GEORGE BROWN PRESS
JOURNALS, BOOKS & PROCEEDINGS

Journal of Computer, Signal, and System Research

Article

Permission Control and Security Improvement in Cross-Platform Mobile Application Development

Yajing Cai [1,*]

[1] Alexa Identity Service, Amazon.com Inc, Seattle, Washington, 98121, USA
* Correspondence: Yajing Cai, Alexa Identity Service, Amazon.com Inc, Seattle, Washington, 98121, USA

Abstract: With the increasing demand for mobile application development, cross-platform development architectures such as React Native and Flutter have become the mainstream, which make rapid development and unified user experience possible. However, cross-platform application rights management and security issues remain important challenges for developers. The purpose of this study is to examine the permission application mechanism in cross-platform applications, explore the permission management strategy between devices and applications, and analyze the technical implementation of dynamic permission management. At the same time, it also puts forward targeted security enhancement strategies, including but not limited to data encryption, OAuth 2.0 authentication, and defense against malicious code attacks, to provide developers with a set of practical security protection measures.

Keywords: cross-platform development; authority control; security enhancement

1. Introduction

In the field of software development for smart devices, the efficiency and maintenance advantages of cross-platform technology have attracted much attention. At present, React Native and Flutter are the most popular development frameworks in the market. However, developers face many challenges when it comes to ensuring the security and rights management of cross-platform software. Accurate permission control and efficient security policies are very important to ensure user information security and prevent data leakage. This study aims to analyze the permission application process of cross-platform software, explore the permission definition between devices and software, and propose effective strategies to enhance security on this basis, aiming at assisting developers to build more secure and reliable cross-platform software.

Received: 02 May 2025
Revised: 07 May 2025
Accepted: 23 May 2025
Published: 26 May 2025

Copyright: © 2025 by the authors. Submitted for possible open access publication under the terms and conditions of the Creative Commons Attribution (CC BY) license (https://creativecommons.org/licenses/by/4.0/).

2. Characteristics of Mainstream Cross-Platform Development Frameworks (Such as React Native and Flutter)

In the current field of cross-platform mobile application development, React Native and Flutter have become the two mainstream frameworks. React Native, built on JavaScript and React technology, enables developers to reuse code to efficiently deploy applications across multiple platforms. Its significant advantages include high development efficiency, strong community support, and abundant third-party library resources. However, React Native lacks in performance, especially in graphics and animation rendering, and there may be gaps compared with native applications [1].

In contrast, Flutter uses Google's Dart programming language to generate the user interface directly with its unique Skia graphics renderer, reducing the need to interface with native components, which helps to improve execution efficiency and image rendering speed. Flutter performs well in ensuring cross-platform user interface uniformity, as well as in graphics rendering, animation presentation, and performance metrics. Although Flutter is relatively difficult to learn, and developers need to be proficient in the

Dart language, its excellent cross-platform unity and near-native level performance make it excellent for developing high-standard mobile applications. Table 1 below compares React Native and Flutter's key features in terms of programming language usage, performance, application areas, and development efficiency to help developers make decisions based on the specific needs of the project [2].

Table 1. Comparative Analysis of React Native and Flutter.

Peculiarity	React Native	Flutter
Programming language	JavaScript	Dart
Performance advantage	Limited performance, dependent on native components	High performance, Native Rendering Engine (Skia)
Application scenario	Fast development, less native function requirements	Highly customized UI, high performance requirements
Development efficiency	High (suitable for developers with JS experience)	Intermediate (need to learn Dart language)

As can be seen from Table 1, React Native has advantages in terms of development efficiency and community support, which is suitable for fast development and teams with a Web development background. Flutter, on the other hand, has an advantage in terms of performance and UI consistency, which is especially suitable for projects with high user interface and performance requirements.

