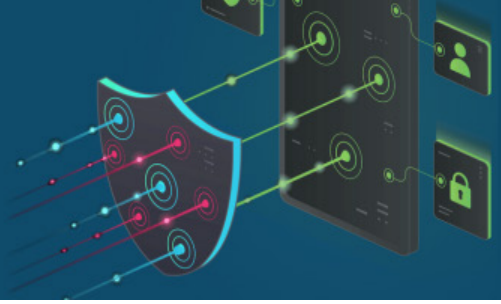
approov

How to Use the 2024 OWASP Mobile Top 10 & OWASP MASVS to Secure Your Mobile Apps

approov

WHITE PAPER



How to Use the 2024 OWASP Mobile Top Ten & OWASP MASVS to Secure Your Mobile Apps

Contents

The OWASP Mobile Top 10 List was updated in early 2024, for the first time since 2016. This list represents the most crucial mobile application security risks in 2024. It complements the more detailed guidelines for mobile app security which are provided by another OWASP project : MASVS (Mobile Application Security Verification Standard) which was updated in early 2023.

This whitepaper gives an explanation and summary of each of the vulnerabilities on the updated Mobile Top 10, as well as discussing the guidelines provided by MASVS and how the two projects are linked together. It is intended to provide guidance to developers on how to use these OWASP resources effectively to improve the security posture of their mobile apps. Another whitepaper by Approov covers OWASP MASVS 2.0 in more detail - https://approov.io/info/how-approov-aligns-with-owasp-masvs-v2.0

What is OWASP?

The Open Worldwide Application Security Project (OWASP) is an online community founded in 2001 that has become highly influential in the realm of web, API and mobile application security. Composed of cybersecurity professionals, researchers, and enthusiasts, the community initially crafted the OWASP Top 10, a list of the most critical *web* app security risks. Other lists such as the OWASP API Security Top 10 are also published and updated by OWASP and there are a large number of active projects covering a range of related topics.

OWASP Projects Focused on Mobile

OWASP recognizes the unique security challenges of mobile apps, there are 2 key projects focused on mobile security:

- The OWASP Mobile Top 10: This provides a description and mitigation information for the 10 worst mobile application security vulnerabilities. The newly issued 2024 version updates the previous list, published in 2016.
- OWASP MAS: The second important OWASP project in mobile security is the MASVS (**Mobile Application Security Verification Standard**) which is intended to set the standard for mobile app security. MAS provides a valuable set of tools for mobile app developers. There are three key documents:
  - 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 granular and 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.
  - A checklist which provides the links between the 2 previous documents - for every control in MASVS, there are pointers in this checklist to the relevant tests and checks in MASTG.

Summary of OWASP MASVS Recommendations

The reworked MASVS application security model is divided into several control groups representing the most critical attack surfaces around a mobile application. It is important to note that it's not just "about the app". The network and the client environment both present significant attack surfaces and MASVS addresses

those.

Here is a summary of the MASVS 2.0.0 security control groups with a short commentary on what's new:

- **MASVS-STORAGE**: Secure storage of sensitive data on a device (data-at-rest). This was vastly simplified down to only 2 controls, providing a good illustration of how everything was cleaned up in V 2.0.0.
- **MASVS-CRYPTO**: This concerns the cryptographic functionality used to protect sensitive data. In V2.0.0. This was also simplified and aligned with NIST standards NIST.SP.800-175B and NIST.SP.800-57p1. There are two controls, the second (MASVS-CRYPTO-2) ensures that the app performs key management according to industry best practices.
- **MASVS-AUTH**: This concerns authentication and authorization mechanisms used by the mobile app. The latest release of MASVS-AUTH cleanly separates client-side and server-side authentication, using the OWASP Application Security Verification Standard (ASVS).
- **MASVS-NETWORK**: This covers secure network communication between the mobile app and remote endpoints (data-in-transit). The latest releases of the documents address head-on some of the confusing guidance (e.g. from Apple and Google!) around certificate pinning and makes it very clear that pinning is absolutely required in order to provide the highest levels of channel security.
- **MASVS-PLATFORM**: Secure interaction with the underlying mobile platform and other installed apps.
- **MASVS-CODE**: Security best practices for data processing and keeping the app up-to-date.
- **MASVS-RESILIENCE**: This covers resilience to reverse engineering and tampering attempts. Obfuscation or code hardening are still advised as a protection against static analysis (in MASVS-RESILIENCE-3) but in V2.0.0 there is a realization that obfuscation is necessary but not sufficient, and that there is an evolving need to protect apps from increasingly sophisticated attacks. This brings a new emphasis in MASVS and MASTG on protecting against dynamic analysis and run time tampering of the app and the client environment it runs on.

