# 



# Mobile Application Security Verification Standard

[date]

# Acknowledgements

## About the Standard

The Mobile Application Security Verification Standard is a list of security requirements for mobile applications that can be used by architects, developers, testers, security professionals, and consumers to define what a secure mobile application is.

## Copyright and License

  
Copyright © 2016 The OWASP Foundation. This document is released under the Creative Commons Attribution ShareAlike 3.0 license. For any reuse or distribution, you must make clear to others the license terms of this work.

|  |  |  |
| --- | --- | --- |
| Project Leads | Lead Authors | Contributors and Reviewers |
| Bernhard Mueller, Sven Schleier | Stephen Corbiaux, Bernhard Mueller | Sven Schleier, Anant Shrivastava, Prabhant Singh, Abdessamad Temmar, Jeroen Willemsen |

# Preface

Welcome to the Mobile Application Security Verification Standard (MASVS) version 1.0. The MASVS is a community effort to establish a framework of security requirements and controls that focus on normalising the functional and non-functional security controls required when designing, developing and testing mobile applications on iOS and Android.

MASVS v1.0 is a culmination of community effort and industry feedback.

## Introduction

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 private lives.

Every new technology introduces new security risks, and keeping up with those changes is one of the main challenges the security industry faces: We're always lagging a bit 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 like 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: Virus scanners don't make sense on modern mobile OSes, and attackers rarely exploit buffer overflows and XSS vulnerabilities in mobile apps (rare exceptions exist).

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, notes, account data, business information, 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 great APIs for secure data storage and communcation, those have to be used correctly to be effective. Data storage, inter-app communication, proper use 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 sensitive 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 pinning? Does not doing it result in a vulnerability? Should it be a requirement if an app handles sensitive data, or is it maybe even counter-productive? What about locking the app after a certain idle time? 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 overkill for another. The MASVS is an attempt to standardize these requirements using verification levels for different use-cases.

Things become even more complicated when we start considering containerization and software protections. Some protective measures are widely assumed to be necessary - for example, many testers will report a lack of identifier renaming or root detection in an Android app as security flaw. On the other hand, we don't usually consider string encryption, debugger detection or control flow obfuscation as mandatory. However, this binary way of looking at things doesn't make sense because software protection is not a binary proposition: The question is not whether an app can be reverse engineered or not, but rather how much has been done to make the process more difficult. Finding the right requirements and testing processes for software protections is a difficult problem unique to mobile security, where containerization and obfuscation is becoming quite common.

Finally, root malware and remote administration kits have made us aware that mobile operating systems themselves have exploitable flaws, so the case can be made that containerization, obfuscation and reactive defences are valid strategies to add protective layers to mobile apps. Our goal is therefore to provide software protection requirements in the higher verification levels, as well as testing procedures for validating the effectiveness of the protections (which will be documented in the Mobile Security Testing Guide).

The overall goal of the MASVS is to offer a baseline list of mobile app security best practices, while also allowing for a defense-in-depth approach that adds additional protection mechanisms. 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 to be used as a basis for mobile app security code review and testing methodologies;
* 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 any contributions and suggestions are welcome.

# Using 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 application developers and application owners with a framework wich allows to measure the security, and thus the degree of trust that can be placed in their mobile applications.
* Use as guidance - To provide guidance in regards to security controls necessary to implement in order to satisfy application security requirements
* Use during procurement - Provide a baseline for mobile app security verification requirements.

Figure 1 - Uses of MASVS for organizations and tool/service providers

## Mobile Application Security Verification Levels

The Mobile Application Security Verification Standard defines five security verification levels, with each level increasing in depth.

### MASVS Level 1 - Standard Security

An application that achieves MASVS level 1 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 Level 2 - Defense-in-Depth

Level 2 introduces advanced security controls that go beyond the standard requirements. To fulfill level 2, a threat model must exist, and security must be considered during the design phase. This level is appropriate for applications that handle sensitive data, such as mobile banking.

### MASVS Level 3 - Defense-in-Depth and Resiliency

In addition to state-of-the-art security and defense-in-depth, level 3 defines mechanisms that increase the cost of reverse engineering the application. It can be applied to add an additional layer of protection for applications that process sensitive data. Vendors may also opt to fulfill these requirements as a means of protecting intellectual property.

