



# Improving Android Security: Application Security for Mobile App on Android 14 or Higher

Miss. Hemlata Chaudhari<sup>1</sup>, Asst. Prof. Sapana Fegade<sup>2</sup>

Research Scholar, Department of Computer Engineering, Shramasadhana College of Engineering & Technology

Jalgaon.MH, India<sup>1</sup>

Assistant Professor, Department of Computer Engineering, Shramasadhana College of Engineering & Technology

Jalgaon.MH, India<sup>2</sup>

**Abstract:** Mobile applications handle a large amount of sensitive information, making application security an important requirement for modern Android systems. This project, “Improving Android Security: Application Security for Mobile Apps on Android 14 and Higher,” focuses on improving the security of Android applications by integrating Android 14+ security features with secure development practices. The proposed approach includes requirement analysis, threat identification, secure application design, secure implementation, security testing, vulnerability analysis, deployment, and continuous maintenance. Security techniques such as least-privilege permissions, application sandboxing, secure authentication, data encryption, Android Key store, HTTPS/TLS communication, input validation, and protected application components are considered to reduce common security risks. Security testing methods such as static analysis, dynamic analysis, dependency scanning, API testing, and penetration testing can be used to identify potential vulnerabilities. The proposed defense-in-depth approach provides multiple layers of protection for user data and application resources and helps reduce the application's attack surface. Overall, the project demonstrates that integrating security throughout the complete Android application lifecycle can contribute to developing safer, more reliable, and better-protected mobile applications.

**Keywords:** Android Security, Android 14+, Mobile Application Security, Secure Application Development, Data Encryption, Android Key store, Authentication, Permission Management, HTTPS/TLS, Vulnerability Analysis, Security Testing, Application Protection.

## INTRODUCTION

Mobile applications have become an important part of communication, banking, education, health, commerce and organizational services. As application functionality increases, the amount of sensitive data handled by mobile software also increases. A security weakness in an application can expose user information, allow unauthorized actions, or create a path from the application to backend services. Therefore, application security must be considered during design, implementation, testing and release, rather than only after an incident occurs.

Android 14 introduces platform and behavior changes that developers must consider when building or updating applications. However, platform protection alone cannot make an application secure. Developers must correctly configure the application manifest, exported components, permissions, intents, Pending Intent usage, storage, authentication, authorization, networking, cryptography and dependencies. The source paper therefore proposes a lifecycle-oriented framework that connects Android 14+ security behavior with OWASP MASVS controls and repeatable testing.

The central motivation is to create a practical method that can be followed by developers and reviewers without treating a single security mechanism as sufficient. For example, encrypted local storage protects a device-side copy of a secret, but it does not replace server-side token validation. TLS protects data in transit, but it does not prevent an authorized user from abusing an API. Play Integrity can provide useful signals about application or device trust, but it does not replace authorization. Obfuscation can increase the cost of reverse engineering, but it should not be treated as a secret-storage mechanism. These examples support a defense-in-depth approach.

The objective of the research is to define and organize security controls for Android 14 or higher into a repeatable assessment lifecycle. The framework aims to reduce accidental exposure of application components, improve protection of credentials and sensitive data, strengthen authentication and authorization, improve network security, detect vulnerable dependencies earlier, and make privacy and release-integrity decisions more consistent.

The scope covers application-side security controls and their verification. It includes architecture and threat modelling, Android manifest configuration, component exposure, intent handling, Pending Intent configuration, storage,



authentication, authorization, network communication, cryptography, dependencies, privacy, integrity, testing and monitoring. The framework is intended as a practical baseline and not as a universal security score.

#### Objectives

- To identify common security threats in Android applications.
- To study the security features of Android 14 and higher.
- To protect sensitive application data using encryption and secure storage.
- To implement secure authentication and permission management.
- To secure communication using HTTPS/TLS.
- To test the application for security vulnerabilities.
- To improve the overall security of the Android application.