Each control group contains individual units labeled MASVS-XXXXX-Y, which provide specific guidance on the security measures that must be implemented to meet the standard.

How OWASP MASVS and The Mobile Top 10 are Connected

At first glance it can be confusing. There is no explicit connection between OWASP MAS and the OWASP Mobile Top 10 and the versions of these documents are not coordinated.

Here is how to think about it:

The OWASP MAS project consists of a series of documents that establish a security standard for mobile apps and a comprehensive testing guide that covers the processes, techniques, and tools used during a mobile application security assessment, as well as an exhaustive set of test cases that enables testers to deliver consistent and complete results.

The OWASP Top 10 on the other hand is really the reference standard for the most critical current mobile application security risks. Updates to the Top 10 Mobile list have proven to be quite infrequent since the process of creating such Top 10 lists is painstaking, time consuming and very thorough: The team starts by collecting data from incident reports, vulnerability databases, and security assessments, then the data is analyzed and sources checked for reliability and consistency. Vulnerabilities are then prioritized based on their impact and likelihood of occuring and finally the list is validated with experts prior to publication.

So, eliminating the OWASP Mobile Top 10 is perhaps the most effective first step towards changing your software development culture to one which is focused on producing secure code.

This table summarizes the mapping from the OWASP Top 10 to OWASP MASVS:

OWASP Mobile Top 10 Risk	Corresponding MASVS Category
M1: Improper Credential Usage	MASVS-AUTH (Authentication)
M2: Inadequate Supply Chain Security	MASVS-CODE (Code)
M3: Insecure Authentication/Authorization	MASVS-AUTH (Authentication & Authorization)
M4: Insufficient Input/Output Validation	MASVS-CODE (Code)
M5: Insecure Communication	MASVS-NETWORK (Network)
M6: Inadequate Privacy Controls	MASVS-PRIVACY (Privacy)
M7: Insufficient Binary Protections	MASVS-RESILIENCE (Resilience)
M8: Security Misconfiguration	MASVS-CODE (Code) & MASVS-PLATFORM (Platform)
M9: Insecure Data Storage	MASVS-STORAGE (Storage)
M10: Insufficient Cryptography	MASVS-CRYPTO (Crypto)

There is a need for a short summary of the OWASP Top 10 for Mobile, so rather than wading through the OWASP documentation here is all you need to know to get started.

Full List of the 2024 OWASP Mobile Top 10

M1. Improper Credential Usage

What is It?

Number 1 on the OWASP list concerns security of credentials, API keys and other secrets. Hackers can exploit hardcoded credentials and improper run time secret usage in mobile applications to gain unauthorized access to mobile app functionality. Also, stolen keys can also be used to access back end systems and APIs, by using publicly available or custom-built tools and scripts.

How to Check

Security testers should attempt to identify hard coded credentials within the mobile app's source code or within any configuration files. Tools such as MobSF can find secrets in mobile APK and .ipa files. Lack of certificate pinning on the communications channel to the backend is a sign that secrets can be intercepted.

How to Mitigate

In summary, never store secrets in your app code and take steps to protect secrets at rest and in transit in the running app. In addition, ensure that you are able to routinely and quickly rotate any keys, especially third-party API keys. Secrets must always be encrypted and communicated securely via HTTPS and certificate pinning must be used. Dynamic secrets management is essential so that you can rotate secrets, pins and certificates easily and often without having to republish your app.

M2. Inadequate Supply Chain Security

What is It?

A 'supply-chain attack' is an attack on the external dependencies or tooling you use in order to introduce vulnerabilities, insecurities or malicious code into the app without being detected. It could be done by a rogue employee inside your organization, or in a piece of third-party code you employ, or even by a hacker who has gained privileged access to one of your systems.