### MASVS Level 4 - Defense-in-Depth and Strong Resiliency

An application that achieves MASVS level 4 has both state-of-the-art security controls and software protection. Such an application is highly resilient against attacks and reverse engineering attempts. Level 4 is meant to be applied to applications that handle highly sensitive data. Vendors may also opt to fulfill these requirements as a means of protecting intellectual property.

## How to use this standard

One of the best ways to use the Mobile Application Security Verification Standard is to use it as blueprint to create a secure coding checklist specific to your application, platform or organization. Tailoring the MASVS to your use cases will increase the focus on the security requirements that are most important to your projects and environments.

## Applying MASVS in Practice

Different threats have different motivations. Some industries have unique information and technology assets and domain specific regulatory compliance requirements.

Below we provide industry-specific guidance regarding recommended MASVS levels. Although some unique criteria and some differences in threats exist for each industry, a common theme throughout all industry segments is that opportunistic attackers will look for any easily exploitable vulnerable applications, which is why MASVS Level 1 is recommended for all applications regardless of industry. (... todo ...)

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Industry | Threat Profile | L1 Recommendation | L2 Recommendation | L3 Recommendation | L4 Recommendation |
| Finance and Insurance | Although this segment will experience attempts from opportunistic attackers, it is often viewed as a high value target by motivated attackers and attacks are often financially motivated. Commonly, attackers are looking for sensitive data or account credentials that can be used to commit fraud or to benefit directly by leveraging money movement functionality built into applications. Techniques often include stolen credentials, application-level attacks, and social engineering. Some major compliance considerations include Payment Card Industry Data Security Standard (PCI DSS),Gramm Leech Bliley Act and Sarbanes-Oxley Act (SOX). | All mobile apps. | Apps that enable access to Personally Identifiable Information. | Apps that enable access to highly sensitive information like credit card numbers, personal information, or that can move limited amounts of money in limited ways. Examples include: (i) transfer money between accounts at the same institution or(ii) a slower form of money movement (e.g. ACH) with transaction limits or(iii) wire transfers with hard transfer limits within a period of time. | Apps that enable access to large amounts of sensitive information or that allow either rapid transfer of large sums of money (e.g. wire transfers) and/or transfer of large sums of money in the form of individual transactions or as a batch of smaller transfers. |
| Manufacturing, professional, transportation, technology, utilities, infrastructure, and defense | These industries may not appear to have very much in common, but the threat actors who are likely to attack organizations in this segment are more likely to perform focused attacks with more time, skill, and resources. Often the sensitive information or systems are not easy to locate and require leveraging insiders and social engineering techniques. Attacks may involve insiders, outsiders, or be collusion between the two. Their goals may include gaining access to intellectual property for strategic or technological advantage. We also do not want to overlook attackers looking to abuse application functionality influence the behaviour of or disrupt sensitive systems. Most attackers are looking for sensitive data that can be used to directly or indirectly profit from to include personally identifiable information and payment data. Often the data can be used for identity theft, fraudulent payments, or a variety of fraud schemes. | All mobile apps. | Apps that enable access to internal information or information about employees that may be leveraged in social engineering. | Apps that enable access to nonessential, but important intellectual property or trade secrets. | Apps that enable access to valuable intellectual property, trade secrets, or government secrets (e.g. in the United States this may be anything classified at Secret or above) that is critical to the survival or success of the organization. Applications controlling sensitive functionality (e.g. transit, manufacturing equipment, control systems) or that have the possibility of threatening safety of life. |
| Healthcare | Most attackers are looking for sensitive data that can be used to directly or indirectly profit from to include personally identifiable information and payment data. Often the data can be used for identity theft, fraudulent payments, or a variety of fraud schemes.For the US healthcare sector, the Health Insurance Portability and Accountability Act (HIPAA) Privacy, Security, Breach Notification Rules and Patient Safety Rule ( <http://www.hhs.gov/ocr/privacy/>= [.](http://www.hhs.gov/ocr/privacy/) | All mobile apps | Apps that enable access to small or moderate amounts of Personally Identifiable Information. | Apps that enable access to sensitive medical information (Protected Health Information), Personally Identifiable Information, or payment data. | Apps used to control medical equipment, devices, or records that may endanger human life. Payment and Point of Sale systems (POS) that contain large amounts of transaction data that could be used to commit fraud. |
| Retail, food, hospitality | Many of the attackers in this segment utilize opportunistic "smash and grab" tactics. However, there is also a regular threat of specific attacks on applications known to contain payment information, perform financial transactions, or store personally identifiable information. Although less likely than the threats mentioned above, there is also the possibility of more advanced threats attacking this industry segment to steal intellectual property, gain competitive intelligence, or gain an advantage with the target organization or a business partner in negotiations. | All mobile apps. | Suitable for apps that enable access to business applications, product catalogue information, internal corporate information, and applications with limited user information (e.g. contact information). | Apps that enable access to highly sensitive information or business applications. | Payment and Point of Sale systems (POS) that process large amounts of transaction data that could be used to commit fraud. Applications that enable access to a large volume of sensitive information like full credit card numbers, mother's maiden name, social security numbers etc. |

