Security Fix Documentation

Broken Access Control in Question Controller Endpoints

Severity: High

Category: Broken Access Control / IDOR

OWASP A01:2021 — Broken Access Control

Component	`questionController.js`
Affected Routes	`POST /api/question/:id/pin` `POST /api/question/:id/note`
Fix Status	**Resolved**
Document Type	Internal Engineering Security Report
Revision	1.0

Backend Security Engineering Team

Contents

1 Overview

This document describes a Broken Access Control vulnerability discovered in two backend API endpoints responsible for managing interview questions. The endpoints allowed any authenticated user to modify another user's question data by supplying a known question ID, bypassing ownership checks entirely.

Issue Summary

Vulnerability: Insecure Direct Object Reference (IDOR) in `togglePinQuestion` and `updateQuestionNote` handlers.

Impact: Any authenticated user could pin, unpin, or overwrite notes on questions that belong to another user's session.

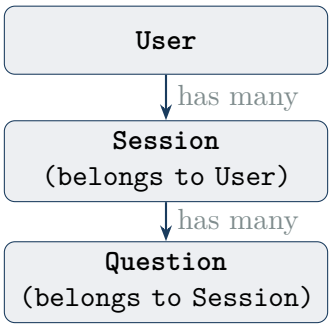
Resolution: Session-level ownership validation added before any mutation is allowed.

2 Application Architecture

Understanding the data model is necessary for evaluating the authorization requirements of question-related operations.

2.1 Data Model Hierarchy

The application uses a three-level ownership chain:



Because `Question` does not embed the user reference directly, authorization must traverse the chain: a user is authorized to modify a question only if they own the session that contains it:

```
session.user.toString() === req.user.id
```

Listing 1: Ownership invariant

2.2 Affected Endpoints

Both routes accept a question ID via the URL path parameter `:id` and require a valid JWT for authentication. Prior to this fix, no further authorization was performed.

3 Vulnerability Description

Method	Route	Description
POST	/api/question/:id/pin	Toggles the isPinned boolean on a question. Used to mark important questions for quick review.
POST	/api/question/:id/note	Writes or updates a free-text note field (note) on a question. Used to attach personal annotations.

Table 1: Vulnerable endpoints in questionController.js

3.1 Vulnerable Code

The original handler logic for togglePinQuestion is shown below. An identical pattern existed in updateQuestionNote.

```
// POST /api/question/:id/pin
const togglePinQuestion = async (req, res) => {
    try {
        const question = await Question.findById(req.params.id
           );

        // No existence check, no ownership check
        question.isPinned = !question.isPinned;
        await question.save();

        res.status(200).json({
            success: true,
            question,
        });
    } catch (error) {
        res.status(500).json({ success: false, message: error.
           message });
    }
};
```

Listing 2: Vulnerable togglePinQuestion handler (before fix)

Security Gap

The handler verifies only that the caller holds a valid JWT token. It does **not** verify that the caller owns the question or its parent session. Any authenticated user who knows a valid question_id can mutate any question in the system.

3.2 What Was Missing

1. **Existence check** — No validation that the question ID resolves to an existing document.
2. **Session resolution** — The parent session of the question was never fetched.

3. **Ownership validation** — No comparison between `session.user` and `req.user.id`.

4 Vulnerability Reproduction

4.1 Prerequisites

- Two registered accounts: **User A** (victim) and **User B** (attacker).
- API testing tool (Postman or `curl`).
- The application running with the *unpatched* codebase.

4.2 Step-by-Step Reproduction

4.2.1 User A — Legitimate Setup

Step 1. Register and authenticate as User A.

```
{
    "email": "userA@example.com",
    "password": "SecurePass123"
}
```

Listing 3: POST /api/auth/register — User A

Save the returned JWT (`tokenA`).

Step 2. Create an interview session.

```
// Authorization: Bearer <tokenA>
{
    "title": "Frontend Interview Session"
}
```

Listing 4: POST /api/sessions — User A

Save the returned `session_id`.

Step 3. Add a question to the session.

```
// Authorization: Bearer <tokenA>
{
    "sessionId": "<session_id>",
    "content": "Explain the virtual DOM."
}
```

Listing 5: POST /api/questions — User A

Save the returned `question_id` (this is the target ID).

4.2.2 User B — Exploiting the IDOR

Step 4. Register and authenticate as User B.

```
{
    "email": "userB@example.com",
    "password": "AnotherPass456"
}
```

Listing 6: POST /api/auth/register — User B

Save the returned JWT (`tokenB`).

