CYBERSECURITY LOCK

OWASP Top 10: A01:2021 – Broken Access Control

Course: Web Application Security

Goal: Understand the concept, impact, detection, exploitation, and mitigation of Broken Access Control vulnerabilities—the #1 ranked security risk in the OWASP Top 10.

Presentation of the Vulnerability

What Is Broken Access Control?

Access control, also known as authorization, determines what actions authenticated users are permitted to perform within an application. It answers the question: "Now that I know who you are, what are you allowed to do?"

Broken Access Control refers to failures in properly restricting what authenticated users can access or modify. When these controls fail, attackers can exploit the weakness to access data or functions outside their intended permissions—acting as other users, viewing confidential information, modifying data, or performing privileged administrative actions.

OWASP Ranking

A01:2021

Ranked #1 in the 2021 edition of the OWASP Top 10, reflecting its prevalence and severe impact across web applications worldwide.

CIA Impact: Understanding the Damage

Broken Access Control vulnerabilities pose significant risks across the CIA triad—Confidentiality, Integrity, and Availability. Understanding these impacts helps prioritize security efforts and communicate risk to stakeholders.

Impact Area	Level	Explanation
Confidentiality	**High**	Attackers can view sensitive data belonging to other users or the system, including personal information, financial records, trade secrets, and proprietary business data.
Integrity	**High**	Attackers can modify or delete data they shouldn't have access to, potentially corrupting critical business information, manipulating transactions, or sabotaging system functionality.
Availability	Low	Rarely impacts system uptime directly, but in extreme cases—such as deleting critical resources or administrative accounts—could lead to service disruption or denial of service.

Deep Dive: Understanding Access Control

Authorization vs. Authentication

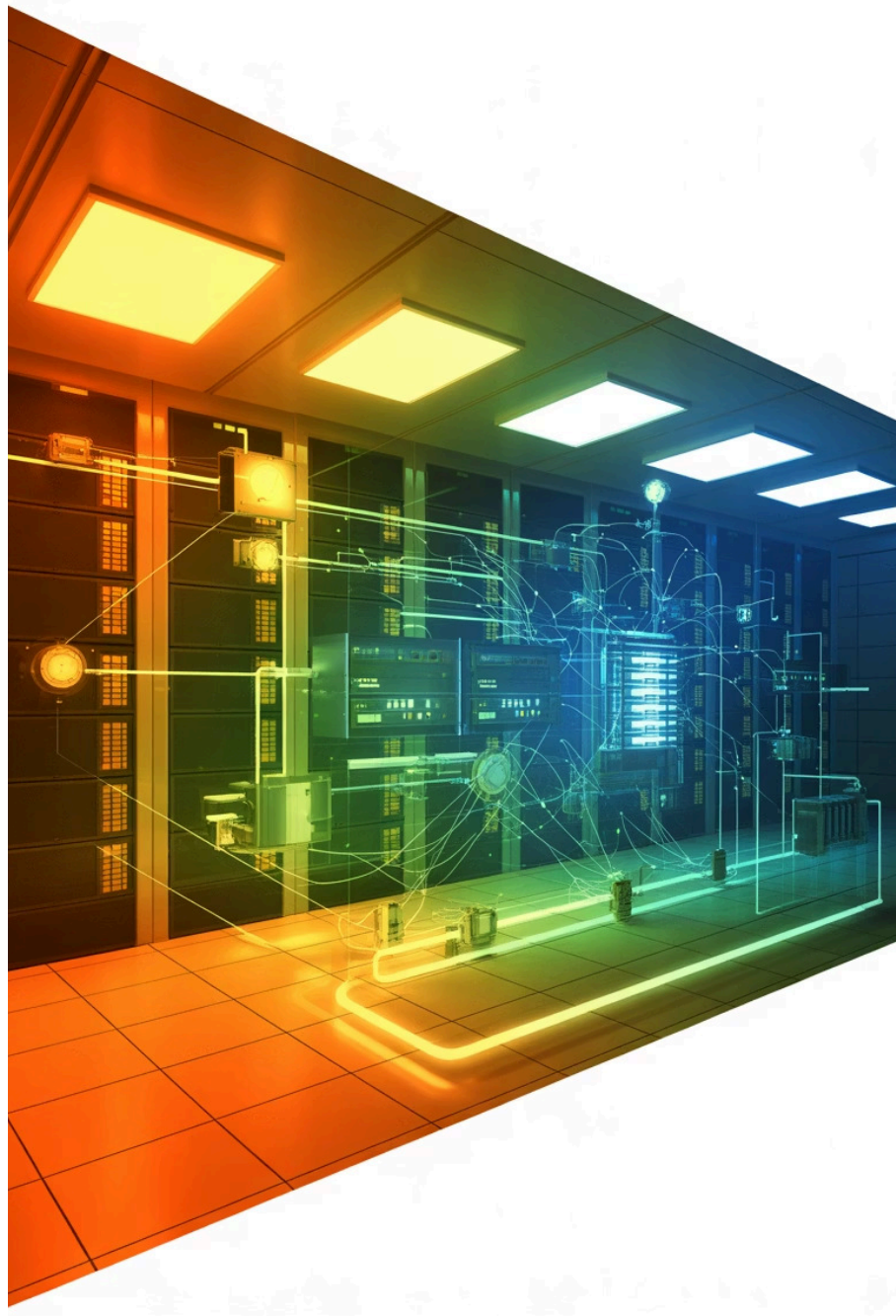
This is about *authorization*, not authentication. It's not about proving who you are (logging in), but about controlling what you can do *after* you've logged in.

