Access control Notes

Eyad Islam El-Taher

November 17, 2025

Introduction

Access control $\Longrightarrow$ the application of constraints on who or what is authorized to perform actions or access resources. In the context of web applications, access control is dependent on authentication and session management:

- **Authentication:** Who are you? (Logging in)
- **Session management:** identifies which subsequent HTTP requests are being made by that same user $\Longrightarrow$ Keeping you logged in.
- **Access control:** What are you allowed to do? (Permissions)

Access control vulnerabilities happen when

an application fails to properly enforce restrictions on what a user is authorized to do, allowing a user to act outside of their intended permissions.

Common Access Control Flaws:

- **Missing Server Checks:** Relying on hiding buttons in the browser. Attackers can just call the function directly if the server doesn't verify permissions.
- **Insecure Object References (IDOR):** The app uses user-provided input (like a filename or user ID in the URL) to access data directly, without checking if the user is allowed to see it.
- **Excessive Permissions:** Users are given more power than they need. If an attacker hijacks such an account, they can do more damage.
- **System Misconfigurations:** Mistakes in the server or framework settings can accidentally create holes that bypass security.
- **Predictable Identifiers:** Using easy-to-guess IDs (like 1, 2, 3...) or storing privileges in insecure locations (like browser cookies) that an attacker can easily forge.

Access control vulnerabilities happen where

- **Authentication and Authorization Mechanisms** The core flaws often stem from incorrectly configured authentication and authorization systems, such as failing to verify the user's identity or role for every request.
- **Server-Side Code** When developers rely solely on client-side (e.g., UI elements like hiding links/buttons) rather than robust, server-side checks, attackers can easily bypass controls by sending direct requests to restricted URLs or API endpoints.
- **URL and API Endpoints** Vulnerabilities are frequently found in the parameters of URLs or API calls, where an attacker can manipulate values (like a user ID or invoice ID) to access another user's account or data (Insecure Direct Object References or IDOR).
- **Session Management systems** Weak session management can lead to access control issues, for instance, if session identifiers don't expire properly after logout or are exposed in the URL, allowing an attacker to hijack a session.

- **Platform configurations** Platform Misconfigurations: Incorrectly set up server or platform rules (e.g., a "debug" mode left enabled, or discrepancies in how different web server components interpret URLs) can create bypass opportunities.
- **Multi-Step Processes** In multi-stage processes (like an e-commerce checkout or an admin function with multiple confirmation steps), developers sometimes enforce access control on initial steps but neglect later ones, assuming a user would only reach the end with proper authorization.

Impact of Access Control Vulnerabilities

Access control vulnerabilities can have significant negative effects on both users and applications, primarily through unauthorized access to sensitive data and system functionality.

Effects on the User

- **Exposure of Sensitive Data:** Attackers can access users' private information (PII, financial records, health data), leading to identity theft or fraud.
- **Unauthorized Actions:** Attackers may perform actions as the user (changing passwords, making purchases, deleting data).
- **Privacy Violations:** User's online activities and personal data can be exposed, causing privacy breaches and reputational damage.
- **Loss of Trust and Security:** Users lose faith in the application's ability to protect their information.

Effects on the Application

- **Data Breaches:** Exposure of large data amounts leading to legal and regulatory consequences.
- **System Integrity Compromise:** Attackers can modify or delete data, causing incorrect information and service disruption.
- **Reputational Damage:** Security breaches damage the organization's reputation, leading to customer loss.
- **Financial Loss:** Costs from breach investigation, user notifications, fines, legal fees, and remediation.
- **Service Disruption:** Attackers may disrupt normal operations, causing downtime and availability issues.

Types of Access Control

Vertical Access Controls

- **Simple Idea:** Restricts access based on **user roles** or privilege levels.
- **Analogy:** The hierarchy in a company.
- **Example:** An **Administrator** can delete any user's account, but a **regular User** cannot. It's about what you are *allowed to do*.

Horizontal Access Controls

- **Simple Idea:** Restricts access to a **subset of data** within the same user level.
- **Analogy:** Personal lockers in a school; everyone has one, but you can only open your own.
- **Example:** In a bank app, you can only view your own accounts, not those of other customers. It's about *which data* you can see.

