approov

The Threats to Mobile Apps and APIs

The Threats to Mobile Apps and APIs

Contents

Introduction

The two foundational elements driving progress in today's digital first world are the mobile app and the API. Mobile apps are the new channel to your customers and the use of APIs is driving innovation, interoperability and new business models in every major industry.

Unfortunately the combination of these technologies also presents some new security challenges, offering novel opportunities to bad actors to access sensitive data and derail your business:

- Mobile apps are downloaded to unmanaged devices and there are a battery of available tools to allow hackers to dissect and manipulate them at their leisure.
- APIs expose application logic and sensitive data such as Personally Identifiable Information (PII) and because of this have increasingly become a target for attackers. Without secure APIs, rapid innovation will be impossible.

If secrets and application logic can be extracted from the app or elsewhere then bad actors can and will use those to stage sophisticated and highly automated attacks on your APIs.

The rich target presented to bad actors by the interaction of the mobile app and its APIs is the new area where protection must be enhanced and security testing must be focused.

Traditionally a penetration test, also known as a pentest, is a simulated cyber attack against your computer system. In the context of web application security, penetration testing is commonly oriented towards trying to identify and exploit vulnerabilities, such as unsanitized inputs that are susceptible to code injection. Pentesting can involve the attempted breaching of any number of application systems (e.g. Application Programming Interfaces (APIs) and frontend/backend servers).

This paper describes each of the new attack surfaces in your platform which are exposed by the mobile channel and the way attackers can exploit them in orchestrated attacks. We will also stress the key distinction between uncovering and managing vulnerabilities, and the importance of testing any app or API shielding technology which prevents vulnerabilities being exploited.

We will refer to the OWASP documents and resources:

- OWASP is well established in (server-side) Application Security, regularly publishing an OWASP Top Ten which has become the benchmark for Web Application Firewall (WAF) offerings and more recently the OWASP API Top Ten via the OWASP API Security Project.
- Also of relevance is the OWASP Mobile Security project. There are two key documents which lay out an approach to mobile application security and define a tiered standard to assess the security of mobile applications: the OWASP Mobile Security Testing Guide, and the Mobile App Sec Verification Standard (MASVS).

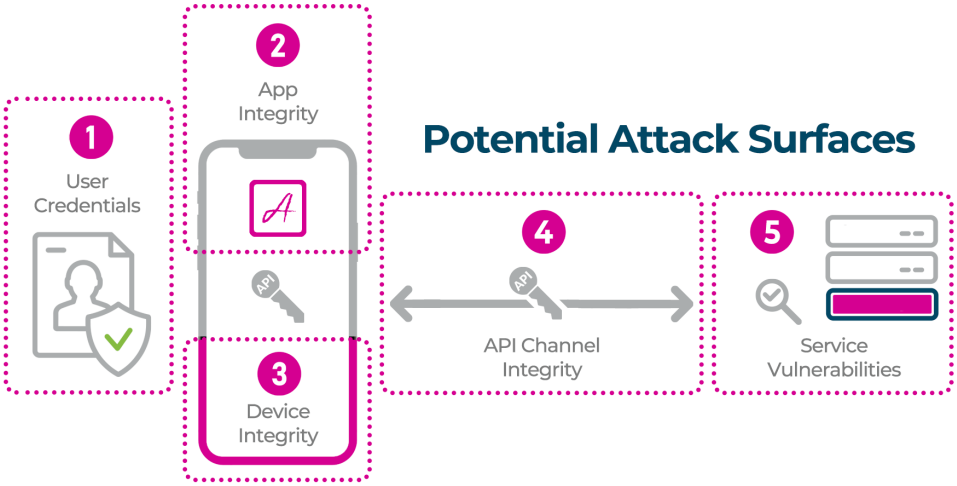
A full description of these and other resources is available in Appendix C of this document.

The Mobile Threat Model

To make it easier to assess and test the security of the mobile channel in its entirety we will introduce a framework which breaks down the overall architecture into five closely linked attack surfaces. This provides a means to direct our testing and to assess the overall security posture of the channel as a whole.

Smooth and secure mobile app operation is built on the assumption that a legitimate user without malicious intentions is accessing your service, using an untampered version of your mobile app, running on an uncompromised device, communicating directly with your API server via a secure channel and that the API cannot be accessed by any other way. The five interlinked attack surfaces that can potentially be exploited are therefore:

1. User Credentials
2. App Integrity
3. Device Integrity
4. API Channel Integrity
5. API and Service Vulnerabilities



We will look at each surface in turn in order to discuss the nature of the potential attack and the testing strategy needed.

Although there is the possibility of an attacker manually using stolen credentials to access a service via a genuine app instance, this scenario does not scale since they must manually use the app, so in general an attacker has the ultimate goal of setting up an automated attack on the API. The automated tool of choice could be a modified app or a script. The API attack can then attempt to extract data by exploiting vulnerabilities such as Broken Object Level Authorization (BOLA) or by using a list of stolen genuine user credentials.

