Advanced_Task-4

MOBILE APPLICATION PENTESTING

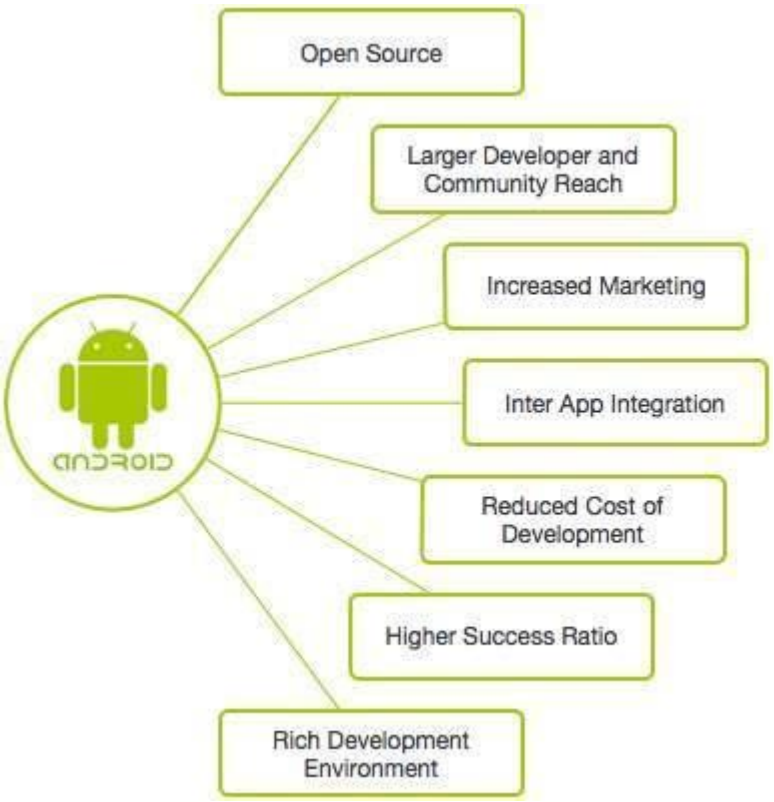
VENKATASAI PITHANI

BATCH NO:CF-08

Introduction to Android and Android Architecture:

1. Introduction to Android

Android is a mobile operating system developed by Google, designed primarily for touchscreen mobile devices like smartphones and tablets. It was initially developed by Android, Inc., which Google acquired in 2005. Since its launch in 2008, Android has gained significant popularity due to its open-source nature, flexibility, and customizability, enabling device manufacturers and developers to tailor it to various device requirements and applications. Android is built on a modified Linux kernel and other open-source software, making it a robust and flexible operating system.


Open Source
Larger Developer and Community Reach
Increased Marketing
Inter App Integration
Reduced Cost of Development
Higher Success Ratio
Rich Development Environment
ANDROID

- Android originated in 2003 with the founding of Android Inc. by Andy Rubin, Rich Miner, Nick Sears, and Chris White.

