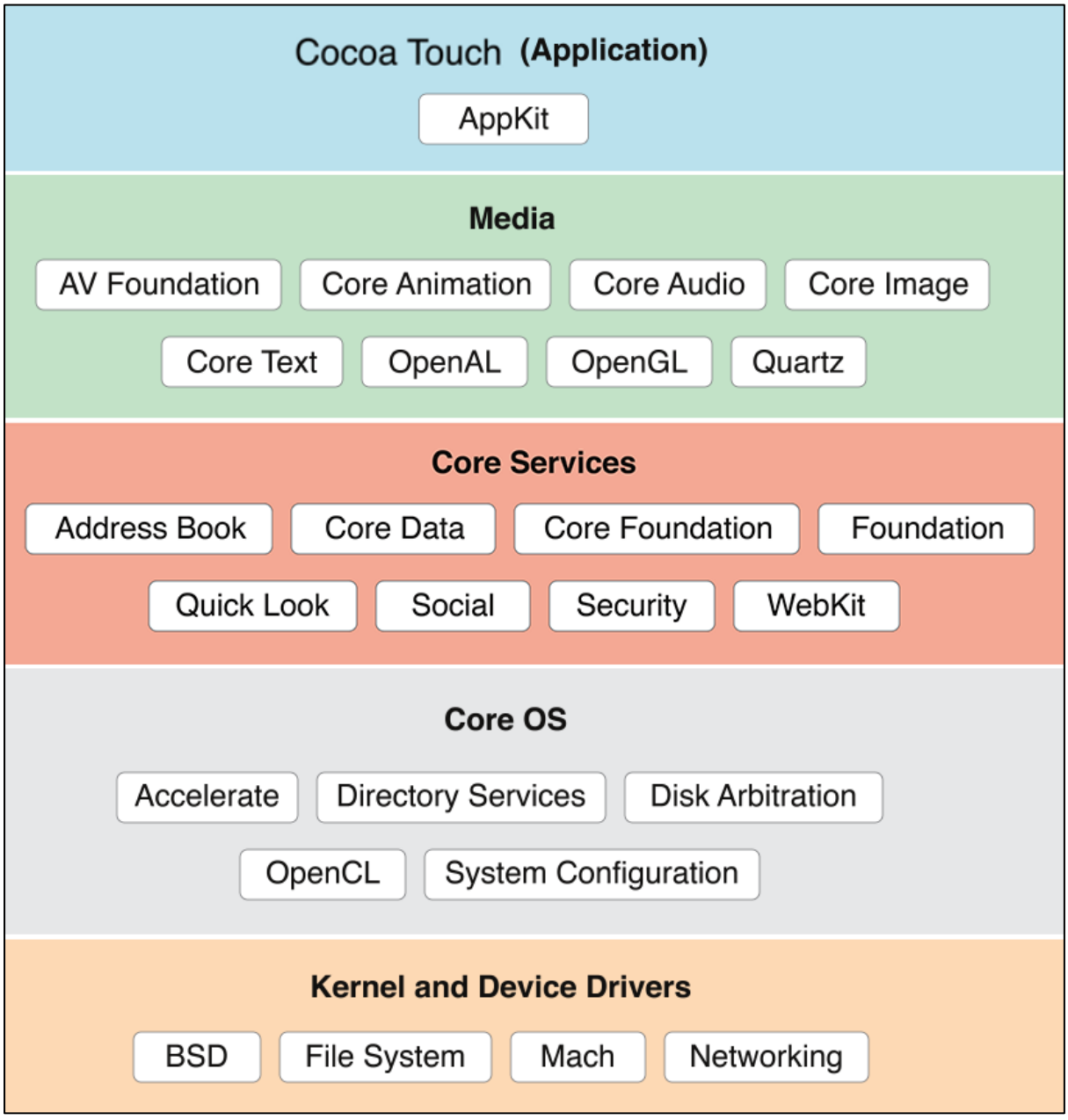
iOS Application Penetration Testing Training Module

Contents

1. iOS Architecture



The diagram explains the different layers that are used between the Application and Hardware level to establish communication & perform different tasks.

