# Broken Access Control: The Web's #1 Vulnerability

## Understanding and Identifying Flaws in "Who Can Do What"

A critical security flaw that allows unauthorized users to access data and functionality they shouldn't touch.

# What is Access Control?

## The Digital Bouncer

Access control is the set of rules and policies that enforce **who** (authentication) is allowed to do **what** (authorization) within an application.

Think of it as the security system that determines which doors you can open and which actions you can perform once inside.

**Hotel Analogy:** Your room key authenticates you and authorizes access to your room (303) and the gym, but not the manager's office or Room 304.

# What is *Broken* Access Control?

## When the Bouncer Isn't Checking IDs

### The Vulnerability

Rules aren't enforced correctly, allowing users to access data or perform actions they shouldn't be authorized for.

### Critical Threat

The **#1 most critical web vulnerability** on the OWASP Top 10 for several years running.

# Type 1: Insecure Direct Object References (IDOR)

## The Most Common Example

IDOR occurs when an application exposes a direct reference to an internal object—like a user ID, file ID, or order number—in the URL or API request.

### The Flaw

The server doesn't verify if the *authenticated user* actually *owns* the object they're requesting.

### The Risk

Attackers can simply change IDs in URLs to access other users' private data, documents, or account information.

# IDOR: Practical Example

## Changing the Number in the URL

### Legitimate User (Alice)

Logs in and views her profile:

```
https://example.com/profile?user_id=123
```

### Attacker (Eve)

Logs in, sees her URL is `user_id=456`, then changes it to:

```
https://example.com/profile?user_id=123
```

### Result: Data Breach

The vulnerable server shows Alice's private information to Eve because it only checked authentication, not authorization.

# Type 2: Vertical Privilege Escalation

## "I'm Not an Admin, But I'll Act Like One"

This vulnerability allows a user with low privileges (standard user) to access functions or data reserved for high-privilege users (administrators).

> Developers often "hide" links to admin pages in the UI but forget to secure the pages themselves on the server side.

# Vertical Privilege Escalation: Example

## Guessing the Admin URL

An attacker logged in as a normal user can't see the "Admin Panel" link in the interface.

### Attacker's Action

They simply try guessing URLs by typing directly into the browser:

- `https://example.com/admin`
- `https://example.com/admin_panel`
- `https://example.com/api/v1/users/delete?id=123`

**Result:** The server grants access to the admin page, allowing the attacker to delete users or change critical settings.