# Assessment and Certification

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

OWASP, as a vendor-neutral not-for-profit organisation, 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 organizations

The Application Security Verification Standard can be used as an open book verification of the application, including open and unfettered access to key resources such as architects and developers, project documentation, source code, authenticated access to test systems (including access to at least one account in each role), particularly for L2 and higher.

Historically, black-box testing and secure code reviews have included issues "by exception" – that is only failed issues appear in the final report. A certifying organization must include in any report the scope of the verification (particularly if a key component is out of scope, such as SSO authentication), 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 and can be really useful as proofs of the findings for the most doubts developers. 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 assurance evidence to demonstrate each and every verified requirement has indeed been tested.

### The role of automated security testing tools

The use of source code scanners and black-box testing tools is encouraged to increase efficiency whenever possible. It is not however not possible to complete MASVS verification using automated tools alone: Every mobile app is different, an understanding the overall architecture, business logic, and technical pitfalls of the specific technologies and frameworks 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 four levels, or by forking MASVS and changing what is required for each application 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 highly testable, with the sole exception of architectural requirements. By building unit and integration tests that test for specific requirements, the mobile app becomes nearly self-verifying with each and every build. For example, additional tests can be crafted for the test suite for (...) todo (...).

### 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 found in the MASVS, rather than the Top 10 negative things not to do.

# Detailed Verification Requirements

V1. Architecture, design and threat modelling

V2. Data storage and privacy

V3. Cryptography

V4. Authentication and session management

V5. Network communication

V6. Interaction with the environment

V7. Coding practices and build process

V8. Resiliency against reverse engineering

# V1: Architecture, design and threat modelling requirements

## Control objective

In a perfect world, security would be considered throughout all phases of development. In reality however, security if often only a consideration at a late stage of the SDLC. Besides the technical controls, the MASVS requires processes 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 app 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 high level requirements are as follows:

* At level 1, components of the application are identified and have a reason for being in the app
* At level 2 and higher, the architecture has been defined and the code adheres to the architecture. Additionally, a threat model has exists that identifies potential threats.

## Requirements

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| # | Description | 1 | 2 | 3 | 4 |
| **1.1** | Verify that all app components are identified and are known to be needed. | ✓ | ✓ | ✓ | ✓ |
| **1.2** | Verify that all third party components used by the app, such as libraries and frameworks, are identified, and tested for known vulnerabilities. | ✓ | ✓ | ✓ | ✓ |
| **1.3** | Verify that a high-level architecture for the mobile app and any connected remote services has been defined and that security has been addressed in that architecture. |  | ✓ | ✓ | ✓ |
| **1.4** | Verify that a threat model for the mobile app and any connected remote services has been produced to identify potential threats and countermeasures to work against them. |  | ✓ | ✓ | ✓ |
| **1.5** | Verify that all third party components are assessed and evaluated (associated risks) before being used or implemented. Whenever a security update is published of an implemented third party component, the change has to be inspected and the risk has to be evaluated. |  | ✓ | ✓ | ✓ |
| **1.6** | Verify that all security controls (including libraries that call external security services) have a centralized implementation. |  | ✓ | ✓ | ✓ |
| **1.7** | Verify that all application components are defined in terms of the business functions and/or security functions they provide. |  | ✓ | ✓ | ✓ |
| **1.8** | Verify that all components that are not part of the application but that the application relies on to operate are defined in terms of the functions, and/or security functions, they provide. |  | ✓ | ✓ | ✓ |
| **1.9** | If the app uses cryptography, verify that there is an explicit policy for how cryptographic keys are managed (e.g. generated, distributed, revoked, and expired), and verify that the key lifecycle is properly enforced. |  | ✓ | ✓ | ✓ |