1. **Kernel and Device Drivers:** is the lowest layer of iOS which mainly includes the kernel and device drivers.

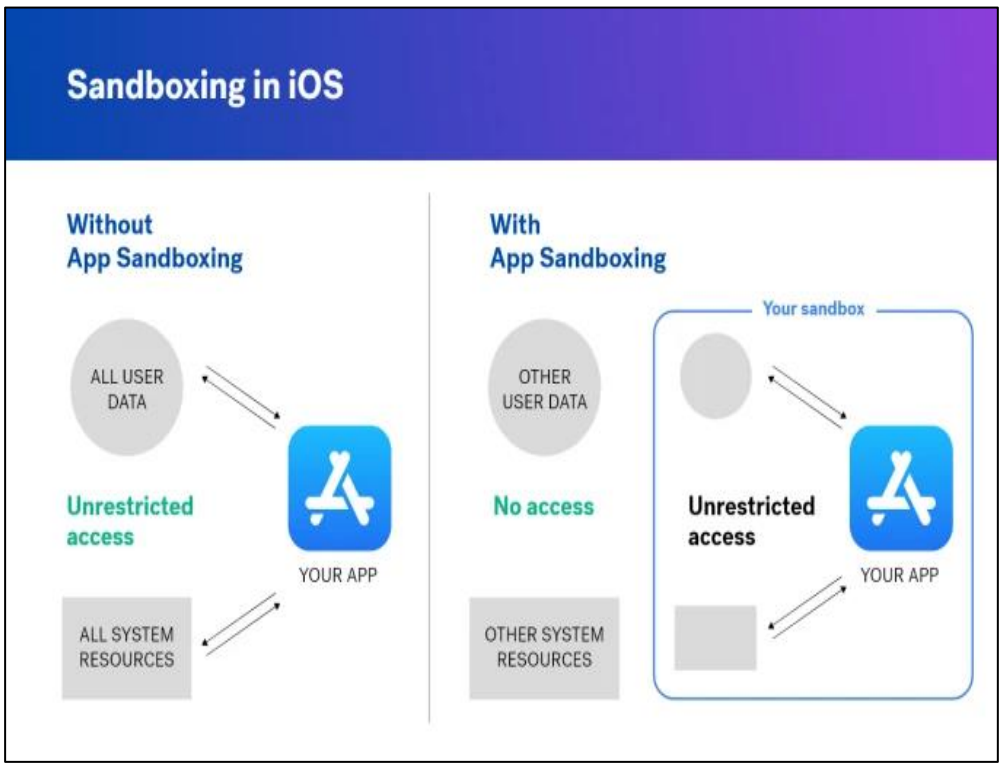
2. **Core OS Layer:** consists of technologies and frameworks which provide low-level services related to low-level hardware and networks. These include Accelerate Framework, Directory Services, System Configuration, memory management, file system handling and threads.

3. **Core Services Layer:** consists of core services that provide essential features to the application. It gives access to fundamental resources needed for an application. These services generally include Address book, Security, Social, foundation, Webkit, etc.

4. **Media Layer:** provides various multimedia services that can be utilized in the device, which basically enables all the audio-visual technologies (2D and 3D graphics, animations, image effects, professional-grade audio and video functionalities). It provides various functions such as Core Image, Core Audio, Core Text, etc.

5. **Cocoa Touch Layer:** is the topmost layer in the architecture and is also known as the Application Layer, which is primarily responsible for the application's appearance. It exposes various APIs for programming the iPhone devices and provides access to the main system functions like Contacts, Camera, touch input, shares with other apps, push notifications, etc.

- ## iOS Application Sandboxing
    - **Sandbox** is a directory on the iOS file system which allows applications to set up a local database on the device and separately segment from the other applications.
    - It protects an application from other applications trying to make unauthorized access to any data that might be stored, such as passwords, payment information, and personal data like photos.



