Intro to mobile testing

Foo-Manroot

https://foo-manroot.github.io/

Index

- What is mobile testing?
- Android platform
- iOS platform
- Prepare the lab
- Some static analysis
- A note or two on dynamic analysis
- Hands-on: example vulnerable apps (Android and iOS)
- Wrapping-up and proposed testing methodology

What is mobile testing?

(for us)

What is mobile testing? (for us)

Testing of mobile *applications*

- We assume that the client's app is not malware
- We consider Android and iOS to be secure (i.e.: permissions can't be bypassed)
- Hardware issues are also not considered
- We have a *very* short time for:
    - iOS
    - Android
    - All web APIs (hopefully, shared between Android and iOS)

What is mobile testing? (for us)

There are a lot of (*very* different) available platforms:

- Android (and derivatives, like LineageOS)
- iOS
- Others (little-to-no adoption, but still present):
    - Windows Phone (RIP In Peace)
    - Plasma Mobile (KDE Plasma, but for mobile)
    - PostmarketOS (Based on Alpine Linux)
    - …

- We focus on Android and iOS, the most common ones

What is mobile testing? (for us)

Also, a ton of frameworks for app development:

- Native
  - Plain Java/Kotlin (Android)
  - Plain Swift/Obj-C (iOS)
- Cross-platform
  - Flutter (Dart / Google)
  - Xamarin (C# / Microsoft)
  - Cordova (JS / Apache)
  - React Native (JS / Facebook)
  - …

- Each has their own challenges for testing

What is mobile testing? (for us)

There are, however, not many testing frameworks:

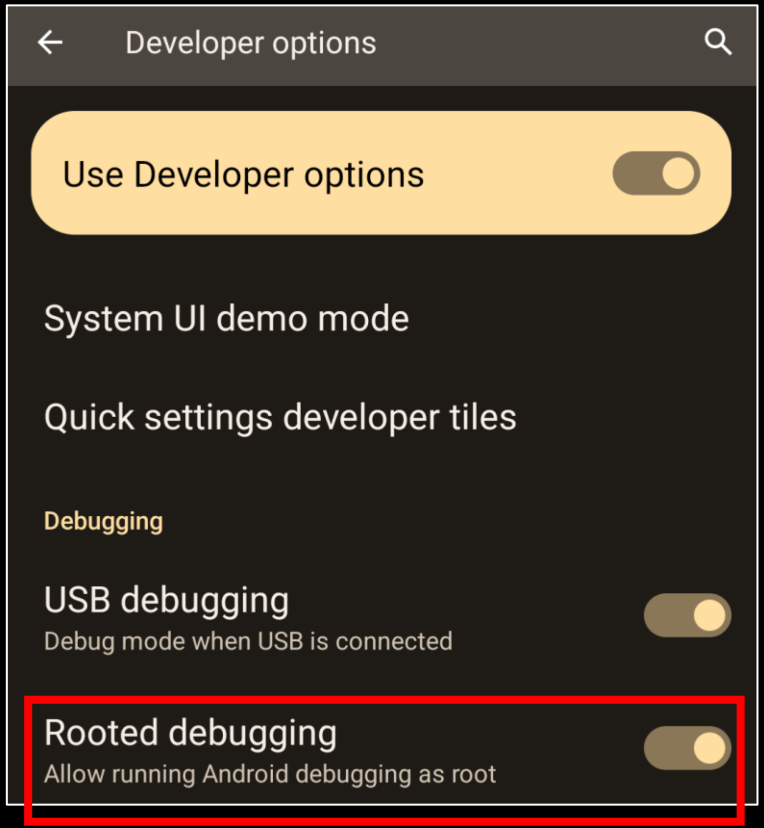
- OWASP Mobile Security Testing Guide (https://mobile-security.gitbook.io)
  - Already a bit outdated, but still really useful
  - Comprehensive introduction to iOS and Android architectures
  - Defines specific tests following the Mobile Application Security Verification Standard (https://mobile-security.gitbook.io/masvs/)
- OWASP Mobile Top 10
  - Ok, not a framework… But still worth mentioning
  - Not updated since 2016 (still interesting to know)
  - https://owasp.org/www-project-mobile-top-10/

What is mobile testing? (for us)

Rooting and jailbreaking

- Gain full access on the system by exploiting the OS
    - Some Android ROMs allow root from the developer settings
    - Other Android devices are basically impossible to root
    - For iOS, a full exploit chain is required
        - ❤️ checkm8
        - https://www.theiphonewiki.com/
    - We won't cover this here, it's too much info

- Not always necessary, but helps a lot



What is mobile testing? (for us)

- Normal requirements for testing (my recommendation):
- Hardware (depends on the app):
    - Mobile device (iOS/Android)
        - Ideally, 1 rooted and 1 unrooted mobile device, just in case the root detection is good
    - Antenna to create a hotspot and intercept traffic (? - maybe, depends on your setup)
    - Kali or MacOS
        - Windows lacks support for iOS, but is perfectly fine for Android testing
    - We'll cover the tools later

- The app to test (duhh):
    - In its prod-like settings (including cert. pinning, root detection, and all that jazz)
    - Ideally, also another build without protections, to test the API
        - Not always provided by the developers

What is mobile testing? (for us)

A final (sad) note:

- We normally get *very little* time for testing (~5 days)
- If we have Android and iOS (*and*, obviously, the APIs), we have to prioritise
  - Some apps don’t require any client-side security
  - Customers normally care more about their own infra
  - Client-side security is relegated to when we have spare time after the API test ☹

- In the end, mobile testing (5 days) ends up being:
  - Bypass certificate pinning (if any) and straight to API testing
  - Very basic client-side vulns detected by automated tools

Android platform

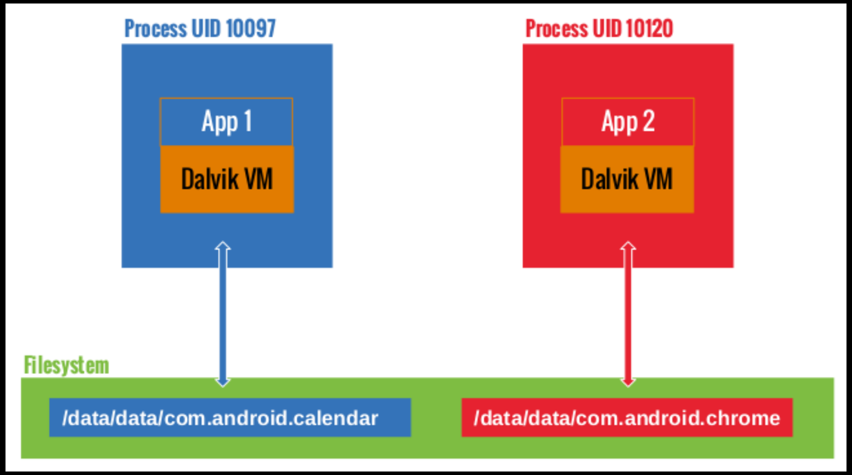
Android platform

- Linux-based, (mostly) open-source: https://source.android.com/docs/setup/build/downloading
- Vendors add their own modules