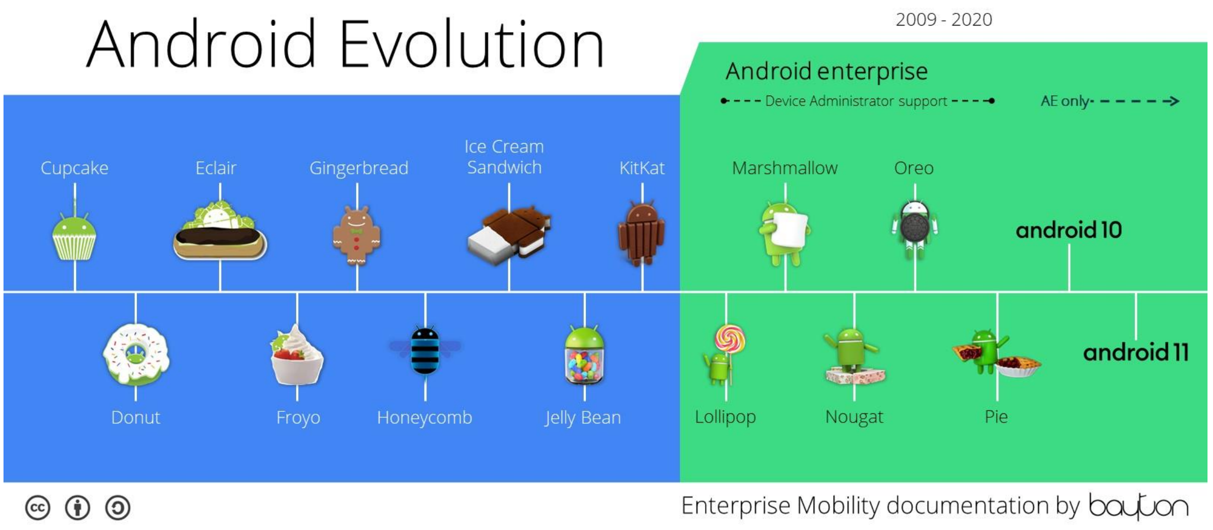
Key Features of Android:

- Open-Source: Developed under the Android Open Source Project (AOSP), Android allows developers to modify its source code, promoting innovation and customization.

- Google Services Integration: Android seamlessly integrates with Google's suite of services, including Gmail, Google Maps, Google Drive, and more.
- Application Ecosystem: The Google Play Store hosts millions of applications, supporting an extensive array of functionalities and catering to diverse user needs.
- Linux Kernel: Android's Linux-based kernel provides a stable foundation, ensuring device security, process management, and memory management.
- Versatility Across Devices: Android's adaptability extends across smartphones, tablets, watches (Wear OS), TVs (Android TV), and vehicles (Android Auto).

Evolution of Android:

The evolution of Android, the operating system developed by Google, has been remarkable since its inception. Here's a breakdown of the major milestones in Android's development:



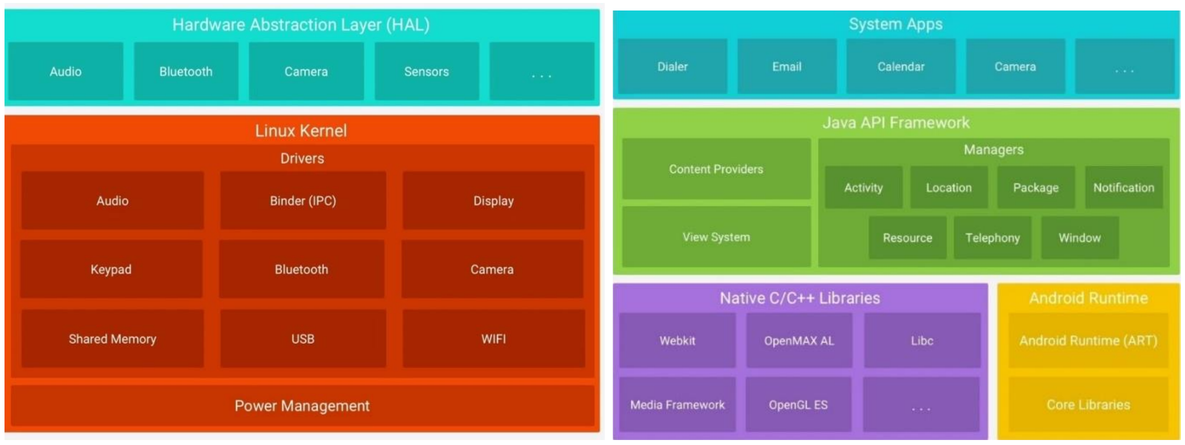
1. Android 1.0 and 1.1 (2008 - 2009)
2. Cupcake, Donut, and Éclair (2009 - 2010)
3. Froyo and Gingerbread (2010 - 2011)
4. Honeycomb (2011)
5. Ice Cream Sandwich (2011 - 2012)
6. Jelly Bean (2012 - 2013)
7. KitKat (2013-2014)
8. Lollipop (2014 - 2015)
9. Marshmallow (2015 - 2016)

