OWASP
Open Web Application Security Project

Standard

# Mobile AppSec Verification

Version 1.1

**Project Leaders: Bernhard Mueller and Sven Schleier**



# Table of Contents


# Mobile Application Security Verification Standard - Release 1.1

14.07.2018

## Foreword by Bernhard Mueller, OWASP Mobile Project

Technological revolutions can happen quickly. Less than a decade ago, smartphones were clunky devices with little keyboards - expensive playthings for tech-savvy business users. Today, smartphones are an essential part of our lives. We've come to rely on them for information, navigation and communication, and they are ubiquitous both in business and in our social lives.

Every new technology introduces new security risks, and keeping up with those changes is one of the main challenges the security industry faces. The defensive side is always a few steps behind. For example, the default reflex for many was to apply old ways of doing things: Smartphones are like small computers, and mobile apps are just like classic software, so surely the security requirements are similar? But it doesn't work like that. Smartphone operating systems are different from Desktop operating systems, and mobile apps are different from web apps. For example, the classical method of signature-based virus scanning doesn't make sense in modern mobile OS environments: Not only is it incompatible with the mobile app distribution model, it's also technically impossible due to sandboxing restrictions. Also, some vulnerability classes, such as buffer overflows and XSS issues, are less relevant in the context of run-of-the-mill mobile apps than in, say, Desktop apps and web applications (exceptions apply).

Over time, our industry has gotten a better grip on the mobile threat landscape. As it turns out, mobile security is all about data protection: Apps store our personal information, pictures, recordings, notes, account data, business information, location and much more. They act as clients that connect us to services we use on a daily basis, and as communications hubs that processes each and every message we exchange with others. Compromise a person's smartphone and you get unfiltered access to that person's life. When we consider that mobile devices are more readily lost or stolen and mobile malware is on the rise, the need for data protection becomes even more apparent.

A security standard for mobile apps must therefore focus on how mobile apps handle, store and protect sensitive information. Even though modern mobile operating systems like iOS and Android offer good APIs for secure data storage and communication, those have to be implemented and used correctly in order to be effective. Data storage, inter-app communication, proper usage of cryptographic APIs and secure network communication are only some of the aspects that require careful consideration.

An important question in need of industry consensus is how far exactly one should go in protecting the confidentiality and integrity of data. For example, most of us would agree that a mobile app should verify the server certificate in a TLS exchange. But what about SSL certificate pinning? Does not doing it result in a vulnerability? Should this be a

requirement if an app handles sensitive data, or is it maybe even counter-productive? Do we need to encrypt data stored in SQLite databases, even though the OS sandboxes the app? What is appropriate for one app might be unrealistic for another. The MASVS is an attempt to standardize these requirements using verification levels that fit different threat scenarios.

Furthermore, the appearance of root malware and remote administration tools has created awareness of the fact that mobile operating systems themselves have exploitable flaws, so containerization strategies are increasingly used to afford additional protection to sensitive data and prevent client-side tampering. This is where things get complicated. Hardware- backed security features and OS-level containerization solutions, such as Android for Work and Samsung Knox, do exist, but they aren't consistently available across different devices. As a band aid, it is possible to implement software-based protection measures - but unfortunately, there are no standards or testing processes for verifying these kinds of protections.

As a result, mobile app security testing reports are all over the place: For example, some testers report a lack of obfuscation or root detection in an Android app as “security flaw”. On the other hand, measures like string encryption, debugger detection or control flow obfuscation aren't considered mandatory. However, this binary way of looking at things doesn't make sense because resiliency is not a binary proposition: It depends on the particular client-side threats one aims to defend against. Software protections are not useless, but they can ultimately be bypassed, so they must never be used as a replacement for security controls.

The overall goal of the MASVS is to offer a baseline for mobile application security (MASVS- L1), while also allowing for the inclusion of defense-in-depth measures (MASVS-L2) and protections against client-side threats (MASVS-R). The MASVS is meant to achieve the following:

- Provide requirements for software architects and developers seeking to develop secure mobile applications;
- Offer an industry standard that can be tested against in mobile app security reviews;
- Clarify the role of software protection mechanisms in mobile security and provide requirements to verify their effectiveness;
- Provide specific recommendations as to what level of security is recommended for different use-cases.

We are aware that 100% industry consensus is impossible to achieve. Nevertheless, we hope that the MASVS is useful in providing guidance throughout all phases of mobile app development and testing. As an open source standard, the MASVS will evolve over time, and we welcome any contributions and suggestions.

# Frontispiece

## About the Standard

Welcome to the Mobile Application Security Verification Standard (MASVS) 1.1. The MASVS is a community effort to establish a framework of security requirements needed to design, develop and test secure mobile apps on iOS and Android.

The MASVS is a culmination of community effort and industry feedback. We expect this standard to evolve over time and welcome feedback from the community. The best way to get in contact with us is via the OWASP Mobile Project Slack channel:

https://owasp.slack.com/messages/project-mobile_omtg/details/

Accounts can be created at the following URL:

http://owasp.herokuapp.com/.

## Copyright and License




| Project Lead | Lead Author | Contributors and Reviewers |
|---|---|---|
| Sven Schleier & Jeroen Willemsen | Bernhard Mueller | Alexander Antukh, Mesheryakov Aleksey, Bachevsky Artem, Jeroen Beckers, Vladislav Chelnokov, Ben Cheney, Stephen Corbiaux, Manuel Delgado, Ratchenko Denis, Ryan Dewhurst, Tereshin Dmitry, Oprya Egor, Ben Gardiner, Sjoerd Langkemper, Vinícius Henrique Marangoni, Martin Marsicano, Roberto Martelloni, Gall Maxim, Sven Schleier, Abhinav Sejpal, Stefaan Seys, Yogesh Shamrma, Prabhant Singh, Nikhil Soni, Anant Shrivastava, Francesco Stillavato, Abdessamad Temmar, Koki Takeyama, Chelnokov Vladislav, Jeroen Willemsen |

This document started as a fork of the OWASP Application Security Verification Standard written by Jim Manico.