Attack Sources

Threats come from both internal sources (malicious insiders, overprivileged users) and external attackers who have gained legitimate credentials.

Attack Targets

Vulnerable components include backend APIs and controllers (primary enforcement points), frontend logic, database queries, and misconfigured infrastructure like Firebase.



Types of Access Control Failures

Access control vulnerabilities manifest in several distinct patterns. Understanding these categories helps security professionals identify and remediate weaknesses systematically.

1

Vertical Privilege Escalation

A regular user gains administrative privileges, allowing them to perform system-wide operations like creating accounts, modifying configurations, or accessing all user data.

2

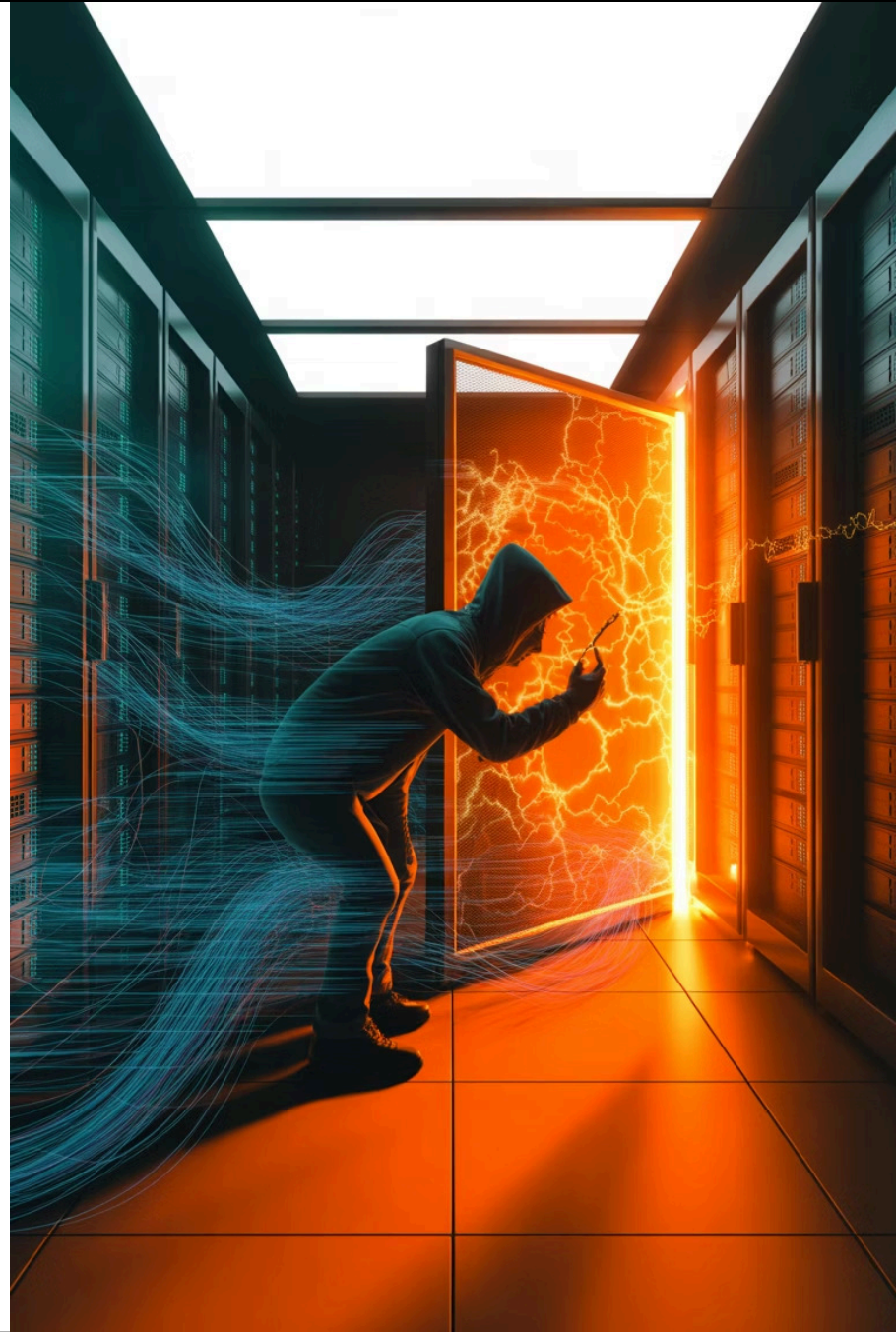
Horizontal Privilege Escalation

A user accesses data or functions belonging to another user at the same privilege level—for example, viewing another customer's orders or personal information.

3

Context-Dependent Access

Access depends on resource ownership or workflow state. For instance, you might edit your own profile but not others', or access a document only during specific approval stages.



Example Scenario 1: Insecure Direct Object Reference (IDOR)

The Vulnerability

An e-commerce website allows users to view their invoices via URLs like `/invoice/12345`. The server authenticates the user but fails to verify whether they actually own invoice 12345.

An attacker simply changes the URL to `/invoice/12346` and instantly accesses another customer's invoice—complete with personal details, purchase history, and payment information.