## References

For more information, see also:

* OWASP Mobile Top 10: M10 - Extraneous Functionality
* OWASP: <https://www.owasp.org/index.php/Application_Security_Architecture_Cheat_Sheet>

# V2: Data Storage and Privacy requirements

## Control objective

The protection of sensitive data, such as user credentials and private informaion, 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.

## Requirements

Fortunately, 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 | 1 | 2 | 3 | 4 |
| **2.1** | Verify that system credential storage facilities are used appropriately to store sensitive data, such as user credentials or cryptographic keys. | ✓ | ✓ | ✓ | ✓ |
| **2.2** | Verify that no sensitive data is written to log files. | ✓ | ✓ | ✓ | ✓ |
| **2.3** | Verify that no sensitive data leaks to cloud storage. | ✓ | ✓ | ✓ | ✓ |
| **2.4** | Verify that no sensitive data is sent to third parties. | ✓ | ✓ | ✓ | ✓ |
| **2.5** | Verify that the keyboard cache is disabled on text inputs that process sensitive data. | ✓ | ✓ | ✓ | ✓ |
| **2.6** | Verify that the clipboard is deactivated on text fields that may contain sensitive data. | ✓ | ✓ | ✓ | ✓ |
| **2.7** | Verify that no sensitive data is exposed via IPC mechanisms. | ✓ | ✓ | ✓ | ✓ |
| **2.8** | Verify that sensitive data, such as passwords and credit card numbers, is not exposed through the user interface, and does not leak to screenshots. | ✓ | ✓ | ✓ | ✓ |
| **2.9** | Verify that sensitive data does not leak to backups. | ✓ | ✓ | ✓ | ✓ |
| **2.10** | Verify that the app removes sensitive data from views when backgrounded. |  | ✓ | ✓ | ✓ |
| **2.11** | Verify that the app does not hold sensitive data in memory longer than necessary, and that the memory is cleared explicitly after use. |  | ✓ | ✓ | ✓ |
| **2.12** | Verify that the app only runs on operating system versions that offer a hardware-backed keystore, and that the device supports the hardware-backed keystore. Alternatively, verify that encryption has been implemented according to the controls in MASVS V3. |  |  | ✓ | ✓ |
| **2.13** | If a remote locking mechansim exists, ensure that local storage is wiped upon locking. |  |  | ✓ | ✓ |

## References

For more information, see also:

* OWASP Mobile Top 10: M2 - Insecure Data Storage
* CWE: <https://cwe.mitre.org/data/definitions/922.html>

# V3: Cryptography Verification 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;
* Handling of errors, so that cryptographic modules fail in a secure manner;
* Suitable random number generator wherever randomness is required;
* Access management for cryptographic keys.