## Sponsors

While both the MASVS and the MSTG are created and maintained by the community on a voluntary basis, sometimes a little bit of outside help is required. We therefore thank our sponsors for providing the funds to be able to hire technical editors. Note that their sponsorship does not influence the content of the MASVS or MSTG in any way. The sponsorship packages are described on the OWASP Project Wiki.

### Honourable Benefactor

# The Mobile Application Security Verification Standard

The MASVS can be used to establish a level of confidence in the security of mobile apps. The requirements were developed with the following objectives in mind:

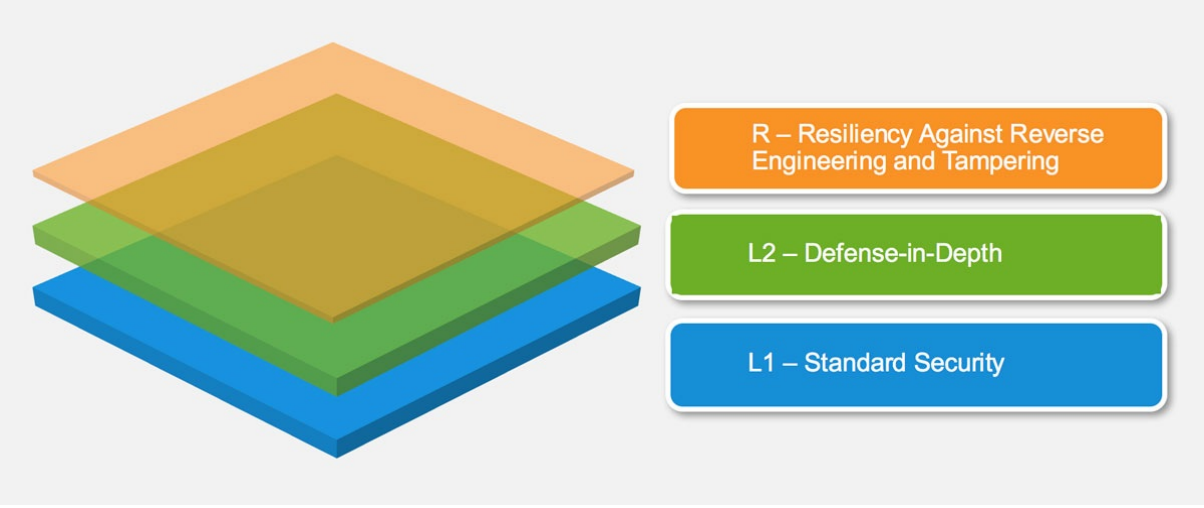
- Use as a metric - To provide a security standard against which existing mobile apps can be compared by developers and application owners;
- Use as guidance - To provide guidance during all phases of mobile app development and testing;
- Use during procurement - To provide a baseline for mobile app security verification.

## Mobile AppSec Model

The MASVS defines two strict security verification levels (L1 and L2), as well a set of reverse engineering resiliency requirements (MASVS-R) that is flexible, i.e. adaptable to an app-specific threat model. MASVS-L1 and MASVS-L2 contain generic security requirements and are recommended for all mobile apps (L1) and apps that handle highly sensitive data (L2). MASVS-R covers additional protective controls that can be applied if preventing client-side threats is a design goal.

Fulfilling the requirements in MASVS-L1 results in a secure app that follows security best practices and doesn't suffer from common vulnerabilities. MASVS-L2 adds additional defense-in-depth controls such as SSL pinning, resulting in an app that is resilient against more sophisticated attacks - assuming the security controls of the mobile operating system are intact and the end user is not viewed as a potential adversary. Fulfilling all, or subsets of, the software protection requirements in MASVS-R helps impede specific client-side threats where the end user is malicious and/or the mobile OS is compromised.

Note that software protection controls listed in MASVS-R and described in the OWASP Mobile Testing Guide can ultimately be bypassed and must never be used as a replacement for security controls. Instead, they are intended to add threat-specific, additional protective controls to apps that also fulfil the MASVS requirements in MASVS L1 or L2.




### Document Structure

The first part of the MASVS contains a description of the security model and available verification levels, followed by recommendations on how to use the standard in practice. The detailed security requirements, along with a mapping to the verification levels, are listed in the second part. The requirements have been grouped into eight categories (V1 to V8) based on technical objective / scope. The following nomenclature is used throughout the MASVS and MSTG:

- *Requirement category:* MASVS-Vx, e.g. MASVS-V2: Data Storage and Privacy
- *Requirement:* MASVS-Vx.y, e.g. MASVS-V2.2: "No sensitive data is written to application logs."

## Verification Levels in Detail

## MASVS-L1: Standard Security

A mobile app that achieves MASVS-L1 adheres to mobile application security best practices. It fulfills basic requirements in terms of code quality, handling of sensitive data, and interaction with the mobile environment. A testing process must be in place to verify the security controls. This level is appropriate for all mobile applications.

## MASVS-L2: Defense-in-Depth

MASVS-L2 introduces advanced security controls that go beyond the standard requirements. To fulfil L2, a threat model must exist, and security must be an integral part of the app's architecture and design. This level is appropriate for applications that handle sensitive data, such as mobile banking.

## MASVS-R: Resiliency Against Reverse Engineering and Tampering

The app has state-of-the-art security, and is also resilient against specific, clearly defined client-side attacks, such as tampering, modding, or reverse engineering to extract sensitive code or data. Such an app either leverages hardware security features or sufficiently strong and verifiable software protection techniques. MASVS-R is applicable to apps that handle highly sensitive data and may serve as a means of protecting intellectual property or tamper-proofing an app.

