SAFE APP STANDARD

CSA SINGAPORE

Cyber Security Agency of Singapore

CSA's Safe App Standard

Version 1.0

Released January 10th, 2024.

In Consultation With:

The Association of Banks Singapore, Standing Committee on Cyber Committee

Deloitte Southeast Asia Risk Advisory

Ernst & Young Advisory Pte. Ltd.

KPMG in Singapore

Lazada

Microsoft Singapore

PricewaterhouseCoopers Risk Services Pte. Ltd.

Disclaimer:

Contents

About the Standard

Introduction

The Safe App Standard is a recommended standard for mobile applications (apps), developed by the Cyber Security Agency of Singapore (CSA) in consultation with industry partners from financial institutions, tech organisations, consultancy firms, and government agencies.

Overview

The objective of the Standard is to put forward a recommended baseline of security controls for mobile app developers and providers to follow. This would ensure that all local apps adhere to a similar set of security controls for mobile apps, thereby raising the security levels of the apps hosted and created in Singapore.

Purpose, Scope, and Intended Audience

This document was developed to provide recommendations and suggestions to developers to assist them in implementing security functions into their apps. Such recommendations and suggestions are aimed towards assisting developers in mitigating against a broad spectrum of cybersecurity threats and protect their apps from the latest mobile scams and mobile malware exploits. The contents herein are non-binding, provided on a non-reliance basis and meant to be informative in nature, and are not intended to exhaustively identify potential cybersecurity threats nor exhaustively specify processes or systems that developers should put in place to address or prevent such threats.

Version 1 of Safe App Standard's guidelines and security controls will focus primarily on providing security guidelines to developers of high-risk apps to counteract the latest mobile malware and scam exploits seen in Singapore's threat landscape. However, these security controls can also benefit and be implemented by other apps. It is recommended that all developers strive to implement these measures for enhanced mobile app security.

Although this Standard has a primary focus area, future iterations will expand to address security best practices and guidelines for the entire mobile app stack.

Notice and Developer Guidance

This is a living document that will be subjected to review and revision periodically. Like many other established standards, the Safe App Standard is a living document that will be regularly updated to match the current and evolving threat landscape and new attack vectors. Please refer to CSA's website to stay updated with the latest version of the Safe App Standard and adapt security measures and controls accordingly.

This Standard should be read in conjunction with and does not replace, vary, or supersede any legal, regulatory, or other obligations and duties of app developers and providers, including those under the Cybersecurity Act 2018, and any subsidiary legislation, codes of practice, standards of performance, or written directions issued thereunder. The use of this document and implementation of the recommendations herein also does not exempt or automatically discharge the app developer and provider from any such obligations or duties. The contents of this document are not intended to be an authoritative statement of the law or a substitute for legal or other professional advice.

Developer guidance on the Safe App Standard security framework

For ease of use, developers should take note that Version 1 of the Safe App Standard targets the following critical areas, and the document itself can be split into the following subsections:

- **Authentication**
- **Authorisation**
- **Data Storage (Data-at-Rest)**
- **Anti-Tamper & Anti-Reversing**

These critical areas are included to ensure the standardisation of mobile app security against the most common attack vectors used by malicious actors in our local ecosystem. The Safe App Standard provides a clear and concise set of security controls, guidelines, and best practices for enhancing the security of mobile apps that provide or enable high-risk transactions.

Document Definitions and Normative References

Document Definitions

The following are some definitions that developers and readers should keep in mind as they utilise this document:

Sensitive data – *User data such as Personal Identifiable Information (PII) and authentication data such as credentials, encryption keys, one-time passwords, biometric data, security tokens, certificates, etc.*

High-risk transactions are those that involve:

- **Changes to financial functions –** *some examples include but are not limited to registration of third-party payee details, increase of fund transfer limit, etc.*
- **Initiation of financial transactions** – *some examples include but are not limited to high-value funds transactions, high-value funds transfers, online card transactions, direct debit access, money storage functions, top-ups, etc.*
- **Changes to the application's security configurations–** *some examples of this include but are not limited to disabling authentication methods, updating digital tokens or credentials, etc.*

Security controls – Operational or technical measures recommended in this document that should be implemented to manage, monitor, and mitigate potential security vulnerabilities or incidents. These security controls have the following IDs attached to them, e.g., AUTHN-BP01, AUTHOR-BP01, STORAGE-BP01, RESILIENCE-BP01.

Normative References

The Safe App Standard references industry standards from the Open Web Application Security Project (OWASP), the European Union Agency for Network and Information Security (ENISA) and the Payment Card Industry Data Security Standard (PCI DSS).

The list of references is as follows:

- OWASP's MASVS (Mobile Application Security Verification Standard)
- OWASP's MASTG (Mobile Application Security Testing Guide)
- ENISA's Secure Development Guidelines (SSDG)
- PCI DSS' Mobile Payment Acceptance Security Guidelines for Developers

01 AUTHENTICATION

1. Authentication

Introduction

Authentication is an essential component of most mobile applications. These applications commonly employ various forms of authentication, including biometrics, PINs, or multi-factor authentication code generators. Ensuring the authentication mechanism is secure and implemented following industry best practices is crucial to validate user identity.

By implementing robust security controls for authentication, developers can ensure that only authenticated users, clients, applications, and devices can access specific resources or perform certain actions. Through secure authentication controls, developers can also mitigate the risk of unauthorised data access, maintain the integrity of sensitive data, uphold user privacy, and protect the integrity of high-risk transaction functionalities.

The controls in this category aim to recommend authentication security controls that the application should implement to safeguard sensitive data and prevent unauthorised access. It also gives developers relevant best practices to implement these security controls.

Security controls

ID	Control
AUTHN-BP01	Use Multi-Factor Authentication to authenticate high-risk transactions.
AUTHN-BP01a	Implement Something-You-Know authentication as one of the MFA factors.
AUTHN-BP01b	Implement Something-You-Have authentication as one of the MFA factors.
AUTHN-BP01c	Implement Something-You-Are authentication as one of the MFA factors.
AUTHN-BP02	Use context-based factors to authenticate.
AUTHN-BP03	Implement secure session authentication.
AUTHN-BP03a	Implement secure stateful authentication.
AUTHN-BP03b	Implement secure stateless authentication.
AUTHN-BP04	Implement secure session termination during logout, inactivity, or application closure.
AUTHN-BP05	Implement brute force protection for authentication.
AUTHN-BP06	Implement transaction integrity verification mechanism.

AUTHN-BP01

Control

The app uses Multi-Factor Authentication (MFA) to authenticate high-risk transactions.

