WHITEPAPER

Mobile App Development Frameworks.

A security perspective.

mobisec

Table of contents

PREPARED BY
Ahmad Zubair Zahid

TECHNICAL SUPPORT
Harshul Vaishnav

PREPARED FOR
Mobisec

Table of contents

Preface

Choosing the right mobile framework in 2025 is not an easy task. With native options like Android and iOS, cross-platform technologies such as Flutter, React Native, and Ionic, and newer contenders like Kotlin Multiplatform (KMP), the decision involves far more than technical preferences. Performance, maintainability, development speed, and access to native features are all part of the equation — often overshadowing another critical aspect: security.

Each technology stack introduces its own set of tools, languages, and constraints — and with those, a distinct security profile. Security, in this context, means understanding how our technical choices shape the threat surface of a mobile application. The framework we choose directly impacts the types of attacks our apps are most exposed to.

As a mobile expert, I have worked with a wide variety of technologies, teams, and mindsets. One thing is clear: in a world where we practically live through our smartphones, protecting the data they hold is not optional — it is essential. Too often, the main business focus is on providing a great user experience and being competitive in terms of time to market — leaving cybersecurity out of the equation or limited to rushed, superficial configurations. In my experience, creating secure apps means being aware, staying well‑informed about the technology, and taking action at the right time.

That is exactly the purpose of the following paper by Mobisec. I appreciated its clear and structured approach to this complex landscape. It maps out the key security concerns that arise when building mobile apps — depending on the framework or platform chosen. This awareness is where any effective security process must begin. In a space often driven by trends, libraries, and development speed, this paper invites us to pause and reflect:

> *Are we making informed architectural decisions, or are we unknowingly introducing weaknesses into our apps?*

Security should be taken into account from the earliest stages of a project and become part of its very foundation. The insights in this paper will guide you in making more informed, secure choices — starting with the framework you build on.

GIORGIO BALDASSARRE

Mobile Lead Engineer

Introduction

Android is the most popular mobile operating system in the world, with an estimated 4 billion active users. Its widespread use is due to its open ecosystem and the wide variety of available devices, which makes it especially popular in regions where affordability and flexibility are key. Apple's iOS is the second most popular globally. It is strong in premium markets but has a smaller overall user base. For developers and businesses aiming to maximize their reach, targeting both Android and iOS is essential.

However, building for both platforms traditionally means keeping two separate codebases: one for Android, typically using Java or Kotlin, and another for iOS, using Swift or Objective-C. This approach doubles the workload, increases costs, and makes maintenance more complicated, especially when trying to ensure that features are the same and users have the same experience on different platforms. To address these challenges, cross-platform frameworks have been created that allow developers to write a single codebase which targets multiple platforms, including mobile, desktop, and web.

Some cross-platform frameworks, like Ionic and Cordova, rely on web technologies like HTML, CSS, and JavaScript. Apps built with these frameworks rely heavily on the WebView components, which are available on both Android and iOS. This method is suitable for apps that don't need high-quality graphics or high performance. It also makes adapting to web and desktop environments easier. However, it may affect performance and restrict access to some device features.

Other frameworks, such as Flutter, do things differently by packaging their own technology stack. Flutter, for example, uses the Dart language and can be compiled into native executables, bypassing the need for a WebView. It has its own rendering engine and communicates with the operating system through native channels.

Modern cross-platform frameworks can be used on more than just mobile devices. Many now support building applications for the web, desktop and even embedded systems from the same code base. This makes development and maintenance much easier. The choice between native and cross-platform development depends on the specific needs of the project. This includes performance requirements, what kind of user experience is wanted, and what development resources are available. For most business applications where it is important that the software is fast to develop, doesn't cost too much and is reliable across multiple platforms, cross-platform frameworks are a practical and effective solution.

This paper aims to help developers and product owners make informed decisions about whether to adopt cross-platform development for their projects. It will examine the strengths and limitations of popular cross-platform frameworks, compare them with native development approaches.

PREPARED BY
Ahmad Zubair Zahid

TECHNICAL SUPPORT
Harshul Vaishnav

PREPARED FOR
Mobisec

The platforms

With the advent of smartphones, two major platforms emerged: Android and iOS. While there are other mobile operating systems, these two now dominate the market.