With this end in mind the bad actor will move through the different attack surfaces to identify and find information such as keys and other secrets as well as understand the logic of the app <-> API interaction, and then prepare an automated attack.

The approach of the attacker can therefore be summarized as:

- Explore all attack surfaces for information on how mobile apps access the API and how the API is used to access data and resources in the backend.
- Use this information to set up an automated attack on the API.

Mobile Attack Surfaces and How They Are Exploited by Bad Actors

Attack Surface 1: User Credentials

The authentication and authorization of the user is critical, particularly if personal data is being accessed via the app. Frameworks such as OpenID Connect, and OAuth2 can be used and 2FA and biometrics provide a further level of protection.

However valid credentials can be stolen via spoofing, phishing and have been exposed via large data breaches: username/password combinations are bought and sold on the dark web and used in credential stuffing attacks. Even 2FA and biometrics checks can be poorly implemented, leaving loopholes for bad actors to use compromised credentials.

Credentials don't even need to be stolen to be used maliciously. Increasingly, there are cases where users type in their own credentials to a fake app or betting site so the credentials are willingly given and then exploited by the attacker.

Finally the onboarding process is often exposed. Anyone (including a hacker) can easily just sign up for the service.

It is much easier to use stolen user credentials if other attack surfaces have vulnerabilities. For example, if somebody's credentials are compromised then nothing can stop a bad actor from typing them into a valid instance of the app. But if controls around App Integrity (Attack Surface 2 - see below) are in place the only way this can be done is by "typing in" rather than running an automated credential stuffing attack against the API - in effect 'at scale' credential stuffing can be stopped.

Similarly if the user authorization token has been extracted from the mobile app and is being used from a script, App integrity controls can effectively block them from being exploited.

Attack Surface 2: App Integrity

When a hacker targets the actual mobile app they are seeking to do one or both of the following:

- Extract information which can be used to mount an automated attack on your API using another tool. As well as gaining useful identifiers and keys this could involve inspecting the logic of the mobile app in order to reverse engineer how the API works with the aim of abusing the business logic through the API.
- Transform the app itself into a tool which can be used in an automated attack or tweak it in some other way, for example to divert payment or advertising revenue or to hijack user information for nefarious purposes.

At run-time the API should seek to establish the validity of what is making the request to the API server. Is it really coming from a genuine instance of your mobile app, or is it a bot, an automated script or an attacker manually accessing your API backend with a tool like Postman? Often no such validation mechanism is in place. The hacker tries to understand the structure of the API calls, and investigate if any mechanism is being employed to validate the app in order to replicate it with any required secrets.

The mobile app may use API keys, device ids, or a user authorization token to communicate what is making the request to the API backend, but often these identifiers are hardcoded in the mobile app source code, thus they can be extracted with the use of reverse engineering techniques, for example a static analysis of the binary.

Statically reverse engineering a mobile app is one of the most common steps taken when attacking a mobile app, and it's listed in 9th position of the most recent OWASP Mobile Top 10 risks. To protect against this threat an obfuscation tool must be used - open source and commercial options are available - and obfuscate everything that is possible to obfuscate. Obfuscation per se will not avoid the decompilation process, but will make it time consuming and expensive to follow the code since variables, functions and classes names will be redacted.

When more sophisticated methods such as calculating the API key at runtime are used to hide these identifiers/secrets then dynamic code instrumentation at runtime or a MitM attack can be used to extract them. A bad actor will try to extract whatever is being used to validate the app so that this can be used in an automated attack on the API.

When the mobile app uses certificates from an encrypted store to sign requests to the API backend, these need to be unencrypted at run-time in order to be used. An instrumentation framework can be used to inject code that will extract the certificates which then can be reused outside the mobile app. This same approach of injecting code can be used to extract anything of interest from the mobile app at runtime.

Code tampering comes in the 8th position of the most recent OWASP Mobile Top 10 risks. This is done by repackaging the mobile app with removed or altered code, and if done correctly the API will see the tampered mobile app behave as the original one did. For example, if the API backend is checking the header with the mobile app binary signature hash, then the repackaged app will have code in place to deliver the same value as the original app.

