Avoid Mobile Application Security Pitfalls

Foundational
Refreshed 27 January 2022, Published 27 July 2020 - ID G00730988 - 20 min read
By Dionisio Zumerle

Mobile applications are increasingly sources of fraud and breaches for organizations. Security and risk management leaders must follow mobile best practices to avoid data leakage from mobile devices and attacks to infrastructure.

Additional Perspectives

- Summary Translation: Avoid Mobile Application Security Pitfalls(24 August 2020)

Overview

Key Challenges

- Mobile application security failures lead enterprises to sensitive data loss, exposure of infrastructure, fraud and noncompliance.
- The architectural decisions made very early on in the process will determine many of the limitations in the security functionality available to security leaders.
- Mobile applications are subject to new types of attacks and require developers to revisit, learn and reprioritize security best practices.
- When seeking advanced mobile application security functionality for particularly sensitive apps, the landscape is fragmented, maturing and technically challenging to grasp.

Recommendations

Security and risk management leaders in charge of application security should:

- Provide early input on the performance-security trade-offs when a mobile architecture (native, hybrid or mobile web) is selected by being involved from the beginning of the process.
- Implement application security best practices with a focus on the specificities of mobile and its associated back end and possible API. In particular, eliminate hardcoded credentials, minimize app permissions, encrypt sensitive data and use certificate pinning where possible.

- Perform mobile application security testing and standardize the mobile security components used by employing ISVs, multiexperience development platforms and UEM capabilities in the process.
- Go beyond obvious controls, such as encryption at rest, for high-security apps by hardening and obfuscating code, preparing against tampering, and reverse-engineering attempts.

Strategic Planning Assumption

Through 2022, mobile application security failures will be the biggest mobile threat for enterprises.

Introduction

Mobile application security has become a tangible problem for enterprises.[1] While mobile device security has not been a major source of preoccupation and breaches, mobile application security failures are increasingly responsible for fraud[2] and enterprise breaches.[3] Often, these are public-facing apps that may be the primary or only way an organization is able to interact with its customers or partners. Because they can run on any mobile device, these apps are built to run in a hostile environment, under the control of an attacker. Security and risk management (SRM) leaders must protect mobile applications to enable the organization to advance toward its digital transformation (see Figure 1). This research highlights the main pitfalls and suggests approaches to avoid security failures when developing and using mobile apps.

Figure 1. Creating a Secure Mobile Application

Source: Gartner (August 2020)

Analysis

Identify the Risks for Your Mobile Applications

Gartner clients often express concerns about mobile applications, without a specific risk in mind. The goal is not to end up in the news for the wrong reasons. To effectively address them, SRM leaders must identify the specific risks that come with mobile applications. To do so, the best way to start would be to conduct a **threat modeling** exercise (illustrated as the first step in Figure 1), looking at the function of the mobile application and the threats applicable during its operation. The main principles of threat modeling persist with mobile app development, and there are various resources that can be used to adapt those principles to mobile.[4,]

[5] Broadly speaking, the main risks for enterprises that use enterprise data with mobile applications are:

- **Sensitive data loss —** Applications that lack adequate protection can allow attackers to obtain sensitive data that may reside in these applications, such as payment credentials and intellectual property. Some threat vectors can be device theft or loss,[6] malware on the device[7] and man in the middle (MitM) over unsecured networks.
- **Exposure of infrastructure —** Mobile applications need to communicate with enterprise back end services. This requires the enterprise to expose internal resources, such as an API or enterprise databases. If uncontrolled, this exposure can lead to a number of attacks, such as API scraping (see Note 1) and denial of service. In the Perform Application Security Testing section, we address the security testing aspects, while the infrastructure security aspects are illustrated in "API Security: What You Need to Do to Protect Your APIs."
- **Fraud —** This risk is particularly accentuated in financial services and retail applications, and wherever a mobile application includes financial transaction functionality where payment credentials are expected to be exchanged.[8, 9] The techniques attackers use include repackaging (see Note 2), SMS grabbing through malware,[10] script injection[11] and overlay attacks (see Note 3).
- **Compliance —** Noncompliance with specific regulations may lead to fines. European regulations such as GDPR and PSD2, are often cited by Gartner clients, but many other equivalent international regulations apply in this context. We do not address matters of regulatory compliance in this report.