Android

Android began as a project by Android Inc in 2003. Google acquired the company in 2005, which marked the start of Android becoming the world's leading mobile OS. Built on the Linux kernel, Android was designed for small devices like smartphones. From the start, the main programming language used to develop Android was Java. Components that are important for performance are often written in C or C++. In 2017, Google added official support for Kotlin.

Using Java or Kotlin to develop apps for Android is still the best and most well supported way. It gives developers immediate access to all new OS features, APIs, and system-level integrations, so they don't have to wait for third-party frameworks or plugins to catch up. Any new security improvements or platform features are available first for Java/Kotlin developers, and the many tools, libraries and documentation are optimized for these languages. The process model, application sandboxing and system interactions are all designed with this native stack in mind.

Choosing native Android development ensures the best performance, immediate access to features, and support for most tools. But this means developers need to keep separate versions of the code for each platform, and to be part of a team who knows how to use the different technologies. For organizations heavily invested in Android, or for products demanding deep system integration and long-term support, this remains the most robust and future-proof approach. Developers can also use a mix of different strategies, like adding Flutter modules to a native app.

Developing apps for Android using Java or Kotlin is the basic structure against which all other frameworks are measured, and any changes to security or architecture at the operating system level will first and foremost affect this development route. These changes will also have indirect effects on cross-platform solutions that ultimately rely on the same underlying mechanisms.

iOS

iOS is Apple’s tightly controlled, optimized OS for iPhones, iPads, and related devices. Native development uses Objective-C and Swift. Objective-C, a C superset with object-oriented features, was the original iOS language and is still key for legacy support and deep system integration. Introduced in 2014, Swift offers a safer, more concise, and efficient alternative.

Native iOS development with Swift or Objective-C, like Android development with Java or Kotlin, offers full access to platform features, new OS capabilities, and top performance and security. Built directly on Apple’s SDKs, it ensures immediate use of new APIs and hardware, smoother animations, faster load times, and better responsiveness crucial for performance-heavy apps like games, AR/VR, or those using advanced device features.

Native iOS development is the best way to get top performance and use new Apple features quickly. Cross-platform tools are helpful but still rely on native code. For teams who want reliable apps that work well with Apple devices, using Swift or Objective-C is the safest choice.

Cross-Platform Frameworks

Cross-platform frameworks such as Flutter, React Native, and .NET MAUI enable developers to use the same code for both iOS and Android, making development faster and cheaper. These frameworks are ideal for projects where quick delivery, code reuse, reaching a wide audience, and easy maintenance are important, since updates and bug fixes only need to be made once.

However, there are also some disadvantages. Cross-platform frameworks typically implement abstraction layers or bridging to interact with native APIs. On Android, for example, the app still starts with a native Java/Kotlin entry point (an Activity) that then controls the cross-platform engine. These added layers can result in reduced performance and increased latency, particularly if an app relies heavily on real-time processing or advanced hardware features.

Additionally, access to the latest operating system (OS) capabilities may be delayed, as framework maintainers need time to update their plug-ins and libraries after Apple or Google releases new features. Another disadvantage is that cross-platform apps tend to be larger than native apps, which can affect download times and storage usage.

Although cross-platform frameworks are growing quickly, they may still have problems with plugin compatibility, debugging and support for edge-case features may still occur. Furthermore, developers must exercise caution when using shared code, as weaknesses in common modules can be transferred to all platforms.

Cross-platform frameworks have improved significantly and now work almost as well as native code for many applications. However, certain advanced or security-sensitive applications still benefit from the control and speed of native code.

Flutter

Open Source: YES
Github starts: 170K

Flutter, which was developed by Google, has become one of the top cross-platform frameworks since it was launched in 2017. Flutter applications are written in Dart, another Google project. These applications are compiled directly to native binaries, which means they can run quickly and feel like other native apps on different platforms.

The framework comes with its own rendering engine, and ships as "libflutter.so" in each application. This design means every Flutter app has its own runtime, which makes the app bigger but lets developers update or fix the runtime separately from the operating system. This is a significant advantage over native apps, where updates to core frameworks typically depend on OS updates, which can be slow or fragmented, especially on Android.