Why It Happens

The server relies only on the presence of a valid session or authentication token. It never checks *ownership* of each specific resource being requested.

Example Scenario 2: Forced Browsing & Privilege Escalation

Hidden But Not Protected

A web application has an administrative panel at `/admin`. The developers hide links to admin features from regular users' interfaces, but they never implemented server-side checks to enforce this restriction.

A normal user manually types `/admin` into their browser and gains full access—simply because there's no backend verification preventing them from viewing the page.

Role Manipulation

Modifying a POST request body to change `"role":"user"` to `"role":"admin"`

Unlisted Endpoints

Accessing hidden API functions like `/deleteUser?id=42` that aren't linked in the UI

Directory Traversal

Discovering sensitive paths through guessing or automated scanning tools

Discovering the Vulnerability: Where to Look

Identifying access control vulnerabilities requires systematic investigation across multiple attack surfaces. Security professionals should examine these key areas during penetration testing and code review.

URLs and Endpoints

Any URL containing user-supplied IDs or resource identifiers is a potential target. Look for predictable patterns like sequential numbers or GUIDs.

Forms and Hidden Fields

Hidden input fields often store resource IDs, user roles, or permission flags that can be manipulated through browser developer tools.

HTTP Headers

Custom headers may contain privilege information or authorization tokens that can be tampered with or replayed to gain unauthorized access.