Context-Dependent Access Controls

- **Simple Idea:** Restricts actions based on the **current situation** or application state.
- **Analogy:** Not being allowed to change your order after the chef has started cooking it.
- **Example:** A website prevents you from modifying your shopping cart **after** you've clicked the "Pay" button.

Examples of Broken Access Control

1- Unprotected Functionality

- **What it is:** The application has sensitive functions but does not lock the "door" to them.
- **How it happens:**
  - An admin page exists at a predictable URL (e.g., `/admin`).
  - The link is hidden from regular users on the webpage, but the page itself has no password or check.
  - Anyone who knows or guesses the URL can access it.
- **Discovery:** Attackers can find these hidden URLs in files like `robots.txt` or by guessing common names.

robots.txt $\Longrightarrow$ a text file that website owners create to give instructions to web crawlers (like search engine bots) about which pages or directories they should not crawl.

Security researchers frequently discover hidden administrative panels and debug URLs through `robots.txt` analysis.

Read this Writeup plz $\Longrightarrow$ • Infosec Writeup Article

2- The Flaw of "Security by Obscurity"

The Concept

- Sensitive functionality (e.g., an admin panel) is hidden at a **less predictable URL**.
- This is known as "**security by obscurity**"—relying on secrecy rather than proper authentication.
- This is **not effective access control** because the hidden URL can be discovered in multiple ways.

Example Scenario

- An application hosts its admin panel at a non-obvious URL:

  `https://insecure-website.com/administrator-panel-yb556`

- While not easily guessable, this URL can be **leaked** to users through various channels.

Common Leakage Vectors The hidden URL might be disclosed in:

- **JavaScript Files:** Client-side code that dynamically builds the UI based on user roles may contain the URLs, even if they are not displayed.
- **HTML Comments:** Developers might leave commented-out code or notes in the page source.
- **API Responses:** Backend endpoints might return the privileged URLs in their responses.
- **Sitemap.xml / Robots.txt:** As previously discussed, these files can accidentally expose paths.

3- Insecure Parameter-Based Access Control

The Flawed Method

- The application decides what a user is allowed to do during login.
- It then stores this permission level in a location that the **hacker can see and control like URL**.
- The application trusts this user-controlled value for all future access checks.

Where is this Data Stored?

- A **hidden field** in an HTML form
- A **cookie** in the browser
- A **request body** in Burp Suite
- A **query parameter** in the URL

Insecure Examples

```
https://insecure-website.com/home?admin=true
https://insecure-website.com/home?role=1
```

4- Horizontal Privilege Escalation

A user can access **another user's data** of the same type, instead of being restricted to their own.

- A user views their profile page at:

  `https://insecure-website.com/myaccount?id=123`

- If they change the URL to `id=124`, they can access another user's account page, data, and functions.
- This is a classic **Insecure Direct Object Reference (IDOR)** vulnerability.

Insecure Direct Object References (IDOR)

Simple Definition

- A subcategory of **access control vulnerabilities**.
- Occurs when an application uses **user-supplied input** (like an ID in a URL) to access objects (like files or database records) **directly**.
- An attacker can simply **modify this input** to gain unauthorized access to other users' data.

The Core Problem

The application trusts user-provided identifiers without verifying if the user has permission to access the specific object they are requesting.

Simple Example

```
# A user accesses their own file:
https://example.com/view_document?doc_id=1001

# An attacker changes the ID to access another file:
https://example.com/view_document?doc_id=1002
```

Advanced IDOR Scenarios

1- Non-Predictable Identifiers (GUIDs)

- Applications may use **Globally Unique Identifiers (GUIDs)** instead of simple numbers.
- Example: `/view_profile?user_id=9b67d2d4-5a7a-4c8b-ba3c-1f5e8d9c0a2b`
- This prevents **guessing or predicting** other users' identifiers through sequence.