#### LITERATURE REVIEW

Tiwari et al. (2015) studied permission-based Android security and its associated problems. Their research identified issues such as over-privileged applications, permission escalation, insufficient permission documentation, and weaknesses in permission administration. The authors reviewed existing countermeasures and suggested improvements to reduce security and privacy risks. Their work is important for the present study because permission management remains a fundamental part of Android application security.

Tse et al. (2014) presented Strategies in Improving Android Security, which examined Android's architecture, applications, Google Play Store and security threats. The authors discussed both technical measures and user awareness as approaches for improving Android security. Their research established that Android security requires a combination of platform-level protection, secure application development and informed user behavior.

Shrivastava, Kumar, Gupta and Rodrigues (2020) conducted a literature review on privacy issues related to Android application permissions. Their study examined research published over approximately ten years and analyzed different approaches for detecting and controlling inappropriate permissions. The researchers highlighted that permission analysis plays an important role in protecting sensitive user information from untrusted applications. This work supports the use of the principle of least privilege in modern Android applications.

Reardon et al. (2019) investigated how applications can bypass Android's permission system. Their study, 50 Ways to Leak Your Data, demonstrated that applications may use covert and side channels to access or transfer protected information. The researchers showed that simply having a permission-based security model does not guarantee complete protection of user data. Their findings emphasize the need for additional application-level security mechanisms and proper security testing.

Peck, Kini and Pyles (MITRE, 2016) analysed Android's security architecture and investigated vulnerabilities that could allow malicious applications or network attackers to exploit application weaknesses. Their work examined important Android mechanisms such as application sandboxing, permissions and restrictions on application access to sensitive device resources. The study demonstrated that although Android's architecture provides significant protection, vulnerabilities in applications can still create opportunities for attacks.

Shrivastava et al. and other researchers studying Android data storage security have highlighted insecure data storage, over-privileged applications and misuse of cryptographic APIs as important sources of Android security problems. Their research shows that developers play a major role in protecting application data because inappropriate storage and cryptographic practices can expose sensitive information.

Ehsan Catal and Mishra (2022) conducted a systematic literature review on Android malware detection using application permissions. Their research analyzed studies using static analysis, dynamic analysis and hybrid approaches for identifying malicious applications. They found that malware can request unnecessary permissions to obtain access to sensitive resources, making permission analysis an important method for detecting potentially malicious applications.

Güliz Seray Tuncay (2024) studied the evolution of Android's permission system and identified permission-related vulnerabilities. The study examined how Android's permission model has changed over time and provided recommendations for developers to reduce permission-related security and privacy risks during application development. This research is particularly relevant to modern Android security because the permission model continues to evolve with newer Android releases.

Solanki, Laxmi and Gaur (2026) conducted a comprehensive survey of Android's permission-based security model. Their research reviewed the evolution of Android permissions and related research, including API-to-permission mapping, permission checking and unresolved permission-related challenges. Their findings indicate that although



Android has moved from an older all-or-nothing permission approach toward more user-controlled permissions, several security challenges remain.

Heid, Sonntag and Heider (2026) investigated permission exploitation through third-party libraries in Android applications. Their work highlights the fact that security risks can also originate from external libraries integrated into applications. This is particularly relevant for modern Android applications because applications commonly depend on third-party SDKs and libraries.

### Research Gap:

Many researchers have studied Android security, including permissions, malware, privacy, data storage and third-party libraries. These studies show that Android applications can face different security risks. However, most studies focus on one security problem at a time. There is a need for a simple and combined approach to improve the security of modern Android applications.

Therefore, this research focuses on important application security issues and security practices for mobile applications running on Android 14 or higher.

## METHODOLOGY

### 1.1 Requirement Analysis

The first stage is to identify the functional and security requirements of the Android application. This includes understanding what the application should do and what type of data it will process or store.

The requirements related to data protection, authentication, permissions, network communication, and application components are identified. Security requirements are defined according to the sensitivity of the information handled by the application and the Android 14+ environment.