The self-contained nature of Flutter apps has both benefits and drawbacks. The upside is that developers can add new features or fix security issues straight away, without waiting for device manufacturers or OS vendors to release updates. This is particularly useful in the Android ecosystem, where users have a lot of different versions of the operating system. The main disadvantage of this is that using the whole Flutter engine in every app makes the software files bigger and could possibly be a security risk. Since unlike modifying the device's native runtime, an attacker can alter the Flutter embedded runtime by repackaging the app and installing it, all without requiring root access. This is a risk that is shared by any framework that is not at the system level.

Flutter's technical strengths include hot reloading for fast changes, a wide and customizable widget library, and direct compilation to ARM code for almost native performance. The framework supports development for iOS, Android, web, desktop, and embedded platforms from a single codebase, making it a great choice for teams that want to reach lots of people and keep things running smoothly.

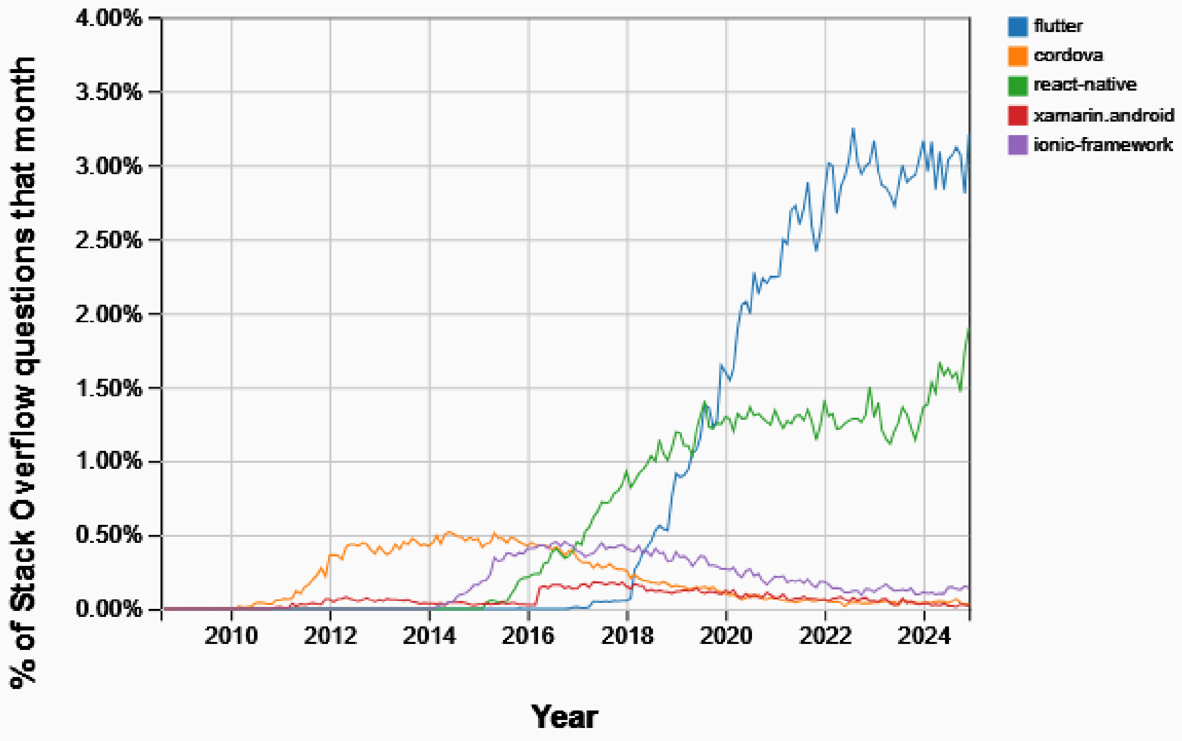
The number of people using Flutter is increasing quickly. It is used by millions of developers worldwide, powers hundreds of thousands of published apps, and is favored by major companies for its scalability and rapid development cycle. Its popularity is also evident in developer surveys and open-source activities. It is consistently one of the most loved and widely used frameworks.

Flutter compiles directly to native code, bypassing WebView and JavaScript bridge limitations found in other frameworks, resulting in faster, more responsive UIs and fewer vulnerabilities.

Flutter supports built-in security features such as code obfuscation, secure storage, and encrypted communications, and its modular architecture allows rapid deployment of updates and fixes. While its design makes reverse engineering more challenging, it is not immune to determined attackers, so robust security depends on careful implementation of best practices.