10. Nougat (2016 - 2017)
11. Oreo (2017 - 2018)
12. Pie (2018 - 2019)
13. Android 10 (2019 - 2020)
14. Android 11 (2020 - 2021)
15. Android 12 (2021 - 2022)
16. Android 13 (2022 - 2023)
17. Android 14 (2023 - Present)

Each version of Android has consistently brought enhancements in user experience, performance, security, and customization, making it one of the most widely used mobile operating systems in the world.

2. Android Architecture

Android architecture is a stack of components that include a kernel, libraries, runtime, application framework, and applications. This layered architecture ensures that Android is modular, secure, and provides an efficient environment for app development and execution.



Components of Android Architecture:

1. **Linux Kernel (Foundation Layer):**
   - The core of Android's architecture is the Linux kernel, offering essential functionalities like process management, memory management, device management, and security.
   - It provides hardware abstraction, enabling the OS to run on a wide variety of devices and ensures efficient resource management and security.

2. **Hardware Abstraction Layer (HAL):**
    - HAL provides standard interfaces for device hardware features, allowing Android to access hardware components like cameras, audio, sensors, and more.
    - Through HAL, the OS can remain hardware-agnostic, promoting compatibility across diverse hardware configurations.
3. **Android Runtime (ART) and Core Libraries:**
    - **Android Runtime (ART):** ART is the managed runtime environment used in Android. It translates the bytecode from Android applications into machine code during installation, optimizing performance.
    - **Core Libraries:** These libraries provide essential APIs to developers, enabling them to build functional applications without needing to work directly with lower-level system components.
4. **Native C/C++ Libraries:**
    - Android includes various C/C++ libraries such as WebKit, OpenGL ES (for graphics), Media Framework (for playback of audio and video), SQLite (database engine), and SSL (for secure communication).
    - These libraries provide the foundational functionalities, which developers can use to enhance their apps.
5. **Application Framework:**
    - The application framework layer consists of APIs that developers use to build and manage Android apps.
    - Key services in the framework include:
        - **Activity Manager:** Manages the app lifecycle and navigational stack.
        - **Content Providers:** Enables apps to share data with other applications.
        - **Resource Manager:** Manages resources such as graphics and layouts.

        - **Notification Manager:** Enables apps to send user notifications.
        - **View System:** Used to create and manage UI components.

6. **Applications:**
    - At the top of the stack are the applications themselves, including both pre-installed system apps (such as Phone, Contacts, and Messages) and third-party apps downloaded from the Google Play Store.

3. Security in Android Architecture

Security is a fundamental design principle within Android's architecture, protecting users and their data. Android achieves this through multiple layers, from the Linux kernel to application isolation, ensuring that both system-level and user-level data remain secure.

Key Security Components:

1. **Linux Kernel Security:**
    - Android's Linux kernel enforces key security policies, including process isolation, user-based permissions, and secure IPC (Inter-Process Communication) through methods like Binder, enabling controlled access across applications.
2. **Application Sandbox:**
    - Every Android app runs in its isolated environment or "sandbox," preventing apps from accessing each other's data unless explicitly permitted.
    - The sandboxing feature mitigates the risk of malicious applications affecting other apps or core system functions.
3. **Permissions Model:**
    - Android has a granular permissions model that requires users to grant apps specific permissions at runtime.
    - This model minimizes risks by restricting an app's access to sensitive data or resources, aligning permissions with user consent and transparency.

