INTELLIGENCE BRIEFING
Security Command Center
TLP:CLEAR
2026-06-01 06:14 UTC

CVE-2025-11993: The WooCommerce Infinite Scroll and Ajax Pagination plugin for WordPress is vulnerable to PHP Object...

CVE VULNERABILITY | HIGH | CVSS 8.8

SCC Item ID	SCC-CVE-2026-0245
Type	CVE Vulnerability
CVE ID	CVE-2025-11993
Severity	HIGH
CVSS Base Score	8.8
EPSS Score	0.0008 (24th percentile)
Affected Products	WooCommerce Infinite Scroll and Ajax Pagination plugin for WordPress, all versions up to and including 1.8
Published	2026-05-29T07:16:13.730
Discovery Source	Nvd

Executive Summary

A PHP Object Injection vulnerability in the WooCommerce Infinite Scroll and Ajax Pagination WordPress plugin (versions 1.8 and below) allows authenticated attackers with minimal privileges to inject arbitrary PHP objects. If any other installed plugin or theme contains a usable exploit chain, this vulnerability can escalate to remote code execution, file deletion, or sensitive data theft. WooCommerce-based e-commerce sites running this plugin should treat this as a priority remediation item. 72-hour remediation window recommended for patch deployment.

Technical Analysis

CVE-2025-11993 is a PHP Object Injection vulnerability (CWE-502: Deserialization of Untrusted Data) affecting WooCommerce Infinite Scroll and Ajax Pagination plugin for WordPress, all versions up to and including 1.8. The flaw resides in the 'import_settings' function, which deserializes user-supplied data passed via the 'settings' parameter without performing capability checks. Authenticated users at Subscriber level or above can inject arbitrary PHP objects. Standalone exploitation is limited, no native Property Oriented Programming (POP) chain exists in the plugin. However, if any co-installed plugin or theme supplies a POP chain, the attack surface expands to arbitrary file deletion, sensitive data retrieval, or remote code execution, mapped to MITRE ATT&CK T1190 (Exploit Public-Facing Application) and T1059 (Command and Scripting Interpreter), specific shell type

depends on POP chain implementation. CVSS base score: 8.8 (High). EPSS: 0.08% (23rd percentile). Not currently listed on CISA KEV. CVSS vector string not yet published by NVD; base score of 8.8 is confirmed pending vector release. Patch status: update to a version above 1.8 if available; verify with the WordPress plugin repository.

Action Checklist

1. Step 1: Containment. Immediately identify all WordPress instances running WooCommerce Infinite Scroll and Ajax Pagination plugin version 1.8 or below. Restrict Subscriber-level account access specifically to the plugin's import_settings endpoint via WAF rule or plugin-specific capability lockdown until patch is applied. Test thoroughly to ensure legitimate customer workflows are not disrupted. Disable the plugin if it is not business-critical. Reference: NIST AC-3 (Access Enforcement), CIS 4.4 (Implement and Manage a Firewall on Servers).
2. Step 2: Detection. Query WordPress access logs and application logs for POST requests targeting the plugin's import_settings endpoint with a 'settings' parameter containing serialized PHP object patterns (e.g., 'O:' prefix in URL-decoded request bodies). Review user activity logs for Subscriber-level accounts making unexpected administrative-style requests. Enable and review WordPress debug logs (WP_DEBUG_LOG) for deserialization errors. Reference: NIST AU-6 (Audit Record Review, Analysis, and Reporting), CIS 8.2 (Collect Audit Logs).
3. Step 3: Eradication. Update the WooCommerce Infinite Scroll and Ajax Pagination plugin to the latest version above 1.8 via the WordPress admin dashboard or WP-CLI ('wp plugin update'). Audit all co-installed plugins and themes for known POP chain vulnerabilities using a WordPress security scanner (e.g., WPScan). Remove or update any plugin or theme that provides a POP chain pairable with this vulnerability. Reference: NIST SI-2 (Flaw Remediation), CIS 7.3 (Perform Automated Operating System Patch Management), CIS 7.4 (Perform Automated Application Patch Management).
4. Step 4: Recovery. After patching, verify the plugin version reflects the updated release. Re-enable any temporarily disabled functionality. Run a full WordPress integrity check to confirm no unauthorized file modifications occurred before remediation. Monitor application logs for continued deserialization attempts. Review all Subscriber-level and above accounts for signs of unauthorized access or privilege escalation. Reference: NIST IR-4 (Incident Handling), NIST SI-4 (System Monitoring), D3-SFA (System File Analysis).
5. Step 5: Post-Incident. Review the WordPress plugin vetting and update process. Implement automated plugin vulnerability scanning as part of the CI/CD or maintenance workflow. Enforce least-privilege account policies, minimize Subscriber-level account creation on WooCommerce stores where not required. Document this event as a case study for deserialization risk in third-party plugin ecosystems. Reference: NIST AC-6 (Least Privilege), CIS 5.4 (Restrict Administrator Privileges to Dedicated Administrator Accounts), CIS 7.1 (Establish and Maintain a Vulnerability Management Process), D3-UAP (User Account Permissions).

