# [Lab Report] File Path Traversal: Arbitrary File Read via Parameter Manipulation

## Project Information

- **Analyst:** Mduduzi William Radebe
- **Date:** 07 January 2026
- **Target:** PortSwigger Web Security Academy (Apprenticeship Lab)
- **Vulnerability Type:** CWE-22: Improper Limitation of a Pathname to a Restricted Directory ('Path Traversal')
- **Severity: High (8.5/10)**

## 1. Executive Summary

During a routine security assessment of the target web application, a **File Path Traversal** vulnerability was discovered within the image retrieval functionality. By manipulating the filename parameter, I successfully bypassed directory restrictions to perform an **Arbitrary File Read**. This allowed for the retrieval of the /etc/passwd file from the underlying Linux server, exposing system user information.

## 2. Technical Analysis & Identification

### Surface Area Mapping

The application functions by loading product images dynamically. I identified an endpoint that retrieves files via a URL parameter:

GET /image?filename=48.jpg

### Vulnerability Hypothesis

Based on the URL structure, I hypothesized that the backend logic was likely executing a file system call similar to:

file_get_contents("/var/www/images/" + $_GET['filename']);

The lack of input sanitization suggested that if I injected "dot-dot-slash" (../) sequences, I could traverse outside the /var/www/images/ directory.

## 3. Exploitation Workflow (Proof of Concept)

### Step 1: Interception & Modification

I used the browser's Developer Tools to inspect the DOM. I manually edited the src attribute of an <img> tag to test for traversal.

### Step 2: Payload Selection

To ensure I hit the root directory regardless of the current folder depth, I utilized a deep traversal string:

../../../../../../etc/passwd

### Step 3: Execution

The browser sent the following modified request to the server:

GET /image?filename=../../../../../../etc/passwd HTTP/1.1

### Step 4: Evidence of Success

The server responded by streaming the contents of the /etc/passwd file directly into the image source container. I confirmed the presence of:

- root:x:0:0:root:/root:/bin/bash
- Multiple service accounts (daemon, bin, sys, etc.)

# 4. Detailed Findings

| **Indicator** | **Details** |
|---|---|
| **Vulnerable Parameter** | filename |
| **Tested Payload** | ../../../../../../etc/passwd |
| **Impact** | Unauthorized access to sensitive OS configuration files |
| **Access Level** | Read-Only (Arbitrary) |

# 5. Risk Assessment (DREAD Model)

- **Damage Potential:** High (Information disclosure of system users).
- **Reproducibility:** High (Easily reproducible with basic tools).
- **Exploitability:** High (Requires no authentication).
- **Affected Users:** All (Internal system data exposed).
- **Discoverability:** High (Commonly tested endpoint).

# 6. Remediation & Hardening

To secure the application against this vulnerability, I recommend the following defensive measures:

### A. Path Validation (Primary Defense)

The application should validate user input against a **whitelist** of permitted file extensions (e.g., .jpg, .png) and ensure the filename contains only alphanumeric characters.

### B. Filesystem API Best Practices

Use the basename() function in PHP or similar language-specific utilities to strip away directory paths, ensuring only the filename is processed.

PHP
// Secure example
$filename = basename($_GET['filename']);
$path = "/var/www/images/" . $filename;

### C. Use of Filesystem Chroot

Run the web application in a chrooted environment or a containerized sandbox (Docker) to restrict the process's access to the wider operating system.

## 7. Conclusion

The successful exploitation of this Path Traversal vulnerability demonstrates a critical failure in the application's input validation logic. While the lab focused on /etc/passwd, a real-world attacker could use this to leak **database credentials, source code, and environment variables**, leading to a full system takeover.

## 8. Lab Status: [COMPLETED]

*Target File Accessed:* /etc/passwd

*Verification:* Contents of the file successfully rendered in the browser.