4. **Android Security Frameworks and APIs:**
    - **KeyStore:** Android KeyStore allows apps to securely generate and store cryptographic keys within the device, enhancing data security.
    - **SafetyNet API:** This API helps developers detect if a device is secure (not rooted or compromised) and ensures app integrity.
    - **SELinux (Security-Enhanced Linux):** Android uses SELinux to enforce mandatory access controls, further reducing the potential impact of vulnerabilities.
5. **Secure Boot and Verified Boot:**
    - **Secure Boot** ensures that the device boots up using authentic firmware and software.
    - **Verified Boot** checks the integrity of the system at boot, ensuring the OS hasn't been tampered with, providing additional protection against rooted devices.
6. **Google Play Protect:**
    - Google Play Protect continuously scans apps on the Google Play Store and user devices, protecting users against potentially harmful apps (PHAs) and providing real-time protection.

An Introduction to iOS and iOS Architecture:

1. Introduction to iOS

iOS is a mobile operating system created by Apple Inc., designed exclusively for Apple's hardware, including iPhones, iPads, iPod Touches, and, to some extent, Apple TVs. Introduced in 2007, iOS revolutionized the mobile industry with its sleek, user-friendly interface and tight integration with Apple's ecosystem. Built on a Unix-based foundation, iOS combines robust security, smooth user experience, and consistent performance, making it one of the most popular and secure mobile operating systems worldwide.

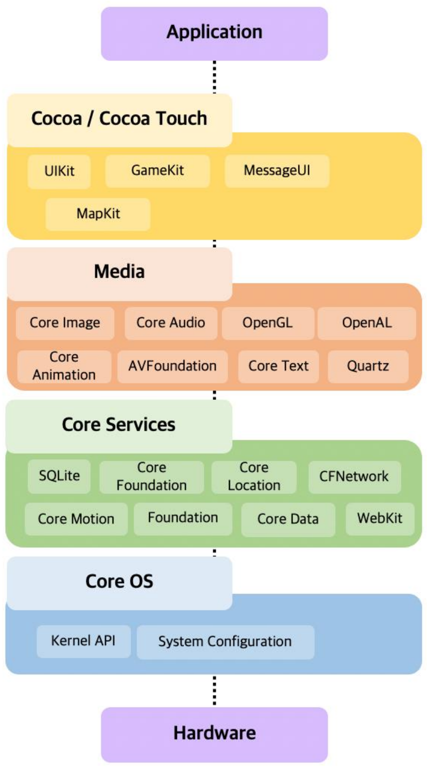
Key Features of iOS:

1. Closed Ecosystem: iOS is tightly controlled by Apple, ensuring high security, stability, and consistent user experience across devices.
2. Unix-based Core: iOS is based on a Unix core, providing a stable, multi-tasking, and secure operating system foundation.
3. App Store Ecosystem: The App Store offers millions of high-quality applications, rigorously vetted for security and performance.
4. Apple Ecosystem Integration: iOS seamlessly integrates with other Apple services and devices, including Macs, Apple Watches, Apple TVs, and HomePods.
5. Focus on Privacy: Apple places a strong emphasis on user privacy, incorporating features like data minimization, transparency, and control over permissions and tracking.

2. iOS Architecture

iOS architecture is designed as a layered system, with each layer providing specific services to enable functionality, performance, and security. This layered approach helps developers interact with the system efficiently and makes the OS modular, manageable, and secure.

