The OWASP Mobile Top 10 Security Risks and Mobile Application Security Verification Standard

What you need to know to address mobile app risks

An overview of how RASP and code hardening technologies can work together to fortify your mobile application security posture

GUARDSQUARE
Mobile application protection

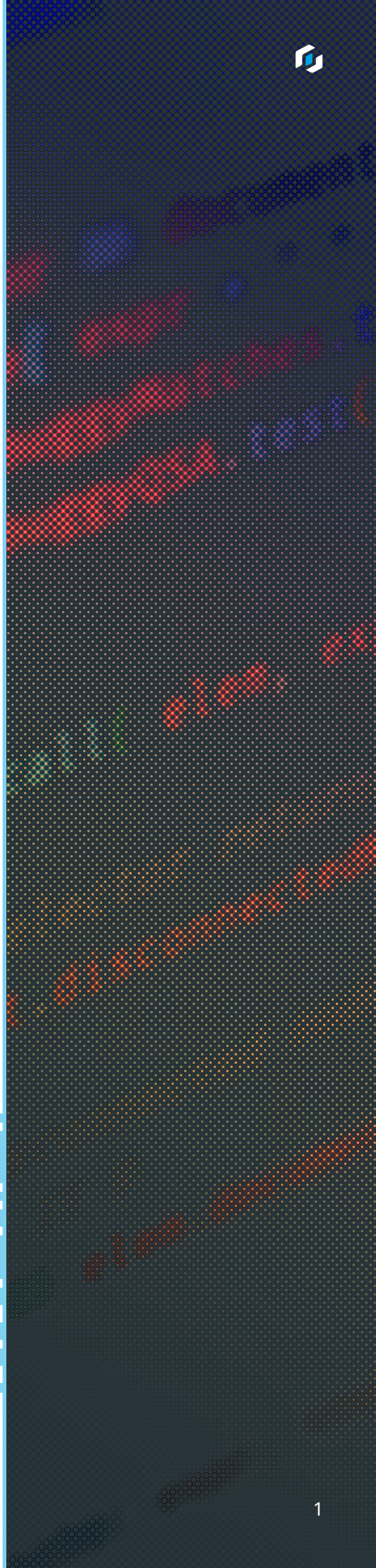
The OWASP Mobile Top 10 Security Risks and Mobile Application Security Verification Standard

Table of contents

Introduction

Mobile applications are a rapidly growing attack surface. The tools and techniques being used to compromise these environments are constantly evolving. **Static attacks** involve trying to revert downloaded machine-readable code into human readable code, which can then be analyzed or searched to understand the functioning of the application, search for sensitive data, identify vulnerabilities that can be exploited, etc. **Dynamic attacks** provide adversaries the ability to enumerate more, and exploit applications *faster*.

Often, the effect of these exploits can have a significant, negative impact on affected organizations – as well as their users.



Why **OWASP**

The OWASP Mobile Security Project & why it matters

The OWASP Mobile Security Project is part of a larger, global, security-based community, called OWASP, or the Open Web Application Security Project. OWASP is best known for publicizing reliable content about emerging vulnerabilities in the web application space. The OWASP Mobile Security Project, MSP for short, focuses specifically on formalizing security requirements and best practices within the mobile application domain.

The internet is saturated with conflicting information making it challenging, even for professionals, to find reliable information. The OWASP Mobile Security Project, specifically the *Mobile Top 10* and "MASVS" which stands for the Mobile Application Security Verification Standard, provide a reliable framework that acts as vital resources when identifying and remediating mobile application vulnerabilities.

This report provides an overview of how a combination of code hardening and runtime application self-protection (RASP) can improve mobile application security by mapping to the OWASP *Mobile Top 10*.

It also briefly explains the significance of OWASP as a community, the OWASP Mobile Security Project and the OWASP *Mobile Top 10*. Lastly, this report provides context into how RASP and code hardening remediate each of the vulnerabilities to which they map, in accordance with OWASP documentation, to enforce industry best practices and to fortify the security postures of mobile applications and SDKs.