Where GUIDs Can Be Leaked

However, these GUIDs might be disclosed elsewhere in the application:

- User messages or chat logs
- Product reviews or comments
- Shared documents or file metadata
- Anywhere users are referenced in the application

2- The Redirect Vulnerability

- Some applications **do detect** unauthorized access attempts.
- They respond with a **redirect to the login page**.
- **However:** The initial response containing the redirect might still **include sensitive data** from the targeted user's account in the response body.
- **Result:** The attack is still successful even with the redirect.

3- Horizontal to Vertical Privilege Escalation

The Attack Chain

- A **horizontal** privilege escalation attack is used to compromise a **higher-privileged user's account**.
- Attacker specifically targets an **administrative user's account**.
- This effectively turns the horizontal attack into a **vertical** privilege escalation.

Impact

- **Doubles the damage:** Combines data theft (horizontal) with system compromise (vertical).
- **Critical severity:** A simple IDOR can lead to complete system takeover if admin accounts are accessible.

4- URL-Based Access Control Circumvented

The Vulnerability

Some applications enforce access control at the platform layer (e.g., a web server or WAF) by blocking access to specific URLs based on user roles. However, if the application supports non-standard HTTP headers like `X-Original-URL` or `X-Rewrite-URL`, attackers can bypass these front-end controls by providing a different URL in the header that the back-end processes instead.

Attack Example: Bypassing Admin Panel Restriction

1. **Initial Blocked Request:**

```
GET /admin HTTP/1.1
Host: vulnerable-website.com
```

Response: 403 Forbidden (blocked by front-end control)

2. **Bypass Using Header Override:**

```
POST / HTTP/1.1
Host: vulnerable-website.com
X-Original-URL: /admin
```

Response: 200 OK - Admin panel loaded (back-end processed the header)

3. **Deleting a User:**

```
POST /?username=carlos HTTP/1.1
Host: vulnerable-website.com
X-Original-URL: /admin/delete
```

Response: User 'carlos' successfully deleted

Why This Works

- **Front-end:** Blocks requests to `/admin/*` URLs
- **Back-end:** Processes the URL from `X-Original-URL` header instead of the actual request line
- **Result:** Front-end controls are completely bypassed

5- Method-Based Access Control Circumvented

The Vulnerability

Some applications enforce access control based on both the URL **and HTTP method** (e.g., blocking `POST /admin/promote`). However, if the application accepts alternative HTTP methods to perform the same action, attackers can bypass these controls by using permitted methods to access restricted functionality.

Attack Example: Promoting Users via Method Manipulation

1. **Admin's Original Request (Blocked for Non-Admins):**

```
POST /admin/promote HTTP/1.1
Host: vulnerable-website.com
Cookie: session=admin_session_token

username=carlos&action=promote
```

Response: 200 OK - User promoted

2. **Non-Admin Attempt (Blocked):**

```
POST /admin/promote HTTP/1.1
Host: vulnerable-website.com
Cookie: session=user_session_token

username=carlos&action=promote
```

Response: 401 Unauthorized

3. **Method Manipulation Bypass:**

```
GET /admin/promote?username=carlos&action=promote HTTP/1.1
Host: vulnerable-website.com
Cookie: session=user_session_token
```

Response: 200 OK - User successfully promoted

Why This Works

- **Front-end Controls:** Block `POST` requests to `/admin/promote` for non-admin users
- **Back-end Logic:** Processes both `POST` and `GET` methods for the same functionality
- **Vulnerability:** Front-end doesn't block `GET` requests to the same endpoint
- **Result:** Access controls bypassed by changing HTTP method

Alternative Attack Vectors

- Changing `POST` to `POSTX` to test parser tolerance
- Using `PUT`, `PATCH`, or custom HTTP methods
- Parameter shifting from body to URL parameters

6- Referer-Based Access Control Bypass

The Vulnerability

Some applications implement access control checks based on the `Referer` HTTP header, which indicates the previous page the user visited. This is insecure because attackers can easily manipulate this header to bypass authorization checks.

