OWASSP TOP 10 MOBILE VULNERABLITIES

AKHIL S PRAMOD

intern@cybersapiens

OWASP stands for Open Web Application Security Project

Is a globally recognised organisation dedicated to improve software security

Top 10 Mobile Risks - Final release 2024

Improper Credential Usage

Threat agents exploiting hardcoded credentials and improper credential usage in mobile applications can include automated attacks using publicly available or custom-built tools. Such agents could potentially locate and exploit hardcoded credentials or exploit weaknesses due to improper credential usage.

Exploitability EASY

Adversaries can exploit vulnerabilities in both hardcoded credentials and improper credential usage. Once these vulnerabilities are identified, an attacker can use hardcoded credentials to gain unauthorized access to sensitive functionalities of the mobile app. They can also misuse credentials, for instance by gaining access through improperly validated or stored credentials, thereby bypassing the need for legitimate access.

Security Weakness

Poor implementation of credential management, such as using hardcoded credentials and improper handling, can lead to severe security weaknesses. A comprehensive security testing process should aim to identify these issues. For instance, security testers should attempt to identify hardcoded credentials within the mobile app's source code or within any configuration files.

Inadequate Supply Chain Security

This is more about exploiting known vulnerabilities and doing things to access the backend server of an application in order to inject malware .

An attacker can manipulate application functionality by exploiting vulnerabilities in the mobile app supply chain. For example, an attacker can insert malicious code into the mobile app's codebase or modify the code during the build process to introduce backdoors, spyware, or other malicious code.

This can allow the attacker to steal data, spy on users, or take control of the mobile device. Moreover, an attacker can exploit vulnerabilities in third-party software libraries, SDKs, vendors, or hardcoded credentials to gain access to the mobile app or the backend servers.

This can lead to unauthorized data access or manipulation, denial of service, or complete takeover of the mobile app or device.

Exploitability AVERAGE

There are multiple ways to exploit Inadequate Supply Chain vulnerability for example- an insider threat agent or an attacker can inject malicious code during the development phase of the app, then they can compromise the app signing keys or certificates to sign malicious code as trusted.

Another way, a threat agent can exploit vulnerabilities in third-party libraries or components used in the app.

Am I vulnerable to 'Inadequate Supply Chain Vulnerability'?

It is possible that you are vulnerable to inadequate supply chain vulnerability, particularly if you use mobile applications that are developed by third-party developers or rely on third-party libraries and components. The vulnerability can arise due to a variety of reasons, such as:

Lack of Security in Third-Party Components: Third-party components, such as libraries or frameworks, can contain vulnerabilities that can be exploited by attackers. If the mobile application developer does not vet the third-party components properly or keep them updated, the application can be vulnerable to attacks.

Malicious Insider Threats: Malicious insiders, such as a rogue developer or a supplier, can introduce vulnerabilities into the mobile application intentionally. This can occur if the developer does not implement adequate security controls and monitoring of the supply chain process.

Inadequate Testing and Validation: If the mobile application developer does not test the application thoroughly, it can be vulnerable to attacks. The developer may also fail to validate the security of the supply chain process, leading to vulnerabilities in the application.

Lack of Security Awareness: If the mobile application developer does not have adequate security awareness, they may not implement the necessary security controls to prevent supply chain attacks.

How Do I Prevent Inadequate Supply Chain Vulnerability'?

- Implement secure coding practices, code review, and testing throughout the mobile app development lifecycle to identify and mitigate vulnerabilities.
- Ensure secure app signing and distribution processes to prevent attackers from signing and distributing malicious code.
- Use only trusted and validated third-party libraries or components to reduce the risk of vulnerabilities.
- Establish security controls for app updates, patches, and releases to prevent attackers from exploiting vulnerabilities in the app.
- Monitor and detect supply chain security incidents through security testing, scanning, or other techniques to detect and respond to incidents in a timely manner

Insecure Authentication/Authorization

Threat agents that exploit authentication and authorization vulnerabilities typically do so through automated attacks that use available or custom-built tools.

Exploitability EASY

Once the adversary understands the vulnerabilities in either the authentication or authorization scheme, they can exploit these weaknesses in one of two ways. They may either fake or bypass the authentication by directly submitting service requests to the mobile app's backend server, circumventing any direct interaction with the mobile app, or they can log into the application as a legitimate user after successfully passing the authentication control and then force-browse to a vulnerable endpoint to execute administrative functionality. Both exploitation methods are typically accomplished via mobile malware within the device or botnets owned by the attacker.