- Functionalities depend on the API level
  - Current stable release (Android 12) > API level 31, 32
  - https://developer.android.com/studio/releases/platforms

- Apps run in the Dalvik VM (similar to Java VM) / Nowadays it's the "AndroidRunTime"
  - Native code can also be run from the app using the Java Native Interface (JNI)
  - DEX Bytecode can be converted to Smali (intermediate language), which can be converted to Java

Android platform



Source: *OWASP MASTG*

- Each app runs in their own sandbox
  - Separate resources
  - Unique user and group per-installation
  - SELinux
  - Seccomp (limits available syscalls)
  - Permissions (access to the network, location, calls, …)

- Full disk encryption since API 21
- File-based encryption (unique keys for different files) since API 24

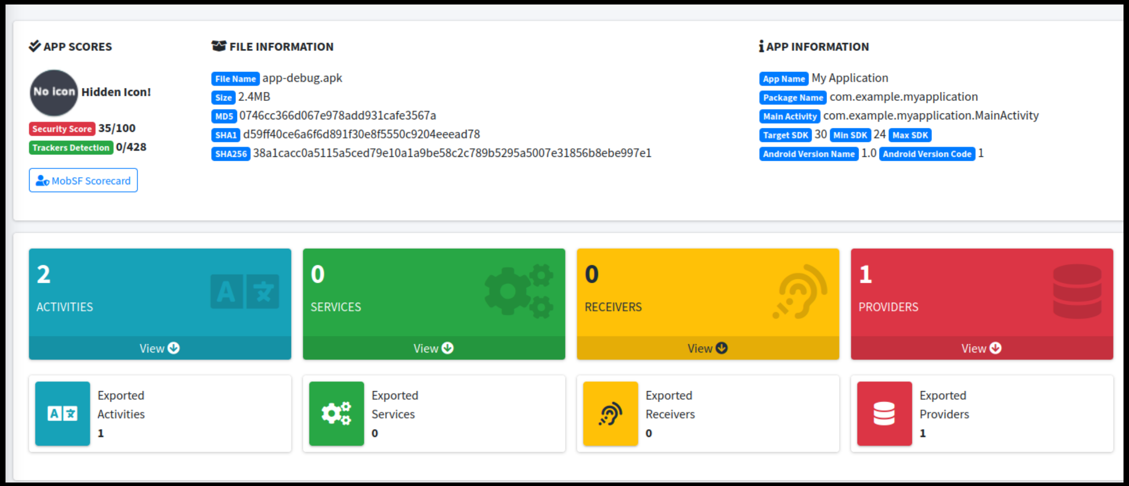
- Trusted Execution Environment (TEE) to protect crypto material

Android platform

- TLS by default
- DNS over TLS (if supported bythe network)

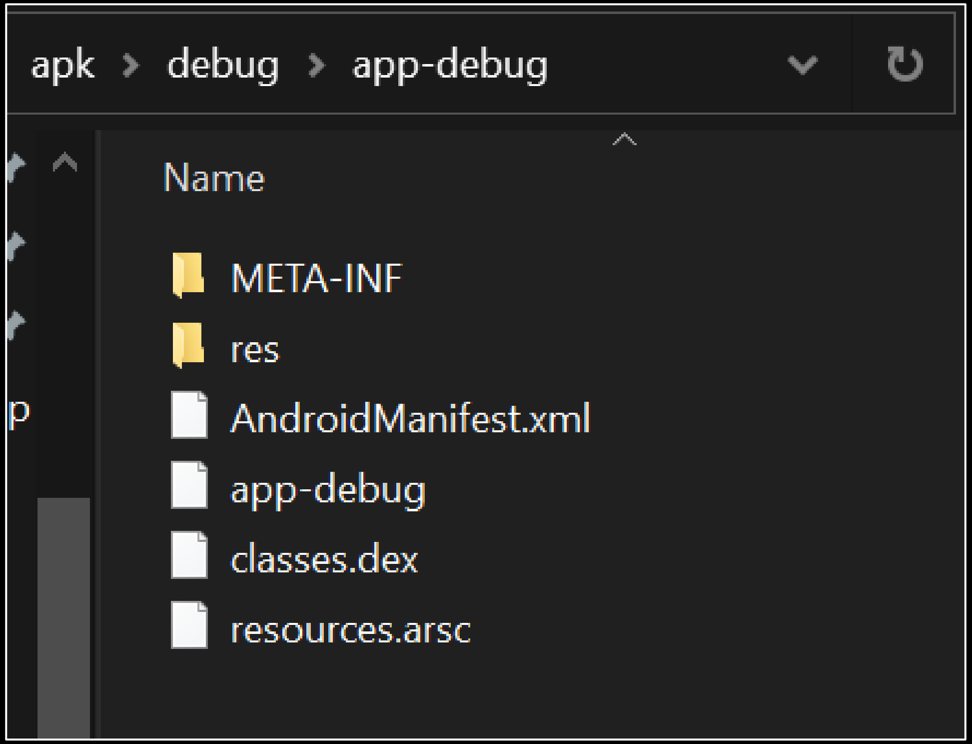
- ASLR, PIE, DEP, …
  - Only relevant when targeting native code

- Apps can communicate using several IPC methods:
  - Intents
  - Content providers
  - Services
  - Broadcast messages



Android platform

- Apps are bundled in an APK (similar to a JAR):
  - Little more than a Zip with a specific structure
    - /META-INF/
    - res/
    - AndroidManifest.xml
    - classes.dex
    - resources.arsc
  - Resources are encoded (binary)



- Split APKs have separate bundles for resources
  - Intended for internationalisation and targeting different devices architecture
    - `com.example.1234.apk ← Base APK`
    - `com.example.1234.config.armeabi_v7a.apk ← Native libs`
    - `com.example.1234.config.en.apk ← I18n`
    - `com.example.1234.config.xxhdpi.apk ← Images and stuff`

Android platform

- AndroidManifest.xml
  - Describes (almost) everything about the app
    - Permissions (`<uses-permission>` and `<permission>`)
    - Services `<service>`, Content Providers `<provider>` and Broadcast receivers `<receiver>`
    - All activities (`<activity>`)
    - Entry point (`android.intent.action.MAIN`)
    - Required API level (`<uses-sdk android:minSdkVersion="24" android:targetSdkVersion="30" />`)
    - App attributes (`allowBackup`, `extractNativeLibs`, `usesCleartextTraffic`)
    - `<intent-filter>`
      - deep-links (`<data android:scheme="example" android:host="asdf" />`)
      - File handlers (`<data android:mimeType="application/pdf">`)
  - Can be decoded with aapt2 (part of Android SDK) or apktool (https://ibotpeaches.github.io/Apktool/)