## Recommended Use

Apps can be verified against MASVS L1 or L2 based on prior risk assessment and overall level of security required. L1 is applicable to all mobile apps, while L2 is generally recommended for apps that handle more sensitive data and/or functionality. MASVS-R (or parts of it) can be applied to verify resiliency against specific threats, such as repackaging or extraction of sensitive data, *in addition* to proper security verification.

In summary, The following verification types are available:

- MASVS-L1
- MASVS-L1+R
- MASVS-L2
- MASVS-L2+R

The different combinations reflect different grades of security and resiliency. The goal is to allow for flexibility: For example, a mobile game might not warrant adding MASVS-L2 security controls such as 2-factor authentication for usability reasons, but have a strong business need for tampering prevention.

## What Verification Type to Choose

Implementing the requirements of MASVS L2 increases security, while at the same time increasing cost of development and potentially worsening the end user experience (the classical trade-off). In general, L2 should be used for apps whenever it makes sense from a risk vs. cost perspective (i.e., where the potential loss caused by a compromise confidentiality or integrity is higher than the cost incurred by the additional security controls). A risk assessment should be the first step before applying the MASVS.

Examples

MASVS-L1

- All mobile apps. MASVS-L1 lists security best practices that can be followed with a reasonable impact on development cost and user experience. Apply the requirements in MASVS-L1 for any app that don't qualify for one of the higher levels.

MASVS-L2

- Health-Care Industry: Mobile apps that store personally identifiable information that can be used for identity theft, fraudulent payments, or a variety of fraud schemes. For the US healthcare sector, compliance considerations include the Health Insurance Portability and Accountability Act (HIPAA) Privacy, Security, Breach Notification Rules and Patient Safety Rule.
- Financial Industry: Apps that enable access to highly sensitive information like credit card numbers, personal information, or allow the user to move funds. These apps warrant additional security controls to prevent fraud. Financial apps need to ensure compliance to the Payment Card Industry Data Security Standard (PCI DSS), Gramm Leech Bliley Act and Sarbanes-Oxley Act (SOX).

MASVS L1+R

- Mobile apps where IP protection is a business goal. The resiliency controls listed in MASVS-R can be used to increase the effort needed to obtain the original source code and to impede tampering / cracking.
- Gaming Industry: Games with an essential need to prevent modding and cheating, such as competitive online games. Cheating is an important issue in online games, as a large amount of cheaters leads to a disgruntled the player base and can ultimately cause a game to fail. MASVS-R provides basic anti-tampering controls to help increase the effort for cheaters.

MASVS L2+R

- Financial Industry: Online banking apps that allow the user to move funds, where techniques code injection and instrumentation on compromised devices pose a risk. In this case, controls from MASVS-R can be used to impede tampering, raising the bar for malware authors.
- All mobile apps that, by design, need to store sensitive data on the mobile device, and at the same time must support a wide range of devices and operating system versions. In this case, resiliency controls can be used as an defense-in-depth measure to increase the effort for attackers aiming to extract the sensitive data.

# Assessment and Certification

## OWASP's Stance on MASVS Certifications and Trust Marks

OWASP, as a vendor-neutral not-for-profit organization, does not certify any vendors, verifiers or software.

All such assurance assertions, trust marks, or certifications are not officially vetted, registered, or certified by OWASP, so an organization relying upon such a view needs to be cautious of the trust placed in any third party or trust mark claiming ASVS certification.

This should not inhibit organizations from offering such assurance services, as long as they do not claim official OWASP certification.

## Guidance for Certifying Mobile Apps

The recommended way of verifying compliance of a mobile app with the MASVS is by performing an "open book" review, meaning that the testers are granted access to key resources such as architects and developers of the app, project documentation, source code, and authenticated access to endpoints, including access to at least one user account for each role.

It is important to note that the MASVS only covers security of the (client-side) mobile app and the network communication between the app and its remote endpoint(s), as well as a few basic and generic requirements related to user authentication and session management. It does not contain specific requirements for the remote services (e.g. web services) associated with the app, safe for a limited set of generic requirements pertaining to authentication and session management. However, MASVS V1 specifies that remote services must be covered by the overall threat model, and be verified against appropriate standards, such as the OWASP ASVS.

A certifying organization must include in any report the scope of the verification (particularly if a key component is out of scope), a summary of verification findings, including passed and failed tests, with clear indications of how to resolve the failed tests. Keeping detailed work papers, screenshots or movies, scripts to reliably and repeatedly exploit an issue, and electronic records of testing, such as intercepting proxy logs and associated notes such as a cleanup list, is considered standard industry practice. It is not sufficient to simply run a tool and report on the failures; this does not provide sufficient evidence that all issues at a certifying level have been tested and tested thoroughly. In case of dispute, there should be sufficient supportive evidence to demonstrate that every verified requirement has indeed been tested.

### Using the OWASP Mobile Security Testing Guide (MSTG)

The OWASP MSTG is a manual for testing the security of mobile apps. It describes the technical processes for verifying the requirements listed in the MASVS. The MSTG includes a list of test cases, each of which map to a requirement in the MASVS. While the MASVS requirements are high-level and generic, the MSTG provides in-depth recommendations and testing procedures on a per-mobile-OS basis.

### The Role of Automated Security Testing Tools

The use of source code scanners and black-box testing tools is encouraged in order to increase efficiency whenever possible. It is however not possible to complete MASVS verification using automated tools alone: Every mobile app is different, and understanding the overall architecture, business logic, and technical pitfalls of the specific technologies and frameworks being used, is a mandatory requirement to verify security of the app.

# Other Uses

## As Detailed Security Architecture Guidance

One of the more common uses for the Mobile Application Security Verification Standard is as a resource for security architects. The two major security architecture frameworks, SABSA or TOGAF, are missing a great deal of information that is necessary to complete mobile application security architecture reviews. MASVS can be used to fill in those gaps by allowing security architects to choose better controls for issues common to mobile apps.