Security Weakness

In order to test for poor authorization and authentication schemes in mobile apps, a number of strategies can be employed by testers. For authorization, testers can perform binary attacks against the mobile app and try to execute privileged functionality that should only be executable with a user of higher privilege, particularly while the mobile app is in 'offline' mode. Testers should also attempt to execute any

privileged functionality using a low-privilege session token within the corresponding POST/GET requests for the sensitive functionality to the backend server.

Poor or missing authorization schemes can potentially allow an adversary to execute functionality they should not be entitled to using an authenticated but lower-privilege user of the mobile app. This risk of privilege escalation attack is heightened when authorization decisions are made within the mobile device instead of through a remote server, a scenario that can often arise due to the mobile requirements of offline usability.

Am I Vulnerable To Insecure Authentication / Authorization'?

Understanding the difference between authentication and authorization is paramount in evaluating mobile application security. Authentication identifies an individual, while authorization verifies if the identified individual has the necessary permissions for a particular action. These two aspects are closely related, as authorization checks should immediately follow mobile device request authentication.

Insecure authorization can occur when an organization fails to authenticate an individual before executing a requested API endpoint from a mobile device, as it is virtually impossible to conduct authorization checks on an incoming request without an established caller's identity.

Here are some straightforward indicators of insecure authorization:

- **Presence of Insecure Direct Object Reference (IDOR) vulnerabilities** - Noticing an IDOR vulnerability may suggest that the code isn't conducting a proper authorization check.
- **Hidden Endpoints** - Developers might neglect authorization checks on backend hidden functionality, assuming that the hidden functionality will only be accessed by a user with the appropriate role.
- **User Role or Permission Transmissions** - Should the mobile app transmit the user's roles or permissions to a backend system as part of a request, this could signal insecure authorization.

Similarly, mobile apps can exhibit various signs of insecure authentication:

- **Anonymous Backend API Execution** - The ability of the app to execute a backend API service request without providing an access token may point to insecure authentication.
- **Local Storage of Passwords or Shared Secrets** - If the app stores any passwords or shared secrets locally on the device, this could be a sign of insecure authentication.

- **Weak Password Policy** - The use of a simplified password-entering process may imply insecure authentication.
- **Usage of Features like FaceID and TouchID** - Employing features like FaceID or TouchID could be indicative of insecure authentication.

How Do I Prevent Insecure Authentication and Authorization'?

To prevent both insecure authentication and authorization, it's crucial to avoid weak patterns and reinforce secure measures.

Avoid Weak Patterns

Insecure Mobile Application Authentication Design Patterns should be avoided:

- If you are porting a web application to a mobile equivalent, ensure the authentication requirements of mobile applications match that of the web application component. It should not be possible to authenticate with fewer factors than the web browser.
- Local user authentication can lead to client-side bypass vulnerabilities. If the application stores data locally, the authentication routine can be bypassed on jailbroken devices through runtime manipulation or binary modification. If offline authentication is a compelling

business requirement, consult additional guidance on preventing binary attacks against the mobile app.

- Perform all authentication requests server-side, where possible. Upon successful authentication, application data will be loaded onto the mobile device, ensuring application data availability only after successful authentication.
- If client-side data storage is necessary, encrypt the data using an encryption key securely derived from the user's login credentials. However, there are additional risks that the data will be decrypted via binary attacks.
- The "Remember Me" functionality should never store a user's password on the device.
- Mobile applications should ideally use a device-specific authentication token that can be revoked within the mobile application by the user, mitigating unauthorized access risks from a stolen/lost device.
- Avoid using spoof-able values for user authentication, including device identifiers or geo-location.
- Persistent authentication within mobile applications should be implemented as an opt-in and not enabled by default.
- Where possible, refrain from allowing users to provide 4-digit PIN numbers for authentication passwords.

Insufficient Input/Output Validation

Insufficient validation and sanitization of data from external sources, such as user inputs or network data, in a mobile application can introduce severe security vulnerabilities. Mobile apps that fail to properly validate and sanitize such data are at risk of being exploited through attacks specific to mobile environments, including SQL injection, Command Injection, and cross-site scripting (XSS) attacks.