Layers of iOS Architecture:



1. **Core OS Layer (Lowest Layer):**
    - This layer is the foundation of the iOS architecture, built on a Unix-based system. It handles essential system services, including memory management, file system handling, networking, security, and power management.
    - The Core OS layer includes low-level frameworks such as Core Bluetooth, Accelerate, External Accessory, and Security, which enable direct access to hardware and low-level functionality.
2. **Core Services Layer:**
    - The Core Services layer offers essential services for iOS applications and manages system-wide services, providing APIs for data storage, cloud computing, location services, and networking.
3. **Media Layer:**
    - This layer contains multimedia services for audio, video, graphics, and animations, enabling a rich media experience in iOS apps.
    - Important frameworks within the Media Layer include:
        - AVFoundation: Manages audio and video playback and recording.
        - Core Graphics and Core Animation: Used for 2D graphics rendering and animations.
        - Metal and SceneKit: Provide low-level, high-performance graphics rendering for 3D applications and games.
        - Core Image: Enables image processing and filtering within applications.
4. **Cocoa Touch Layer (Top Layer):**
    - The Cocoa Touch layer is the highest layer in the iOS architecture, where the majority of app-related development occurs.
    - It contains frameworks used to create the UI and enable user interaction, such as:

        - UIKit: The primary framework for building graphical user interfaces on iOS, offering controls, views, and windows.
        - MapKit: Enables embedding maps within apps, allowing developers to incorporate location data.
        - Push Notification Service: Provides the infrastructure for sending push notifications to users.
        - GameKit: Enables social gaming features such as leaderboards, achievements, and multiplayer gameplay.
    - Cocoa Touch also supports advanced features like multitasking, gesture recognition, and haptics, ensuring smooth interaction between the user and the device.

3. Security in iOS Architecture

iOS has a comprehensive security model integrated throughout its architecture, protecting users and their data while maintaining device functionality. Apple's control over both hardware and software allows for a robust, cohesive security strategy.

Key Security Components:

1. **Secure Boot Chain:**
    - iOS devices use a secure boot chain that verifies every step of the boot process. Each stage checks the integrity and authenticity of the next, ensuring that the system hasn't been tampered with before it loads.
2. **Application Sandbox:**
    - Each iOS app is sandboxed, running in an isolated environment that prevents it from accessing data from other apps or the system without permission. Sandboxing limits potential damage from malicious applications and prevents unauthorized access to system resources.
3. **Data Protection:**
    - iOS offers data protection through hardware encryption, using device-specific keys that protect user data. When the device is

locked, the data becomes inaccessible, safeguarding it from unauthorized access.

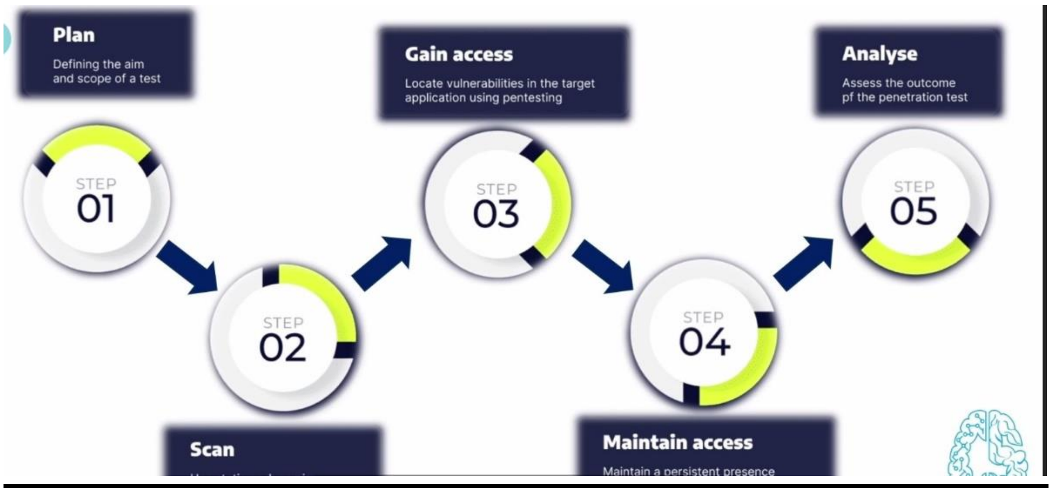
4. **App Store Review Process:**
    - Apple's strict app review process assesses each application for security, performance, and privacy standards before allowing it into the App Store. This limits the distribution of potentially harmful or low-quality applications.
5. **Keychain and Biometric Security:**
    - Keychain is a secure storage system for sensitive information, like passwords and cryptographic keys, that apps and system services can use.
    - Face ID and Touch ID provide biometric authentication, enhancing device security and providing a secure, user-friendly experience.
6. **Permissions Model:**
    - iOS has a strict, granular permissions model, requiring apps to explicitly request user permission to access sensitive data or device components, such as location, camera, or contacts.
7. **Runtime Protections:**
    - To prevent exploitation, iOS includes runtime protections such as Address Space Layout Randomization (ASLR), which makes it difficult for attackers to predict memory locations.