How to Check

This is difficult to test easily. Exploiting vulnerabilities related to business logic flaws often requires understanding the app's functionality and crafting specific test cases. Automating such tests can be complex and might not cover all scenarios but tools are available (e.g. Github Dependabot) which can scan the dependencies you have and notify if any of them have announced CVEs.

How to Mitigate

Obviously, you should Implement secure coding practices, code review, and testing throughout the mobile app development lifecycle. Also, be very careful about the use of third party code: Use only trusted and validated libraries which you know to have good coding practices and proper code review for contributions. Keep clear of low usage software and regularly scan 3rd party code for exposed secrets.

It is important to put in place a process and tools to track and manage CVEs raised across your software supply chain in order to manage dependencies as issues arise and fixes are available.

Also, it is critical to sign your app code prior to distribution so that the app can be attested as genuine and unmodified at run time, and requests from modified apps can be blocked.

M3. Insecure Authentication/Authorization

What is It?

Authentication and authorization are bundled together for number 3 on the OWASP list. For clarity, authentication is the act of verifying the user is who they say they are while authorization is the act of verifying the user has credentials to access a particular resource or level of service. Authentication concerns the theft and use of user credentials and then using these to either log on to the app or by directly accessing backend servers. Authorization issues are usually due to poorly implemented access levels to data and resources.

What to Check

It is particularly important to check offline authentication, so testers should undertake binary attacks against the mobile app while it is offline, aiming to bypass offline authentication and execute functionality that should require authentication. Testers should also try to execute any backend server functionality anonymously by removing any session tokens from any POST/GET requests. Offline authentication needs to be checked to make sure app features are only enabled for app users who have the required permissions (particularly appropriate for apps with in-app purchased content).

How to Mitigate

It is recommended that checks for authentication and authorization should always be reinforced server-side to avoid vulnerabilities in client side offline authentication.

Again app attestation plays an important role. Run time integrity checks can detect any unauthorized code changes. RASP defenses help to protect against attacks to access optional app features in online/offline mode.

M4. Insufficient Input/Output Validation

What Is It?

Common vulnerabilities in APIs are caused by input data being modified by malicious agents. These types of attacks, including SQL injection, command Injection, and cross-site scripting (XSS) attacks are all permanent fixtures on the OWASP Top Ten or the OWASP API Top Ten. Unauthorized code execution, data breaches, data corruption or service interruptions are all possible consequences.

Mobile apps that fail to properly validate and sanitize input can be used to exploit these types of vulnerabilities, so it is important to ensure both your mobile app and APIs are correctly validating user input.

How to Test

Testing for input validation is not only a good practice, but also a requirement for many standards and regulations, such as PCI DSS, and ISO 27001.

It is important to conduct regular security assessments, including penetration testing and code reviews. Thorough testing of input validation at the mobile app is also critical.

How to Mitigate

OWASP contains detailed guidelines on the testing and mitigation of the most common API vulnerabilities.

At the mobile app, you must ensure that you are getting and transmitting the values and format you expect and discard anything else. Again, following secure coding practices and limiting input ranges by using parameterized queries.

An important second line of defense is to put in place the ability to block any requests that do not come from a genuine unmodified app. This can provide effective protection against the exploitation of zero day API vulnerabilities.

M5. Insecure Communication

What is It?

This concerns the threats associated with the transfer or receiving of data. One of the critical attack surfaces in the mobile ecosystem is the API channel between mobile apps and backend servers. Man-in-the-Middle (MitM) attacks on this channel pose a significant threat to mobile users. Intercepting and manipulating communications between devices and servers has become a common attack vector. If an attacker can gain control of the client, and even if SSL is employed, MitM attacks can occur through app repackaging or by using hooking frameworks like Frida to modify app behavior at runtime. It is important to assume that the network layer is subject to eavesdropping and that the trust store on the client device is open to manipulation.

How to Test

The first thing to verify is that SSL/TLS is properly implemented with strong, industry standard cipher suites. Testers should also check and validate which CA providers are used to generate certificates, and if certificate pinning is being used.

Examining an app's network security settings file is a useful exercise for testers: This can immediately iden-

tify red flags such as cleartext traffic being allowed or if self-signed certificates can be used.

How to Mitigate

Certificate pinning is the most effective measure to protect against MitM attacks. By adding a pinned certificate inside the mobile app, communication is restricted only to servers with a valid certificate matching the pinned value. This ensures that unauthorized servers are blocked from establishing a connection.