These vulnerabilities can have detrimental consequences, including unauthorized access to sensitive data, manipulation of app functionality, and potential compromise of the entire mobile system.

Inadequate output validation can result in data corruption or presentation vulnerabilities, allowing malicious actors to inject malicious code or manipulate sensitive information displayed to users.

Exploitability DIFFICULT

Insufficient input/output validation exposes our application to critical attack vectors, including SQL injection, XSS, command injection and path traversal. These vulnerabilities can lead to unauthorized access, data manipulation, code execution, and compromise of the entire backend system.

Am I Vulnerable To ‘Insufficient Input/Output Validation’?

An application can be vulnerable to insufficient input/output validation due to:

- Lack of Input Validation: Failure to properly validate user input can expose the application to injection attacks like SQL injection, command injection, or XSS.
- Inadequate Output Sanitization: Insufficient sanitization of output data can result in XSS vulnerabilities, allowing attackers to inject and execute malicious scripts.
- Context-Specific Validation Neglect: Neglecting to consider specific validation requirements based on data context can create vulnerabilities, such as path traversal attacks or unauthorized access to files.
- Insufficient Data Integrity Checks: Not performing proper data integrity checks can lead to data corruption or unauthorized modification, compromising reliability and security.
- Poor Secure Coding Practices: Neglecting secure coding practices, such as using parameterized queries or escaping/encoding data, contributes to input/output validation vulnerabilities.

How Do I Prevent ‘Insufficient Input/Output Validation’?

To prevent “Insufficient Input/Output Validation” vulnerabilities:

- Input Validation:
    - Validate and sanitize user input using strict validation techniques.
    - Implement input length restrictions and reject unexpected or malicious data.
- Output Sanitization:
    - Properly sanitize output data to prevent cross-site scripting (XSS) attacks.
    - Use output encoding techniques when displaying or transmitting data.
- Context-Specific Validation:
    - Perform specific validation based on data context (e.g., file uploads, database queries) to prevent attacks like path traversal or injection.
- Data Integrity Checks:
    - Implement data integrity checks to detect and prevent data corruption or unauthorized modifications.
- Secure Coding Practices:
    - Follow secure coding practices, such as using parameterized queries and prepared statements to prevent SQL injection.

Insecure Communication

Most modern mobile applications exchange data with one or more remote servers. When the data transmission takes place, it typically goes through the mobile device's carrier network and the internet, a threat agent listening on the wire can intercept and modify the data if it transmitted in plaintext or using a deprecated encryption protocol. Threat agents might have different motives such as stealing sensitive information, conducting espionage, identity theft and more. The following threat agents exist:

- An adversary that shares your local network (compromised or monitored Wi-Fi);
- Rogue carrier or network devices (routers, cell towers, proxy's, etc); or
- Malware on your mobile device.

Security Weakness

While modern mobile applications aim to protect network traffic, they often have inconsistencies in their implementation. These inconsistencies can lead to vulnerabilities that expose data and session IDs to interception. Just because an app uses transport security protocols doesn't mean it's implemented correctly. To identify basic flaws, you can observe the network traffic on

the phone. However, detecting more subtle flaws requires a closer look at the application's design and configuration.

Am I Vulnerable To Insecure Communication'?

This risk covers all aspects of getting data from point A to point B, but doing it insecurely. It encompasses mobile-to-mobile communications, app-to-server communications, or mobile-to-something-else communications. This risk includes all communications technologies that a mobile device might use: TCP/IP, WiFi, Bluetooth/Bluetooth-LE, NFC, audio, infrared, GSM, 3G, SMS, etc.

How Do I Prevent Insecure Communication?

General Best Practices

- Assume that the network layer is not secure and is susceptible to eavesdropping.
- Apply SSL/TLS to transport channels that the mobile app will use to transmit data to a backend API or web service.
- Account for outside entities like third-party analytics companies, social networks, etc. by using their SSL versions when an application runs a routine via the browser/webkit. Avoid mixed SSL sessions as they may expose the user's session ID.

- Use strong, industry standard cipher suites with appropriate key lengths.
- Use certificates signed by a trusted CA provider.
- Never allow bad certificates (self-signed, expired, untrusted root, revoked, wrong host..).
- Consider certificate pinning.
- Always require SSL chain verification.
- Only establish a secure connection after verifying the identity of the endpoint server using trusted certificates in the key chain.
- Alert users through the UI if the mobile app detects an invalid certificate.
- Do not send sensitive data over alternate channels (e.g, SMS, MMS, or notifications).