The OWASP Mobile Top 10: An app security checklist

The OWASP *Mobile Top 10* is a succinct list of the most commonly found security risks in the mobile application environment. These risks can arise from a lack of understanding or difficulty implementing security best practices which can open the door for vulnerabilities. These vulnerabilities often carry a significant business impact in the form of financial or reputational losses when exploited.

> **In short, the OWASP *Mobile Top 10* serves as an excellent blueprint for developers who want to prioritize security within their mobile applications but are unsure of where to begin.**

To view a full copy of the OWASP *Mobile Top 10*, click here >

OWASP Mobile Application Security Verification Standard (MASVS)

The ***Mobile Top 10*** can be thought of as "the tip of the iceberg." It's a list of the top 10 categories of security risks that are recommended priorities when approaching mobile application security.

The **Mobile Application Security Verification Standard**, or MASVS, can be thought of as the rest of the iceberg. The **MASVS** is a more complete list of security risks found in the mobile application space that don't fall into the "Mobile Top 10." The MASVS highlights potential gaps in mobile app security and provides dependable mitigations for the risks they bring about.

MASVS security verification levels

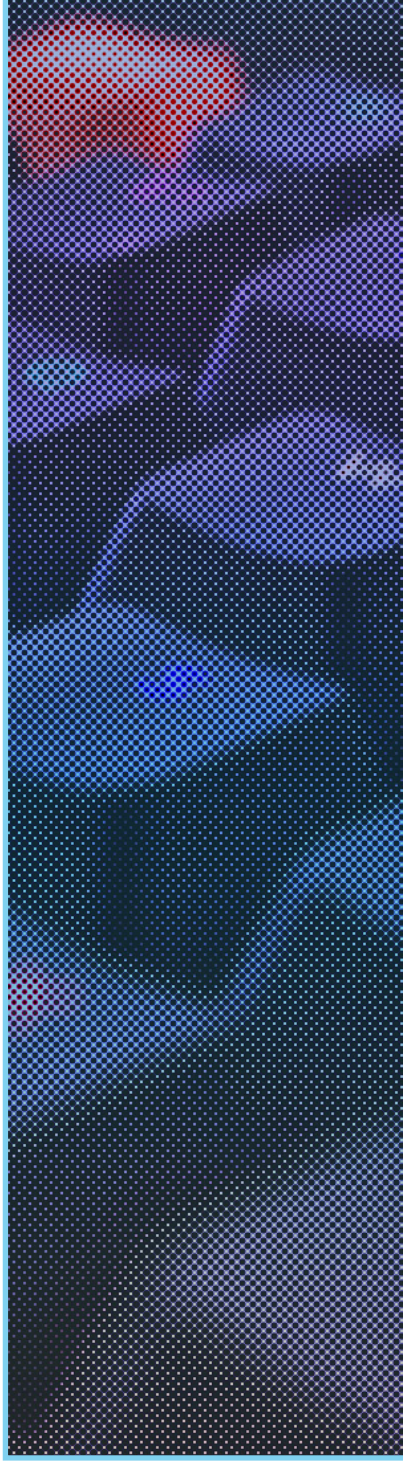
The **MASVS** defines two levels of security verification, MASVS-L1 and MASVS-L2. It also provides resilience guidelines (MASVS-R) that are meant to provide further protections against reverse engineering and code tampering (M8, M9).

MASVS-L1 contains general security requirements that are recommended for all mobile apps. 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 a more resilient app that is fortified against sophisticated attacks. MASVS-L2 is recommended for all apps handling highly sensitive data.

MASVS-R, also called the Resilience layer, encompasses very strict controls that are designed to fortify the security posture and prevent reverse engineering and code tampering of mobile apps.

RASP and code hardening are extremely effective solutions when applying the resilience layer to mobile apps. They specifically map to certain risks outlined in the Mobile Top 10 (M8 and M9), as explained below.



