Authentication & Authorization – Question & Answer Notes

1️⃣What happens if we remove the session cookie?

Answer:
If the session cookie is removed, the server should treat the request as unauthenticated.

- If the application still returns sensitive data → **Authentication flaw**
- If it returns 200 but no sensitive data → could be normal behavior
- If it allows access → **Broken Authentication**

Proper behavior:

- Server must validate session ID from backend storage
- If invalid/missing → return 401 Unauthorized or redirect to login

2️⃣What is Session Fixation?

Answer:
Session fixation is a vulnerability where the application does not rotate the session ID after login.

Root cause:

- Server keeps same session ID before and after authentication.

Why dangerous?

- Attacker can force victim to use a known session ID.
- After victim logs in, attacker reuses same session ID.

Secure behavior:

- Regenerate session ID after successful login.

3️⃣What is IDOR?

Answer:
IDOR (Insecure Direct Object Reference) happens when the application does not verify object ownership.

Example:

```
GET /api/user/123/profile
```

If changing `123` to `124` gives another user's data → IDOR.

Root cause:

- No backend authorization check.
- Server trusts user-controlled parameters.

Fix:

- Always verify ownership server-side.
- Never trust user-supplied IDs.

4️⃣ What is Authorization?

Answer:
Authorization ensures a user can only access resources they are allowed to.

Example:

- Normal user must NOT access:
  - /admin
  - Hidden files
  - Other users' data

If backend check is missing → privilege escalation possible.

5️⃣ Why is frontend validation insecure?

Answer:
Frontend validation can be bypassed using:

- Burp Suite
- DevTools
- Custom requests

If access control exists only in frontend:
→ User can modify request and gain admin privileges.

Security must always be enforced on backend.

6️⃣ What happens if role parameter is modifiable?

Answer:
If user can change:

role=user → role=admin

And server trusts it → Privilege Escalation.

Root cause:

- Server trusting client-side data.

Fix:

- Role must be stored and verified from database only.
- Never trust role in request.

7️⃣ What is Privilege Escalation?

Answer:
Privilege escalation happens when a user gains higher permissions than intended.

Types:

- Vertical → User → Admin
- Horizontal → Accessing other users' data

Causes:

- Missing backend role validation
- IDOR
- Parameter manipulation

8️⃣ What if token is predictable?

Answer:
If session ID or token is predictable:
→ Attacker can guess valid sessions.

This leads to:

- Session hijacking
- Account takeover

Secure practice:

- Use cryptographically secure random tokens.

9️⃣ What happens if logout does not invalidate token?

Answer:
If logout does not:

- Delete session
- Blacklist JWT
- Expire token

Then stolen token can still be used.

Secure logout:

- Destroy server session
  OR
- Maintain token blacklist (for JWT)

🔟 What if JWT is modified?

Answer:
If JWT signature is not verified:
→ Attacker can modify payload.

Example:

role: user → role: admin

Secure behavior:

- Always verify JWT signature.
- Never trust decoded payload blindly.

- Use strong secret key.

1️⃣1️⃣ Why must trust be in the database?

Answer:
Trust must always come from:

- Database
- Server-side session store

Never from:

- Client request
- Headers
- Cookies
- Hidden inputs

Because client-side data can always be modified.

1️⃣2️⃣ What happens if backend does not verify user ID?

Answer:
If backend only checks:

- "Is user logged in?"

But does NOT check:

- "Is this resource owned by this user?"

Then:
→ IDOR
→ Information Disclosure

1️⃣3️⃣ What is Session Hijacking?

Answer:
Session hijacking occurs when attacker steals a valid session ID.

Common causes:

- XSS stealing cookies

- No HTTPS
- Predictable session IDs

Impact:

- Attacker acts as victim user.

1️⃣4️⃣ What is Broken Access Control?

Answer:
Broken Access Control occurs when restrictions are not properly enforced.

Examples:

- Accessing admin endpoints as user
- Changing user ID in request
- Accessing hidden endpoints

This is OWASP Top 10 vulnerability.

1️⃣5️⃣ What should happen if normal user tries admin endpoint?

Answer:
System should:

- Deny access
- Return 403 Forbidden

Server must:

- Check role from backend
- Ignore client-supplied role

1️⃣6️⃣ What is secure token rotation?

Answer:
Token rotation means:

- New token generated after login
- New token generated after privilege change

- Refresh token rotation

Purpose:

- Prevent session fixation
- Reduce impact of token theft

1️⃣7️⃣ What happens if server trusts User-Agent?

Answer:
User-Agent header can be modified easily.

If server trusts it:
→ Security bypass possible

Never rely on:

- User-Agent
- IP alone
- Hidden fields

1️⃣8️⃣ What is the main root cause of most auth vulnerabilities?

Answer:

1. Trusting client-side input
2. Missing backend authorization check
3. No ownership validation
4. Improper session handling
5. Weak token management

Core Security Principle

Never trust the client.
Always validate on the server.
Authentication proves identity.
Authorization verifies permission.

Advanced Authentication & Authorization – Deep Security Q&A

1️⃣ If the system stores role inside JWT for performance, what is a secure alternative?

Answer:

Storing role inside JWT improves performance (stateless auth), but creates risk if role changes in DB.

Secure alternatives:

- Store only `user_id` in JWT.
- Fetch role from database on every sensitive request.
- Use short-lived access tokens.
- Implement token versioning (store token_version in DB and compare).
- Maintain role change invalidation logic (force token refresh when role changes).

Best practice:

> JWT should prove identity, not be the single source of authorization truth.