Looking ahead, Flutter's roadmap includes continued improvements in performance, deeper AI integration, expanded support for different type devices, and enhanced tooling for debugging and profiling. For organizations seeking a modern, scalable, and future-proof solution for cross-platform development, Flutter offers a compelling balance of performance, flexibility, and community support.

The graph below shows how Flutter has grown in popularity compared to other frameworks.



Source: *Tag Trends - Stack Overflow*

Here is a list of some of the companies that have adopted Flutter Framework: AliBaba, Baidu, iRobot, Groupon, eBay, Google Ads, Tencent, Toyota, Ubuntu and Philips Hue.

Cybersecurity Issues

Mobisec DSA assessment of some Flutter-based applications revealed several notable security issues. The applications are not compiled with common security flags such as Stack Canary and Position Independent Code (PIC), which are typically used to mitigate certain classes of memory exploits in native code.

There are also some concerns about network security, but to properly understand them, a brief introduction is needed: Flutter uses Dart, which doesn't rely on the system Certificate Authority (CA) store. Instead, Dart compiles its own list of trusted CAs directly into the application. This means that Dart is not inherently proxy-aware, and network proxies will not work unless the CA list is manually updated or bypassed, effectively providing a form of certificate pinning by default, which is the reason why many Flutter applications lacks certificate pinning or has an improperly configured implementation.

However, when it comes to WebViews within Flutter, these components do use the system CA store, which can leave them vulnerable to man-in-the-middle attacks. This means that manual certificate pinning shall be implemented to ensure equivalent security in both contexts.

React Native

Open source: YES
Github Stars: 122K

Under the mantle of Meta, React Native has become one of the most popular ways of developing mobile apps for different platforms since it was introduced by the former Facebook. It is built on JavaScript and TypeScript, which means that developers can write a single set of code that can be used on both iOS and Android. Unlike web-based frameworks, React Native renders actual native UI components, delivering a user experience and performance level that closely matches fully native apps.

The React Native ecosystem is large and active, backed by a strong community and many libraries. TypeScript has become the preferred choice for new projects, improving code reliability and maintainability. Major companies like Facebook, Instagram, Airbnb, and Shopify rely on React Native, proving its scalability and reliability in production.

Cybersecurity Issues

When it comes to security, there are some considerations compared to native or Flutter development. Its use of JavaScript makes the codebase inherently easier to reverse-engineer, even with obfuscation tools, compared to Flutter's compiled ARM code or native binaries. Data storage is another concern-React Native's default (now removed for community alternatives) AsyncStorage lacks built-in encryption, requiring developers to implement secure wrappers or rely on third-party solutions, whereas Flutter provides more robust, native-like secure storage APIs.

While React Native's new architecture has closed much of the performance gap with native solutions, the framework still lags when it comes to leveraging hardware-backed security features and providing the deep OS integration required for high-security applications. For projects where advanced security is a non-negotiable requirement, native development or Flutter remains the more robust choice.

Looking ahead, React Native's roadmap includes deeper AI integration to optimize the development experience, ongoing improvements in performance and tooling, and further enhancements to its already robust ecosystem. For organizations seeking a mature, flexible, and future-proof framework for building cross-platform mobile apps, React Native remains a compelling choice, balancing developer productivity with near-native user experience.

Here is a list of some of the companies that have adopted React Native Framework: Facebook, Instagram, Microsoft Office, Micorsoft Teams, Xbox Game Pass, Amazon Alexa, Shopify, WiX, Tableau, Klarna, Discord.

Ionic

Open source: YES
Github Stars: 57K

Ionic is also a mature and widely adopted cross-platform framework that leverages standard web technologies-HTML, CSS, and JavaScript-to enable developers to build applications for iOS, Android, and the web from a single codebase. An important part of Ionic is that it relies on a WebView, which is like a box that holds together the app's UI using web standards.

This approach allows for quick development, easy cross-platform deployment, and seamless integration with modern JavaScript frameworks such as React, Angular, and Vue. Developers can choose their preferred framework or even use Ionic without any framework at all. This makes it accessible to a wide range of teams and skill sets. Capacitor, Ionic's official native bridge, lets developers access native device APIs and plugins directly. This allows them to create custom native experiences when needed.

The framework has a vibrant ecosystem, with millions of monthly NPM installations, tens of thousands of GitHub stars, and a global community that actively contributes plugins, tutorials and support. Ionic is popular with both big companies and start-ups. It is used to make apps that can be updated quickly, have the same look and feel on different platforms, and are easy to take care of.