Choose the Right Architecture

Before looking at the architectural options, an initial consideration would have to do with whether the app is published onto a commercial app store or distributed through enterprise distribution. For consumer-facing apps, the choice will be obvious, less so for workforce and partner-facing applications. Privately distributed apps will be less subject to attacks such as reverse engineering, and they will allow somewhat more visibility over the device. In addition, there are various options to deploy and impose security controls, among which are application management via UEM and stand-alone application management solutions ("Market Guide for Mobile Application Management" provides more details).

There are three main architectural options with mobile application development options available today: native, hybrid or pure web-based app. To these three categories, we would add PWAs (see "Assessing Progressive Web Apps to Provide Multiexperience Performance and Capabilities"), which are still emerging. Each architectural option leads to different security considerations and there will be different trade-offs that will need to be made between security and performance.

For instance, while it is fast and easy to convert an enterprise web application into a mobile web app, the cached content of the application will be difficult to encrypt effectively. (An open-source tool we have encountered among Gartner clients frequently is SQLCipher.) If the cached content is minimized and deleted too frequently for security purposes, this will impact the speed of the app and its user experience. Securing the app may prove to be lengthier and more expensive, and the architectural decision might need to be revisited.

A related consideration has to do with device and server-side checks. Security checks performed on the device can be bypassed by an attacker by manipulating the device or the app. For example, on a jailbroken or rooted device, an attacker can bypass native integrity check protections. For certain apps, it might be more suitable to put in place server-side controls, while other use cases may require dedicated on-device protection (refer to the When Needed, Go Beyond the Basics section for this last kind of functionality).

Native development provides access to all the native security capabilities of the iOS and Android platforms. Native applications are also known to offer a better performance since they use native APIs from the OS. Android and iOS have documentation available on best practices developing in the respective environments natively.[12, 13] Both native environments offer security functionality that can cater to basic needs, such as authentication and encryption, but also advanced ones like device attestation[14] and storage of credentials within trusted environments (see Note 4).[15] On the other side, native development will require to maintain at least two separate versions of the application.

For highly competitive app categories, the organization may be obliged to go with a native architecture. In other cases, hybrid architectures can represent a compromise between security and usability (or the usage of cross-platform frameworks such as React Native, Xamarin and Flutter). With hybrid applications, sensitive operations can be conducted with native functions, and the presentation can be delivered using web architecture based on HTML5. "Key Considerations When Building Web, Native or Hybrid Mobile Apps" contains an extended discussion around architectural choices.

Security leaders must seek involvement in the beginning of the application development process.

An added challenge is that often mobile apps are built and delivered independently through a business unit, without IT support or security involvement. To ensure they are involved, security leaders must communicate their policy to the various lines of businesses. Visibility into MX platforms, security champions and application discovery tools are all helpful instruments in this effort (see "DevOps Security Champions Help Organizations Gain Leverage Without Training Everyone").

Secure the Mobile Application

Secure application development (illustrated as the second step in Figure 1) principles do not change within the mobile domain. However, mobile apps move at

least a portion of the software logic onto a mobile device. This means certain application security best practices need to be particularly emphasized. "Toolkit: Security Checklist for Mobile App Developers" provides a comprehensive list, while in this report we point out the main pitfalls and best practices.

Do Not Hardcode Credentials

Mobile app developers tend to hardcode credentials in the app while developing.[16, 17] Rather than the more obvious authentication credentials for the user to authenticate to the application, here we intend authentication credentials for services used by the application. For example, when the app interacts with a third party, API developers will use their own authentication credentials. An attacker can obtain and use those credentials, especially if they are privileged ones, to steal data from an API. The app should authenticate to the API instead (see "API Security: What You Need to Do to Protect Your APIs").

