Vulnerability Analysis: CVE-2026-65694 - Unauthenticated Arbitrary File Read in Microweber CMS

Author: Denizhalil **Site:** denizhalil.com CVSS 7.5 HIGH **Date:** July 2026

INTRODUCTION

Content Management Systems (CMS) form the backbone of modern web applications, handling everything from content publishing and dynamic page creation to user administration and media file management. However, this centralized functionality also presents an attractive attack surface for malicious actors when input sanitization mechanisms fail. When input validation fails within core controllers that serve assets or files, the security impact on the host system can be exceptionally severe. This article explores **CVE-2026-65694**, a high-severity Path Traversal (CWE-22) vulnerability identified in Microweber CMS versions up to and including **2.0.20**. We will examine how an unauthenticated remote attacker can exploit this weakness to traverse directory structures, circumvent access controls, and read sensitive files directly from the underlying host filesystem without requiring any valid credentials or user interaction.

LEARNING OBJECTIVES

By reading this analysis, you will be able to:

- Understand the core mechanics of Path Traversal (CWE-22) vulnerabilities in web applications.
- Analyze the root cause within Microweber CMS's `ServeStaticFileController`.
- Identify affected software versions and potential security impacts on production environments.
- Implement mitigation strategies to secure vulnerable instances against arbitrary file read exploits.

WHAT IS MICROWEBER CMS <= 2.0.20 - UNAUTHENTICATED ARBITRARY FILE READ CVE-2026-65694

CVE-2026-65694 represents a critical security flaw identified in **Microweber CMS <= 2.0.20**, a widely adopted open-source drag-and-drop website builder and content management platform built on the PHP Laravel framework. Designed to simplify web development for businesses and developers alike, Microweber includes various built-in controllers to manage, process, and deliver user-uploaded media and static assets. However, a fundamental flaw in its asset routing mechanism exposes host file systems to unauthorized remote inspection. The vulnerability stems from improper path handling (CWE-22) within the core static file delivery routines, permitting unauthenticated remote attackers to bypass directory restrictions. Because the application fails to enforce strict boundaries on relative path inputs passed via HTTP requests, remote clients can navigate backward through the directory tree. This allows attackers to access confidential files stored far outside the designated web root directory without needing active user sessions or administrative privileges.

With a CVSS v3.1 base score of **7.5 (High)** and a CVSS v4.0 score of **8.7 (High)**, this weakness poses an immediate threat to application confidentiality and overall server integrity. Attackers exploiting this

vulnerability can retrieve critical operational data, leading to severe downstream consequences across the organization's infrastructure:

- **Exposure of Application Configuration Files:** Attackers can read sensitive `.env` files containing raw database passwords, secret app keys (`APP_KEY`), mail server credentials, and third-party API tokens.
- **Leakage of Operating System Assets:** Exploitation allows arbitrary reading of critical system configuration files, such as `/etc/passwd` or process details, aiding in further system reconnaissance.
- **Cryptographic Key Compromise:** Exposure of framework-level encryption keys permits attackers to forge session cookies, decrypt stored data, or craft secondary attacks.
- **Escalation to Full System Takeover:** Exfiltrated credentials frequently grant attackers direct access to administrative databases or SSH channels, turning an information disclosure flaw into complete Remote Code Execution (RCE).

```
// Vulnerable implementation inside ServeStaticFileController.php

namespace MicroweberPackages\App\Http\Controllers;

use Illuminate\Http\Request;

class ServeStaticFileController extends Controller
{
    public function serveFromUserfiles(Request $request)
    {
        // Vulnerable: Directly retrieving path parameter without strict canonicalization
        $relativePath = $request->get('path');

        if (!$relativePath) {
            return response()->json(['error' => 'Path parameter is required'], 400);
        }

        // Inadequate normalization fails to eliminate recursive directory traversal sequences
(../)
        $normalizedPath = normalize_path($relativePath);

        // Base directory concatenation without checking if realpath() stays within userfiles
dir
        $fullPath = userfiles_path() . $normalizedPath;

        if (file_exists($fullPath)) {
            return response()->file($fullPath); // Serves arbitrary files to unauthenticated
users
        }

        return response()->json(['error' => 'File not found'], 404);
    }
}
```