Step 5. Send a pin request targeting User A's question.

```
curl -X POST https://api.example.com/api/question/<
   question_id>/pin \
     -H "Authorization: Bearer <tokenB>"
```

Listing 7: POST /api/question/<question_id>/pin — User B targeting User A's resource

Step 6. Observe the vulnerable response.

```
{
    "success": true,
    "question": {
        "_id": "<question_id>",
        "content": "Explain the virtual DOM.",
        "isPinned": true,
        "session": "<session_id>"
    }
}
```

Listing 8: Vulnerable server response — 200 OK (expected: 403 Forbidden)

Confirmed Exploit

User B successfully pinned User A's question using only a valid JWT and a known question ID. No session ownership was validated. The server returned `200 OK` and persisted the change.

5 Root Cause Analysis

5.1 Authentication vs. Authorization

The middleware chain for the affected routes enforced **authentication** (JWT verification) but implemented no **authorization** (permission to act on the resource).

5.2 OWASP Classification

This vulnerability falls under **OWASP A01:2021 — Broken Access Control**, and more specifically represents an **Insecure Direct Object Reference (IDOR)**:

- Object references (question IDs) are exposed directly in the URL.
- The server performs no authorization check before resolving and mutating the referenced object.

Control	What It Checks	Present?
Authentication	Is the caller who they claim to be? (JWT)	**Yes**
Resource existence	Does the question ID exist?	**No**
Session resolution	Does a parent session exist?	**No**
Ownership validation	Does `session.user == req.user.id`?	**No**

Table 2: Authorization controls present before the fix

- An attacker can substitute any valid ID to access or modify resources that belong to other users.

6 Implemented Fix

6.1 Fix Strategy

The resolution adds three sequential validation steps to both handlers before any mutation is allowed:

1. Verify the question exists.
2. Resolve the parent session from the question.
3. Assert that the session owner matches the authenticated caller.

6.2 Fixed Handler — `togglePinQuestion`

```
// POST /api/question/:id/pin
const togglePinQuestion = async (req, res) => {
    try {
        // Step 1: Confirm the question exists
        const question = await Question.findById(req.params.id
            );
        if (!question) {
            return res.status(404).json({
                success: false,
                message: "Question not found",
            });
        }

        // Step 2: Resolve the parent session
        const session = await Session.findById(question.
            session);
        if (!session) {
            return res.status(404).json({
                success: false,
                message: "Session not found",
            });
        }

```

```
        // Step 3: Enforce ownership
        if (session.user.toString() !== req.user.id) {
            return res.status(403).json({
                success: false,
                message: "Unauthorized access",
            });
        }

        // Mutation only reaches here if all checks pass
        question.isPinned = !question.isPinned;
        await question.save();

        res.status(200).json({ success: true, question });
    } catch (error) {
        res.status(500).json({ success: false, message: error.
           message });
    }
};
```

Listing 9: Patched `togglePinQuestion` handler

6.3 Fixed Handler — `updateQuestionNote`

```
// POST /api/question/:id/note
const updateQuestionNote = async (req, res) => {
    try {
        const { note } = req.body;

        // Step 1: Confirm the question exists
        const question = await Question.findById(req.params.id
           );
        if (!question) {
            return res.status(404).json({
                success: false,
                message: "Question not found",
            });
        }

        // Step 2: Resolve the parent session
        const session = await Session.findById(question.
           session);
        if (!session) {
            return res.status(404).json({
                success: false,
                message: "Session not found",
            });
        }

        // Step 3: Enforce ownership
        if (session.user.toString() !== req.user.id) {
            return res.status(403).json({
```

```
                success: false,
                message: "Unauthorized access",
            });
        }

        // Mutation only reaches here if all checks pass
        question.note = note;
        await question.save();

        res.status(200).json({ success: true, question });
    } catch (error) {
        res.status(500).json({ success: false, message: error.
           message });
    }
};
```

Listing 10: Patched `updateQuestionNote` handler

6.4 Core Ownership Check

Both handlers share the same authorization predicate. Any mismatch results in an immediate `403 Forbidden`:

```
if (session.user.toString() !== req.user.id) {
    return res.status(403).json({
        success: false,
        message: "Unauthorized access",
    });
}
```

Listing 11: Shared ownership guard

7 Secure Behavior — Expected Outcomes

7.0.1 Secure Error Response (403)

```
{
    "success": false,
    "message": "Unauthorized access"
}
```

Listing 12: Response returned when a non-owner attempts modification

8 Verification and Testing

8.1 Manual Verification via Postman

Testing was carried out manually with two separate Postman environments, each holding a distinct JWT for User A and User B respectively.