Minimize App Permissions

Permissions empower apps, but this also opens the door to a number of risks. A legitimate app with unnecessary permissions is not only a privacy and compliance hazard, but it can also become a target for attackers. By default, an app should not require any device permissions. When necessary to conduct specific functions, permissions should be added selectively. Because most developers reuse existing libraries to develop, their apps end up asking for many more permissions than what they really need. SRM leaders should ensure the libraries used are of good quality and do not request permissions that they do not use. This vetting can be performed during testing, for example (see the Secure the Development Life Cycle section).

Protect Sensitive Data

A common pitfall in mobile is storing sensitive information within the app without proper protection.[18] Attackers can reverse engineer the code and steal this information. Access credentials and encryption keys should be safely stored, if they must reside on the device. Both iOS and Android provide dedicated storage for credentials and sensitive data, called Keychain and Keystore, respectively, which can be accessed through native functions (see Note 5). An alternative can be to minimize the sensitive data residing on the device. However, as mentioned, these decisions should be taken by evaluating the impact on performance that client-to-server interaction may require. A more complex alternative to using native key storage is white-boxing mechanisms, discussed in the When Needed, Go Beyond the Basics section.
In addition to using encryption, the app should be able to wipe its data via a remote command to cater for device loss.

Use Certificate Pinning Where Possible

Because they are used on the go, mobile apps connect from unsecured networks more frequently than web applications. Certificate pinning is a technique to counter man-in-the-middle attacks that could take place over such networks (see Note 6). There is some controversy around the usage of certification pinning, and the advice to use it comes with some caveats: NDR tools may not be able to work, as traffic inspection is harder to do (see "Demystifying the Impact of TLS 1.3 on TLS Inspection" for a broader discussion). Also, some browsers may not support certificate pinning, which makes it a challenge for hybrid apps. An open-source tool we have frequently encountered is TrustKit.

Secure the Development Life Cycle

Mobile application security testing (the third step in Figure 1) is the main component of a secure life cycle. Before conducting the testing itself, it is useful to create a scope document for the app, listing the permissions used, the level of sensitivity of data and the use case. For example, is the app published on a commercial app store where it can be downloaded by consumers, or will it run on managed mobile devices, where a UEM performs compliance checks? This information should be used to determine how much time the team will spend making the app secure and what kind of additional protection may be used. Additionally, it is mandatory to perform a threat modeling exercise.

Perform Application Security Testing

To ensure secure coding best practices are followed and identify any vulnerabilities introduced in the application, enterprises should conduct security testing on the application. Keeping in mind that a mobile application is composed of two portions: the app that resides on the mobile device and the back end. There are two levels of mobile application security testing. The first one analyzes the mobile app's source (the portion on the device), byte or binary code statically. This testing will allow issues such as unsecured storage to emerge. The OWASP Mobile Top 10 can serve as a list of what should be identified in this phase as a minimum. SRM leaders should ensure to test the entire app, including any libraries it may be using.

One peculiarity with mobile application development is that even the most mature organizations will conduct it in a DevOps fashion. Often the apps will be written within a line of business, and there will be weekly refreshes. This requires a mobile app security testing solution that is easy to run and fast to provide results. A few vendors that provide such mobile app security testing are Data Theorem, Kryptowire, ImmuniWeb, NowSecure, Pradeo and Zimperium.

The second level of mobile AST is composed of not only the code residing on the mobile device but also its interaction with (one or more) back ends. Some vendors will provide the first level as a free service and the second one as a paid service. Also, most vendors in the "Magic Quadrant for Application Security Testing" provide thorough mobile application security testing, and the "Critical

Capabilities for Application Security Testing" evaluates vendors specifically on that capability.
For critical applications, organizations should perform penetration testing periodically. This exercise may identify risks that the automated solutions may not be able to. For instance, the app or its back end may be vulnerable to specific techniques such as API scraping or password spraying, which an automated scanner may not run. This effort should focus on the business logic of the application.

