

Mobile App Platform Team

Mobile Platform

17 November, 2025

Table of Contents

1	Mobile Platform Team (MPT) Role.....	4
2	Tools	5
3	App Publication	8
3.1	How To Publish An App Developed For Our Company.....	9
4	Technical Documentation	12
4.1	Technology Choices - Hybrid vs. Native Mobile Applications.....	14
4.1.1	Native (preferred).....	15
4.1.2	Hybrid (less preferred).....	16
4.2	Platform Standards.....	17
4.2.1	Android	18
4.2.2	iOS.....	32
4.3	User Acceptance Testing (UAT) and General Pre-Release Testing.....	47
4.3.1	General Testing Advice.....	48
4.3.2	Distribution Channels Summary	55
4.3.3	Google Play Internal, Alpha, and Beta (Android)	58
4.3.4	TestFlight (iOS)	58
4.3.5	BrowserStack Testing.....	59
4.3.6	Test Distribution To Shop Floor Devices	59
4.3.7	Environment Handling During Testing.....	60
4.4	Summary of deliverables.....	62
4.4.1	General Deliverables Requirements.....	63
4.4.2	Screenshot and Icon Specifications	66
4.5	Other Standards	68
4.5.1	Analytics and Crashlytics	68
4.5.2	Bitrise.....	69
4.5.3	Code Quality	70
4.5.4	Data Minimization.....	71
4.5.5	General Standards	74
4.5.6	Guidance on updates.....	79

4.5.7	Localization	80
4.5.8	Repository Management	82
4.5.9	Secrets Handling In Source Code	83
4.5.10	Vulnerability Scan	85

1 Mobile Platform Team (MPT) Role

Our team partners with you to develop and maintain mobile applications published by our company in a cost-effective and policy compliant manner. We offer number of services that can help you asses your project and decide if you want to develop the application yourself, use our development services or asses an external vendor. Products and services we offer include:

- **Product & solution Development**
 - End-to-End development of custom-tailored products fitting your existing landscape or process
 - App assessment and solution proposal review of used technology, time, and costs, in a context of app development, maintenance, and overall publication
 - Assisting ITRMS with vulnerability reviews of mobile applications developed by/for our company
 - Publication of applications developed for/by our company in our company's official App Store and Play Store accounts
 - Management of our company's official App Store and Play Store publication accounts, including application identifier management
 - App specific Firebase configuration access management
- **Consulting**
 - Compliance, maintenance and technical assessment of existing apps resulting in a list of recommendations and proposed fix solutions
 - Informal training and guidance on existing Merck internal processes to speed up time to market and minimize security risks
 - Assist setting vendors' responsibilities for chosen technology and app readiness for our company's environment
- **Guidelines**
 - Written recommendations for compliant development practices, UI & UX, available tools and technologies
 - Regular communication to share and learn new advancements and best practices for mobile development, including Google and Apple policy changes
 - Providing technical guidelines to Animal Health, Human Health, and others to help improve division specific processes
- **Technology enablement**
 - Automated tooling setup to deliver code changes more frequently and reliably for less re-works and quick iterations of your apps
 - Providing build environments for Android and iOS app builds, experimentation or development for internal teams or vendors
 - Inventory of applications developed by/for our company to help reduce duplication, and to allow for better cross-divisional management of our company's public image

2 Tools

- MASP
 - A centralized inventory to maintain a record and publication status of all applications published by our company, whether for internal or external users
 - Provides one reference point for discovering build resources (SCM, Bitrise, Artifactory, etc.) associated with a mobile application
 - Developed and managed directly by the Mobile Platform Team (MPT)
 - Provides a tool that uploads ipa/aab files to the artifactory for apps that we do not own source code for and merely re-sign before publication
 - See <https://masp.merck.com> and [MASP - Knowledge Base](#)¹ for more information
- GitHub
 - Git based source code management (SCM) for all developed content which our company owns source code for
 - Hosts app specific Bitrise workflows for all mobile apps published by our company
 - Managed by the GitHub team
 - Provides code level insights leveraging other tools (e.g. Wiz, secret scanning, issue management, etc.)
 - See <https://github.merck.com/>, use <https://go.msd.com/get-github> for more information
- Bitrise
 - Mobile app specific build system
 - Provides Mac and Ubuntu machines
 - Managed by MPT
 - MPT manages centralized/standardized workflows, as well as some app-specific workflows
 - Handles re-signing services for mobile applications that we do not own source code for
 - Takes source code from GitHub (for apps developed by our company), or ipa/aab files from the Artifactory (for apps that are re-signed)
 - Uploads signed builds to vulnerability scanning, App Store Connect, and Google Play
 - Provides reporting tools for detection of use of font files, and use of the keyword "merck" in many instances
 - See [Bitrise.io](#)² and <https://bitrise.io/> for more information
- Artifactory
 - System Managed by the Artifactory team
 - Artifactory repositories managed by MASP to store builds for re-signing
 - Used in some cases for additional build content

¹ <https://share.merck.com/spaces/FPMP/pages/1515891705/MASP+-+Knowledge+Base>

² <https://share.merck.com/spaces/FPMP/pages/1048458637/Bitrise.io>

- See [Templates | FORGE³](#) to create any other artifactory repositories referenced by workflows/source code
- DataTheorem (replacement in progress)
 - Managed by ITRMS in collaboration with MPT
 - Used by MASP for vulnerability scanning on all mobile apps published by our company
 - MPT provides vulnerability triage/recommendations for mobile applications published by our company depending on app features
 - Integral part of app approval
 - See <https://go.merck.com/appsecrequest> for more information
- App Store Connect
 - Apple's publishing system used for publication of all iOS mobile applications, whether for internal or external users
 - Accounts managed by MPT
 - Also used for supplying TestFlight builds for pre-release testing
- Apple Business Manager
 - Apple's system used for delivery of internal iOS mobile applications to our company's managed iOS devices
 - Managed by the MIS team
 - Delivers internal applications published by MPT in App Store Connect to apps@merck
 - Contact MISEngineering@merck.com⁴ for more information
- Developer.apple.com
 - Accounts managed by MPT
 - Used for creation of mobile app signing certificates and profiles
 - Used for creation of additional elements such as notification certificates, and wallet pass certificates
- Google Play Console
 - Google's publishing system used for publication of all Android mobile applications, whether for internal or external users
 - Accounts managed by MPT
 - Also used for supplying Google Play Internal Test Channel builds for pre-release testing
- Managed Google Play
 - Google's system used for delivery of internal Android mobile applications to our company's managed Android devices
 - Managed by the MIS team
 - Delivers internal applications published by MPT in the Google Play Console
 - Contact MISEngineering@merck.com⁵ for more information

³ <https://forge.merck.com/create/templates/default/dxop-artifactory-local-repo-create-template>

⁴ <mailto:MISEngineering@merck.com>

⁵ <mailto:MISEngineering@merck.com>

3 App Publication

Account Details

iOS Applications developed by/for our company (including template apps those customized for our company) that are for use by those outside of our company (members of the public, customers, healthcare consumers, or healthcare practitioners), and all company owned Android apps are published by our team in our official Merck Sharp & Dohme LLC App Store and Play Store accounts.

iOS Applications developed by/for our company (including template apps those customized for our company) that are for use by our employees and contractors with company managed devices are published in our company's MSD Czech Republic s.r.o. account by our team, and then made available in apps@merck by the MIS team.

Before You Start Questions

Who are the intended users? Which division is the application for?

- Mobile applications developed for users outside of our company are published in the MSD LLC App Store and Play Store accounts and since they are publicly visible and may be considered to be pharmaceutical marketing are subject to additional compliance processes
- All apps, regardless of intended users will need to follow DigitalSDLC
- Different divisions may have additional compliance requirements that must be met
- Human Health *public* (aka external) applications should start by contacting dccmobile@merck.com⁶ or if a named contact is needed sonam.jain@merck.com⁷
 - Human Health public applications currently use the HH Service Desk at <https://go.merck.com/submitapp>, our team works with Human Health to perform technical work
- Human Health *internal* applications can contact our team directly for app publication
- Animal Health applications, whether internal or external, should start by contacting alena.kuznetsova@msd.com⁸
 - All Animal Health applications must go through Animal Health's Service Desk and portfolio demand process, our team works with Human Health to perform technical work
- All other divisions can contact our team directly for app publication for internal or external publication guidance

Mobile Platform Team Role / Expectations

For HH (public) apps, and all AH apps MPT will work with HH/AH for app publication, including onboarding onto GitHub (when necessary), establishing a Bitrise instance, establishing a Firebase instance, managing source code (when necessary), etc.

⁶ <mailto:dccmobile@merck.com>

⁷ <mailto:sonam.jain@merck.com>

⁸ <mailto:alena.kuznetsova@msd.com>

For internal non-AH apps MPT will work directly with the BO/TO, and/or their designated development team.

3.1 How To Publish An App Developed For Our Company

HH (public) apps - contact dccmobile@merck.com⁹ - you will be guided on the appropriate information needed with HH's Service Desk at <https://go.msd.com/submitapp>

AH apps (public and internal) - contact alena.kuznetsova@msd.com¹⁰ - you will be guided on the appropriate information needed with AH's Service Desk

All others - contact fp-mobile-platform@merck.com¹¹

	BO/TO	Mobile Platform Team (MPT)
Project initiation	HH (public) apps - contact dccmobile@merck.com¹² AH apps - contact alena.kuznetsova@msd.com¹³ All others - contact fp-mobile-platform@merck.com¹⁴ BO responsible for organizing developer/vendor access to Merck's systems where necessary BO responsible for setting project requirements	MPT will work with HH/AH/BO where necessary to provide concept and technical advice MPT will provide technical and concept guidance MPT will help establish a GitHub instance if none already exists MPT will establish a Bitrise instance (company mobile app build system) MPT will establish a Firebase instance if necessary MPT will provide access to all of the above for BO/TO's designated users MPT will establish app store instances and bundle IDs MPT will provide access to UAT/DEV app store instances where necessary

⁹ <mailto:dccmobile@merck.com>

¹⁰ <mailto:alena.kuznetsova@msd.com>

¹¹ <mailto:fp-mobile-platform@merck.com>

¹² <mailto:dccmobile@merck.com>

¹³ <mailto:alena.kuznetsova@msd.com>

¹⁴ <mailto:fp-mobile-platform@merck.com>

	BO/TO	Mobile Platform Team (MPT)
Development	Provides content and other project requirement direction to development teams Facilitates provision of source code from development teams and/or app builds (ipa/aab files, as appropriate)	MPT will provide technical guidance to development teams and BO/TO on request MPT will perform vulnerability scans and provide advice on request MPT will submit test versions of apps to TestFlight and/or Google Play Internal Test channel on request MPT will provide guidance on Apple/Google policy requirements where necessary/on request MPT can provide source code reviews on request MPT will provide additional Bitrise access to development team on request
Testing	Provides Apple IDs (iOS) and Google IDs (Android) of testers to MPT Provides testing metadata to MPT Performs testing of application Coordinates with development team for bug fixes/issue remediation	MPT will submit app to TestFlight (iOS) or Google Play Internal Test using provided metadata MPT will add in tester access MPT can perform high level reviews/testing of app behavior on request focussing on adherence to Google/Apple policies MPT can provide access to testing tools such as BrowserStack for BO/TO app reviews on request
Publication	Handles all compliance requirements and DSDLC Provides app store metadata to MPT Provides release approval to MPT	MPT will provide app store metadata advice MPT will update app store listings with provided metadata MPT will submit app to Apple/Google's review MPT will negotiate with Apple/Google review for some app review issues, for all others MPT will provide BO/TO with details of reported app review issues and remediation advice where appropriate MPT will release application on Apple/Google approval and BO/TO approval MPT will update MASP to maintain a company-wide inventory of mobile applications

	BO/TO	Mobile Platform Team (MPT)
Post-Publication	Manages any app content/change requests/reviews statistics Monitors app health issues using established tools Monitors app reviews in public apps for PV/AE/PQC issues as well as app quality issues Manages development team for app maintenance and future updates	MPT will monitor Apple and Google policy changes/alerts and alert BO/TO of issues MPT will monitor app store and alert BO/TO for critical events (e.g. multiple crashes) MPT will alert BO/TO of unexpected critical issues regarding content (e.g. contact from font license holders) MPT will work with HH/AH to provide app health reports on request

4 Technical Documentation

- [Technology Choices - Hybrid vs. Native Mobile Applications \(see page 14\)](#)
- [Platform Standards \(see page 17\)](#)
 - [Android \(see page 18\)](#)
 - [Android / Google Play Store Rejection Reasons \(see page 19\)](#)
 - [Android Keystores and Identifiers \(see page 23\)](#)
 - [Android Other Policies \(see page 24\)](#)
 - [Android Packaging, Design, And Architecture \(see page 27\)](#)
 - [Android Privacy And Sensitive Data Features \(see page 29\)](#)
 - [iOS \(see page 32\)](#)
 - [iOS / App Store Rejection Reasons \(see page 33\)](#)
 - [iOS Other Policies \(see page 41\)](#)
 - [iOS Packaging, Design, And Architecture \(see page 43\)](#)
 - [iOS Privacy And Sensitive Data Features \(see page 44\)](#)
 - [iOS Signing and Identifiers \(see page 46\)](#)
- [User Acceptance Testing \(UAT\) and General Pre-Release Testing \(see page 47\)](#)
 - [General Testing Advice \(see page 48\)](#)
 - [Distribution Channels Summary \(see page 55\)](#)
 - [Google Play Internal, Alpha, and Beta \(Android\) \(see page 58\)](#)
 - [TestFlight \(iOS\) \(see page 58\)](#)
 - [BrowserStack Testing \(see page 59\)](#)
 - [Test Distribution To Shop Floor Devices \(see page 59\)](#)
 - [Environment Handling During Testing \(see page 60\)](#)
- [Summary of deliverables \(see page 62\)](#)
- [Other Standards \(see page 68\)](#)
 - [Analytics and Crashlytics \(see page 68\)](#)
 - [Bitrise \(see page 69\)](#)
 - [Code Quality \(see page 70\)](#)
 - [Data Minimization \(see page 71\)](#)
 - [General Standards \(see page 74\)](#)
 - [Guidance on updates \(see page 79\)](#)
 - [Localization \(see page 80\)](#)
 - [Repository Management \(see page 82\)](#)
 - [Secrets Handling In Source Code \(see page 83\)](#)
 - [Vulnerability Scan \(see page 85\)](#)
 - [Vulnerability Fixes \(see page 87\)](#)
 - [Android App Private Data Exposed Through Backups \(see page 89\)](#)

- [Android App Views are Vulnerable to TapJacking \(see page 90\)](#)
- [Android Enable Verification on Android Security Provider \(see page 91\)](#)
- [Android Enable Android NSC to Prevent Cleartext Traffic \(see page 93\)](#)
- [Android File-Based Cross Site Scripting \(see page 94\)](#)
- [Android Protect App Screens from 3rd Party Apps \(see page 95\)](#)
- [Android Ensure Verify Apps Is Not Disabled \(see page 95\)](#)
- [Android Insecure Random Number Generator Usage \(see page 97\)](#)
- [Android Unsafe HostnameVerifier Defined \(see page 97\)](#)
- [Android Unsafe SSL TrustManager Defined \(see page 98\)](#)
- [iOS Disable Third Party Keyboards \(see page 99\)](#)
- [iOS Dangerous ATS Exemptions \(see page 101\)](#)
- [iOS Enable Data Protection Entitlement \(see page 102\)](#)
- [Android / iOS App Leverages SDK Vulnerable to Zipperdown \(see page 102\)](#)
- [Android / iOS User Password Stored in App Private Storage \(see page 103\)](#)
- [Android / iOS - Cordova Enforce Strict use of Apache Cordova Web Origin Access Parameter \(see page 104\)](#)
- [iOS WebView Disables Same-Origin Policy - baseURL Argument \(see page 104\)](#)
- [iOS Location "Always" Permission Requested \(see page 105\)](#)
- [iOS Switch from Unsafe Serialization API to NSCoder \(see page 105\)](#)
- [iOS App Susceptible to URI Scheme Hijacking \(see page 106\)](#)
- [iOS Device Advertisement Identifier Collected \(see page 107\)](#)
- [iOS Insecure XML Parser Configuration - NSXMLParser \(see page 108\)](#)
- [iOS Enable SSL Pinning - NSURLConnection and NSURLSession \(see page 108\)](#)
- [iOS App Store Blocker: App Uses IPv4-specific APIs \(see page 110\)](#)
- [Android / iOS Credentials Exposed \(see page 111\)](#)
- [iOS Weak Keychain Protection Class - AccessibleAlways \(see page 111\)](#)
- [iOS Prevent Keychain Items on Third Party Backups \(see page 112\)](#)
- [Server Lacks OCSP Stapling \(see page 112\)](#)
- [Server Supports Insecure TLS 1.0 \(see page 113\)](#)
- [Server Lacks HTTP Strict Transport Security Header \(see page 113\)](#)
- [iOS Enable System-Managed Login Input Fields \(see page 114\)](#)
- [iOS Enable Two-Factor Authentication Input Field \(see page 115\)](#)
- [iOS Cleartext HTTP Connections \(see page 116\)](#)
- [iOS App Opens Port to Listen on Local Socket \(see page 116\)](#)
- [iOS Set Expiration on Pasteboard Data \(see page 117\)](#)
- [iOS Raw RSA Algorithm Used with Padding Disabled \(see page 118\)](#)
- [iOS Hash Generated Using Broken Cryptography API \(SHA1\) \(see page 118\)](#)

- [iOS Secure Enclave not Being Used When Possible \(Security.framework\) \(see page 119\)](#)
- [iOS Message Digest Generated Using Broken Cryptography API \(see page 119\)](#)
- [iOS Switch from Deprecated UIWebView API to WKWebView \(see page 120\)](#)
- [iOS Switch to Write-Only Access to User's Photo Library \(see page 120\)](#)
- [iOS App Embeds SQL Query with Dynamic Input \(see page 121\)](#)
- [iOS Utilize Private Photo Access APIs \(see page 122\)](#)
- [iOS App Accesses Private APIs \(see page 122\)](#)
- [iOS App Built with Deprecated SDK \(see page 123\)](#)
- [Android / iOS Application Logs Sensitive Data \(see page 123\)](#)
- [Android JavaScript Interface Injection Vulnerability \(see page 124\)](#)
- [iOS Weak ECB Mode Used for Symmetric Encryption Affected Components \(see page 125\)](#)
- [iOS Null Initialization Vector Used for Symmetric Encryption \(see page 126\)](#)
- [iOS App Store Blocker: App Uses UIWebView APIs \(see page 127\)](#)
- [Android / iOS App Embeds SQL Query with Dynamic Input \(see page 127\)](#)
- [Android / iOS Keys and Passwords Identified in the App \(see page 128\)](#)
- [iOS Missing Privacy Manifest \(see page 129\)](#)
- [Android / iOS App Obfuscation Disabled - Classes, Keys, & Tokens are Clear-text \(see page 129\)](#)

4.1 Technology Choices - Hybrid vs. Native Mobile Applications

The choice over what technologies to use, and whether to create a native or hybrid application depends on several factors including security, cost, suitability, and longevity.

Currently our company works with the following technologies:

Swift - native (iOS) - preferred for long term iOS solutions, single platform

Kotlin - native (Android) - preferred for long term Android solutions, single platform

React-Native - cross-platform solution - for short term cross-platform solutions as more frequent maintenance is required than native Swift/Kotlin apps

.Net MAUI - cross-platform solution - for cross-platform projects which have Enterprise Architecture (EA) approval to use .Net MAUI, or pre-approved for migration of legacy/currently published Xamarin based .Net projects

Unity - cross-platform solution - for AR/VR/3D cross-platform apps, creates large solutions, requires paid licenses to create/build and maintain, *use only for 3D/AR/VR applications with project approval from EA, and after budgetting for ongoing licensing and maintenance costs*

Consult with our Enterprise Architecture team before any development starts on a project in order to ensure it meets our current stack standards: https://collaboration.merck.com/sites/ent_arch/SitePages/Standards.aspx?q=mobile

In general native applications are better for almost every situation, due to their better suitability for app store release, as well as ease of maintenance and less reliance on third party components. React-Native based applications are suitable for short term lifecycles with cross-platform solutions, but have historically had maintenance and security issues and rely heavily on third party components owned by others outside of our control. Unity based applications are good for specialized 3D and virtual/augmented reality based projects and are usually resilient to decompilation but may cause frequent maintenance issues in the long term, it is also possible to create VR/AR applications using native Swift and Kotlin tools provided by Apple and Google without requiring a license.

4.1.1 Native (preferred)

Native mobile applications are applications developed for a specific platform, written in code native to that platform or written in managed code and compiled to be native to the platform either using pre-compilers or just-in-time-compilers. Acceptable native mobile apps for iOS and Android are written in Swift and Kotlin, and must always use the versions of those languages supported in the current release versions of Xcode and Android Studio.

Native applications are easier to optimize, faster, usually offer more control of native features, and offer better access to native APIs. Applications built with native technologies usually have greater maturity and longevity due to the close coordination with Apple's and Google's OS development teams, meaning that they often require less maintenance to keep them buildable and updatable for longer periods than hybrid applications.

Native mobile applications can still suffer from vulnerabilities and exploits including those stemming from poor coding practices, bugs in libraries, and flaws in native operating systems. Good coding, security, and privacy practices can mitigate the presence and possible exploitability of these vulnerabilities. Any development guidance and sample code from Apple and Google is usually written for either Swift or Kotlin, with occasional support for other technologies.

Native apps are preferred by our company, and by Apple/Google.

Information on native mobile apps

iOS	https://developer.apple.com/develop/
Android	https://developer.android.com/docs/

4.1.2 Hybrid (less preferred)

Hybrid technologies are often used to create cross-platform applications which have one code base that is either wrapped or compiled at build-time/run-time for different platforms.

Whilst hybrid apps can be flexible, especially when creating cross-platform mobile applications, they can frequently suffer from poor support, or can provide poor user experiences due to a lowest common denominator approach or be limited with regards to which native OS features are accessible/configurable. They can often be slower than native applications since they can require runtime parsing of code, and can be susceptible to vulnerabilities and exploits common to both native code and unique to the hybrid code. Hybrid applications are usually reliant on many third-party library or service for building, running, and maintenance, which are often poorly maintained or outdated frequently, with additional third-party libraries required for access to features that are immediately available to native applications. Some hybrid technologies may require licensed tools or libraries to develop, publish, or maintain, and since they are often maintained out of step with Apple's or Google's changes, updates to these tools to support current publication requirements and to remediate vulnerability issues may be outside of our control.

Careful consideration of business needs must be taken when deciding to create hybrid mobile applications over native applications or websites, and careful coding practices must be followed in order to mitigate these risks. In general native applications have far greater flexibility, security, stability, and longevity but may sometimes require slightly longer development times and specialized skill-sets when creating apps for both Android and iOS. We strongly recommend that teams create native mobile applications and not hybrid applications in cases where long term publication is needed.

Hybrid applications which merely present web content such as html/javascript pages in a custom coded browser element (for example Ionic/Cordova) are discouraged by Apple for some uses and are also discouraged by our company. Due to the way mobile applications work, applications which consist of website code wrapped in a custom browser will have that website based code accessible to anyone who downloads the app to some extent. Website code is best suited for use on websites and not on mobile applications.

As hybrid applications make extensive use of 3rd party libraries which will usually contain unused code and sometimes may contain undocumented features, careful review of all libraries is required - both Apple and Google require accurate data safety and privacy information and the parent application is responsible for any behaviors of SDKs, even if these behaviors are undocumented and/or outside of the parent application's control. Examples of violations can be 3rd party frameworks that collect and/or analytics information or device IDs, or require excessive permissions - even if those permissions requests are the result of unused code in the framework.

Any updates or new publications of mobile applications using hybrid technologies **must** use the current stable version of those technologies and **must** meet all current platform requirements.

Examples of currently acceptable hybrid mobile app technologies

React-Native	https://facebook.github.io/react-native/
Unity (3D/AR/VR apps only)	https://unity3d.com/
.NET MAUI (if approval received from Enterprise Architecture)	https://dotnet.microsoft.com/en-us/apps/maui

4.2 Platform Standards

All Apple and Google requirements **must** be followed in order for applications to be publishable, whether for internal or public applications. Guidelines and requirements for both platforms are regularly updated and are subject to change without notice. Enforcement action can range from individual apps being blocked from release through to accounts being removed, or our company being blocked from publishing to a platform.

Apple's current guidelines are at <https://developer.apple.com/app-store/review/guidelines/> with updates announced in <https://developer.apple.com/news/>.

Google's current guidelines are at <https://play.google.com/about/developer-content-policy/>.

Applications must have a clear "app-like purpose" that offers some benefit to the user for both Apple's and Google's platforms. In order to meet this guideline, how an app implements features and the marketing presentation can be just as important as the features it has and the content it presents.

An application must make sure users can make fully informed decisions on using it in a professional and clearly understandable manner by ensuring users are aware of who the app is published by, what it does, what device features it accesses, what data it collects, why it needs that data, how that data is used, only requesting the data that is actually needed to perform the operations needed to achieve its features (required to meet both platform and security standards), and how to contact us for support with the service(s) that the application offers.

- [Android \(see page 18\)](#)
 - [Android / Google Play Store Rejection Reasons \(see page 19\)](#)
 - [Android Keystores and Identifiers \(see page 23\)](#)
 - [Android Other Policies \(see page 24\)](#)
 - [Android Packaging, Design, And Architecture \(see page 27\)](#)
 - [Android Privacy And Sensitive Data Features \(see page 29\)](#)

- [iOS \(see page 32\)](#)
 - [iOS / App Store Rejection Reasons \(see page 33\)](#)
 - [iOS Other Policies \(see page 41\)](#)
 - [iOS Packaging, Design, And Architecture \(see page 43\)](#)
 - [iOS Privacy And Sensitive Data Features \(see page 44\)](#)
 - [iOS Signing and Identifiers \(see page 46\)](#)

4.2.1 Android

Google's rules for Android apps can be found at:

<https://play.google.com/about/developer-content-policy>

and

<https://support.google.com/googleplay/android-developer/answer/13837496>

Developers and vendors are expected to be aware of Google's policy changes whilst developing applications, and during any support period. As updates must meet any new policy requirements, applications that need minor business case updates might also require larger updates to satisfy any additional Google requirements.

4.2.1.1 Consequences

Google operates a "strikes" policy, but they do not disclose how many strikes a developer can have. Each violation of a policy counts as a strike, with some more significant violations being treated more seriously.

Individual applications may be reduced in visibility (e.g. demoted in search results, or not presented for certain device ranges in the Google Play Store), have prominent flags alerting users that it may not be compatible with their devices, or removed from the Play Store.

Applications published in accounts which have previously had applications that have issues are subject to longer review times and more in-depth reviews.

Repeated violations of Google's technical and content policies will result in the suspension and termination of not only the developer account those violations occurred in, but also suspension and termination of all associated developer accounts.

Multiple violations of Google's Policies will lead to our company being banned from publishing Android applications.

The below content is a summary of areas where our company commonly sees issues

- [Android / Google Play Store Rejection Reasons](#) (see page 19)
- [Android Keystores and Identifiers](#) (see page 23)
- [Android Other Policies](#) (see page 24)
- [Android Packaging, Design, And Architecture](#) (see page 27)
- [Android Privacy And Sensitive Data Features](#) (see page 29)

4.2.1.2 Android / Google Play Store Rejection Reasons



All content and changes within the Play Store listing for an app, including metadata, policy declarations, and the application bundles themselves are subject to Google's reviews and approvals.

