

Mobile App Protection: Ten Things You Must Consider

Once any mobile app goes public so does its compiled code. This code can be analyzed and manipulated by bad-actors who are able to run it in an environment of their choosing. There are a wide range of very sophisticated tools on both Android and iOS which assist with this process. Various technologies and tools are available to improve your app's protection. There are many players in a somewhat crowded market, with approaches that overlap in some cases.



Here is a quick guide to some of the terms you might come across:

- **Obfuscation:** Build time process that transforms the generated app code that renders its reverse engineered form very difficult to understand. At minimum this involves ensuring that human-readable internal symbols are removed.
- **Anti-Tampering:** Build or run time techniques to ensure that the app code cannot be modified.
- **App Hardening:** Typically refers to a more extensive form of obfuscation that transforms the control flow of the code making it extremely hard to comprehend and in some cases ensuring that the at-rest form of the code is encrypted. Sophisticated hardening approaches also perform continual distributed runtime checks to ensure there has been no code tampering.
- **App Shielding:** Addition of various runtime checks into the app code to verify the integrity of the app and the device it is running upon, generally causing the app to stop or crash if a problem is detected. Some products combine runtime app shielding and build time app hardening.
- **No-Code Shielding/Wrapping:** A shielding approach that applies the protection as a wrapper to a completed app as an additional step before publishing it.
- **In-App Protection:** The general class of approaches that combine hardening and shielding to strengthen the security of mobile apps.
- **Runtime Application Self Protection (RASP):** The general class of runtime instrumentation approaches that allows an app to detect and respond to a threat.
- **White Box Cryptography:** Approach for embedding secret keys and cryptographic implementations within the compiled code of an app to make direct

reverse engineering very difficult. This may be combined with other protections to ensure the whole whitebox module cannot be extracted from an app and manipulated.

- **App Attestation:** An approach that measures the integrity of the app and usually the environment that it is running in and communicates this externally, generally via a cryptographically signed token. This expands the protection beyond the boundary of the mobile app itself, allowing it to prove its veracity to backend services. dynamic pinning functionality.

The set of choices available is confusing, and you should carefully consider the different options. In some cases it might make sense to use multiple tools and approaches, perhaps mixing open source tools with commercial offerings.

In regulated industries, minimum compliance requirements might suggest that only code obfuscation and root/jailbreak detection are needed. But the threat landscape has evolved considerably in recent years and you should be looking to introduce a flexible and future-proof approach that reaches beyond such minimum requirements.

Always remember that there are multiple attack surfaces in the system as a whole which are not limited to the app itself: the client device, the operating system, APIs and communication channels come into play. A comprehensive security solution must address all of these. Your security is only as good as its weakest link.

Here are 10 requirements you must consider before making a decision about an app protection solution:

1. **Is the app code the asset you really want to protect?** App hardening and obfuscation is applied for the purposes of making reverse engineering and modification of the code very difficult. But you must consider if it is the intellectual property embedded in your app code that you are actually trying to protect. If your concern is for the security of the system as a whole then it is likely that this isn't fully addressed via app hardening. Moreover, such protection might only weed out amateur attackers. It may require serious effort, skill, resources, and time to crack hardened code - but it doesn't stop serious criminals.
2. **Does it protect your APIs from abuse?** Protecting the app does not prevent the abuse of the APIs that the app is reliant upon. In many cases it is the potential for those APIs to be abused by spoofed requests that represents the real risk to the security of your business. Even if the app has been hardened or shielded, it may be possible to reverse engineer the API protocol and extract any secrets embedded in the app (such as API keys) that allow requests to be spoofed from an attacker's script. Once those credentials have been extracted, the protection inside the app becomes irrelevant.
3. **Does it stop Man-in-the-Middle (MitM) attacks?** Attacking the API channel between mobile apps and their backend servers, via Man-in-the-Middle (MitM) attack, represents a growing threat for mobile users. MitM attacks allow the interception of mobile communications to gain access to sensitive information. It is common for attackers to use this approach to steal the secrets that allow them to directly attack backend APIs, sidestepping the protections built into the app itself.
4. **Can it protect against the latest new and emerging threats?** Mobile app protection is a continual battleground between attackers and app defense solutions. New tools and techniques are evolving all the time. The solution needs to be able to perform a deep inspection on the app to ensure it is genuine. But beyond that the device on which the mobile app is running may be rooted or jailbroken, and this is a common technique used by attackers (and pentesters) to bypass security mechanisms and limitations imposed by the original version of the OS. Another form of code tampering is to inject code at runtime by using an instrumentation framework. These are used to hook into the key functions which, when manipulated, will produce different app behavior than expected or will change input parameters or output results. Your protection solution needs to detect all these and more as the threats evolve.
5. **Is there fine grain control of the threats to block?** As well as being able to detect the threats, you will need fine grain control of which of those threats you might want to block. These decisions might not be clear cut. For instance, you might find that a legitimate proportion of your user base use rooted devices. You need to consider if your solution is able to accept rooted devices, but reject other associated environment detections that are likely to be nefarious. Consider if it is possible to change the threat posture dynamically, or if you have to reissue your apps through the store to change policy. This can result in many unhappy would-be customers in the meantime.