Mobile Penetration Testing: Process and Tools

1. Introduction to Mobile Penetration Testing

Mobile penetration testing is a security assessment process focused on identifying vulnerabilities in mobile applications and the devices they run on. With the increasing reliance on mobile devices for personal and business use, ensuring their security is crucial to prevent unauthorized access, data leaks, and other malicious activities. Mobile pentesting focuses on evaluating security in two main areas: the mobile application itself and the underlying mobile operating system (Android or iOS).

2. Mobile Penetration Testing Process


Plan
Defining the aim and scope of a test
STEP 01
STEP 02
Scan
Gain access
Locate vulnerabilities in the target application using pentesting
STEP 03
STEP 04
Maintain access
Maintain a persistent presence
Analyse
Assess the outcome pf the penetration test
STEP 05

Step 1: Plan

- **Objective:** Define the aim and scope of the penetration test.
- **Description:** This initial phase involves setting clear objectives for the penetration test, such as identifying security vulnerabilities or assessing the overall security posture. Define the scope, including which systems, applications, and networks will be tested, and ensure alignment with stakeholders. Proper planning helps in creating a structured and effective testing strategy while ensuring compliance with any legal or regulatory requirements.

Step 2: Scan

- **Objective:** Use automated tools to scan for vulnerabilities.
- **Description:** In this step, security professionals use tools like Nmap, Nessus, or OpenVAS to perform vulnerability scans on the target system. The purpose of scanning is to detect open ports, services, and potential vulnerabilities that could be exploited. This phase provides a preliminary view of the system's security weaknesses, which will guide further exploration in the next steps.

Step 3: Gain Access

- **Objective:** Locate vulnerabilities in the target application and attempt to exploit them.
- **Description:** This is the exploitation phase, where the tester tries to gain unauthorized access to the system or application by using vulnerabilities

identified in the scanning phase. Common techniques include exploiting software vulnerabilities, weak passwords, or insecure network configurations. The goal is to determine the potential impact of an attack by simulating real-world attack scenarios.