Attack Surface 3: Device Integrity

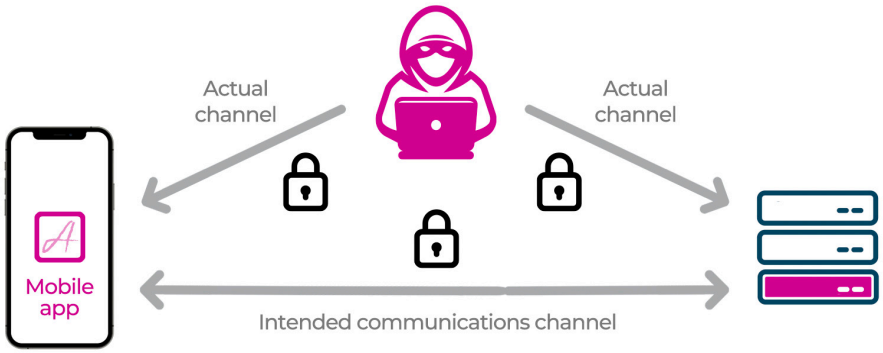
The device on which the mobile app is running may be rooted or jailbroken, and this might be done for legitimate reasons; some users like to run customized or more recent versions of the OS and some users like to sideload genuine apps which may not be available in their local app stores. These actions do not, in and of themselves, indicate malicious activities are going on. That said, rooting/jailbreaking is a common technique used by attackers (and pentesters) to bypass security mechanisms and limitations imposed by the original version of the OS. Rooted and jailbroken devices pose a threat to device integrity, because these actions enable the in-built security mechanisms to be compromised. By extension the threat extends to the mobile app integrity, because the mobile app is now running in an environment that cannot be trusted.

Another form of code tampering is to inject code at runtime by using an instrumentation framework. Such frameworks are used to hook into the key functions which, when manipulated, will produce different app behavior than expected or will change input parameters or output results. In this way fraudsters can intercept and modify genuine user instructions.

To prevent this form of attack the mobile app must employ runtime self defense code to detect rooted/jailbroken mobile devices. Ideally this code will also detect the presence of all known instrumentation frameworks.

Attack Surface 4: API Channel Integrity

The communications channel between the mobile app and API is exposed, often passing via public wifi connections and the internet rather than being contained within a



private network or protected by a VPN.

Setting up communication via HTTPS provides a secure channel, but this may not be sufficient to keep all sensitive information away from attackers. Even when the latest TLS standards are used, an attacker can still perform man-in-the-middle attacks between the API server and a mobile app in order to be able to extract all secrets and understand app queries/responses to and from the API server.

The basic concept of MitM is to convince the client and the server that they are communicating with each other, when in fact a third party is impersonating both ends of the channel. The channel between an API and a mobile app is a target for this kind of attack.

For bad actors MitM is ideal for researching a client-server communication channel in order to establish what attacks might be possible. Examples of potential exploits which could be researched using MitM activities are:

- Comprehension of the API protocols in use, so that scripts can be created to impersonate genuine traffic sequences.
- Extraction of API keys in transit, to be later inserted in scripts in order to convince the server that the communication is coming from a genuine client instance.
- Extraction of user credentials or authentication tokens in transit, to be later inserted in scripts in order to convince the server that the communication is coming from a genuine user.
- Manipulation of transaction requests which are made via the API such that the action requested from the server is different from that initiated by the remote client.
- Researching the existence of API vulnerabilities, e.g. Broken Object Level Authorization (BOLA), which can be later exploited by scripts to access data which should not normally be available to a given user.

The use of certificate pinning does make the MitM attack more difficult to perform, but not impossible, since the mobile app can be repackaged to remove the pinning detection controls or to inject the pins or certificates from the MitM proxy tool. An instrumentation framework such as Frida can also be employed to execute an MitM attack when pinning is deployed.

Check out this for more information about how certificate pinning can be bypassed.

Attack Surface 5: API and Service Vulnerabilities

There are three common scenarios for when a hacker targets an API with an automated tool:

- Login system attacks: Bad actors use credential stuffing and other brute-force mechanisms to test stolen valid credentials from the dark web and determine the credentials' validity on the API. They can then utilize any 'working' credentials to access API services. Bots may execute aggressive attacks or be programmed to run slow attacks designed to stay under the radar of any rate limiting defenses.
- Theft of data: Hackers use APIs to steal files, photos, credit card information, and personal data from accounts available through an API. The approach can range from simple data scraping to more sophisticated attacks which exploit BOLA and other vulnerabilities to infiltrate and manipulate PII data. Again, these activities can be staged as extended-duration data exfiltration attacks to avoid triggering a blocking response from the API gateway.
- DoS: These attacks are intended to impact the availability of the endpoints using a Denial of Service (DoS) attack. A DoS attack renders the API endpoint unusable for legitimate requests by overloading the API endpoint with synthetic API requests in order to knock it offline. While some DoS and Distributed Denial of Service (DDoS) attacks are volumetric in nature (overwhelming the API endpoints with more requests than they can handle), many of the DoS

attacks are layer 7 DDoS attacks, which exploit bugs in the API endpoint that can also render it unable to respond to new requests.

An API that is open to being consumed by public clients needs to identify not just **who** but also **what** is making the request. If a hacker has extracted enough information from the above Attack Surfaces about the validation process and has acquired the necessary keys, then the API can be abused by an automated attack, and any vulnerabilities in the API can be exploited.