    - To ensure there are no conflicts between the application, iOS assigns each app its own sandbox.
    - There are 3 types of sandboxes in an iDevice:

1) **Pre-installed App Directory(/var/Application):** The applications that come pre-installed by default on the iDevice have their app files stored in this directory.
2) **Bundle Directory(/var/containers/Bundle/Application):** The "Bundle" directory, also known as the "IPA Container," contains all of the files that come with apps when downloaded from the Apple App Store. Also, downloaded from other locations as well. The files in this directory will remain the same throughout a particular application version.
3) **Data Directory(/var/mobile/Containers/data/Application):** The "Data" directory, also known as the "Local Data Storage Container," contains files the developer wants to keep. These are items that belong to the application while installed on the device.

2. iOS Application Structure

- iOS Applications use ".ipa" (iOS application archive) as a file extension, which is nothing but a compressed file.
- iOS applications are developed using X-code IDE and objective-c or Swift programming languages.
- The .ipa file structure is as follows:

```
/Payload/
/Payload/Application.app/
/iTunesArtwork
/iTunesArtwork@2x
/iTunesMetadata.plist
/WatchKitSupport/WK
/META-INF
```

  - The "/Payload/" folder contains all the application data.
  - The Code signing is handled in the "/Payload/Application.app/" bundle directory.
  - The "/iTunesArtwork" file is a 512×512-pixel PNG image used as the application's icon.
  - The "/iTunesMetadata.plist" contains various bits of information, ranging from the developer's name and ID, the bundle identifier, copyright information, genre, the name of the application, release date, purchase date, etc.
  - The "/META-INF" folder only contains metadata about what program was used to create the IPA.

- A closer look at the "IPA" file:
    - MyApp: contains the compiled application source code.
    - Application: Contains application icons.

- Info.plist: Contains the configuration information, such as bundle ID, version number and application display name. This file is often checked while performing security assessments as it may contain interesting information or help us find some misconfigurations.
- Launch images: Images showing the initial application interface in a specific orientation. The system uses one of the provided launch images as a temporary background until the application is fully loaded.
- MainWindow.nib: Default interface objects that are loaded when the application is launched. Other interface objects are then either loaded from other nib files or created programmatically by the application.
- Settings.bundle: Application-specific preferences to be displayed in the Settings app.
- Custom resource files: Non-localized resources are placed in the top-level directory and localized resources are placed in language-specific subdirectories of the application bundle. Resources include nib files, images, sound files, configuration files, strings files, and any other custom data files the application uses.

- A keychain is referred to as an encrypted container where an application can store sensitive information and only the authorized application can retrieve the data from it.

3. Jailbreak

- Jailbreak is the method of lifting user restrictions imposed by the manufacturer. There are various jailbreaking methods for iOS - differing from version to version.
- Different types of jailbreak are:
    1. **Untethered Jailbreak:** is a permanent type of jailbreak where even after rebooting the device, it will be in a jailbroken state.
    2. **Tethered Jailbreak:** is a temporary jailbreak type. Once the device is rebooted, the device no longer remains in the jailbreak state.
    3. **Semi-tethered Jailbreak:** is one where the device can start up on its own, but it will no longer have a patched kernel and therefore will not be able to run modified code.
    4. **Semi-untethered Jailbreak:** is similar to an untethered jailbreak in that it allows the device to boot up on its own. The device start-up sequence is unaltered on each boot, and it boots into its original, non-jailbroken state. However, rather than using a computer to jailbreak, as in a tethered or semi-tethered case, the user can re-jailbreak their device using an app that runs on their device.

4. iOS Application Security Approach

1. Take a positive walkthrough from a developer's perspective to know the application.
    a. Ask for ⇒ Description, functionality, owners, version, application workflow, no.of static and dynamic modules, user roles, user credentials for each user role (minimum 3), limitations of the application.
    b. Check the walkthrough from our end. Verify given details are correct and working properly, if not set them as limitations.
    c. Get the "IPA" file from the client.
2. The application should be installed on a jailbroken device.
3. SSL pinning should be disabled in the "IPA" file provided.
4. Reasons:
    a. To perform an end-to-end assessment, we need SSL pinning disabled and the application should be installed on a jailbroken device.
    b. If the application is installed on a jailbroken device, then only we can analyse all the internal packages with root privilege.
    c. If SSL pinning is disabled, then only we'll be able to capture and analyse each request response of the application.
5. Check for sensitive information stored in plist files.
6. Check for sensitive information stored in local databases (.db, .sqlite, .sqlite3).
7. Check request response in burp suite.

