat64: /etc/apt
Hooking native function stat: /etc/apt
stat64 Bypass!!!
stat Bypass!!!
fileExistsAtPath: try to check for /etc/apt was failed
canOpenURL: check for cydia://package/com.example.package was successful with: 0x1, marking it as failed.
fopen: check for /bin/bash was successful with: 0x121e05278, marking it as failed
.
fopen: check for /Applications/Cydia.app was successful with: 0x121e31938, marking it as failed.
fopen: try to check for /Library/MobileSubstrate/MobileSubstrate.dylib was failed
fopen: check for /usr/sbin/sshd was successful with: 0x121d0c298, marking it as failed.
fopen: check for /etc/apt was successful with: 0x121e266e8, marking it as failed.
```

On the right is a screenshot of an iPhone displaying a bypassed jailbreak detection dialog. The screen shows the text "Can you bypass Jailbreak-1 Detection?" above a button labeled "Check for Jailbreak". A callout bubble indicates "This device is not Jailbroken" and has an "OK" button. A red arrow points from the Frida log to this dialog, illustrating the success of the bypass.

- How many different jailbreak detection mechanisms that are being bypassed can you find? You can also have a look at the source code of [jailbreak-bypass-2.js](#).

Congratulations, you are able to bypass Jailbreak detection like a penetration tester!