IR / Forensic Enrichment

Triage Priority	URGENT

Escalation Criteria	Escalate immediately to legal and executive leadership if forensic review of access logs, wp_options, or wp-content/uploads confirms a successful POP chain execution (indicated by unexpected PHP files in uploads, modified core files, or exfiltration of wp-config.php database credentials), as WooCommerce stores processing payment card or customer PII data may trigger PCI-DSS incident notification and applicable state/federal breach notification obligations.
Recovery Notes	After patching the WooCommerce Infinite Scroll and Ajax Pagination plugin above version 1.8, monitor Apache/Nginx access logs and WordPress debug.log continuously for at least 30 days for recurring POST requests to admin-ajax.php with 'action=import_settings' — repeated attempts after patching may indicate an attacker with persistent access (webshell or backdoor account) established before remediation. Verify that no new PHP files exist in wp-content/uploads/ and that the wp_users table contains no unrecognized Subscriber-level accounts added during the exposure window. Run 'wp core verify-checksums' and 'wp plugin verify-checksums --all' weekly for the first month post-recovery to confirm no files were silently modified by a pre-patch exploit.
Forensic Artifacts	Apache/Nginx access logs: POST requests to /wp-admin/admin-ajax.php where the request body contains 'action=import_settings' and a URL-encoded serialized PHP object in the 'settings' parameter (pattern: 'O%3A' or 'O:' after URL decoding) — the primary exploit delivery artifact for CVE-2025-11993. | wp-content/uploads/ directory: Presence of any .php files is anomalous in a standard WooCommerce deployment and indicates successful file-write via a POP chain gadget (e.g., a __destruct or __wakeup method in a co-installed plugin that writes attacker-controlled content to the filesystem). | WordPress database wp_options table (autoloaded entries): A successful PHP Object Injection may persist malicious serialized objects in autoloaded options that execute on every page load — export and inspect with 'wp option list --autoload=yes --format=json' and look for base64-encoded or obfuscated PHP strings in option values. | wp-users and wp_usermeta tables: Attacker-created Subscriber accounts used to authenticate and trigger the import_settings AJAX action — query for accounts registered during the exposure window and cross-reference source IPs against access logs to identify attacker infrastructure. | PHP error log / wp-content/debug.log: Deserialization errors, undefined class warnings, or __wakeup/__destruct method execution traces generated during exploit attempts against CVE-2025-11993 — these appear even for failed exploitation attempts where the POP chain class is not available, providing a forensic record of probe activity.

Per-Action IR Details

Step 1: Containment — Immediately identify all WordPress instances running WooCommerce Infinite Scroll and Ajax Pagination plugin version 1.8 or below. Restrict Subscriber-level and above account registration or access to the import_settings function via WAF rule or WordPress user role lockdown until a patch is applied. Disable the plugin if it is not business-critical. Reference: NIST AC-3 (Access Enforcement), CIS 4.4 (Implement and Manage a Firewall on Servers).

NIST Phase: Containment

Reference: NIST 800-61r3 §3.3 — Containment Strategy

Controls: NIST AC-3 (Access Enforcement), NIST AC-6 (Least Privilege), CIS 4.4 (Implement and Manage a Firewall on Servers), CIS 4.5 (Implement and Manage a Firewall on End-User Devices)

Compensating: Use WP-CLI to enumerate all sites in a multisite network: 'wp site list --field=url | xargs -I{} wp --url={} plugin list --name=woo-ajax-pagination --status=active'. Block the vulnerable endpoint at the server level with an Nginx location block or Apache RewriteRule targeting the AJAX action 'import_settings' (e.g., deny POST requests where the body contains the 'settings' parameter matching the action=import_settings nonce flow). Alternatively, add a ModSecurity rule or Cloudflare free-tier WAF custom rule to block requests containing 'import_settings' to

wp-admin/admin-ajax.php. Disable user registration via Settings > General in wp-admin to prevent net-new Subscriber account creation during the containment window.