3. Permission Control Mechanism for Cross-Platform Mobile Applications

3.1. Permission Request Model and Permission Type Analysis in Cross-Platform Applications

The permission application mechanism of cross-platform applications mostly relies on the application-specific programming interfaces of the respective platforms. However, cross-platform development frameworks such as React Native and Flutter encapsulate these interfaces to achieve a unified interface for permission application. In the initialization stage of the program, the system puts forward the required permission request to the user according to the functional requirements of the program. These permissions can be roughly classified into general permissions, sensitive permissions, and special permissions. Common permissions, such as Internet access, log viewing, and system configuration adjustment, are usually requested at one time during application installation. Sensitive permissions, such as access to the camera, microphone, and location data, are associated with user privacy or device security. These permissions often need to be requested in real time while the application is running. In cross-platform development frameworks, permission management must interact with system APIs in a unified manner to ensure compatibility with the permission management mechanisms across different operating systems [3,4]. The following Table 2 summarizes the common permission types and their request mechanisms.

Table 2. Cross-Platform Permission Types and Their Request Mechanisms.

Permission type	Give an example	Request timing	Request mechanism
General authority	Network access, device information reading	At installation	One-time request at installation
Dangerous authority	Camera, microphone, location services	runtime	Dynamic request, user authorization required
Special authority	Modify system Settings and read SMS messages	Install time or run time	Additional permissions are usually required upon request

As can be seen from Table 2, there are differences in request timing and implementation methods for different types of permissions. General permissions are typically requested during the application deployment stage and may not require explicit user confirmation on some platforms; however, the specific behavior depends on the platform's permission management model. In contrast, sensitive permissions need to be requested in real time while the application is running. These permissions involve personal privacy and sensitive data, requiring stricter security controls. As for special permissions, they tend to be system-level permissions that are requested only when a particular feature is activated. Proper permission request policies can not only enhance user experience, but also effectively reduce redundant permission requests, thereby reducing potential security risks [5].

3.2. Differentiation and Management Policies between Device Permission and Application Permission

In the field of cross-platform software development, the distinction between device permissions and application permissions is well-defined. Device permissions mainly involve the permissions granted by the operating system, including the control of hardware resources such as cameras, microphones, and storage. In contrast, application permissions refer to the authorization required by the software at execution time, covering access to specific functions or services, such as notification push and background execution capabilities. Device permissions are often closely related to sensitive user information, which poses a potential threat to personal privacy. Therefore, special care should be taken when managing device permissions. For cross-platform applications, it is necessary to properly apply for the required device permissions according to the specific permission management system of different platforms, and strictly comply with the relevant regulations on the use of permissions. In the implementation of permission control policy, cross-platform applications should adopt the principle of minimum permission and permission granularity control. The principle of least privilege states that an application should request only the permissions necessary for its operation to prevent excessive or irrelevant permissions from being improperly requested. Permission granularity control precisely defines the scope of permissions according to specific needs, minimizing security risks associated with over-permissioning.

3.3. Dynamic Rights Management: Rights Request and User Authorization Process

During the execution of an application program, dynamic permission control applies for necessary permissions according to the user's operation or the specific requirements of the program. In particular, since Android version 6.0 (API 23), the Android system has adopted a runtime permission mechanism that forces applications to obtain explicit permission from users before invoking specific functions. Cross-platform development tools such as React Native and Flutter also follow this permission application mechanism. The process of dynamic permission control is divided into three steps: checking permission status, requesting permission, and handling authorization results. At the start of the program, the application first checks whether the necessary permissions have been granted. If not granted, it requests permission from the user. Depending on the user's response (yes or no), the application takes subsequent actions, such as explaining the importance of the permissions or disabling certain features. The following is a diagram of the flow of dynamic permission requests (Figure 1):