Client-Side Code

JavaScript files frequently reveal API endpoints, access control logic, or administrative features intended to be hidden from regular users.

API Documentation

Swagger, OpenAPI, or other API documentation files may list all available endpoints, including those not exposed through the normal user interface.

Cookies & Internal Storage

Unsecure code can use user modifiable cookies or internal storage datas to transmit user identity informations.

Recognizing Warning Signs

Certain indicators strongly suggest the presence of access control vulnerabilities. Train your security awareness to recognize these red flags during testing.

ID Manipulation Works

Changing an ID parameter in the URL lets you access someone else's data without triggering an error or access denial.

Tamperable Privilege Data

"Role" or "permission" values appear in requests (cookies, form fields, JSON bodies) and can be modified by the client.

Client-Side Restrictions Only

Administrative features are hidden in the UI through JavaScript or CSS, but direct requests to those endpoints succeed.

Inconsistent Access Denial

"Access Denied" or "Unauthorized" messages are missing, inconsistent, or only appear for some resources but not others.

Direct Action Without Authentication

Sensitive operations (delete, modify, approve) can be performed via direct API requests, bypassing intended workflows and Authentication processes.

Verification Techniques

Manual Testing Methods

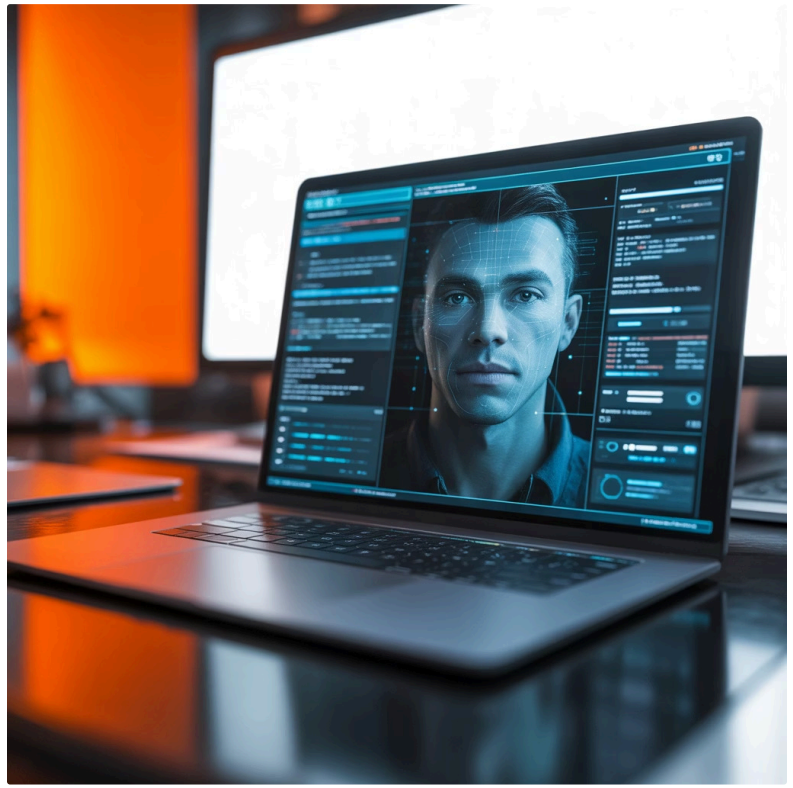
- **ID Enumeration:** Try incrementing or decrementing ID values in URLs, query parameters, or request bodies
- **Direct Navigation:** Attempt accessing endpoints or pages not exposed through the normal user interface
- **Parameter Manipulation:** Modify role fields, permission flags, or user identifiers in requests
- **Session Testing:** Test access with different user accounts or privilege levels to identify inconsistencies

Automated Testing

- **Fuzzing:** Use automated tools to systematically test parameter variations and boundary conditions
- **Proxy Interception:** Capture and modify requests using Burp Suite or OWASP ZAP to test authorization logic

Code Review

- Search for missing authorization checks in controllers, API endpoints, and middleware
- Verify that resource ownership is confirmed on every request, not just during initial page load
- Look for inconsistent access control patterns across different parts of the application



Essential Testing Tools

Professional security testing requires the right tools. These industry-standard applications help identify and exploit access control vulnerabilities effectively.