Description

In a traditional single-factor authentication system, users typically only need to input Something-You-Know[1], such as usernames and passwords. However, if this single factor fails or is compromised, the entire authentication process is vulnerable to threats.

MFA is an authentication procedure that adds layers of identity verification, requiring not only Something-You-Know but also Something-You-Have[2] and Something-You-Are[3]. Implementing MFA makes it more challenging for malicious actors to compromise accounts and enhance the overall security of the authentication process.

Implementation guidance

Developers should use Step-up MFA. It is a specific type of MFA where the app incorporates an authentication strategy that requires an additional authentication level, especially when attempting higher-risk transactions.

Developers should prioritise the following MFA combinations in the order of 1, 2, 3, and 4, with option 1 as the most secure choice.

Factors / Option	1	2	3	4
Something-You-Know	☑	☑	☑	☑
Something-You-Have	☑			
• Software token		☑		
• Hardware token			☑	
• SMS OTP				☑
Something-You-Are	☑			

[1] Something-You-Know refers to information that the user knows, such as PIN (Personal Identification Number), password, or pattern, etc.

[2] Something-You-Have refers to the possession of a physical device, app, or token that generates authentication credentials, which may include time-based One-Time Passwords (OTPs). Examples of such tokens include software tokens, hardware tokens, and SMS OTP.

[3] Something-You-Are refers to biometric identifiers, where the user's unique physical characteristics are used for verification, such as fingerprints, retina scans, facial recognition, or voice recognition.

Developers are strongly advised not to rely on SMS and email OTP as a channel for authentication for high-risk transactions. If not able to, it is critical to implement a biometric factor or an additional authentication factor in conjunction with SMS OTP and email OTP.

Things to note

- It is strongly recommended to choose off-the-shelf solutions when possible.
- Developers should ensure that at least one MFA factor is verified on the client-side, with all others verified on the server-side. In cases where authentication is verified on the client side, especially for Android, enforce Trusted Execution Environment (TEE) based code.
- This security control is referenced in other standards. Please refer to the documentation(s) provided in:
    - OWASP Mobile Application Security Verification Standard (MASVS) v2.0.0 (2023), pg. 21.
    - OWASP Mobile Application Security Testing Guide (MASTG) v1.7.0 (2023), pg. 51, 56.
    - MAS Technology Risk Management Guidelines (2021), pg. 34, 50.
    - ENISA Smartphone Secure Development Guidelines (2016), pg. 11.

AUTHN-BP01a

Control

The app implements Something-You-Know authentication as one of the MFA factors.

Description

Something-You-Know represents a fundamental layer of identity verification involving information known only to the user, such as a PIN (Personal Identification Number), password, pattern, etc.

Implementing Something-You-Know as one of the MFA factors ensures a basic level of identity verification by requiring users to provide unique information associated with their accounts. It is a crucial factor in the principle of "Something-You-Know, Something-You-Have, and Something-You-Are," contributing to a comprehensive and effective multi-layered security strategy.

Implementation guidance

Developers should adopt the following guidelines for creating strong and secure passwords:

- Ensure a minimum password length of 12 characters or more.
- Include a mix of uppercase and lowercase letters, numbers, and special characters limited to ~`! @#$%^&*()_-+=:;,.?

Developers should also recognise and avoid common pitfalls in password creation:

- Avoid using guessable words, phrases, or combinations.
- Refrain from incorporating personal details.
- Steer clear of sequential characters (e.g., "123456") or repeated characters (e.g., "aaaaa").

Things to note

- Developers should enforce credential rotation only on organisational assets or if there is no MFA implementation on the user end, e.g., changed yearly or as per an appropriate timeframe.
- This security control is referenced in other standards. Please refer to the documentation(s) provided in:
    - MAS Technology Risk Management Guidelines (2021), pg. 34.
    - ENISA Smartphone Secure Development Guidelines (2016), pg. 10.

AUTHN-BP01b

Control

The app implements Something-You-Have authentication as one of the MFA factors.

Description

Something-You-Have requires users to authenticate with a physical device, app, or token that generates authentication credentials, which may include time-based One-Time Passwords (OTPs). Examples of such tokens include software tokens, hardware tokens, and SMS OTP.

Implementing Something-You-Have as one of the MFA factors adds complexity to the authentication process by requiring the possession of a tangible element, significantly reducing the likelihood of unauthorised access. It is a crucial factor in the principle of " Something-You-Know, Something-You-Have, and Something-You-Are," contributing to a comprehensive and effective multi-layered security strategy.

Implementation guidance

Developers should use a time-based OTP for software tokens, hardware tokens and SMS OTP. The following guidelines should be followed:

- An OTP should only be valid for no more than 30s.
- An OTP that is incorrectly inputted after 3 attempts should be invalidated and the user's session should be revoked or rejected.

Things to note

- This security control is referenced in other standards. Please refer to the documentation(s) provided in:
    - OWASP Mobile Application Security Testing Guide (MASTG) v1.7.0 (2023), pg. 56-57.
    - MAS Technology Risk Management Guidelines (2021), pg. 50, 51.
    - ENISA Smartphone Secure Development Guidelines (2016), pg. 10.

AUTHN-BP01c

Control

The app implements Something-You-Are authentication as one of the MFA factors.

Description

Something-You-Are requires users to authenticate with biometric identifiers such as fingerprints, retina scans, or facial recognition.

Implementing Something-You-Are as one of the MFA factors adds a highly personalised and difficult-to-replicate authentication factor. It provides a more robust means of verifying user identity than Something-You-Know and Something-You-Have factors, reducing the risk of unauthorised access. It is a crucial factor in the principle of " Something-You-Know, Something-You-Have, and Something-You-Are," contributing to a comprehensive and effective multi-layered security strategy.

Implementation guidance

Developers should implement server-side biometric authentication using a trusted biometric identification platform like Singpass.

However, if it is not feasible, developers should implement client-side biometric authentication through the device's Trusted Execution Environments (TEEs) mechanisms, such as CryptoObject and Android Protected Confirmation for Android or Keychain services for iOS.

Things to note

- Developers should limit apps' functionalities on devices lacking hardware Trusted Executed Environment (TEE) or biometrics. For example, Android devices lacking TEE can be detected using the "isInsideSecureHardware" Android API.

- Developers should invalidate biometric authentication if changes occur in the biometric mechanism, like enrolling a new fingerprint on the device. Both iOS and Android platforms support setting an app crypto key to expire in response to such changes.
- This security control is referenced in other standards. Please refer to the documentation(s) provided in:
    - OWASP Mobile Application Security Testing Guide (MASTG) v1.7.0 (2023), pg. 227-233, 422-426.
    - MAS Technology Risk Management Guidelines (2021), pg. 51.
    - ENISA Smartphone Secure Development Guidelines (2016), pg. 11, 26.