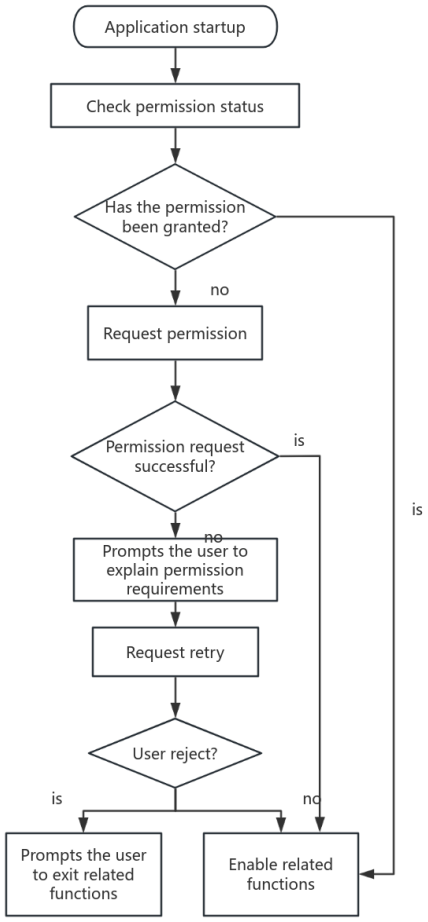

Figure 1. Dynamic Permission Request Flow.

3.4. Vulnerability and Security Risk Assessment of Cross-Platform Permission Control

While the rights management features of cross-platform software can bring convenience to developers, it also has security implications. On the one hand, since permission application methods vary across operating systems, developers may overlook platform-specific permission handling details during development, leading to excessive permission requests or leakage of permission information. On the other hand, the possible security flaws of the cross-platform technology framework itself can also be exploited to launch attacks. In terms of permission control, common risks include the abuse, promotion and bypass of permission. Abuse of permissions refers to applications that request unnecessary permissions that increase the likelihood of being attacked, especially when dealing with sensitive information such as storage and contacts. Privilege escalation means that an attacker may exploit vulnerabilities to obtain operational rights beyond their authorized permissions, causing malicious effects on applications or devices. Permission bypass refers to some cross-platform frameworks failing to fully account for the security mechanisms of each platform, allowing the permission checks to be bypassed, thereby increasing the security risks to applications. To enhance the security of cross-platform software, developers need to conduct regular security audits and risk assessments to ensure the robustness of the permission management system, and strengthen permission control through effective measures.

4. Strategies and Technologies to Improve Cross-Platform Application Security

4.1. Data Encryption and Secure Transmission Technology

Data encryption technology is the cornerstone of cross-platform application security, especially when data is transmitted over a network. In order to protect sensitive information from being illegally tampered with or stolen during storage or transmission, developers need to adopt a strong encryption strategy. Common encryption strategies in cross-platform software include symmetric encryption, asymmetric encryption, and secure transport protocols (such as TLS/SSL).

In symmetric encryption mechanisms, such as AES (Advanced Encryption Standard), the same key is used for both encryption and decryption. This method is popular for its fast encryption speed and computational efficiency, making it suitable for encrypting large amounts of data. However, the secret of the key is the core of security, once the key is leaked, the security of encrypted data will be completely broken. Asymmetric encryption methods, such as RSA (Rivest-Shamir-Adleman algorithm), use a pair of keys for operation, one for encryption and the other for decryption, which provides greater flexibility for key management and is very suitable for key exchange during secure communication.

In the cross-platform interaction scenario, the key is to ensure the security of data during transmission. Generally, the TLS protocol is used to maintain the confidentiality, integrity, and trust of data between networks. The TLS protocol integrates symmetric encryption and asymmetric encryption technologies to build a secure communication tunnel, effectively defending against man-in-the-middle attacks, illegal data tampering, and sensitive information disclosure risks. The formula of encryption technology in application is expressed as follows:

Symmetric Encryption (AES):

$$C = E(K, P) \tag{1}$$

Among them, C indicates the ciphertext, E indicates the encryption function, K indicates the encryption key, and P indicates the plaintext.