Cybersecurity Issues

When it comes to security and how easy it is to maintain, Ionic has the same strengths and limitations as other web-based technologies. It's important to follow the best security practices, like secure data storage, HTTPS enforcement, and regular dependency audits, as the WebView model can expose apps to the same vulnerabilities as traditional web applications if not properly managed. However, the framework's maturity and active community address security issues promptly.

When it comes to performance, Ionic has come a long way with its hardware acceleration and optimized rendering. But it might not be as fast or as deeply integrated into the operating system as other frameworks like Flutter or fully native development. For most business and productivity apps, this trade-off is fair, especially because development is quick and code can be shared easily. For applications with high graphics or that need to process data in real-time, native or cross-platform solutions may still be the best option.

Ionic's best features are that it is easy to maintain, flexible, and can give users a consistent experience across different platforms with little extra work. Its architecture is especially suited for teams who are very experienced in web development, projects that need to be deployed to the web and mobile at the same time, or scenarios where it is important to create a prototype quickly and make improvements. Ionic remains a compelling choice for a proven, scalable approach to cross-platform development.

Here is a list of some of the companies that have adopted Ionic Framework: CAT, BBC, Burger King, NBC.

.NET MAUI

Open Source: YES
Github Stars: 22K

Microsoft's .NET MAUI (Multi-platform App UI) has quickly become a popular choice in the cross-platform development landscape. .NET MAUI is based on Xamarin.Forms. It lets developers create native applications for Android, iOS, macOS and Windows. They can do this from a single C# codebase.

The architecture of .NET MAUI is designed for deep native integration. Applications built with MAUI leverage the .NET Base Class Library and run on the Mono runtime for Android, iOS, and macOS, and on the .NET Core CLR for Windows. MAUI provides a variety of UI controls, advanced layout engines, and cross-platform APIs for accessing device features such as GPS, sensors, and battery status.

Cybersecurity Issues

Security in .NET MAUI is robust, benefiting from the maturity of the .NET platform. The framework supports secure storage, encrypted communications, and regular updates through Microsoft's established security processes. But, as with any technology that works across different platforms, there are risks if third-party libraries or plugins are not vetted properly. It has recently been reported that malicious actors are exploiting MAUI's architecture to conceal and evade detection, particularly in Android malware campaigns.

Community engagement and ecosystem support for .NET MAUI are strong and growing. The framework is open to contributions; it includes a variety of free tools and professional user interface suites. Microsoft's regular updates and transparent roadmap provide confidence in the platform's long-term viability.

Compared to other cross-platform frameworks, .NET MAUI stands out for its smooth integration with native APIs, its ability to target both mobile and desktop from a single codebase, and its deep ties to the .NET ecosystem. While frameworks like Flutter and React Native offer strong cross-platform capabilities, MAUI is particularly good for organizations already using Microsoft technologies or looking for a strong, scalable solution for multi-platform development. For teams that prioritize maintainability, performance and access to the latest platform features, .NET MAUI offers a compelling balance of flexibility and power.

Here is a list of some of the companies that have adopted .NET MAUI Framework: Dynamics 365, Tyler Technologies, Azure App, Civica, NBC Sports Next; Fidelity Investments.

Honorable Mentions

Kotlin Multiplatform

Kotlin Multiplatform (KMP), made by JetBrains and officially approved by Google, is a technology that lets you share code across Android, iOS, web, desktop and more. Unlike full-stack frameworks, which conceal platform differences, KMP shares business logic and core functionality. It also allows developers to use native languages like Swift for iOS and Kotlin for Android to create platform-specific user interfaces and integrations. This approach keeps the performance and access to platform APIs the same, which is great for teams that need efficiency and flexibility.

KMP excels when you want to share business logic, data models, and networking code across platforms while keeping unique native UIs. Leading companies like Netflix, McDonald's, Baidu, Forbes, and Shopify use KMP to reduce code duplication, accelerate testing, and maintain performance. The ecosystem is rapidly growing, with strong Android Studio support and multiplatform libraries. KMP delivers native-level speed and full platform control, making it an effective choice for teams using Kotlin or seeking flexible, maintainable code.