There are two additional steps which must be taken to make certificate pinning an effective method of protecting the channel. The first is that the pinning solution must work in conjunction with RASP defenses on the device to block hooking frameworks/debuggers such as Frida from manipulating or "unpinning" a connection to permit an MitM to be launched.

Additionally the pinning must be dynamic in the sense that pins can be changed dynamically and immediately as server certificates are rotated. This eliminates any risk of service interruption if (or when) certificates need to be rotated by the devops team. With this type of solution in place, channels to owned and third-party backends can be fully secured.

M6. Inadequate Privacy Controls

What is It?

A brand new entry at number 6 is 'Inadequate Privacy Controls'. Privacy controls are concerned with protecting Personally Identifiable Information (PII),This information could be used by attackers for the purposes of fraud or blackmail.

In general, PII could either be leaked (i.e. a violation of confidentiality), manipulated (violation of integrity) or destroyed/blocked (violation of availability).

How to Test

The first thing to do is to verify that the application is in compliance with privacy regulations such as GDPR and CCPA. These set clear guidelines about how personal data is collected, stored, and used.

Any app that uses PII may expose it like any other sensitive data. This can be verified in the context of other threats on the TOP 10 list as follows:

Insecure data storage and communication (see M5, M9),

Data access with insecure authentication and authorization (see M3, M1), and

Insider attacks on the app's sandbox (see M2, M4, M8).

How to Mitigate

PII that is not stored cannot be hacked, so the safest approach to prevent privacy violations is to keep the amount and variety of PII that is processed to the absolute minimum.

The remaining PII should not be stored or transferred unless absolutely necessary. If it must be stored or transferred, access must be protected with proper authentication and authorization. The other OWASP Mobile Top 10 risks suggest measures to securely store, transfer, access and otherwise handle sensitive data.

Static and dynamic security checking tools might reveal common pitfalls, like logging of sensitive data or leakage to clipboard or URL query parameters.

M7. Insufficient Binary Protections

What is It?

App binaries usually can be downloaded from the app stores or copied from mobile devices, so binary attacks are easy to set up to steal intellectual property or modify the functioning of the app.

As well as IP theft, an app binary could be subject to two types of attacks:

- Reverse engineering: The app binary is decompiled and scanned for valuable information, like secret keys, algorithms, or vulnerabilities.
- Code tampering: The app binary is manipulated, e.g., to gain a game advantage or circumvent paywalls. Alternatively, the app can be injected with malicious code.

How to Test

For a quick check, developers can inspect their own app binaries using the same tools an attacker would use. There are many free or affordable tools, like MobSF, apktool and Ghidra that are also quite easy to use and well documented.

How to Mitigate

Tools exist to provide binary obfuscation of your app's binaries (i.e. your `.apk/.aar` for Android or `.ipa` for iOS) to make it hard for hackers to steal information or repackage apps. None of these approaches are completely secure and with time and skill hackers can usually get what they need from the code.

Using obfuscation to protect code is really an arms race between the developers investing in defenses and attackers who find a way around them. It definitely can slow attackers down and should be considered, especially as a means to protect IP from being used by competing apps.

An important second line of defense is to employ run time protection and app integrity checks to identify and block apps which have been modified in any way. In addition, a means to quickly deploy and rotate API keys and secrets as described in M1 above is also a key element of the mitigation strategy.

M8. Security Misconfiguration

What is It?

This is really the "other" category in the list. Security misconfigurations can happen in mobile apps due to factors such as time constraints, lack of awareness, or human error during development.

How to Test

Detecting security misconfigurations is relatively easy. To determine if your app is vulnerable, you should conduct a thorough security assessment, including code review, security testing, and configuration analysis.

How to Mitigate

Preventing security misconfigurations in mobile apps requires following secure coding and configuration practices. Some suggestions are provided by OWASP, including taking particular care with default credentials and applying the "least privilege principle" prevention measures: Request only the permissions necessary for the proper functioning of the application and avoid storing application files with overly permissive permissions like world-readable and/or world-writable.

M9. Insecure Data Storage

What is It?