Externalize Security Controls

Unless there is significant security internal know-how, organizations should avoid creating and implementing their own security controls. Most of the security issues identified with mobile apps are linked to incorrect implementations of cryptography, authentication and other security functionalities. SRM leaders should instead identify and standardize key security components. Look at your existing multiexperience development platforms and access management solutions for functionality such as integrity checking, jaibreak/root detection, authentication, authorization, tokenization and encryption (see "Magic Quadrant for Access Management"). MX platforms and Git environments can also help set up and govern the application development and deployment process (as just one illustrative example, see AppDome's App Workflow).
Some organizations with enough internal expertise and resources have managed to correctly build their security controls in-house. Carefully reflect on the effort you can dedicate in the long-run before taking that route. Over time, with new attacks and threats and respective new defenses, controls will become obsolete and inefficient, requiring ongoing maintenance that can become overwhelming. In fact, we have observed organizations that have built their own controls, only to have to dismantle them and look for external functionality. For workforce-facing apps, some of this security functionality can be provided and standardized via UEM tools (refer to the Application and Content Management section in the "Critical Capabilities for Unified Endpoint Management Tools").

When Needed, Go Beyond the Basics

When applications are critical, SRM leaders may need to go beyond the security controls illustrated so far. To identify candidate apps for these measures, there are three signs to look for:

- The app carries highly sensitive data or can be used for transactions.
- The app is public, either because it is published on a commercial app store or is a consumer app.
- The app has a significant portion of the software logic that resides on the device.

Harden Applications Against Reverse Engineering

Apps, especially ones published on commercial app stores, are frequently the victim of reverse engineering. Attackers reverse engineer applications to understand how the application works and how it can be attacked, to steal the data within the app, and also to clone the application. In this last case, also called "repackaging," the attacker adds malicious behavior and then reissues the app. Unaware users may think they are downloading and using the legitimate app, accidentally handing over their banking credentials, for instance. Application hardening (the fourth step in Figure 1) can make it more difficult for attackers to reverse engineer an application.

Code Obfuscation

To protect from repackaging, SRM leaders can use code obfuscation. Obfuscation scrambles (rather than encrypt) the code, making it harder to review and repurpose the code. There is not a single technique, but a multitude, from renaming portions of the code or changing its sequence to adding unused code and many others. Obfuscation is a dissuasive measure. In most cases, a determined and skillful attacker will be able to recompose the application, but the effort and time dedicated may not be worth it. Sample vendors are Arxan, Guardsquare, Intertrust, Irdeto, Preemptive Security and Verimatrix. There are many more options that we can discuss during inquiry.

White-Box Cryptography

This term refers to the set of techniques used to hide and protect sensitive application data such as keys and credentials stored in an app on a device. This can be used as an alternative when someone does not want to rely on native resources such as the previously mentioned iOS Keychain. This can be the case when, to increase adoption, the organization wants to allow its app to run on jailbroken devices, which cannot benefit from iOS Keychain protection. In other cases, the rationale for white-boxing is that the attacker sees the default location for credentials on a device as the first place to try to attack. Therefore, moving the credentials elsewhere avoids these attacks, despite all the fortifications that credential storages have on devices.

Mobile App Monitoring

For completeness, an alternative or addition to hardening the app against reverse engineering is to scout for modifications and cloned versions of the applications. RiskIQ, BrandProtect and other providers will monitor app stores and other resources and look for applications that may bear parts of their clients' code, logo or other information.

Verify the Environment Can Be Trusted With Anti-Tampering Controls

Public-facing apps need to be able to verify whether the device they run on can be trusted, and to what degree. To do so, SRM leaders can add anti-tampering controls to the application (the last step in Figure 1). Typical functionality are checks for the presence of a debugger (one of the tools used when conducting reverse engineering) or an emulator, in addition to checks for jailbreak or rooting. Build38, Promon and many of the vendors mentioned in the Harden Applications Against Reverse Engineering section provide anti-tampering. There are many more checks that can be performed, from checking for the presence of malicious apps on the device (Lookout, Pradeo and Zimperium have offerings in this space) to observing behavioral indicators on the device to determine whether the device is part of a mobile bot farm (Distil Networks is an example vendor here).
The "Market Guide for In-App Protection" provides more details on this. Some further vendors such as OneSpan, Cleafy, Kobil, XTN and others that can be found in the "Market Guide for Online Fraud Detection" can also provide advanced mobile application security functionality as part of a broader platform that can be used to detect and counter fraud.
As explained in "Teach Your Applications the Art of Self-Defense" and further discussed in "Don't Treat Your Customer Like a Criminal," especially in the consumer-facing world, the context, use case and industry will determine the reaction to an indicator of compromise. Some consumer-facing scenarios may work best by maximizing the user experience while accepting some risk. Others, such as employee-facing and critical apps in healthcare may be much more restrictive in the way they remediate.