The Open Web Application Security Project (OWASP) publishes a Security Top 10 list for API vulnerabilities that have caused recent data breaches and posed common security hazards. The list includes everything from broken authorization and authentication to excessive data exposure, lack of rate limiting, logging and monitoring, and improper asset management and security configurations. Of course there are unknown or "zero day" vulnerabilities - something that a hacker has discovered that is not on a list of "known" vulnerabilities.

Beyond this, there is the issue of API Abuse. These attacks do not depend on API vulnerabilities or implementation issues, relying instead on legitimate access to business logic and data in unexpected ways to undermine security.

See here for more information about how to protect from vulnerabilities and weaknesses in APIs.

Conclusion

An attacker can orchestrate an attack across the above Attack Surfaces including one or more of the following activities.

Attack Preparation:

- Acquire user credentials (Attack Surface: User Credentials) from the dark web or from phishing/spoofing attacks.
- Harvest secrets and business logic from app code (Attack Surface:App Integrity).
- Tamper with the client environment in order to understand the operation of the service and acquire secrets (Attack Surface: Device Integrity).
- Investigate app <-> API dialog via a MitM attack (Attack Surface: Channel Integrity) to intercept secrets and understand the logic of the app and API.

Attack Execution:

- Use acquired information to construct valid queries and set up automated tools to target the API (Attack Surface: Service and API Integrity) in order to:
  - harvest data using commonly known or new vulnerabilities
  - abuse the API business logic
  - interfere with the operation of the service
- Use harvested information to tamper with the app and deploy a modified version in order to divert financial transactions, advertising revenue or simply to steal data (Attack Surfaces: App and Device Integrity).

The mobile security project of OWASP covers the required approach to mobile app security including discussion of the key steps of threat modeling, building a secure SDLC, how security is integrated into DevOps, the role of pentesting and more. See Appendix C and https://mobile-security.gitbook.io/mobile-security-testing-guide/overview/0x04b-mobile-app-security-testing

It is useful here however to remember that the security best practice is "Shift left, but shield right". Shift left of course means using processes, tools and development discipline to address security early in the development process in order to identify, manage and eliminate security issues before deployment. Shield right means to put controls in place to protect the running service. Understanding and making explicit the interplay between these two approaches ensures optimal security and should be reflected in the approach to testing.

The Appendices of this document provide more information about how to protect the attack surfaces described in this document from being exploited and how to evaluate and test the effectiveness of the defenses which are deployed.

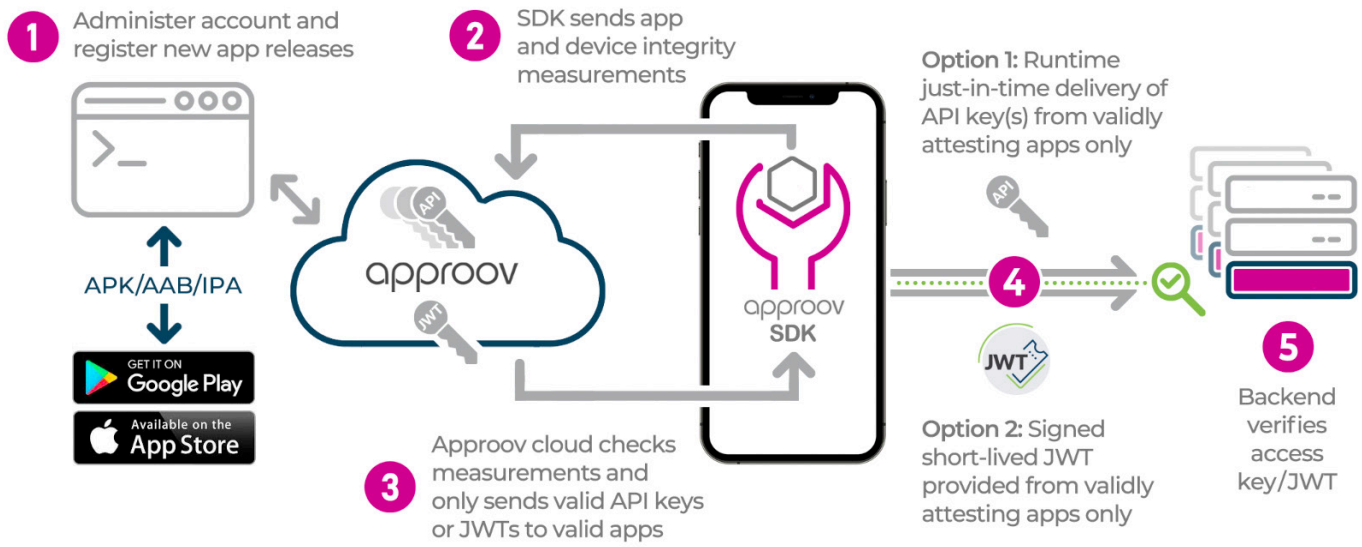
Approov provides a run-time shielding solution which is easy to deploy and protects your mobile apps, APIs and the channel between them from any automated attack. It effectively blocks the execution of attacks, irrespective of the vulnerabilities which are already known or uncovered through testing. See Appendix A for more information on how Approov does this.