Hackers try to extract data for a range of nefarious purposes. Insecure data storage in a mobile application exposes vulnerabilities that threat actors can exploit. Exploits include using unauthorized access to the device's file system, exploiting weak encryption, intercepting data transmissions, and leveraging malware or malicious apps installed on the device. Additionally, rooted or jailbroken devices provide an opportunity for attackers to bypass security measures and gain direct access to sensitive data.

How to Test

Testing should focus on local storage as the app executes. Watch out in particular for sensitive data such as passwords or keys stored in plaintext, output of unprotected logging data or temporary files, or careless caching of sensitive data.

How to Mitigate

Previously covered best practices all have a role to play here: implementing access controls and session management, ensuring strong encryption and secure data transmission, and keeping on top of third-party code and dependencies are all important.

In addition, run time protections can block any hackers trying to steal data by manipulating the running app or the client environment.

M10. Insufficient Cryptography

What is It?

Insufficient Cryptography is the threat posed to mobile apps by poor adoption of modern cryptography best practices. This could be through implementing algorithms that are deemed insecure or not using secure data transport (HTTPS). The attack vector for insecure cryptography in a mobile application involves exploiting vulnerabilities in the cryptographic mechanisms used to protect sensitive information, leading to data breaches or unauthorized access.

How to Test

Evaluate the encryption and decryption processes used in the mobile application, in order to check that they adhere to established cryptographic libraries and frameworks. Avoid custom encryption implementations, as they are more prone to errors and vulnerabilities.

Check also that encryption keys are securely stored on the mobile device and they are not in plain text or easily accessible locations.

How to Mitigate

Best practices are outlined in the OWASP documentation. When utilizing cryptography, ensure you follow the best practices for your particular needs such as using algorithms like AES with at least 128-bit block size or RSA with at least 2048 bits. Stay updated with current cryptographic standards and avoid deprecated or weak algorithms.

Other key approaches are as follows:

- Ensure encryption keys are securely stored on the mobile device. Avoid storing keys in plain text or easily accessible locations. Consider using secure storage mechanisms provided by the operating system or better still deliver credentials to the app only as needed at run time.
- Use secure transport layer protocols, such as HTTPS (HTTP Secure), for transmitting data over networks. Implement proper certificate validation and ensure secure communication channels between the mobile app and backend systems.

General best practices apply also: Make sure you stay current with updates, patches and recommendations about any libraries, frameworks or providers you use, and keep up to date with NIST (National Institute of Standards and Technology) and IETF (Internet Engineering Task Force) guidelines.

Evolution of the OWASP Mobile Top 10 Since 2016

The following table summarizes how the threats have evolved between 2016 and 2024:

Comparison Between 2016-2024		
OWASP-2016	**OWASP-2024-Release**	**Comparison Between 2016-2024**
M1: Improper Platform Usage	M1: Improper Credential Usage	New
M2: Insecure Data Storage	M2: Inadequate Supply Chain Security	New
M3: Insecure Communication	M3: Insecure Authentication / Authorization	Merged M4&M6 to M3
M4: Insecure Authentication	M4: Insufficient Input/Output Validation	New
M5: Insufficient Cryptography	M5: Insecure Communication	Moved from M3 to M5
M6: Insecure Authorization	M6: Inadequate Privacy Controls	New
M7: Client Code Quality	M7: Insufficient Binary Protections	Merged M8&M9 to M7
M8: Code Tampering	M8: Security Misconfiguration	Rewording [M10]
M9: Reverse Engineering	M9: Insecure Data Storage	Moved from M2 to M9
M10: Extraneous Functionality	M10: Insufficient Cryptography	Moved from M5 to M10

Table from OWASP documentation (https://owasp.org/www-project-mobile-top-10/)

Future Evolution

OWASP Noted that there were some serious vulnerabilities which didn't make it onto the Top 10 list this time around. The following are mentioned:

- Data Leakage
- Hardcoded Secrets
- Insecure Access Control
- Path Overwrite and Path Traversal
- Unprotected Endpoints
- Unsafe Sharing

How the Approov Solution Addresses the OWASP Mobile Top 10

Approov directly addresses M1, M5 and because of its patented approach to app and client attestation, provides prevention of exploitation at scale of all the others on the list. The following describes what Approov does and how it fits in.

