CHECKLIST

Checklist: Secure JWT Authentication in ASP.NET Core APIs

A practical checklist for implementing and validating JWT authentication in ASP.NET Core APIs, with emphasis on token validation, claim design, key handling, transport security, and production readiness.

Secure JWT Authentication in ASP.NET Core APIs ? Checklist

Use this checklist to verify that your JWT-based API authentication design is secure, production-ready, and operationally manageable.

1) Confirm JWT is the right fit

- [ ] Use JWTs only for stateless API access where server-side sessions are not required.
- [ ] Keep access tokens short-lived.
- [ ] Use refresh tokens or another re-authentication strategy if long-lived access is required.
- [ ] Avoid JWTs when immediate centralized revocation is a hard requirement and no revocation strategy exists.

2) Define the token contract before implementation

- [ ] Identify the token issuer.

- [ ] Define the intended audience for the API.
- [ ] Set a clear token lifetime.
- [ ] Decide which claims are required.
- [ ] Keep the claims model minimal and purpose-built.
- [ ] Document expected roles, scopes, or permissions.

3) Protect signing keys and secrets

- [ ] Store signing keys in a secure secret store.
- [ ] Never commit signing keys to source control.
- [ ] Restrict access to signing keys to the smallest practical set of services and operators.
- [ ] Plan for key rotation before production.
- [ ] Validate how the API behaves during key rollover.
- [ ] Use different keys for different environments.

4) Configure ASP.NET Core JWT validation correctly

- [ ] Enable JWT bearer authentication in the API.
- [ ] Validate the token signature on every request.
- [ ] Validate the issuer.
- [ ] Validate the audience.
- [ ] Validate token lifetime.
- [ ] Set clock skew conservatively.
- [ ] Reject tokens signed with unexpected algorithms or keys.
- [ ] Ensure invalid tokens are rejected before application logic runs.

5) Enforce secure transport and request handling

- [ ] Require HTTPS for all authenticated traffic.
- [ ] Do not allow bearer tokens over unencrypted channels.
- [ ] Treat the Authorization header as sensitive data.
- [ ] Avoid logging full tokens.
- [ ] Sanitize authentication failure logs.
- [ ] Confirm proxies, gateways, and load balancers preserve authentication headers correctly.

6) Design claims carefully

- [ ] Include identity claims only when needed.
- [ ] Use tenant, role, or scope claims for authorization context.
- [ ] Avoid sensitive business data in token payloads.
- [ ] Avoid passwords, API secrets, connection strings, or internal records in claims.
- [ ] Keep tokens small enough to reduce exposure and replay value.
- [ ] Validate required claims exist and are well formed.

7) Apply authorization separately from authentication

- [ ] Use authentication only to verify the caller and token validity.
- [ ] Use authorization policies to decide endpoint access.
- [ ] Restrict endpoints by role, scope, claim, or policy as appropriate.
- [ ] Confirm public, authenticated, and privileged routes are clearly separated.
- [ ] Review multi-tenant access rules carefully.

8) Validate expiration and clock behavior

- [ ] Test with expired tokens.

- [ ] Test with tokens that are not yet valid, if applicable.
- [ ] Verify issuer and API clocks are reasonably synchronized.
- [ ] Confirm the application behaves consistently near token expiry.
- [ ] Monitor for failures caused by clock drift across environments.

9) Review operational failure modes

- [ ] Confirm what happens when the signing key is rotated.
- [ ] Confirm what happens when the issuer is unreachable, if your design depends on it.
- [ ] Confirm what happens when audience or issuer values are misconfigured.
- [ ] Confirm expired or malformed tokens fail closed.
- [ ] Confirm authentication failures do not expose internal implementation details.

10) Prepare for production

- [ ] Document the token issuer, audience, lifetime, and claims contract.
- [ ] Document key management and rotation procedures.
- [ ] Document incident response steps for suspected key compromise.
- [ ] Add tests for valid, expired, tampered, and mis-audience tokens.
- [ ] Validate behavior in staging with production-like configuration.
- [ ] Review logs, metrics, and alerts for authentication anomalies.

11) Production readiness checklist

Before launch, confirm all of the following:

- [ ] Signature validation is enforced.
- [ ] Issuer validation is enforced.
- [ ] Audience validation is enforced.

- [ ] Lifetime validation is enforced.
- [ ] HTTPS is mandatory.
- [ ] Signing keys are stored securely.
- [ ] Claims are minimal and non-sensitive.
- [ ] Authorization policies are in place.
- [ ] Key rotation has been tested.
- [ ] Failure logging is safe and useful.
- [ ] Clock skew has been checked.
- [ ] Token revocation strategy has been considered.

Quick rule of thumb

If the token is long-lived, overloaded with data, weakly validated, or difficult to rotate safely, the implementation is not ready for production.

Keep access tokens short, validate them strictly, and treat key management and transport security as part of authentication itself.