2️⃣ If refresh token reuse is detected, what should the system do?

Answer:

Refresh token reuse usually indicates token theft.

Secure response:

- Immediately revoke all active sessions of that user.
- Invalidate all refresh tokens.
- Force user re-authentication.
- Log the incident for security monitoring.
- Optionally notify the user.

This is called **Refresh Token Rotation with Reuse Detection**.

3️⃣ If user changes email but old password reset token is still valid for previous email , what is the issue?

Answer:

This creates an **Account Takeover risk**.

Problem:

- Reset token tied to old email still works.
- Attacker with access to old email can reset password.

Secure behavior:

- Invalidate all password reset tokens when email changes.
- Bind reset token strictly to user ID, not email string.

4️⃣ If rate limiting is applied only to login but not to password reset OTP , what can happen?

Answer:

Possible attacks:

- OTP brute force attack.
- Account takeover.
- Enumeration attack.

Security principle:

> All authentication-related endpoints must have rate limiting.

5️⃣ If user role changed in DB (admin → user) but active JWT still says admin , what should system do?

Answer:

System must NOT trust old JWT blindly.

Secure options:

- Use short-lived access tokens.
- Use token versioning.
- Check role from DB on critical operations.
- Force logout when role changes.

Authorization must reflect current database state.

6️⃣ If access control middleware is applied only to web routes but not API routes , what is the risk?

Answer:

This creates a **Broken Access Control vulnerability**.

Attack:

- Attacker bypasses UI.
- Directly calls API endpoints.

Impact:

- Privilege escalation
- Data exposure
- Admin actions via API

All access control must be centralized.

7️⃣ If project ownership check exists but child objects (tasks) are not validated , what vulnerability exists?

Answer:

This is **Indirect Object Reference / Broken Object-Level Authorization (BOLA)**.

Example:

- User owns Project A.
- Changes task ID to access task from Project B.

Root cause:

- Ownership validation not applied consistently to nested objects.

8️⃣ If access token expires but refresh token can be reused unlimited times , what flaw exists?

Answer:

This creates:

- Long-term session abuse
- No effective session expiration
- Token replay risk

Secure practice:

- Rotate refresh tokens.
- Set refresh token expiration.
- Detect reuse attempts.

9️⃣ If JWT is valid and signed, but user is disabled in DB , should access be allowed?

Answer:

No.

Even if JWT is cryptographically valid:

- System must check account status.
- Disabled users must be denied access.

Identity validation ≠ Authorization validation.

🔟 If DB says user is admin, but JWT says role=user , which should be trusted?

Answer:

Database must be trusted.

JWT claims can be outdated.

Authorization decisions must reflect current DB state.

1️⃣1️⃣ If access control exists only in frontend, not backend , what happens?

Answer:

Attacker can:

- Modify request via Burp Suite.
- Call restricted endpoints directly.
- Escalate privileges.

Frontend checks are cosmetic only.

1️⃣2️⃣ If resource owner check exists but indirect reference is predictable , what is the risk?

Answer:

Even with owner check, predictable references may allow:

- ID enumeration attempts.
- Resource discovery.
- Metadata leakage.

IDs should be:

- Random
- Non-sequential
- Hard to guess

1️⃣3️⃣ If logout does not invalidate JWT , what is long-term risk?

Answer:

If JWT remains valid:

- Stolen token can still be used.
- Session hijacking possible.
- No real logout.

Secure logout requires:

- Token blacklist
  OR
- Very short token lifetime

1️⃣4️⃣ If admin delete API checks role but update-role API does not , what can happen?

Answer:

Privilege escalation.

Attack:

- Normal user calls update-role API.
- Changes own role to admin.

This is **Function-Level Authorization bypass**.

1️⃣5️⃣ If logout blocks UI access but background API still works , what issue exists?

Answer:

This indicates:

- Token not invalidated properly.
- Session still active in backend.

UI logout ≠ real session destruction.

1️⃣6️⃣ If role is checked only during login but not on every request , what is the risk?

Answer:

If role changes later:

- User keeps old privileges.
- Authorization becomes stale.
- Privilege abuse possible.

Authorization must be enforced per request.

1️⃣7️⃣ If session rotates on login but remains active after logout , is it secure?

Answer:

No.

Logout must:

- Destroy session server-side.
- Invalidate tokens.
- Clear cookies.

Otherwise session hijacking remains possible.

1️⃣8️⃣ If user A cannot access data, but error message reveals user B’s email , is it vulnerability?

Answer:

Yes.

This is **Information Disclosure**.

Impact:

- User enumeration
- Targeted phishing
- Privacy violation

Error messages must not reveal sensitive data.

1️⃣9️⃣ If admin endpoint is blocked, but same action possible via normal endpoint ,what problem is this?

Answer:

This is **Access Control Inconsistency**.

Security must be:

- Action-based
- Not URL-based

All business logic paths must enforce same authorization.

2️⃣0️⃣ If JWT is properly signed but modifying role claim still works , what is root issue?

Answer:

Possible causes:

- Signature not being verified.
- Using "alg: none".
- Server decoding JWT without verifying.
- Weak secret key.
- Incorrect verification implementation.

Cryptographic validation must always be enforced.

Core Security Principles Summary

1. Never trust client input.
2. Authorization must be server-side.
3. Identity ≠ Permission.
4. Tokens must be short-lived.
5. Role must be validated per request.
6. Logout must invalidate tokens.
7. Access control must be consistent across all endpoints.

Documented by: ***Mohammad Riyazuddin***