Our externally facing mobile applications must meet the same standards as all applications published in app stores, set out in each platform's guidelines. In order to successfully publish with minimum issues careful decisions must be made in the design and development stages of mobile app development, including attention to application stability, vulnerability, UI, UX, solution architecture, API design and back-end server architecture.

All Android applications are reviewed both automatically and some are reviewed manually, with some guidelines are enforced more strictly than others depending on Google's current focus and depending on the reviewer. All content and changes in Google's system are subject to Google's review and approval to ensure it meets Google's policies, including adding/removing of countries, adding/removing store listing localizations, updating policy declarations, updating privacy policy websites, unpublishing applications from the Play Store, etc. as well as the technical design and implementation of applications and content shown within applications. To help try to prevent app rejection some examples of rejection reasons for Android applications are listed below.

The content below is for example use only, Google may have many other rejection reasons

4.2.1.2.1 Generally not meeting Google Play's policies

If there are a number of issues, Google may send a generic notice, indicating that the app doesn't meet their policies, along with the current status of the app. To fix - the app *and* the store listing must be updated to meet all policies.

Example rejection notification:

Not adhering to Google Play Developer Program policies

Status

- Update rejected
- Previous version available on Google Play

Rejected

May 19, 2022, 7:31 AM

Your app does not adhere to [Google Play Developer Program policies](#)¹⁵

How to fix

Address any issues and make sure your app meets all policy requirements.

4.2.1.2.2 Not supporting required targetSDK range

Applications must meet Google's targetSDK range in order to remain live, this range is updated by Google each year. To fix - the production app must be updated to meet both the current targetSDK range and all other Google Play policies. Google will include a deadline, an indication of the current status, and indication of the current enforcement action.

Example rejection notifications:

App must target Android 12 (API level 31) or higher

Status

- App not available to new users on Android versions higher than your target API level

Reduced availability from
Aug 30, 2023

Warning sent
Aug 10, 2023

Warning deadline
Aug 30, 2023

¹⁵ <https://play.google.com/about/developer-content-policy/>

To provide users with a safe and secure experience, Google Play requires all apps to meet target API level requirements.

From Aug 30, 2023, if your target API level is not within 2 years of the latest Android release, your app will not be available to new users running Android versions higher than your app's target API level.

App must target Android 13 (API level 33) or higher

Status

- *error* You won't be able to release app updates

Enforced

Aug 30, 2023

Warning sent

Aug 10, 2023

Warning deadline

Aug 30, 2023

To provide users with a safe and secure experience, Google Play requires all apps to meet target API level requirements.

From Aug 30, 2023, if your target API level is not within 1 year of the latest Android release, you won't be able to update your app.

4.2.1.2.3 Privacy Policy doesn't meet Google's requirements

Google requires that privacy policies meet their requirements, rather than any specific country's or region's legal obligations. As this policy is checked automatically using AI scrapes of websites and applications there are many false positives whilst at the same time appeals are almost impossible due to a lack of developer support from Google. The only fix possible is to make sure that the privacy policy is specific to the application (mentioning it by name) or specific to the publisher name in the Play Store, and matches data usage/collection practices, data safety declarations, etc. exactly and resubmitting.

Google's systems check for an exact match for the app name or developer account name and will reject *any* change to an application, including non-technical changes such as changes to distribution countries or store listing text, if the supplied privacy policy website landing page does not have an exact match (including spelling, capitalization, and punctuation).

Example rejection notification:

Privacy policy in Google Play User Data policy: Invalid privacy policy

Status

- Update rejected
- Previous version available on Google Play

Rejected

Nov 4, 2024, 9:56 AM

Your app's privacy policy does not meet necessary policy requirements.

Under the [User Data](#)¹⁶ policy, you must link to a privacy policy on your app's store listing page and within your app. Apps that do not access any personal and sensitive user data must still submit a privacy policy.

Please [add or update your privacy policy](#)¹⁷, and make sure it is available on an active URL (no PDFs), is non-editable, applies to your app, and specifically covers user privacy.

Please review the following compliance checklist for privacy policies. Your privacy policy must:

- Be linked on your app's store listing page in Play Console and within the app itself.
- Hold reference to the entity (for example, developer, company) named in the app's Google Play listing or the app.
- Include a clear labeling as a privacy policy.
- Include a privacy point of contact or a mechanism to submit inquiries.
- Be readable in a standard browser without any plug-ins or special handlers (no PDFs).
- Be available on an active, publicly accessible and non-geofenced URL.
- Be non-editable.
- Comprehensively disclose how your app accesses, collects, uses, and shares user data.

This must include:

- Disclosing the types of personal and sensitive user data your app accesses, collects, uses, and shares; and any parties with which any personal or sensitive user data is shared.
- Secure data handling procedures for personal and sensitive user data.
- Developer's data retention and deletion policy.

Where this issue was found

¹⁶ <https://support.google.com/googleplay/android-developer/answer/10144311>

¹⁷ <https://support.google.com/googleplay/android-developer/answer/113469#privacy>

This issue may also be found in other locations. Check all areas of your app when fixing this issue.

Location
Evidence

Privacy policy

URL provided <https://msdprivacy.com> does not link to a valid privacy policy page, missing reference to the app or to the entity (for example, developer, company) named in the app's Google Play listing.

4.2.1.3 Android Keystores and Identifiers

4.2.1.3.1 Keystore generation and storage

Keystores for Android applications will be generated, managed, and stored by the mobile platform team.

Keystores will not be provided to vendors or agencies.

When an application bundle is first uploaded to a new app the keystore used for it becomes the application's keystore. It cannot be easily changed once set outside of a support request to Google, and even then only if the app is enrolled in Google Play Managed Signing. Therefore applications will be enrolled in Google Play Managed Application Signing to avoid future keystore loss.

The keystore is an independent but related concept from the application ID.

4.2.1.3.2 Android Application Transfers

Keystores for Android applications which are transferred to our company accounts must be provided at the time of transfer.

4.2.1.3.3 Application IDs

Once set and a build uploaded (including to the Google Play Internal Test channel) an application ID is locked to the account.

An application ID cannot be changed during the life of an application as it identifies the app in Google's ecosystem.

4.2.1.3.4 White label Application Handling

White label applications (applications where we don't own the source code and where vendors supply a pre-built AAB) will be resigned by the mobile platform team under a release keystore for our account.

The resigning process for Android apps does not allow changing of the application ID in AAB files, this must be set at time of build.

Vendors providing AAB files must discuss this with the mobile platform team before building an application.

4.2.1.3.5 Application names

App names should be unique for the country of release, and should be globally unique in order to avoid future conflicts.

App names that are already in use by others cannot be used by our company unless we submit a trademark/copyright claim.

Our company should own the trademark/copyright of any app names to reduce our legal risk.

Short name variants can be used to display app names under the icon when installed, but must generally match the published name shown in the app stores.

App names must be less than 30 characters long including any whitespace characters, but ideally should be less than 12 to 15 characters. Names longer than about 12 characters will be truncated when shown on devices depending on device form-factor and user-customizations.

App names should not contain the work "Merck" unless they are to only be available to Merck based regions and users for their entire lifetime. "MSD" is acceptable to use in app names instead of "Merck" as there are no restrictions on using "MSD" in Merck based regions.

4.2.1.4 Android Other Policies

4.2.1.4.1 Credentials

Google will block publication of any application which requires a login if they have not been provided with valid credentials for their testing. If these credentials are changed after publication the app will be removed by Google.

Apps which require a login **must** have a demonstration mode that illustrates all the app's features or credentials **must** be provided.

Apps that require a login but do not have a demo mode **must** provide credentials that work on production and which are kept active for the life of the app, apps which do not provide these details cannot be published.

Apps which do not have features, or a legal reason, for requiring a user to have an account (either to log in or to create an account) should not require a user to log in to use it.

4.2.1.4.2 Age rating

Android applications which could be seen to appeal to children, even if not aimed at or designed for children, must also meet Google's Family's policy requirements at <https://play.google.com/about/families/>, and will undergo additional reviews or testing by Google.

Due to state-level law changes some applications may require additional authorization and implementation of age rating APIs to be able to be distributed to users residing in some states in the United States. See <https://support.google.com/googleplay/android-developer/answer/16569691> for details on this and steps that developers may take. Google have not announced details on the exact application features which might require these controls yet, but they may be required to remove or restrict distribution and use of mobile applications to users in states that have enacted or are enacting restrictive laws on technology such as Texas, Utah, and Louisiana.

4.2.1.4.3 Metadata

Android apps in the Play Store must provide an accurate disclosure and description of their functionality in all parts of the store metadata. App descriptions must describe what the app actually does. App screenshots must show the app in use. If the features or UI of an app change with an update then the screenshots must be changed to match the current appearance of the app. App categories chosen must match what an app does. App metadata must not include references to trademarks or IP belonging to others without documented permission. An app's functionality must be clear to users before they download it, and while in use, and must not contain any hidden, dormant, or undocumented features.

4.2.1.4.4 Misrepresentation

Google expressly state that "Techniques to evade app reviews are not allowed. Apps may be required to provide additional details to ensure user safety, system integrity, and policy compliance." This means that any mis-representation in any metadata for an app must be accurate to the app that is submitted to Google's review. Attempts to disguise an app's policy compliance with vague statements, for example to attempt to justify permissions usage that would not otherwise meet Google's requirements are not allowed and may result in account termination.

4.2.1.4.5 Health Related Apps

Google requires that applications which are health related (including those which only offer health related information and are not classified as medical apps) complete a Health Apps declaration. In addition to this health related applications must have a privacy policy both inside the app and in the app's store listing which: is not geofenced; "together with any in-app disclosures, comprehensively disclose[s] the access, collection, use, and sharing of [personal or sensitive user data](#)¹⁸, not limited by the data disclosed in the Data safety section"; fulfills all applicable [prominent disclosure and consent requirements](#)¹⁹; contains no permissions that are "not required for a health app to perform its core functionality" with all unused permissions removed; *and* meets all legal requirements for other aspects (e.g. medical device approval, human subject trials related information, government affiliation, etc.)

This applies to applications which provide any sort of health or disease management (for example symptom trackers), clinical decision information (such as dosage calculators), health status monitoring, disease prevention (such as medical dictionaries), emergency/first aid instructions, healthcare services management (for example appointment reminders or apps that support users to with features to connect with healthcare providers), medical reference and education (for example the MSD Manuals apps), medication/treatment managers (such as dosage calendars), and more.

Aside from an overt declaration by our company that an app is health related, Google will automatically classify some applications as health-related based on permissions requested (e.g. a permission to access a user's physical activity), or through use of the "health" or "medical" app

¹⁸ <https://support.google.com/googleplay/android-developer/answer/10144311>

¹⁹ <https://support.google.com/googleplay/android-developer/answer/10144311>

metadata tags, or other keywords. On an app being classified as health-related, even if it is not health-related, stricter review and enforcement of other policy elements will be applied by Google.

The most relevant aspects of this requirement for most applications we publish will be that the applications **remove unused permissions, and have a privacy policy which comprehensively discloses all data collection, sharing, and use**. The privacy policy requirement includes any web pages that are presented within the application (such as login/user registration forms, etc.). The easiest way to ensure the privacy policy requirement is manageable is to ensure data minimization: if an application does not need a piece of information to function (such as user's name, email, address, device ID, camera access, location, full media library access, etc.) then any libraries used should not ask for it, and if it does need that information to make sure that the collection and usage are prominently disclosed in the privacy policy and within the application itself.

All applications which contain any type of health, pharmaceutical, or medical information should also provide citations within the app to show the source of the data (e.g. studies/trials/data sheets/etc. even if, as pharmaceutical manufacturers, we are the primary source for that data).

For more information see <https://support.google.com/googleplay/android-developer/answer/13996823> and <https://support.google.com/googleplay/android-developer/answer/14738291>.

4.2.1.4.6 Page Sizes

From November 2025 Google requires all apps to support 16kb page sizes in order to be published or updated. Apps that are currently published can request an extension until May 2026. Building with the latest Android SDK and NDK, and using the latest versions of libraries that support the correct page size should resolve this issue. If a particular library does not support 16kb page sizes due to included shared objects then it must be updated to meet Google's requirements to be used.

For more information see <https://developer.android.com/guide/practices/page-sizes>.

4.2.1.4.7 COVID and Health Misinformation

To combat misinformation Google currently requires that applications do not refer to COVID, keywords related to COVID, or provide COVID-related symptom advice without approval by, and references to the official governmental body providing the information for the intended publication country(ies). Any other sources for references other than official government health bodies **must** be vetted, evaluated, and pre-approved by Google's medical teams before any app submission. Violation of this requirement may automatically cause account suspension. Apps may not offer or appear to offer any sort of contact tracing or tracking services without full government approval from all countries of release.

Google also requires health-related apps to accurately represent their functions, features, and data, including full citations, accurate descriptions, no mis-leading claims, etc. This rule applies to metadata contained within store listings as well as the app itself. To best avoid applications being incorrectly flagged by Google, as well as ensuring that any store metadata is attractive to end-users, ensure the store description accurately and clearly state what an application does, what information it may present and/or collect, where that information comes from, and avoid generic store descriptions that could be considered to be marketing puffery.

4.2.1.4.8 News Apps

Apps that describe themselves as news apps, including apps which are published under the category "News and Magazines" as well as apps which use the keywords in their metadata which could make the app appear to be "news" based apps must provide ownership information about both the app and the source of the news articles, including the original publisher/author of each article. They must also have an easy to find dedicated website/in-app page that is clearly labelled as communicating contact information including a contact email address and phone number. Social media accounts are not sufficient methods of identifying article authors/publishers.

4.2.1.4.9 Government Associated Apps

Google requires that any apps that are developed for, in association with, or by government organizations - to include individual departments of local, regional, state, or national governments, as well as health authorities must declare their affiliation. This information is not shown to users but is used by Google in their app reviews. Any relationship with governmental organizations can be explained when submitting an application for publication.

4.2.1.4.10 Financial Features

Any financial features within an application must be declared to Google, this includes any digital transactions, payment handling, rewards points, etc. For more details see <https://support.google.com/googleplay/android-developer/answer/13849271>.

4.2.1.5 Android Packaging, Design, And Architecture

4.2.1.5.1 Packaging

All Android apps **must be packaged to use .aab format**, although our company mobile development teams will handle most of the final builds, solutions must use recent versions of libraries and be architected to support .aab format. For most native Kotlin apps no additional configuration changes will be required, however some hybrid solutions such as Unity, React-Native, or projects with unknown complexities related to chosen architectures may require deeper project configuration changes to ensure they support being packaged as aab.

4.2.1.5.2 Targeting/Architecture

Android applications **must** have a targetSDK level meeting or exceeding the standards at <https://developer.android.com/google/play/requirements/target-sdk> - the targetSDK requirement increments annually.

An app's activities **must** have the correct exported attribute set, applications which do not have this set will be blocked by Google.

Currently published applications which do not have a targetSDK within two years of the current requirements will not be available to download for new users on current devices or users with devices running a current Android version. Users will receive a warning that apps are not compatible with their devices, and Google will block any updates until the targetSDK requirement is met.

4.2.1.5.3 Debug Elements

Release AABs must be free from all debug elements as Google will reject release builds. In addition to this, our network does not allow additional debug elements that are often inserted by third party app generation tools such as React Native, Expo, Unity, etc. If a white-label AAB is provided then this **must not** contain DebugProbesKt.bin.

4.2.1.5.4 Spam and Minimum Functionality

Google expects applications to have app-specific functions, and not consist of (for example) static content, or present only PDF files. Applications with very little content that do not provide a good user experience will also be flagged by Google as not meeting their minimum functionality policy.

Applications that force close, as well as crash, freeze, stop responding, or have abnormal behavior will be flagged by Google.

Applications that present the same content as other apps without any original value, or applications that have highly similar functionality will be flagged as spam by Google.

4.2.1.5.5 SDK Responsibility

Even if an SDK or library is provided by a third party, the app it is in, and the publisher of the app (our company) are responsible for its behavior - including data collection, malware, crashes, sensitive data access, etc. Adverse behavior, malware, or undeclared data collection by an SDK will cause an application to be rejected or removed by Google, undeclared data collection by an SDK may lead to privacy issues.

If a library or an SDK requires access to sensitive data that is not used in the applications core features, then it cannot be used - for example an application which uses a library to access media files cannot request broad access to media files, even if that broad access permission code is unused code within the library / SDK.

4.2.1.5.6 Stability

Any applications which have stability or other technical issues, or which do not meet Google's requirements cannot be published or uploaded to Google Play Closed or Google Play Open testing channels. Applications which do not meet Google's technical or content standards (including content

shown on store listings) may be blocked from publication and removed from the stores by Google, including if issues are only in test versions.

Google will mark any app that has a perceived stability rate that exceeds their allowed thresholds (>0.47% percentage of users overall, and/or >8% users on particular devices - see https://support.google.com/googleplay/android-developer/answer/9844486?hl=en#user_perceived_anr_rate_threshold for the current thresholds) with warnings to users that the app may not work on their device. Google may also choose to hide the app, or may remove the app from distribution entirely.

4.2.1.6 Android Privacy And Sensitive Data Features



Android requires that privacy policies must be available in app as well as hosted on a website.

4.2.1.6.1 Privacy Policy Content

Google requires that all applications, whether they access sensitive data or not, include a privacy policy that is specific to the app, publisher account, and company that publishes the app. The checks are automated and scrape the privacy policy for compliance to their policies, with little room for appeal due to Google's lack of non-automated developer support. Applications, and updates to applications, will be rejected by Google if this requirement is not met, or if the privacy policy website becomes inaccessible during the life of the app. This requirement is more strictly enforced for apps that Google considers to be health related (including apps presenting medical information as well as those which are formally classified as medical apps).

The privacy policy must:

- Be available as a website linked to from the store listing page in the Google Play Console and from within the app itself.
- Reference the entity (for example, developer or company) named in the app's Google Play listing (e.g. Intervet B.V. for MSD Animal Health applications, or Merck Sharp & Dohme LLC for Human Health applications) or the app name. The name presented must match exactly.
- Include a privacy point of contact or a mechanism to submit inquiries.
- Include clear labeling as a privacy policy.
- Be readable in a standard browser without any plug-ins or special handlers (e.g. without needing a PDF reader)
- Be available on an active, publicly accessible and non-geofenced URL.
- Be non-editable.
- Reference the use of any sensitive data related permissions, including how that data is used, collected, and (if relevant) shared. For example an app that has the GPS permission set must reference the use of the user's location and how that data is used in the privacy policy.

- Comprehensively disclose how the app accesses, collects, uses, and shares user data. This must include:
- Disclosing the types of personal and sensitive user data your app accesses, collects, uses, and shares, and any parties with which any personal or sensitive user data is shared.
- Secure data handling procedures for personal and sensitive user data.
- The data retention and deletion policy.

4.2.1.6.2 Sensitive Data Use

Any applications which access personal or sensitive data APIs such as location, files, photos, etc. **must** adhere to Google's policies at https://support.google.com/googleplay/android-developer/answer/9888076#personal_sensitive and <https://support.google.com/googleplay/android-developer/answer/13986130>. Applications which access location may only do so as the result of a user triggered action, may only do so if the app has a good reason for needing this information, must only request the minimum request the minimum of access necessary to achieve their core features, may only do so whilst needed by the feature which are submitted, and may not access location when in the background without a core feature that requires it. Apps submitted to Google which do not meet these requirements will be rejected automatically.

Applications which use background location **must** display a prominent notification to the user prior to asking for the permission containing the following elements:

- Include the term "location"
- Include at least the following sentence localized to all supported languages, adapted to include all the relevant features requesting access to location in the background in the app that are readily visible to the user: "This app collects location data to enable ["feature"], ["feature"], & ["feature"] even when the app is closed or not in use."
- Include a list of all the features of the app which use background location
- Include any other details necessary to make it clear to the user how and why you are using location in the background. While additional content is permitted prior to the notification it should not prevent the required content from being immediately visible.
- Must be within the app itself, not only in the app description or on a website
- Must be displayed in the normal usage of the app and not require the user to navigate into a menu or settings
- Must describe the data being accessed or collected
- Must explain how the data will be used and/or shared
- **Cannot** only be placed in a privacy policy or terms of service
- **Cannot** be included with other Disclosures unrelated to personal or sensitive data collection
- Does not need explicit consent such as an "accept" or "I understand" granted by the user as this is done in the runtime prompt that immediately follows; enabling the user to close or swipe away are acceptable ways to migrate out of the disclosure.

A video link **must** be provided demonstrating the feature, including the prompt, a prominent disclosure and showing the entire process from the background. This **must** be provided to Google and be accessible to them - so **must** be hosted on YouTube (public) or shared via Google Drive.

For more information on background location permissions see <https://support.google.com/googleplay/android-developer/answer/9799150>

Applications which require access to a user's files must do so on a granular level using the OS APIs to access scoped storage, and must not request broad access without a core feature that is in the permitted use cases.

Google require an app privacy statement similar to Apple's App Privacy statement for iOS. App owners and development teams must make this information available when submitting an app, and we are responsible for any data usage by third party SDKs as well as our own code. Some information is discovered automatically through vulnerability scans and Google's reviews, however some may be undetectable without knowledge of the app/service and its behaviors. In addition to this, Google will flag any app that accesses a user's device or other ID, even if only accessed by Google's SDKs. Apps that use analytics libraries should use a non-device ID variant of those libraries or must declare their gathering and transmission of device IDs.

Applications that request permissions to access features that Google describes as sensitive must have a good use case meeting those which Google describes at <https://support.google.com/googleplay/android-developer/answer/9888170>. Applications which do not meet those use cases and provide the requested evidence cannot be published on Google Play (including test channel versions).

4.2.1.6.3 Photo and Media Library Access

Google Play regards photos and media kept on a device as a user's personal information, Android apps may only access photos and videos for purposes directly related to app functionality. Apps may not request access on behalf of any third-party or for any purpose unrelated to the user-facing app functionality. **Applications are only allowed to access photos and videos for purposes directly related to an app's functionality.**

Apps that have a one-time or infrequent need to access these files *must* use the system picker, such as the Android [photo picker](#)²⁰ and remove legacy access methods.

From 31st October 2024 Apps that request access to the READ_MEDIA_VIDEO or READ_MEDIA_IMAGES permission must successfully demonstrate a core use case that requires (in Google's sole opinion) persistent or frequent need of photo/video access located in shared storage. See <https://support.google.com/googleplay/android-developer/answer/14115180> and <https://developer.android.com/about/versions/14/changes/partial-photo-video-access> and <https://support.google.com/googleplay/android-developer/answer/14115180> for more information.

This requirement includes using any libraries (such as the many React-Native libraries) that leverage these methods, even if the permissions are not directly used by the application - in this case latest versions of library supporting the updated access methods should be used.

²⁰ <https://developer.android.com/training/data-storage/shared/photopicker>

4.2.1.6.4 FULL_SCREEN_INTENT and USE_EXACT_ALARM Permissions

From Android 14 Google require apps to only use FULL_SCREEN_INTENT if their core features are alarms and calls. Special permission is required from Google to use this intent. See <https://source.android.com/docs/core/permissions/fsi-limits> and https://support.google.com/googleplay/android-developer/answer/9888170?sjid=10605832113554880418-NC#:~:text=USE_FULL_SCREEN_INTENT%20permission%20if%20the%20core%20functionality for more information.

Similarly, SCHEDULE_EXACT_ALARM is restricted only to apps whose core features are alarms, calendar alerts, and timers. Special permission is required from Google to use this intent. See <https://developer.android.com/about/versions/13/features#use-exact-alarm-permission>. If your app's core features do not meet the allowed use cases then you can continue to use SCHEDULE_EXACT_ALARM, but the app must gracefully handle a user denying that permission.

4.2.1.6.5 Analytics / Advertising ID

If apps use analytics, then they must be configured to use a version of Firebase which does not collect the advertising ID. Since Firebase collects the advertising ID by default, this can be done following the instructions at https://firebase.google.com/docs/analytics/configure-data-collection?platform=android#disable_advertising_id_collection. If the advertising ID is required for functionality then user permission must be obtained to access it, and a functional pathway must still exist for users who decline. Any use of the Advertising ID must be declared to Google (even if caused by Google libraries).

4.2.1.6.6 Account handling

Apps whose core features do not require an account should not require a user to log in.

Applications which offer users the ability to create an account **must** allow users to delete that account from within the app. Apps which have accounts and do not have the in-app delete account feature will not be publishable.

Applications which offer users the ability to create an account **must** have a web interface outside of the app to allow users who have deleted the app to delete their accounts.

For more information on Google's account deletion requirements, see - <https://support.google.com/googleplay/android-developer/answer/13327111>, apps without the required information may be removed by Google.

Using a web page for in-app login and account deletion is generally considered a poor user experience, although less rigidly enforced for Android than on iOS - account management and login features in app should aim to provide a native in-app experience.

A demo mode that allows for full exploration of an app is encouraged.

4.2.2 iOS

Apple's rules for iOS apps can be found at:

<https://developer.apple.com/app-store/review/guidelines>

and

<https://developer.apple.com/app-store/submissions/>

Developers and vendors are expected to be aware of Apple's policy changes whilst developing applications, and during any support period. As updates must meet any new policy requirements, applications that need minor business case updates might also require larger updates to satisfy any additional Apple requirements.

The below content is a summary of areas where our company commonly sees issues

- [iOS / App Store Rejection Reasons](#) (see page 33)
- [iOS Other Policies](#) (see page 41)
- [iOS Packaging, Design, And Architecture](#) (see page 43)
- [iOS Privacy And Sensitive Data Features](#) (see page 44)
- [iOS Signing and Identifiers](#) (see page 46)

4.2.2.1 iOS / App Store Rejection Reasons

Our externally facing mobile applications must meet the same standards as all applications published in app stores, set out in each platform's guidelines. In order to successfully publish with minimum issues careful decisions must be made in the design and development stages of mobile app development, including attention to application stability, vulnerability, UI, UX, solution architecture, API design and back-end server architecture.

All Apple applications are reviewed both automatically and manually, and some guidelines are enforced more strictly than others depending on Apple's current focus and depending on the reviewer. To help try to prevent app rejection some examples of rejection reasons for iOS applications are listed below. Others are listed by Apple at <https://developer.apple.com/app-store/review/#:~:text=Avoiding%20common%20issues>.

The content below is for example use only, Apple may have many other rejection reasons

4.2.2.1.1 Different country's applications published as separate app rather than one localized version

If an application is to be published in a different country then the application should be localized to offer content for that country and not published as a separate version unless there is a verifiable legal reason for doing so (e.g. a country's laws require it). Applications for both iOS and Android must meet this requirement. Not meeting this guideline risks the suspension of our accounts and the removal of all of our company's applications. From a technical standpoint all apps can be localized to provide both different text and different content/features based on a user's locale and there are many ways provided by platforms for doing so. If there is a verifiable reason for not updating an existing application to offer localized content and instead publishing a separate app, documentation for this must be provided at the time of submission.

Localizing existing applications is also often cheaper and faster than developing separate applications.

Example rejection notification:

"Guideline 4.3 - Spam

Upon further review, we noticed that your app provides the same feature set as other apps you have submitted to the App Store; it simply varies in content or language, which is considered a form of spam.

When creating multiple apps where content or language is the only varying element, you should offer a single app to deliver differing content to customers.

Next step:

- Review the Design section of the [App Store Review Guidelines](https://developer.apple.com/app-store/review/guidelines/)²¹.
- Ensure your app is compliant with all sections of the [App Store Review Guidelines](https://developer.apple.com/app-store/review/guidelines/)²² and the [Terms & Conditions](https://developer.apple.com/terms/)²³ of the Apple Developer Program.
- Once your app is fully compliant, resubmit your app for review.