AUTHN-BP02

Control

The app uses context-based factors to authenticate.

Description

Context-based factors introduce dynamic elements such as user location and device attributes. While MFA provides a robust layer of security by requiring multiple authentication factors, incorporating context-based factors creates a more comprehensive and adaptive authentication process that can offer additional benefits in addressing the evolving risks of unauthorised access.

Implementing context-based factors minimises the reliance on static credentials, making it more challenging for malicious actors to attempt unauthorised access.

Implementation guidance

Developers should consider the following contextual factors to verify the identity of a user:

- **Geolocation**: Allow access based on the real-world location of a device using GPS, Wi-Fi, or IP address geolocation.
- **Device Type**: Allow access based on the characteristics of a device. e.g., screen size can determine if a device is a smartphone or tablet.

Things to note

This security control is referenced in other standards. Please refer to the documentation(s) provided in:

- OWASP Mobile Application Security Testing Guide (MASTG) v1.7.0 (2023), pg. 56, 58.
- ENISA Smartphone Secure Development Guidelines (2016), pg. 11.

AUTHN-BP03

Control

The app implements secure session authentication.

Description

Secure session authentication ensures robust session management for both stateful and stateless authentication. Poorly managed sessions, irrespective of whether the app follows stateful[4] or stateless[5] authentication methods, can lead to security threats such as unauthorised access, session hijacking, or data breaches.

Implementing secure session authentication for stateful sessions employs secure session identifiers, encrypted communication, and proper timeouts to prevent unauthorised access. For stateless authentication, it ensures that tokens are tamper-resistant, maintaining authentication integrity without relying on server-side storage.

Implementation guidance

Developers should implement secure session authentication by adopting the following best practices for stateful (AUTHN-BP03a) and stateless (AUTHN-BP03b) authentication methods for sessions.

Things to note

This security control is referenced in other standards. Please refer to the documentation(s) provided in:

- OWASP Mobile Application Security Testing Guide (MASTG) v1.7.0 (2023), pg. 51-55.
- MAS Technology Risk Management Guidelines (2021), pg. 51.
- ENISA Smartphone Secure Development Guidelines (2016), pg. 10.

[4] Stateful authentication refers to the management of session states on the server side, typically requiring the use of session identifiers.

[5] Stateless authentication refers to the management of sessions without storing user-related information on the server side.

AUTHN-BP03a

Control

The app implements secure stateful authentication.

Description

Secure stateful authentication involves protecting and maintaining persistent sessions. While stateful authentication provides a seamless user experience through persistent user sessions, it can be vulnerable to various security threats, such as malicious actors attempting to steal session identifiers.

Implementing secure stateful authentication protects user accounts from unauthorised access and potential vulnerabilities associated with session management without compromising the balance between usability and security.

Implementation guidance

Developers should identify server-side endpoints that expose sensitive information or critical functionalities.

Developers should also adopt the following stateful session authentication best practices:

- Reject requests with missing or invalid session IDs or tokens.
- Generate session IDs randomly on the server side without appending them to URLs.
- Enhance session ID security with proper length and entropy, making guessing difficult.
- Exchange session IDs only over secure HTTPS connections.
- Avoid storing session IDs in persistent storage.
- Verify session IDs for user access to privileged app elements.
- Terminate sessions on the server side, deleting session information upon timeout or logout.

Things to note

If in doubt, consider using trusted authentication platforms and protocols.

This security control is referenced in other standards. Please refer to the documentation(s) provided in:

- OWASP Mobile Application Security Testing Guide (MASTG) v1.7.0 (2023), pg. 52.

AUTHN-BP03b

Control

The app implements secure stateless authentication.

Description

Secure stateless authentication involves secure token practices for effective and scalable authentication. While stateless authentication provides benefits, it can be more vulnerable to security threats such as user impersonation if tokens are not securely generated, transmitted, and stored.

Implementing secure stateless authentication ensures that each authentication token is securely handled while reaping the benefits of efficiency and scalability, reducing the risk of unauthorised access.

Implementation guidance

Developers should adopt the following stateless session authentication best practices:

- Generate tokens on the server side without appending them to URLs.
- Enhance token security with proper length and entropy, making guessing difficult.
- Exchange tokens only over secure HTTPS connections.
- Verify that no sensitive data, such as PII, is embedded in tokens.
- Avoid storing tokens in persistent storage.
- Verify tokens for user access to privileged app elements.
- Terminate tokens on the server side, deleting token information upon timeout or logout.
- Cryptographically sign tokens using a secure algorithm, avoiding the use of null algorithms.

Things to note

- If in doubt, consider using trusted authentication platforms and protocols.
- This security control is referenced in other standards. Please refer to the documentation(s) provided in:
    - OWASP Mobile Application Security Testing Guide (MASTG) v1.7.0 (2023), pg. 52-53.

AUTHN-BP04

Control

The app implements secure session termination during logout, inactivity, or app closure.

Description

Secure session termination ensures the effective closure of user sessions. In scenarios such as logout, inactivity, or app closure scenarios, there is a potential for malicious actors to exploit any lingering access points if sessions are not appropriately managed.

Implementing secure session termination during logout, inactivity or app closure can significantly reduce the risk of unauthorised access by automatically terminating user sessions and safeguarding user information from being accessed by unauthorised parties.

Implementation guidance

Developers should reauthenticate users after logging out, app inactivity, idleness, backgrounding, absolute session timeouts, or abrupt/force closure.

Developers should also generate new session identifiers on the server whenever users move up to a new authentication level to prevent session fixation.

Things to note

- Developers should ensure that session termination includes clearing or deauthorising all locally stored tokens or session identifiers.
- Developers should determine the idle timeout value based on the risk and nature of financial services.
- This security control is referenced in other standards. Please refer to the documentation(s) provided in:
    - OWASP Mobile Application Security Testing Guide (MASTG) v1.7.0 (2023), pg. 55-56, 58.
    - MAS Technology Risk Management Guidelines (2021), pg. 51.
    - ENISA Smartphone Secure Development Guidelines (2016), pg. 11.

AUTHN-BP05

Control

The app implements brute force protection for authentication.

Description

Brute force attacks involve automated and systematic attempts to guess user credentials, for example, by trying various combinations of usernames and passwords to gain unauthorised access.

Brute force protection restricts the number of login attempts within a specified period. Implementing brute force protection for authentication can significantly mitigate the risk of unauthorised access, protect user accounts, and maintain the integrity of the authentication process.