Inadequate Privacy Controls

Privacy controls are concerned with protecting Personally Identifiable Information (PII), e.g., names and addresses, credit card information, e-mail and IP addresses, information about health, religion, sexuality and political opinions.

This information is valuable to attackers for several reasons. For example, an attacker could

- Impersonate the victim to commit a fraud,
- Misuse the victim's payment data,
- Blackmail the victim with sensitive information or
- Harm the victim by destroying or manipulating the victim's critical data.

Security Weakness

Almost all apps process some kind of PII. Many even collect and process more than they need to fulfill their purpose, which makes them more attractive as a target without business needs.

Risks of privacy violations increase due to careless handling of PII by developers. PII should always be processed with the possibility in mind that an attacker could access communication and storage media.

Hence, an app is vulnerable to privacy infringements if some personal data it collects motivates an attacker to manipulate or abuse that data through a storage or transmission medium that is insufficiently secured.

Am I Vulnerable To 'Inadequate Privacy Controls'?

An app can only be vulnerable to Inadequate Privacy Controls if it processes some form of personally identifiable information. This is almost always the case: Client apps' IP addresses visible to a server, logs of the apps' usage, and metadata sent with crash reports or analytics are PII that apply to most apps. Usually, an app will collect and process additional, more sensitive PII from its users, like accounts, payment data, locations and more.

Given an app that uses PII, it might expose it like any other sensitive data. This most notably happens through

- Insecure data storage and communication (cf. M5, M9),
- Data access with insecure authentification and authorization (cf. M3, M1), and
- Insider attacks on the app's sandbox (cf. M2, M4, M8).

The other OWASP Mobile Top 10 risks provide deeper insights on how an app might be vulnerable to the different attack vectors.

How Do I Prevent 'Inadequate Privacy Controls'?

Something that does not exist cannot be attacked, so the safest approach to prevent privacy violations is to minimize the amount and variety of PII that is processed. This requires full awareness of all PII assets in a given app. With that awareness, the following questions should be assessed:

- Is all PII processed really necessary, e.g., name and address, gender, age?
- Can some of the PII be replaced by less critical information, e.g., fine-grained location by coarse-grained location?

- Can some of the PII be reduced, e.g., location updates every hour instead of every minute?
- Can some of the PII be anonymized or blurred, e.g., by hashing, bucketing, or adding noise?
- Can some of the PII be deleted after some expiration period, e.g., only keep health data of the last week?

Insufficient Binary Protection

Attackers who target app binaries are motivated by various reasons.

The binary could contain valuable secrets, such as commercial API keys or hardcoded cryptographic secrets that an attacker could misuse. In addition, the code in the binary could be valuable on its own, for example, because it contains critical business logic or pre-trained AI models. Some attackers might also not target the app itself but use it to explore potential weaknesses of the corresponding backend to prepare for an attack.

Besides collecting information, attackers could also manipulate app binaries to access paid features for free or to bypass other security checks. In the worst case, popular apps could be modified to contain malicious code and be distributed via third-party app stores or under a new name to exploit unsuspecting users. One common attack example is reconfiguring the payment identifiers in an app, repackaging it, and distributing it via app stores. Then, when users download this unauthorized copy from the app store, the attacker receives the payments instead of the original provider.

Am I Vulnerable To 'Insufficient Binary Protection'?

All apps are vulnerable to binary attacks. Binary attacks can become particularly harmful if the app has sensitive data or algorithms hardcoded in its binary or if it is very popular. If there are additional protective measures, like obfuscation, encoding of secrets in native code (for Android) or similar, successful attacks become harder to achieve but never impossible.

Whether the app is sufficiently secure depends on the business impact that different binary attacks could have. The more motivating it is for attackers and the greater the impact would be, the more effort should be put into protection. Hence, “vulnerability” to binary attacks is highly specific to the given app.

For a quick check, developers can inspect their own app binaries using similar tools as attackers would use. There are many free or affordable tools, like MobSF, otool, apktool and Ghidra that are also quite easy to use and well documented.

How Do I Prevent ‘Insufficient Binary Protection’?