When creating multiple apps where content is the only varying element, you should offer a single app to deliver differing content to customers."

²¹ <https://developer.apple.com/app-store/review/guidelines/>

²² <https://developer.apple.com/app-store/review/guidelines/>

²³ <https://developer.apple.com/terms/>

4.2.2.1.2 Applications requiring content downloads on launch

All iOS applications should contain enough content to function at first launch without requiring additional downloads. This does not have to be an application's complete data set, but it should have enough content to allow a user to navigate and use the application albeit within reasonable limits. This can usually be avoided by containing some data elements locally, and fetching and caching additional data sets as/when the user accesses that data. A well-balanced caching strategy is advised.

In some situations initial content downloads may be unavoidable in which case the user must be kept informed and as long as there is good technical and/or legal justification for doing so we can sometimes submit appeals to Apple. Poor solution architecture or back-end design does not count as technical justification and in general large downloads of archives containing an application's entire data set provide a poor user experience and should be avoided.

Example rejection notification:

"Guideline 4.2.3 - Design - Minimum Functionality

Your app did not include sufficient content in the binary for the app to function at launch, and we were required to download or unpack additional resources before we could use it.

Next Steps

To resolve this issue, please revise your app to ensure that users can use the app on launch without needing to download or unpack additional resources."

4.2.2.1.3 Applications initiate downloads without obtaining user consent

All iOS applications which do require large content to be downloaded must always inform the user of the download size and obtain their consent before proceeding. Applications can sometimes avoid this issue completely by having a well designed back-end API supplying and caching information as/when needed rather than downloading all application content en-masse, however if unavoidable applications must obtain the user's informed consent before starting downloads, should show determinate progress indicators keeping users informed of the download process, and allow the process to be paused, cancelled, or resumed as required by users and network situations.

Example rejection notification:

"4. 2 Design: Minimum Functionality

Guideline 4.2.3 - Design - Minimum Functionality

We were required to download or unpack additional resources to continue using your app; however, the size of the download was not disclosed, and we were not prompted to choose to download additional resources. If your app requires additional resources, you must disclose the size of the download and prompt users before doing so."

4.2.2.1.4 Applications missing purpose strings

All iOS applications must include purpose string descriptions in the application's .plist file for accessing sensitive device features, these should be localized and describe exactly why the application needs access to the user's sensitive data in as clear and succinct a manner as possible. Apple's checks this using an automated scan of the app's OS API usage. If an application intentionally has features which require the use of sensitive data they must include a purpose string, if they do not then the calls and methods which access the sensitive data should be removed, even if the app itself does not access the code referring to sensitive APIs. Some third party libraries include methods and calls which trigger this alert - if our company has access to the source code for the libraries then a custom branch of the libraries without these API calls should be created.

Effective 2020 by inference from rejections during Apple's review process the only acceptable fix in this case is to use versions of libraries which don't contain methods that access unused sensitive APIs. Apps with the plist key for access to sensitive data must follow https://developer.apple.com/documentation/uikit/protecting_the_user_s_privacy and must provide a clear pathway showing how the feature is accessed, what steps are in place for users who choose not to provide access, and readily accessible details of how that data is used.

Example rejection notification:

"Your app's code references one or more APIs that access sensitive user data. The app's Info.plist file should contain a NSLocationWhenInUseUsageDescription key with a user-facing purpose string explaining clearly and completely why your app needs the data. Starting Spring 2019, all apps submitted to the App Store that access user data are required to include a purpose string. If you're using external libraries or SDKs, they may reference APIs that require a purpose string. While your app might not use these APIs, a purpose string is still required. You can contact the developer of the library or SDK and request they release a version of their code that doesn't contain the APIs."

4.2.2.1.5 Applications missing a clear description

All applications must have a clear and concise metadata (description and screenshots) which explain/show the app's purpose and features to help potential users decide whether to install the application. Both Apple and Google can reject applications based on unclear descriptions. To avoid rejection always provide full descriptions and screenshots in all target languages which clearly illustrate the application's purpose, features, and intended audience, as well as full release notes for updates.

Good application UI and UX design making the application's purpose, features, and use clear, as well as a clear application description in all target languages will avoid this issue.

Example rejection notification:

"2. 1 Performance: App Completeness

Guideline 2.1 - Information Needed

We have started the review of your app, but we are not able to continue because we need additional information about your app.

Next Steps

To help us proceed with the review of your app, please provide detailed information to the following questions. The more information you can provide upfront, the sooner we can complete your review.

- Who is the target audience?
- What is the purpose of the app?
- What are the main features?"

4.2.2.1.6 Applications which collect health related data

Applications which *in Apple's opinion* appear to conduct health-related human subject research must provide users clear consent to do so, including information regarding:

- the nature, purpose, and duration of the research
- procedures, risks, and benefits to the participants
- information about confidentiality and handling of data (including any sharing with third parties)
- a point of contact for participant questions
- the withdrawal process

- a disclaimer in the store description and in the app advising users to consult a health professional

Applications should also provide evidence at the time of submission of independent ethics review boards which will conduct periodic reviews of app related data. To avoid this, provide details of the consent process, as well as details of exactly where and how data is stored/used, and details of any independent review boards, as well as clear documentation of the app's purpose when submitting for publication.

If an application has the potential to be considered to be conducting human health research (such as symptom trackers or patient diaries) then full verifiable information, as well as in-app legal documentation, must be provided at the time of submission to fully cover questions such as - what the data collected is being used for and who has access to the data collected. This information must be approved for distribution to Apple.

Example rejection notification:

"Guideline 5.1.3 - Legal - Privacy - Health and Health Research

Your app is conducting health-related human subject research, but we were unable to locate all of the required consent disclosures.

Next Steps

To resolve this issue, you must obtain consent from research participants or, in the case of minors, their parent or guardian. The consent must disclose:

- the nature, purpose, and duration of the research
- procedures, risks, and benefits to the participants
- information about confidentiality and handling of data (including any sharing with third parties)
- a point of contact for participant questions
- the withdrawal process

Additionally, apps conducting health-related human subject research must secure approval from an independent ethics review board. Please provide proof of such approval."

4.2.2.1.7 Applications with inaccurate screenshots

All apps published in the app stores must have screenshots which accurately show the app in use, taken on the correct devices for the screen shot sizes used. If the app's layout or features change depending on if the user is on an iPad or an iPhone the screenshots for those sizes must portray that correctly. If a mock-device frame is used in an image along with marketing text then that frame must

match the device expected for that screenshot size (e.g. for notched sizes then a notched frame must be used) and the content within that frame must still be of the app in use on that device.

The rejection notification will usually specify which screenshot sizes are inaccurate. A new build is not always required to fix this issue, but new screenshots will be required.

Example rejection notification:

"Guideline 2.3.3 - Performance - Accurate Metadata

We noticed that your screenshots do not sufficiently reflect your app in use.

Specifically, your 12.9-inch iPad Pro, 12.9-inch iPad Pro (3rd Generation), 6.5-inch iPhone, and 5.5-inch iPhone screenshots do not display the actual app in use.

Next Steps

To resolve this issue, please revise your screenshots to ensure that they accurately reflect the app in use on the supported devices. For example, a gaming app should feature screenshots that capture actual gameplay from within the app. Marketing or promotional materials that do not reflect the UI of the app are not appropriate for screenshots.

The iPhone screenshots should reflect use on iPhone devices.

The iPad screenshots should reflect use on iPad devices.

App Store screenshots should accurately communicate your app's value and functionality. Use text and overlay images to highlight your app's user experience, not obscure it. Make sure app UI and product images match the corresponding device type in App Store Connect. This helps users understand your app and makes for a positive App Store experience."

4.2.2.1.8 Applications with development artifacts in screenshots

All apps published must contain accurate metadata with no reference to unfinished or unimplemented features, test information (for example the words "test version"), etc. in both description and screenshots. The rejection notification will sometimes say what is wrong and where, but will not always go into such detail. To fix this ensure that the screenshots and description do not contain development artifacts or elements such as a TestFlight link, obvious testing content, or references to anything which is not included the release candidate. Take screenshots on a release candidate and if a test account is needed populate it with dummy data that mirrors what a user would see in reality and is not obviously test information.

Example rejection notification:

"Guideline 2.3.10 - Performance - Accurate Metadata

We noticed your submission includes irrelevant references to your app's development process.

Specifically, your screenshots includes Back to TestFlight buttons.

Since apps on the App Store should be ready for users, they should not include references to the development process."

4.2.2.1.9 Test application submitted to production

Apple requires that only release candidates are submitted for Apple's full release review. Test versions of applications should not be released to either private/custom distribution or public distribution. TestFlight should be instead, or if not possible then ad-hoc testing.

Example rejection notification:

"Guideline 2.2 - Performance - Beta Testing

Your app appears to be a pre-release, test, or trial version with a limited feature set. Apps that are created for test or trial purposes are not appropriate for the App Store."

4.2.2.1.10 Unmaintained applications

Once published apps must be updated to remain live. Apps which are not updated regularly, or which no longer function will be removed by Apple. Apple review published apps at least once every 2 years, more frequently if issues are reported to them.

The only solution is to ensure applications are updated and maintained on a regular basis - a good standard is to set out a maintenance and improvement plan for continual development once an app is released, with a good industry guideline to be to publish updates at least every 6 months to a year.

Example rejection notification:

"We noticed that your app has not been updated in a significant amount of time.

Next Steps

To keep your app on the App Store, submit an updated version for review and make sure it follows the latest App Review Guidelines. If you are unable to submit an update within 30 days, your app will be removed from the App Store until you submit an update, and it is approved.

If Your App is Removed

Your app will remain fully functional for current users. They will experience no interruption to services, will still be able to buy in-app purchases, and can re-download the app while restoring from an iCloud or iTunes backup. However, we recommend that you update your app as soon as possible to reinstate it on the App Store and ensure that it remains functional and engaging for new and existing customers.

You can continue to use your current app name, as your app has not been deleted from your account. However, please note that if your app name is longer than 50 characters, you will need to modify it to meet current naming guidelines.

The TestFlight version of this app will also be unavailable for external and internal testing and all public TestFlight links will no longer be functional until the production version of your app is submitted for review and approved."

4.2.2.2 iOS Other Policies

4.2.2.2.1 Credentials handling

App Owners must provide credentials valid for production for any applications which involve user authentication or any other login/activation methods.

If login requires multi-factor authentication an account which bypasses this must be provided to be used for Apple's review.

A demo mode is a preferred alternative to providing Apple with user credentials.

4.2.2.2.2 Store metadata

Application store listings metadata (store descriptions, screenshots, etc.) must meet all of Apple's store content requirements - the content must be accurate, reflect the app's features and functions, and must not include irrelevant details, puffery, or unrelated marketing blurb.

Application metadata must match the branding and core functionality of the application being published, and must be focussed on the "Apple" experience. Do not mention features planned to be added in the future, and do not mention other platforms. The store metadata is one of the main elements that new users will use when choosing to install an application, make the metadata attractive to the target users.

4.2.2.2.3 Age rating

Due to state-level law changes some applications may require additional authorization and implementation of age rating APIs to be able to be distributed to users residing in some states in the United States. See <https://developer.apple.com/news/?id=btkirlj8> for details on this and steps that developers may take. Apple have not yet announced full details on the exact application features which might require these controls yet, but they may be required to remove or restrict distribution and use of mobile applications to users in states that have enacted or are enacting restrictive laws on technology such as Texas, Utah, and Louisiana.

4.2.2.2.4 Contact details

The support URL, privacy URL, and marketing URL (optional) in an application's store listing must be valid and kept up to date for the life of the application. Failure to have working URLs and appropriate contact details in the landing pages of those URLs may also be the cause of litigation in countries which require it.

4.2.2.2.5 Medical advice requirements

Applications which, in Apple's sole opinion, present medical advice will be reviewed with greater scrutiny by Apple. Those apps must present a plan showing they will be regularly maintained and provide citations in-app to support any medical calculations or information, they must also include advice to users to consult with their doctor in addition to using the app and before making medical decisions.

If apps have received regulatory approval this approval must be provided to Apple as well as cited in app.

Dosage calculators, medical information, and similar advice can only be published in the accounts of pharmaceutical manufacturers, hospitals, universities, health insurance companies, pharmacies, or official approved entities (with regulatory approval) - this means that such content cannot be published by agencies or vendors and must be published in our accounts.

4.2.2.2.6 TestFlight use

TestFlight packages will undergo beta reviews by Apple prior to being made available on TestFlight. Applications submitted for external TestFlight must be substantially complete release candidates and must meet the same Apple review guidelines.

4.2.2.2.7 Application catalogs (promoting or listing other applications)

Applications may promote other applications published by us (<https://developer.apple.com/app-store/review/guidelines/#acceptable>), but may not advertise applications published by third parties (<https://developer.apple.com/app-store/review/guidelines/#unacceptable>).

4.2.2.3 iOS Packaging, Design, And Architecture

4.2.2.3.1 Architecture and purpose

If an app is merely presenting website content in a wrapper (whether a locally hosted or a remotely hosted website) then it needs to be a website and not an app. Apps **must** be app-like in features, appearance, use, design, and content. An app must have a clear purpose and not simply consist of marketing materials (<https://developer.apple.com/app-store/review/guidelines/#minimum-functionality>).

4.2.2.3.2 iOS SDK and Swift versions

All iOS applications must be built with a minimum of the currently released Xcode version and iOS SDK. At the time of writing this is Xcode 15/iOS SDK 17, from September 2024 it is expected to change to iOS SDK 18.

Swift based applications **must** be developed with a version of Swift supported by the currently released Xcode version. Apps which do not meet these requirements will not be publishable.

4.2.2.3.3 Device form-factor / orientation support

iPhone apps must be designed to support all iPhone screen sizes. All iPad apps must support all iPad screen sizes and orientations unless the requires full-screen attribute is set.

Once an app is published for a device set (i.e. iPad/iPhone) then all subsequent updates must include support for that device set.

Do not publish separate apps for iPhone and iPad - create one universal app that covers both form factors and adapt the features/presentation as appropriate.

4.2.2.3.4 Localization

Applications should not be published as different versions for different languages or regions without documented externally verifiable legal justification, doing so is considered as spam by Apple and may cause company-wide suspension of app publication. Internal company requirements/needs do not count as externally verifiable justification. Although there are some undocumented workarounds, one application instance localized to different languages and regions is the required method for handling these situations.

Applications may not arbitrarily restrict features by geographic region - for our purposes we may have good reason to restrict some app content to a particular region, in which case a reason must be presented as part of the application review notes. <https://developer.apple.com/app-store/review/guidelines/#unacceptable>

4.2.2.3.5 Launch screen (splash screen) requirements

All iOS applications must use a launch storyboard to provide the app's launch screen, and not UILaunchImages.

4.2.2.3.6 Dark mode

All iOS applications should fully support dark mode.

4.2.2.3.7 Code completeness/updates

Applications may not download, install, or execute code which introduces or changes features or functionality of the app (<https://developer.apple.com/app-store/review/guidelines/#software-requirements>). This means that vendors who promise automatic updates without having to update an app in the store are presenting a flawed model.

4.2.2.3.8 Webview requirements

iOS apps may not use UIWebView. Apple requires all applications that display website content in-app to use WKWebView instead, with the exception of login/account related processes presenting information to users which must use SafariViewController (<https://developer.apple.com/app-store/review/guidelines/#data-use-and-sharing>).

4.2.2.3.9 Stability

Applications that in Apple's opinion look to be incomplete, which have technical problems, or which crash or hang will be rejected by Apple. Always ensure your production application functions correctly before submission - this means testing is always required on production applications, even if more extensive tests have been performed on development environments, and that production environments must be live before applications are submitted.

4.2.2.4 iOS Privacy And Sensitive Data Features

4.2.2.4.1 Privacy, App Tracking Transparency and Sensitive Data APIs

Applications must only collect and use the minimum data needed to accomplish the specific tasks related to the functionality of the app (<https://developer.apple.com/app-store/review/guidelines/#data-collection-and-storage>).

To increase user privacy, developers are "advised" to use media pickers that limit access to specific media rather than blanket access to media using legacy APIs. While only advised by Apple, at some point in the near future this will become mandatory.

If an app, or any data it collects, including through third party SDKs/services, and including through cookie use on webviews, are possible to be used to track a user then the App Tracking Transparency permissions framework must be used. If a user declines to be tracked, the user must still be able to use the app's core features.

All iOS applications must only use sensitive data APIs provided by the OS if the application's features require it. Info.plist keys must be provided which concisely and precisely describe the app's need and use of the sensitive data feature so that users can evaluate whether to grant the permissions.

An application's use of certain APIs (including by any 3rd party SDKs) as well as any tracking of sensitive data requires a privacy declaration in the manifest in order for the build to be uploaded and submitted see - https://developer.apple.com/documentation/bundleresources/privacy_manifest_files and https://developer.apple.com/documentation/bundleresources/privacy_manifest_files/describing_use_of_required_reason_api. This privacy manifest forms part of the code base for acceptance.

All iOS apps in the App Store must still submit a privacy/data statement for new apps and app updates. This statement is a multiple-choice form and includes a declaration of the information collected in our apps by both our custom code and by 3rd party SDKs, and including any data gathered by in-app webviews. A full list of the data types which we must declare to Apple is available at: <https://developer.apple.com/app-store/app-privacy-details/#data-type>

Since the app's privacy declaration will appear in app store listings and be publicly visible to all potential users, it must accurately detail the app's data collection and usage – including any data collection or transmission by third party SDKs or frameworks.

iOS applications must provide an export compliance key/value pair in their Info.plist indicating the compliance to US/Canadian encryption laws, and App Owners must provide legal documentation if an application uses non-exempt encryption methods.

Apps which use analytics must use the version of Firebase which does not use IDFA. As Firebase collects the IDFA by default, developers must configure their podfile as described at

<https://firebase.google.com/docs/analytics/configure-data-collection?platform=ios#disable-IDFA-collection>.

Applications must implement "appropriate" data security measures (<https://developer.apple.com/app-store/review/guidelines/#data-security>).

4.2.2.4.2 Permissions handling

All iOS applications must continue if permissions are not granted, or if the permission is core to the app's main feature fail gracefully. All iOS applications must react gracefully to permissions being revoked by users.

Explicit permission must be requested using OS APIs before any type of user tracking - including presenting webviews which use tracking or analytics cookies in app. If a user declines to be tracked then the app must still be usable.

4.2.2.4.3 Account handling

Apps that have core features which do not logically require an account should not require a user to log in to use those features.

iOS applications which offer users the ability to create an account **must** allow users to delete that account from within the app. Apps which have accounts and do not have the delete account feature will not be publishable.

Apps which require a login **must** have a demonstration mode that illustrates all the app's features or credentials **must** be provided.

Account credentials for apps that do not have a demo mode **must** be provided and work on production, apps which do not provide these details cannot be published.

Applications which use social network login frameworks (e.g. Google, Facebook, etc.) must also support Apple ID login, and must provide users the ability within the app to disconnect their account from that social media network.

Using a web page for in-app login and account deletion is a poor user experience - account management and login features should provide a native experience.

4.2.2.5 iOS Signing and Identifiers

4.2.2.5.1 Certificate and Profile Generation and Storage

iOS applications must be signed by certificates and profiles generated in our company's Apple Developer accounts under the required bundle ID.

Those certificates and profiles are managed by the mobile platform team.

Certificates and profiles will not be provided to vendors or agencies.

4.2.2.5.2 Bundle IDs

Bundle IDs must be unique in Apple's ecosystem. Once generated a bundle ID is locked to the account it is generated in. Therefore the mobile platform team will generate and manage the bundle ID for applications.

A bundle ID cannot be changed during an application's life.

if an application is developed using libraries or services locked to a bundle ID (for example Google Firebase) then this must be planned for when configuring or licensing those services. If an application requires libraries or services locked to a bundle ID then contact the mobile platform team.

4.2.2.5.3 White label Application Handling

White label applications (applications where we don't own the source code and where vendors supply a pre-built IPA) will be resigned by the mobile platform team under a distribution certificate and profile for our account.

The resigning process for iOS apps allows changing of the bundle ID used, the version number, the build number, and the app name.

4.2.2.5.4 Application names

App names must be unique for the country of release, and should be globally unique in order to avoid future conflicts.

App names that are already in use by others cannot be used by our company unless we submit a trademark/copyright claim.

Our company should own the trademark/copyright of any app names to reduce our legal risk.

Short name variants can be used to display app names under the icon when installed, but must generally match the published name shown in the app stores.

App names must be less than 30 characters long including any whitespace characters, but ideally should be less than 12 to 15 characters. Names longer than about 12 to 15 characters will be truncated when shown on devices depending on device form-factor and user-customizations.

App names should not contain the work "Merck" unless they are to only be available to Merck based regions and users for their entire lifetime. "MSD" is acceptable to use in app names instead of "Merck" as there are no restrictions on using "MSD" in Merck based regions.

4.3 User Acceptance Testing (UAT) and General Pre-Release Testing

Applications must be tested during development, using a development or testing environment where applicable, and re-tested prior to publication by Job Owner(s) in a production environment.

Development environments should match the setup of production environments, with data populated to include the average data expected plus extremes to ensure unexpected events are handled. Application testing must include testing of expected data, unexpected data, that the application functions as expected, that the application is usable, that the application UI and UX are up to company and platform standards, that the application performs its expected functions well, and that any errors are handled gracefully. Details on other recommended aspects to be tested can be found at [General Testing Advice](#) (see page 48).

Budgeting for time and cost of testing prior to delivery, including access to physical test devices, should always be considered at the early stages of a project, and testing on a physical device is required by both Apple and Google in their respective guidelines. Testing on a range of physical target devices prior to publication is also required to ensure that applications function as expected, are stable, are usable, feel natural for users, handle issues gracefully, meet platform (Android or iOS) standards, our company's standards, and for general usability in different situations on devices from different manufacturers, models, and OS levels.

Although applications must always be tested pre-publication on physical devices, due to our company's restrictions on device purchasing, it is also recognized that teams may find difficulty in purchasing or accessing physical test devices. In such cases, although it may lead to quality and usability issues on real world devices (especially with stability and UX/UI) services such as Amazon Web Service's Device Farm, Xcode and Android Studio simulators, etc. may be used as a substitute when physical devices are not possible.

Completed or substantially complete mobile applications which meet Google's and/or Apple's publication standards only may be submitted to App Store (TestFlight) or Google Play (Internal/Alpha/Beta channels) using distribution signed release builds pointing to production environments.

Any testing via TestFlight or Google Play Internal Test channels is restricted by a list of testers, (including Apple ID addresses for TestFlight or Google IDs for Google Play), and should include information on what is to be tested, with release notes on what has changed. Any application which requires users to login distributed via TestFlight or Google Play will also need at least one testing account accessible to Apple and Google which must be maintained for the duration of the approval process.

- [General Testing Advice](#) (see page 48)
- [Distribution Channels Summary](#) (see page 55)
- [Google Play Internal, Alpha, and Beta \(Android\)](#) (see page 58)
- [TestFlight \(iOS\)](#) (see page 58)
- [BrowserStack Testing](#) (see page 59)
- [Test Distribution To Shop Floor Devices](#) (see page 59)
- [Environment Handling During Testing](#) (see page 60)

4.3.1 General Testing Advice



The aim of these guidelines are to promote consistency and simplicity for the user, so that a user of one application knows instinctively how to use another, without being confused by changes of language, labeling or excessive changes of style.

Two keywords to consider when looking at any mobile application are

- 1) **Clarity** – does the application do what it says it will do? Does the application make sense? Are the controls and content of the application easy for a user to understand without documentation. Are the controls clearly labeled? Does the application have a clear purpose? Does it use jargon? Is its language and reasoning simple to follow? Is the application simple to navigate? Is the application interface fluid and fast? Does the application provide clear and understandable warnings/messages when necessary? If the user makes a mistake can they correct it with two gestures/clicks?
- 2) **Consistency** – does the application do what it says it will do 100% of the time? Does the application fit in with the target platform design guidelines? Does it change any interaction which is standard for the platform? Does the application behave predictably consistently and responsibly under all circumstances, even those created outside of its control (networks/third party servers/ etc.)? Are the fonts consistent? Are there any typing/spelling mistakes? Does the app use poor grammar anywhere? Are the images consistently sized? Is the UI consistently themed? Is the application interface consistently fluid and fast? Does it react in the same way in all its pages? Test on as wide a range of physical devices as possible - does the app consistently work and present information as expected on different devices and different form factors?

The exact testing required for any application is dependent on its purpose, features, implementation, and scope. At its most basic the app must do what it is intended to do, do what it says it does in any metadata and descriptions (including those in the app and in any marketing or store descriptions), do it well, and only do it in a way that meets company, legal, and platform requirements.

All apps, whether submitted for TestFlight, Google Play Internal Testing, App Store and Google Play Store, must have valid credentials or a demonstration mode available for Apple and Google to test with. *If these credentials or a demo mode are not supplied or do not work the app will be rejected.* If MFA is implemented, it is recommended to provide a demo mode, if not a demo account in the production environment that bypasses MFA and does not require VPN in order to make Apple's and Google's reviews possible.