Android platform

- Once installed, the app is under `/data/app/<hash>/<app-name>/base.apk`

```
OnePlus6T:/data/app/~~oXIOtU6JOmze2PS9g5QdXw==/ch.twint.payment-_a92T_5AZCBBBJ95x201EQ== # l * */*/* -d
-rw-r--r-- 1 system system   43709143 2022-08-13 07:37 base.apk
drwxr-xr-x 3 system system       4096 2022-08-13 07:37 lib
-rwxr-xr-x 1 system system    2402848 1981-01-01 01:01 lib/arm64/liba6a12c.so
-rwxr-xr-x 1 system system    6478544 1981-01-01 01:01 lib/arm64/libb2c197.so
-rwxr-xr-x 1 system system      76440 1981-01-01 01:01 lib/arm64/libce9d.so
-rwxr-xr-x 1 system system     674104 1981-01-01 01:01 lib/arm64/libconcealjni.so
-rwxr-xr-x 1 system system    1153432 1981-01-01 01:01 lib/arm64/libdae.so
-rwxr-xr-x 1 system system     891168 1981-01-01 01:01 lib/arm64/libe35259.so
-rwxr-xr-x 1 system system     465048 1981-01-01 01:01 lib/arm64/libfb.so
-rwxr-xr-x 1 system system    3392968 1981-01-01 01:01 lib/arm64/libsqlcipher.so
drwxrwx--x 3 system install      4096 2022-08-13 07:37 oat
-rw-r--r-- 1 system all_a283   299952 2022-08-18 13:24 oat/arm64/base.odex
-rw-r--r-- 1 system all_a283 39076900 2022-08-18 13:24 oat/arm64/base.vdex
-rw-r--r-- 1 system system    1092641 2022-08-13 07:37 split_config.xhdpi.apk
```

- Data* is stored under `/data/data/<app-name>/`

```
OnePlus6T:/data/data/ch.twint.payment # l
total 104
drwx------  11 u0_a283  u0_a283        4096 2022-04-11 20:24 .
drwxrwx--x 262 system   system        20480 2022-08-18 09:44 ..
drwxrwx--x   2 u0_a283  u0_a283        4096 2022-04-11 20:24 app_textures
drwx------   3 u0_a283  u0_a283        4096 2022-04-29 14:58 app_webview
drwxrws--x   6 u0_a283  u0_a283_cache  4096 2022-05-26 18:17 cache
drwxrws--x   2 u0_a283  u0_a283_cache  4096 2022-01-30 20:57 code_cache
drwxrwx--x   2 u0_a283  u0_a283        4096 2022-01-30 23:07 databases
drwxrwx--x   4 u0_a283  u0_a283        4096 2022-07-26 13:32 files
drwxrwx--x   2 u0_a283  u0_a283        4096 2022-04-17 23:00 no_backup
drwx------   3 u0_a283  u0_a283        4096 2022-01-30 23:06 oat
drwxrwx--x   2 u0_a283  u0_a283        4096 2022-07-26 13:33 shared_prefs
```

*Shared storage is `/sdcard/`, and since Android 10 there's scoped storage on `/sdcard/Android`

Crypto material ***should*** be stored in the KeyStore

Android platform

- To interact with the phone, we use the Android Debugging Bridge (adb), part of the Android SDK platform tools (https://developer.android.com/studio/releases/platform-tools)

```
└─$ adb shell
OnePlus6T:/ # getprop
[af.fast_track_multiplier]: [2]
[apexd.status]: [ready]
[audio.deep_buffer.media]: [true]
[audio_offload_min_duration_secs]: [30]
```

Other useful commands (within the shell):

- `run-as` (Runs as another user)
- `pm` (Package Manager)
- `am` (Activity Manager)

- Logs can be read with `logcat`

```
└─$ adb logcat  | head
--------- beginning of kernel
08-19 09:56:15.956     0     0 I binder  : 12961:12961 ioctl 40046210 7fca7a9cd4 returned -22
08-19 09:56:16.916     0     0 E synaptics,s3320: all finger up
08-19 09:56:17.696     0     0 I CPU7    : update max cpu_capacity 967
08-19 09:56:21.322     0     0 E synaptics,s3320: all finger up
08-19 09:56:22.594     0     0 I CPU4    : update max cpu_capacity 1024
08-19 09:56:23.305     0     0 E synaptics,s3320: all finger up
08-19 09:56:23.356     0     0 E synaptics,s3320: all finger up
08-19 09:56:23.390     0     0 E synaptics,s3320: all finger up
08-19 09:56:24.276     0     0 E synaptics,s3320: all finger up
```

iOS platform

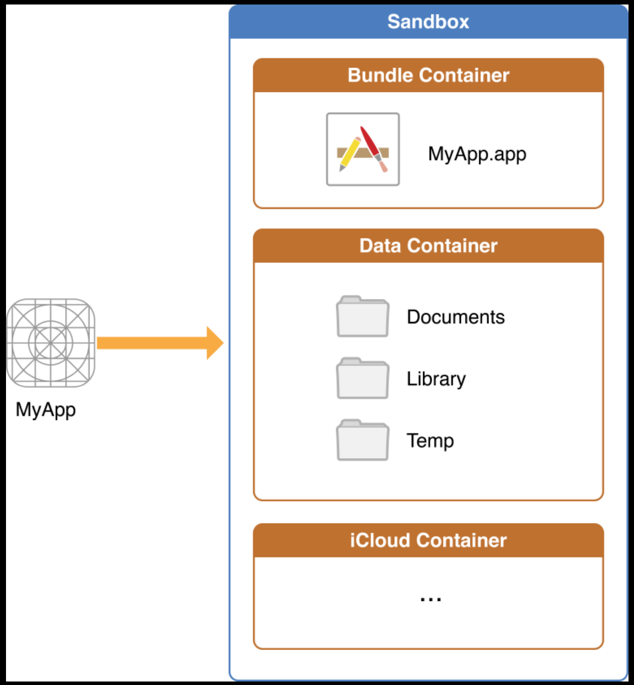
iOS platform

- XNU based (UNIX-like)
- Apple controls the hardware
  - Integrated AES-256 processor and encryption keys:
    - UID → fused into the Application Processor (AP). Unique per device
    - GID → compiled into the AP*(and another on the SEP?)* Unique per processor model (i.e.: all A10 processors share the same GID)
  - Secure Enclave Processor (SEP) to handle crypto operations
  - Only code signed by Apple can be run
  - Apps from the AppStore are encrypted (FairPlay)
    - The decryption key is linked with the Apple account and stored on the SEP

- Apps are written in Swift or Obj-C and compiled (Mach-O)
  - ASLR, NX, PIE, …

iOS platform

- Each app runs in their own sandbox
  - Filesystem (APFS) permissions are not leveraged
    - All apps run under the same user: “mobile”
  - Apps are chrooted to `/var/containers/Bundle/Application/<Bundle ID>`