See Appendix B for more guidance on pentesting mobile apps and APIs, particularly when using Approov as a shielding technology.

Appendix C provides a summary of some of the tools you can use to test the security of mobile applications and APIs.

Finally Appendix D provides a summary of useful resources and guidelines to help evaluate threats and plan and execute a solution.

Appendix A - How Approov Works



Approov provides a run-time shielding solution which is easy to deploy and protects your APIs and the channel between your apps and APIs from any automated attack. It uses a cryptographically signed "Approov token" to allow the app to provide proof that it has passed the runtime shielding process.

Integration involves including an SDK in your mobile app via a mobile app quickstart and adding an Approov token check in your backend API implementation. A full set of frontend and backend Quickstarts are available to facilitate integration with common native and cross-platform development environments.

Approov also provides Runtime Secrets Protection whereby API keys can be securely held in the Approov cloud and only transmitted to app instances which are valid and untampered. This approach allows Approov to be used for 3rd party APIs where the backend API cannot be modified

By ensuring only an untampered genuine mobile app running in an uncompromised environment can access the API, Approov prevents the exploitation at scale of:

- Stolen user identity credentials.
- Vulnerabilities in your apps or APIs, irrespective of whether the vulnerabilities are already known, uncovered through testing or "zero-day".
- Malicious business logic manipulation of the API.
- Man in the middle attacks.

The following sections refer to the diagram above to show how the Approov flow works in detail.

1. Mobile App Registration

The Approov CLI (Command Line Interface) tool is downloaded to your development environment. It is used to access and administer the Approov account provided upon sign up. The tool is also used to register new apps that are to be released to the app stores. This is achieved by analyzing the app (in either .apk, .aab or .ipa format) and creating a unique signature which captures all aspects of the application and is virtually impossible to access or replicate. This unique "DNA" of the app is added to a database in the Approov cloud service for your account. No application code is stored or uploaded to the Approov service. The particular build of the app then becomes recognized as being official.

2. App Makes API Calls

The quickstart integrations either hook into the app's network stack, or provide a networking stack implementation with the same interface your app is already using. Either way, only minimal code changes are required to integrate Approov

into the app. When an API call is made that is to be protected with Approov, and no cached Approov token is available, the integrity assessment process is initiated automatically. This causes the Approov SDK to communicate with the Approov cloud service.

3. Integrity Assessment

The integrity check process requires the SDK and the Approov cloud service to work together. The SDK analyzes the runtime environment of the app and the authenticity of the app that is being measured. These checks are implemented in hardened code and communications are protected by TLS, certificate pinning and also by a secondary level of request integrity signing. The app gathers and passes data and measurements to the Approov service. The Approov cloud service performs analysis on the data provided by the SDK and makes a decision based on this and the security policy criteria you set for your account. These policies are dynamic and can be updated in the cloud service at any time.

If the criteria are met then the Approov cloud service provides the short lived cryptographically signed Approov token. If the criteria are not met then a token is still issued, but it is not signed with the correct secret. The mobile app itself is never able to distinguish between a valid and an invalid Approov token.

If Runtime Secrets Protection is used then the API keys are securely transmitted from the Approov cloud to the app at this point, where they can be added to the request.

4. Approov Token Delivered to App and included in API Request

The obtained Approov token is added automatically as a header to the outgoing API request, which then continues. The same Approov token may be used for up to 5 minutes (as long as no change in the environment is detected) to avoid further communication with the Approov cloud.

It is important that all communications made by these APIs are pinned so that no Man-in-the-Middle (MitM) interception is possible that could make a copy of API request data, including the Approov token. Pinning TLS connections are managed automatically by the Approov dynamic pinning functionality.

5. Approov Token Check on the API Backend

If Approov tokens are used then the backend API must check the validity of the Approov token by checking if it has been correctly signed and has not expired. If valid, then you know that the API request is really coming from an official registered version of your app and that it is not being spoofed by some other entity. Moreover, a valid Approov token also indicates that the checks on the runtime environment have passed, as defined by the security policy you have set in your account. Since the signing key is never put inside the app, an attacker cannot reverse engineer it in order to create their own signed Approov tokens.

Any request that fails the Approov token check is coming from:

- A script, bot, or a manual request and not from a genuine and unmodified instance of the mobile app.
- An app which is running in a compromised environment as defined by policy e.g. from a mobile app that is under attack via MitM, code injection, etc.
- A repackaged or tampered mobile app.