General elements to check for:

App icon and name:

- 1) *Is the app icon design relevant to the app content?*
- 2) *Does the app icon display unnecessary information?*
- 3) *Is the app icon clearly presented in all color schemes, including high contrast settings?*
- 4) *Does the app icon have the same look and feel in all color schemes?*
- 5) *Does the app use notifications? Are they necessary? Do they make sense? Does any platform alert badge (e.g. an unread email counter) have acceptable contrast with the icon to users with medical conditions?*
- 6) *Does the app icon contain any unnecessary branding or any advertising?*
- 7) *Is the title under the app icon or in other usages truncated?*
- 8) *Does the app have a unique name? Does the name match the content and intent of the application?*
- 9) *Does the app have unique content that would be of benefit to an ideal user?*

App launch:

- 10) *Does the app start within 5 seconds*
 1. *Start it up, time it*
 2. *Ensure that it is usable **within** 5 seconds*
- 11) *Does the app display a splash screen*
 1. *If just a logo is the logo centered?*
 2. *Is the splash screen content clear and clean, are there any glitches?*

3. *Does the splash screen display any text?*
 4. *Does the splash screen fill the screen at all resolutions?*
 5. *Are there any graphical inconsistencies?*
 6. *If the app uses an extended splash screen it should not be noticeable unless it takes a long time to load necessary resources, in which case it should show a progress ring*
 7. *If the app uses an extended splash screen, suspend the app whilst the splashscreen is showing and relaunch. Does the app re-launch successfully?*
- 12) *Does the app interfere with any other apps or processes?*
1. *Experiment by suspending the app opening up other apps that use the same or similar resources and processes*
- 13) *Does the app resume within 2 seconds*
1. *Suspend the app*
 2. *Switch back to the app*
 3. *Time it, ensure that it is usable **within** 2 seconds*
- 14) *Does the app resume to the same place, and with the same data as when it was suspended?*
1. *Launch the app*
 2. *Navigate the app to show particular data*
 3. *Suspend and resume*
 4. *Check data shown*
- 15) *Does the app start afresh*
1. *if requested (e.g. if app is fully closed and relaunched or through any manual reset process)*
 2. *if a lengthy period of time has elapsed since in the foreground*
- 16) *If the app presents changeable data (e.g. weather etc.) does the data refresh on resume?*

App performance/progress ring usage

- 17) *Does the app experience any lags in performance?*
- 18) *Does the app crash in extremes or during median use?*
- 19) *Does the app present unnecessary errors, alerts or warnings at either extreme or during median use?*
- 20) *If the app accesses external resources (networks, external site content, files etc.) does it display a determinate progress ring or indicator?*
- 21) *If the app relies on downloading large external resources does it notify the user of the size, and obtain consent before doing so?*

- 22) *If the app needs to access services which it cannot be reasonably expected to calculate the estimated time does the app display an indeterminate progress ring?*
- 23) *If the app initiates external processes which would require a determinate progress ring or bar, does the progress ring calculate an accurate time? Does the progress ring reset or reverse at any point?*
- 24) *If the app requires a progress ring, does it calculate all progress in one estimation, or does it separate out each process's progress?*
- 25) *If an app uses resources such as the camera or Bluetooth access does the app meet guidelines on their usage?*
- 26) *Is the app's usage of any resources necessary? Can the user be expected to have at least basic functionality without that resource?*
- 27) *Is the app's usage of resources only when needed for a specific task – for example the camera should only be accessed if the user has requested to take a photo or record a video, an app's usage should start when it is logical – i.e. to actually take the picture etc. and not before it is necessary.*
- 28) *Does the app correctly release access to resources once it has finished with them?*
- 29) *Does the app correctly release access to resources or files if it crashes or is suspended even if the event is caused by an external source?*

App style elements

- 30) *Does the app present a clear and consistent necessary purpose? Does the app fulfill platform guidelines of presenting a unique content, intent or purpose?*
- 31) *Does the app color scheme fit in with the user expectations guidelines, is it too understated? Is it too bold?*
- 32) *Does its color scheme make sense for the application?*
- 33) *Does the color usage vary across different elements or pages?*
- 34) *Is the app's entire color usage from the correct branding palette?*
- 35) *Does the app use correct branding colors?*
- 36) *Does the app contain any spelling or grammar mistakes in either mutable strings or permanent text?*
- 37) *Is any text truncated?*
- 38) *Is the text in the correct case – title case for titles, sentence case for content.*
- 39) *Does the app use single spacing after periods?*
- 40) *Are important keywords in the correct case – for example close/reset buttons use lower case when mentioned in the text, copyrighted or trademarked elements use the correct case.*
- 41) *Is the font usage consistent?*
- 42) *Is any application header text sufficiently far from the left on all resolutions?*

- 43) *Is any content text sufficiently from the left at all resolutions?*
- 44) *Is the content text consistently justified?*
- 45) *Is the text legible?*
- 46) *Is there at least 2px of padding between any touchable elements at all resolutions?*
- 47) *Does other content meet the platform and company padding and layout specifications? At all resolutions and at all sizes?*
- 48) *Does the app, its layout, content and intent make sense?*
- 49) *Does the app's style match its content and target audience?*
- 50) *Is the app's formatting consistent and clear?*

App buttons

- 51) *Are buttons consistently laid out on screen?*
- 52) *Is button usage and layout intuitive?*
- 53) *Are the buttons and controls used as required by platform guidelines?*
- 54) *Are the buttons and controls clearly labeled?*
- 55) *Do the buttons do what they say?*
- 56) *Do the buttons contain too much text?*
- 57) *Are all controls labeled?*
- 58) *Do buttons react visually?*
- 59) *Do buttons react consistently and in a manner that makes sense to touchscreen events?*
- 60) *Are tooltips shown where necessary?*
- 61) *Do the labels make sense?*
- 62) *Do input boxes/text editing areas/etc. have tips/suggestions in them?*
- 63) *Are appropriate controls neatly grouped near their relevant content? At all resolutions?*
- 64) *Are input boxes and buttons necessary for the app's purpose?*
- 65) *Does the application bar obscure any vital content?*
- 66) *If the application bar is opaque are its buttons still clearly discernible and usable?*
- 67) *Is any clickable/interactive area placed too close to the screen edges to be usable?*
- 68) *Are the app's controls or buttons sized appropriately?*

App warnings/alerts/messages

- 69) *Are the warning/error/alert messages consistent with platform themes?*
- 70) *If a dialog asks a question, is it necessary or justifiable?*
- 71) *Are consent and warning messages given for destructive user instigated actions (deleting items, etc.)?*
- 72) *Can a user cancel a destructive action within two gestures (clicks/touches)?*
- 73) *Do any flyout messages close when the screen is clicked away from the flyout? Is any related action canceled?*
- 74) *Are in-line messages given for user resolvable errors such as password mistakes?*
- 75) *Do the in-line messages disappear once the error has been resolved?*
- 76) *Are overt messages given for errors that the user can ignore?*
- 77) *Do overt messages require any user input to dismiss?*
- 78) *If an application has a non-critical but significant error is a message clearly shown to the user?*
- 79) *If an application cannot continue with an action for any reason, does it display a message dialog?*
- 80) *Are error messages concise, clear and non-patronizing?*
- 81) *Do error messages give users suggestions on how to resolve the error?*
- 82) *Do error messages contain any icons/glyphs?*
- 83) *Do any error messages use the word “OK” or similar language in a button? An error is never “OK”.*
- 84) *Do buttons contained within error messages perform as expected?*
- 85) *Are in-line and flyout messages located near the action/input area that caused them?*
- 86) *Are default selections initially set to the safest or most obvious choice?*

App flow

- 87) *Are elements within the app that are marked as required **actually** required to perform an action?*
- 88) *Can a user bypass any required step, action or input?*
- 89) *Do log-in forms work with the supplied test data?*
- 90) *Do in-line errors appear on correctable mistakes (username/password/etc.)*
- 91) *Do the in-line errors appear near their relevant text-boxes?*
- 92) *Does the app bypass/subvert any standard platform interactions?*

App legal elements

Sensitive information includes email addresses, phone number, real name, address, partial address including ZIP/postal codes, email address, user credentials, phone number or any other contact details

as well as credit card information, bank account information, health-care information, or any other personally identifiable information.

93) If a privacy policy, contact address, company, contact details,, statement of terms and conditions, cookie warning or any other documentation is legally required, is it present?

94) Are all privacy policies and terms and conditions, etc. clearly accessible? Can a user reasonably be expected to see them and have read them?

95) If a user supplies sensitive information is it stored in a way that meets both legal and platform requirements? Is the information required to be stored to perform the app's functions?

96) If a user supplies sensitive information is it transmitted in a way that meets both legal and platform requirements? Is the information required to be transmitted to perform the app's functions?

97) If an app allows the user to use an online identity, user name, or social media login is it handled in a way that meets all legal, platform, and security requirements? If it uses a social login does it also leverage Apple SSO and Google SSO as required by those platforms?

98) If an app gives a user access to external content, social media, or any similar content outside of the application does it supply the user with all required warnings before any information is requested?

99) Does an app allow the user to confirm/cancel any social media actions before they occur?

100) Does the app conform to any current platform guidelines on social media usage (e.g. offering Apple login as well as Facebook login)?

Comps

101) Does the app match the comps and expected layouts in all form-factors (if provided)?

General Purpose and features

The exact testing required will depend on the app's purpose and features.

102) Does the app have a clear and unique purpose? What is the problem it aims to solve, and how does it solve this? Does the app merely duplicate another app's features? What makes this different from other apps? Why would a user install this app?

103) Does the app's description and metadata match its purpose? Does the description state clearly what the app does? Do the screenshots reflect this? Do the app name and icon reflect its primary goal? Does the metadata contain any prohibited terms or app store blockers?

104) Do the app's features match the purpose and description? Does the application deliver what it promises to do? Does it do so without creating burdens on the user? Does the app's purpose make sense?

105) Do the features work? Use exploratory testing to check all visible features. If a feature is included it must work, and work in a way that is easily discernible to a reasonable user.

106) Does the source code suggest additional hard-to-access features not visible from the description? If so were they implemented and do these affect the scope of the application? Do these make sense? Do they work?

107) Do the app's permissions requests match the app's features and purpose? Check with scanning tools, or in the app's manifests for all permissions the app requests. If a permission is included does its permission string match what it says it does? Are translations included for all the app's languages? Does the app need to use that permission to achieve its primary purpose? Is the usage covered in compliance documentation and in the in-app legal agreements? Can the user still use the app if they reject or withdraw those permissions?

108) Do all input fields and methods sanitize input? Test with a range of extremes, and injection strings. Ensure that errors are gracefully handled. Ensure that non-sanitized inputs are not passed through to any backend servers.

109) Does the app provide clear and easy access to contact details, privacy, and legal agreements? Are these clearly accessible to the user at all times? Are the details current?

110) Does the app present all required legal agreements prior to use? Ensure the user is not pestered with agreements once agreed, but ensure that there is clear informed consent before using the application.

111) Does the in-app versioning match the planned publishing version numbers? Ensure that users are not confused by version numbers differing from store versions. Ensure that the intended versioning is available (store versions must increment in general, build versions must always increment independently of version numbers for Android, build versions must increment within a version number for iOS).

112) Does the app target the correct SDK levels for the platform (see <https://developer.android.com/google/play/requirements/target-sdk> for Android, or <https://developer.apple.com/support/xcode> for iOS)? If SaaS, can this be updated easily (e.g. using apktool, iReSign, etc.) or is a new build needed? Does this cause any breakages?

113) Does the app (and any related content) meet all other publication requirements from Google and Apple? Are there any possible rejection reasons?

114) If a user can create an account in the app, can they delete that account in the app with clear, easy, simple to understand steps?

4.3.2 Distribution Channels Summary

Pre-submission testing and functional reviews are the responsibility of the business owner, however we provide support to development and business teams to help them with distribution if requested.



Code Status

All apps uploaded to Google Play Testing (Android) or TestFlight (via App Store for iOS) must be substantially complete and meet all Google's and Apple's policies and functional requirements.

A demo mode or valid test credentials are required for any app that requires a login or keycode to access when uploaded to Google Play or TestFlight, Apple and Google will use these credentials or the demo mode in their reviews.

Our team has a number of different test distribution methods, which have different advantages/disadvantages:

Method	Platform	Pros	Cons
Google Play Internal Test Channel	Android	Accessible only to defined users. Accessible to any geographic location. Same bundle ID and build as published app.	Requires approval by Google. Initial review for a new app can take several days. Complete, ready to publish applications only. Requires full store listing. Applications which contravene Google's guidelines can cause entire account to be suspended. Requires Google IDs of all testers. Requires password(s) or test accounts to be distributed to Google. May be blocked by MDM solutions.
Google Play Closed (Alpha) Test Channel	Android	Accessible only to defined users. Accessible only to defined geographic location(s). Same bundle ID and build as published app.	Requires approval by Google. Review can take several days. Complete, ready to publish applications only. Requires full store listing. Applications which contravene Google's guidelines can cause entire account to be suspended. Accessible only to defined geographic location(s). Requires Google IDs of all testers. Requires password(s) or test accounts to be distributed to Google. May be blocked by MDM solutions.

Method	Platform	Pros	Cons
Google Play Open (Beta) Test Channel	Android	Accessible only to defined geographic location(s). Same bundle ID and build as published app.	Requires approval by Google. Review can take several days. Complete, ready to publish applications only. Requires full store listing. Applications which contravene Google's guidelines can cause entire account to be suspended. Accessible only to defined geographic location(s). Publicly visible. Requires password(s) or test accounts to be distributed to Google. May be blocked by MDM solutions.
TestFlight	iOS	Ease of use. Can be configured to be locked to known testers or openly distributed. Same bundle ID and build as published app.	Requires approval by Apple. Review can take several days. Complete, ready to publish applications only. Applications which contravene Apple's guidelines can cause entire account to be suspended. Accessible only to defined geographic location(s). Requires password(s) or test accounts to be distributed to Apple. Requires at least partial store listing/app description. Only available for up to 90 days from initial upload.
BrowserStack	Cross-platform	Ease of use. Allows access to real devices over browser. Allows for testing on a wide range of devices.	Does not fit all use cases/features due to devices being remote. Limited licenses available. Limited log access. Limited control over devices. VPN access using tunneling is possible but not fully explored or supported - will require additional work and approvals. Advised for use when teams do not have particular test devices.

Method	Platform	Pros	Cons
Ad-Hoc	iOS	Accessible by shopfloor devices Bypasses Apple's reviews	Limited to set of pre-registered devices Addition of devices to a profile causes previous distribution profiles to be invalidated Limited number of devices allowed on a developer account Number of devices registered only resets once a year Requires coordination with MIS for delivery

4.3.3 Google Play Internal, Alpha, and Beta (Android)

Android applications must be signed with the release keystore and in release mode.

All Android applications to be submitted for distribution using Google Play Alpha, Google Play Internal, and Google Play Beta should be accompanied by the same assets required for publication, a list of countries specifically for test build availability (in the case of Alpha/Beta testing), and a list of changes made in the latest version. Google bases their policy reviews on all channels for an application, including Internal, Alpha, Beta, and Production. An application that is uploaded to the Internal test channel, even if no version of the application is in production, must meet the same general user protection and disclosure policies as a Production application.

A list of Google IDs for the test users must be provided for Internal Test and Alpha test channels. Beta test channels are for open testing and will be publicly available, these should be only used in limited situations.

4.3.4 TestFlight (iOS)

iOS apps submitted to TestFlight must be signed with a distribution provisioning profile registered to our company's account and must be substantially complete.

TestFlight is only for applications which are **beta or release candidates** (see 2.2 Beta Testing at <https://developer.apple.com/app-store/review/guidelines/#performance> - "Any app submitted for beta distribution via TestFlight should be intended for public distribution and should comply with the App Review Guidelines.") Apple requires any application submitted to TestFlight and distributed to testers to pass a similar review to that which is used for publicly released applications therefore substantially incomplete or non-working applications may be rejected.

iOS applications submitted for Test Flight must be provided with a brief description (a beta version of the store description), a feedback email, a privacy URL and a marketing URL. If special EULA or other licensing information is required to be displayed to beta testers use then this must also be included.

Any iOS application submitted for publication will be reviewed both automatically and manually by Apple and any application which contains features which require explanation, or which implement access control, must have full, concise, politely worded, human-readable testing information. If an application requires login credentials then these must be supplied and work outside of Merck's ecosystem in order for Apple's reviewers to access the application's features.

4.3.5 BrowserStack Testing

In order to provide better test coverage, especially in situations where particular markets may not have access to test devices, the Foundational Platforms' Mobile Platform team can provide access to the BrowserStack AppLive service. This service provides access via a browser to real iOS and Android devices.

This service has some limitations as it is suitable for applications which do not rely on elements which would require physical access to a device such as use of Bluetooth accessories, and we are limited to 10 concurrent users at a time due to licensing restrictions. This number of users may increase if the service is proven to be useful for multiple teams.

Full documentation on the service is at <https://www.browserstack.com/docs/app-live>.

For access please contact fp-mobile-platform@merck.com²⁴.

4.3.6 Test Distribution To Shop Floor Devices

Procedure for distributing test builds of internal ABM mobile applications to shop floor iOS devices for UAT purposes. This process is limited to UAT builds of applications intended for deployment on shop floor devices. Production builds will be distributed using ABM.

4.3.6.1 Key notes

- Distribution via internal TestFlight is not possible due to our company's iOS shop floor devices being configured to not have Apple IDs.
- Apple explicitly do not allow release publication of applications that are not ready for release/distribution.
- Apple's guidelines state to not distribute non-release candidates to external TestFlight, and require internal TestFlight distribution to be only to specified Apple IDs.

²⁴ <mailto:fp-mobile-platform@merck.com>

- Applications intended for use on shop floor devices are required to be tested prior to release.
- It is possible to gather UDIDs from shop floor devices, and add them as known devices to our company's ABM account, with a maximum of 100 devices allowed, with the device count able to be reset annually.
- It is possible to create ad-hoc builds for known devices, and distribute them via our company's MDM solution.

4.3.6.2 Process

4.3.6.2.1 Device UDID maintenance:

- All shop floor devices to have their UDID recorded by MIS using information from MDM.
- UDID to be provided to FPMP for inclusion in Apple Developer Program account.
- Annually any retired devices to be removed from Apple Developer Program account by FPMP.

4.3.6.2.2 Builds/Signing:

- For iOS app builds created by Bitrise for shop floor apps, two signing configurations to be used, creating two ipa files for each build.
 - a. These should have the same version and build number and be created from the same source code/commit.
 - b. ABM distribution – configured for production, this is the release candidate.
 - i. This will be uploaded for ABM distribution, and
 - ii. for use in TestFlight to non-shop floor devices.
 - c. Ad-hoc distribution – this is the test app,
 - i. it should be functionally the same as the release candidate, but may be configured for UAT environments.
 - ii. Provided to MIS for distribution via MDM.
- On UAT passing, the release candidate can be submitted for Apple's publication review.
- On Apple publication review passing, the test app can removed from devices, and the release candidate can be distributed using ABM.

4.3.7 Environment Handling During Testing

If your mobile app is designed to leverage different environments during testing (for example a development server as well as a production server), then in order to avoid confusion with production release candidates we will set up a separate app listing instance to test your application. Having separate environments for release testing is optional depending on your application's exact requirements, data handling, features, and architecture. The separated listings for alternative environment must not be submitted for full release to the app stores (including ABM), and will only be available via TestFlight or Google Play Internal Test channels. Applications on all environments must

meet the same publication standards for passing TestFlight and Google Play reviews as production applications.

4.3.7.1 When is a separate test/QA/UAT environment needed?

If your application has a backend that handles data that has been flagged by our company as requiring a separate test environment (i.e. if you've been told to).

If submitting test requests from your client mobile application to your back end could potentially corrupt production data or disturb production operations.

4.3.7.2 When is a separate test/QA/UAT environment not needed?

If your application doesn't have a back end or server.

If the only remote server the application leverages is a content management, translation, authentication or similar server with all other operations being handled in the client.

If the application only leverages a server for read operations.

If all test data sent/received from the application can be kept separate from production users by user management processes.

4.3.7.3 Production testing

Even if testing against a test environment pre-release, all production release candidates **must** also be tested in the production environment before release to help ensure quality, stability, and adherence to Apple's, Google's, and our company's policies.

Apple and Google both require a demonstration mode or a test user in production in order to review an application before publication, therefore solutions must be designed to support production test access.

4.3.7.4 Naming conventions

In your solution you should set up schemes (iOS) and/or flavors (Android) to match the environments your application uses with the app name being set matching that environment, and when creating tickets you (or your development team) should state explicitly which environment you would like builds created for/uploaded to testing.

These listings will be named as in the examples below (relevance depends on your solution architecture, not all will be required):

Environment	App Listing Name	Scheme or Flavor name
Production	App Name	App Name
Test	App Name TEST	App Name TEST
UAT	App Name UAT	App Name UAT
QA	App Name QA	App Name QA
Staging	App Name Staging	App Name Staging

Applications that are currently using one combined listing for production and staging/QA/etc. builds will be moved to separated listings on their next request.

4.4 Summary of deliverables

For mobile applications to be published in the external app stores (Google Play/Apple App Store) and for internal app stores (Apple App Store as a "Custom App" delivered via ABM), all source code/app packages and accompanying deliverables should be transferred when submitting a publication request. Apps submitted for publication cannot be submitted to Apple's or Google's review until all compliance and deliverables requirements are received and meet Google's and/or Apple's requirements.



General Deliverables Requirements

The information below are the standard common requirements - specialized applications may require further elements.

Marketplace assets and copy can be supplied using the Visual Store Mockup tool at <https://go.merck.com/vsmu>.

Images supplied embedded in Word documents, Excel files, PDF files, etc. are *not* usable. Graphic assets must be separate files, in the sizes as documented below, in png format, with no transparency.

Any other supporting elements which may not be necessary for building or publishing the application but which are available (such as test notes, build notes, FAQs/caveats, technical contacts, environment details, legal supporting documentation, etc.) should be included.

If any part of the application collects data (including anonymous analytics and crashlytics) the documentation must include a brief description of what data is collected (including by third party libraries), what that data is used for, and should also include a summary of any other usage of features such as location, background location, SMS/phone access or log permissions, etc. For more information on Apple's App Privacy details, visit: <https://developer.apple.com/app-store/app-privacy-details/>, and for more information on Google's Data Safety details see <https://support.google.com/googleplay/android-developer/answer/10787469>.

If any part of the application has a log-in feature or passcode then a demo mode or credentials for this which are valid for production must be provided and be usable by Apple and Google without any VPN or MFA requirements. Applications which require any sort of log-in cannot be published without valid credentials or a demo mode. Details provided should also include what features and additional benefits the log-in provides to end users.

Applications which are intended for use solely by our company employees or contractors and which serve no purpose for the general public/HCP/HCC, should be published as ABM / custom apps and not released to the general public.

Any missing deliverables will cause a delay in app publication and may cause rejection by Apple or Google.

App builds will be checked prior to submission, however this is not a full QA phase and we will not make code changes or fixes. A full QA/QC cycle must be performed before submitting apps for publication. A brief summary of aspects to check is available at <https://go.merck.com/prereleasetestadvice>.

4.4.1 General Deliverables Requirements

Deliverables **must** include:

4.4.1.1 1) Source code

1. If Merck owned or created code then buildable and working source code, including any libraries and dependencies must be provided.
2. If non-Merck owned source code (e.g. white label applications) then ipa or aab files must be provided which are suitable to be re-signed with our company profiles or keystores.
3. Source code must not contain whitespace or non-ASCII characters in filenames or in internal executables.
4. Source code must have been tested to build and to meet company, platform, and project functional and compliance requirements.

5. Source code must build using current versions of IDEs on current OS levels.
6. Full configuration and build documentation must be included.
7. If an application uses Node as part of its build system (such as React-Native apps) it should be a suitable current LTS version.
8. iOS apps must build with currently released Xcode and iOS SDK versions.
9. Android apps must have a targetSDKversion which meets Google's targetSDK policies.
10. If the app requires a login/other credentials, developers should implement a demonstration mode which exhibits the app's features without the need for a login.



Source code organization

Source code should be clearly organized, use consistent naming conventions, should include all necessary elements and information to build on a clean system, and should not include any extraneous files (e.g. bin/obj/temp folders).

Automated platform specific output from Unity/.Net MAUI or other similar tools is not generally considered to be full source code. Xcarchives are not acceptable for iOS app publication and are not considered full source code..

If specific IDE, library, or node versions are required then these must be specified in the application's [README.md](#)²⁵ file.

If licenses are required for libraries/dependencies/assets then either pre-licensed distributable versions of these or licenses will need to be provided. Business owners must record all licensed content use in their project documentation, and ensure that those licenses are paid for during the entire life of the app. A failure to properly meet any licensing requirements for any content used in applications (including fonts, libraries, images, etc.) may cause legal action against our company. If Firebase is used for the solution then the configuration file must include the publication bundle ID/ package ID in order to function.

4.4.1.2

2) Supporting elements

1. build instructions,
2. feature list,
3. wire-frames,
4. full details of the application's data use and privacy standards,
5. a list of any licensed libraries or APIs linked to the app's ID along with licenses for those libraries (NB info on paid libraries should also be in project documentation, approved financially, and fully paid for during the life of the app)

²⁵ <http://README.md>

6. if the app has a passcode or log-in valid credentials (a demo mode is preferred) and account details, along with a description of what purpose the log-in serves, how a user gets an account, etc.,
7. marketplace assets (screenshots, feature graphics and app store icons),
8. marketplace copy (description, category, EULA, support URL, support email, privacy policy, etc.),
9. attachments or externally accessible links to regulatory or other supporting information (required if the app has received external regulatory approval, has any regulatory requirements, or could be considered by Apple or Google to need approval),
10. if the app provides any type of medical features (drug advertising, dosage calculators, etc.) supporting proof that we are the legal owner/manufacture of those products,
11. details of ongoing app maintenance plans to be able to present evidence to Apple and/Google that the app will be maintained (required for apps which Google/Apple consider in their sole opinion to be medical, advised for others),
12. full details about the app's collection and/or transmission of data to us and any third parties in order to meet Apple's and Google's privacy requirements (required to be publishable, must be accurate and includes data transmission for analytics use), including how that data is processed/anonymized/deleted etc.
13. supporting information to justify use of any features such as non-Apple login, location, sensitive data APIs, etc.
14. details of why a login or code is required, and how that login/code is distributed to users, along with independently verifiable legal citations if the login must be restricted to certain groups (Apple prefer publicly released apps to be available to use by the public and require that if access is restricted that it is only restricted for legitimate reasons),
15. if the app allows for account creation then a description of how the app handles account deletion in the app, including any legal citations/regulatory requirements that may require us to have additional account deletion steps
16. if the app is for Android and allows for account creation then a URL for a website where a user can delete their account must be provided



Test Accounts

Test account details must be provided if the app has any login features, including passcodes or lot-codes. These must be valid for the production environment and must be kept live for the life of the application. If an app uses 2FA then Apple requires the test user to bypass the 2FA mechanism.

Applications which require login or passcodes and which do not provide a valid login or passcode for Apple's and Google's testing in production will be rejected by Apple and Google. A demonstration mode is strongly encouraged and preferred as an alternative.

4.4.2 Screenshot and Icon Specifications

4.4.2.1 1) General

1. Screenshots must show in-app functionality (i.e., something other than a splash screen).
2. Screenshots must be representative of the app's functions and features.
3. Screenshots must accurately represent the application's content.
4. Screenshots must not contain any overtly test content (content which could be convincing as real world use is acceptable, content which shows words such as "TEST" is not).
5. Screenshots must not contain content which could indicate the application has unfinished or unimplemented features.
6. To meet our company's legal requirements, you must usually ensure that company and product specific branding is absent from screenshots. The keyword "Merck" must never be visible in non-US markets.
7. Screenshots should not display information that shows any real user's information or sensitive information.
8. App icons should not consist only of our company's beacon logo.
9. App icons must be unique and relevant to the application's features.
10. Files should be provided in .png format.

4.4.2.2 2) iOS

1. Icons
 - a. App icons should **not** contain rounded corners, the iOS App Store applies a mask that rounds icon corners automatically.
 - b. App icons must be 1024x1024 px @1x, flattened with no transparency, using an SRGB or P3 color space.
 - c. The 1024x1024 icon must be included in the ipa's icon asset catalog as well as supplied separately.
 - d. Asset catalogs must include all relevant icon sizes.
 - e. App icons must follow the rules and guidelines at <https://developer.apple.com/design/human-interface-guidelines/ios/icons-and-images/app-icon/>.
2. Screenshots
 - a. Current required and optional screenshot sizes along with the device names for the sizes are listed at <https://developer.apple.com/help/app-store-connect/reference/screenshot-specifications/>
 - b. You must provide a set of screenshots, a minimum of two in each of the required sizes for the application's platform.
 - c. If the application supports iPad then the iPad screenshots marked as required below must be supplied.

- d. If the application supports iPhone then the iPhone screenshots marked as required below must be supplied.
- e. If the application is Universal then both iPhone and iPad screenshots marked as required below must be supplied.
- f. You must not add rounded corners to screenshots.
- g. A maximum of 10 screenshots per device size are allowed.
- h. Promotional images featuring a screenshot within a device frame must match that screenshot size's device appearance (e.g. the frame for 1242x2688 must be have the notch as in an iPhone XS Max, 1242x2208 must be iPhone 8 Plus not have a notch). Use of device frames in store screenshots is not recommended by Apple, and apps will be rejected if they have the wrong device.
- i. Screenshots must display real-world appearance and not display any overt "test" information (e.g. a user named "Test").

4.4.2.3 3) **Android**

1. Icons

- a. App icons must be in 32-bit png format, in sRGB format, and measure 512x512 px.
- b. App icons must not contain transparent elements or rounded corners, Google Play will dynamically apply a 20% radius onto icons.
- c. App icons must not contain shadows.
- d. Application icons must follow the rules and guidelines specified in <https://developer.android.com/google-play/resources/icon-design-specifications>.

