1. BROKEN ACCESS CONTROL

OWASP Top 10 — Module 1

A01: Broken Access Control — Complete Pentester Module

---

1. Introduction

Broken Access Control is currently the **#1 most exploited vulnerability on the web**.

It occurs when an application **fails to enforce permissions**, allowing attackers to:

- Access other users' data
- Perform actions of higher privilege
- Access admin-only endpoints
- Modify IDs to escalate access
- Bypass authorization checks

This module teaches you:

- How access control works
- How to identify weak implementations
- Tools & commands to automate detection
- Manual exploitation logic
- Reporting & mitigation

---

2. Theory: What Is Access Control?

Access control ensures that **users can only perform actions they're allowed to**.

Three core types:

1. Vertical Access Control

Privileged vs non-privileged

Example:

- User → Should NOT access admin panel
- Admin → Should access admin panel

2. Horizontal Access Control

User → Should only access **their own** data.

Example:

`GET /user/12345/orders`

Changing ID to

`GET /user/12346/orders` → **IDOR**

3. Context-Based Access Control

Permissions change based on state.

Example:

- Verified account vs unverified
- Paid user vs free user

---

3. Attack Surface Checklist

When testing for Broken Access Control, examine:

✔️ Endpoints with user IDs

✔️ Hidden API routes

✔️ Parameter-based access

✔️ Admin panels

✔️ Account takeover flows

✔️ Multi-step workflows (logic flaws)

✔️ File download/upload endpoints

✔️ PUT/DELETE methods

✔️ Cookies & tokens (role stored inside?)

---

4. Manual Pentesting Methodology

Step 1 — Identify User-Based Parameters

Look for:

- `?id=123`
- `/user/12345`
- `/order/view/9811`
- POST body containing `"user_id": 123`

Try to change them.

Step 2 — Privilege Escalation Testing

Vertical Escalation Examples:

1. Access admin only page

```
GET /admin/dashboard
```

If response is **200 OK** → Vulnerable.

2. Hidden admin functionality

```
POST /admin/deleteUser
Content-Type: application/json
{"user_id": 12}
```

If a normal user can delete → Vulnerable.

Step 3 — Horizontal Escalation (IDOR)

Example Request

```
GET /api/v1/users/2001/profile
```

Modify:

```
GET /api/v1/users/2002/profile
```

If data leaks → **IDOR confirmed**.

Step 4 — Method Tampering

Try changing request method:

```
POST → GET
PUT → DELETE
PATCH → POST
```

Example:

```
DELETE /api/v1/users/2001
```

If accepted → Broken access control.

Step 5 — Mass Assignment Attack

If user-supplied JSON is directly mapped to database fields:

Payload:

```
{
  "name": "Atharv",
```

```
  "is_admin": true}
```

If you become admin → Vulnerable.

---

5. Tools & Techniques

Below are tools used by real pentesters.

🔧 Tool 1 — Burp Suite

Purpose:

- Inspect requests
- Modify parameters
- Automate IDOR detection

Key methods:

1. Burp Repeater (Manual Testing)

Used to manipulate IDs, roles, cookies.

2. Burp Intruder (Automation)

Bruteforce IDs:

```
Position 1: /api/user/§1000§
Payload: 1000–2000
```

Analyze responses for:

- Different content length
- Missing "Unauthorized"
- Exposed data

🔧 Tool 2 — Autorize (Burp Extension)

Feature:

Automatically detects access control failures.

How to Use:

1. Login as admin → "Baseline Session"
2. Login as low-privilege → "Attacker Session"
3. Autorize will test every request as low-privileged.

Result:

Any 200 OK response = **broken access control**.

🔧 Tool 3 — X8 / Arjun (Parameter Discovery)

Commands:

```
arjun -u https://target.com/api/profile
```

Finds hidden parameters like:

- `userid`
- `account_id`
- `role`
- `is_admin`

These often break access control.

🔧 Tool 4 — FFUF / Dirsearch (Endpoint Discovery)

Example:

```
ffuf -u https://site.com/admin/FUZZ -w /usr/share/wordlists/admin.txt
```

Admin endpoints found = test for access control.

---

6. Real Payload Examples

1. URL Tampering

```
/account?user_id=10 → /account?user_id=11
```

2. Cookie role modification

```
role=USER → role=ADMIN
```

Try Base64-encoded or JWT.

---

7. JWT Manipulation

If JWT is not verified correctly:

Decode:

```
jwt_tool.py token.jwt
```

Change role:

```
{
  "role": "admin"
}
```

Resign:

```
jwt_tool.py -S -I -pc role -pv admin
```

If accepted → Critical.

---

8. API Testing Commands

Using cURL:

```
curl -H "Authorization: Bearer USER_TOKEN" \
https://api.example.com/admin/stats
```

If returns 200 → Vulnerable.

---

9. Case Study (Real-World)

A fintech platform stored user id in API calls:

```
GET /wallet/transactions?user_id=1001
```

Attacker iterated:

```
1002
1003
1004
```

Exposed:

- Bank details
- Transaction history
- Personal info

Impact: **Massive data breach**

---

10. Mitigation (Developer Perspective)

✔️ Enforce server-side authorization

✔️ Do NOT trust user-input IDs

✔️ Implement object-level permission checks

✔️ Use UUIDs instead of numeric IDs

✔️ Disable client-side role enforcement

✔️ Deny by default

✔️ Implement RBAC/ABAC

✔️ Use secure JWT validation

---

11. Reporting Template

Title: Broken Access Control / IDOR

Severity: Critical

Endpoint: `/api/v1/user/orders?id=102`

Impact: Full personal data exposure

Reproduction Steps:

1. Login as normal user
2. Modify ID parameter
3. View other user's data

Mitigation: Implement object-level authorization

Evidence: Include screenshots, request/response pairs