The resilience layer in depth

The resilience layer covers defense-in-depth measures that are recommended for mobile apps that process, or have access to, sensitive data or functionality. Not having these controls does not inherently cause a vulnerability. However, having these controls in place will increase the app's resilience against reverse engineering and specific client-side attacks.

These controls 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. **The more resilience added, the more secure the app will be against reverse engineering and code tampering.**

A layered combination of runtime application self-protection (RASP) and code hardening can fortify the security posture of mobile apps by shoring up the app's resilience layer.

OWASP suggests taking counsel from the OWASP document, "Technical Risks of Reverse Engineering and Unauthorized Code Modification Reverse Engineering and Code Modification Prevention," to find out more about the specific implementations and resulting benefits provided by adding additional resilience to mobile applications and SDKs.

Summarizing the value of OWASP's approach to security for mobile apps

The **MASVS** is a standard for mobile application security. It outlines the security requirements of a mobile application and provides test cases to ensure your efforts are sufficient. It also provides levels of verification to identify the respective level of a mobile application's security.

Similar to starting with an initial operating capability (IOC) and progressing toward a full operating capability (FOC), this approach makes implementing application security a more manageable goal.

While it does not always mention specific vulnerabilities, it provides a clear reference and checklists to help standardize mobile application security through the implementation of secure coding practices.

To view a full copy of the Mobile Application Security Verification Standard (MASVS), click here >

What follows in this report is an explanation of how mobile application security testing, as well as RASP and code hardening, map directly to the OWASP *Mobile Top 10* and the MASVS.

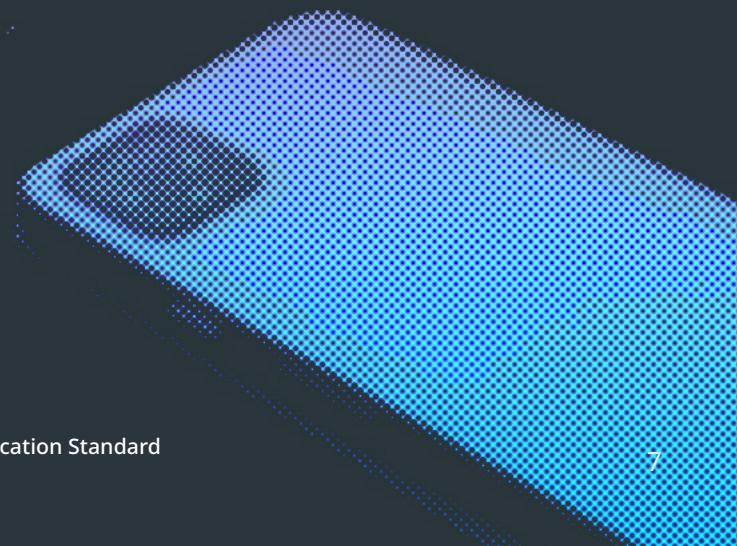
Table 1, seen on the following page, is the 2016 (most recent) publication of the OWASP *Mobile Top 10*, along with a high-level description of each vulnerability encompassed within the document.

For more information and to learn more about the OWASP *Mobile Top 10*, click here >