## Requirements

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| # | Description | 1 | 2 | 3 | 4 |
| **3.1** | Verify that that the application does not rely on symmetric cryptography with hardcoded keys as a sole method of encryption. | ✓ | ✓ | ✓ | ✓ |
| **3.2** | Verify that the cryptographic primitives used adhere to industry standards. Algorithms that are widely considered weak should not be used. | ✓ | ✓ | ✓ | ✓ |
| **3.3** | Verify that cryptographic modules operate using parameters that are considered secure (e.g. mode, key length). | ✓ | ✓ | ✓ | ✓ |
| **3.4** | Verify that the application doesn't re-use the same cryptographical key for multiple purposes. | ✓ | ✓ | ✓ | ✓ |
| **3.5** | Verify that all random numbers, random file names, random GUIDs, and random strings are generated using a secure random number generator. |  | ✓ | ✓ | ✓ |
| **3.6** | Verify that all keys and passwords are changeable, and are generated or replaced at installation time. |  | ✓ | ✓ | ✓ |
| **3.7** | Verify that random numbers are created with proper entropy during the application lifecycle. |  |  | ✓ | ✓ |
| **3.8** | Verify that cryptographic controls do not keep any reference to (or a copy of) a (working)key in memory. |  |  | ✓ | ✓ |
| **3.9** | Verify that consumers of cryptographic services do not have direct access to key material if the keying material is locally generated. Isolate cryptographic processes, including master secrets through the use of strong software protections or a hardware key vault (HSM). |  |  |  | ✓ |
| **3.10** | Verify that keying materials are refreshed per session and stored at the server for non-repudiation instead of on the device. |  |  |  | ✓ |
| **3.11** | Verify that keying materials have been applied as such that forward secrecy is provided. |  |  | ✓ | ✓ |

## References

For more information, see also:

* OWASP Mobile Top 10: M5 - Insufficient Cryptography
* CWE: <https://cwe.mitre.org/data/definitions/310.html>

# V4: Authentication and Session Management Requirements

## Control objective

In most cases, user login to 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 managed. The requirements can be easily verified without access to the source code of the service endpoint.

## Requirements

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| # | Description | 1 | 2 | 3 | 4 |
| **4.1** | If the app requires access to a remote service, verify that an acceptable form of authentication such as username/password authentication is performed at service endpoints. | ✓ | ✓ | ✓ | ✓ |
| **4.2** | Verify that a password policy exists and is enforced at the remote endpoint. | ✓ | ✓ | ✓ | ✓ |
| **4.3** | Verify that the remote service terminates an existing session when the user logs out. | ✓ | ✓ | ✓ | ✓ |
| **4.4** | Verify that sessions are terminated at the remote end after a predefined period of inactivity. | ✓ | ✓ | ✓ | ✓ |
| **4.5** | Verify that the remote service blocks login attempts in response to an excessive number of unsuccessful attempts. | ✓ | ✓ | ✓ | ✓ |
| **4.6** | Verify that a second factor of authentication exists at the remote endpoint, and that the 2FA requirement is enforced at the remote end. |  | ✓ | ✓ | ✓ |
| **4.7** | Verify that the remote service generate short-lived access token to authenticate client requests without sending the user's credentials. |  | ✓ | ✓ | ✓ |

## References

For more information, see also:

* OWASP Mobile Top 10: M4 - Insecure Authentication, M6 - Insecure Authorization
* CWE: <https://cwe.mitre.org/data/definitions/287.html>

# V5: Network communication requirements

## Control objective

The purpose of this control is to ensure the confidentiality and integrity of intormation 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 communcation using regular PKI infrastructure. For level two or higher, additional defense-in-depth measure such as SSL pinning are required.

## Requirements

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| # | Description | 1 | 2 | 3 | 4 |
| **5.1** | Verify that sensitive data is encrypted on the network using secure communication protocols. Verify that the secure channel is used consistently throughout the app. | ✓ | ✓ | ✓ | ✓ |
| **5.2** | Confirm that the app verifies the identity of the remote endpoint when the secure channel is established. Verify that this is done doing standard methods such as X.509 certificates, and that only certificates signed by a valid CA are accepted. | ✓ | ✓ | ✓ | ✓ |
| **5.3** | Confirm that communication of 3rd party libraries that send data to their own hosts (e.g. tracking libraries, crash reporter), is only established by using secure communication protocols. | ✓ | ✓ | ✓ | ✓ |
| **5.4** | Verify that the app either uses its own certificate store, or pins the endpoint certificate. |  | ✓ | ✓ | ✓ |
| **5.5** | Verify that the app doesn't rely on insecure communication channels (e.g email and SMS) for critical operations, such as enrollments and OTPs. |  | ✓ | ✓ | ✓ |
| **5.6** | Verify that the app pins the endpoint certificate, and subsequently does not establish connections with endpoints that offer a different certificate, even if signed by a trusted CA. |  |  | ✓ | ✓ |
| **5.7** | Confirm that the remote endpoint verifies the identity of the app when secure channel is established (PKI mutual authentication). |  |  | ✓ | ✓ |
| **5.8** | Verify that the app uses additional payload encryption in case of a broken SSL implementation on the device. |  |  |  | ✓ |