## As a Replacement for Off-the-shelf Secure Coding Checklists

Many organizations can benefit from adopting the MASVS, by choosing one of the two levels, or by forking MASVS and changing what is required for each application's risk level in a domain-specific way. We encourage this type of forking as long as traceability is maintained, so that if an app has passed requirement 4.1, this means the same thing for forked copies as the standard evolves.

## As a Basis for Security Testing Methodologies

A good mobile app security testing methodology should cover all requirements listed in the MASVS. The OWASP Mobile Security Testing Guide (MSTG) describes black-box and white-box test cases for each verification requirement.

## As a Guide for Automated Unit and Integration Tests

The MASVS is designed to be highly testable, with the sole exception of architectural requirements. Automated unit, integration and acceptance testing based on the MASVS requirements can be integrated in the continuous development lifecycle. This not only increases developer security awareness, but also improves the overall quality of the resulting apps, and reduces the amount of findings during security testing in the pre-release phase.

## For Secure Development Training

MASVS can also be used to define characteristics of secure mobile apps. Many "secure coding" courses are simply ethical hacking courses with a light smear of coding tips. This does not help developers. Instead, secure development courses can use the MASVS, with a strong focus on the proactive controls documented in the MASVS, rather than e.g. the Top 10 code security issues.

# V1: Architecture, Design and Threat Modeling Requirements

## Control Objective

In a perfect world, security would be considered throughout all phases of development. In reality however, security is often only a consideration at a late stage in the SDLC. Besides the technical controls, the MASVS requires processes to be in place that ensure that the security has been explicitly addressed when planning the architecture of the mobile app, and that the functional and security roles of all components are known. Since most mobile applications act as clients to remote services, it must be ensured that appropriate security standards are also applied to those services - testing the mobile app in isolation is not sufficient.

The category “V1” lists requirements pertaining to architecture and design of the app. As such, this is the only category that does not map to technical test cases in the OWASP Mobile Testing Guide. To cover topics such as threat modelling, secure SDLC, key management, users of the MASVS should consult the respective OWASP projects and/or other standards such as the ones linked below.

## Security Verification Requirements

The requirements for MASVS-L1 and MASVS-L2 are listed below.

| # | Description | L1 | L2 |
|---|---|---|---|
| 1.1 | All app components are identified and known to be needed. | ✓ | ✓ |
| 1.2 | Security controls are never enforced only on the client side, but on the respective remote endpoints. | ✓ | ✓ |
| 1.3 | A high-level architecture for the mobile app and all connected remote services has been defined and security has been addressed in that architecture. | ✓ | ✓ |
| 1.4 | Data considered sensitive in the context of the mobile app is clearly identified. | ✓ | ✓ |
| 1.5 | All app components are defined in terms of the business functions and/or security functions they provide. | | ✓ |
| 1.6 | A threat model for the mobile app and the associated remote services has been produced that identifies potential threats and countermeasures. | | ✓ |
| 1.7 | All security controls have a centralized implementation. | | ✓ |
| 1.8 | There is an explicit policy for how cryptographic keys (if any) are managed, and the lifecycle of cryptographic keys is enforced. Ideally, follow a key management standard such as NIST SP 800-57. | | ✓ |
| 1.9 | A mechanism for enforcing updates of the mobile app exists. | | ✓ |
| 1.10 | Security is addressed within all parts of the software development lifecycle. | | ✓ |

## References

For more information, see also:

- OWASP Mobile Top 10: M10 - Extraneous Functionality: https://www.owasp.org/index.php/Mobile_Top_10_2016-

M10-Extraneous_Functionality

- OWASP Security Architecture cheat sheet: https://www.owasp.org/index.php/Application_Security_Architecture_Cheat_Sheet
- OWASP Thread modelling: https://www.owasp.org/index.php/Application_Threat_Modeling
- OWASP Secure SDLC Cheat Sheet: https://www.owasp.org/index.php/Secure_SDLC_Cheat_Sheet
- Microsoft SDL: https://www.microsoft.com/en-us/sdl/
- NIST SP 800-57: http://csrc.nist.gov/publications/nistpubs/800-57/sp800-57-Part1-revised2_Mar08-2007.pdf

# V2: Data Storage and Privacy Requirements

## Control Objective

The protection of sensitive data, such as user credentials and private information, is a key focus in mobile security. Firstly, sensitive data can be unintentionally exposed to other apps running on the same device if operating system mechanisms like IPC are used improperly. Data may also unintentionally leak to cloud storage, backups, or the keyboard cache. Additionally, mobile devices can be lost or stolen more easily compared to other types of devices, so an adversary gaining physical access is a more likely scenario. In that case, additional protections can be implemented to make retrieving the sensitive data more difficult.

Note that, as the MASVS is app-centric, it does not cover device-level policies such as those enforced by MDM solutions. We encourage the use of such policies in an Enterprise context to further enhance data security.

### Definition of Sensitive Data

Sensitive data in the context of the MASVS pertains to both user credentials and any other data considered sensitive in the particular context, such as:

- Personally identifiable information (PII) that can be abused for identity theft: Social security numbers, credit card numbers, bank account numbers, health information;
- Highly sensitive data that would lead to reputational harm and/or financial costs if compromised: Contractual information, information covered by non-disclosure agreements, management information;
- Any data that must be protected by law or for compliance reasons.