Asymmetric Encryption (RSA):

$$C = M^e \bmod n \tag{2}$$

Where C is ciphertext, M is plaintext, e is public key, and n is modular.

TLS encryption:

$$\text{Session Key} = E(K_{\text{server}},\ K_{\text{client}}) \tag{3}$$

TLS uses the above encryption algorithms to encrypt communication data through negotiated and shared session keys.

Through these encryption methods, cross-platform applications can ensure the security of data during storage and transmission, preventing data leakage and malicious tampering.

4.2. Authentication and Authorization Mechanism Based on OAuth 2.0 and JWT

In cross-platform applications, authentication and authorization are key technologies to ensure user information security and data access control. OAuth 2.0, as a popular authorization protocol for network applications, is becoming more and more important, especially when third-party applications need to obtain user resource rights. OAuth 2.0 uses the token system to enable users to authorize third-party applications to access their resources on different platforms without disclosing their passwords, greatly enhancing security.

JSON Web Token (JWT), as a token type frequently used in the OAuth 2.0 protocol, plays a central role in the process of information transfer between the client and the server. JWT consists of three parts: header, payload and signature, of which the signature part is responsible for ensuring the integrity of the information transmission process and the reliability of the source. JWT is widely used to store user authentication information and permission data, which not only improves the security of the system, but also brings higher scalability and flexibility.

The combination of OAuth 2.0 and JWT enables flexible and secure authentication and authorization control for cross-platform applications. The OAuth 2.0 authorization code flow and the JWT token generation formula are as follows:

$$\text{Access Token} = \text{OAuth 2.0 Authorization Code Flow}(Client, Server) \quad (4)$$

The construction formula of *JWT* is:

$$JWT = Base64(Header) + " \cdot " + Base64(Payload) + " \cdot " + HMAC - SHA256(Base64(Header) + \cdot + Base64(Payload), Secret) \quad (5)$$

The combination of these two protocols gives cross-platform applications the flexibility to manage user rights and avoid unauthorized access while ensuring user identity.

4.3. Preventing Malicious Attacks: Prevents Security Threats Such as XSS and SQL Injection

When applications are used across platforms, they often encounter various security risks, especially XSS attack and SQL injection attack. By embedding malicious scripts into applications, XSS attacks enable attackers to illegally obtain users' private data or even perform illegal operations without authorization. The key to effective defense against XSS attacks is to carefully screen and verify the data submitted by users, while ensuring that the output is properly encoded (for example, HTML encoding) to prevent the execution of malicious scripts.

By embedding harmful SQL commands into database query statements, attackers can evade security checks and achieve unauthorized access or tampering of database information. The key strategy to block such attacks is to adopt parameterized queries or preprocessed statements, even if the malicious user enters illegal data, it can not destroy the basic structure of the SQL statement.

In order to protect against multiple security threats such as XSS attacks and SQL injection, software developers must conduct strict screening and verification of user submitted data. Here are some specific defenses:

1) Implement a strict whitelist policy for user input to mask all unauthorized characters.
2) Perform HTML coding on all dynamically generated output content to prevent the implantation of malicious scripts.
3) Parameterized queries are preferred rather than directly concatenating SQL statements to reduce the risk of SQL injection.

The security formula for preventing XSS attacks can be expressed as:

$$\text{Sanitized Input} = \text{sanitize}(Input) \quad (6)$$

The best practice to prevent SQL injection is to use parameterized queries:

$$\text{Safe Query} = \text{Prepared Statment}(Input) \quad (7)$$

These protection technologies can effectively reduce the risk of malicious attacks faced by cross-platform applications and ensure application security.

4.4. Security Optimization for Cross-Platform Applications

In addition to encryption and authentication techniques, there are several strategies to enhance the security of cross-platform software. One common optimization technique is code obfuscation, which converts source code into an incomprehensible form and effectively prevents criminals from reverse-engineering critical code or stealing critical data. Code obfuscation not only increases the complexity of cracking, but also provides an additional safeguard for the security of the application.