## References

For more information, see also:

* OWASP Mobile Top 10: M3 - Insecure Communication
* CWE: <https://cwe.mitre.org/data/definitions/319.html>
* CWE: <https://cwe.mitre.org/data/definitions/295.html>

# V6: Interaction with the environment

## Control objective

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

## Requirements

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| # | Description | 1 | 2 | 3 | 4 |
| **6.1** | Verify that that the app does not request any unnecessary permissions. | ✓ | ✓ | ✓ | ✓ |
| **6.2** | Verify that all inputs from external sources is validated. This includes data reveived via the GUI, IPC mechanisms such as intents, custom URLs, and network communication. | ✓ | ✓ | ✓ | ✓ |
| **6.3** | Verify that the app does not export sensitive functionality via custom URL schemes. | ✓ | ✓ | ✓ | ✓ |
| **6.4** | Verify that the app does not export sensitive functionality through IPC facilities. | ✓ | ✓ | ✓ | ✓ |
| **6.5** | Verify that Javascript is disabled in all WebViews unless explicitly required. | ✓ | ✓ | ✓ | ✓ |
| **6.6** | Verify that file access is disabled in all WebViews unless explicitly required. | ✓ | ✓ | ✓ | ✓ |
| **6.7** | If Javascript is required in a WebView, ensure that the WebView is restricted to a specific URL, and that no unfiltered user input is rendered in the WebView. | ✓ | ✓ | ✓ | ✓ |
| **6.8** | Verify that the app does not load user-supplied local resources into WebViews. | ✓ | ✓ | ✓ | ✓ |
| **6.9** | If Java objects are exposed in a WebView, verify that the WebView only renders JavaScript contained within the APK (Android). | ✓ | ✓ | ✓ | ✓ |
| **6.10** | Verify that the app leverages operating system features that allow updating of outdated system components. |  | ✓ | ✓ | ✓ |
| **6.11** | Verify that the app provides a custom keyboard whenever sensitive data is entered. |  |  | ✓ | ✓ |
| **6.12** | Verify that custom ui-components are used when sensitive data is displayed. The UI-component should not rely on immutable data structures. |  |  | ✓ | ✓ |

## References

For more information, see also:

* OWASP Mobile Top 10: 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.

## Requirements

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| # | Description | 1 | 2 | 3 | 4 |
| **7.1** | Verify that the application catches and handles possible exceptions. | ✓ | ✓ | ✓ | ✓ |
| **7.2** | Verify that all debugging code is removed from the release build, and that the app does log detailed error messages. | ✓ | ✓ | ✓ | ✓ |
| **7.3** | Verify that error handling logic in security controls denies access by default. | ✓ | ✓ | ✓ | ✓ |
| **7.4** | Do not concatenate untrusted external input into database queries or dynamically executed code. | ✓ | ✓ | ✓ | ✓ |
| **7.5** | If the app contains unmanaged code, verify that memory is allocated, freed and used securely. | ✓ | ✓ | ✓ | ✓ |
| **7.6** | Verify that the app is marked as a release build. | ✓ | ✓ | ✓ | ✓ |
| **7.7** | Verify that security features offered by the compiler, such as stack protection, PIE support and automatic reference counting, are activated. | ✓ | ✓ | ✓ | ✓ |
| **7.8** | Verify that static and dynamic application security testing are performed as part of the development lifecycle, and that the configuration of the SAST and DAST tools is tailored to the app. |  | ✓ | ✓ | ✓ |

## References

For more information, see also:

* OWASP Mobile Top 10: M7 - Client 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: Resiliency Against Reverse Engineering Requirements

## Control objective

This chapter covers defense-in-depth measures that are 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 resiliency against reverse engineering, making it more difficult for adversaries to gain an understanding of the app's internals or extract data from the app.

