WordPress Core 6.9-7.0.1 - Pre-Auth Blind SQL Injection (Batch-Route Confusion)

Author: Synthex **Site:** denizhalil.com CVSS 9.8 CRITICAL **Date:** July 2026

INTRODUCTION

Discovered as a major security flaw in the core architecture of WordPress, the vulnerability chain colloquially known as **wp2shell** represents one of the most severe threat vectors impacting the Content Management System ecosystem. Because WordPress powers over 40% of all websites globally, the blast radius of this default-configuration flaw is exceptionally wide. This exploit chain targets two integral components of WordPress Core: the REST API batching infrastructure and the underlying `WP_Query` database abstraction layer. By seamlessly combining these internal architectural flaws, a completely remote, unauthenticated attacker can bypass standard perimeter defenses to achieve full Remote Code Execution (RCE) on a stock installation—requiring absolutely no third-party plugins or active themes to succeed. With a CVSS base score of **9.8 (Critical)**, this vulnerability represents an existential threat to unpatched servers, requiring immediate attention, auditing, and remediation from system administrators and security operations teams worldwide.

LEARNING OBJECTIVES

By the end of this article, you will understand:

- The structural design and mechanics of the WordPress REST API batching system.
- The root cause of the "Batch-Route Confusion" logic flaw tracked under **CVE-2026-63030**.
- Proactive identification strategies and mitigation techniques to secure vulnerable environments.
- How the input sanitization failure in `WP_Query` leads to the SQL Injection tracked under **CVE-2026-60137**.
- The end-to-end chaining process that allows low-severity bugs to escalate into unauthenticated server compromise.

WHAT IS WORDPRESS CORE 6.9-7.0.1 - PRE-AUTH BLIND SQL INJECTION CVE-2026-63030

CVE-2026-63030 is an improper access control vulnerability residing natively in the batch request handler of the WordPress REST API framework. The WordPress REST API features a dedicated batching endpoint (`/wp-json/batch/v1`) designed to allow clients, developers, and blocks to bundle multiple individual API calls into a single HTTP request to optimize performance and lower server round-trips. This endpoint is active by default across millions of installations, meaning the underlying architectural flaw is exposed out-of-the-box without requiring any custom API modifications or configuration

changes by the site administrator. The vulnerability itself stems from a fundamental logical parsing error within the array processing loop of this batch endpoint. When the server is forced to handle a meticulously malformed payload, the internal routing mechanics lose synchronization, causing an array index misalignment. This desynchronization tricks the server into evaluating a high-privilege, restricted internal route against the lax or nonexistent permission check callbacks intended for an entirely different, public-facing endpoint. Consequently, an anonymous visitor can cleanly execute internal API routines that should normally require strict administrative authentication.

While this authorization bypass is highly restricted in terms of what it can execute on its own, it acts as the primary gateway for a catastrophic exploit chain. By providing an unauthenticated vehicle to reach internal code pathways, it directly exposes deeper database management classes to malicious manipulation. When chained with secondary flaws in input handling, this flaw transforms from a simple routing oversight into a severe exploit mechanism capable of completely compromising the underlying database layer.

To break down its core characteristics, the vulnerability is defined by the following fundamental traits:

- **Zero-Authentication Requirement:** The flaw can be triggered remotely by completely anonymous users before any login screen, session cookie, or API token is requested or validated by the application.
- **Core Vulnerability Status:** The security defect resides strictly within the core WordPress codebase, meaning every stock installation within the affected version window is vulnerable regardless of active plugins or themes.
- **Desynchronization Trigger:** The exploit relies on an array index drift within the `serve_batch_request_v1` method, which directly maps incoming data structures to internal permission validation routines.
- **Gateway for Deep Exploitation:** It acts as the necessary first stage of the `wp2shell` chain, smuggling restricted parameters into internal queries to ultimately execute the blind SQL injection.