Typically you would configure your backend to block such requests.

Appendix B - Pentesting Mobile Apps and APIs

Effective penetration testing requires a diligent effort to find enterprise weaknesses, just as a malicious individual would.

This Appendix contains two sections. First there is a discussion of how to pentest when the Approov solution is protecting your APIs and second some specific guidance for each phase of a typical pentest.

Testing Apps and APIs when Approov is Deployed

If Approov is deployed as a defense then this needs to be taken into account in order to enable pentesting to proceed smoothly. Approov effectively blocks access to the API from tampered apps, compromised environments, instrumentation frameworks and scripting tools, all of which are tools and tactics often employed by pentesters as well as hackers. This section provides guidance to the pentester on how to facilitate testing of vulnerabilities which would otherwise be impossible to exploit with Approov in place.

Accessing API Backend Endpoints Protected by Approov

During the pentest activities the pentester may need to pentest directly the API endpoints which are protected by Approov and the following approaches are possible:

- Force Device to Always Pass: To pentest the API behind Approov, the device ID of the originating mobile device can be provided to always pass the device with the Approov cloud service so that it always delivers valid Approov tokens. The device ID can be extracted by using one of the methods described in the Approov docs. Once the device ID has been extracted, it can be forced to pass using these instructions.
- Disabling Approov Certificate Pinning: Some pentest activities may be easier to perform if pentesters can have pinning disabled, and Approov makes this easier by leveraging the Approov dynamic config and its over the air update capabilities. To unpin a specific device follow these instructions in the Approov docs.
- Approov Example Tokens: These allow Approov tokens to be generated with a 1-hour validity period for the purposes of testing the backend API. The customer can issue, via the Approov CLI, an example token to be used by the pentesters during their security assessment using tools such as Postman.

Approov Visibility for Pentesting

In order to provide visibility to the pentesters during their security assessment the customer has several options which can be used all together or individually.

- Approov Token Info: More visibility for each Approov token can be obtained by querying a special Approov endpoint. This requires a cURL command example obtained from the Approov CLI by following these instructions in the Approov docs. This also requires that the Approov Attestation Response Code be enabled on the Approov token.
- Device Specific Information: When a device is forced to pass, it's possible to retrieve more information about the device attestations with the Approov cloud service. Follow these instructions to learn how to use this feature.

Pentesting Guidance by Phase

The table in this section provides guidance by phase based on the PTES Guidelines.

Testing Stage	Purpose	Specific Guidance
Pre-Engagement Planning	During this pre-phase, the scope of the testing is defined in collaboration with the penetration testing company in order to outline the logistics of the test, expectations, legal implications, objectives and goals the customer would like to achieve. Planning will include whether this is a black box, white-box or gray-box pen-test and goals should be aligned to specific pentesting outcomes.	• Use the company Security Policy to guide the scoping exercise which should explicitly consider how deeply app and API vulnerabilities will be explored versus testing deployed shielding technology. Also consider whether API abuse is in scope: how automation can be used to subvert a company's business model, even in the absence of specific API vulnerabilities. • Specify clearly the environment to be tested - use our threat guide to the attack surfaces in the application channel and clearly identify the app (versions) and APIs to be tested. • Outline the objectives and activities for each phase and build a project plan. • Write the SOW(s) for any external contractors. • Have a communications plan - who needs to know pen-testing is taking place eg the operations team.
Intelligence Gathering	The organization being tested will provide the penetration tester with general information about in-scope targets, and the tester will gather additional details from publicly accessible sources.	• Determine if certificate pinning is being employed. • Determine if any public APIs are being accessed. • Determine which countermeasures are in place and research them to evaluate if circumvention techniques exist and can be employed. • Determine the authentication method being used by the API for users (e.g. Auth0) or apps (e.g. API keys).
Threat Modelling	Threat modeling is a process for prioritizing where remediation strategies should be applied to keep a system secure.	• Threat modelling evaluates the level of risk based on the value of assets exposed (e.g. PII) and the motivation and capabilities of bad actors. • The primary threat to consider is the exploitation of the API at-scale in order to access sensitive data assets. • The tester should assume that if the assets are valuable enough, authentication information can and will be obtained by bad actors in order to attack APIs directly and countermeasures should be in place to protect from these kinds of attacks.