5. Important Directories/Files to Look for in iOS

1. Insecure Data Storage (.plist files)
    - Plist File in the App Directory: Plist (Property List) is a flexible and convenient format for storing application configuration data. It can be called as the manifest for an iOS application. It decides what icon to use for a bundle, what document types an app can support, and many other behaviours that have an impact outside the bundle itself. These files may contain sensitive data.
    - NSUserDefaults: is one of the most common methods of saving user preferences and properties in an application using NsUserDefault. Even if you close the application and relaunch it, the NsUserDefault information persists. It stores data in "Plist" file format under the preference folder.

        - Navigate to /var/mobile/Containers/Data/Application directory.
        - Search for the application: find | grep "<app_name>".
        - Navigate to the folder containing the application data: cd <app_directory>.
        - Search for ".plist" files: find ./ -name "*.plist".
        - Now establish SFTP connection: sftp root@<iOS_device_ip>.
        - Retrieve the ".plist" file to the machine: get -r /<path>/<filename.plist>.

2. Database Stored Locally without Encryption(.db/.sqlite/.sqlite3)
    - Database Files in the App Directory: Data required by an iOS application is often stored in SQLite databases. Testers should look for sensitive information stored in database files. Some extensions we can look for are ".db", ".sqlite" and ".sqlite3".

        - Navigate to /var/mobile/Containers/Data/Application directory.
        - Search for the application: find | grep "<app_name>".
        - Navigate to the folder containing the application data: cd <app_directory>.
        - Search for ".plist" files: find ./ -name "*.db", or find ./ -name "*.sqlite", or find ./ -name "*.sqlite3"
        - Now establish SFTP connection: sftp root@<iOS_device_ip>
        - Retrieve the ".db" or "sqlite" or "sqlite3" file to the machine: get -r /<path>/<filename.extension>

- Keychain: is a password and certificate management system for iOS. It can be used to securely store sensitive bits of data, such as encryption keys and session tokens. Keychain is implemented as an SQLite database that can only be accessed through the Keychain APIs. It is common storage for all WIFI passwords as well as application data.

All these items are stored in an encrypted database whose path is "/var/Keychains/keychain-2.db".

6. Extracting an IPA file

1. The iDevice should be on the same network and connect to the iPhone by establishing SSH connection: ssh root@<iOS_device_ip> (Default Password is Alpine/alpine).
2. Navigate to the following directory: /var/containers/Bundle/Application
3. Search for the application: find | grep "<app_name>"
4. Navigate to the folder containing the application: cd <app_directory>
5. Create a directory with the name Payload: mkdir Payload
6. Copy the "appname.app" data into the Payload directory: cp -r <appname.app> / Payload/
7. Zip the Payload directory into IPA format: zip -r /var/root/<appname.ipa> Payload/
8. Now establish SFTP connection: sftp root@<iOS_device_ip>
9. Retrieve the ".ipa" file to the machine: get -r /var/root/<appname.ipa>

7. Running Static Analysis with MobSF

- After extracting the IPA file, the next step is to perform static analysis using MobSF.

  Install MobSF: https://github.com/MobSF/Mobile-Security-Framework-MobSF

  MobSF Documentation: https://mobsf.github.io/docs/#/

  NOTE: To analyse iOS application, MobSF has to be installed on Mac OS or in any Linux OS.

