

Introduction

Android Application Dynamic Analysis is a method for evaluating the security and performance of an Android application by examining its behavior during runtime. Unlike static analysis, dynamic analysis involves interacting with and monitoring the application while running, either on an emulator or a real device. The primary goal of dynamic analysis is to identify security vulnerabilities, runtime errors, and performance issues that only become apparent during the application's execution. This process is crucial for detecting issues related to data handling, user interface, interaction with other apps and system components, and network communication. As with static analysis, dynamic analysis is also a key part of comprehensive security auditing, complementing static analysis by revealing issues that are only observable in an app's live environment. In the following sections, we will explore various methodologies for performing dynamic analysis on Android applications, covering several different scenarios and topics using a variety of techniques and tools. But first, let's take a look at some key points of the dynamic analysis.

Benefits of Dynamic Analysis

Benefit	Description
Real-Time Vulnerability Detection	Dynamic analysis allows you to identify vulnerabilities that only become apparent during the execution of the application. This includes runtime errors, memory leaks, and other issues that cannot be detected through static analysis.
Understanding Application Behavior	It helps in understanding how the application interacts with various system components, other applications, and external servers. This is crucial for identifying potential security risks in these interactions.
Testing Under Different Conditions	Dynamic analysis enables testing the application under different network conditions, hardware configurations, and user interactions, ensuring that the application behaves as expected in a variety of real-world scenarios.
Security of Sensitive Data	It helps ensure that sensitive data and personal information are handled securely by the application, particularly during data transmission and storage.

How it Works

Dynamic Analysis tools interact with a running instance of an application to identify potential security and performance issues. These tools focus on observing and manipulating the application's behavior during execution, using techniques like instrumentation, network traffic sniffing, and user interaction simulation. For Android apps, dynamic analysis typically involves monitoring the app as it runs on either an emulator or a real device.

The process involves tracking the app's interactions with the Android operating system, other installed applications, and remote servers. Tools used in dynamic analysis can intercept and log network requests and responses, monitor system logs, and track API calls. They may also simulate user interactions or manipulate runtime data to test the app's response to unexpected or malicious input.

Types of Vulnerabilities that Dynamic Analysis Can Detect

Benefit	Description
Runtime Permission Abuse	Improper use of Android runtime permissions, allowing the app to access more data than required or misuse granted permissions.
Improper Handling of Intents	Vulnerabilities related to how the app processes intents, potentially leading to intent interception or injection attacks.
Insecure Data Transmission	Detecting unencrypted or poorly encrypted data being sent over the network during app operation.
Insecure Storage of Sensitive Information	Detecting sensitive information, such as personal data, credentials, and financial information, that are stored insecurely on the device.
Authentication and Session Management Flaws	Issues with how the app manages user sessions and authentication, especially when interacting with remote servers.
Leakage of Sensitive Information	Unintended disclosure of sensitive information through logs, UI components, or other runtime interactions.
Dynamic Code Execution	Identifying issues related to the dynamic execution of code (e.g., loading an external library dynamically at runtime), which can be a vector for various security attacks.

Preparing the Testing Environment

In order to proceed with testing Android apps dynamically, we will need to set up an emulator. We can find all the steps for setting up an emulator by looking back to the [Android Emulators](#) section of the [Android Fundamentals](#) module. In the following example, we will be using an AVD (Android Virtual Device | Pixel 7a Google APIs) so we can have access to the application's internal and external storage.

Next ➔

+10 Streak pts

✔ Mark Complete & Next













 Cheat Sheet

Table of Contents




Enumerating and Exploiting Installed Apps

Introduction
 Enumerating Local Storage
 Exported Activities
 Insecure Logging
 Pending Intents
 Exploiting WebViews
 Insecure Library Load Through Deep Linking

Dynamic Code Instrumentation

 Hooking Java Methods
 Altering Method Values
 Hooking Native Methods
 Bypassing Detection Mechanisms
 Authentication Token Manipulation

Intercepting HTTP/HTTPS Requests

 Intercepting API Calls
 IDOR Attack
 SSL/TLS Certificate Pinning Bypass

Skills Assessments

 Skills Assessment

My Workstation

OFFLINE

▶ Start Instance

∞ / 1 spawns left