For each app, it should be assessed whether any critical content is contained in the binary or whether its popularity mandates binary protection. If yes, a threat modeling analysis helps to identify the highest risks and their expected financial impact in case they occur. For the most relevant risks, countermeasures should be taken.

Security Misconfiguration

Security misconfiguration in mobile apps refers to the improper configuration of security settings, permissions, and controls that can lead to vulnerabilities and unauthorized access. Threat agents who can exploit security misconfigurations are attackers aiming to gain unauthorized access to sensitive data or perform malicious actions. Threat agents can be an attacker with physical access to the device, a malicious app on the device that exploits security misconfiguration to execute unauthorized actions on the target vulnerable application context.

Am I Vulnerable to Security Misconfigurations?

Mobile apps are vulnerable to security misconfigurations if they have not been properly configured to follow security best practices. Common indicators of vulnerability to security misconfigurations include:

- Default settings not reviewed: Using default configurations without reviewing security settings, permissions and default credentials.
- Lack of secure communication: Using unencrypted or weakly encrypted communication channels.
- Weak or absent access controls: Allowing unauthorized access to sensitive functionality or data.

- Failure to update or patch: Not applying necessary security updates or patches to the app or underlying components.
- Improper storage of sensitive data: Storing sensitive data in plain text or weakly protected formats.
- Insecure file provider path settings: a file content provider that was meant for internal application use is exposed to other apps or users, which could potentially compromise sensitive data or allow unauthorized access to application resources.
- Exported activities: an activity that is meant for internal application use is exported and/or browsable, which exposes an additional attack surface.

To determine if your app is vulnerable to security misconfigurations, you should conduct a thorough security assessment, including code review, security testing, and configuration analysis.

How Do I Prevent Security Misconfigurations?

Preventing security misconfigurations in mobile apps requires following secure coding and configuration practices. Here are some key prevention measures:

- Secure default configurations: Ensure that default settings and configurations are properly secured and do not expose sensitive information or provide unnecessary permissions.
- Default credentials: Refrain from using hardcoded default credentials.

- Insecure permissions: Avoid storing application files with overly permissive permissions like world-readable and/or world-writable.
- Least privilege principle: Request only the permissions necessary for the proper functioning of the application
- Secure network configuration: Disallow cleartext traffic and use certificate pinning when possible.
- Disable Debugging: Disable debugging features in the production version of the app.
- Disable backup mode (Android): By disabling backup mode on Android devices, you prevent the inclusion of app data in the device's backup, ensuring that sensitive data from the app is not stored in the device backup.
- Limit application attack surface by only exporting activities, content providers and services that are necessary to be exported

Insecure Data Storage

Insecure data storage in a mobile application can attract various threat agents who aim to exploit the vulnerabilities and gain unauthorised access to sensitive information. These threat agents include skilled adversaries who target mobile apps to extract valuable data, malicious insiders within the organisation or app development team who misuse their privileges, state-sponsored actors conducting cyber espionage, cybercriminals seeking financial gain through data theft or ransom, script kiddies utilising pre-built tools for simple attacks, data brokers looking to exploit insecure storage for selling personal information, competitors and industrial spies aiming to gain a competitive advantage, and activists or hacktivists with ideological motives.

These threat agents exploit vulnerabilities like weak encryption, insufficient data protection, insecure data storage mechanisms, and improper handling of user credentials. It is crucial for mobile app developers and organisations to implement strong security measures, such as robust encryption, secure data storage practices, and adherence to best practices for mobile application security, to mitigate the risks associated with insecure data storage.

Am I Vulnerable To 'Insecure Data Storage'?

Insecure data storage and unintended data leakage in a mobile application can manifest in several ways, leading to potential privacy breaches and unauthorised access to

sensitive information. Here are common manifestations of these issues:

Lack of Access Controls: Insufficient access controls within the application may allow unauthorised users or attackers to gain access to sensitive data stored on the device or in the app's databases.

Inadequate Encryption: Failure to properly encrypt sensitive data can result in unintended data leakage if an attacker gains access to the storage location. Without encryption, the data is easily readable and can be exploited.

Unintentional Data Exposure: Mobile applications may inadvertently expose sensitive data through application logs, error messages, or debug features, allowing unauthorised individuals to view or capture sensitive information.

Poor Session Management: Weak session management can lead to unintended data leakage. If session tokens or user authentication information are not adequately protected or managed, they can be intercepted or manipulated, allowing unauthorised access to sensitive data.