Implementation guidance

Developers should implement brute force mechanisms through the following best practices:

- Implement anti-automation checks.
- Apply rate limiting for login attempts.
- Incorporate progressive incremental time delays (e.g., 30 seconds, 1 minute, 2 minutes, 5 minutes) for login attempts.
- Enforce account lockouts.

Things to note

- Developers should note that all MFA mechanisms are vulnerable to brute force.
- Developers should convey the reasons for the account lockout and provide accessible means for users to authenticate themselves and remove the lockout. Examples include calling a helpline or utilising biometric verification.
- This security control is referenced in other standards. Please refer to the documentation(s) provided in:
    - ENISA Smartphone Secure Development Guidelines (2016), pg. 10, 16.

AUTHN-BP06

Control

The app implements transaction integrity verification mechanism.

Description

While authentication ensures the user's identity, it does not eliminate the possibility of fraudulent activities during the transaction process.

Transaction integrity verification mechanisms are auxiliary security functions that give users the time and tools to react to potential frauds. Implementing a transaction integrity verification mechanism ensures that each transaction undergoes thorough scrutiny to confirm its accuracy and authenticity.

Implementation guidance

Developers can implement the following suggested best practices:

- Initiate a transaction verification/confirmation call.
- Provide a real-time transaction history.
- Implement a cooldown period of 12 hours to 24 hours.
- Disable overseas transactions by default; enable only through MFA.

Things to note

This security control is referenced in other standards. Please refer to the documentation(s) provided in:

- OWASP Mobile Application Security Testing Guide (MASTG) v1.7.0 (2023), pg. 57-58.

02 AUTHORISATION

2. Authorisation

Introduction

Authorisation security operates in conjunction with authentication security. Authorisation security in mobile apps is a crucial line of defence as it delineates who can access what resources within an app. It creates systematic controls and validates user access rights within an app.

Developers can ensure that only authorised users, clients, apps, and devices can access specific resources or perform certain actions by implementing strong authorisation controls and authorisation setups. Through authorisation controls, developers can also mitigate the risk of unauthorised data access, maintain the integrity of sensitive data, uphold user privacy, and protect the integrity of high-risk transaction functionalities. Although the enforcement of these mechanisms must be on the remote endpoint, it is equally important for the client-side app to follow relevant best practices to ensure the secure use of the involved authorisation protocols.

The controls in this category provide authorisation security controls that the app should implement to safeguard sensitive data and prevent unauthorised access. It also gives developers relevant best practices for implementing these security controls.

Security controls

ID	Control
AUTHOR-BP01	Implement server-side authorisation.
AUTHOR-BP02	Implement client-side authorisation via device binding.
AUTHOR-BP03	Notify users of all required permissions before they start using the app.
AUTHOR-BP04	Notify users of all high-risk transactions that have been authorised and completed.

AUTHOR-BP01

Control

The app implements server-side authorisation.

Description

Server-side authorisation refers to validating and granting access permissions to users or apps by a server or an authorisation server. This ensures that access control decisions and permissions are managed and enforced on the server-side rather than the client.

By implementing server-side authorisation, developers reduce opportunities for malicious attackers to tamper or bypass security measures on the app to gain unauthorised access to sensitive data (i.e., PIIs and Authentication data).

Implementation guidance

Developers should implement server-side authorisation after successful authentication before granting access permissions.

Developers should ensure that users are granted access based on the following:

- **Assigned role with permissions**: Ensure that users can only perform tasks relevant to their responsibilities.
- **Contextual factors**: Dynamic access scenarios such as Time of Access and Behavioural Analysis.

Things to note

This security control is referenced in other standards. Please refer to the documentation(s) provided in:

- OWASP Mobile Application Security Testing Guide (MASTG) v1.7.0 (2023), pg. 50-55, 58.
- PCI Mobile Payment Acceptance Security Guidelines v2.0.0 (2017), pg. 10.
- ENISA Smartphone Secure Development Guidelines (2016), pg. 10-11.

AUTHOR-BP02

Control

The app implements client-side authorisation via device binding.

Description

Client-side authorisation is the process of managing access permissions within a mobile app. This is risky as relying on the client-side can expose apps to vulnerabilities such as unauthorised access and potential fraud.

If an app’s business functions (e.g., instantiating software tokens) require client-side authorisations, device binding (a security practice that associates authorisations to access privileges on a particular device) is recommended. By implementing device binding, apps can verify device identity and establish trust. This reduces the risks associated with unauthorised access and maintains a secure, trusted path between devices, apps, and servers.

Implementation guidance

Developers should establish binding between apps and the device when a user's identity is used for the first time on an unregistered mobile device.

Developers should also verify that apps:

- Check for modifications to the device since the last runtime.
- Check for modifications to the device identity markers.
- Check that the device running the app is in a secure state (e.g., no jailbreaking or rooting).

The above are just some examples of best-practice techniques used by the industry. As the ecosystem of mobile devices evolves, these techniques may become out of date. As such, developers should keep abreast of the latest industry best practices to verify device bindings.

Things to note

To verify device binding on Android devices, developers can:

- Obtain unique identifiers like IMEI or Android ID.
- Retrieve build information.
- Leverage native OS API features, such as Google's SafetyNet.

To verify device binding on iOS devices, developers can:

- Leverage native OS services, such as Apple's device ID, via UIDevice.

This security control is referenced in other standards. Please refer to the documentation(s) provided in:

- OWASP Mobile Application Security Testing Guide (MASTG) v1.7.0 (2023), pg. 316-317, 516.
- MAS Technology Risk Management Guidelines (2021), pg. 51, 56.

AUTHOR-BP03

Control

The app notifies users of all required permissions before they start using the app.

Description

Required permissions are specific rights and capabilities that the app requests from the mobile device. These permissions define what resources or functionalities the app can access on users' devices. Some examples include, but are not limited to, camera, microphone, location, etc.

By implementing proper notifications that inform the users of what permissions are being requested, developers can prevent users from unknowingly granting excessive permissions, which may allow malicious actors to exploit vulnerabilities and steal sensitive data (i.e., PIIs and Authentication Data). Such notifications will also allow users to make informed decisions about the apps they install.

Implementation guidance

Developers should use In-App (In-App) alerts to request users for access permission. Developers should also ensure that Notifications/Alerts do not display sensitive data.

Things to note

Developers should only request essential permissions for the app's functionality.

This security control is referenced in other standards. Please refer to the documentation(s) provided in:

- OWASP Mobile Application Security Testing Guide (MASTG) v1.7.0 (2023), pg. 56, 58.
- ENISA Smartphone Secure Development Guidelines (2016), pg. 8, 18, 28.
- Apple Developer Guide on Privacy, https://developer.apple.com/design/human-interface-guidelines/privacy (Jan 2024).
- Android Developer Guide on Privacy, https://developer.android.com/quality/privacy-and-security (Jan 2024).