2. Screenshots

- a. Current required screenshot specifications for the Play Store are listed at <https://support.google.com/googleplay/android-developer/answer/9866151?hl=en#zippy=%2Cscreenshots>
- b. A minimum of 2 and a maximum of 8 screenshots are allowed for phones.
- c. If the application supports Android tablets then a maximum of 8 screenshots are allowed for tablets.
- d. If tablet screenshots are not provided the app will not be eligible for Google Play's "Designed for tablets" section.
- e. Screenshots must measure a minimum of 320 px on the shortest side and a maximum of 3840 px on the longest side.
- f. The longest side for screenshots can't be more than twice as long as the minimum dimension (maximum aspect ratio of 2:1, preferred aspect ratio of 16:9 or 9:16, NB Google may restrict this to only 16:9/9:16 in the near future).
- g. If a device frame is used this must be representative of a real-world Android device. Device art can be generated at <https://developer.android.com/distribute/marketing-tools/device->

[art-generator](#). However use of device frames in store screenshots is not recommended by Google.

- h. A feature graphic image measuring 1024x500 px is required.



Icons

We will use the 1024px by 1024px application icon to generate the Android icon (unless Android uses a different icon entirely) and may use it to re-generate any required icon sizes which may be missing from within the application's source code.

Since required screenshots have different aspect ratios if these are missing and cannot be readily re-created then publication may be delayed until they are available.

Due to Apple's requirements iOS applications cannot have their screenshots or the store listing changed after publication without a new build (which must meet all current publication requirements at the time of submission).

**** Google and Apple both consider store listings an equal but separate part of app publication and may reject apps that do not meet their minimum store listing requirements and policies, regardless of actual app quality. Applications cannot be published unless all required assets and metadata are provided and meet Apple's and/or Google's requirements. ****

4.5 Other Standards

- [Analytics and Crashlytics](#) (see page 68)
- [Bitrise](#) (see page 69)
- [Code Quality](#) (see page 70)
- [Data Minimization](#) (see page 71)
- [General Standards](#) (see page 74)
- [Guidance on updates](#) (see page 79)
- [Localization](#) (see page 80)
- [Repository Management](#) (see page 82)
- [Secrets Handling In Source Code](#) (see page 83)
- [Vulnerability Scan](#) (see page 85)

4.5.1 Analytics and Crashlytics

All applications owned by our company which implement analytics must use an analytics account owned by our company.

All applications which implement analytics or crashlytics must declare its use to Apple/Google as appropriate.

All applications which implement analytics will use **Google Firebase**, and app owners will need to contact the **Digital Measurement & Insights Team** to arrange a Firebase instance and configuration file to be used in the solution, event/screen-view implementation advice, as well as access for users to retrieve analytics data.



Firebase Configuration Files

If Firebase is used in the application, including for non-Analytics purposes, the configuration file in the application and the Firebase project *must* contain the bundle ID used for publication. Firebase services will not work if the bundle ID and configuration file/Firebase project IDs do not match.

The Digital Measurement & Insights (DMI) information can be reached at:

<https://go.merck.com/datmobile>

Analytics and Crashlytics Firebase features have had approval through the DMI team, any other Firebase features will require risk, privacy and other compliance approvals. If other features are needed please discuss with the DMI Team at the planning phase, before development work is started.

4.5.2 Bitrise

Our company uses Bitrise for automated builds. This system is currently only available for internal users, however developers can check in code which leverages Bitrise as a build service and for pull request reviews. Access is planned to be expanded to external developers soon.

Solutions must be architected to meet our company's requirements, the latest Apple/Google standards, and be compatible with Bitrise's system by:

- Use of shared schemes (iOS)
- Preferred use of native code (Kotlin and Swift)
- Must build on a clean system
- Must build on ARM based macs (M1/M2)
- Must not use private repositories for libraries
- Agencies should not use paid license libraries without including them in source as pre-built binaries
- Must build with the current release version of Xcode or Android Studio

To onboard your project onto Bitrise, see [\[BITRISE\] Request tickets](#)²⁶.

4.5.3 Code Quality

Developers must write code to be clearly legible and maintainable, following a few set principles:

- A README.md file must be included that explains what a solution does, along with the steps to build/test/implement it
- Comments should be used sparingly when needed to explain logic or functionality
- Code should be self explanatory, objects and variables must have consistent and meaningful names
- Code and logic should not be unnecessarily duplicated (Don't Repeat Yourself)
- Code should be logically grouped with consistent indentation
- Line lengths and file sizes should be limited
- Use US English for comments and object names
- Perform unit and other tests before handing code to our company
- Use linting solutions (SwiftLint etc.) to ensure code is legible
- Wherever possible avoid the use of the keyword "merck" in object/variable/etc. names and in comments

Developers are expected to use the standard language conventions for their chosen technology:

Language	Style Guide
Swift	https://google.github.io/swift/ and https://www.swift.org/documentation/api-design-guidelines/
Kotlin	https://kotlinlang.org/docs/coding-conventions.html and https://developer.android.com/kotlin/style-guide
TypeScript (React Native)	https://google.github.io/styleguide/tsguide.html
JavaScript (React Native)	https://google.github.io/styleguide/jsguide.html
C# (.Net MAUI)	https://learn.microsoft.com/en-us/dotnet/csharp/fundamentals/coding-style/coding-conventions
C# (Unity)	https://unity.com/how-to/naming-and-code-style-tips-c-scripting-unity

²⁶ <https://share.merck.com/spaces/FPMP/pages/1187814940/BITRISE+Request+tickets>

4.5.4 Data Minimization

To meet legal requirements, our company's policies, as well as to meet Google's and Apple's standards, all applications must adhere to data minimization practices. Only data types relevant to the application's *core* features can be required. Other data types may be requested from users but only as optional elements. Users must be able to make fully informed choices on whether to share data with our company.

Apple and Google, as well as various governmental entities around the world are highly focussing on user privacy and data subject rights. The less unnecessary data our company collects, the less security and legal risk we have.

Example scenarios

A dosage calculator app that is published using publicly available medication leaflet information cannot require that a user enters their ZIP code to use it, or have location permissions, although it must include citations on the source of information.

A dosage calculator app published in a region where we are legally required to verify a user to be over 18 can have a process to require a user to register and to validate their age, if we provide citations as evidence of that requirement to Apple/Google and in the app, keep the user informed of the process and their data rights, and ensure the data collection, registration, and validation process is fully consensual. A dosage calculator app published in a region where we are not legally required to verify a user to be over 18 to access the information cannot collect the same data or have a registration process.

A supply chain verification application cannot require a user to provide their precise location, unless it is legally required and unless it is made clear to the user why this is required information to perform the core features of the app.

A veterinary records app that tracks pet vaccinations could ask the user for their phone number as a convenience for sharing with a vet, however as that information is not related to the core features of the app the information must be optional.

A supply chain verification application that has a core feature which allows users to navigate to a nearby pharmacist to consult with them on the medication batch they scanned can allow a user to provide their location as it is a core feature of the service, as long as the user is kept fully informed of the data accessed, what is done with the data, and how they can delete it if stored, however a well designed app in this example should also allow for a user to have an alternative system where they manually enter a location.

A farm records management application can ask a user to create an account using their email address and to verify their farm address or licensing details, however it cannot ask the user for their date of birth or additional contact details unless they are used to perform the core functions of the application.

An application that shows publicly available addresses of pharmacists in a city could ask for the user's exact location in order to provide route-finding features from their current location, but if the core features are just to list the pharmacists in a city it must be able to function with minimal data such as by asking the user to manually enter a city/postcode/zip code/etc.

A symptom diary that allows a user to upload a photo recording their condition to show to a medical professional at a later date can ask a user for access to their photos, however this must be done using iOS' or Android's photo access APIs which are designed to allow users to limit access to particular images rather than by asking for full access to the user's files.

A customer facing application that has a good reason to require a user to create an account, but not a good reason to collect a user's home address, can ask the user for their home address and make it optional for the user to provide it, but it cannot have design aspects that make it look like it is mandatory to provide, and must be clearly marked as optional.

Company policies

[Corporate Policy 20 \(Privacy and Data Protection\)](#)²⁷, states that we process personal data only in line with the principles of 'Fairness' (i.e., Only collect and process Personal Information in ways that are fair to data subjects) and 'Purpose Limitation' (i.e., Only process Personal Information in a manner compatible with the original legitimate business purposes defined for the process).

CPOL 20 is implemented with the help of the Privacy Rulebook – the extension of the CPOL20. The Rulebook Standards take the CPOL20 requirements a step further and provide more granular requirements that must be met to ensure compliance with data protection laws and regulations and with company policies that may be more stringent. One of the standards talk to the rules of data collection ([Data Collection](#)²⁸ v1.18 - released 29 Sept 2023). Data Collection standard defines the requirement of Data Minimization in more detail:

Section 2.5 Purpose states:

2.5.2. Only relevant, adequate, and not excessive Personal Information can be collected.

2.5.4. The Company will only require information that is necessary to complete the transaction.

Apple policies

Appstore guidelines <https://developer.apple.com/app-store/review/guidelines/#data-collection-and-storage>

5.1.1 (iii) - states that data accessed must be related to the core functionality.

"Apps should only request access to data relevant to the core functionality of the app and should only collect and use data that is required to accomplish the relevant task."

5.1.1 (iv) – states that if data isn't required to make it optional, and to not employ design practices to coerce users to provide it.

"Access: Apps must respect the user's permission settings and not attempt to manipulate, trick, or force people to consent to unnecessary data access."

²⁷<https://webview.merck.com/webview/getContentById/Corporate+Policy+20+Privacy+and+Data+Protection?DocbaseName=medsprd&version=CURRENT&format=pdf&ObjectId=0902e71d8ad50b47&cid=0902e71d8ad4cd53>

²⁸https://collaboration.merck.com/:w:/r/sites/mporulebook/Standards_Specifications/Data%20Collection%20Standard/Data%20Collection%20Standard.docx?d=wfd6b6cff4abe495bb3a24e9c0110cfc9&csf=1&web=1&e=PzrzXB

5.1.1 (v) – states that if we are legally required to collect a piece of data that we can do so.

“Apps may not require users to enter personal information to function, except when directly relevant to the core functionality of the app or required by law.”

Google policies

Google's core app quality policy requires that applications are designed to minimize data collection to only that which is necessary - <https://developer.android.com/docs/quality-guidelines/core-app-quality#sc> and <https://developer.android.com/privacy-and-security/about>.

Deceptive Design

There have been many cases of companies facing and losing lawsuits based on their use of deceptive design practices (also known as "Dark Patterns") in order to gather user data. Deceptive design practices must be avoided.

Be clear to users about what data they are providing and why it is needed, and only request the data that is needed for a user to achieve a goal. Do not use design choices or wording that attempts to trick or force users into providing more data than is necessary to fulfill the core user facing features of the application. User satisfaction in our apps affects how people see our company - a user who is fully informed (and therefore confident in using an app and/or providing data in an app) is more likely to be a happy user, a user who is confused about what will happen to their information may feel that an app (and therefore our company) is unnecessarily intrusive and is more likely to become a dissatisfied user.

For more information on deceptive design practices, see <https://www.deceptive.design/>

Security practices

OWASP – Data minimization <https://mas.owasp.org/MASVS/controls/MASVS-PRIVACY-1/>

Legal requirements

Legal requirements vary by region and by time. Many countries have legislation that requires us to handle data safely, including requiring explicit consent, and minimizing what data is collected to that which is reasonable and/or that which is required to fulfill the main purpose of the service, whilst some countries have legislation that requires companies operating in licensed industries to restrict access to particular users which would require collecting information to validate the user that may not be relevant to the core features of the application. Consult with legal and/or privacy teams at the design phase before designing a system that requires data collection from users, and ensure that if we are required to collect data by a region's laws to validate access to an application's features that the reference to the particular law requiring it is included in information to both users in the app before they provide that data and to Apple/Google on app submission. Our company desiring information to be collected on users does not have the same standing as a legal requirement to validate particular data elements

4.5.5 General Standards

Mobile applications should meet the below as minimum standards:

- **Package architecture**
 - when built, Android aabs cannot be larger than 150MB per architecture.
 - when built, iOS executables must not exceed the limits stated in <https://help.apple.com/app-store-connect/#/dev611e0a21f>.
 - Android applications must support 64-bit native code as required by Google in <https://developer.android.com/distribute/best-practices/develop/64-bit>.
- **Versioning**
 - Use a *minimum* of one element ("n") and a *maximum* of 3 elements ("n.n.n").
 - Even though Android allows it, do not use non-numeric characters (other than periods) in version names as it is not fully supported in dependent systems ("n.n.n" is good, "n.n.n:mrklse-01.34+" may break dependent systems).
 - iOS uses the version number as the primary key for versions, with the build number resetting for each version string. Release candidates inside a release train are identified using the build number. For faster TestFlight review times of release candidates, maintain the same version number and increment the build number until the candidate is published.
 - Android uses the version code (aka build number in iOS) as the primary key for versions which must increment for every build independently from version names, however the version name (aka version number in iOS) must also increment for each published release. Android version codes must be an integer.
- **Content**
 - The keyword "merck" must never be used on its own outside of the USA or Canada without the full company name and a geographical signifier ("Merck & Co. Inc., Rahway, NJ, USA").
 - Applications must do what they claim to do, and do it well.
 - Applications which only re-present content that is already available on our website(s) cannot be published - apps must have their own features and purpose.
 - Apps must be "app-like" - the core features must have some purpose for being an app. If the concept or core features are more suited to be a website, an ebook, or other medium, Apple or Google may block publication.
 - User controls, images, text and other content should never clip or be clipped by any other content.
 - All controls should be accessible, responsive, touchable and usable.
 - Controls which are not usable or applicable on specific pathways should be removed from views.
 - Important controls should be called out or emphasized, in a manner consistent with the application's design semiology and the target platform's design semiology.

- Where a user has the choice between a control which initiates a dangerous action and a safe action, the safe action should be the default action, ensuring that users must explicitly choose to take a dangerous action.
- Content should be designed to be visible and usable on all form-factors available on the target OS, using distinct view layouts when necessary to correctly handle extreme form-factors.
- All applications must make a privacy policy, EULA, copyright statement, and version number accessible to a user under an "About" view. The "About" view should be clearly labelled and must be accessible to a user from any other view in the application.
- Versioning must be consistently maintained in the application source code.
- Versioning metadata must be coherent, indicative of changes performed.
- iOS Applications must never use hidden OS APIs.
- Application content must meet all platform guidelines and requirements.
- Applications should not contain unnecessary content and language (e.g. avoid filler language, self promotional language, reference to other platforms).
- Content should be clear and concise.

- **Fonts**

- Applications should use our company's official Invention font, an OS font, or fonts for which we have an agreed license.
- App owners are responsible for business justification, securing licensing, and budgeting for any fonts which require licenses during the entire life of their application.

- **Image assets and icons**

- Images must be provided in png format.
- Icons must not have rounded corners.
- Icons must not have an alpha channel.
- Icons should avoid including text and small detail.
- Icon designs should clearly represent the application's product, purpose or functionality.
- iOS applications must include a 1024px x 1024px icon.
- Icons must not consist of only the Merck or MSD icon.
- Icons must be unique for the application.

- **Application launch**

- Applications should aim to launch and be responsive within 5 seconds.
- Applications should aim to resume and be responsive within 2 seconds.
- Applications which appear to be unresponsive after a reasonable time during launch will not be publishable.

- Applications must present a well-designed splash screen on launch, indicative of the application's style and purpose.
- Applications which require additional processing in order to be usable after the splash screen has been dismissed by the OS must present an extended splash screen displaying a progress indicator providing users information about the process in progress.

- **Interaction**

- Applications must never perform any work on the UI thread or block the UI thread.
- Applications must perform any network actions or CPU intensive processing on background threads (for example content must be downloaded on a background thread).
- Applications should always be responsive to touch and input.
- Applications must allow users to cancel long running or resource intensive tasks (for example downloading content).
- Applications must display progress indicators on long running or resource intensive tasks, preferably determinate progress indicators, giving users information on the amount of time remaining.

- **Network**

- Applications should ask the user for permission before making any network calls.
- Applications must explicitly ask the user for permission before making long running or intensive network calls (for example downloading content) or before transmitting any potentially personally identifiable information.
- Applications must explicitly notify the user of download sizes of additional resources required, and prompt users to continue before initiating downloads.
- Determinate progress indicators should be used to indicate download progress and estimated duration/time remaining.
- Long running or intensive network calls and downloads must be cancellable by users.
- Applications must always use encrypted connections (for example SSL) for web API and other network connections.
- iOS Application Transport Security should never be bypassed.
- In cases where an essential API or external content is only available over http, an exception explicitly stating the scope of use and scope of data must be documented and approved at review stages.
- SSL certificates for target web APIs must be provided by a reputable 3rd party certificate authority, and be maintained to be valid for the lifecycle of the application.
- API and network usage must be secure by design (for example using OAUTH or AD methods). OAUTH over SSL using regularly expired tokens is strongly encouraged for any web API access.

- **Data**

- Applications must only request data which is essential to their features.
- Applications must only transmit or receive personal information which is essential to features.
- Data must always be stored securely, for example by using encrypted storage or password protected databases.
- Personal information must never be stored or sent in plain text.
- If potentially PII, data which is stored on an external server or sent to a third-party must be de-personalized whenever possible, unless explicit user permission is obtained.
- All local privacy and data standards should be adhered to. If an application serves multiple markets with varying data storage standards then the application must be engineered to meet/exceed the most stringent data standards of the target markets.
- Applications must allow users to erase their supplied data.
- Applications should allow users to erase any cached content managed by the application.

- **Reliability**

- Applications must be resilient by design.
- In cases of exceptions, applications must be designed to fail gracefully, allowing users to continue if at all possible.
- Applications must provide clear, concise, user-friendly information in any error messages or warnings.
- Error messages or warnings must always be relevant to the exception which caused them.
- Applications must always check for null references before using objects.
- Applications must disable debug and console output in release configuration.
- In the event of a crash an application should not become stuck in an infinite loop on re-launching.

- **Maintenance**

- Applications that are to be published for longer than 6 months to a year must have a maintenance plan in place to supply regular updates at least once a year to meet OS changes.
- Maintenance plans should include fixes for vulnerability, security, stability, or operational issues that are discovered during the planned life of the app.
- Applications that are not regularly maintained will be removed by Apple/Google, and jeopardize both our company reputation and our account(s).

- **Permissions**

- Applications must only request those permissions which are required for their features.
- Application's features and permissions must be as documented.
- If a permission is revoked or not accepted by a user the application must handle this condition gracefully, and must not crash.

- If a permission is revoked or not accepted by a user the application should continue to function in a limited capacity if possible. If successful application functioning is completely prevented by the revocation of permissions then it must fail gracefully.
- Permissions descriptions strings must be included for iOS in the application's Info.plist.
- Permissions requests presented to users must concisely, yet fully, describe what the user's data is being used for/why the permission is needed.

- **Account management**

- If the features of an application can be used without an account then they should be accessible without an account.
- If an account is required to use an application's features the user must be able to review and must explicitly accept any privacy and terms and conditions specific to the application and account management service before they log in or create an account.
- If a user is required or able to log in to use features, they must have the ability to log out from within the application.
- If account features are managed by a remote server a website must be provided for the account to be deleted.
- The user must have the ability to erase their account content from within the application if they have the ability to create an account in the application (with re-authorization if necessary).
- If an account is needed for any part of the application a demo mode (preferred) or valid credentials must be supplied for Apple's/Google's review using the production app without any VPN or MFA. A demo mode is acceptable in lieu of providing credentials.

- **Location**

- If a user's location is not required for the features of the application it must not be requested by the application.
- If a feature merely requires a user's general location (for example their country) then applications should use IP or device locale (preferred) based logic rather than GPS based location logic.
- If a user's location is required and cached by the application then the application must provide the ability for users to erase this cache.
- A reason for accessing the location must be presented to the user which matches the applications features and the user's expectations, along with examples of how this data is used.
- Location may not be accessed when the application is in the background unless the core aim and features of the app require this (e.g. a route-finding app).

- **Localization**

- Applications should be localized or localization ready, leveraging standard practices for the target platform(s).

- The same application should not be published as separate instances for separate markets unless there is a verifiable legal need to do so (e.g. local country laws).
- **Target Devices**
 - Applications should aim to support both phone and tablet layouts for the target OS (e.g. iPhone and iPad, Android phone and Android tablet). Do not submit separate apps for tablet and phone - create one app that supports both.
 - Applications must be built to *support* the most commonly available version of the target operating system in the target release regions.
 - Applications should *target* the most recent available version of the target operating system in the target release regions.
 - Applications must be *designed* to support current device form factors for the target OS.
- **Privacy, Copyright, and EULA**
 - Privacy, copyright and EULA statements must be included in applications, must always be presented to users on first launch, and must be accessible to users from any point in the application.
 - More information about Privacy, EULA, and copyright policies can be found in Section 2.2.3.
- **ISO/Other standards**
 - Health or medical related apps should aim to exceed any ISO, ANSI, or BSI standards in place at the time of development (see <https://cmhaff.healthservice.com/>, <https://shop.bsigroup.com/forms/PASs/PAS-2772015/>, <https://www.iso.org/standard/78182.html>, etc.)

4.5.6 Guidance on updates

OS Updates

Applications must be re-tested on operating system updates before those operating system updates are released. Both Google and Apple make pre-release versions of operating systems available to registered developers well in advance of public releases for this purpose. Applications which are broken by, or have their behavior changed by operating system updates present a risk to our company in terms of reliability and reputation.

Applications must meet all Apple or Google requirements to be published, including being maintained when that guidance or operating systems change.

Library, Framework, and SDK Updates

Libraries, frameworks, and SDKs used within applications must be updated on a regular basis in order to continue to meet security and platform standards. Libraries that are unmaintained pose a risk that applications will be removed by Apple/Google due to incompatible features, as well as containing unfixed security issues. If a particular library that is used by an application is abandoned by its developers then an alternative must be found. Additionally a company publishing an application is responsible for the actions of its libraries - if a library contains code that violates security or data collection requirements then the library must be fixed or an alternative must be found.

Non-maintained Applications

In order to ensure that applications remain live where requested, regular updates and maintenance must be planned for and performed on applications. Apple and Google will conduct periodic reviews of all apps published and will remove apps that no longer function or do not meet the current publication guidelines. If OS updates break or change an application's behavior, or if an application has not been updated on a regular basis, then it may be removed from the store by Google or Apple during their periodic reviews, including internal applications published via Apple Business Manager.

Non-maintained applications present a risk to our company and may be removed due to abandonment, inability to identify an owner, unresolved security issues, or content issues.

App Compliance Audits

Individual divisions undertake regular audits of applications to identify compliance issues as well as functional or technical issues, any updates required to correct those compliance issues (such as text changes in apps or store listings) may require technical updates to applications themselves to meet current Apple and Google standards. Regular maintenance of applications must be planned during an app's lifecycle to be able to resolve non-technical compliance issues.

App Health Checks

Google automatically monitors application health and will actively remove applications from the Play Store that have a crash rate or application not responding (ANR) rate that exceeds a certain threshold. Google sets this threshold and may adjust it periodically. When planning updates, developers should review reports to ensure that any crashes or ANRs reported by Google are resolved.

4.5.7 Localization

For publishing in both Apple's and Google's app stores it is always preferable to localize a single application for multiple markets, however regional laws and standards in the medical or pharmaceutical industry may require separate applications to be published for separate markets. If an application exists already for the product with the required features but is just available in a different market, always consider localizing the existing application rather than publishing a country specific version of the same application with a different name.

Apple and Google consider publishing the same application under different names to be spam unless there is a verifiable reason to do so and have the right to ban our company from app publication on the platform entirely. *If an application cannot be localized due to legal or regulatory requirements then externally verifiable documentation supporting this must be provided as either an attachment or a link. This information will be provided to Apple or Google to meet store publication requirements. Since Apple and Google can ban accounts for repeated infractions not following this policy risks not just publication of your application, but all of our company's mobile applications and mobile development capabilities for that platform.*

Even if an app is already approved for one country or region, business owners must ensure that company compliance steps are followed for any additional publication region.

Applications themselves may be localized in code by developers to any country and language and to present country specific content or country specific logic/features, as long as the language is detectable on the device (usually in the form of an ISO-3166 type code). However **Application store listings** can only be localized within the available sets provided by each platform's storefront. Store languages are handled independently of store countries and are displayed according to the user's device locale - for example a store listing set to French and English available in both the USA and France will be presented in French to users who have their device set to French no matter whether they are in the USA or France.

Applications which have implemented accessibility features (for example image or button descriptions) should ensure that those accessibility features are also localized into the target language.

Available languages

Google Play (Android) store listings: <https://support.google.com/googleplay/android-developer/answer/9844778#zippy=%2Cview-list-of-available-languages>²⁹

App Store (iOS) store listings: <https://developer.apple.com/help/app-store-connect/reference/app-store-localizations/>

Setting an app to be available in some countries may require additional documentation, licensing, and/or regulatory approvals in addition to medical-legal reviews, for example publishing a mobile application in China might require an ICP number or an NPPA number (see <https://developer.apple.com/help/app-store-connect/reference/app-information/#:~:text=Availability%20in%20China%20mainland>), and publishing an app that uses certain device features might not be possible at all in a particular country depending on the laws of that country. These rules are set by individual governments and requirements/enforcement practices for any given country might change due to political decisions, it is important to verify the exact regulatory

²⁹ https://support.google.com/googleplay/android-developer/answer/9844778?hl=en&ref_topic=7071529&sjid=17657089343462012646-NA#zippy=%2Cview-list-of-available-languages

requirements with legal representatives familiar with laws regarding mobile applications in each of the targeted countries.

Useful information on **localization standards/practices** can be found at the following links:

<https://developer.apple.com/app-store/review/guidelines/#spam>

<https://github.com/react-native-community/react-native-localize>

<https://developer.android.com/guide/topics/resources/localization.html>

<https://developer.android.com/guide/topics/resources/localization.html#managing-strings>

<https://developer.apple.com/internationalization/>

<https://developer.apple.com/library/content/documentation/MacOSX/Conceptual/BPInternational>

<https://docs.microsoft.com/en-us/windows/uwp/globalizing/globalizing-portal>

<https://docs.microsoft.com/en-us/windows/uwp/globalizing/guidelines-and-checklist-for-globalizing-your-app>

4.5.8 Repository Management

Our company uses an enterprise GitHub system. Developers will be provided with a git repository and are expected to check in code on a regular basis as agreed with their project manager.

Outside developers using our company GitHub instance can access it after providing a unique public SSH key - <https://docs.github.com/en/authentication/connecting-to-github-with-ssh/generating-a-new-ssh-key-and-adding-it-to-the-ssh-agent#generating-a-new-ssh-key>

Our repositories require the use of a `.merck.yaml` file that contains either the asset ServiceNow short name, or the ISID of the project lead/developers. The repository must also have a team/group set as an administrator, and be set as a mobile app in the custom settings. In order to use our build system, repositories must have the FPMP team "github-mobile-platform" set as an administrator. The FPMP team may set mobile app flags as necessary in order to meet our company's standards.

Best repository strategies:

- Use a main / default branch containing the production code
- Use a release branch for development of updates
- Use separate branches for individual features
- Limit the number of active branches, merge code often and delete old branches
- Merge branches into the relevant update/release branch using pull requests as/when needed
- Merge branches into the main branch as released

4.5.9 Secrets Handling In Source Code

Our company requires that secret information such as credentials, keys, certificates, etc. not be kept in source code repositories, and will block commits into our company's GitHub instance of source containing detected secrets.

Due to how mobile applications work, some secret environment information may be required at build time, or at run time for mobile applications. Secret or confidential access keys/passwords/other sensitive information used at build time should be evaluated and removed from source code, and populated in builds using Bitrise's secrets management. Contact fp-mobile-platform@merck.com³⁰ to include these keys in Bitrise Secrets and to establish a consistent environment variable name.