## Security Verification Requirements

The vast majority of data disclosure issues can be prevented by following simple rules. Most of the controls listed in this chapter are mandatory for all verification levels.

| # | Description | L1 | L2 |
|---|---|---|---|
| 2.1 | System credential storage facilities are used appropriately to store sensitive data, such as PII, user credentials or cryptographic keys. | ✓ | ✓ |
| 2.2 | No sensitive data should be stored outside of the app container or system credential storage facilities. | ✓ | ✓ |
| 2.3 | No sensitive data is written to application logs. | ✓ | ✓ |
| 2.4 | No sensitive data is shared with third parties unless it is a necessary part of the architecture. | ✓ | ✓ |
| 2.5 | The keyboard cache is disabled on text inputs that process sensitive data. | ✓ | ✓ |
| 2.6 | No sensitive data is exposed via IPC mechanisms. | ✓ | ✓ |
| 2.7 | No sensitive data, such as passwords or pins, is exposed through the user interface. | ✓ | ✓ |
| 2.8 | No sensitive data is included in backups generated by the mobile operating system. | | ✓ |
| 2.9 | The app removes sensitive data from views when backgrounded. | | ✓ |
| 2.10 | The app does not hold sensitive data in memory longer than necessary, and memory is cleared explicitly after use. | | ✓ |

| | | | |
|---|---|---|---|
| 2.11 | The app enforces a minimum device-access-security policy, such as requiring the user to set a device passcode. | | ✓ |
| 2.12 | The app educates the user about the types of personally identifiable information processed, as well as security best practices the user should follow in using the app. | | ✓ |

# References

The OWASP Mobile Security Testing Guide provides detailed instructions for verifying the requirements listed in this section.

- For Android - https://github.com/OWASP/owasp-mstg/blob/master/Document/0x05d-Testing-Data-Storage.md
- For iOS - https://github.com/OWASP/owasp-mstg/blob/master/Document/0x06d-Testing-Data-Storage.md

For more information, see also:

- OWASP Mobile Top 10: M2 - Insecure Data Storage: https://www.owasp.org/index.php/Mobile_Top_10_2016-M2-Insecure_Data_Storage
- CWE: https://cwe.mitre.org/data/definitions/922.html

# V3: Cryptography Requirements

## Control Objective

Cryptography is an essential ingredient when it comes to protecting data stored on a mobile device. It is also a category where things can go horribly wrong, especially when standard conventions are not followed. The purpose of the controls in this chapter is to ensure that the verified application uses cryptography according to industry best practices, including:

- Use of proven cryptographic libraries;
- Proper choice and configuration of cryptographic primitives;
- A suitable random number generator wherever randomness is required.

## Security Verification Requirements

| # | Description | L1 | L2 |
|---|---|---|---|
| 3.1 | The app does not rely on symmetric cryptography with hardcoded keys as a sole method of encryption. | ✓ | ✓ |
| 3.2 | The app uses proven implementations of cryptographic primitives. | ✓ | ✓ |
| 3.3 | The app uses cryptographic primitives that are appropriate for the particular use-case, configured with parameters that adhere to industry best practices. | ✓ | ✓ |
| 3.4 | The app does not use cryptographic protocols or algorithms that are widely considered depreciated for security purposes. | ✓ | ✓ |
| 3.5 | The app doesn't re-use the same cryptographic key for multiple purposes. | ✓ | ✓ |
| 3.6 | All random values are generated using a sufficiently secure random number generator. | ✓ | ✓ |

## References

The OWASP Mobile Security Testing Guide provides detailed instructions for verifying the requirements listed in this section.

- Android - https://github.com/OWASP/owasp-mstg/blob/master/Document/0x05e-Testing-Cryptography.md
- iOS - https://github.com/OWASP/owasp-mstg/blob/master/Document/0x06e-Testing-Cryptography.md

For more information, see also:

- OWASP Mobile Top 10: M5 - Insufficient Cryptography: https://www.owasp.org/index.php/Mobile_Top_10_2016-M5-Insufficient_Cryptography
- CWE: https://cwe.mitre.org/data/definitions/310.html

# V4: Authentication and Session Management Requirements

## Control Objective

In most cases, users logging into a remote service is an integral part of the overall mobile app architecture. Even though most of the logic happens at the endpoint, MASVS defines some basic requirements regarding how user accounts and sessions are to be managed.

## Security Verification Requirements

| # | Description | L1 | L2 |
|---|---|---|---|
| 4.1 | If the app provides users access to a remote service, some form of authentication, such as username/password authentication, is performed at the remote endpoint. | ✓ | ✓ |
| 4.2 | If stateful session management is used, the remote endpoint uses randomly generated session identifiers to authenticate client requests without sending the user's credentials. | ✓ | ✓ |
| 4.3 | If stateless token-based authentication is used, the server provides a token that has been signed using a secure algorithm. | ✓ | ✓ |
| 4.4 | The remote endpoint terminates the existing session when the user logs out. | ✓ | ✓ |
| 4.5 | A password policy exists and is enforced at the remote endpoint. | ✓ | ✓ |
| 4.6 | The remote endpoint implements a mechanism to protect against the submission of credentials an excessive number of times. | ✓ | ✓ |
| 4.7 | Sessions are invalidated at the remote endpoint after a predefined period of inactivity and access tokens expire. | ✓ | ✓ |
| 4.8 | Biometric authentication, if any, is not event-bound (i.e. using an API that simply returns "true" or "false"). Instead, it is based on unlocking the keychain/keystore. | | ✓ |
| 4.9 | A second factor of authentication exists at the remote endpoint and the 2FA requirement is consistently enforced. | | ✓ |
| 4.10 | Sensitive transactions require step-up authentication. | | ✓ |
| 4.11 | The app informs the user of all login activities with their account. Users are able view a list of devices used to access the account, and to block specific devices. | | ✓ |

## References

The OWASP Mobile Security Testing Guide provides detailed instructions for verifying the requirements listed in this section.

