M3: Insecure Authentication/Authorization

Threat Agents

Application Specific

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

Attack Vectors

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

Prevalence COMMON

Detectability AVERAGE

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.

In terms of poor authentication schemes, testers can undertake binary attacks against the mobile app while it’s in ‘offline’ mode, aiming to bypass offline authentication and then execute functionality that should require offline authentication. Testers should also try to execute any backend server functionality anonymously by removing any session tokens from

- November 2-6, 2026

OWASP Global AppSec EU 2027 - Vienna, Austria
- June 21-25, 2027

OWASP Global AppSec USA 2027 - Atlanta, GA
- September 20-24, 2027

OWASP Global AppSec EU 2028 - Vienna, Austria
- June 19-23, 2028

any POST/GET requests for the mobile app functionality.

Poor or missing authentication schemes can allow an adversary to anonymously execute functionality within the mobile app or the backend server used by the mobile app. These weaknesses in mobile app authentication are fairly common due to the mobile device's input form factor, which often encourages short passwords or 4-digit PINs.

Mobile apps face unique authentication requirements that can diverge from traditional web authentication schemes, largely due to their varying availability requirements. Unlike traditional web apps where users are expected to be online and authenticate in real-time with a backend server, mobile apps may need to fulfill uptime requirements that necessitate offline authentication due to the unreliability or unpredictability of mobile internet connections. This requirement can significantly impact the factors developers must consider when implementing mobile authentication.

Technical Impacts

Impact SEVERE

The technical impact of poor authorization and authentication in a system can be wide-ranging, significant, and similar, largely depending on the type of over-privileged functionality that is executed. When it comes to poor authorization, for instance, over-privileged execution of remote or local administration functionality may destroy systems or access to sensitive information.

The technical repercussions of poor authentication occur when the solution is unable to identify the user performing an action request. This can immediately result in the inability to log or audit user activity since the user's identity cannot be established. This lack of identity verification contributes to an inability to detect the source of an attack, understand the nature of any underlying exploits, or devise strategies to prevent future attacks.

Moreover, failures in authentication can also expose underlying authorization failures. When authentication controls fail, the solution is unable to verify the user's identity, which is closely tied to a user's role and associated permissions. If an attacker can anonymously execute sensitive functionality, it indicates that the underlying code is not verifying the permissions of the user issuing the request for the action. Consequently, the anonymous execution of code underscores failures in both authentication and authorization controls.

Business Impacts

Impact SEVERE

The business impact of poor authentication and authorization will typically result in the following at a minimum:

- Reputation Damage;
- Information Theft;
- Fraud;
- Unauthorized Access to Data.

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.

Reinforce Authentication

- Developers should assume that all client-side authorization and authentication controls can

be bypassed by malicious users. Server-side reinforcement of these controls is critical.
- Due to offline usage requirements, mobile apps might need to perform local authentication or authorization checks. In such cases, developers should instrument local integrity checks to detect any unauthorized code changes. Consult additional guidance on detecting and reacting to binary attacks.
- Use FaceID and TouchID to unlock biometrically locked secrets and securely protect sensitive authentication materials, like session tokens.

Insecure Authorization Prevention

To avoid insecure authorization:

- Backend systems should independently verify the roles and permissions of the authenticated user. Do not rely on any roles or permission information that comes from the mobile device.
- Assume that all client-side authorization can be bypassed, hence reinforcing server-side authorization controls whenever possible.
- If offline authorization checks are necessary within the mobile app’s code, developers should perform local integrity checks to detect unauthorized code changes.

Example Attack Scenarios

The following scenarios showcase weak authentication or authorization controls in mobile apps:

Scenario #1: Hidden Service Requests: Developers assume that only authenticated users will be able to generate a service request that the mobile app submits to its backend for processing. During the processing of the request, the server code does not verify that the incoming request is associated with a known user. Hence, adversaries submit service requests to the back-end service and anonymously execute functionality that affects legitimate users of the solution.

Scenario #2: Interface Reliance: Developers assume that only authorized users will be able to see the existence of a particular function on their mobile app. Hence, they expect that only legitimately authorized users will be able to issue the request for the service from their mobile devices. The back-end code that processes the request does not bother to verify that the identity associated with the request is entitled to execute the service. Hence, adversaries are able to perform remote administrative functionality using fairly low-privilege user accounts.

Scenario #3: Usability Requirements: Due to usability requirements, mobile apps allow for passwords that are 4 digits long. The server code correctly stores a hashed version of the password. However, due to the severely short length of the password, an adversary will be able to quickly deduce the original passwords using rainbow hash tables. If the password file (or data store) on the server is compromised, an adversary will be able to quickly deduce users' passwords.

Scenario #4: Insecure Direct Object Reference: A user makes an API endpoint request to a backend REST API that includes an actor ID and an OAuth bearer token. The user includes their actor ID as part of the incoming URL and includes the access token as a standard header in the request. The backend verifies the presence of the bearer token but fails to validate the actor ID associated with the bearer token. As a result, the user can tweak the actor ID and attain the account information of other users as part of the REST API request.

Scenario #5: Transmission of LDAP roles: A user makes an API endpoint request to a backend REST API that includes a standard oAuth bearer token along with a header that includes a list of LDAP groups that the user belongs to. The backend request validates the bearer token and then inspects the incoming LDAP groups for the right group membership before continuing on to the sensitive functionality. However, the backend system does not perform an independent validation of LDAP group membership and instead relies upon the incoming LDAP information coming from the user. The user can tweak the incoming header and report to be a member of any LDAP group arbitrarily and perform administrative functionality.

References

- OWASP
  - OWASP
- External
  - External References

Edit on GitHub

Spotlight: Bloomberg

Bloomberg®

Bloomberg is a global leader in business and financial information, delivering trusted data, news, and insights that bring transparency, efficiency, and fairness to markets. The company helps connect influential communities across the global financial ecosystem via reliable technology solutions that enable our customers to make more informed decisions and foster better collaboration.

Corporate Supporters

tenable BDO appdome

Checkmarx AUTOMATTIC Fortify

GUARDSQUARE Mobile application protection Beagle Security salesforce

Become a corporate supporter

HOME PROJECTS CHAPTERS EVENTS ABOUT

PRIVACY SITEMAP CONTACT