What Approov Does

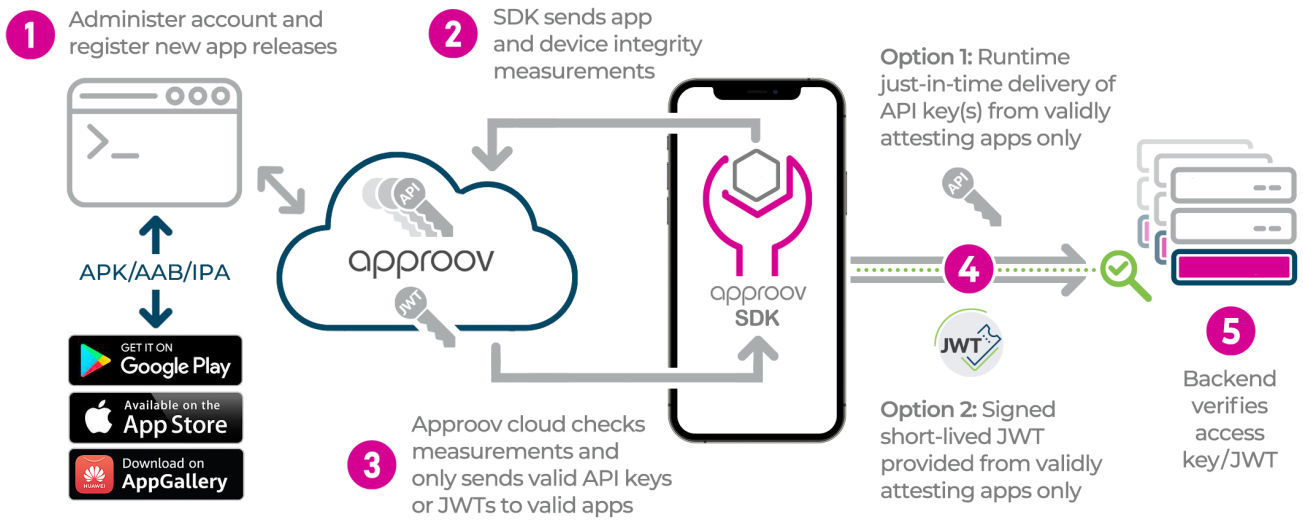
Approov provides a unique and patented run-time shielding solution which is easy to deploy and protects your APIs and the channel between your apps and APIs from any automated attack. It uses a cryptographically signed "Approov token" to allow the app to provide proof that it has passed the runtime shielding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

Additionally, Approov provides a runtime secrets capability whereby app secrets are only delivered just-in-time to the app if it passes attestation. These secrets can then be used by the app, including as API keys to authenticate access to other APIs. In this case no modification to the backend API is required at all, making integration very fast and straightforward.

In summary, Approov provides a line of defense for all of the vulnerabilities on the OWASP Top 10, addresses OWASP MASVS and in addition, prevents exploitation of any vulnerabilities on the OWASP API Top 10 List. Approov provides this protection via the following security features:

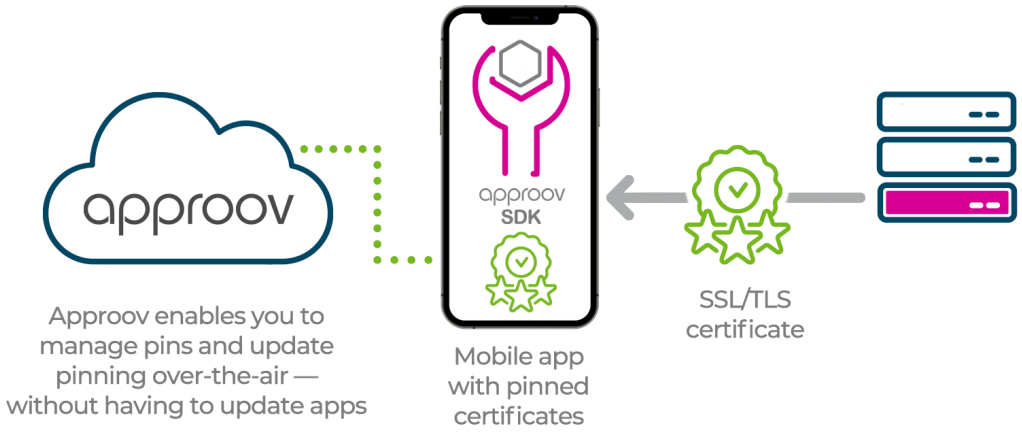
- **App Attestation:** Effective across all app development and deployment platforms and app stores, Approov allows only genuine mobile app instances, running in safe environments, to access your API and blocks all scripts, bots, modified apps and fake apps from accessing your API. Only apps that have been registered with an Approov cloud customer account and which meet the runtime environmental criteria are issued with valid JWT Approov tokens. App registration can be immediately added and revoked from the Approov cloud, allowing tight control of which app versions can access your API.



