
WEEK 2 – DAY 10

BROKEN ACCESS CONTROL (OWASP TOP 10)

---

1. WHAT IS BROKEN ACCESS CONTROL?

Simple Definition

Broken Access Control occurs when users can **perform actions or access data they are not authorized to**.

Authentication answers:

“Who are you?”

Authorization answers:

“What are you allowed to do?”

Most serious breaches happen **after login**, due to **missing or incorrect authorization checks**.

---

2. WHY BROKEN ACCESS CONTROL IS SO DANGEROUS

Because:

- Attackers do **not need to hack passwords**
- They only need to **change IDs or URLs**
- Exploitation is often **silent**
- It directly exposes **business-critical data**

OWASP consistently ranks this as **#1 or #2**.

---

3. IDOR (INSECURE DIRECT OBJECT REFERENCE)

What Is IDOR?

IDOR happens when an application exposes internal object identifiers (IDs) and does **not verify ownership or permissions**.

Real-World Example

URL:

GET /api/users/123/profile

Attacker changes it to:

GET /api/users/124/profile

If server responds → **IDOR vulnerability**

Why This Happens

- Developer trusts client input
- Authorization logic is missing
- “User is logged in” is incorrectly treated as “User is authorized”

---

4. COMMON IDOR SCENARIOS

- Viewing another user’s profile
- Downloading someone else’s invoice
- Modifying another user’s data
- Accessing admin endpoints

---

5. HANDS-ON: EXPLOITING IDOR

Vulnerable Flask Example

```
@app.route("/profile/<int:user_id>")

@login_required

def profile(user_id):

   user = User.query.get(user_id)

   return render_template("profile.html", user=user)
```

Exploit

1. Login as user ID 5
2. Change URL to /profile/1
3. Access another user’s data

---

6. MISSING AUTHORIZATION CHECKS

Very Common Mistake

```
@app.route("/admin")

@login_required

def admin_panel():

   return render_template("admin.html")
```

Problem:

- Only checks login
- No role or permission validation

Attack Impact

- Regular user accesses admin features
- Data manipulation
- Privilege escalation

---

7. FIXING IDOR: OWNERSHIP VALIDATION

Secure Version

```
@app.route("/profile/<int:user_id>")
@login_required
def profile(user_id):
    if current_user.id != user_id:
        abort(403)

    user = User.query.get(user_id)
    return render_template("profile.html", user=user)
```

Even Better Design

```
@app.route("/profile")
@login_required
def profile():
    user = current_user
    return render_template("profile.html", user=user)
```

Best practice:
Do not accept user IDs from the client if unnecessary.

---

8. FIXING AUTHORIZATION (ROLE-BASED)

```
@app.route("/admin")
```

```
@login_required
def admin_panel():
   if not current_user.has_role("admin"):
      abort(403)
   return render_template("admin.html")
```

---

9. DEFENSE-IN-DEPTH STRATEGIES

- Server-side authorization checks
- Deny by default
- Centralized access control
- Least privilege
- Logging unauthorized access attempts

---

10. INTERVIEW QUESTIONS & MODEL ANSWERS

Q1: What is IDOR?

“IDOR is an access control vulnerability where an application exposes object identifiers without validating whether the user is authorized to access them.”

Q2: Why is Broken Access Control dangerous?

“Because attackers can access sensitive data without breaking authentication.”

Q3: How do you prevent IDOR?

“By enforcing ownership validation, role checks, and avoiding exposing internal IDs.”

Q4: Why is authentication not enough?

“Because login does not imply permission.”

---

11. ATTACKER VS DEFENDER THINKING

Attacker:

- What IDs can I change?
- Can I access admin endpoints?

- Are permissions enforced?

Defender:

- Is every request authorized?
- Are permissions verified server-side?
- Do we deny by default?