Testing Stage	Purpose	Specific Guidance
Vulnerability Analysis and Assessment	Penetration testers are expected to identify, validate, and evaluate the security risks posed by vulnerabilities. This analysis of vulnerabilities aims to find flaws in an organization's systems that could be abused by a malicious individual. Vulnerability scans tend to use automated tools, with some manual support, to identify known weaknesses in a target enterprise.	• The extent and focus of vulnerability evaluation depends on the scope of the testing: in some cases the aim will be to validate mitigation is in place and working and the vulnerability is not accessible; while in other instances the goal may be to discover all applicable vulnerabilities. • This is particularly relevant in the case of mobile applications where there is effectively an arms-race between the deployment of increasingly sophisticated techniques to protect code and secrets on the one hand and the increasingly sophisticated tools and techniques available to bad actors to breach these controls. For the tester a tradeoff should be considered between the cost and effort of static, dynamic and manual efforts to uncover vulnerabilities and the assumption that secrets permitting access to APIs will be uncovered. • If finding vulnerabilities in the app is in scope, apps can be reverse engineered using Mobile Security Framework (MobSF), an open source security framework designed to automate the static and dynamic code analysis of mobile applications, supporting APK and IPA file formats as well as zipped source code. MobSF is a layered framework of different tools, one of which is apktool. Apktool handles decompiling and decoding of the compiled sources in an APK file. • A similar tradeoff is true for API and/or application vulnerabilities. For example extensively testing every vulnerability on the OWASP Top 10 should be balanced against evaluating the effectiveness of countermeasures to block exploitation of the most critical threats and vulnerabilities in the back-end.

Testing Stage	Purpose	Specific Guidance
Exploitation	The exploitation phase of a penetration test focuses solely on establishing access to a system or resource by bypassing security restrictions based on the high value target list established by the vulnerability assessment. The main focus is to identify the main entry points into the organization and to identify high value target assets.	• This stage should primarily be aimed at attempting to exploit the most critical vulnerabilities in the APIs. The three most common tactics and techniques used by adversaries to breach APIs affect authentication, authorization, and availability. • Authentication: For authentication attacks adversaries combine dumped credentials from previous breaches in an account takeover (ATO) style attack referred to as credential stuffing where usernames and passwords are sent to the API until a successful authentication is established.This is also called brute forcing. • Authorization: Authenticating with an API with either a legitimate account or with an API key or token, doesn't mean that the individual is authorized to read or write the data. An example of a devastating authorization vulnerability is number one on the OWASP API Top10 list, Broken Object Level Authorization (BOLA).This vulnerability is also referred to as Insecure Direct Object Reference (IDOR). Using genuine credentials the hacker can attempt to exploit this and other vulnerabilities. • Availability: The third type of attack affects availability of the endpoints using a Denial of Service (DoS) attack. A DoS attack renders the API endpoint unusable for legitimate requests by overloading the API endpoint with synthetic API requests in order to knock it offline. While some DoS and Distributed Denial of Service (DDoS) attacks are volumetric in nature (overwhelming the API endpoints with more requests than they can handle), many of the DoS attacks simply exploit bugs in the API endpoint that can also render it unable to respond to new requests. • For penetration testing of the APIs, different applications can be employed eg. custom API requests can be created using Postman, and Burp Suite Pro can be used for intercepting the mobile app traffic, modifying it and replaying it to the APIs.

Testing Stage	Purpose	Specific Guidance
Final Analysis and Review	After the exploitation phase is complete, the goal is to document the methods used to gain access to your organization's valuable information. The penetration tester should be able to determine the value of the compromised systems and any value associated with the sensitive data captured. Once the penetration testing recommendations are complete, the tester should clean up the environment, reconfigure any access he/she obtained to penetrate the environment, and prevent future unauthorized access into the system through whatever means necessary.	• Identify strengths and especially weaknesses in the security. Give a sense of both the likelihood and severity of each vulnerability being exploited. • Document each repeatable test scenario which exposes a vulnerability. • Make recommendations on how to resolve and/or minimize the impact of each vulnerability. • When cleaning up the test environment, for Approov specifically, ensure that all device-specific security policies and pinning overrides have been reset.
Use of Test Results	Reporting is often regarded as the most critical aspect of a pentest. It's where you will obtain written recommendations from the penetration testing company and have an opportunity to review the findings from the report with the ethical hacker(s). The findings and detailed explanations from the report will offer you insights and opportunities to significantly improve your security posture. The report should show you exactly how entry points were discovered in the Threat Modeling and Vulnerability Assessment phases as well as how you can remediate the security issues found during the Exploitation phase.	• Summarize the final review results to highlight the overall business exposure with some measure of the cost to correct. • Integrate the summary with the more detailed final review material to form a complete report.

Appendix C - Tools for Pentesting

The OWASP Mobile Security Testing Guide covers a lot of different tools that can be used to pentest mobile apps. This list summarizes the most useful.

All in One

Name	License	Notes
MobSF - Mobile Security Framework	Open-source	Mobile Security Framework (MobSF) is an automated, all-in-one framework for mobile application (Android/iOS/Windows) pentesting, malware analysis and security assessment and is capable of performing static and dynamic analysis.