Scenario	HTTP Status	Success	Notes
Owner modifies their own question	`200 OK`	`true`	Mutation applied and saved
Authenticated non-owner targets another user's question	`403 Forbidden`	`false`	Ownership check fails
Request with an invalid/non-existent question ID	`404 Not Found`	`false`	Existence check fails
Unauthenticated request (missing/invalid JWT)	`401 Unauthorized`	`false`	Auth middleware rejects

Table 3: Expected HTTP responses after applying the fix

8.1.1 Test Case 1 — Unauthorized Cross-User Access (Must Fail)

1. Authenticate as User A; create a session and question. Record the `question_id`.
2. Authenticate as User B; send `POST /api/question/<question_id>/pin` using `tokenB`.
3. **Expected:** `HTTP 403 Forbidden`, `"success": false`, `"message": "Unauthorized access"`.
4. **Result:** **PASS** — 403 confirmed.

8.1.2 Test Case 2 — Authorized Owner Access (Must Succeed)

1. Authenticate as User A; send `POST /api/question/<question_id>/pin` using `tokenA`.
2. **Expected:** `HTTP 200 OK`, `"success": true`, updated `isPinned` value returned.
3. **Result:** **PASS** — 200 confirmed, toggle applied correctly.

8.1.3 Test Case 3 — Invalid Question ID (Must 404)

1. Send `POST /api/question/000000000000000000000000/pin` with any valid JWT.
2. **Expected:** `HTTP 404 Not Found`.
3. **Result:** **PASS** — 404 confirmed.

8.1.4 Test Case 4 — `updateQuestionNote` Parity

Repeat Test Cases 1–3 against the `/note` endpoint with a `{"note": "test"}` JSON body.

- Cross-user: **PASS** — 403 returned.
- Legitimate owner: **PASS** — 200 returned, note saved.
- Invalid ID: **PASS** — 404 returned.

8.2 Regression Summary

Test Case	Endpoint	Result
Owner pins own question	`/pin`	PASS
Non-owner pins another user's question	`/pin`	PASS
Invalid question ID on pin	`/pin`	PASS
Owner updates own note	`/note`	PASS
Non-owner updates another user's note	`/note`	PASS
Invalid question ID on note	`/note`	PASS

Table 4: Regression test results — all cases passing post-fix

9 Security Impact

9.1 Risk Before the Fix

Pre-Fix Risk Exposure

- **Unauthorized data modification:** Any authenticated user could alter the `isPinned` flag or `note` content of any question in the database.
- **Cross-user tampering:** Persistent changes were written to the database with no audit trail linking the mutation to the actual actor.
- **Privilege abuse:** The authentication layer provided a false sense of security; a valid token was sufficient to grant write access to the entire question collection.
- **API misuse:** Automated enumeration of sequential or guessable IDs could be used to tamper with data at scale.

9.2 Risk After the Fix

Post-Fix Security Posture

- Mutation is gated behind a session-level ownership assertion.
- Only the authenticated user who owns the parent session can modify questions within it.
- Any cross-user access attempt returns `403 Forbidden` with no data leakage in the response body.
- Invalid resource references return `404 Not Found`, preventing confirmation of resource existence to unauthorized callers.

9.3 OWASP A01:2021 — Broken Access Control

This vulnerability is a direct instance of the top-ranked risk in the OWASP Top 10 (2021). The fix aligns the implementation with the following OWASP mitigation guidance:

- Enforce access control checks at the server on every request, not just at the middleware level.

- Deny access by default; require explicit authorization for each resource operation.
- Use domain model ownership (user → session → question) to validate authorization rather than relying on object ID secrecy.
- Return `403` (not `401` or `200`) when authorization fails.

10 Recommendations

1. **Audit all mutation endpoints** — Review every `PUT`, `PATCH`, `POST`, and `DELETE` route that accepts a resource ID from the caller. Confirm ownership validation is present.
2. **Abstract ownership checks into middleware** — Consider extracting the session-ownership guard into a reusable Express middleware function to avoid duplicating the pattern across controllers.
3. **Add automated integration tests** — Write tests that assert cross-user mutation attempts return `403` for every protected endpoint. Include these in CI to prevent regressions.
4. **Enable request logging** — Ensure failed authorization attempts (`403` responses) are logged with the caller's user ID and target resource ID for anomaly detection.
5. **Apply the principle of least privilege broadly** — Read endpoints that return sensitive user data should undergo the same ownership review.

11 Conclusion

A Broken Access Control / IDOR vulnerability in `questionController.js` allowed any authenticated user to modify questions owned by other users. The root cause was an absent ownership validation step between authenticating the caller and applying the mutation.

The fix introduces a three-step guard — question existence, session resolution, and session ownership comparison — that runs before any write operation is permitted. All test cases, including regression tests for legitimate owners, pass after the patch was applied.

No changes to the authentication middleware, data model, or routing layer were required. The fix is contained to the two affected handler functions and introduces no breaking changes to the API contract for authorized callers.

End of Document