TECHNICAL DETAIL: HOW THE VULNERABILITY WORKS

The technical underlying cause of **CVE-2026-65694** lies within the static asset handling architecture implemented by Microweber CMS, specifically inside the `ServeStaticFileController::serveFromUserfiles()` endpoint. Designed to dynamically handle user-uploaded media and cached resources, this controller receives asset requests, resolves their location relative to the installation's web storage root, and returns the raw file contents. However, the mechanism fails to enforce boundary constraints on user-controlled inputs, breaking the fundamental security principle of input isolation. When a client sends a request to fetch an asset, the application parses the target resource directly from incoming HTTP parameters without ensuring the input is bound strictly to the designated public directory. Instead of resolving the canonical path via secure filesystem checks and validating that the target resides inside the application's `/userfiles/` directory, the controller trusts modified path strings. Consequently, path traversal payloads containing relative sequences (such as `../`) are passed straight into native filesystem functions, instructing the underlying operating system to step out of the intended folder structure.

Because the underlying PHP routines resolve these directory traversal characters natively, the application seamlessly walks back through the filesystem taxonomy until reaching the server's root or application base path. This fundamental flaw in request processing introduces critical exploitation vectors across the target host:

- **Direct Parameter Extraction:** The controller accepts raw input via GET parameters (e.g., `?path=...`) directly into the file resolution pipeline without enforcing strict routing validation.
- **Flawed Path Normalization:** Legacy helper functions (like `normalize_path()`) focus on standardizing directory slashes rather than stripping recursive `../` traversal strings or URL-encoded variations.
- **Missing Boundary Checks:** The core execution flow lacks checks using `realpath()` to verify that the absolute path stays confined to the designated `/userfiles/` directory before serving the resource.
- **Unrestricted File Delivery:** Upon locating the file on the host filesystem, `response()->file()` streams the target binary or text content directly back to the unauthenticated HTTP client.

```
GET /api/userfiles/serve?path=../../../../.env HTTP/1.1
Host: target-microweber-instance.com
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64)
Accept: */*
Connection: close
```

AFFECTED SOFTWARE & PLUGINS

The scope of **CVE-2026-65694** spans core application components responsible for handling static asset delivery and user-uploaded media within vulnerable Microweber installations. Any deployment exposed to the public internet without external filtering mechanisms remains directly vulnerable to unauthenticated path traversal and unauthorized data exfiltration:

- **Core Application Versions:** All instances running Microweber CMS versions **<= 2.0.20**.
- **Vulnerable Component:** The `ServeStaticFileController::serveFromUserfiles()` endpoint along with its underlying path resolution and normalization functions.

- **Exposed Environments:** Any server architecture hosting affected releases where static file API endpoints are publicly accessible without secondary Web Application Firewall (WAF) rule sets.

CONCLUSION

CVE-2026-65694 serves as a stark reminder of the persistent security risks associated with improper path sanitization and insecure file handling in modern web applications. By failing to strictly validate user-supplied parameters within asset delivery controllers, Microweber CMS leaves an open door for unauthenticated remote attackers to bypass access controls and extract critical system assets. The severity of this flaw cannot be understated given its low barrier to entry and high potential impact. Because exploitation requires zero prior authentication or complex interaction, automated scanners and malicious actors can effortlessly identify exposed instances and retrieve sensitive environment files. The exposure of sensitive configuration keys, database credentials, and cryptographic parameters often provides the initial foothold required to orchestrate far more devastating full-system compromises. To effectively mitigate this threat, administrators and organization security teams running Microweber CMS must prioritize immediate remediation. Running installations should be thoroughly audited to identify active versions, and systems operating on affected releases must be updated beyond version 2.0.20 as soon as vendor patches are applied. In environments where updates cannot be deployed instantaneously, access to the vulnerable endpoint should be restricted or blocked at the web server level.

Beyond applying immediate software patches, organizations should adopt defense-in-depth measures to safeguard their web infrastructure against similar directory traversal vulnerabilities. Enforcing the principle of least privilege on web server processes, utilizing Web Application Firewalls (WAFs) equipped with path traversal detection rules, and implementing strict filesystem access controls will significantly reduce the attack surface and prevent unauthorized asset exfiltration across all layers of the application environment.