This lab is a perfect example of Information Disclosure leading to Broken Access Control. In the industry, we call this "Security through Obscurity," and it is a common finding in SOC reports.

I have restructured your notes into a high-impact, professional Markdown report. This version emphasizes your ability to perform Static Analysis (reading code) and highlights the critical difference between Client-Side and Server-Side security.

[Lab Report] Vertical Privilege Escalation via Client-Side Information Disclosure

Project Information

- **Analyst: Mduduzi William Radebe**
- **Date: 07 January 2026**
- **Platform: PortSwigger Web Security Academy**
- **Vulnerability Type: CWE-200: Exposure of Sensitive Information to an Unauthorized Actor**
- **Severity: High (7.5/10)**

1. Executive Summary

This report documents a successful Vertical Privilege Escalation exploit. The target application attempted to protect administrative functionality by using an "unpredictable" (obfuscated) URL. However, this sensitive path was leaked within the application's client-side JavaScript. By performing static source code analysis, I discovered the hidden endpoint and accessed the administrative panel without authentication, eventually performing unauthorized user management actions.

2. Technical Analysis & Discovery

The "Security by Obscurity" Flaw

The developers implemented a "hidden" URL to manage the application, assuming that if the URL was random (unpredictable), it would be secure from attackers. This is a fundamental security failure as it relies on a secret path rather than a robust authentication mechanism.

Static Analysis of Client-Side Code

During the reconnaissance phase, I performed a manual review of the HTML source code. I identified a JavaScript block that contained logic intended only for administrators.

Leaked JavaScript Logic:

JavaScript

```
if (isAdmin) {

  adminPanelTag.setAttribute('href', '/admin-ygn04o');

}
```

Findings:

- **Client-Side Vulnerability: Even though the isAdmin variable was false for my session, the entire script—including the sensitive /admin-ygn04o path—was delivered to my browser.**
- **Information Disclosure: The "secret" administrative URL was effectively public knowledge to anyone who inspected the page source.**

3. Exploitation Methodology (Proof of Concept)

Step 1: URL Extraction

I extracted the hardcoded path /admin-ygn04o from the script.

Step 2: Access Control Bypass

By appending the discovered path to the base domain, I attempted to access the administrative interface directly:

https://[LAB-ID].web-security-academy.net/admin-ygn04o

Step 3: Verification of Broken Access Control

The server responded with the administrative panel. This confirmed two critical failures:

1. **Authorization Failure: The server did not check my user role.**
2. **Authentication Failure: No login was required to view the page.**

Step 4: Unauthorized Administrative Action

To demonstrate the full impact of the vulnerability, I targeted the user account carlos for deletion. The request was successfully processed by the server, confirming that I had full, unauthorized control over the user registry.

4. Root Cause Analysis

Component	Status	Security Risk
URL Predictability	Low	The URL used random characters (ygn04o), making it difficult to brute-force.
Path Exposure	Critical	The path was hardcoded in JavaScript and sent to unauthenticated users.
Server-Side Security	None	No validation was performed on the backend to verify the requester's identity.

5. Remediation & Hardening

A. Implement Server-Side Authorization (Primary)

Never rely on client-side logic (JavaScript) to restrict access to sensitive pages. Every request to /admin-ygn04o (or any admin path) must be intercepted by a server-side middleware that verifies the user's session and administrative role.

B. Remove Sensitive References from Code

Sensitive URLs and administrative logic should never be rendered in the HTML/JavaScript provided to standard users. If a user is not an administrator, the /admin reference should not exist in their source code at all.

6. Conclusion

This project proves that Security by Obscurity is not security. The use of a random URL failed because the path was disclosed in the public source code. This vulnerability could allow an attacker to gain total control over user data and system settings. This finding underscores the importance of Defense in Depth and the requirement for all security checks to happen on the Server-Side.