Public keys such as Google's Firebase API keys are not considered secret but may still be flagged, however public keys for Google's paid API services (such as Google's geolocation APIs) must be restricted using Google's built in tools to the applications that will consume them using the package ID and SHA-1 of the Google Play managed keystore (for Android), or the bundle ID (for iOS).

If a secret is required to be included in delivered code for use at run time then steps must be taken to protect it and the resources it accesses - such as locking the secret using server side tools to known clients and obfuscation.

Coordination will be required with FPMP to maintain values for secret variables, especially in cases where frequent rotation is required. If a secret is required to be valid in the delivered code at run time and it is required to be rotated frequently then frequent updates prior to secret rotation will be required

³⁰ <mailto:fp-mobile-platform@merck.com>

to maintain those applications, as well as potentially maintaining two secrets with overlapping rotation dates in order to ensure applications work as expected.

Examples of expected handling

Valid detected secrets - e.g. AWS Secret Keys, SSH keys, vendor supplied keys to private NuGet repositories

Contact fp-mobile-platform@merck.com³¹ to include these keys in Bitrise Secrets.

Create matching variable to populate at build time.

Rotate the keys periodically after ensuring coordination with fp-mobile-platform@merck.com³² and publication timelines.

False positives - e.g. public keys, Google API Firebase keys, Google paid API keys

Firebase's configuration files (google-services.json, GoogleService-Info.plist) can be kept in source code with no changes required for use.

For new/previously unknown situations, provide research to prove that it is a false positive such as product specific documentation stating that the particular flagged element is considered public/non-secret.

Google's paid API keys can be kept in source code but must be locked to the production bundle ID.

Majority of use cases

Most Swift, Kotlin, and React Native will detect any key/value pair set in Bitrise Secrets as an environment variable available on the build machine, as long as it is not overwritten by the solution or other workflow logic. This means that code can be written to consume the Bitrise Secrets key/value pair the same as any other environment variable defined on a development machine.

NuGet Repo Secret Key Handling

Due to C# following a slightly different syntax for accessing environment variables, with key SOMEREPO_NUGET_KEY defined in Bitrise Secrets with the required value for accessing the SOMEREPO NuGet package, in the solution this would be set in NuGet.config as:

```
<?xml version="1.0" encoding="utf-8"?>
<configuration>

  <packageRestore>
```

³¹ <mailto:fp-mobile-platform@merck.com>

³² <mailto:fp-mobile-platform@merck.com>

```

    <add key="enabled" value="True" />
    <add key="automatic" value="True" />
  </packageRestore>

  <activePackageSource>
    <add key="All" value="(Aggregate source)" />
  </activePackageSource>
  <packageSources>
    <add key="SOMEREPO" value="https://nuget.pkg.github.com/someRepo/index.json" />
  </packageSources>
  <packageSourceCredentials>
    <SOMEREPO>
      <add key="Username" value="someRepoUserName" />
      <add key="ClearTextPassword" value="%SOMEREPO_NUGET_KEY%" />
    </SOMEREPO>
  </packageSourceCredentials>
</configuration>

```

4.5.10 Vulnerability Scan



Pre-release: All applications published by our company must pass a vulnerability scan prior to publication.

Post-release: All applications published in our company's app store accounts will be scanned automatically at regular intervals.

All applications must have security (including code security and data handling security) as a primary consideration at each stage of the application's lifecycle from the initial concept through to retirement.

All applications may be subject to regular post-publication reviews, including black, grey, and white box penetration testing and vulnerability reviews of published packages. Business owners are responsible for ensuring that all vulnerabilities reported post-release are remediated promptly. Timeframes for remediation are set out in the SSDLC and Vulnerability Management baselines.

Regular manual and automated reviews of source code to identify and remediate potential vulnerabilities following OWASP standards for mobile applications during the development lifecycle are a basic recommended practice for development teams and can help resolve many issues prior to pre-release vulnerability scans.

Our company uses AppSecure (aka Mobile Secure) which is a service offered by DataTheorem for automated scans. For applications that are built and/or re-signed by the mobile platform team, builds

will be automatically scanned in AppSecure from our CI/CD system. For other applications a scan instance can be requested through the ITRMS ASSA team by creating a Jira ticket at:

<https://go.merck.com/appsecrequest>

Information about the App Secure service can be found at <https://go.merck.com/applicationsecurity>.

We can assist business owners who are not able to request scans prior to publication, and with advice on vulnerability resolution, however as most applications are developed by vendors under the guidance of the business owner, it remains the business owner's responsibility to ensure that applications are fully reviewed and that issues are remediated.

Supply of all required deliverables and well documented, complete, source code at the time of request will improve scan times. Pre-publication scans will require the app to build and run in release configuration.

4.5.10.1 Vulnerability Management Standards

The System Lifecycle Security baseline setting out our company's requirements is available at:

<https://go.merck.com/ssdlcbaseline>

Our company's **Vulnerability Management baseline** sets out remediation steps, and is at:

<https://go.merck.com/vulnmanagement>

4.5.10.2 Useful Links

AppSecure is available at <http://securetheorem.com>³³.

³³ <http://securetheorem.com/>

Advice on how to resolve some common vulnerabilities in mobile applications is available at <http://go.merck.com/vulnerabilityfixes>.

Designers and developers may find the following links useful:

https://www.owasp.org/index.php/Main_Page

<https://owasp.org/www-project-mobile-security/>

https://cheatsheetseries.owasp.org/cheatsheets/User_Privacy_Protection_Cheat_Sheet.html

<https://github.com/OWASP/owasp-mstg>

<https://github.com/OWASP/owasp-masvs>

<https://github.com/MobSF/Mobile-Security-Framework-MobSF>

<https://github.com/rednaga/APKiD>

<https://www.cvedetails.com/>

<https://nvd.nist.gov/>

<https://www.cybok.org/>

4.5.10.3 Vulnerability Fixes

A brief set of information on how to resolve some commonly reported vulnerabilities in mobile applications.

Developers and QA are also strongly encouraged to review the standards set in <https://owasp.org/www-project-mobile-app-security/>, the training plan for developers for our company at [Application Security Training for Developers - Annual Requirement](#)³⁴, and the AppSec Sharepoint site at <https://go.merck.com/applicationsecurity>.

- [Android / iOS App Embeds SQL Query with Dynamic Input \(see page 127\)](#)
- [Android / iOS App Leverages SDK Vulnerable to Zipperdown \(see page 102\)](#)
- [Android / iOS Application Logs Sensitive Data \(see page 123\)](#)
- [Android / iOS App Obfuscation Disabled - Classes, Keys, & Tokens are Clear-text \(see page 129\)](#)
- [Android / iOS - Cordova Enforce Strict use of Apache Cordova Web Origin Access Parameter \(see page 104\)](#)
- [Android / iOS Credentials Exposed \(see page 111\)](#)
- [Android / iOS Keys and Passwords Identified in the App \(see page 128\)](#)
- [Android / iOS User Password Stored in App Private Storage \(see page 103\)](#)
- [Android App Private Data Exposed Through Backups \(see page 89\)](#)
- [Android App Views are Vulnerable to TapJacking \(see page 90\)](#)
- [Android Enable Android NSC to Prevent Cleartext Traffic \(see page 93\)](#)
- [Android Enable Verification on Android Security Provider \(see page 91\)](#)
- [Android Ensure Verify Apps Is Not Disabled \(see page 95\)](#)
- [Android File-Based Cross Site Scripting \(see page 94\)](#)
- [Android Insecure Random Number Generator Usage \(see page 97\)](#)
- [Android JavaScript Interface Injection Vulnerability \(see page 124\)](#)
- [Android Protect App Screens from 3rd Party Apps \(see page 95\)](#)
- [Android Unsafe HostnameVerifier Defined \(see page 97\)](#)
- [Android Unsafe SSL TrustManager Defined \(see page 98\)](#)
- [iOS App Accesses Private APIs \(see page 122\)](#)
- [iOS App Built with Deprecated SDK \(see page 123\)](#)
- [iOS App Embeds SQL Query with Dynamic Input \(see page 121\)](#)
- [iOS App Opens Port to Listen on Local Socket \(see page 116\)](#)
- [iOS App Store Blocker: App Uses IPv4-specific APIs \(see page 110\)](#)
- [iOS App Store Blocker: App Uses UIWebView APIs \(see page 127\)](#)
- [iOS App Susceptible to URI Scheme Hijacking \(see page 106\)](#)
- [iOS Cleartext HTTP Connections \(see page 116\)](#)
- [iOS Dangerous ATS Exemptions \(see page 101\)](#)
- [iOS Device Advertisement Identifier Collected \(see page 107\)](#)
- [iOS Disable Third Party Keyboards \(see page 99\)](#)

³⁴ <https://share.merck.com/spaces/ITRMS/pages/599276205/Application+Security+Training+for+Developers+-+Annual+Requirement>

- iOS Enable Data Protection Entitlement (see page 102)
- iOS Enable SSL Pinning - NSURLConnection and NSURLSession (see page 108)
- iOS Enable System-Managed Login Input Fields (see page 114)
- iOS Enable Two-Factor Authentication Input Field (see page 115)
- iOS Hash Generated Using Broken Cryptography API (SHA1) (see page 118)
- iOS Insecure XML Parser Configuration - NSXMLParser (see page 108)
- iOS Location "Always" Permission Requested (see page 105)
- iOS Message Digest Generated Using Broken Cryptography API (see page 119)
- iOS Missing Privacy Manifest (see page 129)
- iOS Null Initialization Vector Used for Symmetric Encryption (see page 126)
- iOS Prevent Keychain Items on Third Party Backups (see page 112)
- iOS Raw RSA Algorithm Used with Padding Disabled (see page 118)
- iOS Secure Enclave not Being Used When Possible (Security.framework) (see page 119)
- iOS Set Expiration on Pasteboard Data (see page 117)
- iOS Switch from Deprecated UIWebView API to WKWebView (see page 120)
- iOS Switch from Unsafe Serialization API to NSSecureCoding (see page 105)
- iOS Switch to Write-Only Access to User's Photo Library (see page 120)
- iOS Utilize Private Photo Access APIs (see page 122)
- iOS Weak ECB Mode Used for Symmetric Encryption Affected Components (see page 125)
- iOS Weak Keychain Protection Class - AccessibleAlways (see page 111)
- iOS WebView Disables Same-Origin Policy - baseUrl Argument (see page 104)
- Server Lacks HTTP Strict Transport Security Header (see page 113)
- Server Lacks OCSP Stapling (see page 112)
- Server Supports Insecure TLS 1.0 (see page 113)

4.5.10.3.1 Android App Private Data Exposed Through Backups



If an app allows backups then all app data, including sensitive user data, can be exported from the device (either through SD cards or over device connections).

For more information see:

<https://developer.android.com/guide/topics/manifest/application-element>
<https://pentestlab.blog/2017/01/24/security-guidelines-for-android-manifest-files/>
<https://securitygrind.com/exploiting-android-backup/>

Sample Code

```
//in AndroidManifest.xml set android:allowBackup to false to prevent full app backups

<application
...
    android:allowBackup="false"
...
/>
```

4.5.10.3.2 Android App Views are Vulnerable to TapJacking

For more information see: <https://blog.devknox.io/tapjacking-android-prevent/> and <https://stackoverflow.com/questions/22179773/android-tap-jacking-how-to-prevent-it>

Sample Code

```
//Option 1 - in layout xml setFilterTouchesWhenObscured
```

```
<Button
    android:id="@+id/button"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="sample button"
    android:filterTouchesWhenObscured="true"/>
```

```
//Option 2 - in code setFilterTouchesWhenObscured
```

```
mButton = (Button) findViewById(R.id.button);
mButton.setFilterTouchesWhenObscured(true);
```

```
//Option 3 - create a custom button which checks if button is obscured for all
CustomButtons
```

```
public class CustomButton extends Button {
    public CustomButton(Context context) {
        super(context);
    }
}
```

```

@Override
public boolean onFilterTouchEventForSecurity(MotionEvent event) {
    // check if the button view is
    // partly or wholly obscured by another visible window above it.
    if ((event.getFlags() & MotionEvent.FLAG_WINDOW_IS_OBSCURED) != 0) {
        // dispatch the event
        return true;
    }
    // drop the event
    return false;
}
}

//Option 4 - in code check if individual button is obscured

mButton = (Button) findViewById(R.id.button);
mButton.setOnTouchListener(new View.OnTouchListener() {
    @Override
    public boolean onTouch(View v, MotionEvent event) {
        // check if the button view is
        // partly or wholly obscured by another visible window above it.
        if ((event.getFlags() & MotionEvent.FLAG_WINDOW_IS_OBSCURED) != 0) {
            if (event.getAction() == MotionEvent.ACTION_UP) {
                // Process the touch on the button
            }
            // Return true to prevent the button from processing the touch.
            return true;
        }
        return false;
    }
});

```

4.5.10.3.3 Android Enable Verification on Android Security Provider

From: <https://developer.android.com/training/articles/security-gms-provider.html>

Sample Code

```

/**
 * Sample activity using {@link ProviderInstaller}.
 */
public class MainActivity extends Activity
    implements ProviderInstaller.ProviderInstallListener {

```

```

private static final int ERROR_DIALOG_REQUEST_CODE = 1;

private boolean retryProviderInstall;

//Update the security provider when the activity is created.
@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    ProviderInstaller.installIfNeededAsync(this, this);
}

/**
 * This method is only called if the provider is successfully updated
 * (or is already up-to-date).
 */
@Override
protected void onProviderInstalled() {
    // Provider is up-to-date, app can make secure network calls.
}

/**
 * This method is called if updating fails; the error code indicates
 * whether the error is recoverable.
 */
@Override
protected void onProviderInstallFailed(int errorCode, Intent recoveryIntent) {
    GoogleApiAvailability availability = GoogleApiAvailability.getInstance();
    if (availability.isUserRecoverableError(errorCode)) {
        // Recoverable error. Show a dialog prompting the user to
        // install/update/enable Google Play services.
        availability.showErrorDialogFragment(
            this,
            errorCode,
            ERROR_DIALOG_REQUEST_CODE,
            new DialogInterface.OnCancelListener() {
                @Override
                public void onCancel(DialogInterface dialog) {
                    // The user chose not to take the recovery action
                    onProviderInstallerNotAvailable();
                }
            });
    } else {
        // Google Play services is not available.
        onProviderInstallerNotAvailable();
    }
}

@Override
protected void onActivityResult(int requestCode, int resultCode,
    Intent data) {
    super.onActivityResult(requestCode, resultCode, data);
    if (requestCode == ERROR_DIALOG_REQUEST_CODE) {
        // Adding a fragment via GoogleApiAvailability.showErrorDialogFragment
        // before the instance state is restored throws an error. So instead,
        // set a flag here, which will cause the fragment to delay until

```

```

        // onPostResume.
        retryProviderInstall = true;
    }
}

/**
 * On resume, check to see if we flagged that we need to reinstall the
 * provider.
 */
@Override
protected void onPostResume() {
    super.onPostResume();
    if (retryProviderInstall) {
        // We can now safely retry installation.
        ProviderInstaller.installIfNeededAsync(this, this);
    }
    retryProviderInstall = false;
}

private void onProviderInstallerNotAvailable() {
    // This is reached if the provider cannot be updated for some reason.
    // App should consider all HTTP communication to be vulnerable, and take
    // appropriate action.
}
}

```

4.5.10.3.4 Android Enable Android NSC to Prevent Cleartext Traffic

From: <https://developer.android.com/training/articles/security-config.html>



Starting with Android 9 (API level 28), cleartext support is disabled by default.

Sample Code

```

//In AndroidManifest.xml

<?xml version="1.0" encoding="utf-8"?>
<manifest ... >
    <application android:networkSecurityConfig="@xml/network_security_config"
        ... >
        ...
    </application>
</manifest>

```

//In res/xml/network_security_config.xml, blocks all cleartext traffic, makes sure that connections to secure.example.com are over https

```
<?xml version="1.0" encoding="utf-8"?>
<network-security-config>
  <domain-config cleartextTrafficPermitted="false">
    <domain includeSubdomains="true">secure.example.com</domain>
  </domain-config>
</network-security-config>
```

4.5.10.3.5 Android File-Based Cross Site Scripting



For more information see: <https://support.google.com/faqs/answer/7668153>

Sample Code

```
//Prevents redirection to local file based JavaScript
webview = new WebView(this);

// Code to turn off setAllowFileAccessFromFileURLs and
setAllowUniversalAccessFromFileURLs
if(Build.VERSION.SDK_INT <= Build.VERSION_CODES.ICE_CREAM_SANDWICH_MR1) {
  webSettings.setAllowFileAccessFromFileURLs(false);
  webSettings.setAllowUniversalAccessFromFileURLs(false);
}

//If JavaScript isn't needed
// Code to Disable JavaScript in the WebView
webview.getSettings().setJavaScriptEnabled(false);

//If file access isn't needed
//Code to Disable file access within WebView
webview.getSettings().setAllowFileAccess(false);
```

4.5.10.3.6 Android Protect App Screens from 3rd Party Apps

Protecting the app's screens from 3rd party apps will block screen-sharing - this means that remote app testing services such as BrowserStack will not work. It is recommended to implement this privacy protection element only if the app handles sensitive data, and only in production builds/release candidates.

Sample code from: <https://developer.android.com/reference/android/media/projection/package-summary>

Sample Code

```
// Secure code for protecting one Activity
public class SecureActivity extends MerckActivity {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);

        // Set the Secure flag for this Window
        getWindow().setFlags(LayoutParams.FLAG_SECURE, LayoutParams.FLAG_SECURE);
    }
}

//Define a BaseActivity and set the FLAG_SECURE in that Activity :
public class BaseActivity extends MerckActivity {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        /**
         * approach 1: create a base activity and set the FLAG_SECURE in it,
         * Extend all other activities, Fragments from this activity
         */
        getWindow().setFlags(LayoutParams.FLAG_SECURE,
            LayoutParams.FLAG_SECURE);
    }
}

//Take this BaseActivity as superclass for all the other Activities.

//public class LoginActivity extends BaseActivity
//public class MainActivity extends BaseActivity
```

4.5.10.3.7 Android Ensure Verify Apps Is Not Disabled

From: <https://developer.android.com/training/safetynet/verify-apps.html>

Sample Code

```

public class MainActivity extends Activity {

    private static final String TAG = "MainActivity";

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        SafetyNet.getClient(this).isVerifyAppsEnabled()
            .addOnCompleteListener(new
        OnCompleteListener<SafetyNetApi.VerifyAppsUserResponse>() {
            @Override
            public void onComplete(Task<SafetyNetApi.VerifyAppsUserResponse>
task) {
                if (task.isSuccessful()) {
                    SafetyNetApi.VerifyAppsUserResponse result =
task.getResult();
                    if (result.isVerifyAppsEnabled()) {
                        // App logic goes here..
                    } else {
                        Log.d(TAG, "The Verify Apps feature is disabled.");
                        // Open the dialog to enable Verify Apps
                    }
                }
            }
        });
        SafetyNet.getClient(context).enableVerifyApps().addOnCompleteListener(
            new
            OnCompleteListener<SafetyNetApi.VerifyAppsUserResponse>() {
                @Override
                public void
                onComplete(Task<SafetyNetApi.VerifyAppsUserResponse> task) {
                    if (task.isSuccessful()) {
                        SafetyNetApi.VerifyAppsUserResponse
result = task.getResult();
                        if (result.isVerifyAppsEnabled()) {
                            Log.d(TAG, "The user gave
consent " +
                                "to enable the Verify
Apps feature.");
                        } else {
                            Log.d(TAG, "The user didn't
give consent " +
                                "to enable the Verify
Apps feature.");
                        }
                    } else {
                        Log.e(TAG, "A general error
occurred.");
                    }
                }
            }
        });
    }
}

```

```

        });
    }
} else {
    Log.e(TAG, "A general error occurred.");
}
}
});
}
}
}

```

4.5.10.3.8 Android Insecure Random Number Generator Usage



Issue Description

From: https://www.owasp.org/index.php/Insecure_Randomness

Many standard random number generation methods do not produce random numbers - they merely produce "random-enough" results which repeat frequently. Secure random number generators must be used.

From: <https://developer.android.com/reference/java/security/SecureRandom>

SecureRandom Example

```

import java.security.SecureRandom;
SecureRandom random = new SecureRandom();

// Generate a secure random integer 0 to 100 inclusive.
int number = rand.nextInt(101);

// Generate a set of random bytes
byte bytes[] = new byte[20];
random.nextBytes(bytes);

```

4.5.10.3.9 Android Unsafe HostnameVerifier Defined



See <https://support.google.com/faqs/answer/7188426>³⁵ as well as <https://developer.android.com/training/articles/security-ssl#CommonHostnameProbs> for details on the issue and more advice.

³⁵ <https://support.google.com/faqs/answer/7188426?hl=en>

Secure Hostname Verification Sample

```

HostnameVerifier hostnameVerifier = new HostnameVerifier() {
    @Override
    public boolean verify(String hostname, SSLSession session) {
        // Use the system's default hostname validation
        boolean allow =
        HttpURLConnection.getDefaultHostnameVerifier().verify(hostname, session);
        // Add additional checks on the hostname here
        // allow = false to reject any unknown hostnames
        return allow;
    }
};

```

4.5.10.3.10 Android Unsafe SSL TrustManager Defined

For more information see: <https://developer.android.com/training/articles/security-ssl>

Sample Code

```

//Performs safe SSL validation using Android's default trust X509TrustManager.
//Allows for custom validation of the certificate

public class CustomTrustManager implements X509TrustManager {

    // Retrieve the system's default trust manager
    private static final X509TrustManager defaultTrustManager =
    getSystemTrustManager();

    private static X509TrustManager getSystemTrustManager() {
        X509TrustManager systemTrustManager = null;
        TrustManagerFactory trustManagerFactory;
        try {
            trustManagerFactory = TrustManagerFactory.getInstance(
                TrustManagerFactory.getDefaultAlgorithm()
            );
        } catch (NoSuchAlgorithmException e) {
            throw new IllegalStateException("Should never happen");
        }
    }
}

```

```

    }

    try {
        trustManagerFactory.init((KeyStore)null);
    } catch (KeyStoreException e) {
        throw new IllegalStateException("Should never happen");
    }

    for (TrustManager trustManager : trustManagerFactory.getTrustManagers()) {
        if (trustManager instanceof X509TrustManager) {
            systemTrustManager = (X509TrustManager)trustManager;
        }
    }

    if (systemTrustManager == null) {
        throw new IllegalStateException("Should never happen");
    }
    return systemTrustManager;
}

// This is the method responsible for validating the server's certificate
@Override
public void checkServerTrusted(X509Certificate[] x509CertificateArray,
                              String authType) throws CertificateException {

    // Use the system's trust manager to do baseline SSL validation
    defaultTrustManager.checkServerTrusted(x509CertificateArray, authType);
    // If the default validation succeeded, add additional checks on the
    certificate here

}

@Override
public X509Certificate[] getAcceptedIssuers() {
    return new X509Certificate[0];
}

@Override
public void checkClientTrusted(X509Certificate[] x509Certificates,
                              String s) throws CertificateException {}
}

```

4.5.10.3.11 iOS Disable Third Party Keyboards



For more information see: <https://attack.mitre.org/techniques/T1417/>

Sample Code

```
// To be added to the App's UIApplicationDelegate
- (BOOL)application:(UIApplication *)application shouldAllowExtensionPointIdentifier:
(UIApplicationExtensionPointIdentifier)extensionPointIdentifier
{
    // Disable third-party keyboards on iOS 8+
    return NO;
}
```

Automatically Add Code To All AppDelegate

```
#!/usr/bin/env python
#Automatically adds code to disable third party keyboards
#Useful for hybrid js apps

import sys, os

def thirdpartykeyboardstring():
    return """-(BOOL)application:(UIApplication *)application
    shouldAllowExtensionPointIdentifier:(NSString *)extensionPointIdentifier
    {
        if (extensionPointIdentifier ==
            UIApplicationKeyboardExtensionPointIdentifier)
        {
            return NO;
        }
        return YES;
    }
    """

def alreadytherestring():
    return thirdpartykeyboardstring().split('\n')[0]

def addtoappdelegate(appdelegateLocation):
    lines = []
    linesOut = []
    with open(appdelegateLocation) as i:
        lines = i.readlines()
    if len(lines)<=0:
        return False
    alreadythere = False
    with open(appdelegateLocation, 'w') as o:
```

```

        for line in lines:
            lineOut = line
            if alreadytherestring() in line:
                alreadythere = True
                print("3rd party keyboards already disabled")
            if "@end" in line and not alreadythere:
                lineOut = thirdpartykeyboardstring() + line
            o.write(lineOut)

    return True

if len(sys.argv) > 0:
    rootDir = sys.argv[1]
    #pathwalk to find AppDelegate
    found = False
    for root,folders,files in os.walk(rootDir):
        for name in files:
            if "AppDelegate.m" in name:
                found = True
                print("Adding third party keyboard restriction to "+os.path.join(root,name))
                addtoappdelegate(os.path.join(root,name))
                print("Done")
        if not found:
            print("AppDelegate.m not located in " + rootDir)

```

4.5.10.3.12 iOS Dangerous ATS Exemptions

See https://developer.apple.com/library/archive/documentation/General/Reference/InfoPlistKeyReference/Articles/CocoaKeys.html#//apple_ref/doc/uid/TP40009251-SW33 for more information.

Sample Code

```

//False (enabled) by default.

<key>NSAppTransportSecurity</key>
<dict>
    <key>NSAllowsArbitraryLoads</key>
    <false/>
</dict>

//NSAllowsArbitraryLoads should never be set to true, this disables ATS globally.
Setting this requires justification.
//https should always be used, however if exemptions are truly needed for legacy
domains which only support http then they should be allowed only for those specific

```

domains. Company justification should be provided, and no PII or other sensitive data should ever be sent over http.

```
<key>NSAppTransportSecurity</key>
<dict>
  <key>NSExceptionDomains</key>
  <dict>
    <key>example.com</key>
    <dict>
      <key>NSIncludesSubdomains</key>
      <true/>
      <key>NSThirdPartyExceptionAllowsInsecureHTTPLoads</key>
      <true/>
    </dict>
  </dict>
</dict>
```

4.5.10.3.13 iOS Enable Data Protection Entitlement



See https://developer.apple.com/documentation/uikit/protecting_the_user_s_privacy/encrypting_your_app_s_files and https://developer.apple.com/documentation/bundleresources/entitlements/com_apple_developer_default-data-protection.

Setting the Data Protection Entitlement requires a profile matching this entitlement. Since all publication builds are generated and signed by the HH IT MAS team this should not impact development signing, but applications should be set to support data protection entitlements.

If configuration changes are needed to iOS profiles - please contact HH IT MAS at dccmobile@merck.com.

4.5.10.3.14 Android / iOS App Leverages SDK Vulnerable to Zipperdown



For more information see: https://www.owasp.org/index.php/Path_Traversal and <https://zipperdown.org/>

To fix -

- 1) Upgrade to the latest versions of unzipping libraries, for example v2.1.3+ of SSZipArchive <https://github.com/ZipArchive/ZipArchive>, and avoid using zip libraries which are still susceptible to path traversal.
- 2) Only access resources over https from known sources with certificate pinning, but do not rely on transport security for ensuring remote resources are valid/non-malicious, always have your app assume that any remote resource could be malformed before using.