AUTHOR-BP04

Control

The app notifies users of all high-risk transactions that have been authorised and completed.

Description

If an app has high-risk transaction functionalities, users should be notified immediately when a transaction has been authorised and completed.

By implementing this control, developers can ensure that users are made aware immediately when high-risk transactions have been authorised and completed so that they may be able to identify potential fraudulent transactions as soon as possible.

Implementation guidance

Developers should adopt the following methods to alert users:

- In-Application (In-App) alerts.
- Email notifications.
- Short Message Service (SMS) notifications.

Developers should also ensure that Notifications/Alerts do not display sensitive data.

The above are just some examples of best-practice notification techniques used by the industry. As the ecosystem of mobile devices evolves, these techniques may become out of date. As such, developers should keep abreast of the latest industry best practices to notify users of high-risk transactions that are authorised and completed.

Things to note

Developers should request only essential permissions for the app's functionality.

This security control is referenced in other standards. Please refer to the documentation(s) provided in:

- MAS Technology Risk Management Guidelines (2021), pg. 52.
- PCI Mobile Payment Acceptance Security Guidelines v2.0.0 (2017), pg. 10.
- ENISA Smartphone Secure Development Guidelines (2016), pg. 8.
- Apple Developer Guide on Privacy, https://developer.apple.com/design/human-interface-guidelines/privacy (Jan 2024).
- Android Developer Guide on Privacy, https://developer.android.com/quality/privacy-and-security (Jan 2024).

03 DATA STORAGE (DATA-AT-REST)

3. Data Storage (Data-at-Rest)

Introduction

Data Storage security for data-at-rest pertains to safeguarding the integrity and confidentiality of sensitive data (i.e., PIIs and Authentication data) stored locally on the client-side device and app server-side when it is not actively being used or transmitted. This encompasses the best practices, protective measures and encryption techniques employed to secure data stored in databases, files, caches, memory, and Trusted Execution Environment (TEE) on mobile devices and similar areas in app servers.

Developers can ensure that user data is preserved and protected by implementing strong security controls for storing data at rest. Proper data-at-rest controls also ensure that the app can mitigate the risks of unauthorised access, device compromise, potential data breaches, and data leaks and fortify the app defences.

The following controls ensure that any sensitive data intentionally stored by the app is adequately protected, regardless of the target location. It also covers unintentional leaks due to improper use of APIs or system capabilities.

Security controls

ID	Control
STORAGE-BP01	Store sensitive data that is only necessary for transactions.
STORAGE-BP02	Implement secure storage of sensitive data.
STORAGE-BP02a	Store sensitive data securely on server-side.
STORAGE-BP02b	Store sensitive data securely on client-side in a Trusted Execution Environment (TEE).
STORAGE-BP03	Delete sensitive data when no longer necessary.

STORAGE-BP01

Control

The app stores sensitive data that is only necessary for transactions.

Description

Sensitive data is defined as user data (PIIs) and authentication data (e.g., credentials, encryption keys, etc.) Developers should only store sensitive data that is necessary for app business functions. Accumulating unnecessary information increases the impact of potential security breaches, making an app an attractive target for malicious actors.

By implementing this security control, developers can ensure that exposure is limited to the data required for specific business functions, minimising the impact in the event of unauthorised access or data breaches.

Implementation guidance

Developers should classify data being used by the app based on an organisation's sensitivity levels and based on legal law requirements.

Developers should adopt the following guidelines to secure data that are classified as sensitive:

1. Implement a secure storage solution based on its sensitivity on the client-side/server-side.
2. Apply data protection measures (e.g., tokenising, hashing with salt, encrypting)
3. Delete sensitive data when no longer necessary.

Things to note

This security control is referenced in other standards. Please refer to the documentation(s) provided in:

- OWASP Mobile Application Security Testing Guide (MASTG) v1.7.0 (2023), pg. 190, 398.
- MAS Technology Risk Management Guidelines (2021), pg. 9-10, 36, 38.
- ENISA Smartphone Secure Development Guidelines (2016), pg. 6.

STORAGE-BP02

Control

The app implements secure storage of sensitive data.

Description

Secure storage for mobile apps refers to implementing techniques and practices to protect sensitive data stored on mobile devices and app servers from unauthorised access, theft, or tampering. This involves best practices such as encryption, hashing, tokenisation, and proper access controls.

By implementing secure storage, developers can mitigate against unauthorised access, device compromise, potential data breaches and data leaks.

Implementation guidance

Developers should implement a secure storage solution that commensurate with the sensitivity of data.

Developers should also prioritise the following order for secure storage solutions (from the most sensitive data to the least sensitive data):

1. Server-side (all sensitive data should be stored on the server-side).
2. Client-side within the Trusted Execution Environment (in the case where server-side is not possible, store all sensitive data in the client-side TEE).

Things to note

This security control is referenced in other standards. Please refer to the documentation(s) provided in:

- OWASP Mobile Application Security Verification Standard (MASVS) v2.0.0 (2023), pg. 17-18.
- OWASP Mobile Application Security Testing Guide (MASTG) v1.7.0 (2023), pg. 190-203, 398-406.
- ENISA Smartphone Secure Development Guidelines (2016), pg. 06-07.

STORAGE-BP02a

Control

The app stores sensitive data securely on server-side.

Description

Storing sensitive data on the server-side refers to storing data on remote app servers or databases. Such an approach creates a better environment to protect data from unauthorised access or breaches, enabling more secured access control and options to implement better security measures such as more complex encryptions and provisions of quicker security updates.

By implementing server-side storage of sensitive data, developers can mitigate against the inherent risks of client-side data storage, as client-side storage is inherently more susceptible to data storage exploitation techniques commonly used by malicious actors in mobile scams.

Implementation guidance

Developers should apply at least 1 of the following data protection measures:

1. For passwords only, developers can use hashing with salt[6]. Instead of storing actual passwords, unique salts are generated and combined with passwords, creating salted hashes.
2. Developers can encrypt[7] sensitive data with encryption standards such as AES-128.
3. Developers can implement tokenisation[8] with self-managed tokenisation or a tokenisation service, replacing sensitive data with tokens where possible. In addition, developers should ensure tokenisation is of sufficient length and complexity (backed by secure cryptography) based on the data sensitivity and business needs.

The above are just some examples of best practices used by the industry. As the ecosystem of mobile devices evolves, these best practices may become out of date. As such, developers should abreast of the latest industry best practices to store sensitive data securely on the server-side.