### 1.2 Threat Identification

In this stage, possible security threats and attack surfaces are identified. The application is analyzed to determine where unauthorized access, data leakage, malicious input, or other attacks could occur.

Threats related to authentication, insecure storage, excessive permissions, network communication, exposed components, and malicious input are considered. The potential effect of each threat on the confidentiality, integrity, and availability of application resources is analysed.

### 1.3 Secure Application Design

After identifying the threats, a secure architecture is designed. Security is incorporated into the application before implementation begins.

The design applies the principle of least privilege, where the application receives only the permissions required for its functionality. Secure mechanisms for data storage, encryption, authentication, network communication, and application components are also planned.

The design also considers the protection of Android components such as Activities, Services, Broadcast Receivers, and Content Providers.

### 1.4 Secure Implementation

In this stage, the designed security mechanisms are implemented in the Android application. The application uses appropriate Android security APIs and secure programming practices. Android Key store can be used for cryptographic key management, while HTTPS/TLS is used to protect network communication. Input validation is implemented to prevent malicious data

Sensitive information such as passwords, private keys, and API credentials should not be hard-coded into the application. Secure coding practices are followed throughout development.

### 1.1 Security Testing

After implementation, the application undergoes security testing to identify weaknesses and verify that the implemented security controls work correctly.

#### Testing can include:

- 1.1.1 Static Application Security Testing (SAST)
- 1.1.2 Dynamic Application Security Testing (DAST)
- 1.1.3 Permission testing
- 1.1.4 Network security testing



- 1.1.5 Storage security testing
- 1.1.6 Component and intent testing
- 1.1.7 APK analysis
- 1.1.8 Authentication and authorization testing

## 1.2 Vulnerability Analysis

The results obtained from security testing are analyzed in this stage. Identified vulnerabilities

Sensitive information such as passwords, private keys, and API credentials should not be hard-coded into the application. Secure coding practices are followed throughout development.

## 1.3 Security Testing

After implementation, the application undergoes security testing to identify weaknesses and verify that the implemented security controls work correctly.

**Testing can include:**

- 1.3.1 Static Application Security Testing (SAST)
- 1.3.2 Dynamic Application Security Testing (DAST)
- 1.3.3 Permission testing
- 1.3.4 Network security testing
- 1.3.5 Storage security testing
- 1.3.6 Component and intent testing
- 1.3.7 APK analysis
- 1.3.8 Authentication and authorization testing

## 1.4 Vulnerability Analysis

The results obtained from security testing are analyzed in this stage. Identified vulnerabilities are examined according to their type, severity, and potential impact on the application.

For example, vulnerabilities related to insecure data storage, weak authentication, excessive permissions, insecure network communication, or improperly exposed components are documented.

The analysis helps determine which vulnerabilities require immediate correction and which security controls need improvement.

## 1.5 Security Improvement

After vulnerabilities are identified, appropriate corrective measures are implemented. Security configurations are improved, insecure code is corrected, unnecessary permissions are removed, and vulnerable dependencies are updated. The application is then tested again to verify that the identified problems have been addressed.

**This creates an iterative process:**

Test → Identify Vulnerability → Fix → Re-test → Improve.

## 1.6 Development and Deployment

Once the application satisfies the required functional and security requirements, a release version is prepared for deployment.

Before deployment, unnecessary debugging information is removed, permissions and application components are reviewed, and the application is digitally signed. The release build can then be distributed through an appropriate application distribution platform.

Security considerations remain important during deployment because incorrect configuration can introduce new security risks.

## 1.9 Maintenance

Security does not end after deployment. The final stage is continuous maintenance and monitoring.

The application should be monitored for newly discovered vulnerabilities. Dependencies and libraries should be updated regularly, security patches should be applied, and the application should be tested again when major changes are introduced.

The application should also be reviewed for compatibility with newer Android versions and changing security requirements.