OWASP Top 10 - Mobile Applications

VULNERABILITY	DESCRIPTION OF IMPACT
M1 Improper Platform Usage	This category refers to the misuse of a platform feature or failure to use platform security controls. The specific ways this manifests depend on the operating system, but may include, "Android intents, platform permissions, misuse of TouchID, the Keychain, or some other security control that is part of the mobile operating system." There are several ways that mobile apps can become vulnerable to this risk.
M2 Insecure Data Storage	This covers insecure data storage and unintended data leakage, which are very common vulnerabilities in mobile applications. This happens when dev teams assume users or malware can't access a mobile device's files in spite of them being easy to access. Data should always be stored securely.
M3 Insecure Communication	Mobile apps typically exchange data with servers in a typical client-server relationship. Insecure communication arises when data being transmitted by a mobile app can be intercepted. An analysis of the wireless network configuration is necessary to ensure the appropriate access, logging, encryption, and other controls are in place.
M4 Insecure Authentication	This category captures notions of not authenticating the end user (failing to identify and/or maintain identification) or bad session management.
M5 Insufficient Cryptography	Mobile app code should apply cryptography to any sensitive information asset. However, M5 arises when the cryptography is insufficient in some way. (Note that anything and everything related to TLS or SSL goes in M3. Also, if the app fails to use cryptography at all when it should, that belongs in M2.) This category is for issues where cryptography was attempted, but done incorrectly.
M6 Insecure Authorization	This is a category to capture any failures in authorization (e.g., authorization decisions on the client side, forced browsing). It is distinct from authentication issues (e.g., device enrollment, user identification).

M7 **Client Code Quality**	This is the catch-all for code-level implementation problems in the mobile client. It is distinct from server-side coding mistakes. This would capture things like buffer overflows, format string vulnerabilities, and various other code-level mistakes where the solution is to rewrite some code that's running on the mobile device.
M8 **Code Tampering**	Once the application is delivered to the mobile device, the code and data resources are resident there. The code tampering category covers binary patching, local resource modification, method hooking, method swizzling, and dynamic memory modification. An attacker can either directly modify the code, change the contents of memory dynamically, change or replace the system APIs that the application uses, or modify the application's data and resources. This can provide a direct method of subverting the intended use of the software for personal or monetary gain.
M9 **Reverse Engineering**	This category includes analysis of the final core binary to determine its source code, libraries, algorithms, and other assets. Easily attained binary inspection tools give the attacker insight into the inner workings of the application. This may be used to exploit other nascent vulnerabilities in the application, as well as revealing information about back-end servers, cryptographic constants and ciphers, and intellectual property.
M10 **Extraneous Functionality**	Often, developers include hidden backdoor functionality or other internal development security controls that are not intended to be released into a production environment. For example, a developer may accidentally include a password as a comment in a hybrid app. Another example includes disablement of multi-factor authentication during testing.



Layered solutions to the security risks introduced by OWASP Mobile Top 10

Mobile application security testing

Mobile Application Security Testing is the first layer of defense, which empowers teams to assess the security risk of their mobile applications early in the development process, prior to being released. Developers can scan their apps, with each build, to identify potential vulnerabilities early, and not just with release candidates.

A well-integrated and developer-friendly security testing solution is one that automates the security testing as part of the build process. Ideally, the right tool is one that focuses on the relevant application security domain (e.g. Mobile) to ensure accuracy and relevance of findings. Learn more >

Runtime Application Self-Protection (RASP)

RASP is a security solution that makes use of runtime instrumentation to provide advanced detection of real-time indicators of threat and compromise. It does this by taking advantage of information from inside the running application.

RASP can provide detection for jailbreaking, rooting, hooking and code tampering, in addition to obstruction for debugger and emulator attachments. Some RASP providers are much better than others, so it is very important to vet vendors carefully.

RASP provides advanced detection to stop attackers in their tracks. The goal is to increase the amount of time and effort it takes for bad actors to reverse engineer applications. Additionally, it should be the goal of any development team, security team or security provider to reduce the amount of context available to attackers, thus reducing their ability to stage sophisticated attacks.

With sophisticated protection mechanisms in place, most adversaries will simply move onto other, less secure applications. Their goal is to liquidate profit as quickly and easily as possible.

Code hardening

Code hardening is the process of reducing readability and introducing entropy within code, such that attackers are unable to nefariously reverse engineer and make sense of it. Proper code hardening protects mobile applications and SDKs for Android and iOS from code tampering (M8) and reverse engineering (M9) by deploying robust, layered obfuscation and encryption. These techniques transform the representation of code, making it both illegible and inaccessible to threat actors. Hardened code is protected against both automated and manual analysis.