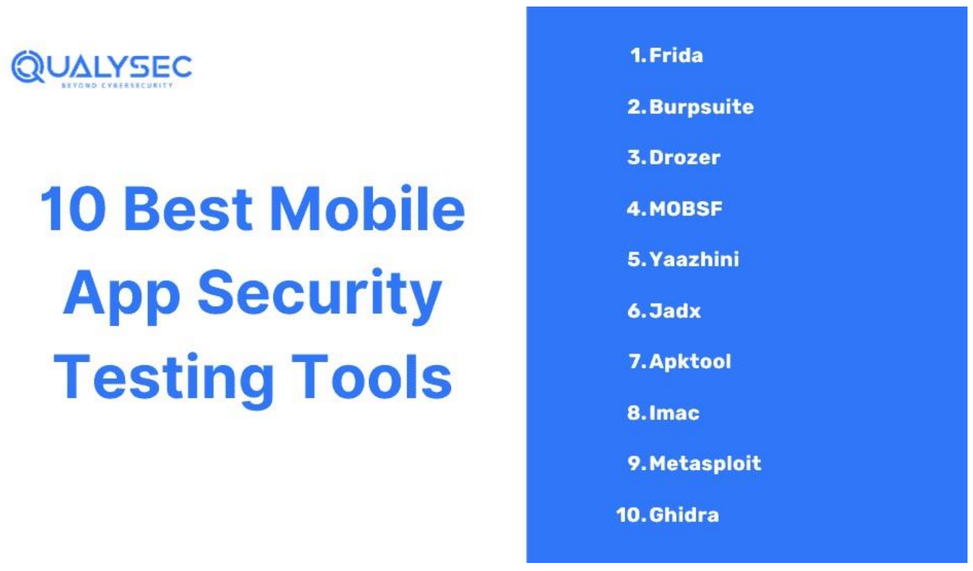
Step 4: Maintain Access

- **Objective:** Maintain a persistent presence within the system to assess potential impacts.
- **Description:** Once access has been gained, the next step is to maintain a foothold in the system. This may involve installing backdoors or using other persistence techniques to simulate an attacker's actions. By maintaining access, testers can assess the extent to which an attacker could compromise the system and how long they could remain undetected.

Step 5: Analyse

- **Objective:** Assess the outcome of the penetration test.
- **Description:** In this final phase, the findings of the penetration test are analyzed and documented. The report should include details of all vulnerabilities discovered, the methods used to exploit them, the potential impact, and recommended mitigation steps. This phase is crucial for providing actionable insights to improve the security posture of the system and prevent future attacks.

3. Mobile Penetration Testing Tools



QUALYSEC
BEYOND CYBERSECURITY

10 Best Mobile App Security Testing Tools

1. Frida
2. Burpsuite
3. Drozer
4. MOBSF
5. Yaazhini
6. Jadx
7. Apktool
8. Imac
9. Metasploit
10. Ghidra

1. Frida

Overview:

There are various tools so let's describe Frida briefly, Frida is a dynamic instrumentation tool kit aimed at developers, researchers, and reverse engineers. It enables you to hook scripts into the running process, which facilitates analyzing and testing the **security of Mobile apps** in real-time. Frida is used extensively for crashing apps on Android and iOS.

Key Features:

- Monitoring of moving applications in real-time.
- Cross-platform compatibility, for devices with Android and iOS operating systems.
- Can interrupt and change the operation of resident applications.
- Enables script injection both in the processes of the user and in the system processes.
- Supports popular programming languages such as JavaScript.

2. Burp Suite

Overview:

Burp Suite is an open-source framework for testing web applications that would often be ranked as top-of-the-line web vulnerability scanners. It is mainly utilized in the context of **penetration testing and security assessment of mobile applications** as well as web applications. In Burp Suite there are free and paid editions, however, depending on the extent of the advanced tools included.

Key Features:

- It should provide a broad and deep ability of web vulnerability scanning.
- One of the uses of this newly developed structure is the automated scanning of mobile apps.
- The HTTP proxy server for interception and modification of HTTP connections' requests and responses.

- It supports the SSL/TLS traffic inspection.
- Fully extensible and it has a large number of plugins.

3. Drozer

Overview:

Drozer is a tool that works as an information-gathering security testing framework that has been developed for Android. It enables security analysts as well as developers by allowing them to make an assessment of the attack vectors of **mobile applications** and do the ordinary test privilege escalation, data leakage, and so on.

Key Features:

- Holds the capability to acknowledge security defects & issues relating to the applications of Android.
- Capability to take advantage of and evaluate security threats.
- Easy-to-use command-line interface.
- Provides a list of parts of apps that are exposed and therefore vulnerable to attack.
- It is useful in mimicking real-life attack scenarios.

4. Mobile Security Framework (MobSF)

Overview:

MobSF is a powerful and automated Security Testing framework to analyze **Android, IOS, and Windows mobile apps**. For static analysis it offers complete elements together with dynamic analysis; therefore security specialists can find a complete solution for their work.