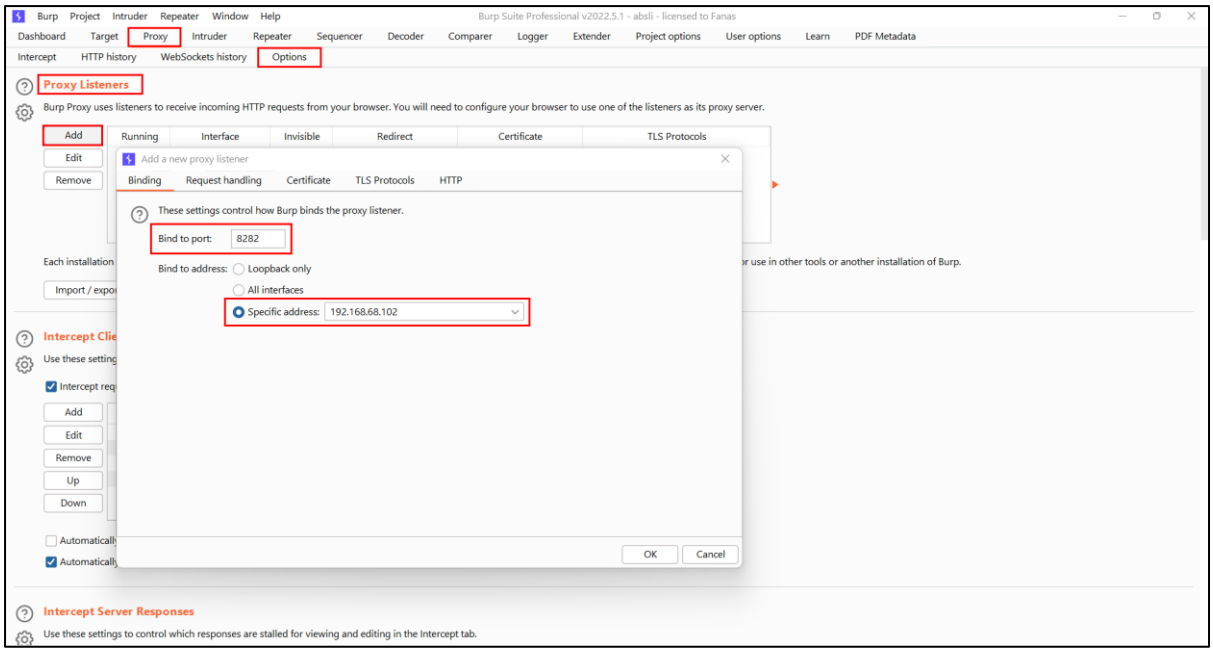
- MobSF: Mobile Security Framework (MobSF) is an automated, open source, all-in-one mobile application (Android/iOS/Windows) pen-testing framework capable of performing static, dynamic and malware analysis.

Steps to perform the static analysis:

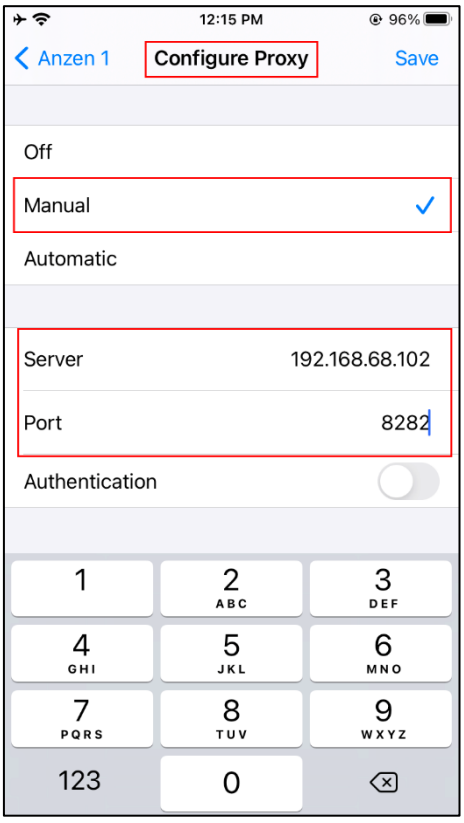
1. Run the MobSF Web Interface.
2. Drop the IPA file and run static analysis.
3. Once the static analysis is finished look for misconfigurations.