Obfuscation is the process of making code more difficult to read. It is accomplished by:

- Stripping out potentially sensitive context or descriptive metadata
- Replacing plain-text identifiers with meaningless ones
- Complicating simple instructions to reduce ROI potential by increasing the investment of time and effort required for attackers
- Adding meaningless code and equivalently shuffling logic to mislead attackers

Examples of obfuscation types include (but are not restricted to):

- **Name obfuscation**
  - Replaces sensitive identifiers with meaningless ones
- **Control flow obfuscation**
  - Equivalently shuffles program logic, and flattens control flow
- **Arithmetic obfuscation**
  - Replaces simple arithmetic constructs with very complex mathematical equivalents

Encryption is the process of mathematically transforming plain-text information into secret code, known as cipher-text. In simpler terms, it is the process of reducing readability and predictability in code, by replacing sensitive context with a mathematically encrypted representation. Encryption is used to protect code and the confidentiality of classified information.

Examples of encryption types for mobile security include:

- **String encryption**
- **Asset / resource encryption**
- **Class encryption**

As with many aspects of security, a layered approach is the best way to make applications secure and increase the effort required to hack or misuse them.

Mapping **mobile application security testing** to the Mobile Top 10

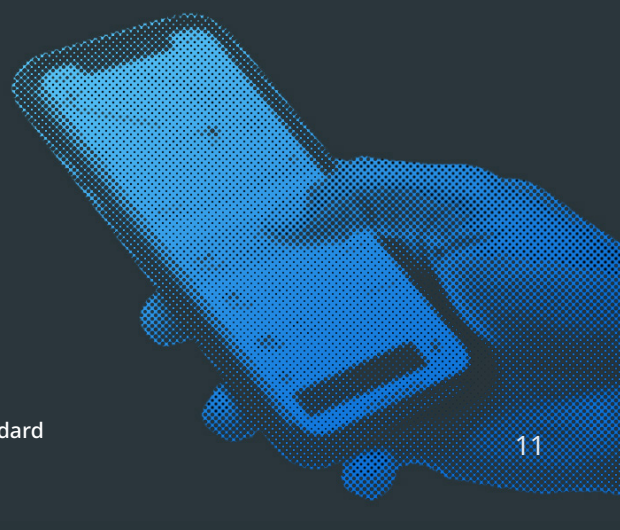
The table provided below maps Guardsquare's mobile application security testing solution, **AppSweep**, to the *Mobile Top 10* by showing which security risks the software can help identify. The security checks of AppSweep are continuously being improved with new checks being regularly added. A description of how the software maps to each security risk found in the OWASP *Mobile Top 10* is provided.

VULNERABILITY	SOLUTION
M3 **Insecure communication**	AppSweep by Guardsquare can identify various forms of insecure communication. This includes hardcoded HTTP URLs, insecure loading of JavaScript code, hardcoded authorization credentials and hardcoded S3 buckets
M4 **Insecure authentication**	AppSweep can identify the use of hardcoded keys, which is a form of insecure authentication.
M5 **Insufficient cryptography**	AppSweep by Guardsquare searches applications to uncover outdated and improper use of cryptography, identifying scenarios such as usage of insecure ciphers, ciphers with improper defaults, legacy cryptography usage, insecure cryptography, insecure seeding and insecure pseudo-random number generators.
M7 **Client code quality**	AppSweep incorporates client code quality checks, including identification of elements not protected from tapjacking.
M8 **Code tampering**	AppSweep can detect when a release build has a debuggable flag set and when the application lacks sufficient anti-tampering protections.
M9 **Reverse engineering**	AppSweep by Guardsquare can detect builds that include hardcoded HTTPS URLs and other sensitive strings or resources that lack sufficient code hardening protection.

Mapping RASP to the Mobile Top 10

The tables provided below map RASP solutions to the *Mobile Top 10* by showing which security risks the software can, and cannot, help defend against. A description of how the software maps to each security risk found in the OWASP *Mobile Top 10* is provided.