Evidence: Before disabling the plugin, capture a snapshot of the active plugin list via 'wp plugin list --format=json > plugin_inventory_$(date +%F).json'. Preserve the plugin's PHP source file at wp-content/plugins/woo-ajax-pagination/ in a read-only archive — this establishes the vulnerable code baseline and confirms version 1.8 is present. Record all current Subscriber-level user accounts: 'wp user list --role=subscriber --format=json > subscribers_$(date +%F).json'. These artifacts establish the pre-containment attack surface for post-incident review.

Step 2: Detection — Query WordPress access logs and application logs for POST requests targeting the plugin's import_settings endpoint with a 'settings' parameter containing serialized PHP object patterns (e.g., 'O:' prefix in URL-decoded request bodies). Review user activity logs for Subscriber-level accounts making unexpected administrative-style requests. Enable and review WordPress debug logs (WP_DEBUG_LOG) for deserialization errors. Reference: NIST AU-6 (Audit Record Review, Analysis, and Reporting), CIS 8.2 (Collect Audit Logs).

NIST Phase: Detection Analysis

Reference: NIST 800-61r3 §3.2 — Detection and Analysis

Controls: NIST AU-6 (Audit Record Review, Analysis, and Reporting), NIST AU-2 (Event Logging), NIST AU-3 (Content Of Audit Records), CIS 8.2 (Collect Audit Logs)

Compensating: Run the following grep against your Nginx or Apache access log to identify serialized PHP object payloads sent to the AJAX endpoint: 'grep -E "POST.*admin-ajax\.php" /var/log/nginx/access.log | grep -i "import_settings" | grep -P "O%3A|O:"'. URL-decode suspicious entries with: 'python3 -c "import urllib.parse,sys; print(urllib.parse.unquote(sys.argv[1]))" "'". For WordPress debug logging, add 'define("WP_DEBUG", true); define("WP_DEBUG_LOG", true);' to wp-config.php and monitor wp-content/debug.log for 'unserialize()' warnings or '__wakeup'/'__destruct' method traces. Install the free WP Activity Log plugin to capture Subscriber-level AJAX actions with timestamps and source IPs.

Evidence: Capture raw Apache/Nginx access logs covering at least 90 days prior to detection — specifically POST requests to '/wp-admin/admin-ajax.php' with 'action=import_settings' in the body. Extract and URL-decode the 'settings' POST parameter from any matching entries; a valid exploit payload will contain a PHP serialized object string beginning with 'O:' followed by a class name and property count (e.g., 'O:29:"WP_Community_Events_Location":2:{...}'). Preserve WordPress debug.log if WP_DEBUG_LOG was previously enabled, as failed or partially successful deserialization attempts may generate PHP notices. Pull authentication logs from wp-login.php and xmlrpc.php for Subscriber-level account logins that preceded the AJAX calls — this establishes whether the attacker self-registered or used a compromised credential.

Step 3: Eradication — Update the WooCommerce Infinite Scroll and Ajax Pagination plugin to the latest version above 1.8 via the WordPress admin dashboard or WP-CLI ('wp plugin update'). Audit all co-installed plugins and themes for known POP chain vulnerabilities using a WordPress security scanner (e.g., WPScan). Remove or update any plugin or theme that provides a POP chain pairable with this vulnerability. Reference: NIST SI-2 (Flaw Remediation), CIS 7.3 (Perform Automated Operating System Patch Management), CIS 7.4 (Perform Automated Application Patch Management).