# Type 3: Client-Side Response Manipulation

## Trusting the Client to Keep a Secret

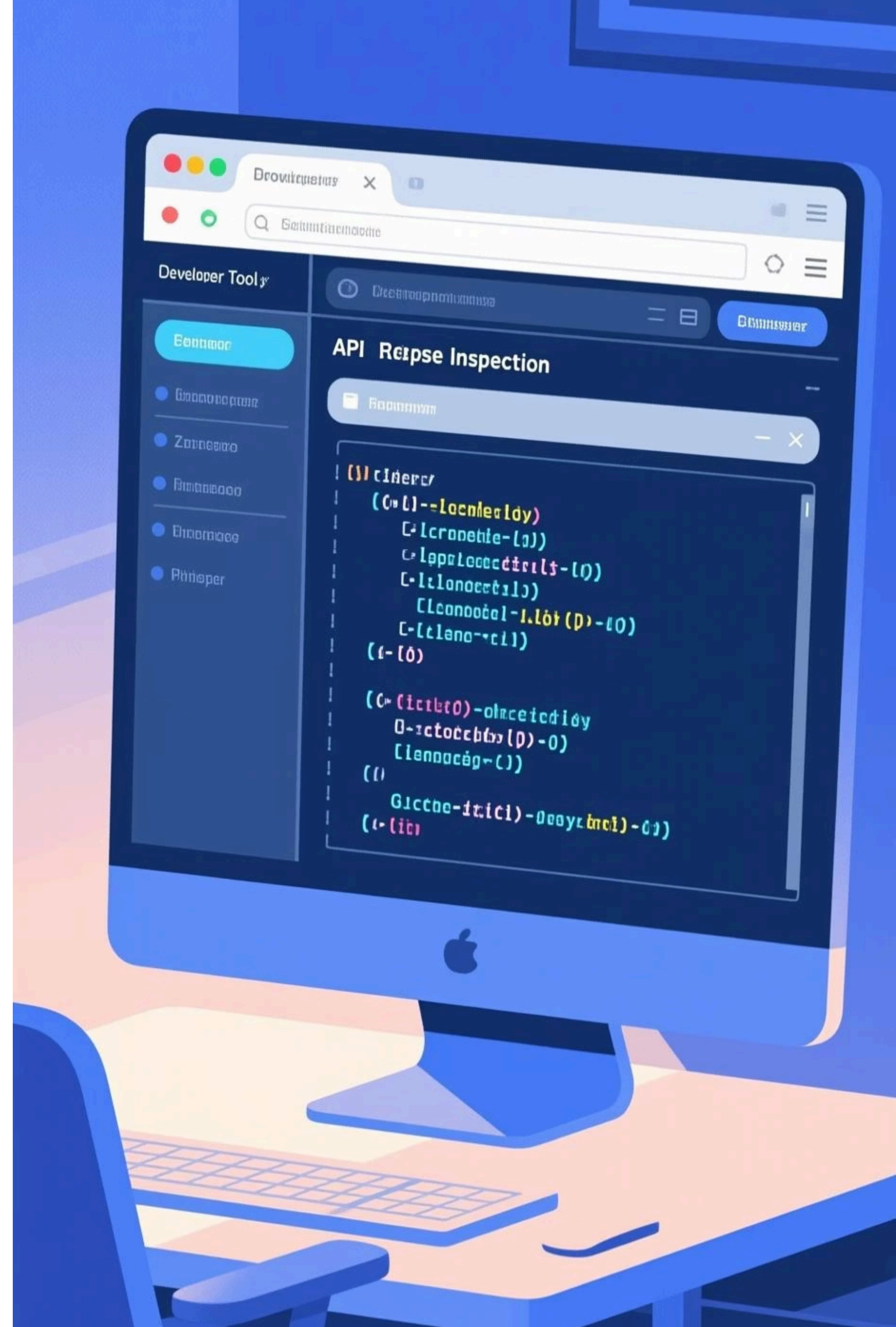
This flaw occurs when the server sends a *complete* data object (including sensitive information) to the client, then relies on client-side JavaScript to hide parts the user shouldn't see.

### Server Sends This JSON:

```
{
  "username": "bob",
  "is_admin": false,
  "address": "123 Main St",
  "admin_secret_key": "xyz789"
}
```

### The Critical Flaw

The UI hides `admin_secret_key`, but attackers using browser DevTools or Burp Suite can easily read the raw API response and steal the key.



# How to Test for Broken Access Control

## A Tester's Mindset

01

### Change All IDs

In URLs, API requests, and form bodies, change every ID you see (user, order, invoice, file) to other valid-looking numbers.

02

### Login as Two Users

Use two different browsers. Log in as User A in one and User B in the other. Try to request User A's data using User B's session.

03

### Browse as User, Attack as Admin

Log in as a normal user and map the site. Then try to directly browse to URLs you guess an admin would have.

04

### Inspect All API Responses

Look for hidden data in the JSON or HTML source code that shouldn't be sent to your privilege level.

# Prevention: The Golden Rule

## How to Fix It: Deny by Default

### Server-Side Checks are Mandatory

Never, ever rely on client-side code (JavaScript, CSS) to hide sensitive data or restrict access.

### The Golden Rule

On **every single request**, the server must verify that the *authenticated user* has the *required role* for the *requested action* on the *requested object*.

### Use Frameworks

Leverage the built-in authorization middleware of modern web frameworks rather than building your own from scratch.

### Example Check:

```
if (currentUser.id == requestedUser.id ||
 currentUser.isAdmin()) {
 // Allow access
}
```