How It Works

- Application checks if `Referer` header contains authorized page (e.g., `/admin`)
- If `Referer` matches, request is allowed regardless of user permissions
- Attackers can forge the `Referer` header to bypass these checks

Attack Example: Bypassing Admin Function Protection

1. **Legitimate Admin Request:**

```
POST /admin/deleteUser HTTP/1.1
Host: vulnerable-website.com
Referer: https://vulnerable-website.com/admin
Cookie: session=admin_session

username=carlos
```

Response: 200 OK - User deleted

2. **Non-Admin Without Referer (Blocked):**

```
POST /admin/deleteUser HTTP/1.1
Host: vulnerable-website.com
Cookie: session=user_session

username=carlos
```

Response: 403 Forbidden - Missing Referer

3. **Non-Admin With Forged Referer (Bypass):**

```
POST /admin/deleteUser HTTP/1.1
Host: vulnerable-website.com
Referer: https://vulnerable-website.com/admin
Cookie: session=user_session

username=carlos
```

Response: 200 OK - User deleted (bypassed!)

Why This is Insecure

- **Client-Controlled:** `Referer` header is fully controllable by the client
- **No Authentication:** Checks page origin instead of user permissions
- **Easy Bypass:** Simply add the required `Referer` header to any request
- **False Security:** Creates illusion of protection while being easily bypassable

7- Multi-Step Process Access Control Bypass

The Vulnerability

When applications implement multi-step processes, they often enforce access controls on initial steps but neglect final confirmation steps, assuming users can only reach them through the proper workflow. Attackers can bypass these controls by directly accessing unprotected final steps.

Common Scenario

- **Step 1:** Load user details form (*Access controlled*)
- **Step 2:** Submit changes (*Access controlled*)
- **Step 3:** Review and confirm changes (*No access control*)
- **Flaw:** Assumption that users reach Step 3 only after completing controlled steps

Attack Example: User Promotion Bypass

1. **Step 1 - Admin Loads Form (Controlled):**

   ```
   GET /admin/load-user-form?user=carlos HTTP/1.1
   Host: vulnerable-website.com
   Cookie: session=admin_session
   ```

   Response: 200 OK - Form loaded

2. **Step 2 - Admin Submits Changes (Controlled):**

   ```
   POST /admin/submit-changes HTTP/1.1
   Host: vulnerable-website.com
   Cookie: session=admin_session

   user=carlos&role=admin
   ```

   Response: 200 OK - Changes submitted

3. **Step 3 - Final Confirmation (Unprotected):**

   ```
   POST /admin/confirm-changes HTTP/1.1
   Host: vulnerable-website.com
   Cookie: session=user_session

   user=carlos&role=admin&confirm=true
   ```

   Response: 200 OK - Changes confirmed (bypassed!)

Why This Works

- **Inconsistent Controls:** Access checks applied unevenly across workflow
- **Faulty Assumption:** Trusts that users follow intended sequence
- **Direct Access:** Attackers can skip to unprotected final step
- **Parameter Reuse:** Same parameters work across all steps

Remediation and Prevention

Fundamental Principles

1. Principle of Least Privilege

- Users should only have the minimum permissions necessary to perform their tasks
- Regularly review and audit user privileges
- Implement role-based access control (RBAC) with clearly defined permissions

2. Never Trust Client-Side Controls

- **All access control decisions must be made server-side**
- Never rely on hidden fields, cookies, or client-side JavaScript for authorization
- Validate permissions for every request, regardless of the user interface

3. Deny by Default

- Default to denying access unless explicitly authorized
- Implement fail-secure mechanisms that block access when errors occur

Technical Implementation Guidelines

Authentication & Session Management

- Implement strong, multi-factor authentication for sensitive operations
- Use secure, random session identifiers that expire properly
- Invalidate sessions after logout, password changes, and periods of inactivity

Authorization Checks

- Use a centralized access control mechanism
- Apply the same authorization logic consistently across all endpoints
- Verify permissions for both the resource AND the action being performed

IDOR Prevention

- Use indirect reference maps instead of direct object references
- Implement proper authorization checks for each object access
- Consider using UUIDs instead of sequential IDs, but don't rely on obscurity

API & Endpoint Security

- Document all API endpoints and their required permissions
- Use standardized authentication methods (OAuth 2.0, JWT)
- Implement rate limiting and monitor for suspicious access patterns

Multi-Step Process Security

- Maintain server-side state for workflows
- Validate completion of previous steps before allowing progression
- Use unique, single-use tokens for each step