Evidence

[1] "Mobile App Fraud Jumped in Q1 as Attackers Pivot From Browsers," DARKReading.
[2] "Digital Trust & Safety Index: A Rapidly-Changing Fraud Landscape," sift.
[3] "Fitness Tracking App Strava Gives Away Location of Secret U.S. Army Bases," The Guardian.
[4] "Mobile Application Threat Analysis," The OWASP Foundation.
[5] "The Developer's Guide to Securing Mobile Applications," Synopsys.
[6] The 2020 VDBIR found that 97% of mobile incidents involved a loss of a mobile device.
[7] "The StrandHogg Vulnerability," Promon. Promon security researchers have found proof of a dangerous Android vulnerability, dubbed "StrandHogg," that allows real-life malware to pose as legitimate apps, with users unaware they are being targeted.
[8] "FBI Expects Increased Targeting of Mobile Banking Applications," SecurityWeek.
[9] "RSA Quarterly Fraud Report Q2 2019," RSA.
[10] "Eurograbber Botnet Deftly Steals 36 Million Euros From Banks," Wired.

[11] "Inside the Magecart Breach of British Airways: How 22 Lines of Code Claimed 380,000 Victims," RiskIQ.
[12] "Introduction to Secure Coding Guide," Mac Developer Library.
[13] "Security Tips," Developers.
[14] "SafetyNet Attestation API," Developers.
[15] "Storing Keys in the Secure Enclave," Apple Developer.
[16] "Many Mobile Apps Unnecessarily Leak Hardcoded Keys: Analysis," SecurityWeek.
[17] "We Reverse Engineered 16k Apps, Here's What We Found," Hacker Noon.
[18] "Cybersecurity Conference Exposed Attendee Info via a Garbage App Because of Course It Did," Mashable.

Note 1: API Scraping

API scraping refers to the act of extracting a large amount of data using an API as an interface. API scraping can be used in a malicious way by an attacker that abuses an API to extract a large amount of data and use it in an unauthorized way. For instance, the attacker could attempt to resell the data. API scraping can be countered through rate limiting, but this practice could also block legitimate requests.

Note 2: Repackaging Attack

For consumer-facing apps, this is the primary risk to be added on top of the list above. Repackaging consists of reverse engineering a legitimate app, adding malicious code to it and reuploading it in a public app store. This has been the principal vector of attack for banking applications. Reverse engineering is relatively simple, and it can be achieved via a decompiler in Android or a disassembler in iOS.

Note 3: Overlay attack

An overlay attack is an attack that uses an overlay of a window or application over the legitimate one. This creates to the user the illusion they are still using the legitimate application or web page and it can attempt to steal credentials.

Note 4: Trusted Environment

A trusted environment can be a secure portion of a device. Its purpose is to ensure secure storage and processing of sensitive data, such as digital rights management (DRM)-related or payment data.

Note 5: iOS and Android Keychain and Keystore

In Android, there are two separate mechanisms, the Keychain and the Keystore. The Keychain is for systemwide credentials, while stand-alone apps should resort to the Keystore resource. In iOS the Keychain has different protection classes and keys. The class should be set to make credentials readable only when the device is unlocked.

Note 6: Certificate Pinning

Applications typically provide transport security using certificate validation as part of a security protocol, such as TLS. The check can, for instance, be that the certificate is provided by a valid root certificate authority. While this check can be fine when the identity of the other party is known, an unknown attacker over an unsecure network could perform a man in the middle attack, establish a secure connection with the client and the server, by providing valid certificates. By pinning the certificate, the client or server expects a specific certificate to be presented to them. A valid certificate from an attacker would therefore be rejected.

Thank you for your feedback!

- Send Feedback