4.5.10.3.15 Android / iOS User Password Stored in App Private Storage



Storing a password within the App's container may expose it to a sophisticated attacker. For iOS applications an attacker with physical access to the device can only access files within the App's container if they have the device's passcode, and one of the following is true:

- The App has enabled "Document Sharing" (via `UIFileSharingEnabled`³⁶), thereby allowing access to files stored in the App's *Documents* folder via iTunes.
- The App is saving some of its file to device backups via iTunes.
- The device is jailbroken, which grants access to all the files in the App's container.

A password is a sensitive piece of data even if accessing non-sensitive data because users often tend to reuse the same password for online accounts or unrelated websites. Consequently, an attacker with the ability to extract the user's password from the device may be able to access the victim's accounts for other services.

User passwords should never be stored by an application - if session persistence is required then:

1) For servers under our control use an app key/secret approach and user specific access token instead.

Using access tokens provides the following benefits:

- Tokens have a lifetime and expire after a certain time.
- Tokens can be revoked if they have been compromised, or if access from a specific device is no longer required.
- Unlike passwords, tokens are not reused across different services, thereby limiting exposure if they were to be compromised.

See <https://oauth.net/2/> for more information on one possible implementation.

2) If convenience for users is required - flag input fields as password auto fill.

See https://developer.apple.com/documentation/security/password_autofill/ and https://developer.apple.com/documentation/security/password_autofill/enabling_password_autofill_on_a_text_input_view for more information on iOS implementation.

See <https://developer.android.com/training/articles/security-tips#Credentials> and <https://developer.android.com/guide/topics/text/autofill> for Android credential handling.

3) For iOS tokens should be stored in the [Keychain](#)³⁷ instead of the App's private container. A wrapper class for the official iOS Keychain APIs (including `SecItemAdd()` and `SecItemCopyMatching()`) in order to simplify the process of storing and retrieving data from the Keychain is suggested.

³⁶ https://developer.apple.com/library/content/documentation/General/Reference/InfoPlistKeyReference/Articles/iPhoneOSKeys.html#//apple_ref/doc/uid/TP40009252-SW1

³⁷ <https://developer.apple.com/documentation/security/1401659-secitemadd>

4) Using a well-established and effective helper library such as <https://github.com/square/Valet>.

4.5.10.3.16 Android / iOS - Cordova Enforce Strict use of Apache Cordova Web Origin Access Parameter



See: <https://cordova.apache.org/docs/en/4.0.0/guide/appdev/whitelist/index.html>.

Since Cordova merely presents content in WebViews, access origin should be limited only allow access to domains and protocols which are needed for the application to function.

4.5.10.3.17 iOS WebView Disables Same-Origin Policy - baseURL Argument



To explicitly enable the same-origin policy in the vulnerable WebViews, use the `[NSURL URLWithString:@"about:blank"]` value as the `baseURL` parameter when loading content in the WebView, using the `loadHTMLString:baseURL:` or `loadData:MIMETYPE:textEncodingName:baseURL:` methods.

Using `about:blank` as the `baseURL` argument will enable the strictest same-origin policy, thereby preventing documents rendered within the WebView from being able to read the content of any website or any file stored in the App's folder.

It should be noted that setting the `baseURL` to `about:blank` will prevent local files from being referenced by the main page via a relative URL (such as `/Library/local_folder/image.png`) . If a `file://` URL needs to be set as the `baseURL` in order to support relative paths for referencing other resources stored on the device (images, style sheets, scripts, etc.), consider implementing one of the following solutions:

- Migrating to the `WKWebView` API³⁸ available since iOS 8. This API provides a safe method for loading local files, `loadFileURL:allowingReadAccessToURL:`, which allows specifying a local directory from which resources can be loaded, without also enabling access to all other files and domains.
- Modifying the HTML content to have all local resources (images, style sheets, etc.) being inlined inside the HTML page instead of being referenced via a `file://` URL. This will allow the `baseURL` parameter to be set to the safest value, `about:blank`, while allowing the data to be displayed.

³⁸https://developer.apple.com/library/ios/documentation/WebKit/Reference/WKWebView_Ref/#!/apple_ref/occ/instm/WKWebView/loadFileURL:allowingReadAccessToURL:

WebView Sample Code

```
[UIWebView loadHTMLString:htmlData baseURL:[NSURL URLWithString:@"about:blank"]]
```

4.5.10.3.18 iOS Location "Always" Permission Requested

All applications must only request the least permissions necessary to perform their function, excessive permissions should always be removed.

All applications must provide a string succinctly but fully describing the application's need to access permissions.

All applications must be able to stably handle users refusing to provide their location automatically, and should provide the user alternative methods to manually set location.

All applications must handle personal information such as location securely.

Review the app's requirement for accessing the user's location while the application is in the background, and whether this functionality is needed for the applications features. Usually if geo-location data is needed then it can be limited to whilst the application is in the foreground (when-in-use), rather than always. In this case the "nslocationalwaysandwheninuseusedescription" key should be removed, as well as any code which relies on always having location data. The "nslocationwheninuseusedescription" key can be used for handling situations where location is needed only when the application is in the foreground.

On the rare occasions that location is truly always needed applications must provide a descriptive string value for the "nslocationalwaysandwheninuseusedescription" key in the app's plist, as well as ensure that data is handled securely, sensibly, and that the application behaves correctly if a user chooses not to share their location.

4.5.10.3.19 iOS Switch from Unsafe Serialization API to NSSecureCoding

`NSCoding` is designed to allow serialization and deserialization of code objects. However, this protocol does not verify the type of object upon deserialization. Thus, it is vulnerable to object substitution attacks. A maliciously crafted payload which is deserialized via the `NSCoding` protocol can result in attacker-controlled code being executed.

Apple provides the `NSSecureCoding` protocol³⁹ which is robust against this type of attack. `NSSecureCoding` protects against object substitution attacks by requiring the programmer to declare the expected type of object before deserialization completes. Thus, if an invalid object is deserialized, the error can be handled safely. More information is available at: <https://developer.apple.com/documentation/foundation/nssecurecoding>.

³⁹ <https://developer.apple.com/documentation/foundation/nssecurecoding?language=objc>

Even if prior to deserialization custom sanity checks are performed on code objects
 NSDataSecureCoding is still recommended to be used.

4.5.10.3.20 iOS App Susceptible to URI Scheme Hijacking



Any URI scheme defined by the App can be hijacked by another App, if the malicious App registers the same scheme and also gets installed on the device. Specifically, the Apple documentation states the following:

"If more than one third-party app registers to handle the same URL scheme, there is currently no process for determining which app will be given that scheme."

Consequently, the rogue App could potentially receive messages sent via URI schemes instead of the real App; such messages could include OAuth tokens, search queries, etc.

While there were no solutions to replace URI schemes previously, iOS 9 [has introduced "Universal Links"](#)⁴⁰, which allow an App to register web domains that it owns. The App will then have the ability to "receive" and open any HTTP or HTTPS link to the associated domains, when the link gets clicked within any App on the device. If the App is not installed, the link will be opened in Safari instead. As opposed to URI schemes, Universal Links prevent another App from intercepting messages because they rely on web domain ownership, which needs to be proved by uploading a specific file to the web domain to be associated with the App. Therefore, a malicious App would not be able to hijack the domain as it would not be able to prove ownership.

As Universal Links are only available on iOS 9, developers should start migrating away from URI schemes but any App running on iOS 8 or below will still have to support schemes, until versions older than iOS 9 are no longer supported.

Universal Links and SFSafariViewController

As stated in the Apple documentation: <https://developer.apple.com/library/content/documentation/General/Conceptual/AppSearch/UniversalLinks.html>:

"When a user is browsing your website in Safari and they tap a universal link to a URL in the same domain as the current webpage, iOS respects the user's most likely intent and opens the link in Safari. If the user taps a universal link to a URL in a different domain, iOS opens the link in your app."

⁴⁰ <https://developer.apple.com/library/prerelease/ios/documentation/General/Conceptual/AppSearch/UniversalLinks.html>

Hence, if the App needs to be opened from Safari or an `SFSafariViewController`⁴¹ when the user is already browsing the App's domain (for example as part of a login flow that gets completed in Safari), the link that should be sent to the App must be hosted on a different domain; otherwise the user will open the link in Safari, instead of the App.

If a different domain cannot be used, other options include:

- Implementing the web flow within a `WKWebView`⁴² instead of an `SFSafariViewController`.
- Using a URI Scheme to ensure that the App always receives the link; doing so exposes your App to the security problems URI schemes have, as described in this issue.

If appropriate checks can also be performed in any code which receives universal links to ensure that any data to the app from links comes from legitimate sources, sensitive data or personally identifiable information should also never be transmitted in universal links.

More information can be found at: https://developer.apple.com/documentation/uikit/inter-process-communication/allowing_apps_and_websites_to_link_to_your_content/defining_a_custom_url_scheme_for_your_app.

4.5.10.3.21 iOS Device Advertisement Identifier Collected



The IDFA is a sensitive piece of data because it uniquely identifies a device and can therefore be used to create user profiles and track users across different Apps.

Additionally, [Apple's documentation](#)⁴³ states that the identifier "can only be used for serving advertisements", and should not be used for tracking users outside of a system serving ads.

When submitting an App to iTunes Connect, the developer must specify one of the following reasons to justify why the IDFA is collected:

- Serve advertisements within the App.
- Attribute this App installation to a previously served advertisement.
- Attribute an action taken within this App to a previously served advertisement

Misusing the IDFA is a common reason for Apple to reject Apps from the App Store, as described in Apple's "[Common App Rejections](#)" guide⁴⁴:

"When submitting your app for review, you'll be asked whether your app uses the Advertising Identifier (IDFA) to serve advertisements. If you indicate that your app uses the IDFA, but it does not have ad functionality or does not display ads properly, your

⁴¹ <https://developer.apple.com/documentation/safariservices/sfsafariViewController>

⁴² <https://developer.apple.com/documentation/webkit/wkwebview>

⁴³ https://developer.apple.com/library/ios/documentation/AdSupport/Reference/ASIdentifierManager_Ref/ASIdentifierManager.html

⁴⁴ <https://developer.apple.com/app-store/review/rejections/>

app may be rejected. Make sure to test your app on an iOS device to verify that ads work correctly. Similarly, if you indicate that your app does not use the IDFA, but it does, your app will be put into the "Invalid Binary" status."

Lastly and [starting with iOS 10](#)⁴⁵, if the user has decided to enable the "Limit Ad Tracking" setting on their device, the IDFA returned by iOS will be `00000000-0000-0000-0000-000000000000`, making the IDFA unsuitable for user tracking on such devices.

Our company's applications should never have any reason to use or collect the IDFA or use SDKs which do so.

4.5.10.3.22 iOS Insecure XML Parser Configuration - NSXMLParser

i Vulnerable code locations rely on the `NSXMLParser` class to parse XML content, but enable the processing of external entities by setting the `shouldResolveExternalEntities` to `YES`. This setting causes the XML parser to be vulnerable to XML external entity attacks, which allow an attacker to potentially retrieve the content of files or URLs accessible to the App, by sending specifically crafted XML documents to the vulnerable parser.

Update the vulnerable XML parsers to disable the parsing of external entities. Specifically, modify the `NSXMLParser` object to set the `shouldResolveExternalEntities` property to `NO` (which is the default value).

See: <https://developer.apple.com/documentation/foundation/xmlparser/1414143-shouldresolveexternalentities> for more information.

4.5.10.3.23 iOS Enable SSL Pinning - NSURLConnection and NSURLSession

i Vulnerable code locations within the App will accept SSL certificates signed by any Certificate Authority (CA) installed within the device's trust store. As part of a high-profile attack, this could allow an entity in control of a CA private key (such as a government) to perform a man-in-the-middle attack against any HTTPS traffic initiated by the target code locations.

Additionally, several trusted Certificate Authorities have been compromised in the past and recently (as described in <http://googleonlinesecurity.blogspot.com/2014/07/maintaining-digital-certificate-security.html>) which allowed the attackers to issue malicious but valid SSL certificates for any domain.

Lastly, users can be tricked or required (for example per their employer's policies) to install new CA certificates within the device's trust store, further increasing the potential for compromise.

⁴⁵ <https://datatheorem.github.io/ios/2016/08/16/ios10-changes/>

Rather than using the iOS default certificate validation mechanism which trusts dozens of Certificate Authorities by default, implement SSL pinning by embedding the server (or CA) certificate or public key for the App's server endpoints inside the App. The App should then ensure that the server's certificate chain contains the pinned key or certificate.

To minimize the development effort required to implement SSL pinning, Data Theorem has released **TrustKit**: an open source library which makes it very easy to implement SSL pinning in iOS and OS X Apps. More information about TrustKit is available at <https://datatheorem.github.io/TrustKit/>.

Alternatively, SSL pinning can be manually implemented by modifying the identified code locations to validate that the SSL certificate sent by the server at runtime as part of its certificate chain is the same as the certificate or key hard-coded within the App. This can be achieved by implementing the `connection:willSendRequestForAuthenticationChallenge:` method defined in `NSURLConnectionDelegate` protocol. See https://developer.apple.com/library/mac/documentation/Foundation/Reference/NSURLConnectionDelegate_Protocol/Reference/Reference.html and https://developer.apple.com/documentation/foundation/url_loading_system/handling_an_authentication_challenge/performing_manual_server_trust_authentication for additional guidelines.

NSURLSession Sample Code

```
func URLSession(session: NSURLSession, didReceiveChallenge challenge:
NSURLAuthenticationChallenge, completionHandler (NSURLSessionAuthChallengeDisposition,
NSURLCredential?) -> Void) {

    let serverTrust = challenge.protectionSpace.serverTrust

    let certificate = SecTrustGetCertificateAtIndex(serverTrust!, 0)

    // Set SSL policies for domain name check

    let policies = NSMutableArray();

    policies.addObject(SecPolicyCreateSSL(true, (challenge.protectionSpace.host)))

    SecTrustSetPolicies(serverTrust!, policies);

    // Evaluate server certificate

    var result: SecTrustResultType = 0

    SecTrustEvaluate(serverTrust!, &result)

    let isServerTrusted:Bool = (Int(result) == kSecTrustResultUnspecified ||
Int(result) == kSecTrustResultProceed)
```

```

// Get local and remote cert data

let remoteCertificateData:NSData = SecCertificateCopyData(certificate!)

let pathToCert = NSBundle.mainBundle().pathForResource(githubCert, ofType: "cer")

let localCertificate:NSData = NSData(contentsOfFile: pathToCert!!)


if (isServerTrusted && remoteCertificateData.isEqualToData(localCertificate)) {

    let credential:NSURLCredential = NSURLCredential(forTrust: serverTrust!)

    completionHandler(.UseCredential, credential)

} else {

    completionHandler(.CancelAuthenticationChallenge, nil)

}
}

```

4.5.10.3.24 iOS App Store Blocker: App Uses IPv4-specific APIs



Starting June 1, 2016, Apple has started blocking Apps that [do not support IPv6-only networks](#)⁴⁶:

"Starting June 1, 2016, all apps submitted to the App Store must support IPv6-only networking. [...] if your app utilizes IPv4-specific APIs or hard-coded IP addresses, you will need to make changes."

Affected code locations leverage network functions that are IPv4-only, and therefore will not work properly if the device is on an IPv6-only network. It should be noted that this issue has no security impact, but may cause the App to be rejected by the App Store review team.

⁴⁶ <https://developer.apple.com/support/ipv6/>

4.5.10.3.25 Android / iOS Credentials Exposed



For both iOS and Android, data hard-coded in files within an app's package can be retrieved by anyone who downloads an app and then inspects the content, or by anyone who has visibility of network traffic if credentials are exposed in URLs.

Hardcoded credentials are potentially publicly exposed and may give access to the app's servers or even other resources or even other accounts if the credentials are being reused.

Examples include setting credentials as strings in Java files, or in XML/plist files, as well as passing credentials as part of a URL (e.g. <https://example.com?user=username&pass=password>, or <http://user:pass@example.com>).

Investigate why hard-coded credentials are being used, and remove them from the App's resource files. Any other sensitive piece of information should be stripped from the App before shipping it to production.

If credentials are needed to be sent to a server for login/other capabilities use a token based system, and POST data in secure requests rather than including them as part of URLs.

4.5.10.3.26 iOS Weak Keychain Protection Class - AccessibleAlways



The App sets the items' Keychain protection class to `kSecAttrAccessibleAlways`, which is not only the weakest accessibility class but also results in the items being always decrypted and accessible, regardless of whether the device is locked.

Consequently, an attacker with physical access to the device may be able to extract this data from the device while it is locked and without access to the user's passcode. The attacker would have to compromise the device by exploiting a vulnerability to gain code execution (such as a buffer overflow in a driver) or within the App. After code execution, Keychain items protected with `kSecAttrAccessibleAlways` could be recovered even if the device is locked.

Additionally, the [Apple documentation](#)⁴⁷ for the `kSecAttrAccessibleAlways` protection class specifies that it is "not recommended for App use."

In order to ensure Keychain items are protected especially when the device is locked, do not use the `kSecAttrAccessibleAlways` protection class. Instead, use the strongest protection class possible for each Keychain item:

- `kSecAttrAccessibleWhenUnlockedThisDeviceOnly` for Keychain items that do not need to be accessible to the App while the device is locked
- `kSecAttrAccessibleAfterFirstUnlockThisDeviceOnly` for Keychain items that need to be accessible while the device is locked, for example if the App is performing tasks while in the background.

⁴⁷ <https://developer.apple.com/reference/security/ksecattraccessiblealways>

More information regarding Keychain protection classes is available at <https://developer.apple.com/library/ios/documentation/security/Reference/keychainservices/Reference/reference.html>.

Additionally, because the official iOS Keychain APIs (including `SecItemAdd()` and `SecItemCopyMatching()`) are overly complex and difficult to use, consider leveraging a wrapper instead in order to simplify the process of storing and retrieving data from the Keychain.

4.5.10.3.27 iOS Prevent Keychain Items on Third Party Backups



When adding vulnerable or sensitive items to the Keychain, the App does not explicitly exclude the items from device backups, which will allow sensitive data to get exported to iTunes and iCloud backups.

Having sensitive data sent to iCloud exposes it to both Apple, Inc. and an attacker with the ability to compromise the user's iCloud account (similarly to the iCloud celebrity hack, described in <http://en.wikipedia.org/wiki/August2014celebrityphotoleaks/>).

Additionally, exporting data to iTunes backups allows an attacker with physical access to the device and its passcode to use the iTunes encrypted backup functionality to extract all the Keychain items (that are flagged for backups) from the device.

In order prevent Keychain data from being migrated to iTunes and iCloud backups, explicitly set a `ThisDeviceOnly` accessibility class (such

as `kSecAttrAccessibleWhenUnlockedThisDeviceOnly`) for all Keychain items. More information about such accessibility classes is available at <https://developer.apple.com/library/ios/documentation/security/Reference/keychainservices/Reference/reference.html>.

Additionally, because the official iOS Keychain APIs (including `SecItemAdd()` and `SecItemCopyMatching()`) are overly complex and difficult to use, consider leveraging a wrapper instead in order to simplify the process of storing and retrieving data from the Keychain.

4.5.10.3.28 Server Lacks OCSP Stapling



The affected server endpoints consumed by the application do not return their SSL certificate's revocation status information via OCSP Stapling.

OCSP stapling is a feature that can be enabled in most web servers to provide clients (such as browsers or mobile Apps) connecting to the server over HTTPS with the information needed to ensure that the certificate has not been revoked. Certificates can be revoked for a variety of reasons, including the compromise of the SSL key or when a certificate was mistakenly issued. Giving HTTPS clients the ability to check for revocation can mitigate the impact of invalid or compromised certificates, in specific attack scenarios.

Update the affected mobile endpoints' configuration to enable support for OCSP Stapling. Most modern web servers including [Apache](#) and [nginx](#)⁴⁸ support stapling.

⁴⁸ <https://www.digitalocean.com/community/tutorials/how-to-configure-ocsp-stapling-on-apache-and-nginx>

4.5.10.3.29 Server Supports Insecure TLS 1.0



TLS 1.0 is a legacy protocol, which has been found to be vulnerable to numerous attacks since it was released.

Such attacks include the *POODLE* vulnerability, which was discovered in 2014 [by the Google security team](#)⁴⁹, as well as other flaws affecting both on the protocol's key exchange mechanism and on the encryption schemes it supports.

A summary of most of the vulnerabilities the TLS 1.0 protocol is vulnerable to is available in [RFC 7457](#)⁵⁰.

Most mobile Apps would usually negotiate a stronger version of the protocol (such as TLS 1.2), making this issue difficult to exploit in the context of a an App. However, specific browsers (such as Firefox) accessing the vulnerable endpoints may be exposed [through protocol downgrade attacks](#)⁵¹. Lastly, TLS v1.0 is no longer acceptable for PCI compliance due to the vulnerabilities mentioned above (such as *POODLE*), as stated in [Appendix A2 of the PCI DSS Requirements](#)⁵²:

"After June 30, 2018, all entities must have stopped use of SSL/early TLS as a security control, and use only secure versions of the protocol"

Disable TLS v1.0 across all of the App's endpoints, and require TLS 1.2 or later. It should be noted that disabling TLS v1.0 will also prevent legacy clients from connecting to the endpoints.

Guidelines on secure server-side TLS configurations are available in [the Mozilla Server Side TLS guides](#)⁵³.

Additionally, regularly monitor and validate the deployed SSL/TLS configuration on the App's server endpoints using tools such as [SSLyze](#)⁵⁴ or [SSL Labs](#)⁵⁵.

4.5.10.3.30 Server Lacks HTTP Strict Transport Security Header



The HTTP Strict Transport Security (HSTS) header allows the server to inform clients such as browsers or Apps that they should never access the server over HTTP, and that they should automatically convert all attempts to access the site using HTTP to HTTPS requests instead.

It is supported by all recent browsers and most mobile networking APIs (such as `NSURLSession` on iOS). Hence, setting the HSTS header will ensure that the mobile App is always accessing these endpoints over HTTPS.

⁴⁹ <http://googleonlinesecurity.blogspot.com/2014/10/this-poodle-bites-exploiting-ssl-30.html>

⁵⁰ <https://tools.ietf.org/html/rfc7457>

⁵¹ <https://community.qualys.com/blogs/securitylabs/2014/10/15/ssl-3-is-dead-killed-by-the-poodle-attack>

⁵² https://www.pcisecuritystandards.org/documents/PCI_DSS_v3-2.pdf

⁵³ https://wiki.mozilla.org/Security/Server_Side_TLS

⁵⁴ <https://github.com/nabla-c0d3/sslyze>

⁵⁵ <https://www.ssllabs.com/ssltest/>

For endpoints accessed by the App over HTTPS only, set the HTTP Strict Transport Security Header in the endpoints' responses in order to further protect the App's network traffic, and to ensure that the App will connect to the APIs over HTTPS only; the following HTTP header can be added to the endpoint's responses for this purpose:

```
Strict-Transport-Security: max-age=10886400
```

Additionally and to further secure connections to the endpoints, consider:

- Enabling HSTS for all of the endpoint's subdomains using the `includeSubDomains` directive.
- Adding the endpoint to the HSTS preload list, which will automatically enable HSTS for the endpoint in most recent browsers. The process is described in details at <https://hstspreload.appspot.com/>.

This can be achieved by deploying the following HSTS header:

```
Strict-Transport-Security: max-age=10886400; includeSubDomains; preload
```

More information about HSTS is available on [Mozilla's HSTS page](#)⁵⁶.

4.5.10.331 iOS Enable System-Managed Login Input Fields



The App does not specify text fields marked as username and password input fields. Starting with the iOS 12 SDK, [Apple has added new text field properties](#)⁵⁷ to automate the process of password generation and login credentials entry.

If two text fields have the `textContentType` property set to `UITextContentTypeUsername` and `UITextContentTypePassword`, iOS will recognize that the current view represents a login context, and will prompt the user to input their saved iCloud Keychain credentials. If a field has the `textContentType` property set to `UITextContentTypeNewPassword`, iOS will prompt the user with a secure, system generated password to use.

This enhancement is also available if the App's login flow occurs within a WebView. In this case, set the `autocomplete` attribute of the login HTML elements, as further described in the [Apple documentation](#)⁵⁸.

Utilizing these APIs makes it easier for users to create and manage secure credentials.

Leverage the `UITextContentType` [property available in iOS 12](#)⁵⁹ to automate the process of generating secure passwords and inputting login credentials for your users.

From: https://developer.apple.com/documentation/security/password_autofill/enabling_password_autofill_on_a_text_input_view

⁵⁶ https://developer.mozilla.org/en-US/docs/Web/Security/HTTP_strict_transport_security

⁵⁷ <https://developer.apple.com/documentation/uikit/uitextinputtraits/1649656-textcontenttype?language=objc>

⁵⁸ https://developer.apple.com/documentation/security/password_autofill/enabling_password_autofill_on_an_html_input_element?language=objc

⁵⁹ <https://developer.apple.com/documentation/uikit/uitextinputtraits/1649656-textcontenttype?language=objc>

By default, the system selects the a keyboard based on the input view's `textContentType` property; however, you can mix the input view's text content type and keyboard type to explicitly define the desired keyboard. For example, if your site uses email addresses as user names, set the input view's `textContentType` property to `.username`, and set the `keyboardType` property to `.UIKeyboardType.emailAddress`.

iOS supports Password AutoFill on `UITextField`, `UITextView`, and any custom view that adopts the `UITextInput` protocol.

NB Leverage the system keyboard rather than implementing a keyboard directly in your app's view hierarchy.

Sample Code

```
//This example defines text fields for logging in:
userTextField.textContentType = .username
userTextField.keyboardType = .emailAddress
passwordTextField.textContentType = .password

//When creating a new account or changing the password, use the .newPassword
text content type instead:
newPasswordTextField.textContentType = .newPassword
confirmPasswordTextField.textContentType = .newPassword

//Additionally, you can autocomplete security codes from single-factor SMS
login flows:
singleFactorCodeTextField.textContentType = .oneTimeCode
```

4.5.10.332 iOS Enable Two-Factor Authentication Input Field



Starting with iOS 12 SDK, [Apple has added a new text field property](#)⁶⁰ to automate the process of inputting two-factor authentication SMS codes.

If a text field has the `textContentType` property set to `UITextContentTypeOneTimeCode`, iOS will automatically scan incoming SMS messages for an authentication code. When the user interacts with the text field, iOS will prompt the user to AutoFill the authentication code. This process makes using two-factor authentication a smoother experience for users.

This enhancement is also available if the App's two-factor authentication flow occurs within a `WebView`. In this case, set the `autocomplete` attribute of the appropriate HTML elements, as [further described here](#)⁶¹.

Two-factor authentication is a valuable mechanism to ensure that even if a user's credentials are leaked, access to account data within the App is still restricted without physical access to their devices or other accounts.

⁶⁰ <https://developer.apple.com/documentation/uikit/uitextinputtraits/1649656-textcontenttype?language=objc>

⁶¹ https://developer.apple.com/documentation/security/password_autofill/enabling_password_autofill_on_an_html_input_element?language=objc

If appropriate ensure two-factor authentication is offered or enforced in your service. Leverage the `UITextContentTypeOneTimeCode` [content type available in iOS 12](#)⁶² to automate the process of entering SMS authentication codes for your users.

4.5.10.3.33 iOS Cleartext HTTP Connections

 Accessing a server over unencrypted HTTP results in the user's data being exposed to an attacker on the network, for example when using a public WiFi access point at the coffee shop or at the airport. This would allow the attacker to steal the user's authentication tokens and perform actions on behalf of the user or access the user's data on the server. Additionally, data sent over HTTP can be inspected, logged and/or modified by any endpoint or device on the network between the user's device and the App's server. This has for example been used by Internet Service Providers [to inject ads in web traffic](#)⁶³. Lastly, cleartext HTTP connections have been deprecated in iOS 9, which introduces a security mechanism called [App Transport Security](#)⁶⁴. By default, Apps are no longer able to initiate cleartext HTTP connections and if insecure connections are needed, an App has to specify an exception within the App's *Info.plist* for each domain. Adding unnecessary exemptions may cause app rejection.