Things to note

This security control is referenced in other standards. Please refer to the documentation(s) provided in:

- OWASP Mobile Application Security Verification Standard (MASVS) v2.0.0 (2023), pg. 19-20.
- OWASP Mobile Application Security Testing Guide (MASTG) v1.7.0 (2023), pg. 71-77, 219-227, 416-421.
- MAS Technology Risk Management Guidelines (2021), pg. 30, 36-37, 39.
- PCI Mobile Payment Acceptance Security Guidelines v2.0.0 (2017), pg. 9.
- ENISA Smartphone Secure Development Guidelines (2016), pg. 6-9.

[6] Hashing with salt is used to add an extra layer of security by making it computationally intensive for attackers to decipher original sensitive data. In the context of password storage or key derivation, developers should utilise one-way key derivation functions or slow hash algorithms, such as PBKDF2, bcrypt, or scrypt.
[7] Encryption is used to transform data into an unreadable format, ensuring that even if accessed without authorisation, sensitive data remains confidential.
[8] Tokenisation is used to substitute sensitive data with tokens to minimise the risk of sensitive data exposure.

STORAGE-BP02b

Control

The app stores sensitive data securely on client-side in a Trusted Execution Environment (TEE).

Description

The Trusted Execution Environment (TEE) is an isolated area within a mobile device's hardware or processor architecture that provides a highly secure environment for storing sensitive data and executing sensitive or critical operations. It is designed to protect sensitive data, cryptographic keys and critical processes from unauthorised access or tampering. If an app's business functions require storage of sensitive data on the client-side, it is recommended to store it in the device's TEE.

By implementing proper storage of sensitive data in the client-side TEE, developers can mitigate against threats originating from within a compromised device and from external malicious actors. Such storage can also mitigate against unauthorised access to user's sensitive data on an app and prevent any encryption keys from being stolen.

Implementation guidance

Developers should store sensitive data securely on client-side in a Trusted Execution Environment (TEE) such as Android's ARM's TrustZone, Apple's Secure Enclave.

Developers should also store minimally the following list of sensitive data in a TEE:

- Biometric identifiers.
- Authentication tokens.
- Cryptographic keys in a secure key management system such as Android Keystore, or iOS Keychain.

The above are just some examples of what sensitive data developers should store in the TEE. As the ecosystem of mobile devices evolves, developers should exercise the freedom to store any data they deem necessary to be stored in the TEE.

Things to note

For devices without hardware TEEs, developers may consider the usage of virtualised TEEs.

Alternatively, developers can consider disabling the app or disabling high-risk transaction functions of the app, as the app is deemed insecure for high-risk transactions.

This security control is referenced in other standards. Please refer to the documentation(s) provided in:

- OWASP Mobile Application Security Verification Standard (MASVS) v2.0.0 (2023), pg. 19-20.
- OWASP Mobile Application Security Testing Guide (MASTG) v1.7.0 (2023), pg. 75, 93, 194-200.
- MAS Technology Risk Management Guidelines (2021), pg. 51.
- PCI Mobile Payment Acceptance Security Guidelines v2.0.0 (2017), pg. 07-09, 14.
- ENISA Smartphone Secure Development Guidelines (2016), pg. 10.

STORAGE-BP03

Control

The app deletes sensitive data when no longer necessary.

Description

Deleting sensitive data refers to the process of permanently removing or erasing confidential, private or sensitive data from storage devices, servers or databases. This process ensures that sensitive data is irrecoverably removed and cannot be accessed, retrieved, accidentally exposed, or reconstructed by unauthorised individuals or through data recovery methods.

By implementing this process, developers can minimise the window in which attackers can exploit vulnerabilities to steal sensitive data.

Implementation guidance

Developers should use the following persistent storage security techniques:

- Clear stored cookies on app termination or use in-memory cookie storage.
- Remove all sensitive data on app uninstallation.
- Manually remove all database files that contain sensitive data (e.g., iOS WebView caches) from the file system when related business functions cease to exist.

The above are just some examples of best practices used by the industry. As the ecosystem of mobile devices evolves, these techniques may become out of date. As such, developers should be abreast of the latest industry best practices to delete sensitive data when no longer necessary.

Things to note

Developers should be mindful of adhering to widely accepted standards and relevant data retention law, including but not limited to:

- The Personal Data Protection Act (PDPA)
- The General Data Protection Regulation (GDPR)
- The Payment Card Industry Data Security Standard (PCI DSS)

This security control is referenced in other standards. Please refer to the documentation(s) provided in:

- OWASP Mobile Application Security Testing Guide (MASTG) v1.7.0 (2023), pg. 199, 206-214, 403-414.
- MAS Technology Risk Management Guidelines (2021), pg. 39.
- ENISA Smartphone Secure Development Guidelines (2016), pg. 07, 09-10.

04
ANTI-TAMPERING AND ANTI-REVERSING

4. Anti-Tampering & Anti-Reversing

Introduction

Anti-Tampering and Anti-Reversing security controls are additional measures that developers can implement to counteract attacks attempting to tamper or reverse engineer apps. By implementing both features, developers add multiple layers of defences to apps, making it more difficult for malicious actors to successfully tamper or reverse engineer apps, which could result in:

- The theft or compromise of valuable business assets such as proprietary algorithms, trade secrets, or sensitive data,
- Financial losses of users who utilise the app for high-risk transactions,
- Financial losses of organisations due to loss of revenue or legal action,
- Damage to brand reputation due to negative publicity or customer dissatisfaction

The controls ensure that apps run on trusted platforms, prevent tampering at runtime and ensure the integrity of the apps' functionalities. In addition, the controls impede comprehension by making it difficult for attackers to figure out how the apps operate.

Security controls

ID	Control
RESILIENCE-BP01	Sign with certificates from official app stores.
RESILIENCE-BP02	Implement jailbreak/root detection.
RESILIENCE-BP03	Implement emulator detection.
RESILIENCE-BP04	Implement anti-malware detection.
RESILIENCE-BP05	Implement anti-hooking mechanisms.
RESILIENCE-BP06	Implement overlay, remote viewing, and screenshot countermeasures.
RESILIENCE-BP07	Implement anti-keystroke capturing or anti-keylogger against third-party virtual keyboards.

RESILIENCE-BP01

Control

The app is code signed with certificates from official app stores.

Description

Apps are often spoofed by malicious actors and distributed via less strictly regulated channels. Signing an app with certificates provided by official app stores assures the mobile OS and users that the mobile app is from a verified source.