Avalonia

Avalonia is an open-source, cross-platform UI framework for .NET that lets developers build desktop and embedded applications. Unlike frameworks that wrap native controls, Avalonia uses its own rendering engine, ensuring consistent interfaces across Windows, macOS, Linux, iOS, Android, and WebAssembly. This approach gives developers full control over the UI and visual behavior, making it well-suited for applications where consistency and performance matter. Avalonia is compatible with any .NET language and integrates well with existing .NET libraries and tooling.

Security and performance are comparable to other native .NET solutions, and the framework is a good candidate for teams with WPF experience or those seeking a modern, flexible .NET UI platform.

NativeScript

NativeScript, made by Telerik (now Progress Software), lets developers make native mobile apps using JavaScript or TypeScript, with direct access to native APIs. Unlike WebView-based frameworks, NativeScript renders native UI components, providing a more authentic user experience and better performance. It is compatible with popular frameworks like Angular and Vue, and is used by companies such as SAP, Puma, Deloitte, and Sennheiser for business applications.

NativeScript's workflow enables code reuse and faster development cycles, with gains in developer productivity and time-to-market. However, its popularity has waned, and its long-term viability is less certain compared to newer alternatives. The security considerations are like those of other JavaScript-based frameworks. This means developers need to be careful with dependencies, code obfuscation and secure storage practices.

Cordova

Apache Cordova, originally called PhoneGap, lets developers create mobile apps using web technologies like HTML, CSS, and JavaScript, packaging them in a native container with access to device features. This cross-platform approach and plugin ecosystem made Cordova popular for rapid prototyping and cost-effective development, with companies like Adobe and Microsoft using it for various solutions.

However, Cordova’s reliance on WebView limits performance and native integration, especially for graphics-heavy apps. Its plugin ecosystem is aging, community support has declined, and security risks are higher compared to modern frameworks like Flutter and React Native. Today, Cordova is mainly used for legacy apps, while most new projects favor faster, more robust alternatives.

Others Notable Frameworks

These frameworks are often limited by gaps in long-term support, smaller, fragmented communities, less mature security tooling and unpredictable update cycles or licensing issues.

- **Kivy**: An open-source Python framework for multi-touch and natural user interfaces, mainly used in education, prototyping and hobby projects due to its limited commercial support and security infrastructure.
- **LynxJS**: A JavaScript framework by ByteDance that uses Rust-based tooling for enhanced performance and native UI rendering. It is still in the early stages of being used and is not yet recommended for production.
- **Qt**: (Multiplatform Mobile App Development with Qt) Qt is a mature, high-performance C++ framework widely used in embedded and desktop systems and is deployed by large organizations. It offers enterprise-grade security features and is used in many security-sensitive domains. But Its licensing model and complexity can be challenging.
- **Uno Platform**: Extends WinUI/UWP-based development to Android, iOS, macOS, and WebAssembly, targeting teams with Microsoft XAML expertise.
- **Xamarin:** was a pioneering cross-platform framework. It enabled developers to build native apps for iOS, Android, and Windows using C# and the .NET ecosystem. With Microsoft ending support for Xamarin in 2024, new projects are encouraged to migrate to MAUI.
- **Tauri**: It's a new toolkit that puts security and small file sizes first. It has a Rust backend and a web-based frontend. It's ready for desktop use, but to date support for mobile devices is still being tested.

Suitability and Security Considerations

If GitHub stars are a measure of popularity and maturity, and we consider real-world adoption by established companies, native development remains the gold standard for performance, security and access to platform features. Among cross-platform solutions, Flutter leads momentum and adoption, followed by React Native. Ionic is a mature and widely used framework, but it relies on web technologies and WebView containers, which can affect performance and security. .NET MAUI, which is backed by Microsoft, is the newest of these, and has a lot of potential for business use, especially for teams that are already using the .NET ecosystem.