NIST Phase: Eradication

Reference: NIST 800-61r3 §3.4 — Eradication

Controls: NIST SI-2 (Flaw Remediation), CIS 7.3 (Perform Automated Operating System Patch Management), CIS 7.4 (Perform Automated Application Patch Management), CIS 2.2 (Ensure Authorized Software is Currently Supported), CIS 7.2 (Establish and Maintain a Remediation Process)

Compensating: Run WPScan (free tier) against the site to enumerate all installed plugins and themes and cross-reference against its vulnerability database: 'wpscan --url https://yoursite.com --enumerate p,t --plugins-detection aggressive --api-token '. Specifically look for co-installed plugins with known POP chain exposure — common culprits include plugins using Yoast SEO's WPSEO_Date_Archive_Widget, Guzzle, or Monolog classes. After updating the vulnerable plugin with 'wp plugin update woo-ajax-pagination', verify the installed version: 'wp plugin get

woo-ajax-pagination --field=version'. Use YARA rules targeting PHP webshell patterns to scan wp-content/uploads/ and wp-content/plugins/ for any files dropped by a successful pre-patch exploit: 'yara /path/to/php_webshell.yar /var/www/html/wp-content/'.

Evidence: Before applying the patch, preserve a complete file hash manifest of the wp-content/plugins/ directory using: 'find /var/www/html/wp-content/plugins -type f -name "*.php" -exec md5sum {} \; > plugin_hashes_pre_patch_$(date +%F).txt'. This baseline enables post-patch integrity comparison to detect any files modified by a POP chain exploit that may have occurred before containment. Also export the WordPress options table to check for injected malicious data: 'wp option list --format=json > options_export_$(date +%F).json' — a successful Object Injection exploit may have written malicious serialized data into wp_options autoloaded entries.

Step 4: Recovery — After patching, verify the plugin version reflects the updated release. Re-enable any temporarily disabled functionality. Run a full WordPress integrity check to confirm no unauthorized file modifications occurred before remediation. Monitor application logs for continued deserialization attempts. Review all Subscriber-level and above accounts for signs of unauthorized access or privilege escalation. Reference: NIST IR-4 (Incident Handling), NIST SI-4 (System Monitoring), D3-SFA (System File Analysis).

NIST Phase: Recovery

Reference: NIST 800-61r3 §3.5 — Recovery

Controls: NIST IR-4 (Incident Handling), NIST SI-4 (System Monitoring), NIST AU-6 (Audit Record Review, Analysis, and Reporting), CIS 5.1 (Establish and Maintain an Inventory of Accounts), CIS 5.3 (Disable Dormant Accounts)

Compensating: Verify plugin version post-patch: 'wp plugin get woo-ajax-pagination --field=version'. Run WordPress core integrity check to detect any file-level tampering that may have occurred via a POP chain file-write gadget: 'wp core verify-checksums' and 'wp plugin verify-checksums --all'. Compare the pre-patch file hash manifest (captured in Step 3) against a fresh scan: 'find /var/www/html/wp-content/plugins -type f -name "*.php" -exec md5sum {} \; > plugin_hashes_post_patch_$(date +%F).txt && diff plugin_hashes_pre_patch_*.txt plugin_hashes_post_patch_*.txt'. Audit all Subscriber-level accounts for role changes using: 'wp user list --role=subscriber --format=json' and compare against the pre-containment export. Set up a cron job to alert on any new PHP files written to wp-content/uploads/ (a common webshell drop path): 'find /var/www/html/wp-content/uploads -name "*.php" -newer /tmp/baseline_timestamp -exec ls -la {} \;'.

Evidence: Before re-enabling functionality, capture the post-remediation state of wp_options for serialized data artifacts: 'wp option list --autoload=yes --format=json > options_post_patch_$(date +%F).json' and diff against the pre-patch export. Review wp-content/uploads/ for any PHP files, which would not be present under normal WooCommerce operation and would indicate a successful pre-patch file-write via a POP chain gadget. Pull the server's file access times (atime) for all files in wp-content/plugins/woo-ajax-pagination/ to identify if any plugin files were modified during the exposure window — this is the forensic boundary for determining breach vs. failed exploit attempts.

Step 5: Post-Incident — Review the WordPress plugin vetting and update process. Implement automated plugin vulnerability scanning as part of the CI/CD or maintenance workflow. Enforce least-privilege account policies — minimize Subscriber-level account creation on WooCommerce stores where not required. Document this event as a case study for deserialization risk in third-party plugin ecosystems. Reference: NIST AC-6 (Least Privilege), CIS 5.4 (Restrict Administrator Privileges to Dedicated Administrator Accounts), CIS 7.1 (Establish and Maintain a Vulnerability Management Process), D3-UAP (User Account Permissions).