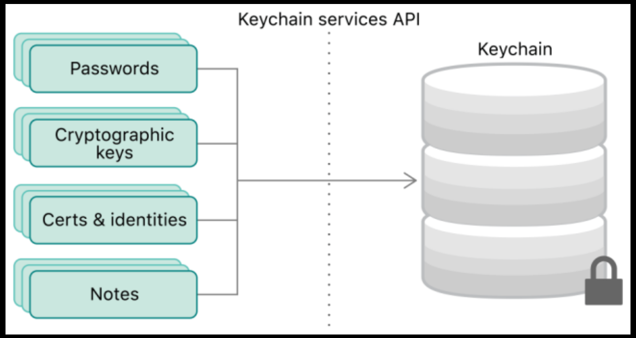


Source: [Apple’s documentation on the directory structure]

iOS platform

- Files are encrypted according to their data protection (set by the developer):
    - Complete protection (`NSFileProtectionComplete`): Encrypted by the user passcode + device UID (→ can only be decrypted on the device). Decrypted only when device is unlocked
    - Protected Unless Open (`NSFileProtectionCompleteUnlessOpen`): like Complete, but the file is still decrypted while the device is unlocked. The key is passcode+UID
    - Protected Until First User Authentication (`NSFileProtectionCompleteUntilFirstUserAuthentication`): Decrypted after first boot and always available (even when the device is locked). The key is passcode+UID
    - No Protection (`NSFileProtectionNone`): The key is just the UID. Can be read from or written to at any time

iOS platform



- Keychain
  - Protects secrets
    - One keychain for all apps (in MacOS there can be multiple keychains)
    - Secrets can be shared between apps (signed by the same developer)
  - Data protection similar to the data protection one:

Name	Description	Backed-up
`kSecAttrAccessibleWhenPasscodeSetThisDeviceOnly`	Only accessible while the device is *unlocked with a passcode*	NO
`kSecAttrAccessibleWhenUnlockedThisDeviceOnly`	Only accessible while the device is *unlocked*	NO
`kSecAttrAccessibleWhenUnlocked`	Only accessible while the device is *unlocked*	YES
`kSecAttrAccessibleAfterFirstUnlockThisDeviceOnly`	Only accessible after *first boot* (locked+unlocked)	NO
`kSecAttrAccessibleAfterFirstUnlock`	Only accessible after *first boot* (locked+unlocked)	YES
`kSecAttrAccessibleAlwaysThisDeviceOnly`	Can *always* be accessed	NO
`kSecAttrAccessibleAlways`	Can *always* be accessed	YES

iOS platform

- (still about the Keychain):
  - The authentication mechanism can be defined:
    - `kSecAccessControlDevicePasscode` (Passcode)
    - `kSecAccessControlBiometryAny` (Biometrics - **adding or removing a biometric will NOT invalidate the entry** // like `setInvalidatedByBiometricEnrollment(false)` on Android)
    - `kSecAccessControlBiometryCurrentSet` (Biometrics - adding or removing a biometric will invalidate the entry)
    - `kSecAccessControlUserPresence` (either Biometrics or passcode)
  - **The Keychain is NOT wiped after uninstalling**
  - Backed-up (except for the entries specifically disabled), but still encrypted (UID)

iOS platform

- IPA
  - Again, just a Zip with a specific structure
  - Important files:
    - `Info.plist`: config info (bundle ID, supported devices, …)
    - `Settings.bundle`: App preferences (can be changed from the settings menu)
    - `<app>.entitlements`: Requested permissions, registered URL schemes, …
  - After installing:
    - Payload goes to `/var/containers/Bundle/Application/<App ID>/<App>.app/`
    - Data stored in `/var/mobile/Containers/Data/Application/<App ID>/`

```
└─$ l Payload/Runner.app/
total 916
drwxr-xr-x 4 kali kali   4096 Aug 15  2020 .
drwxr-xr-x 3 kali kali   4096 Aug 25 12:03 ..
-rw-r--r-- 1 kali kali    399 Aug 15  2020 AppFrameworkInfo.plist
-rw-r--r-- 1 kali kali   1911 Aug 15  2020 AppIcon20x20@2x~ipad.png
-rw-r--r-- 1 kali kali   1911 Aug 15  2020 AppIcon20x20@2x.png
-rw-r--r-- 1 kali kali   2532 Aug 15  2020 AppIcon20x20@3x.png
-rw-r--r-- 1 kali kali    921 Aug 15  2020 AppIcon20x20~ipad.png
-rw-r--r-- 1 kali kali   3061 Aug 15  2020 AppIcon29x29@2x~ipad.png
-rw-r--r-- 1 kali kali   3061 Aug 15  2020 AppIcon29x29@2x.png
-rw-r--r-- 1 kali kali   6964 Aug 15  2020 AppIcon29x29@3x.png
-rw-r--r-- 1 kali kali   1269 Aug 15  2020 AppIcon29x29~ipad.png
-rw-r--r-- 1 kali kali   1269 Aug 15  2020 AppIcon29x29.png
-rw-r--r-- 1 kali kali   6221 Aug 15  2020 AppIcon40x40@2x~ipad.png
-rw-r--r-- 1 kali kali   6221 Aug 15  2020 AppIcon40x40@2x.png
-rw-r--r-- 1 kali kali  11644 Aug 15  2020 AppIcon40x40@3x.png
-rw-r--r-- 1 kali kali   1911 Aug 15  2020 AppIcon40x40~ipad.png
-rw-r--r-- 1 kali kali  11644 Aug 15  2020 AppIcon60x60@2x.png
-rw-r--r-- 1 kali kali  25420 Aug 15  2020 AppIcon60x60@3x.png
-rw-r--r-- 1 kali kali  20296 Aug 15  2020 AppIcon76x76@2x~ipad.png
-rw-r--r-- 1 kali kali   5578 Aug 15  2020 AppIcon76x76~ipad.png
-rw-r--r-- 1 kali kali  20910 Aug 15  2020 AppIcon83.5x83.5@2x~ipad.png
-rw-r--r-- 1 kali kali 540207 Aug 15  2020 Assets.car
drwxr-xr-x 4 kali kali   4096 Aug 15  2020 Base.lproj
drwxr-xr-x 8 kali kali   4096 Aug 15  2020 Frameworks
-rw-r--r-- 1 kali kali   1420 Aug 15  2020 Info.plist
-rw-r--r-- 1 kali kali      8 Aug 15  2020 PkgInfo
-rw-r--r-- 1 kali kali 200280 Aug 15  2020 Runner
```

```
└─$ file Payload/Runner.app/Runner
Payload/Runner.app/Runner: Mach-O universal binary with 2 architecture
s: [armv7:\012- Mach-O armv7 executable, flags:<NOUNDEFS|DYLDLINK|TWOL
EVEL|PIE>] [arm64:Mach-O 64-bit arm64 executable, flags:<NOUNDEFS|DYLD
LINK|TWOLEVEL|PIE>]
```