- **Client Environment Integrity:** Approov Run Time Application Self Protection (RASP) detects a full range of potentially unsafe mobile device environments including device rooting/jailbreaking, emulator or debugger usage, malicious instrumentation frameworks, and cloned apps. Customers can customize which detections should cause attestation failures by selecting from a set of predefined policies or by individually choosing specific threats from the available detections. Changes to security policies roll out

immediately to active apps without the need for new app releases.

- **Man-in-the-Middle Protection:** Approov provides robust protection against Man-in-the-Middle (MitM) attacks by establishing fully pinned communication channels between your apps and APIs. Our dynamic pinning capability guarantees uninterrupted service without requiring app updates. DevOps teams can rest easy knowing that Approov delivers secure over-the-air instant pin updates seamlessly, eliminating management headaches and disruptions. The dynamic pinning feature allows you to update the pins your apps apply by specifying the pin changes in your Approov cloud account. New pinning constraints are distributed as needed whenever apps connect to perform attestations. This flexibility surpasses the limitations of static pinning, while the combined strength of dynamic pinning and app/client attestation enhances defense against MitM attacks on the mobile channel.



- **Secret Management:** Approov supports the secure management of critical secrets used in mobile apps, such as API keys. With the Approov approach, the secret definitions are completely removed from the app code and instead added to your Approov cloud account. Secrets are then delivered to your apps at runtime after a successful app attestation and environment check report that the requesting app is legitimate and safe. This gives your secrets the best protection, unlike other mobile app hardening or obfuscation solutions, they are never even in the app until it is shown to be safe at the moment the secrets are required. The same encryption utility can also be used by the app to store and persist device specific secrets which remain accessible so long as attestations remain valid. The way Approov combines app attestation, RASP and secrets management is unique in the market and its dynamic update facility means that deployed secrets can be updated as easily as certificate pins can be changed. If an API key or other secret is compromised, it can be changed in your Approov account and all deployed apps will receive the update on their next successful attestation.
- **Over-the-air Updates:** In addition to dynamic updates to security policies, pinning configurations and runtime secret keys and values, Approov's over-the-air update capability also enables the Approov threats team to react to the rapidly changing threat landscape. Key aspects of Approov's attestation flow can also be modified through dynamic updates and, where possible, this is used to add detections for new threats as they emerge and without requiring new app releases.
- **Analytics and Reporting:** To simplify reporting, audit and compliance, Approov analytics shows what is happening in your service, with real-time and historical data. Information is presented via a dashboard presenting a top level view and then allowing deeper investigation via the various graphs and options which are available. This is important not only to measure and report on the effectiveness of the solution

but also to help secure and maintain regulatory compliance.

Conclusion

In summary, App teams should integrate the OWASP Mobile Top 10 and MASVS into their entire development lifecycle, from initial design through development, testing, and maintenance. Together, the OWASP projects provide a comprehensive guide for implementing and verifying security in mobile applications throughout the lifecycle.

MASVS in particular can be used:

- In order to understand mobile threat models and build an overall mitigation strategy
- As a way to guide security-aware development
- To use verification levels to guide risk-based testing based on app use-case and risk profile
- As a checklist for procuring security solutions and evaluating third party mobile apps and components
- As a standard to allow security benchmarking between apps

The OWASP Mobile Top 10 on the other hand, can be used as an initial assessment to provide a high level measure of the robustness of your app. If any single one fails, it is a clear sign that you must, as a priority, dedicate more focus and resources on security.

Approov protects your mobile apps from being modified and prevents any manipulation of the client environment, keeps your API secrets secure, and protects the communications channel to the server, providing first and second level protection against all of the OWASP Mobile Top 10.