Figure 1. Methodology Workflow Diagram

The developed security approach includes secure authentication, encryption, secure data storage, permission management, secure network communication, protected application components, input validation, code obfuscation, and security testing. These mechanisms work together to provide a layered security approach for Android applications.

### 2.1. Secure Authentication

The application uses secure authentication mechanisms to ensure that only authorized users can access protected resources. Strong authentication reduces the possibility of unauthorized access and helps protect user accounts and application data.

### 2.2. Data Encryption

Encryption is used to protect sensitive information from unauthorized access. Important data can be encrypted before storage or transmission. The Android Keystore can be used for secure management of cryptographic keys.

### 2.3. Secure Data Storage

Sensitive information such as credentials, authentication tokens, and personal data should not be stored in plain text. Secure storage techniques help prevent sensitive information from being easily accessed if the application or device is compromised.

### 2.4. Permission Management

The application follows the principle of least privilege by requesting only the permissions necessary for its functionality. Proper permission management reduces unnecessary access to sensitive device resources such as location, camera, microphone, contacts, and storage.

### 2.5. Secure Network Communication

Secure communication protocols such as HTTPS/TLS are used for communication between the application and remote servers. This helps protect transmitted information against interception and unauthorized modification.

### 2.6. Protected Application Components

Android components such as activities, services, broadcast receivers, and content providers are properly configured. Unnecessary component exposure is avoided, and external input is validated before it is processed.

### 2.7. Input Validation

Input validation is implemented to ensure that data received from users or external sources is checked before processing. This helps reduce the risk of malicious input and injection-related vulnerabilities.

### 2.8. Code Obfuscation

Code obfuscation and shrinking can make reverse engineering more difficult. Tools such as R8 can be used to reduce unnecessary code and make application code harder to understand. However, sensitive passwords and cryptographic keys should never be hard-coded into the application.

### 2.9. Security Testing

Security testing is performed to identify possible vulnerabilities in the application. Testing can include static analysis, dynamic analysis, permission testing, network testing, storage testing, component testing, and APK analysis.



Figure 2. Methodology Workflow Diagram

### Evaluation Criteria And Discussion

The proposed framework can be evaluated using control coverage, number and severity of findings, remediation time, regression rate and security-test repeatability. A useful study design is to assess an application before and after applying the framework. Dependent variables can include the number of high/critical findings, percentage of MASVS controls with evidence, failed security tests and mean remediation time.

Because the source paper does not introduce a newly collected production dataset, numerical effectiveness values should not be assumed. Instead, the methodology should be implemented on selected applications and the results should be measured. A strong future evaluation would compare multiple Android applications, classify weaknesses by MASVS group, and calculate pre/post security scores.

#### A. Expected Security Improvements

- Reduced accidental exposure of internal Android components.
- Lower probability of insecure Pending Intent and implicit-intent handling.
- Improved protection of local credentials, tokens and sensitive application data.
- Stronger authentication and server-side authorization practices.
- Better resistance to network interception and unsafe API communication.
- Earlier detection of vulnerable dependencies and insecure coding patterns.
- More consistent privacy and permission decisions.
- Improved release integrity and post-release monitoring.

#### B. Security-by-Design Discussion

Security controls are most effective when they are layered. Secure storage protects a device-side copy of a secret but does not replace server-side token validation. TLS protects data in transit but does not prevent an authorized user from abusing an API. Play Integrity can provide useful application or device trust signals but does not replace authorization. Obfuscation can raise the cost of reverse engineering but should not be used as a secret-storage mechanism. These relationships demonstrate why ASALF favors defense in depth.