VULNERABILITY	DESCRIPTION OF IMPACT
M8 **Code tampering**	DexGuard and iXGuard integrate a full suite of configurable integrity checks to protect applications against attempts to modify their intended behavior and alter resources and data. The combined security provided by jailbreak/root detection, hook detection and method swizzling prevention (iOS) ensures app behavior is not modified at runtime. Tamper detection, repackaging detection and certificate checks enable apps and SDKs to detect unauthorized code modifications and to verify the integrity of individual files. Polymorphic protection ensures that mobile app security defenses change with each build, resetting the clock on attackers
M9 **Reverse engineering**	The runtime self protection mechanisms provided by DexGuard and iXGuard make dynamic analysis of applications infeasible, preventing unauthorized parties from gaining insight into their inner workings. DexGuard and iXGuard's debugger detection prevent third parties from using debugging tools to analyze specific operations by executing and halting the program at specific points. Their emulator checks make sure applications or SDKs are not being executed in an emulator.



Mapping code hardening to the Mobile Top 10

Tables 2 & 3 below map code hardening solutions to the *Mobile Top 10* by showing which security risks the solutions can help defend against. A description of how the software maps to each security risk found in the OWASP *Mobile Top 10* is provided.

Specifically, this table reviews how Guardsquare's iOS mobile app security solution, **iXGuard**, and Android mobile app security solution, **DexGuard**, protect against two key and difficult-to-address vulnerabilities within the MTT.

VULNERABILITY	SOLUTION
M8 **Code tampering**	DexGuard and iXGuard's layered code hardening effectively prevents third parties from gaining access to the source code of Android and iOS applications and modifying it. The applied hardening techniques also ensure the implemented RASP mechanisms (see below) are not removed or tampered with so that they can fully protect applications against attempts to modify their behavior at runtime.
M9 **Reverse engineering**	The multiple code hardening techniques DexGuard and iXGuard deploy make it virtually impossible to reverse engineer applications and gain insight into their internal logic. The combined protection offered by techniques such as name obfuscation, string encryption, control flow obfuscation and code virtualization effectively prevent unauthorized parties from finding and exploiting vulnerabilities in the code, stealing IP or extracting sensitive information like API keys or cryptographic constants and ciphers.

Mapping **mobile app security solutions** to MASVS

What follows is a list of recommended security and resiliency measures for mobile applications, as described in the OWASP MASVS. Below, we explain how various Guardsquare products map directly to these MSTG (Mobile Security Testing Guide) recommendations.

Impede dynamic analysis and tampering

#	MSTG-ID	DESCRIPTION	COMPLIANCE
8.1	MSTG-RESILIENCE-1	The app detects, and responds to, the presence of a rooted or jailbroken device either by alerting the user or terminating the app.	Guardsquare products implement root and jailbreak checks that can terminate the app or allow the app developer to define another behavior, such as alerting the user.
8.2	MSTG-RESILIENCE-2	The app prevents debugging and/or detects, and responds to, a debugger being attached. All available debugging protocols must be covered.	Guardsquare products implement debugger detection and prevention covering all debugging protocols.
8.3	MSTG-RESILIENCE-3	The app detects, and responds to, tampering with executable files and critical data within its own sandbox.	Guardsquare products implement tampering detection and allow the developer to terminate the app or define another response.
8.4	MSTG-RESILIENCE-4	The app detects, and responds to, the presence of widely used reverse engineering tools and frameworks on the device.	Guardsquare products implement detection of widely used reverse engineering tools and allow the developer to terminate the app or define another response.