Here is a comparison of the top four frameworks, with a focus on security and how widely they are used:

Framework	Backed By	GitHub Stars (May 2025)	Year	Reverse Engineering Risk	SSL Pinning Support	Security Tooling/Notes
Flutter	Google	170K+	2017	Moderate (AOT, Dart)	Built-in for Dart HTTP, manual for WebView	Code obfuscation, secure storage, custom CA store, regular advisories
React Native	Meta	122K	2015	High (JavaScript)	3rd-party packages required	Requires JS obfuscation, secure storage via plugins, regular audits
Ionic	Ionic Team	57K+	2013	Very High (WebView)	Plugin-dependent	WebView inherits browser risks, relies on web security practices
.NET MAUI	Microsoft	22K	2022	Moderate (C# IL)	Native support (platform APIs)	.NET encryption, AOT, secure storage, regular MS security updates

Key Security Considerations

- Native frameworks always provide the earliest access to new platform security features and the highest level of integration.
- Flutter offers some resistance to reverse engineering and has a unique CA store for SSL, but it may lack certain native-level exploit mitigations unless configured.
- React Native is more likely to be reverse engineered and depends on the security of JavaScript and its ecosystem. Advanced protection requires extra tools.
- Ionic is most susceptible to vulnerabilities and reverse engineering because it inherits the attack surface of web applications.
- .NET MAUI uses the .NET security stack but requires code obfuscation to mitigate IL decompilation risks; it is well-suited for regulated industries with proper configuration.

Mobisec DSA Security Evaluation

Here are some other interesting results from the Mobisec DSA (Distributed Security Assessment) test of mobile apps for Android and iOS, as well as cross-platform frameworks. The evidences are split into two pages.

Security Issue	Description	Platform-Specific?	OS	Severity Level	Affected Frameworks	Impacted App %	Notes and Special Cases
Cleartext Traffic Support	Older Android devices allow cleartext traffic by default, and apps often do not explicitly disable it.	Framework-independent (OS issue)	Both	Medium	All	80% Android ~10% iOS	
Weak or Deprecated Cryptography	Use of outdated or weak hashing (SHA1, MD5), encryption algorithms (DES, ECB mode), or insecure PRNGs for cryptographic functions.	No, found across all frameworks	Both	Info level	All	90% Android ~50% iOS	Flutter notes AES CBC with PKCS5/PKCS7 may trigger a false positive
Use of Vulnerable Third-Party Libraries	Integration of external libraries that contain known security vulnerabilities (CVEs).	No, found across all frameworks	Both	Info level	All	80% Android ~40% iOS	
Insecure Data Storage	Sensitive data stored in shared preferences or unencrypted databases without proper cleanup upon logout. iOS backups often include this data.	No, found across all frameworks	Both	Low	All	~90%	
Security Provider Misconfiguration	Incorrect use of ProviderInstaller can prevent the app from using updated SSL libraries, making it vulnerable.	No, found across all frameworks	Android	Info level	All	~90%	Confirmed by Android
Insecure Deep Link Implementation	Deep links are not verified or are poorly configured, allowing unintended app behavior or access.	No, found across all frameworks	Android	Medium	All	~30%	
Sensitive Data Retained in Memory	Sensitive variables are not cleared (overwritten or nulled) after use, leaving data in memory.	No, found across all frameworks	Both	Low	All	90%	

Security Issue	Description	Platform-Specific?	OS	Severity Level	Affected Frameworks	Impacted App %	Notes and Special Cases
Unrestricted WebView Navigation and Caching	Apps do not whitelist domains or clear WebView caches upon logout, increasing exposure to malicious links.	No, found across all frameworks	Both	Medium	All	~50%	
Vulnerable to Tapjacking (Overlay Attacks)	Apps fail to check if user input occurs through overlays, allowing potential UI redress attacks.	No, found across all frameworks	Android	Medium	All	~50%	
Weak App Signing Algorithm or Key Size	Apps are sometimes signed using weak hash algorithms like SHA1 or insufficient key sizes (e.g., RSA with 1024 bits).	Android-specific issue	Android	Info level	All	~10%	
Missing or Improper Certificate Pinning	Apps either don't implement certificate pinning or configure it incorrectly.	Yes, for Flutter	Both	Medium	All except Flutter	~50%	Flutter uses Dart with a bundled CA list (acts like pinning). WebViews on Flutter still require manual pinning since they use system CA store.
Sensitive Data Logged	Logging sensitive information, especially in production environments, can expose user data.	No, found across all frameworks	Both	Low	All	10%	
Misconfigured Biometric Authentication	Android: CryptoObject is not used or is improperly implemented. iOS: Uses only Boolean from LocalAuthentication framework, which is patchable and lacks proof.	OS-specific implementation issue	Both	Low	All	75% of apps using the biometric feature	
Missing Native Security Features	Native libraries lack security flags such as Stack Canary, PIC, etc., reducing resilience against exploits.	Yes, for Flutter and Swift	Both	Info level	All	~30%	Flutter and Swift are memory-safe. React Native flagged but often a false positive (source).
Sensitive Data in Background Screenshots	When an app is backgrounded, the OS may take a screenshot of the UI. If sensitive data is visible, it could be leaked or accessed by spyware.	Framework-independent (OS issue)	Both	Low	All	~60%	