Burp Suite

Industry-leading proxy tool for intercepting, modifying, and replaying HTTP requests. The Intruder feature automates parameter fuzzing and testing of sequential ID values.

OWASP ZAP

Free, open-source alternative to Burp Suite with excellent request interception capabilities and automated scanning features for access control testing.

Postman

Powerful API testing platform that allows manual endpoint testing, request scripting, and automated test suite creation for authorization verification.

Fuzzing Tools

Specialized tools like wfuzz and ffuf automate the process of brute-forcing ID values, discovering hidden endpoints, and testing parameter variations at scale.

Exploitation Principles

Understanding how attackers exploit access control vulnerabilities helps defenders build more robust protections. These core principles underpin most real-world attacks.

ID Manipulation

Systematically changing resource identifiers in URLs, cookies, or request bodies to access data belonging to other users or restricted resources.

Direct API Calls

Bypassing client-side restrictions by calling backend endpoints directly through tools like curl, Postman, or custom scripts—ignoring UI-based controls entirely.

Parameter Tampering

Altering form fields, query strings, HTTP headers, or JSON payloads to escalate privileges, change user roles, or access unauthorized features.

Request Replay and Reuse

Capturing legitimate requests and replaying them with modified parameters, or reusing authorization tokens in unintended contexts to perform unauthorized actions.

Practical Exploitation Demo

Lab Example: Horizontal Privilege Escalation

Target Application: `/user/profile/123`

You're logged in as Alice (user ID 123). The application authenticates you successfully and shows your profile. However, the backend only checks that you have a *valid login session*—it never verifies that you own the requested profile.

Exploitation Steps

1. Log in as Alice (alice:password) and note your profile URL: `/user/profile/123`
2. Change the URL to Bob's profile: `/user/profile/124`
3. Observe full access to Bob's personal information, preferences, and account details
4. Test editing or deleting Bob's data by intercepting and modifying POST requests
5. Enumerate other users by systematically testing ID values: 125, 126, 127...

Key Insight

This exploitation requires **no special tools** or advanced hacking skills. A curious user with a web browser can discover and exploit the vulnerability in minutes.

The simplicity of the attack makes it especially dangerous—and unfortunately common in real-world applications.

Why These Vulnerabilities Exist: Root Causes

Understanding why access control failures occur helps development teams prevent them. These fundamental issues appear repeatedly across vulnerable applications.

Missing or Incomplete Authorization Checks

Developers implement authentication to verify user identity but forget to add authorization checks that verify resource ownership or permission for specific actions. This is the most common root cause.

Trusting the Client

Applications rely on client-side controls (hidden buttons, disabled fields, JavaScript validation) to prevent unauthorized actions, assuming users won't manipulate requests directly.

Broken Access Control Logic

Flawed conditional statements like `if (user.isAdmin || user.id == req.user.id)` create unintended bypass conditions or fail to account for all scenarios.

Insecure Defaults

Frameworks and APIs default to open access, requiring explicit configuration to restrict resources. Developers often forget to apply these restrictions, leaving endpoints publicly accessible.

High-Risk Features and Patterns

Certain application features inherently carry higher access control risk. Security architects should apply extra scrutiny to these areas during design and implementation.

Direct Object References

URLs or parameters that directly expose database IDs, filenames, or internal identifiers without abstraction or indirection layers

Role Management via UI

Frontend interfaces that allow users to select, modify, or view their own role or permission settings without rigorous backend validation

Bulk Data Access APIs

Endpoints that return multiple records or entire datasets without proper filtering based on the requesting user's permissions

Update & Delete operations

Update and delete operations require to check that a user is both identified and the owner of the resource he want to modify.

Defensive Strategy: Core Principles

Effective access control requires a holistic defensive approach built on these foundational security principles. Every secure application must implement these concepts consistently.