8.5	MSTG-RESILIENCE-5	The app detects, and responds to, being run in an emulator.	Guardsquare's DexGuard implements emulator detection for Android and allows the developer to terminate the app or define another response. For iOS, emulation is possible on ARM Macs. iXGuard implements ARM Mac detection in the scope of Jailbreak detection and allows the developer to terminate the app or define another response.
8.6	MSTG-RESILIENCE-6	The app detects, and responds to, tampering the code and data in its own memory space.	Guardsquare products implement code tampering detection in memory and allow the developer to terminate the app or define another response.
8.7	MSTG-RESILIENCE-7	The app implements multiple mechanisms in each defense category (8.1 to 8.6). Note that resiliency scales with the amount and diversity of the originality of the mechanisms used.	Guardsquare products allow developers to automatically generate and inject a large amount of diverse checks, with a polymorphic approach that ensures every app's build comes with a unique combination of locations and exact checks.
8.8	MSTG-RESILIENCE-8	The detection mechanisms trigger responses of different types, including delayed and stealthy responses.	Guardsquare products allow developers to implement their own response strategies, which can include delayed and stealthy responses. The standard response strategy that terminates the application does so in a way that does not make the reason for termination immediately obvious.
8.9	MSTG-RESILIENCE-9	Obfuscation is applied to programmatic defenses, which in turn impede de-obfuscation via dynamic analysis.	Guardsquare products implement a layered hardening approach, which applies obfuscation on programmatic defenses.

Impede comprehension

#	MSTG-ID	DESCRIPTION	COMPLIANCE
8.11	MSTG-RESILIENCE-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.	Guardsquare products implement code and data obfuscation, which makes it impossible for trivial static analysis to reveal important code or data.
8.12	MSTG-RESILIENCE-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.	Guardsquare products implement obfuscation methods that are resilient against manual and automated attacks. Guardsquare has scientific and security penetration testing personnel who are responsible for continuous maintenance and improvement of the obfuscation methods' effectiveness.

Applying **code hardening, RASP, and frequent mobile application security testing** to your mobile apps

Many teams think they have what it takes to implement detection for dynamic attacks because resources exist online. However, most teams are not prepared to DIY runtime application self-protection, and few are able to seamlessly integrate code hardening into the software development lifecycle without support. That's perfectly normal and understandable, especially given the security talent shortage.

Ideally, teams should invest in a layered mobile application security solution that offers both code hardening and RASP. This is the best way to defend apps against some of the most difficult to address and potentially damaging top 10 risks outlined in the *Mobile Top 10*. Moreover, a strong combination of code hardening and RASP helps secure mobile apps by implementing the resilience layer, as outlined in OWASP MASVS.

Beyond this, best practices, specifically those offered within the OWASP MASVS documentation, are recommended throughout all phases of the SDLC in order to ensure a well-secured environment. The rest of the OWASP *Mobile Top 10* vulnerabilities can largely be mitigated by writing more secure applications and ensuring that security best practices are followed when it comes to implementing platform-specific guidelines, avoiding extraneous and potentially revealing code, and storing and transmitting data securely.

The approach described in this report will prevent attackers from using their favorite reverse engineering tools to easily decompile and hack apps.

In summary, an effective mobile security strategy requires a comprehensive approach, which includes automated security testing of your mobile application, early and often through the development lifecycle. Reverse engineering and tampering, two particularly difficult-to-address risks outlined in the OWASP *Mobile Top 10*, requires a high-quality layered solution for code hardening and runtime protection. By implementing these practices, organizations will benefit from a strong security posture that aligns with the current best practices in mobile app security.

About **Guardsquare**

Guardsquare offers the most complete approach to mobile application security on the market. Built on the open source ProGuard technology, Guardsquare's software integrates seamlessly across the development cycle. From app security testing to code hardening to real-time visibility into the threat landscape, Guardsquare solutions provide enhanced mobile application security from early in the development process through publication.

More than 700 customers worldwide across all major industries rely on Guardsquare to help them identify security risks and protect their mobile applications against reverse engineering and tampering.

Content contributor: Jack Butler, Solutions Engineer, Guardsquare

The creators of **ProGuard**

GUARDSQUARE
Mobile application protection