The Big Question

Choosing the right cross-platform framework is a decision that depends on a project's technical requirements, business goals, and long-term strategy. While the abundance of frameworks offers flexibility, this can also make things more complicated. There is no single choice that is always best. Instead, choosing the best solution depends on looking closely at several important factors that directly impact whether a project will be successful and sustainable or not.

The expected lifespan of the product is a key factor. For long-term or very critical applications, framework maturity and backing are crucial. Popular frameworks like Flutter (Google), React Native (Meta), and .NET MAUI (Microsoft) have strong support from big companies, are still being developed, and have clear plans for the future. This means that developers can be sure that the framework will be kept up to date and will work well with new operating system updates. Newer or less popular frameworks, even those from major companies, may not be stable or have all the features you need. They could have risks like undiscovered bugs or security flaws.

The industry context and specific regulatory challenges directly influence mobile framework selection. Regulations in sectors like healthcare and finance, though they don't mandate a particular framework, they do require the software to implement security features like encryption, access controls, audit logging, secure data transmission and more. These technical requirements mean that only frameworks capable of supporting or being extended to support such features are viable choices for regulated environments.

The company's technical set-up is also important. If a company already has a team who know C# and use Microsoft products, .NET MAUI may make it easier to make the change and lower the training costs. Also, organizations with strong web development expertise may find React Native or Ionic more approachable due to their JavaScript foundations.

The size of the community and the talent available are important things to take into account. Frameworks with large, active communities-such as Flutter and React-Native offer better access to libraries, learning resources, third-party integrations, and rapid bug fixes. This helps to solve problems more quickly and make development cycles more predictable. On the other hand, frameworks with small communities and support may have slower updates and less help from outside, which can make it hard to deal with issues.

Security is a big and important issue with many layers. Mature frameworks often have better security practices and tools, but each comes with its own risks. For example, Native Android apps (Java/Kotlin), React Native (JavaScript), and Xamarin (C#) are easier to reverse engineer because their code compiles to bytecode or intermediate formats that can be decompiled using common tools. Without obfuscation and anti-tampering, sensitive business logic can be exposed.

Some frameworks, like Flutter, compile to native ARM code, offering more resistance to reverse engineering, though they may lack the mature security tooling found in more established platforms.

In the end, the "best" framework is the one that matches your product's longevity, industry requirements, technical environment, available talent, and security posture. There is no clear winner – only the best option for your specific situation. Whatever the framework, secure storage, network encryption, and regular security testing are always essential.

Sources

Chapter "Flutter"

Tag Trend | Stack Overflow
Available at: https://trends.stackoverflow.co/?tags=flutter%2Ccordova%2Creact-native%2Cxamarin.android%2Cionic-framework

List of companies that uses Flutter
Available at: https://flutter.dev/showcase

Security:
- https://www.cve.org/CVERecord/SearchResults?query=flutter
- https://docs.flutter.dev/security
- https://docs.flutter.dev/reference/security-false-positives
- https://mas.owasp.org/MASTG/techniques/android/MASTG-TECH-0109/#intercepting-traffic-using-proxydroid-iptables-with-frida
- https://mas.owasp.org/MASTG/techniques/android/MASTG-TECH-0112/#using-blutter

Chapter "React Native"

List of companies that uses React Native
Available at: https://reactnative.dev/showcase

Security:
- https://reactnative.dev/docs/security
- https://security.snyk.io/package/npm/react-native
- https://owasp.org/blog/2024/10/02/Securing-React-Native-Mobile-Apps-with-OWASP-MAS

Chapter "Ionic"

List of companies that uses Ionic
Available at: https://ionic.io/

Security:
- https://ionicframework.com/docs/techniques/security

Chapter ".NET MAUI"

List of companies that uses .NET MAUI
Available at: https://dotnet.microsoft.com/en-us/platform/customers/maui

staysafe@mobisec.com

Mobisec

Viale della Repubblica 22 / III
Villorba (Treviso) - Italia

mobisec.com