iOS platform

- Inter-Process Communication (IPC)
  - Not as much IPC as in Android

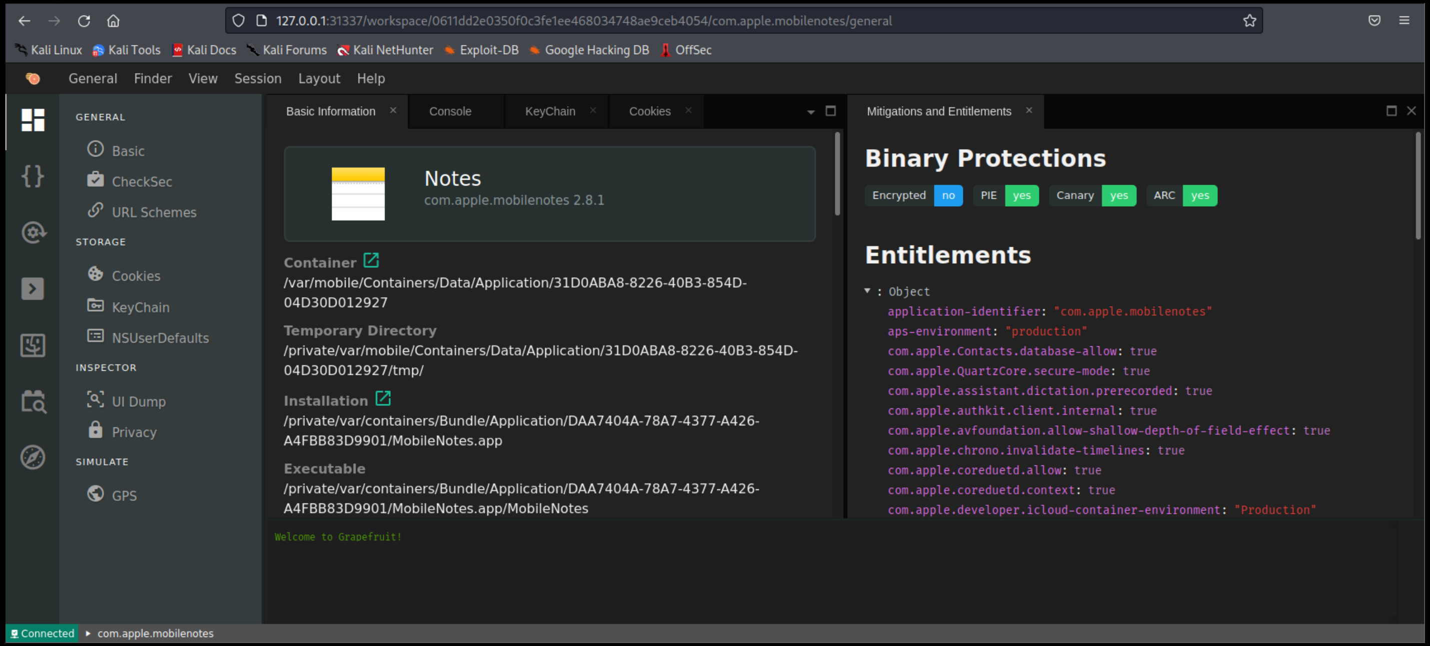
  - XPC Services
  - Mach Ports
  - NSFileCoordinator1

  - Deep-links:
    - Custom URL scheme (roles: Editor || Viewer): `CFBundleURLTypes` within `Info.plist`
    - Universal links (like verified links on Android): `com.apple.developer.associated-domains`, inside `<App>.entitlements`

    - Support for opening files: `CFBundleDocumentTypes` (within `Info.plist`)

iOS platform

- Interaction through libimobiledevice
  - Shell with OpenSSH (cydia - must be jailbroken) → `mobile:alpine` // `root:alpine`
  - Read logs with `idevicesyslog`
  - Install apps with `ideviceinstaller`
- Another useful tool:
  - https://github.com/chichou/grapefruit



Prepare the lab

Prepare the lab

- Tools that we might want to use:
  - https://github.com/frida/frida/ (`pip instal frida-tools`)
  - https://github.com/sensepost/objection (`pip instal objection`)
  - https://github.com/chichou/grapefruit (`npm install -g igf`)
  - https://developer.android.com/studio/releases/platform-tools (for adb and that stuff)
    - Alternative: install Android Studio https://developer.android.com/studio/
  - https://mobsf.github.io/docs/#/installation
  - https://libimobiledevice.org/
  - https://ibotpeaches.github.io/Apktool/ (to compile and decompile apk)

  - https://docs.darlinghq.org/build-instructions.html
    - Or MacOS for some tools like otool or to compile things with Xcode
  - IDA / Ghidra / Binary Ninja / something to decompile Mach-O (arm)

Prepare the lab

1. Obtain the APK/IPA
   - Download from the app store (the IPA has to be decrypted → obtain it from memory at runtime)
   - Ask the client for the binaries
     - If there are anti-RE protections (cert. pinning, root/jailbreak detection, …), we could ask the customer for a non-protected version, in case the anti-RE is well implemented

2. Install on the device (if not already done) - assuming it's already rooted/jaibroken
   - `adb install` or `cat app-debug.apk | pm install -S <size in Bytes>` (`-d` to allow downgrade) ← Damned Redmi...
   - `ideviceinstaller -i <app.ipa>`

3. Set up hotspot FROM KALI (to intercept traffic), proxy (device settings) and Burp

Prepare the lab

BREAK!

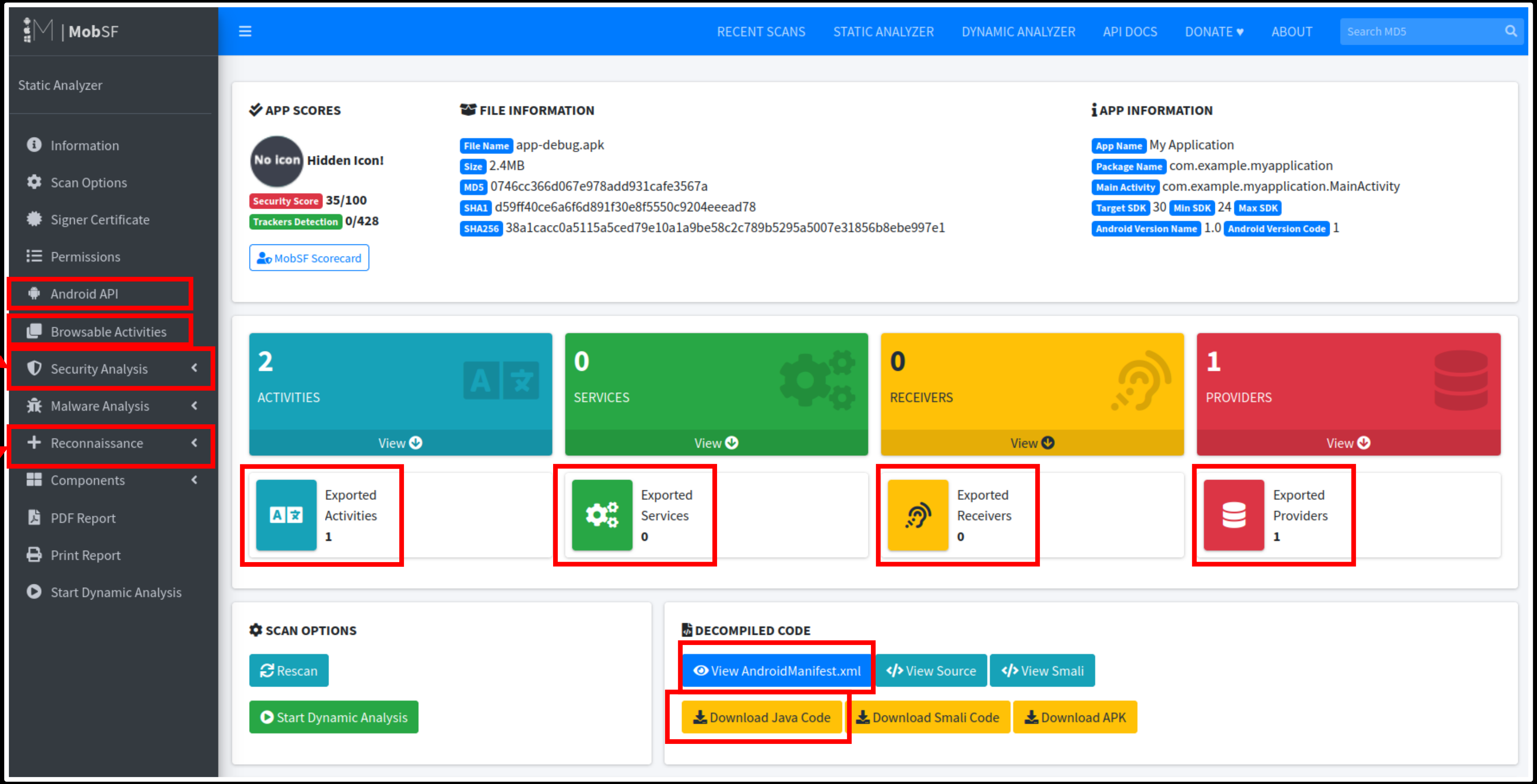
(5-10’?)