- For Android - https://github.com/OWASP/owasp-mstg/blob/master/Document/0x05f-Testing-Authentication.md
- For iOS - https://github.com/OWASP/owasp-mstg/blob/master/Document/0x06f-Testing-Authentication-and-Session-Management.md

For more information, see also:

- OWASP Mobile Top 10: M4 - Insecure Authentication: https://www.owasp.org/index.php/Mobile_Top_10_2016-M4-Insecure_Authentication

- OWASP Mobile Top 10: M6 - Insecure Authorization: https://www.owasp.org/index.php/Mobile_Top_10_2016-M6-Insecure_Authorization
- CWE: https://cwe.mitre.org/data/definitions/287.html

# V5: Network Communication Requirements

## Control Objective

The purpose of the controls listed in this section is to ensure the confidentiality and integrity of information exchanged between the mobile app and remote service endpoints. At the very least, a mobile app must set up a secure, encrypted channel for network communication using the TLS protocol with appropriate settings. Level 2 lists additional defense-in-depth measure such as SSL pinning.

## Security Verification Requirements

| # | Description | L1 | L2 |
|---|---|---|---|
| 5.1 | Data is encrypted on the network using TLS. The secure channel is used consistently throughout the app. | ✓ | ✓ |
| 5.2 | The TLS settings are in line with current best practices, or as close as possible if the mobile operating system does not support the recommended standards. | ✓ | ✓ |
| 5.3 | The app verifies the X.509 certificate of the remote endpoint when the secure channel is established. Only certificates signed by a trusted CA are accepted. | ✓ | ✓ |
| 5.4 | The app either uses its own certificate store, or pins the endpoint certificate or public key, and subsequently does not establish connections with endpoints that offer a different certificate or key, even if signed by a trusted CA. | | ✓ |
| 5.5 | The app doesn't rely on a single insecure communication channel (email or SMS) for critical operations, such as enrollments and account recovery. | | ✓ |
| 5.6 | The app only depends on up-to-date connectivity and security libraries. | | ✓ |

## References

The OWASP Mobile Security Testing Guide provides detailed instructions for verifying the requirements listed in this section.

- Android - https://github.com/OWASP/owasp-mstg/blob/master/Document/0x05g-Testing-Network-Communication.md
- iOS - https://github.com/OWASP/owasp-mstg/blob/master/Document/0x06g-Testing-Network-Communication.md

For more information, see also:

- OWASP Mobile Top 10: M3 - Insecure Communication: https://www.owasp.org/index.php/Mobile_Top_10_2016-M3-Insecure_Communication
- CWE: https://cwe.mitre.org/data/definitions/319.html
- CWE: https://cwe.mitre.org/data/definitions/295.html

# V6: Platform Interaction Requirements

## Control Objective

The controls in this group ensure that the app uses platform APIs and standard components in a secure manner. Additionally, the controls cover communication between apps (IPC).

## Security Verification Requirements

| # | Description | L1 | L2 |
|---|---|---|---|
| 6.1 | The app only requests the minimum set of permissions necessary. | ✓ | ✓ |
| 6.2 | All inputs from external sources and the user are validated and if necessary sanitized. This includes data received via the UI, IPC mechanisms such as intents, custom URLs, and network sources. | ✓ | ✓ |
| 6.3 | The app does not export sensitive functionality via custom URL schemes, unless these mechanisms are properly protected. | ✓ | ✓ |
| 6.4 | The app does not export sensitive functionality through IPC facilities, unless these mechanisms are properly protected. | ✓ | ✓ |
| 6.5 | JavaScript is disabled in WebViews unless explicitly required. | ✓ | ✓ |
| 6.6 | WebViews are configured to allow only the minimum set of protocol handlers required (ideally, only https is supported). Potentially dangerous handlers, such as file, tel and app-id, are disabled. | ✓ | ✓ |
| 6.7 | If native methods of the app are exposed to a WebView, verify that the WebView only renders JavaScript contained within the app package. | ✓ | ✓ |
| 6.8 | Object deserialization, if any, is implemented using safe serialization APIs. | ✓ | ✓ |

## References

The OWASP Mobile Security Testing Guide provides detailed instructions for verifying the requirements listed in this section.

- Android - https://github.com/OWASP/owasp-mstg/blob/master/Document/0x05h-Testing-Platform-Interaction.md
- iOS - https://github.com/OWASP/owasp-mstg/blob/master/Document/0x06h-Testing-Platform-Interaction.md

For more information, see also:

- OWASP Mobile Top 10: M1 - Improper Platform Usage: https://www.owasp.org/index.php/Mobile_Top_10_2016-M1-Improper_Platform_Usage
- CWE: https://cwe.mitre.org/data/definitions/20.html
- CWE: https://cwe.mitre.org/data/definitions/749.html

# V7: Code Quality and Build Setting Requirements

## Control Objective

The goal of this control is to ensure that basic security coding practices are followed in developing the app, and that "free" security features offered by the compiler are activated.

## Security Verification Requirements

| # | Description | L1 | L2 |
|---|---|---|---|
| 7.1 | The app is signed and provisioned with a valid certificate, of which the private key is properly protected. | ✓ | ✓ |
| 7.2 | The app has been built in release mode, with settings appropriate for a release build (e.g. non-debuggable). | ✓ | ✓ |
| 7.3 | Debugging symbols have been removed from native binaries. | ✓ | ✓ |
| 7.4 | Debugging code has been removed, and the app does not log verbose errors or debugging messages. | ✓ | ✓ |
| 7.5 | All third party components used by the mobile app, such as libraries and frameworks, are identified, and checked for known vulnerabilities. | ✓ | ✓ |
| 7.6 | The app catches and handles possible exceptions. | ✓ | ✓ |
| 7.7 | Error handling logic in security controls denies access by default. | ✓ | ✓ |
| 7.8 | In unmanaged code, memory is allocated, freed and used securely. | ✓ | ✓ |
| 7.9 | Free security features offered by the toolchain, such as byte-code minification, stack protection, PIE support and automatic reference counting, are activated. | ✓ | ✓ |

## References

The OWASP Mobile Security Testing Guide provides detailed instructions for verifying the requirements listed in this section.