Static Analysis

Name	License	Notes
MobSF - Mobile Security Framework	Open-source	Android, iOS. Docker image.
ApkTool	Open-source	A tool for reverse engineering 3rd party, closed, binary Android apps. It can decode resources to nearly original form, make modifications and rebuild.
Ghidra	Open-source	This framework includes a suite of full-featured, high-end software analysis tools that enable users to analyze compiled code on a variety of platforms including Windows, macOS, and Linux. Capabilities include disassembly, assembly, decompilation, graphing, and scripting, along with hundreds of other features. Ghidra supports a wide variety of processor instruction sets and executable formats and can be run in both user-interactive and automated modes.
Radare2	Open-source	The Radare project started as a forensics tool: a scriptable command-line hexadecimal editor able to open disk files, but later added support for analyzing binaries, disassembling code, debugging programs, and attaching to remote gdb servers.
IDA	Free/pro	iOS binary analysis.

Repackaging Apps

Name	License	Notes
apk-mitm	Open-source	A CLI application that automatically prepares Android APK files for HTTPS inspection.
ApkTool	Open-source	A tool for reverse engineering 3rd party, closed, binary Android apps. It can decode resources to nearly original form and rebuild them after making some modifications.
OpTool	Open-source	Interfaces with MachO binaries in order to insert/remove load commands, strip code signatures, resign, and remove aslr.
Fridapa	Open-source	An automated wrapper script for unpacking, patching (insert the load command into binary), re-signing and deploying apps on non-jailbroken device.
Radare2	Open-source	Read about iOS code signing here.

Instrumentation Frameworks

Name	License	Notes
Frida	Open-source	Android, iOS.
Objection	Open-source	Runtime mobile exploration toolkit, powered by Frida, built to help you assess the security posture of your mobile applications, without needing a jailbreak.
Magisk	Open-source	Android rooting tool, with direct support for Xposed modules.
xPosed	Open-source	Android instrumentation framework.
Cycript on Frida	Open-source	iOS

MitM Attacks

Name	License	Notes
mitmproxy	Open-source	Available as a docker image.
Appmon	Open-source	AppMon is an automated framework for monitoring and tampering system API calls of native macOS, iOS and android apps. It is based on Frida.

Burp Suite	Free/pro	Proxy your HTTPS traffic, edit and repeat requests, decode data, and more.
Fiddler	Free/pro	Capture all HTTP(S) traffic between your computer and the Internet with Fiddler HTTP(S) proxy. Inspect traffic, set breakpoints, and fiddle with requests & responses.

API Testing

Name	License	Notes
APICheck	Open-source	Collection of tools for API testing. See docs.
zaproxy	Open-source	Automate API pentesting. See here and here.
insomnia	Open-source	Create requests to try against your API.
Postman	Free	Create requests to try against your API.

Appendix D: Mobile Security Resources and References

OWASP provides relevant guidance to developers and testers via several key projects.

The OWASP (Open Worldwide Application Security Project) MASVS (Mobile Application Security Verification Standard) is a valuable resource for mobile app developers seeking to improve the security posture of their iOS and Android applications. The standard is based on the collective knowledge of security experts from around the world and provides both a baseline and a benchmark for security requirements for mobile apps.

There are 3 key documents which can be downloaded from the OWASP site:

- The verification standard (MASVS) which describes at a high level of abstraction the controls or attack surfaces which should be protected in any mobile app.
- The testing guide (MASTG), which gives much more practical technical guidance on how to test and attempt to reverse engineer mobile apps in order to verify the controls in the MASVS. MASVS is agnostic in terms of OS while MASTG delves deep into IOS and Android examples.
- The checklist which provides the links between the 2 previous documents - for every control in MASVS, there are pointers to the relevant tests and checks in MASTG.

The team makes it clear that MASVS documents can be updated at any time, and there was a major refactoring of the recommendations culminating in V2.0.0 of MASVS which was released in the Spring of 2023.

The other relevant OWASP projects and their associated resources deal primarily with Attack Surface 5 (Service and API Integrity) but without any specific content on the challenges of Mobile App API access in particular.

- OWASP API Security - https://owasp.org/www-project-api-security/ focuses on strategies and solutions to understand and mitigate the unique vulnerabilities and security risks of Application Programming Interfaces (APIs) publishes and maintains a list of the top ten API vulnerabilities.
- OWASP Top 10 - https://owasp.org/www-project-top-ten/ The OWASP Top 10 is a standard awareness document for developers and web application security. It represents a broad consensus about the most critical security risks to web applications, most of which can be exploited via APIs.
- OWASP Mobile Top 10 - https://owasp.org/www-project-mobile-top-10/ Similar to the OWASP Top 10 but aimed at mobile app developers, this represents a broad consensus about the most critical security risks to mobile apps and their APIs.

Approov protects your mobile apps from being modified and from any manipulation of the client environment, keeps your API secrets secure, and protects the communications channel to the server. This whitepaper lays out how Approov implements MASVS.