## Practical Security Checklist

| Area           | Checklist item                                                                                                |
|----------------|---------------------------------------------------------------------------------------------------------------|
| Architecture   | Threat model completed; trust boundaries documented; security requirements identified.                        |
| Manifest       | Only required components exported; permissions minimized; provider access controlled.                         |
| Intents        | Internal calls use explicit intents; external input validated; sensitive extras minimized.                    |
| Pending Intent | Immutable by default; explicit target; mutability justified and controlled.                                   |
| Storage        | Sensitive values protected; keys managed securely; logs do not expose secrets.                                |
| Authentication | Modern credentials supported; local/biometric authentication used where appropriate; session expiry enforced. |
| Authorization  | Backend independently verifies user, role and resource ownership.                                             |

## LIMITATIONS AND FUTURE WORK

The principal limitation is that the source paper presents a framework and assessment criteria rather than a large-scale experimental study on a controlled set of production applications. Security outcomes depend on application architecture, backend design, business risk and device configuration. The scoring approach is therefore best treated as a prioritization method, not as a universal security metric.

Future work can implement ASALF in a sample Android application and perform a pre/post assessment. Automation is another important direction: manifest analysis, dependency scanning, static code checks, API security tests and MASVS evidence collection can be integrated into CI/CD pipelines. Further research can investigate how Android 15 and later platform changes affect the same security controls and whether security findings can be predicted from architectural features.

## CONCLUSION

Android 14 or higher provides strong platform-level security mechanisms, but secure mobile applications still depend on correct developer decisions. The proposed approach combines multiple security techniques, including application sandboxing, least-privilege permissions, secure authentication, encrypted data storage, Android Keystore, HTTPS/TLS communication, secure application components, input validation, dependency management, and code protection. These mechanisms provide multiple layers of security and help protect application data and resources from unauthorized access.

## REFERENCES

- [1] D. Tse, X. Liu, C. Nusaputra, B. Hu, Y. Wang, and M. W. Xing, "Strategies in Improving Android Security," Proceedings of the 18th Pacific Asia Conference on Information Systems (PACIS), 2014. Paper Link
- [2] R. T. Tiwari et al., "Permission based Android security: Issues and countermeasures," Computers & Security, vol. 43, pp. 205–218, 2014, doi: 10.1016/j.cose.2014.02.007. Paper Link
- [3] J. Reardon, Á. Feal, P. Wijesekera, A. Elazari Bar On, N. Vallina-Rodriguez, and S. Egelman, "50 Ways to Leak Your Data: An Exploration of Apps' Circumvention of the Android Permissions System," Proceedings of the 28th USENIX Security Symposium, pp. 603–620, 2019. USENIX Paper.
- [4] A. Ehsan, C. Catal, and A. Mishra, "Detecting Malware by Analyzing App Permissions on Android Platform: A Systematic Literature Review," Sensors, vol. 22, no. 20, p. 7928, 2022, doi: 10.3390/s22207928. Paper Link
- [5] G. S. Tuncay, "Android Permissions: Evolution, Attacks, and Best Practices," IEEE Security & Privacy, vol. 22, no. 6, pp. 40–49, 2024, doi: 10.1109/MSEC.2024.3461629. IEEE/DOI Link
- [6] J. Senanayake, H. Kalutarage, M. O. Al-Kadri, A. Petrovski, and L. Piras, "Android Source Code Vulnerability Detection: A Systematic Literature Review," ACM Computing Surveys, vol. 55, no. 9, 2023, doi: 10.1145/3556974.
- [7] S. Garg and N. Baliyan, "Android Security Assessment: A Review, Taxonomy and Research Gap Study," Computers & Security, vol. 100, 2021, Art. no. 102087.
- [8] S. Acharya, U. Rawat, and R. Bhatnagar, "A Comprehensive Review of Android Security: Threats,



Vulnerabilities, Malware Detection, and Analysis,” Security and Communication Networks, 2022, doi: 10.1155/2022/7775917.

- [9] J. Qiu, J. Zhang, W. Luo, L. Pan, S. Nepal, and Y. Xiang, “A Survey of Android Malware Detection with Deep Neural Models,” ACM Computing Surveys, vol. 53, no. 6, 2020, doi: 10.1145/3417978.
- [10] T. Sharma and D. Rattan, “Malicious Application Detection in Android—A Systematic Literature Review,” Computer Science Review, vol. 40, 2021, Art. no. 100373.