- Android - https://github.com/OWASP/owasp-mstg/blob/master/Document/0x05i-Testing-Code-Quality-and-Build-Settings.md
- iOS - https://github.com/OWASP/owasp-mstg/blob/master/Document/0x06i-Testing-Code-Quality-and-Build-Settings.md

For more information, see also:

- OWASP Mobile Top 10: M7 - Poor Code Quality: https://www.owasp.org/index.php/Mobile_Top_10_2016-M7-Poor_Code_Quality
- CWE: https://cwe.mitre.org/data/definitions/119.html
- CWE: https://cwe.mitre.org/data/definitions/89.html
- CWE: https://cwe.mitre.org/data/definitions/388.html
- CWE: https://cwe.mitre.org/data/definitions/489.html

# V8: Resilience Requirements

## Control objective

This section covers defense-in-depth measures recommended for apps that process, or give access to, sensitive data or functionality. Lack of any of these controls does not cause a vulnerability - instead, they are meant to increase the app's resilience against reverse engineering and specific client-side attacks.

The controls in this section should be applied as needed, based on an assessment of the risks caused by unauthorized tampering with the app and/or reverse engineering of the code. We suggest consulting the OWASP document "Technical Risks of Reverse Engineering and Unauthorized Code Modification Reverse Engineering and Code Modification Prevention" (see references below) for a list business risks as well as associated technical threats.

For any of the controls in the list below to be effective, the app must fulfil at least all of MASVS-L1 (i.e., solid security controls must be in place), as well as all lower-numbered requirements in V8. For examples, the obfuscation controls listed in under "impede comprehension" must be combined with "app isolation", "impede dynamic analysis and tampering" and "device binding".

Note that software protections must never be used as a replacement for security controls. The controls listed in MASVR-R are intended to add threat-specific, additional protective controls to apps that also fulfil the MASVS security requirements.

The following considerations apply:

1. A threat model must be defined that clearly outlines the client-side threats defended against. Additionally, the grade of protection the scheme is meant to provide must be specified. For example, a stated goal could be to force authors of targeted malware seeking to instrument the app to invest significant manual reverse engineering effort.
2. The threat model must be sensical. For example, hiding a cryptographic key in a white-box implementation is besides the point if the attacker can simply code-lift the white-box as a whole.
3. The effectiveness of the protection should always be verified by a human expert with experience in testing the particular types of anti-tampering and obfuscation used (see also the "reverse engineering" and "assessing software protections" chapters in the Mobile Security Testing Guide).

### Impede Dynamic Analysis and Tampering

| # | Description | R |
|---|---|---|
| 8.1 | The app detects, and responds to, the presence of a rooted or jailbroken device either by alerting the user or terminating the app. | ✓ |
| 8.2 | The app prevents debugging and/or detects, and responds to, a debugger being attached. All available debugging protocols must be covered. | ✓ |
| 8.3 | The app detects, and responds to, tampering with executable files and critical data within its own sandbox. | ✓ |
| 8.4 | The app detects, and responds to, the presence of widely used reverse engineering tools and frameworks on the device. | ✓ |
| 8.5 | The app detects, and responds to, being run in an emulator. | ✓ |
| 8.6 | The app detects, and responds to, tampering the code and data in its own memory space. | ✓ |
| | The app implements multiple mechanisms in each defense category (8.1 to 8.6). Note that | |

| | | |
|---|---|---|
| 8.7 | resiliency scales with the amount, diversity of the originality of the mechanisms used. | ✓ |
| 8.8 | The detection mechanisms trigger responses of different types, including delayed and stealthy responses. | ✓ |
| 8.9 | Obfuscation is applied to programmatic defenses, which in turn impede de-obfuscation via dynamic analysis. | ✓ |

### Device Binding

| # | Description | R |
|---|---|---|
| 8.10 | The app implements a 'device binding' functionality using a device fingerprint derived from multiple properties unique to the device. | ✓ |

### Impede Comprehension

| # | Description | R |
|---|---|---|
| 8.11 | All executable files and libraries belonging to the app are either encrypted on the file level and/or important code and data segments inside the executables are encrypted or packed. Trivial static analysis does not reveal important code or data. | ✓ |
| 8.12 | If the goal of obfuscation is to protect sensitive computations, an obfuscation scheme is used that is both appropriate for the particular task and robust against manual and automated de-obfuscation methods, considering currently published research. The effectiveness of the obfuscation scheme must be verified through manual testing. Note that hardware-based isolation features are preferred over obfuscation whenever possible. | ✓ |

## References

The OWASP Mobile Security Testing Guide provides detailed instructions for verifying the requirements listed in this section.

- Android - https://github.com/OWASP/owasp-mstg/blob/master/Document/0x05j-Testing-Resiliency-Against-Reverse-Engineering.md
- iOS - https://github.com/OWASP/owasp-mstg/blob/master/Document/0x06j-Testing-Resiliency-Against-Reverse-Engineering.md

For more information, see also:

- OWASP Mobile Top 10: M8 - Code Tampering: https://www.owasp.org/index.php/Mobile_Top_10_2016-M8-Code_Tampering
- OWASP Mobile Top 10: M9 - Reverse Engineering: https://www.owasp.org/index.php/Mobile_Top_10_2016-M9-Reverse_Engineering
- WASP Reverse Engineering Threats - https://www.owasp.org/index.php/Technical_Risks_of_Reverse_Engineering_and_Unauthorized_Code_Modification
- OWASP Reverse Engineering and Code Modification Prevention - https://www.owasp.org/index.php/OWASP_Reverse_Engineering_and_Code_Modification_Prevention_Project