Static analysis

Static analysis – intro to MobSF

Config, code quality, …

Strings, hardcoded secrets, …

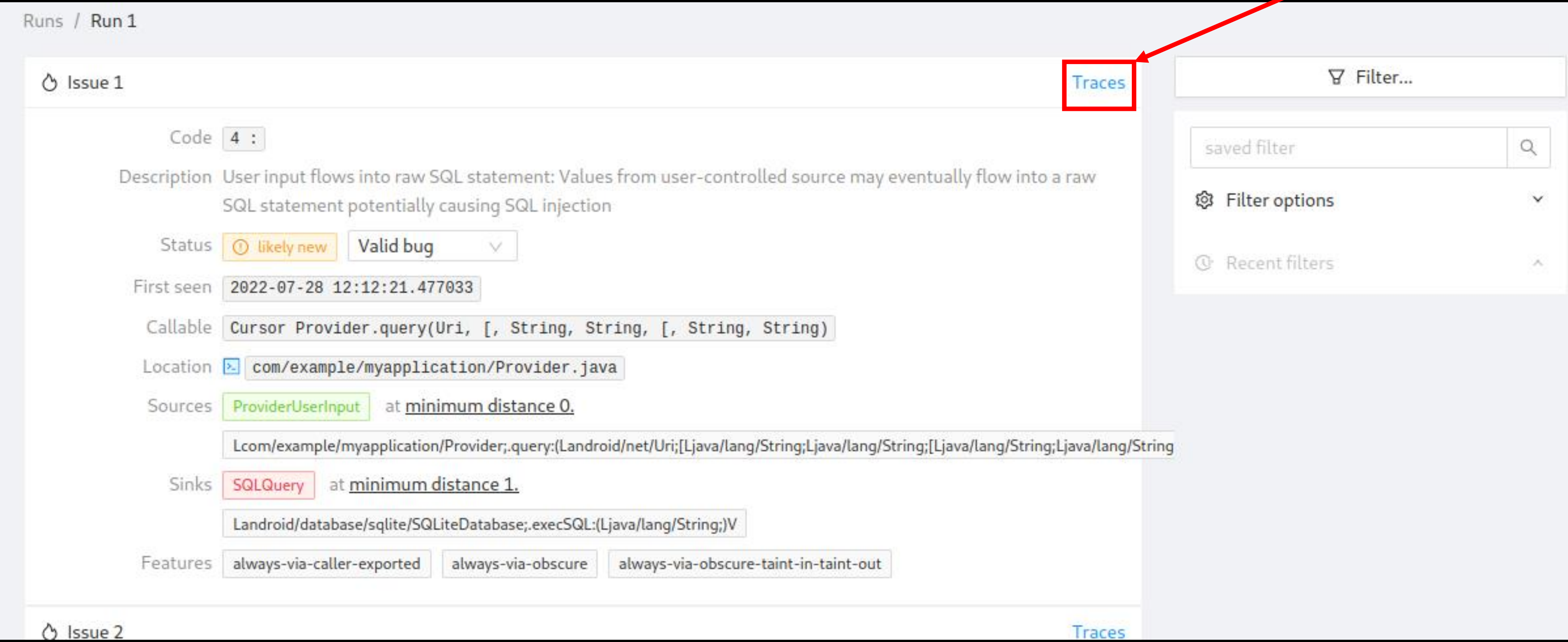


We will use it later and you will learn it in no time, don't worry ;)

Static analysis – Mariana Trench demo

- https://github.com/facebook/mariana-trench (instructions on README.md)
- Static Analysis of source code
  - We can try to use the decompiled code created by MobSF

Get more info about the issue



Static analysis – Mariana Trench demo

- We can use the example application (https://github.com/facebook/mariana-trench/tree/main/documentation/sample-app) to familiarise ourselves
- Issues detected:
  1. SQLi on `com/example/myapplication/Provider.java`
     - Can be exploited with `adb shell content query --uri content://com.example.Provider` or with a custom malicious app (“PoC 1 – AttackerQueryContent.7z”)
  2. SQLi on `com/example/myapplication/Provider.java` (basically the same as issue 1)

  3. Arbitrary file write on `com/example/myapplication/Provider.java`
     - `adb shell am start -a android.intent.action.VIEW -n "com.example.myapplication2/com.example.myapplication.MainActivity" -d "https://example.com/" --es "incoming_url" "https://asdf.com" --es "log_urls" "../yxcv"`
  4. XSS on `com/example/myapplication/Provider.java` (Same exploit as issue 3, but using a XSS payload)

Static analysis – Mariana Trench demo

5. RCE on `androidx/fragment/app/FragmentManagerImpl`
   - `adb shell am start -a android.intent.action.MAIN -n "com.example.myapplication/.MainActivity" --es "command" "sh"`, and check the running processes with `adb shell "ps -e | grep \ sh --color"`
   - Not really exploitable (unless there's already a malicious binary) due to the way `Java.exec()` is used
6. RCE on `androidx/fragment/app/FragmentManagerImpl` (same as 5, with another path)

7. (False positive)
8. (False positive)