8. Setting Up Proxy

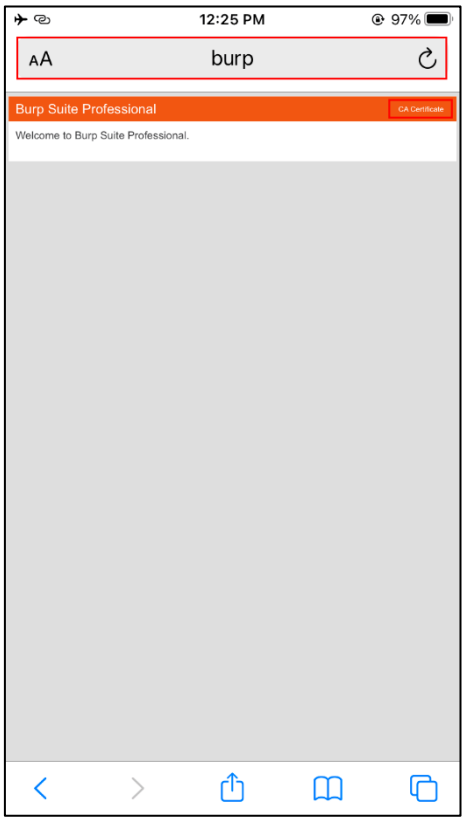
1. In the Burp Suite, go to Proxy → Options → Proxy Listeners, add a listener to Specific Interfaces with IP address and port.



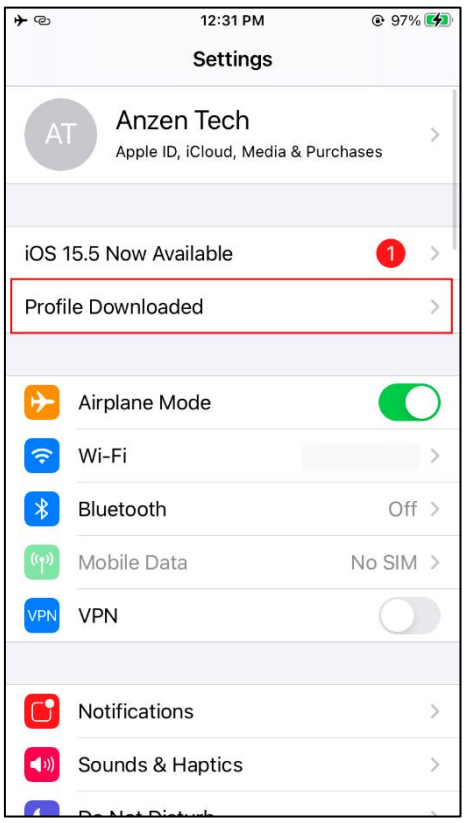
2. In iPhone device, enable proxy to manual in the wireless setting, add proxy details with IP address and port as specified in burp suite.



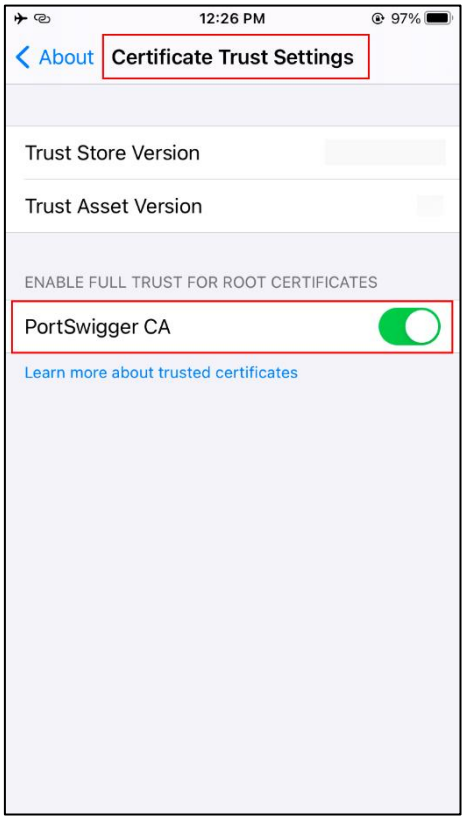
3. Open the browser, visit “http://burp” and download the CA certificate.