Insufficient Cryptography

Threat agents who exploit insecure cryptography in mobile applications can undermine the confidentiality, integrity, and authenticity of sensitive information. These threat agents include attackers who target cryptographic algorithms or implementations to decrypt sensitive data, malicious insiders who manipulate cryptographic processes or leak encryption keys, state-sponsored actors engaged in cryptanalysis for intelligence purposes, cybercriminals who exploit weak encryption to steal valuable data or conduct financial fraud, and attackers who leverage vulnerabilities in cryptographic protocols or libraries.

Security weakness

Am I Vulnerable To 'Insufficient Cryptography'?

There are several ways in which insecure cryptography and insecure hash functions can manifest in a mobile application:

Weak Encryption Algorithms: The mobile app may use encryption algorithms that are known to be weak or vulnerable to attacks. These algorithms may have known weaknesses, be outdated, or lack the necessary level of security to protect sensitive data effectively.

Insufficient Key Length: Inadequate key length can weaken the encryption strength. If the mobile app uses short or easily guessable encryption keys, it becomes easier for attackers to

decrypt the encrypted data through brute-force or other cryptographic attacks.

Improper Key Management: Poor key management practices, such as storing encryption keys insecurely or transmitting them in plain text, can expose the keys to unauthorized access. Attackers who gain access to the keys can decrypt the data without difficulty.

Flawed Encryption Implementation: The encryption/decryption process itself may be implemented incorrectly or contain programming flaws. These implementation errors can introduce vulnerabilities that attackers can exploit to bypass or weaken the encryption protections.

How Do I Prevent 'Insufficient Cryptography'?

To prevent "insufficient cryptography" vulnerabilities in the mobile application, consider the following best practices:

Use Strong Encryption Algorithms: Implement widely accepted and secure encryption algorithms, such as AES (Advanced Encryption Standard), RSA (Rivest-Shamir-Adleman), or Elliptic Curve Cryptography (ECC). Stay updated with current cryptographic standards and avoid deprecated or weak algorithms.

Ensure Sufficient Key Length: Select encryption keys with an appropriate length to ensure strong cryptographic strength.

Follow industry recommendations for key lengths, considering the specific encryption algorithm being used.

Follow Secure Key Management Practices: Employ secure key management techniques, such as using key vaults or hardware security modules (HSMs) to securely store encryption keys. Protect keys from unauthorized access, including restricting access to authorized personnel, encrypting keys at rest, and using secure key distribution mechanisms.

Implement Encryption Correctly: Carefully implement encryption and decryption processes in the mobile application, adhering to established cryptographic libraries and frameworks. Avoid custom encryption implementations, as they are more prone to errors and vulnerabilities.

Secure Storage of Encryption Keys: Ensure encryption keys are securely stored on the mobile device. Avoid storing keys in plain text or easily accessible locations. Consider using secure storage mechanisms provided by the operating system or utilizing hardware-based secure storage options.

Comparison between 2016 and 2024

OWASP-2016	OWASP-2024-Release	Comparison Between 2016-2024
M1: Improper Platform Usage	M1: Improper Credential Usage	New
M2: Insecure Data Storage	M2: Inadequate Supply Chain Security	New
M3: Insecure Communication	M3: Insecure Authentication / Authorization	Merged M4&M6 to M3
M4: Insecure Authentication	M4: Insufficient Input/Output Validation	New
M5: Insufficient Cryptography	M5: Insecure Communication	Moved from M3 to M5
M6: Insecure Authorization	M6: Inadequate Privacy Controls	New
M7: Client Code Quality	M7: Insufficient Binary Protections	Merged M8&M9 to M7
M8: Code Tampering	M8: Security Misconfiguration	Rewording [M10]
M9: Reverse Engineering	M9: Insecure Data Storage	Moved from M2 to M9
M10: Extraneous Functionality	M10: Insufficient Cryptography	Moved from M5 to M10

Vulnerabilities that didn't make the place on the initial release list, but in the future, we may consider them.

- Data Leakage
- Hardcoded Secrets
- Insecure Access Control
- Path Overwrite and Path Traversal
- Unprotected Endpoints (Deeplink, Activitity, Service …)
- Unsafe Sharing

Reference :

https://owasp.org/www-project-mobile-top-10/