6. **Does it work with my app technology?** App protection needs to be deeply integrated into the runtime and, for hardening approaches, into the build time of your app. Different tools and technologies may only be able to support a subset of the various different app development technologies that are available. You need to consider if your choices are truly compatible and if there is good evidence that the app protection vendor can keep pace with the progression of the app development platform itself. You don't want your developers to be constrained by the compatibility of your app protection choices.
7. **Is there an impact on app performance?** Since code obfuscation brings about deep changes in the code structure, it may result in a drop in performance, or increase in size, of your app. The deeper and more sophisticated the hardening techniques, the greater this impact. Expect a heated discussion between developers and security!
8. **Does it impact your build and support processes?** It is common to add SDKs into apps to support particular capabilities and features. Once those SDKs are included they shouldn't cause any additional drag on your development processes. However, app hardening tools may substantially impact the build and test processes of an app since all of the compiled code needs to be processed. If this is only applied to production release versions then this creates a certain amount of risk, since the hardening process might introduce bugs or performance issues that are not caught during the normal development test process. Moreover, field crash reports from hardened apps may be significantly more demanding to interpret since code locations must be expressed in the obfuscated and intentionally hard to follow protected code. This can negatively impact your ability to support your app.
9. **Can you see what is happening in your apps?** If you've applied protection into your app it is hard to know if it is working, or even to be sure that it is not causing unintended blocks to legitimate users. You need to consider if the solution can collect live metrics from running devices so you measure its effectiveness. Can you see live threats emerging and being

repelled in your user base?

10. **How much is it going to cost?** Effective app protection will have a cost associated with it, and you need to think about this holistically. It is a necessary part of protecting your business against the range of threats that exist. Under investment in security can easily result in significant reputational damage and lost business. A key question you need to ask is how well will the costs scale with the success of your business. A SaaS based model is ideal as costs will scale directly with your success as you grow your app user base.

Summary

Traditional app hardening makes it hard to decompile, comprehend and manipulate the code of your mobile apps, and there are many solutions to choose from. The shortcomings of using hardening alone can be summed up in one word: runtime. They do not protect you from runtime threats. This is why the traditional app hardening vendors are scrambling to inject some run-time protection and monitoring into their product portfolio.

But even app runtime shielding approaches have a significant limitation - they act in isolation. If app runtime shielding detects a problem it will shut down the app. However, if the behavior of the app and its embedded credentials can be reverse engineered from MitM analysis then an attacker can build a script that acts just like the real app, free from any such protection. If your goal is to protect the system as a whole, including the backend APIs and services, then you need to prevent this by ensuring that only valid apps can authenticate themselves by using an attestation approach.

Our advice is to think about a comprehensive approach to the security of your apps and APIs and in that spirit consider carefully how well solutions address the ten items above. Consult external resources such as the [OWASP Mobile Security Project](#), which also covers the security aspects of the implementation of your code as well.

As part of your assessment process, you should evaluate [www.approov.io](#) which has been designed from the start to protect all of the relevant attack surfaces for all round protection.