4. Go to Settings → Profiles → Install the CA certificate.



5. Navigate to Settings → General → About → Certificate Trust Settings and enable Portswigger CA Certificate.



6. Now turn “Intercept on” in the burp suite proxy and start capturing and analysing the request response.

09. Introduction to OWASP Mobile Top 10

1. **Improper Platform Usage:** refers to improper or mismanaged use of mobile platform security controls, where an attacker can abuse the security control features provided by the iOS platform. This can be anything from file permissions, microphone permissions, biometric authentication (Touch/Face ID), keychain, platform permission, etc.

2. **Insecure Data Storage:** An iOS application may often store some data locally in different components or when the developers avoid encryption of sensitive data and store that in clear text, it can be exploited if an attacker has access to the physical device of the user.

3. **Insecure Communication:** An iOS application communicates in a client-server architecture. Due to improper/weak implementation of the communication standards such as communication over HTTP, an attacker can attempt to steal sensitive information over unencrypted checks, perform Man-in-the-Middle or attempt to analyse the request/response by capturing them with proxy tools such as Burp Suite. SSL pinning can be used to ensure that all data is transferred between a web server and client securely.

4. **Insecure Authentication:** Weak Authentication is one of the root causes of many security risks, when an application fails to properly perform the authentication checks or allows an attacker to manipulate the login/authentication requests to gain access to the victim user's account, it is generally considered under the Insecure Authentication.

5. **Insufficient Cryptography:** Cryptography is the process of converting plain text data to an unreadable form. Most developers tend to ignore cryptography as it's complex to implement or use weak cryptography. Due to a lack of strong cryptography, an attacker may attempt to access the data that is encrypted and gain hold of sensitive information.

6. **Insecure Authorization:** Weak implementation of authorization mechanism leads to an improper authorization by which low-level users can access information of any high privileged user or can perform attacks that were out of normal user's privilege and access level. Improper authorization give rise to many business-level vulnerabilities as well. Some of the general risks that occur under this category are IDOR and privilege escalation (Vertical & Horizontal).

7. **Client Code Quality:** Due to poor code quality, an attacker may pass crafted inputs to function calls made within an app to execute them or to observe the application's behaviour. It may lead to some malfunction and exploitable scenarios in the application. Maintaining code quality while developing mobile applications is an essential task. Attacks such as buffer overflow, remote code execution, memory exhaustion happens due to bad code quality.

8. **Code Tampering:** Code Tampering refers to a process in which attackers exploit code modification via malicious forms. If the application does not implement the code tampering checks, an attacker may attempt to modify the application code and inject malicious code such as a backdoor and distribute it using third-party app stores available over the internet which can lead to stealing sensitive information and comprising the user's device as well.

9. **Reverse Engineering:** Reverse Engineering is a process to decompile the mobile application to understand the application logic and various function implementations. To prevent attackers from reading the application code and understanding the logic, code obfuscation can be done.

10. **Extraneous Functionality:** Before pushing an application to the production environment, the developers may keep codes to have easy access to the backend server, create logs to analyse errors or carry staging information and testing details. This code is extraneous to the functioning of the app. Such functionalities have no use to the end user and it is required only during the development cycle. Attackers try to understand these extraneous functionalities by exploring hidden functionalities of the backend framework. In certain cases, this code can carry information related to databases, user details, user permissions, API endpoints, etc.

10. Tools Used

- Filza
- Plist Editor
- DB Browser (SQLite)
- Mobile Security Framework (MobSF)
- Burp Suite
- FileZilla
- iMazing
- Frida
- Objection

11. References

- https://blog.yeswehack.com/yeswerhackers/getting-started-ios-penetration-testing-part-1/
- https://blog.yeswehack.com/yeswerhackers/getting-started-ios-penetration-testing-part-2/
- https://www.cobalt.io/blog/ios-pentesting-101