# Appendix A: Glossary

- 2FA – Two-factor authentication(2FA) adds a second level of authentication to an account log-in.
- Address Space Layout Randomization (ASLR) – A technique to make exploiting memory corruption bugs more difficult.
- Application Security – Application-level security focuses on the analysis of components that comprise the application layer of the Open Systems Interconnection Reference Model (OSI Model), rather than focusing on for example the underlying operating system or connected networks.
- Application Security Verification – The technical assessment of an application against the OWASP MASVS.
- Application Security Verification Report – A report that documents the overall results and supporting analysis produced by the verifier for a particular application.
- Authentication – The verification of the claimed identity of an application user.
- Automated Verification – The use of automated tools (either dynamic analysis tools, static analysis tools, or both) that use vulnerability signatures to find problems.
- Black box testing – It is a method of software testing that examines the functionality of an application without peering into its internal structures or workings.
- Component – a self-contained unit of code, with associated disk and network interfaces that communicates with other components.
- Cross-Site Scripting (XSS) – A security vulnerability typically found in web applications allowing the injection of client-side scripts into content.
- Cryptographic module – Hardware, software, and/or firmware that implements cryptographic algorithms and/or generates cryptographic keys.
- CWE – CWE is a community-developed list of common software security weaknesses. It serves as a common language, a measuring stick for software security tools, and as a baseline for weakness identification, mitigation, and prevention efforts.
- DAST –Dynamic application security testing (DAST) technologies are designed to detect conditions indicative of a security vulnerability in an application in its running state.
- Design Verification – The technical assessment of the security architecture of an application.
- Dynamic Verification – The use of automated tools that use vulnerability signatures to find problems during the execution of an application.
- Globally Unique Identifier (GUID) – a unique reference number used as an identifier in software.
- Hyper Text Transfer Protocol (HTTP) – An application protocol for distributed, collaborative, hypermedia information systems. It is the foundation of data communication for the World Wide Web.
- Hardcoded keys – Cryptographic keys which are stored in the device itself.
- IPC – Inter Process Communications,In IPC Processes communicate with each other and with the kernel to coordinate their activities.
- Input Validation – The canonicalization and validation of untrusted user input.
- JAVA Bytecode – Java bytecode is the instruction set of the Java virtual machine(JVM). Each bytecode is composed of one, or in some cases two bytes that represent the instruction (opcode), along with zero or more bytes for passing parameters.
- Malicious Code – Code introduced into an application during its development unbeknownst to the application owner, which circumvents the application's intended security policy. Not the same as malware such as a virus or worm!
- Malware – Executable code that is introduced into an application during runtime without the knowledge of the application user or administrator.
- Open Web Application Security Project (OWASP) – The Open Web Application Security Project (OWASP) is a worldwide free and open community focused on improving the security of application software. Our mission is to make application security "visible," so that people and organizations can make informed decisions about application security risks. See: http://www.owasp.org/

- Personally Identifiable Information (PII) - is information that can be used on its own or with other information to identify, contact, or locate a single person, or to identify an individual in context.
- PIE – Position-independent executable (PIE) is a body of machine code that, being placed somewhere in the primary memory, executes properly regardless of its absolute address.
- PKI – A PKI is an arrangement that binds public keys with respective identities of entities. The binding is established through a process of registration and issuance of certificates at and by a certificate authority (CA).
- SAST – Static application security testing (SAST) is a set of technologies designed to analyze application source code, byte code and binaries for coding and design conditions that are indicative of security vulnerabilities. SAST solutions analyze an application from the “inside out” in a nonrunning state.
- SDLC – Software development lifecycle.
- Security Architecture – An abstraction of an application's design that identifies and describes where and how security controls are used, and also identifies and describes the location and sensitivity of both user and application data.
- Security Configuration – The runtime configuration of an application that affects how security controls are used.
- Security Control – A function or component that performs a security check (e.g. an access control check) or when called results in a security effect (e.g. generating an audit record).
- SQL Injection (SQLi) – A code injection technique used to attack data driven applications, in which malicious SQL statements are inserted into an entry point.
- SSO Authentication – Single Sign On(SSO) occurs when a user logs in to one Client and is then signed in to other Clients automatically, regardless of the platform, technology, or domain the user is using. For example when you log in in google you automatically login in the youtube , docs and mail service.
- Threat Modeling – A technique consisting of developing increasingly refined security architectures to identify threat agents, security zones, security controls, and important technical and business assets.
- Transport Layer Security – Cryptographic protocols that provide communication security over the Internet
- URI/URL/URL fragments – A Uniform Resource Identifier is a string of characters used to identify a name or a web resource. A Uniform Resource Locator is often used as a reference to a resource.
- User acceptance testing (UAT) – Traditionally a test environment that behaves like the production environment where all software testing is performed before going live.
- Verifier – The person or team that is reviewing an application against the OWASP ASVS requirements.
- Whitelist – A list of permitted data or operations, for example a list of characters that are allowed to perform input validation.
- X.509 Certificate – An X.509 certificate is a digital certificate that uses the widely accepted international X.509 public key infrastructure (PKI) standard to verify that a public key belongs to the user, computer or service identity contained within the certificate.

# Appendix B: References

The following OWASP projects are most likely to be useful to users/adopters of this standard:

- OWASP Mobile Security Project - https://www.owasp.org/index.php/OWASP_Mobile_Security_Project
- OWASP Mobile Security Testing Guide - https://www.owasp.org/index.php/OWASP_Mobile_Security_Testing_Guide
- OWASP Mobile Top 10 Risks - https://www.owasp.org/index.php/Projects/OWASP*Mobile_Security_Project*-_Top_Ten_Mobile_Risks
- OWASP Reverse Engineering and Code Modification Prevention - https://www.owasp.org/index.php/OWASP_Reverse_Engineering_and_Code_Modification_Prevention_Project

Similarly, the following web sites are most likely to be useful to users/adopters of this standard:

- MITRE Common Weakness Enumeration - http://cwe.mitre.org/
- PCI Security Standards Council - https://www.pcisecuritystandards.org
- PCI Data Security Standard (DSS) v3.0 Requirements and Security Assessment Procedures https://www.pcisecuritystandards.org/documents/PCI_DSS_v3.pdf