TECHNICAL DETAIL: HOW THE VULNERABILITY CHAIN WORKS

The structural danger of the `wp2shell` vulnerability chain lies in how it seamlessly bridges the gap between an abstract logical flaw and direct database interaction. In modern web architectures, security models heavily rely on layered boundaries, assuming that if a request passes the API gateway, the underlying components can trust its structure. This vulnerability turns that assumption into a liability by undermining the integrity of the WordPress routing parser before data ever reaches the internal database abstraction layers. By meticulously breaking down the communication protocols of the core application, the exploit demonstrates how minor, isolated structural oversights can lead to absolute system compromise when chained sequentially. Rather than breaking through a reinforced cryptographic firewall, the attack subtly misaligns the internal gears of the application loop. Once this alignment is lost, the application itself inadvertently ferries the malicious payload past its own defensive perimeters, exposing deep, unescaped database query parameters directly to the attacker.

STEP 1: BATCH-ROUTE CONFUSION (CVE-2026-63030)

WordPress utilizes the `WP_REST_Server::serve_batch_request_v1()` method to iterate through an array of sub-requests provided by the client, executing them sequentially and matching each route with its corresponding permission check callback. An attacker exploits this behavior by constructing a batch payload containing an intentionally malformed or structurally anomalous entry mid-sequence. During processing, this structural anomaly causes an array index misalignment within the execution loop. The internal pointer mapping individual request objects to their mandatory validation filters drifts out of sync. As a result, a highly restricted administrative internal route is evaluated using the lax or empty permission schema intended for a completely different public endpoint. Passing through the misaligned gate, the server executes the privileged routine with parameters fully controlled by the unauthenticated user.

To illustrate this structural breakdown, consider how the server handles a typical JSON batch request array compared to a malicious payload designed to induce index drift:

```
/* Concept of a typical, valid batch request array */
[
  { "method": "GET", "path": "/wp/v2/posts" },      /* Public Route  -> Checks public
permissions */
  { "method": "GET", "path": "/wp/v2/comments" }   /* Public Route  -> Checks public
permissions */
]

/* Concept of an exploit payload forcing index desynchronization */
[
  {
    "method": "POST",
    "path": "/wp/v2/users",
    "invalid_structure_trigger": [null]
  },                                                /* Malformed Entry -> Causes loop
pointer misalignment */
  {
    "method": "GET",
    "path": "/wp/v2/internal/privileged-endpoint",
    "smuggled_params": { ... }
  }                                                 /* Privileged Entry -> Evaluated
using public permission filter */
]
```

STEP 2: SMUGGLING THE SQL INJECTION (CVE-2026-60137)

Once the authorization layer is bypassed via route confusion, the attacker successfully smuggles parameter payloads into internal query handlers (such as `/wp/v2/posts`) that are normally shielded from direct anonymous access. This exposes the application to **CVE-2026-60137**. The smuggled

parameters are passed directly into the core `WP_Query` engine. Specifically, the handling of the `author__not_in` (or `author_exclude`) parameter fails to properly escape, sanitize, or cast user-supplied inputs before appending them into raw database operations. By supplying structured malicious payloads into this parameter, the attacker executes a Time-Based or UNION-Based Blind SQL Injection. This allows the attacker to exfiltrate administrative password hashes and the site's secret authentication salts (`AUTH_KEY`, `SECURE_AUTH_KEY`). By reconstructing authentication cookies or cracking the extracted hashes, the attacker gains full administrative access, uploads a malicious plugin or theme, and establishes complete Remote Code Execution (RCE) over the underlying server host.