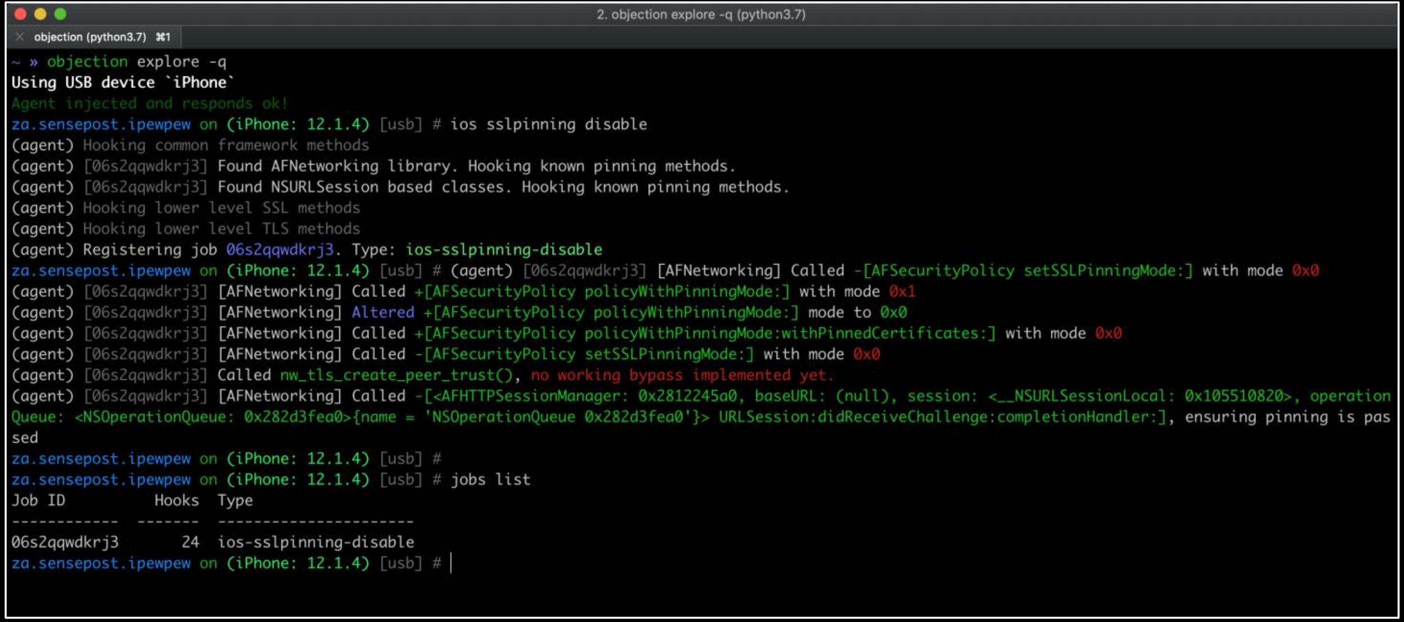
9. RCE on `androidx/fragment/app/FragmentManagerImpl` (same as 5, with another path)

10. RCE via arbitrary class load
    - `am start -a android.intent.action.MAIN -n "com.example.myapplication/.MainActivity" --ez "redirect" "true" --es "component" "com.example.myapplication.WebViewActivity"`
    - Possible scenario: a malicious app forces the legitimate app to launch a phishing screen on the malicious app, or any other internal vulnerable activity on the victim app
    - (check `logcat *:S AndroidRuntime:D TEST:D` for exceptions)

Dynamic analysis

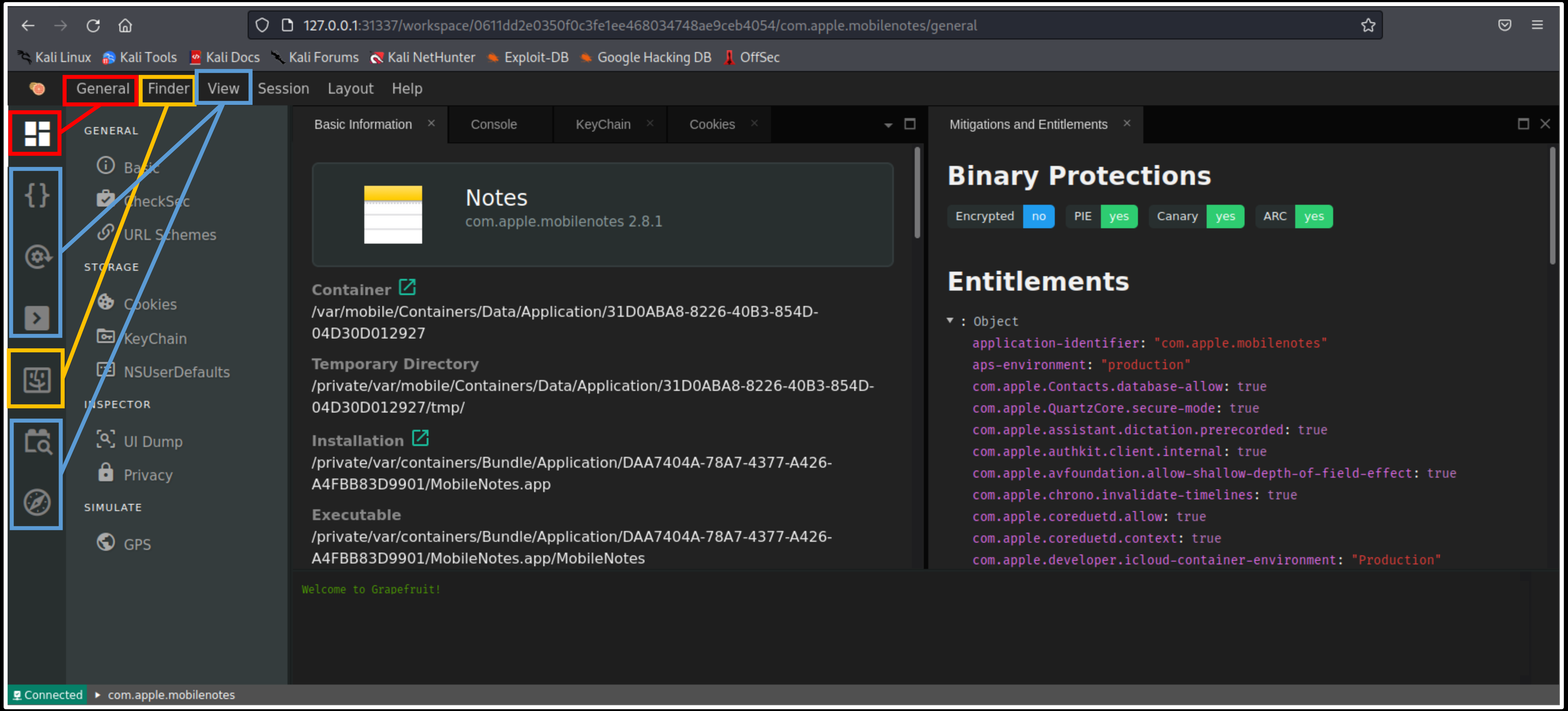
Dynamic analysis – intro to Frida and Objection

- Frida
  - Injects a JS engine into the running process
    - Can hook functions to change params, return value, or just inspect data
  - Requires either root access, or patching the app
  - Example patcher (shameless self-plug XD): https://github.com/Foo-Manroot/apk-patcher/
- Objection
  - Leverages Frida with custom scripts to easily interact with the app (cert pinning bypass, view data, query the keychain/keystore, take screenshots, …)



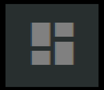
Source: objection's Wiki page

Dynamic analysis – intro to Grapefruit



*(also leverages Frida)

Dynamic analysis – intro to Grapefruit

- “General” ()
  - “Basic” -> info like decoded Info.plist, installation directories, etc.
  - “Checksec”: Binary protections (PIE, Canary and ARC). Make sure to read https://sensepost.com/blog/2021/on-ios-binary-protections/
  - “URL schemes”: useful to check deep-linking