Key Features:

- The support matrices as far as static, dynamic, and malware analysis are concerned have been presented here.
- Compatible with Android, iOS, and Windows operating systems.
- Access to quick results and comprehensive reports regarding the security of the Website.

- It can locate Code, configuration, and permissions weaknesses.
- Complete coupling for **OWASP Mobile Top 10**.

5. Yaazhini

Overview:

Yaazhini is a heavy weapon in the arsenal of **mobile application security testing targeting iOS applications**. This can be used in the identification of risks especially in areas of data, encryption, and authentication among others. Yaazhini is particularly useful for developers and security analysts, particularly in Apple iOS Security.

Key Features:

- More specifically it focuses solely on vulnerabilities within iOS apps.
- Discovers data leakage and poor coding practices.
- It provides for static and dynamic analysis of the code.
- It has a plain and easy-to-use model of operation and a clear structure of menus.
- Works to identify the poor encryption schemes that might have been used.

6. JADX

Overview:

Among those, there is a tool called JDAX which is a decompiler of **Android applications** that helps with the reverse engineering of APK files. It enables the user to have an interface to the source code of the application and assists in detecting security flaws in Android apps.

Key Features:

- The use of Android Asset Packaging File format in their APK, this includes decompiling APK files into Java source code.
- Can be useful to discover some weaknesses that can be used by attackers in Android applications.
- Ease of use and an uncluttered and clean design.

- Good for recreation and reviewing the code.
- Provides a GUI for exploratory purposes to allow users to browse decompilation results.

7. Apktool

Overview:

Numerous tools are used to reverse engineer Android applications, one of the most commonly used tools is Apktool. This means that the users can pull an APK apart and put it back together once they have made their changes. This makes it easier to manage the code and access it for revision, probing for risks, and mastering the layout of the app.

Key Features:

- Bears the capability of decompiling APK files so that the code can easily be read.
- Allows for the APK assembly across which it has been disassembled once modified.
- Used in the identification of security vulnerabilities in Android applications.
- Gives a clear understanding of the app design and a general overview of the app.
- Very suitable for both security analysis as well as for developing Android applications.

8. ImmuniWeb Mobile Suite

Overview:

ImmuniWeb Mobile Suite is a cloud-based solution that provides an extra level of mobile app security testing. Together with the static, dynamic, and interactive methodologies, it is used to assess the level of risks and **compliance** of the apps to the security standards.

Key Features:

- Security testing of mobile applications using Artificial Intelligence.

- Covers both the analysis of structures at a certain point in time and the changes that take place.
- Find out about compliance to standards that apply in industries such as GDPR and PCI DSS compliance.
- The continuous monitoring of security and security alerts in real-time.
- User-friendly cloud-based platform.

9. Metasploit

Overview:

Metasploit is one of the most recognized open-source Pentesting frameworks used for **penetration testing** and to find and take advantage of various system weaknesses; mobile apps inclusive. It has a big archive of modules therefore, it can be a useful tool for **vulnerability assessment**.

Key Features:

- Framework for the complete type of penetration searching.
- Material Support for Many Devices such as Androids and iPhones.
- The large variety of exploits and the payloads.
- The ability to provide frequent updates on newly discovered vulnerabilities.
- Automates software testing, as well as identifying the different vulnerabilities that are present.

10. Ghidra

Overview:

Ghidra is a reverse engineering tool that was created by the National Security Agency. At first, it offers advanced and varied methods for profiling compiled code on numerous operating systems, including mobile apps. This one is more beneficial when it comes to assessing the application and discovering its **security vulnerabilities**.

Key Features:

- It also provides decompilation and reverse engineering features.
- It supports multiple types of platforms such as web, IOS, and more interestingly, android apps.
- Scrolling, colorful interface to deal with code data.
- Saves time as most of the activities of reverse engineering are automated.
- It is less highly customizable with scripting support as compared to SOFA.