Implementing code signing helps operating systems determine whether to allow software to run or install based on the signatures or certificates used to sign the code. This helps prevent the installation and execution of potentially harmful apps. In addition, code signing also assists with integrity verification, as signatures will change if the app has been tampered with.

Implementation guidance

Developers should code-sign their apps with certificates. This section provides examples of how to do this via the two most popular platforms, iOS, and Android.

For Apple's App Store, it can be done by enrolling in the Apple Developer Programme and creating a certificate signing request in the developer portal. Developers can register for the Apple Developer Programme and can reference the following developer guide for code signing under things to note.

For Android, there are a variety of App stores. For Google's Play Store, it can be done by configuring Play App Signing, which is a requirement for distribution through Google's Play Store, for more information on how to do so developers can visit the Android developer guide under things to note.

For other official stores, refer to their respective developer guidelines on app source code signing.

Things to note

This security control is also a requirement for publishing apps on official app stores; as such, the recommendation is for your app to be code-signed with certificates from official app stores.

This security control is referenced in other standards. Please refer to the documentation(s) provided in:

- Apple Developer Programme Guide for Code Signing, https://developer.apple.com/support/code-signing (Jan 2024).
- Android Developer Guide on Privacy, https://developer.android.com/quality/privacy-and-security (Jan 2024).
- OWASP Mobile Application Security Testing Guide (MASTG) v1.7.0 (2023), pg. 325-326, 522-523.
- ENISA Smartphone Secure Development Guidelines (2016), pg. 21.

RESILIENCE-BP02

Control

The app implements jailbreak or root detection.

Description

Rooted and jailbroken devices are generally considered insecure. Rooted or jailbroken devices allow users to gain elevated privileges, enabling easier circumvention of security and OS limitations. Such elevated privileges can be unsafe for apps as these privileges allow malicious actors to potentially exploit vulnerabilities, steal credentials, take over user devices and execute fraudulent app transactions.

By implementing jailbreak or root detection, developers can prevent the abovementioned exploits from happening, protect apps' intellectual property, ensure the stability of apps, and prevent the bypass of in-app systems.

Implementation guidance

Developers should implement jailbreak or root detection by implementing the following checks in their app for Android devices:

1. Check for superuser or SU binary.
2. Detect root file system changes.
3. Look for rooted apps.
4. Check for custom recovery.
5. Check for unsafe API usage.

Developers should implement jailbreak or root detection by implementing the following checks in their app for iOS devices:

1. Detect the use of restricted APIs.
2. Look for jailbreak tweaks like mods.
3. Look for unofficial app stores, e.g., check for Cydia App Store signature.
4. Look for kernel modifications.
5. Check for the integrity of the critical file systems.
6. Use 3rd-party libraries designed to detect device tampering.

The above are just some examples of best-practice checks used by the industry. As the ecosystem of mobile devices evolves, these checks may become out of date. As such, developers should be abreast of the latest industry best practices to implement jailbreak or root detection.

Things to note

This security control is referenced in other standards. Please refer to the documentation(s) provided in:

- OWASP Mobile Application Security Verification Standard (MASVS) v2.0.0 (2023), pg. 31.
- OWASP Mobile Application Security Testing Guide (MASTG) v1.7.0 (2023), pg. 319-320, 506[9], 518-519.
- MAS Technology Risk Management Guidelines (2021), pg. 50.
- ENISA Smartphone Secure Development Guidelines (2016), pg. 11, 23.

[9] https://github.com/crazykid95/Backup-Mobile-Security-Report/blob/master/Jailbreak-Root-Detection-Evasion-Study-on-iOS-and-Android.pdf

RESILIENCE-BP03

Control

The app implements emulator detection.

Description

Emulators are software used to test mobile apps by allowing a user to test a mobile app on a variety of mimicked mobile versions and devices. Although useful for testing, apps should not allow themselves to be mounted on emulators outside of the development environment.

By implementing emulation detection, developers can prevent malicious actors from running dynamic analysis, rooting, debugging, instrumentation, hooking, and fuzz testing on an emulated device they can control. In doing so, developers can prevent malicious actors from discovering vulnerabilities within the app for exploitation.

Implementation guidance

Developers should implement the following detection strategy to identify features for commonly used emulation solutions. Some recommendations of things to check for are:

- Check battery usage.
- Check timestamps and clocks.
- Check multi-touch behaviours.
- Check memory and performance analysis.
- Perform network checks.
- Check whether it is hardware-based.
- Check what the OS is based on.
- Check for device fingerprints.
- Check build configurations.
- Check for emulator services and apps.

The above are just some examples of best-practice checks used by the industry. As the ecosystem of mobile devices evolves, these checks may become out of date. As such, developers should be abreast of the latest industry best practices to implement emulator detection.

Things to note

This security control is referenced in other standards. Please refer to the documentation(s) provided in:

- OWASP Mobile Application Security Verification Standard (MASVS) v2.0.0 (2023), pg. 31-32.
- OWASP Mobile Application Security Testing Guide (MASTG) v1.7.0 (2023), pg. 325, 521.

RESILIENCE-BP04

Control

The app implements anti-malware detection.

Description

Malware apps are increasingly used by malicious actors as a vector to compromise users' mobile devices as such devices provide users with the convenience needed to perform day-to-day transactions. Malware apps primarily utilise sideloading features as a channel to get users to install malware on their devices.

By implementing anti-malware detection capabilities on an app at runtime, developers can prevent users from being exploited via malware exploiting app vulnerabilities and OS vulnerabilities, stealing credentials, taking over the device, and executing fraudulent transactions.

Implementation guidance

Developers should implement anti-malware detection capabilities in their apps. This can be done in a variety of ways, but are not limited to:

- Incorporate a Runtime-Application-Self-Protection (RASP) Software Development Kit (SDK) into their apps.
- Utilise RASP SDKs to check for and detect malware apps at runtime.
- Check for and prevent overlays.
- Prevent clickjacking.
- Prevent app memory hooking.

The above are just some examples of best-practice checks used by the industry. As the ecosystem of mobile devices evolves, these checks may become out of date. As such, developers should be abreast of the latest industry best practices to implement anti-malware detection.

Things to note

If any form of maliciousness is detected, developers should disable the app, provide the user with the necessary information on why the app has been disabled, and urge the user to uninstall the malicious app(s) on their device.

Alternatively, developers should warn the user and disable high-risk functions on the app until the user remediates the malicious app(s).

This security control is referenced in other standards. Please refer to the documentation(s) provided in:

- OWASP Mobile Application Security Verification Standard (MASVS) v2.0.0 (2023), pg. 31.
- MAS Technology Risk Management Guidelines (2021), pg. 40, 49.
- ENISA Smartphone Secure Development Guidelines (2016), pg. 23.