Deny by Default

Explicitly refuse all access unless specifically permitted. Never grant access based on the absence of a denial—always require positive authorization.

Server-Side Enforcement Only

Never trust client-side controls, hidden fields, or JavaScript for access decisions. All authorization must occur on the server where attackers cannot manipulate it.

Consistent Authorization Logic

Centralize access control checks in middleware, filters, or dedicated security layers. Avoid scattering authorization code throughout the application.

Use Secure Frameworks

Leverage well-tested, industry-standard access control libraries and frameworks rather than building custom authorization systems from scratch.

Concrete Fix: Secure Implementation

Implementing Proper Authorization Checks

On every request that accesses user-specific resources, you must verify:

1. Is the user authenticated (logged in)?
2. Is the user authorized to access this specific resource?
3. Does the user own the requested object (profile, file, record)?
4. Does the user's role permit this action (view, edit, delete)?

Critical principle: Never rely solely on hiding features in the user interface. Always enforce access control on the server side.

Example Fix (Python/Flask)

```
@app.route('/user/notes/<int:note_id>')
@login_required
def user_note(note_id):
    note = get_note(note_id)

    # Check resource ownership
    if user_id != note.owner_id:
        abort(403)  # Forbidden

    # Retrieve and return profile
    return render_template('note.html',
                           note=note)
```

This code explicitly verifies that the authenticated user can only access their own notes, returning a 403 Forbidden error for unauthorized attempts.

Hands-On Practice: CTF Challenge

Account Takeover via Broken Access Control

Challenge Scenario

You're a security researcher evaluating a fintech startup's web application. During initial reconnaissance, you suspect that users might be able to view each other's account balances and transaction histories due to insufficient access controls.

Your Mission

1. Exploit an IDOR vulnerability to access another user's account information
2. Document your exploitation methodology and findings
3. Get bank account number of Bernard and use it as CTF key
4. Propose a concrete patch to prevent this vulnerability
5. Submit your fix as a GitHub pull request with detailed explanation

Challenge Details

URL: https://bac.owasp.ajani.ovh/

Test Accounts:

- alice / password

Expected Duration: 90 minutes

Reminder: First exploit the vulnerability to understand its impact, then propose a proper server-side fix that addresses the root cause.

Exploitation Demonstration

Live Walkthrough: Step-by-Step Attack

This demonstration shows the complete exploitation process, from initial discovery through successful unauthorized access. Follow along to understand both the attacker's perspective and the defensive measures needed.

Initial Reconnaissance

Log in as Alice and explore the application. Note URL patterns, particularly those containing user IDs or resource identifiers.

Identify Target

Navigate to Alice's account page. Observe the URL structure: `/account/balance?user_id=123`

Attempt Exploitation

Modify the user_id parameter to 124 (Bob's ID). Observe whether you can access Bob's account balance and transaction history.

Validate Access

Confirm unauthorized access by verifying you can view information that should be restricted to Bob's account only.

Document and Report

Create a detailed writeup including: vulnerability description, exploitation steps, proof of unauthorized access, and proposed server-side fix with code examples.

Knowledge Check: Quiz

Test your understanding of Broken Access Control concepts with these questions. Consider each carefully—they cover key principles discussed throughout this presentation.

https://qcm.ajani.ovh/owasp-bac

PS : Use ChatGPT and see how quick I will show you the door 🚪 ...

Key Takeaways

What It Is

Broken Access Control occurs when users can access resources or perform actions beyond their authorized privileges due to missing or improperly implemented authorization checks.

When It Appears

These vulnerabilities emerge when authorization is not enforced on the server side, when applications rely on client-side controls, or when access checks are inconsistently applied across the application.

How to Protect

Always enforce access controls server-side. Use the principle of least privilege. Centralize authorization logic. Deny by default. Regularly review and test access control mechanisms through code review and penetration testing.

Remember: Access control is not a one-time implementation—it requires continuous vigilance, testing, and refinement throughout the application lifecycle. Make security reviews a standard part of your development process.

Always ask yourself: Does the user have the right to do this ?