Multi-factor authentication (MFA) is an enhanced authentication technology that requires users to provide additional authentication factors, such as SMS verification codes and biometrics, in addition to entering a password. Even if the password information is stolen, it is difficult for an attacker to hack into an account without multi-step verification. This multi-authentication approach significantly improves the security of the authentication process. Ongoing security checks and fixes for known defects are also key ways to improve application security. Software developers should periodically check the security

of applications to identify possible security risks and take corrective measures quickly. In this way, developers can effectively defend against hackers targeting known vulnerabilities and ensure the security and stability of applications.

Code obfuscation technology improves the difficulty of cracking, effectively preventing attackers from stealing key parts of programs through reverse engineering; Multi-factor authentication strengthens the stability of identity verification and reduces the risk caused by password leakage. Systematic periodic reviews and bug fixes ensure that cross-platform software can effectively withstand evolving security challenges throughout its lifecycle. With these diversified security strengthening measures, cross-platform software not only maintains the security of user information and privacy, but also enhances the protection against potential infringements, ensuring the long-term safe operation of the software.

5. Conclusion

In the context of the current popularity of cross-platform mobile applications, ensuring the security of these applications has become a key challenge that developers must face. This paper discusses several strategies and techniques for cross-platform applications in terms of permission management and security enhancement, including data encryption, authentication mechanism, anti-malicious intrusion measures and application security optimization approaches. By implementing effective encrypted transport, using authentication technologies such as OAuth 2.0 and JWT, and strengthening protection against risks such as XSS attacks and SQL injection, application security can be significantly enhanced. At the same time, regular security reviews and timely patching of security vulnerabilities are the basis for ensuring long-term stable application security. With the rapid advancement of technology, the security protection of cross-platform applications is an area that requires constant attention and innovation in order to effectively respond to increasingly complex security threats and protect users' privacy and data from being compromised.

References

1. M. Orosoo, I. Goswami, F. R. Alphonse, G. Fatma, M. Rengarajan, B. K. Bala, et al., "Enhancing Natural Language Processing in Multilingual Chatbots for Cross-Cultural Communication," in *2024 5th Int. Conf. Intell. Commun. Technol. Virtual Mobile Netw. (ICICV)*, pp. 127–133, Mar. 2024, doi: 10.1109/ICICV62344.2024.00027.
2. M. Tan, A. Liu, X. Wang, S. Shang, N. Wang, and X. Du, "A cross-domain access control mechanism based on model migration and semantic reasoning," *KSII Trans. Internet Inf. Syst.*, vol. 18, no. 6, pp. 1599–1618, 2024, doi: 10.3837/tiis.2024.06.010.
3. B. Seo, "A Case Study of Combining Two Cross-platform Development Frameworks for Storybook Mobile App," *KSII Trans. Internet Inf. Syst.*, vol. 17, no. 12, pp. 3345–3363, 2023, doi: 10.3837/tiis.2023.12.007.
4. A. A. Muhammad, A. Soliman, H. Zayed, A. H. Yousef, and S. Selim, "Automated library mapping approach based on cross-platform for mobile development programming languages," *Softw. Pract. Exper.*, vol. 54, no. 5, pp. 683–703, 2024, doi: 10.1002/spe.3281.
5. C. L. Chang, Y. L. Chen, and J. S. Li, "A cross-platform recommendation system from Facebook to Instagram," *Electron. Libr.*, vol. 41, no. 2/3, pp. 264–285, 2023, doi: 10.1108/EL-09-2022-0210.

Disclaimer/Publisher's Note: The statements, opinions and data contained in all publications are solely those of the individual author(s) and contributor(s) and not of GBP and/or the editor(s). GBP and/or the editor(s) disclaim responsibility for any injury to people or property resulting from any ideas, methods, instructions or products referred to in the content.