RESILIENCE-BP05

Control

The app implements anti-hooking mechanisms.

Description

Hooking refers to a technique used by attackers to intercept or modify the behaviour of a mobile app at runtime. This involves inserting or hooking into the execution flow of an app to either monitor its activities, alter its behaviour, inject malicious code, or modify existing code functions to exploit vulnerabilities.

By implementing anti-hooking mechanisms on apps, developers can prevent the above attacks from happening and prevent unauthorised access, protect high-risk transaction operations, detect, and prevent tampering and modification attempts, preserve intellectual property, and maintain app reliability.

Implementation guidance

Developers should implement the following example mechanisms to mitigate against hooking attacks:

- Implement protections to block code injections.
- Implement protections to prevent method hooking by preventing modifications to the app source code (both on the client and server).
- Implement protections to prevent the execution of modified codes in your app.
- Implement protections to prevent memory access and memory manipulation of your app.
- Implement tamper resistant algorithms or anti-tampering SDKs (commonly known as Runtime-Application-Self-Protection SDKs).
- Check for insecure parameters such as obsolete APIs and parameters.

The above are just some examples of best-practice checks used by the industry. As the ecosystem of mobile devices evolves, these checks may become out of date. As such, developers should be abreast of the latest industry best practices to implement anti-hooking mechanisms.

Things to note

This security control is referenced in other standards. Please refer to the documentation(s) provided in:

- OWASP Mobile Application Security Verification Standard (MASVS) v2.0.0 (2023), pg. 31.
- OWASP Mobile Application Security Testing Guide (MASTG) v1.7.0 (2023), pg. 135-140, 189, 318-319, 339-340, 390, 520.
- MAS Technology Risk Management Guidelines (2021), pg. 56.
- ENISA Smartphone Secure Development Guidelines (2016), pg. 23, 26.

RESILIENCE-BP06

Control

The app implements overlay, remote viewing, and screenshot countermeasures.

Description

Sensitive information can be captured or recorded without the user's explicit consent when an app has screen recording, screenshot or overlay functionalities. For example:

- Overlay attacks deceive users by creating a fake layer mimicking trusted apps, aiming to steal sensitive data.
- Remote viewing attacks involve unauthorised access to a device's screen, allowing attackers to harvest sensitive data remotely.
- Screenshot attacks occur when malicious actors capture a device's screen without user consent, extracting sensitive data.

Implementing overlay, remote viewing, and screenshot countermeasures can ensure that sensitive information remains secure, user privacy is upheld, and sensitive data is protected against inadvertent loss or misuse.

Implementation guidance

Developers should implement anti-tampering and anti-malware checks via RASP SDKs to prevent malicious apps from utilising overlays and remote viewing exploits.

For screenshots, developers can utilise the FLAG_SECURE flag for Android apps and similar flags for iOS to block out all screenshot capabilities when using the app. However, suppose business functions require screenshot capabilities (e.g., Taking a screenshot of a completed PayNow transaction). In that case, the recommendation is to disable screenshot capabilities for screens or pages that include sensitive data (PII and Authentication Data).

Developers can also consider masking input with sensitive data and sensor screens when the app is backgrounded.

Things to note

Some examples of where to disable these screenshot capabilities include but are not limited to: Login pages, Multi-Factor Authentication pages, Security Credentials, and PII changing pages, etc.

This security control is referenced in other standards. Please refer to the documentation(s) provided in:

- OWASP Mobile Application Security Verification Standard (MASVS) v2.0.0 (2023), pg. 31.
- OWASP Mobile Application Security Testing Guide (MASTG) v1.7.0 (2023), pg. 166-168, 257, 259, 265-267, 366, 480-481.
- MAS Technology Risk Management Guidelines (2021), pg. 56.
- ENISA Smartphone Secure Development Guidelines (2016), pg. 8.

RESILIENCE-BP07

Control

The app implements anti-keystroke capturing or anti-keylogger against third-party virtual keyboards.

Description

Keystroke capturing and keylogging are methods malicious actors utilise to monitor, log, and record the keys pressed on a keyboard without the user's knowledge and consent. This allows logging and capturing of potentially sensitive data (i.e., PII and Authentication Data).

By implementing keystroke and keylogging countermeasures, developers can prevent the unnecessary loss of sensitive data. More specifically, this control is targeting Android devices, as the native keyboard of Android devices can be changed. Such changes can expose apps to security vulnerabilities as the trusted path between keyboard inputs and apps has untrusted parties between them.

Implementation guidance

Developers should not allow insecure third-party virtual keyboards to be used for inputs that may contain sensitive data. A secure in-app custom keyboard is preferred for such inputs.

By implementing an in-app keyboard, developers can control where the logging data goes and mitigate against the risk of insecure third-party virtual keyboards acting as keyloggers to capture keystrokes.

Along with using in-app keyboards, developers should implement the following suggestions for inputs that require sensitive data (i.e., PII and Authentication Data): Disable autocorrect, autofill, autosuggestion, cut, copy, and paste for functions/or apps that contain sensitive data.

Things to note

Some examples of functions that should utilize in-app keyboards include but are not limited to logging in, entering an OTP, or other verification factors, etc.

This security control and best practice primarily targets Android devices.

The main goal is to ensure the security of the trusted path. Since Android does not provide a method by which to enforce the usage of native/trusted keyboards, developers should implement an in-app keyboard to ensure insecure third-party virtual keyboards do not log information.

Implementing a secure in-app keyboard does not mitigate the risks associated with a compromised device.

This security control is referenced in other standards. Please refer to the documentation(s) provided in:

- OWASP Mobile Application Security Verification Standard (MASVS) v2.0.0 (2023), pg. 31.
- OWASP Mobile Application Security Testing Guide (MASTG) v1.7.0 (2023), pg. 203, 214-215, 257, 259, 400, 414-415.
- MAS Technology Risk Management Guidelines (2021), pg. 56.
- ENISA Smartphone Secure Development Guidelines (2016), pg. 08, 23.

References

S/N	Document	Source	Dated
1	OWASP Mobile Application Security Verification Standard (MASVS) v2.0.0	OWASP	2023
2	OWASP Mobile Application Security Testing Guide (MASTG) v1.7.0	OWASP	2023
3	MAS Technology Risk Management Guidelines,	MAS	2021
4	PCI Mobile Payment Acceptance Security Guidelines v2.0.0	PCI-DSS	2017
5	ENISA Smartphone Secure Development Guidelines	ENISA	2016
6	Android Developers	Android	2024
7	Apple Developer Documentation	Apple	2024