To visualize how the absence of input casting breaks down at the database layer, consider the difference between a standard query and the unescaped injection string generated by the vulnerability:

```
-- Expected execution behavior for legitimate numeric IDs:
SELECT * FROM wp_posts WHERE post_author NOT IN (2, 5, 12);

-- Exploitation execution behavior resulting from unescaped string injection:
SELECT * FROM wp_posts WHERE post_author NOT IN (0) UNION SELECT 1,2,3,4,user_pass
FROM wp_users WHERE user_login='admin' -- );
```

AFFECTED SOFTWARE & PLUGINS

The critical severity of this vulnerability stems primarily from its placement within the native architecture of the platform, completely independent of any third-party plugins, custom extensions, or active themes. Because the flaw impacts the core REST API and query routing logic, every standard installation operating within the targeted release branches is exposed by default, significantly expanding the overall attack surface across the web ecosystem. While the WordPress Core security team quickly responded by deploying forced automatic background updates to mitigate the threat globally, relying solely on automated systems introduces operational risk. Server-side constraints, strict file permission hardening, or aggressive caching reverse proxies can silently block these upstream updates from applying, making manual verification a necessity for security teams.

- **Vulnerable Core Branches:** Applies natively to all environments running WordPress Core versions **6.9.0 through 6.9.4**, as well as versions **7.0.0 through 7.0.1**.
- **Patched Suffixes:** The underlying architecture was fully secured and mitigated with the official distribution of versions **6.9.5** and **7.0.2**.
- **Zero-Dependency Risk:** The vulnerability requires no specific plugin configurations or administrative modifications to be exploited, impacting stock installations completely out-of-the-box.

CONCLUSION

The discovery of the Pre-Auth Blind SQL Injection and Batch-Route Confusion chain underscores the foundational cyber security principle of defense-in-depth. A logic error that appears isolated at the API

routing layer can quickly become fatal when combined with insufficient input validation at the database abstraction layer. When application layers trust each other blindly without strict verification at every boundary, even complex web ecosystems remain highly vulnerable to catastrophic exploitation. This specific flaw highlights that perimeter defenses alone are insufficient if the core application components fail to maintain rigid boundaries during cross-component data transmittal. Web administrators must urgently verify their running WordPress core versions to ensure their environments are completely insulated from this attack vector. If immediate updates are not feasible due to legacy constraints, staging environments, or strict production freeze windows, temporary virtual patching via Web Application Firewalls (WAF) must be deployed immediately. Specifically, network teams should implement strict edge rules to drop all external traffic targeting the `/wp-json/batch/v1` endpoint and its query parameter variations. Furthermore, implementing real-time monitoring on database logs for unusual `UNION` queries targeting the users table can serve as an invaluable secondary detection layer.

Ultimately, this vulnerability serves as a stark reminder that modern threat landscapes move at an accelerated pace, often outpacing passive security measures. Relying on default configurations or assuming background systems always function flawlessly creates a dangerous gap in visibility that sophisticated attackers are quick to exploit. In modern web infrastructure, maintaining a proactive, rigorously audited, and rapid patch management cycle remains the single most effective defense against systemic server compromise. Security teams must transition from passive reliance on automation to active, verified governance of their application dependencies.

REFERENCES

OSI Model: Data Communication in Computer Networks
https://denizhalil.com/2024/09/26/osi-model-computer-networks-data-communication/

WPScan: An Essential Tool for WordPress Security Scanning
https://denizhalil.com/2024/10/15/wpscan-wordpress-security/

Introduction to Blockchain Technology with a Python Example
https://denizhalil.com/2024/02/10/introduction-python-blockchain-example/

Exploitation of Microsoft Defender Elevation of Privilege Vulnerability (CVE-2026-50656)
https://denizhalil.com/2026/06/18/cve-2026-50656-microsoft-defender-eop-vulnerability-analysis/

Critical Vulnerability in WordPress YMC Filter: Unauthenticated Content Disclosure (CVE-2026-10823)
https://denizhalil.com/2026/06/30/cve-2026-10823-ymc-filter-wordpress-vulnerability/

WP2Shell PoC
https://github.com/Icex0/wp2shell-poc/

WP2Shell WordPress Core Pre-Auth Batch Route Confusion Leading to SQL Injection
https://pentest-tools.com/vulnerabilities-exploits/wp2shell-wordpress-core-690-694-and-700-701-pre-auth-batch-route-confusion-leading-to-sql-injection_29452

CVE-2026-63030 – ProjectDiscovery Library
https://cloud.projectdiscovery.io/library/CVE-2026-63030