  - “Cookies”, “Keychain”: self-explanatory
  - “UserDefaults”: settings

  - “UI Dump”: no clue ☹ It looks to be just the UI hierarchy, but I don’t know how to interpret this…
  - “Privacy”: permissions

  - “GPS”: can simulate any GPS location

Dynamic analysis – intro to Grapefruit

- "Runtime Classes" ({}): can select any class to show the decompiled code
- "Process Modules" ( ): lists the loaded libraries, and can maybe even decompile
- "REPL" ( ): Create and run custom Frida scripts
- "Finder" ( ): File explorer
- "Search API" ( ): I'm not sure about the syntax, but allows to search modules
- "WebViews and JavaScriptCore Instances" ( ): Allows arbitrary JS injection on WebViews, and inspection of scripts already present

Dynamic analysis

- Final notes
  - It’s always good to have multiple tools at hand (i.e.: grapefruit or objection might kill the device if it’s not powerful enough, so bare Frida or even pure gdb/lldb might be needed)

  - Intercepted network traffic can have multiple formats:
    - HTTP + JSON/XML (most common)
    - HTTP + protobuf (not uncommon, either)
    - HTTP + other binary (I haven’t really seen it in the wild)
    - Full binary/custom protocol on bare sockets (possible, *in theory*…)

OWASP Mobile Top 10 (2016)

OWASP Mobile Top 10 (2016)

- M1: Improper Platform Usage
  - "misuse of a platform feature or failure to use platform security controls. It might include Android intents, platform permissions, misuse of TouchID, the Keychain, or some other security control that is part of the mobile operating system."
  - (e.g.: using local storage instead of the Keychain)

- M2: Insecure Data Storage
  - "Threats agents include the following: an adversary that has attained a lost/stolen mobile device; malware or another repackaged app acting on the adversary's behalf that executes on the mobile device."
  - (e.g.: storing credentials in a local file, that can also be backed-up on the cloud)

OWASP Mobile Top 10 (2016)

- M3: Insecure Communication
    - “Threat agents might exploit vulnerabilities to intercept sensitive data while it’s traveling across the wire”
    - (e.g.: accepting any TLS certificate, or even using plaintext)

- M4: Insecure Authentication
    - “Once the adversary understands how the authentication scheme is vulnerable, they fake or bypass authentication by submitting service requests to the mobile app’s backend server and bypass any direct interaction with the mobile app”
        - Quite related with web API
    - (e.g.: using a 4-digit PIN as a password for the account)

OWASP Mobile Top 10 (2016)

- M5: Insufficient Cryptography
  - “Threat agents include the following: anyone with physical access to data that has been encrypted improperly, or mobile malware acting on an adversary’s behalf.”
  - (e.g.: rolling your own crypto, or using deprecated algos)

- M6: Insecure Authorization
  - “Once the adversary understands how the authorization scheme is vulnerable, they login to the application as a legitimate user”
    - Again, very related to web API
  - (e.g.: IDOR)

- M7: Client Code Quality
  - “Threat Agents include entities that can pass untrusted inputs to method calls made within mobile code”
  - (e.g.: buffer overflow via malicious deep-link)

OWASP Mobile Top 10 (2016)

- M8: Code Tampering
  - “Typically, an attacker will exploit code modification via malicious forms of the apps hosted in third-party app stores. The attacker may also trick the user into installing the app via phishing attacks.”
  - (e.g.: using a vulnerable signature scheme - CVE-2017-13156 a.k.a. Janus)

- M9: Reverse Engineering
  - “An attacker will typically download the targeted app from an app store and analyze it within their own local environment using a suite of different tools”

- M10: Extraneous Functionality
  - “Typically, an attacker seeks to understand extraneous functionality within a mobile app in order to discover hidden functionality in in backend systems. The attacker will typically exploit extraneous functionality directly from their own systems without any involvement by end-users.
  - (e.g.: leftover debug code or dev environments/keys)

Hands-on: let’s test something! (Android edition)

Hands-on: Android

- https://github.com/dineshshetty/Android-InsecureBankv2
- Run the server from `/opt/test-apps/Android-InsecureBankv2-master/start_server.sh`
- The APK is under `/opt/test-apps/Android-InsecureBankv2-master/InsecureBankv2.apk`
    - `adb install InsecureBankv2.apk`
- Run Burp to intercept traffic
- Exercises:
    1. Try to find a way to create a user (still WIP functionality!!)
    2. Find and exploit a vulnerable broadcast receiver

Hands-on: let’s test something! (iOS edition)

Hands-on: iOS

- https://github.com/prateek147/DVIA-v2
- (maybe AppSync Unified is needed -> https://cydia.akemi.ai // and also Frida -> https://build.frida.re)

- Step 1 (common for everyone): "**Network Layer Security**"

- Step 2: Each team has to:
  - select one vulnerability / use-case
  - develop an exploit
  - explain one vuln to the other teams

Wrapping-up: Methodology

Wrapping-up: proposed methodology

My proposal (heavily based on the MSTG):

1. Evaluate data requisites (classification from https://mobile-security.gitbook.io/masvs/0x03-using_the_masvs#verification-levels-in-detail):
    - Level 1: no special requisites, besides the usual security measures
    - Level 2: May require extra security (like encrypting local storage)
    - Level R: Extra anti-RE measures (cert. pinning, code obfuscation, ...)

    - L1: All regular apps
    - L2: Banking, health-care, ... Basically: handling of sensitive info/functionality
    - L1+R: Games (to avoid cheating) or stuff like that, where no sensitive info is at risk
    - L2+R: Banking, allowing to move funds and do more damage than with L2 apps; or something, idk

(check MASVS for the complete list of L1, L2 and LR measures)

Wrapping-up: proposed methodology

2. Static analysis with MobSF and (if possible) with Mariana-Trench
    - Focus on IPC (time is limited → 1-2 days max., on a 5-day engagement including web API)

3. Bypass certificate pinning and root/jailbreak detection (if necessary)
    - Pre-requisite for network interception

4. Network traffic inspection / API testing

Wrapping-up: proposed methodology

5. Platform-specific tests:

Android:

- `AndroidManifest.xml`:
  - `usesCleartextTraffic` should be set to false (or not appear) → [Info], if no cleartext traffic is observed (or nothing useful is leaked)
  - content providers
  - broadcast receivers
  - services
  - handlers for deep-links
  - handlers for file types
- backed-up data (sensitive files backed-up to GCloud)
- logcat
- Check if JS is enabled on WebViews (XSS)

iOS:

- Info.plist
  - `NSAllowsArbitraryLoads` should be false (or not appear) → [Info], if no cleartext traffic is observed (or nothing is leaked)
  - `CFBundleDocumentTypes` (file handlers)
- `<app>.entitlements`
  - custom URL schemes without the "Editor" role set up
  - handlers for deep-links
- `kSecAccessControlBiometryAny`: Register another fingerprint and try to access
- backed-up data (sensitive files backed-up to iCloud + backed-up Keychain ← leakage when reinstalling an app after selling it refurbished, for example)
- `Idevicesyslog`
- Check if JS is enabled on WebViews (XSS):
  - `UIWebView` **always** has JS
  - `WKWebView` (JS **enabled by default**)
  - `SFSafariViewController` **always** has JS