To resolve modify the App to access all web resources over HTTPS instead of unencrypted HTTP. Doing so will prevent any attacker on the network from being able to recover or modify the App's network traffic. Additionally, switching to HTTPS will help with [App Transport Security](#)⁶⁵ compliance, which requires all connections to be encrypted using HTTPS. Only ever implement exemptions if absolutely necessary, never send potentially sensitive or personally identifiable information in cleartext or over HTTP.

4.5.10.3.34 iOS App Opens Port to Listen on Local Socket

 The vulnerable code locations open a port to listen for incoming network connections, thereby adding an entry point into the App and increasing the attack surface. Depending on the type of socket configuration being used, the exposure varies:

- If the socket is only listening on the loopback interface (ie. localhost), all the other Apps installed on the device will be able to connect to it.
- If the socket is listening on all network interfaces, any device on the device's network will also be able to connect to the App's socket.

⁶² <https://developer.apple.com/documentation/uikit/uitextinputtraits/1649656-textcontenttype?language=objc>

⁶³ <https://arstechnica.com/tech-policy/2013/04/how-a-banner-ad-for-hs-ok/>

⁶⁴ https://developer.apple.com/library/ios/releasenotes/General/WhatsNewIniOS/Articles/iOS9.html#//apple_ref/doc/uid/TP40016198-SW14

⁶⁵ https://developer.apple.com/library/ios/releasenotes/General/WhatsNewIniOS/Articles/iOS9.html#//apple_ref/doc/uid/TP40016198-SW14

This can lead to the compromise of user data, depending on what the socket is used for and what kind of data it exposes.

Additionally, many apps that exhibit the same behavior [have been exposed by security researchers](#)⁶⁶ and flagged as insecure.

As exposing an open port is difficult to do securely within a mobile app, modify the vulnerable code locations to disable the creation of the web server.

For most use cases, more secure implementations are possible (such as [using AirDrop](#)⁶⁷ for sending files, or [Multipeer Connectivity](#)⁶⁸ for a more general SDK for sharing data)

If the server is mandatory for debugging purposes, consider using a build flag such as `IS_DEBUG` to prevent the server from ever being created in a production environment.

4.5.10.3.35 iOS Set Expiration on Pasteboard Data



This is a vulnerability only if sensitive information under the app's control is able to be placed in the clipboard. The best fix is to not allow sensitive information to be placed in the clipboard.

The vulnerable code locations leverage the `UIPasteBoard` [API](#)⁶⁹ to write data into the device's global pasteboard, without configuring an expiration date for the data.

Data stored in the pasteboard is accessible to any App on the device that is brought to the foreground. As pasteboard data does not expire by default, if a user leverages the copy function on any sensitive field, the data will be accessible until it is overwritten, which may not happen in a long time.

As of iOS 10, `UIPasteBoard` items can be assigned an expiration date: at the configured time and date, iOS will remove the pasteboard items from the pasteboard. Hence, it is recommended that any item saved into the pasteboard be given an expiration date.

More information on `UIPasteboard` options can be found at <https://developer.apple.com/documentation/uikit/uipasteboard>.

Sample Code

```
let pasteboard = UIPasteboard.generalPasteboard()

// Add a string
let aLocalOnlyStringKey = "Local only string key"
let aLocalOnlyStringValue = "Local only string value"

// Set the string in the LOCAL pasteboard
```

⁶⁶ <https://www.wired.com/2017/04/obscure-app-flaw-creates-backdoors-millions-smartphones/>

⁶⁷ <https://developer.apple.com/documentation/uikit/uiactivitytypeairdrop>

⁶⁸ <https://developer.apple.com/documentation/multipeerconnectivity>

⁶⁹ <https://developer.apple.com/reference/uikit/uipasteboard>

```

pasteboard.setItems([[aLocalOnlyStringKey: aLocalOnlyStringValue]], options:
[UIPasteboardOption.localOnly : true])

//Add an expiringString
let aExpiringStringKey = "Local only string key"
let aExpiringStringValue = "Local only string value"

// Create a date 5 minutes from now
let expiryTime = Date().addingTimeInterval(60*5)
// Add the string and mark it to expire at expiryTime
pasteboard.setItems([[aExpiringStringKey: aExpiringStringValue]], options:
[UIPasteboardOption.expirationDate: expiryTime])

```

4.5.10.3.36 iOS Raw RSA Algorithm Used with Padding Disabled



Code within the app uses the `Security` framework to perform RSA encryption, and relies on the insecure `kSecPaddingNone` padding constant.

When using `kSecPaddingNone` as the padding setting, the RSA operations don't use any padding and implement the textbook RSA algorithm which is insecure and not suitable for real world implementations. As described at <http://rdist.root.org/2009/10/06/why-rsa-encryption-padding-is-critical/>, failure to properly use padding with RSA could allow attackers to forge signatures, decrypt ciphertext, or even recover your private key.

Avoid directly using cryptographic algorithms (such as RSA or AES) as implementing secure encryption or protocols on top of raw cryptographic algorithms is a difficult task, due to subtle implementation flaws (such as timing attacks) and sophisticated cryptographic attacks. If RSA needs to be used directly, specify a padding constant of `kSecPaddingOAEP` in order to enable a strong padding algorithm for RSA encryption.

4.5.10.3.37 iOS Hash Generated Using Broken Cryptography API (SHA1)



Code within the app uses `CC_SHA1` or `CryptoKit.Insecure.SHA1` functions.

The vulnerable code locations use the `CC_SHA1` or `CryptoKit.Insecure.SHA1` hashing functions, which leverage hashing algorithms that are proven to be vulnerable to [collision attacks](#)⁷⁰, and are unsuitable for modern use.

Apple officially considers this algorithm insecure. They state in [iOS 13 CryptoKit documentation](#)⁷¹:

⁷⁰ <https://shattered.it/>

⁷¹ <https://developer.apple.com/documentation/cryptokit/insecure/sha1>

"This hash algorithm isn't considered cryptographically secure, but is provided for backward compatibility with older services that require it. For new services, prefer one of the secure hashes, like SHA512."

Switch each usage of these outdated hashing functions to use a stronger algorithm with better collision resistance properties, such as SHA-256 or SHA-512.

4.5.10.3.38 iOS Secure Enclave not Being Used When Possible (Security.framework)

- i** The App uses Apple's Security framework to generate cryptographic key pairs, but does not leverage the frameworks' Secure Enclave features introduced with iOS 13, to securely store private keys. Starting with iOS 13, developers can add an extra layer of security [by leveraging the hardware based key manager located in the Secure Enclave](#)⁷², available in any iOS devices with an A7 chip or newer. When a private key is stored in the Secure Enclave, the key never actually leaves the Secure Enclave and your App or the OS itself never has to store a plain text version of the key in memory, in order to use it. Instead, your App instructs the Secure Enclave to create the key, securely store it, and perform cryptographic operations with it, and then only receives the output of these operations (such as encrypted data, or the result of a signature verification).

Leverage the Secure Enclave capabilities when performing any cryptographic operation on the device, in order to add an extra layer of security for the App's private keys.

- Refer to Apple's official documentation at https://developer.apple.com/documentation/security/certificate_key_and_trust_services/keys/storing_keys_in_the_secure_enclave for more details.

4.5.10.3.39 iOS Message Digest Generated Using Broken Cryptography API

- i** The vulnerable code locations use the `CC_MD` hashing functions, which leverage hashing algorithms (including MD2 and MD5) that are proven to be vulnerable to [collision attacks](#)⁷³, and are unsuitable for modern use. Apple has officially deprecated these APIs in the iOS 13.0 SDK. They state in the CommonCrypto headers:

⁷² https://developer.apple.com/documentation/security/certificate_key_and_trust_services/keys/storing_keys_in_the_secure_enclave

⁷³ <https://arstechnica.com/security/2012/10/sha1-crypto-algorithm-could-fall-by-2018/sha>

"These functions are cryptographically broken and should not be used in security contexts. Clients should migrate to SHA256 (or stronger)."

Switch each usage of these outdated hashing functions to use a stronger algorithm with better collision resistance properties, such as SHA-256 or SHA-512.

4.5.10.3.40 iOS Switch from Deprecated UIWebView API to WKWebView

 Starting in the iOS 8.0 SDK, Apple introduced `WKWebView`, which is a modern implementation of a web content view. [Apple states](#).⁷⁴

"Important Starting in iOS 8.0 and OS X 10.10, use WKWebView to add web content to your app. Do not use UIWebView or WebView."

In the iOS 12 SDK, `UIWebView` has been officially deprecated in favor of `WKWebView`.

The `WKWebView` API brings several improvements over `UIWebView`, including:

- Support for modern web features, such as IndexedDB and ArrayBuffer.
- A much faster JavaScript engine. Web page load times are roughly twice as fast compared to `UIWebView`.
- Support for iOS' Credentials AutoFill feature.
- Significant security improvements as `WKWebView` runs in a separate process that is isolated from the App; this ensures the App cannot be compromised with a WebKit exploit.

Migrate WebViews within your App to use the `WKWebView` class instead of `UIWebView`. `UIWebView` will be removed in a future release and its use is discouraged. All apps are recommended to upgrade to `WKWebView` as soon as possible.

4.5.10.3.41 iOS Switch to Write-Only Access to User's Photo Library

 The app requests permission to read and write to the user's photo library through the `NSPhotoLibraryUsageDescription` key in the `Info.plist` file.

⁷⁴ <https://developer.apple.com/documentation/webkit/wkwebview?language=objc>

Starting with iOS 11⁷⁵, Apple provides the more granular `NSPhotoLibraryAddUsageDescription` permission. This permission allows the app to save new assets to the user's photo library but does not allow reading stored photos. Using the write-only permission is safer than exposing all of the user's photos to the app. If the app does not require reading the entire photos library, utilize the `NSPhotoLibraryAddUsageDescription` permission to minimize the app's access level.

Evaluate whether the app can deliver its functionality with write-only access to the photo library. If so, migrate to the more granular `NSPhotoLibraryAddUsageDescription` permission to minimize the sensitive user data exposed to the app.

4.5.10.3.42 iOS App Embeds SQL Query with Dynamic Input



Vulnerable client-side SQL queries are populated at runtime via format strings. If the string contents are derived via user input, an attacker may be able to perform an [SQL injection](#)⁷⁶ attack. This may result in data leakage, data loss, or data corruption.

As user input may be unsafe, it should not be directly included in database accesses. By utilizing [prepared statements](#)⁷⁷ and [bind variables](#)⁷⁸, the App can perform dynamic SQL queries without directly including potentially unsafe user input in database accesses.

Wherever possible, utilize [parameterized inputs](#)⁷⁹ in dynamic SQL statements to mitigate the possibility of an SQL injection attack. Otherwise, ensure all user input is properly sanitized before including it in an SQL query, and close this issue as "Risk Accepted".

Sample Code

```
// Unsafe code, vulnerable to SQL injection
NSString* studentName = ...;
const char* sql_query = [[NSString stringWithFormat:@"INSERT INTO students (name)
VALUES (%@), studentName] UTF8String];
sqlite3_stmt* stmt;
sqlite3_prepare_v2(db, sql_query, 1, &stmt, NULL);
sqlite3_step(stmt);
sqlite3_finalize(stmt);

// Safe code, using bound parameters
NSString* studentName = ...;
const char* sql_query = "INSERT INTO students (name) VALUES (?)";
```

⁷⁵ https://developer.apple.com/library/archive/documentation/General/Reference/InfoPlistKeyReference/Articles/CocoaKeys.html#//apple_ref/doc/uid/TP40009251-SW73

⁷⁶ https://www.owasp.org/index.php/SQL_Injection

⁷⁷ https://cheatsheetseries.owasp.org/cheatsheets/SQL_Injection_Prevention_Cheat_Sheet.html

⁷⁸ <https://www.ibm.com/developerworks/library/se-bindvariables/index.html>

⁷⁹ https://www.sqlite.org/c3ref/bind_blob.html

```
sqlite3_stmt* stmt;
sqlite3_prepare_v2(db, sql_query, 1, &stmt, NULL);
sqlite3_bind_text(stmt, 1, [studentName UTF8String], -1, SQLITE_TRANSIENT);
sqlite3_step(stmt);
sqlite3_finalize(stmt);
```

4.5.10.3.43 iOS Utilize Private Photo Access APIs



The app requests permission to read and write to the user's photo library through the `NSPhotoLibraryUsageDescription` key in the `Info.plist` file.

The user's photo library is one of the most sensitive stores of content on the device. To follow the principle of least privilege, the app should request the minimum access permissions necessary to fulfill its functionality.

Starting with iOS 11, the `UIImagePickerController` API allows the user to provide a single photo to the app, without the app requiring permission to read and write to the user's full photo library. As Apple states in [the photo access documentation](#)⁸⁰, this API does not require full photo library access:

"When using the `UIImagePickerController` to bring up the user's photo library, your app doesn't need to request permission explicitly."

If the App requires only granular or infrequent photo access, switch to the `UIImagePickerController` API to receive a photo, and remove the permission which grants the App full read and write access to the user's photo library.

4.5.10.3.44 iOS App Accesses Private APIs



Classes and symbols in iOS frameworks which are not exposed via the public iOS SDK are considered private, and accessing these APIs may result in automatic rejection from the App Store. Additionally, these APIs should be considered unstable, as they are undocumented and very often change between iOS versions. Their functionality is not guaranteed or even stated, and any usage of these APIs creates a potential instability vector.

It should be noted that this issue has no security impact, but may cause the app to be rejected by the App Store review team on the next submission.

Remove all references to private iOS APIs within your app. Do not use private classes or frameworks to trigger undocumented behavior. Find a way to achieve the desired goal using only the public iOS SDK.

⁸⁰ https://developer.apple.com/documentation/photokit/requesting_authorization_to_access_photos?language=objc

4.5.10.3.45 iOS App Built with Deprecated SDK

 The App was compiled with an iOS SDK older than iOS 12.1. As of **March 27, 2019**, [Apple will reject submissions and updates for Apps targeting an SDK version below iOS 12.1](#)⁸¹.

Building the App with an outdated SDK also means that the App will not ship with any security patches made within system frameworks used by the App. Additionally, the App will be unable to make use of modern APIs only present in the latest SDK.

The requirement of a current SDK is now enforced for both new apps and app updates.

Recompile the app with the highest iOS SDK version available and do not rely on functionality only present in older SDK versions. More information about leveraging the latest iOS SDK version while maintaining backward compatibility with older devices is available in [the Apple developer documentation](#)⁸².

If the app cannot be compiled using the latest SDK version, as of January 2020 at a minimum it must at least be built with the iOS 12.1 SDK or newer in order to avoid an App Store rejection, however this minimum will increase with new updates from Apple.

4.5.10.3.46 Android / iOS Application Logs Sensitive Data

 As a rule, applications should disable logging output (e.g. NSLog/Log.d) for production environments. Although logging output is necessary for development and debugging, sensitive data should never be logged in production builds.

There are multiple ways of satisfying this requirement however implementing a custom class which disables logging using compiler flags which switch off logging is recommended.

Swift Example Code

```
//Swift example custom logging with compiler flag
```

```
func stringFromAny(_ value:Any?) -> String {
    if let nonNil = value, !(nonNil is NSNull) {
        return String(describing: nonNil)
    }
    return ""
}
```

```
public func MSDLog(_ message:Any) {
```

⁸¹ <https://developer.apple.com/news/?id=03202019a>

⁸² https://developer.apple.com/library/content/documentation/DeveloperTools/Conceptual/cross_development/Configuring/configuring.html

```

    #if !NDEBUG
    NSLog("%@", stringFromAny(message))
    #endif
}

public func MSDLog(label: String, message: Any) {
    #if !NDEBUG
    NSLog("%@, %@", label, stringFromAny(message))
    #endif
}

```

4.5.10.3.47 Android JavaScript Interface Injection Vulnerability



WebViews that expose native objects to JavaScript code via `addJavascriptInterface` and load untrusted web content are vulnerable to JavaScript Interface Injection. The untrusted content can execute any of the exposed object's methods annotated with `@JavascriptInterface`, leading to data leakage, data corruption, or even arbitrary code execution.

The App does not explicitly block plaintext HTTP traffic, by setting `android:usesCleartextTraffic=false` or via a Network Security Config that disallows HTTP traffic in the App's Manifest, allowing a remote attacker to inject malicious JavaScript content in the WebView.

Regardless of whether the affected classes get invoked at runtime, Google Play is blocking any App that defines insecure WebViews, as detailed on [Google's support page](#)⁸³.

To prevent security issues with the App's WebView, follow these recommendations by Google:

1. Ensure that WebViews do not add Objects to the JavaScript interface. Ensure that there are no objects added to the JavaScript interface of any WebView that loads untrusted web content. You can do this in two ways: Ensure that no objects are ever added to the JavaScript interface via calls to `addJavascriptInterface`.

Remove objects from the JavaScript interface in `shouldInterceptRequest` via `removeJavascriptInterface` before untrusted content is loaded by the WebView.

2. Ensuring that WebViews with dangerous settings do not load untrusted web content. If a WebView needs to enable these dangerous settings, you must ensure that it does not load untrusted web content, which includes web content loaded over HTTP. You can set `android:usesCleartextTraffic=false` or place a "Network Security Config" that disallows HTTP traffic in your Manifest. Alternatively, you can ensure that any WebViews with dangerous settings does not load any URLs with HTTP schemes.

⁸³ <https://support.google.com/faqs/answer/9095419?hl=en-GB>

Example Code

```

@Override
public WebResourceResponse shouldInterceptRequest (final WebView view,
WebResourceRequest request) {
    Uri requestUri = request.getUrl();

    if (requestUri.getScheme().equals("http")) {
        webView.removeJavascriptInterface("jsinterface");
    }

    return super.shouldInterceptRequest(view, request);
};

```

4.5.10.3.48 iOS Weak ECB Mode Used for Symmetric Encryption Affected Components

Using ECB as the mode of operation for AES encryption can leak information about the content of the encrypted data, because identical plaintext blocks are encrypted into identical ciphertext blocks. Consequently, ECB does not hide data patterns well and does not provide proper message confidentiality; this is demonstrated in the screenshot which shows how patterns can leak information about the plaintext.

As a consequence, ECB mode is not recommended for use in cryptographic protocols at all.

As raw cryptographic algorithms are difficult to use securely due to subtle implementation flaws such as weak modes of operation or timing attacks, avoid directly using low-level cryptographic APIs (such as `CCCrypt`) to encrypt sensitive data.

Instead, consider relying on a high-level cryptographic library such as:

- Apple's [CryptoKit framework](#)⁸⁴ which is available starting with iOS 13.
- If older versions of iOS need to be supported, `RNCryptor` (<https://github.com/RNCryptor/RNCryptor>) can be used instead.

If AES needs to be used directly, rely on the Cipher Block Chaining (CBC) mode in order to hide patterns within the ciphertext and ensure that random Initialization Vectors are used for each cryptographic operation.

Example Code

```

// Example showing how to encrypt data with RNCryptor
NSData *data = [@"Data" dataUsingEncoding:NSUTF8StringEncoding];
NSError *error;

```

⁸⁴ <https://developer.apple.com/documentation/cryptokit>

```
NSData *encryptedData = [RNEncryptor encryptData:data
                               withSettings:kRNCryptorAES256Settings
                               password:aPassword
                               error:&error];
```

```
// Example showing how to encrypt data with CryptoKit
let data = "Data".data(using: String.Encoding.utf8)
let key = SymmetricKey(size: .bits256)
let sealedBox = try! ChaChaPoly.seal(data!, using: key)
let encryptedData = sealedBox.combined
```

4.5.10.3.49 iOS Null Initialization Vector Used for Symmetric Encryption



The vulnerable code locations perform AES encryption using the CBC mode. In this mode of operation, an Initialization Vector (IV) must be supplied to ensure that the same message encrypted with the same key multiple times produces completely different outputs. Hence, the IV needs to be rotated on every cryptographic operation to ensure a different output. The App instead uses a NULL IV, which never changes across different encryption operations. This allows for an example an attacker that is watching all the encrypted messages exchanged to identify when a previously sent message is sent again.

As raw cryptographic algorithms are difficult to use securely due to subtle implementation flaws such as weak modes of operation or timing attacks, avoid directly using low-level cryptographic APIs (such as `CCCrypt`) to encrypt sensitive data.

Instead, consider relying on a high-level cryptographic library such as:

- Apple's [CryptoKit framework](#)⁸⁵ which is available starting with iOS 13.
- If older versions of iOS need to be supported, `RNCryptor` (<https://github.com/RNCryptor/RNCryptor>) can be used instead.

Using a high-level library would avoid having to manually manage Initialization Vectors when implementing symmetric encryption.

Example Code

```
// Example showing how to encrypt data with RNCryptor
NSData *data = [@"Data" dataUsingEncoding:NSUTF8StringEncoding];
NSError *error;
NSData *encryptedData = [RNEncryptor encryptData:data
                               withSettings:kRNCryptorAES256Settings
                               password:aPassword
                               error:&error];
```

⁸⁵ <https://developer.apple.com/documentation/cryptokit>

```
error:&error];
```

```
// Example showing how to encrypt data with CryptoKit
let data = "Data".data(using: String.Encoding.utf8)
let key = SymmetricKey(size: .bits256)
let sealedBox = try! ChaChaPoly.seal(data!, using: key)
let encryptedData = sealedBox.combined
```

4.5.10.3.50 iOS App Store Blocker: App Uses UIWebView APIs



Apple has announced that it will stop accepting submissions of applications which use UIWebView APIs. The UIWebView API has been deprecated since iOS8.0 and contains several well known vulnerabilities.

Apple advises applications submitted after iOS8.0 should use the WKWebView class instead of UIWebView, and additionally setting the WKPreferences property `javaScriptEnabled` to false unless files accessed are intended to run JavaScript. A deadline after which applications which use UIWebView will not be accepted has not yet been announced by Apple.

See <https://developer.apple.com/documentation/uikit/uiwebview> for more information

4.5.10.3.51 Android / iOS App Embeds SQL Query with Dynamic Input



A variant of SQL injection vulnerability. the app uses dynamic SQL queries which were detected such as:

```
INSERT OR REPLACE INTO %@ (
SELECT count(*) from %@;
SELECT * FROM %@ WHERE %@=?
DELETE FROM %@
CREATE TABLE IF NOT EXISTS %@ (
```

These SQL queries are populated at runtime via format strings. If the string contents are derived via un-sanitized user input, an attacker may be able to perform an SQL injection attack. This may result in data leakage, data loss, or data corruption. As user input may be unsafe, it should not be directly included in database accesses. By utilizing prepared statements and bind variables, the app can perform dynamic SQL queries without directly including potentially unsafe user input in database accesses. Wherever possible, utilize parameterized inputs in dynamic SQL statements to mitigate the possibility of an SQL injection attack. Otherwise, ensure all user input is properly sanitized before including it in an SQL query.

For more information on this vulnerability and potential fixes see:

https://www.owasp.org/index.php/SQL_Injection
https://cheatsheetseries.owasp.org/cheatsheets/SQL_Injection_Prevention_Cheat_Sheet.html
<https://www.ibm.com/developerworks/library/se-bindvariables/index.html>
https://www.sqlite.org/c3ref/bind_blob.html
<https://download.oracle.com/oll/tutorials/SQLInjection/index.htm>

Sample Code

```
// Unsafe code, vulnerable to SQL injection
NSString* studentName = ...;
const char* sql_query = [NSString stringWithFormat:@"INSERT INTO students (name)
VALUES (%@), studentName] UTF8String];
sqlite3_stmt* stmt;
sqlite3_prepare_v2(db, sql_query, 1, &stmt, NULL);
sqlite3_step(stmt);
sqlite3_finalize(stmt);

// Safe code, using bound parameters
NSString* studentName = ...;
const char* sql_query = "INSERT INTO students (name) VALUES (?)";
sqlite3_stmt* stmt;
sqlite3_prepare_v2(db, sql_query, 1, &stmt, NULL);
sqlite3_bind_text(stmt, 1, [studentName UTF8String], -1, SQLITE_TRANSIENT);
sqlite3_step(stmt);
sqlite3_finalize(stmt);
```

4.5.10.3.52 Android / iOS Keys and Passwords Identified in the App



Any app which is publicly released can be downloaded by any member of the public, and potentially decompiled. Some technologies are harder to decompile than others, and some include their entire source code in plain text. As a result any key/password/secret needed for an app to function may be discovered by an end user and potentially could be abused.

In general keys, passwords, and other secrets should not be included in an app, however this can be unavoidable in certain cases (e.g. accessing Google Firebase, accessing Google Maps, etc.). Any key that is required to be in code in an app should be treated as potentially compromised.

Potential Remediation steps could be:

Is the key/password/secret required to be present in the code for an app to function? If not remove it.

Is the key part of Firebase configuration file for analytics use? If so this is a non-issue as having it implemented there is required by Google.

Is the key already publicly released or a false positive (e.g. Mapbox's hash values)? If so this is a non-issue but will flag in our vulnerability scans. Research before closing.

Is the key part of the Google APIs (e.g. Maps) or provided by a service that is restrictable? If so add restrictions based on the app's bundle ID to prevent use outside of the app.

Is it possible to leverage a proxy server which hosts the key? Whilst this hides the key itself it opens up potential issues, e.g. how the app would authenticate with that proxy server.

Is the key used as part of custom authentication services? If so don't rely on a key alone - consider the key to be analogous to wearing a company uniform - letting the remote service know that it is allowed to proceed further with the authentication dialog, but is not a promise of validity or part of the authentication process itself. Consider changing authentication methods to not require a local key to access a remote service.

Can the key be obfuscated or encrypted? This will potentially delay an attacker, but at somepoint the app would have to de-encrypt it to use it.

Regenerate API keys periodically to reduce their exposure.

4.5.10.3.53 iOS Missing Privacy Manifest

Apple's guidelines on privacy manifests: https://developer.apple.com/documentation/bundleresources/privacy_manifest_files

Resolution:

- 1) Check to see if the app has a privacy manifest that is included in the ipa - in source code if available and by examining the ipa (unzip and review).
- 2) Check that the privacy manifest matches what the app does and how it handles information (i.e. that all appropriate attributes are set), referencing the guide at https://developer.apple.com/documentation/bundleresources/privacy_manifest_files/describing_data_use_in_privacy_manifests
- 3) Check that any library, SDK, or framework also includes a privacy manifest.
- 4) If all above is good - attempt upload to Apple, Apple's systems will report if a privacy manifest is detected as needed but is not present.
- 5) If Apple do not report issues, close out as false positive as DataTheorem have a known issue with their detection process.

4.5.10.3.54 Android / iOS App Obfuscation Disabled - Classes, Keys, & Tokens are Clear-text

Applications should use built in code obfuscation tools. Android applications should leverage Android Studio's built in ProGuard tools for release builds, however iOS may risk rejection under guideline 2.3.1 if ipas are obfuscated by some methods.

It is recommended that additional steps for code obfuscation are taken for applications that handle sensitive data or perform sensitive actions (including information presentation).

For iOS, this includes using Data Theorem's Mobile Protect SDK, which will require an access key and a configuration file specific to our Data Theorem account.

Cocoapods example is at <https://datatheorem.atlassian.net/servicedesk/customer/portal/1/article/1790410753>

SPM example (recommended) is at <https://datatheorem.atlassian.net/servicedesk/customer/portal/1/article/1894875137>