## Requirements

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| # | Verified | 1 | 2 | 3 | 4 |
| **8.1** | Verify that debugging symbols have been removed from native binaries. |  | ✓ | ✓ | ✓ |
| **8.2** | Verify that any Java bytecode has been obscured through identifier renaming. |  | ✓ | ✓ | ✓ |
| **8.3** | Verify that the application detects whether it is being executed on a rooted or jailbroken device. Depending on business requirement, users should be warned, or the app should terminate if the device is rooted. |  | ✓ | ✓ | ✓ |
| **8.4** | Verify that the app has been installed from a legitimate source. |  | ✓ | ✓ | ✓ |
| **8.5** | Verify that the app has some form of debugger detection and terminates when a debugger is detected, or that the app prevents attaching a debugger using any method. All available means of debugging must be covered (e.g. JDWP and native). |  | ✓ | ✓ | ✓ |
| **8.6** | Verify that the app implements two or more methods of root detection and responds to the presence of a rooted device. |  |  | ✓ | ✓ |
| **8.7** | Verify that the app either prevents, or detects and responds to, the presence of debuggers using at least two additional, functionally different methods that don't rely on library calls or well-known high-level APIs. |  |  | ✓ | ✓ |
| **8.8** | Verify that the app detects and responds to file tampering. |  |  | ✓ | ✓ |
| **8.9** | Verify that the app detects the presence of common reverse engineering tools, such as hooking frameworks and debugging servers. |  |  | ✓ | ✓ |
| **8.10** | Verify that the app detects whether it is run inside an emulator using any method, and terminates when an emulator is detected. |  |  | ✓ | ✓ |
| **8.11** | Verify that that the app implements multiple different responses to tampering, debugging and emulation, including stealthy responses that don't simply terminate the app. |  |  | ✓ | ✓ |
| **8.12** | Verify that all executable files and libraries belonging to the app are either encrypted on the file level and/or code and data segments inside the executables are encrypted or packed. Trivial static analysis should not reveal important code or data. |  |  | ✓ | ✓ |
| **8.13** | Verify that the application implements a 'device binding' functionality when a mobile device is treated as being trusted. Verify that the device fingerprint is derived from multiple device properties. |  |  | ✓ | ✓ |
| **8.14** | Verify that all executable files and libraries belonging to the app are either encrypted on the file level and/or code and data segments inside the executables are encrypted or packed. Trivial static analysis should not reveal important code or data. |  |  | ✓ | ✓ |
| **8.15** | Verify that obfuscating transformations and reactive defenses are interdependent and well-integrated throughout the app. |  |  | ✓ | ✓ |
| **8.16** | Verify that the app uses at least two additional, functionally different checks to detect whether it is running in an emulator, and responds appropriately when an emulator is detected. |  |  |  | ✓ |
| **8.17** | Verify that sensitive computations are obfuscated using advanced methods, and that the obfuscating transformations significantly increase the algorithmic complexity and apparent randomness of the relevant code and/or data. |  |  |  | ✓ |

## References

For more information, see also:

* OWASP Mobile Top 10: M8 - Code Tampering, M9 - Reverse Engineering

# Appendix A: Glossary

(.. todo ...)

* **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.
* **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:

(...) TODO (...)  
<https://www.owasp.org/index.php/Projects/OWASP_Mobile_Security_Project_-_Top_Ten_Mobile_Risks>

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>

# Appendix C: Standards Mappings

(...) TODO. In the ASVS, we have a mapping to PCI-DSS 3.0. Update this to match the MASVS.. (...)

|  |  |  |
| --- | --- | --- |
| PCI-DSS 3.0 | MASVS 1.0 | Description |
| 6.5.2 Buffer overflows | TODO | Exact mapping |
| 6.5.3 Insecure cryptographic storage | TODO | TODO |
| 6.5.4 Insecure communications | TODO | TODO |
| 6.5.5 Improper error handling | TODO | Exact mapping |
| 6.5.8 Improper Access Control (such as insecure direct object references, failure to restrict URL access, directory traversal and failure to restrict user access to functions). | TODO | TODO |
| 6.5.10 Broken authentication and session management. | TODO | TODO |