NIST Phase: Post Incident

Reference: NIST 800-61r3 §4 — Post-Incident Activity

Controls: NIST AC-6 (Least Privilege), NIST IR-4 (Incident Handling), CIS 5.4 (Restrict Administrator Privileges to Dedicated Administrator Accounts), CIS 7.1 (Establish and Maintain a Vulnerability Management Process), CIS 7.2 (Establish and Maintain a Remediation Process), CIS 2.1 (Establish and Maintain a Software Inventory)

Compensating: Integrate WPScan into a weekly cron job or GitHub Actions workflow to automatically check all plugins against the WPScan Vulnerability Database API: 'wpscan --url https://yoursite.com --enumerate p --api-token --format json --output wpscan_report_$(date +%F).json'. Configure WooCommerce to disable open user registration if guest checkout is sufficient — navigate to WooCommerce > Settings > Accounts & Privacy and uncheck 'Allow customers to create an account on the My account page'. Add a PHP disable_functions restriction in php.ini to block

dangerous deserialization-adjacent functions ('unserialize' cannot be fully disabled, but restrict 'system', 'exec', 'shell_exec', 'passthru', 'proc_open') to limit POP chain execution impact. Document the specific POP chain risk profile of CVE-2025-11993 in your internal plugin vetting checklist as a pattern to screen for in future plugin reviews — specifically any plugin that passes user-controlled input to unserialize() without allowlisting.

Evidence: Compile the final incident timeline using: the Subscriber account creation timestamp from wp-users table ('wp db query "SELECT ID, user_login, user_registered, user_status FROM wp_users WHERE user_registered > DATE_SUB(NOW(), INTERVAL 90 DAY)"'), the first observed POST to import_settings from access logs, and any file system modification timestamps from wp-content/. This timeline documents the exposure window and is the primary artifact for any breach notification assessment. Preserve the WPScan output from Step 3 as the authoritative record of the co-installed plugin/theme POP chain risk landscape at the time of the incident.

Detection Guidance

Monitor WordPress server access logs for POST requests to endpoints associated with the plugin's import_settings function. Look for request bodies containing URL-encoded or raw PHP serialized object strings, specifically patterns beginning with 'O:' followed by integer and class name structures, which indicate PHP object serialization. Query web application firewall (WAF) logs for blocked or flagged requests matching deserialization payloads. Review WordPress authentication logs for Subscriber-level accounts performing actions outside normal browsing behavior. Enable WP_DEBUG_LOG and monitor for PHP unserialize() warnings or errors tied to plugin functions. If a SIEM is in use, create a rule alerting on HTTP POST requests to '/wp-admin/admin-ajax.php' or plugin-specific endpoints with 'settings' parameter values containing serialized data patterns. Reference: NIST AU-2 (Event Logging), AU-3 (Content of Audit Records), CIS 8.2 (Collect Audit Logs), D3-SFA (System File Analysis).

Framework Mappings

MITRE-ATTACK

- **T1059.004** — Unix Shell
- **T1190** — Exploit Public-Facing Application

NIST-800-53R5

- **CM-7** — Least Functionality
- **SI-3** — Malicious Code Protection
- **SI-4** — System Monitoring
- **CA-8** — Penetration Testing
- **RA-5** — Vulnerability Monitoring and Scanning
- **SC-7** — Boundary Protection
- **SI-2** — Flaw Remediation
- **SI-7** — Software, Firmware, and Information Integrity
- **SI-10** — Information Input Validation
- **AT-2** — Literacy Training and Awareness

OWASP-TOP10-2021

- **A08:2021** — Software and Data Integrity Failures

CIS-V8

- **16.10** — Apply Secure Design Principles in Application Architectures
- **14.2** — Train Workforce Members to Recognize Social Engineering Attacks

ISO-27001-2022

- **A.8.8** — Management of technical vulnerabilities

MITRE ATT&CK Mapping

Technique ID	Technique Name	Tactic
T1059.004	Unix Shell	Execution
T1190	Exploit Public-Facing Application	Initial-Access

Sources

Source	URL	Tier
nvd	https://nvd.nist.gov/vuln/detail/CVE-2025-11993	T1
CVE-2025-11993 - CVE Record	https://www.cve.org/CVERecord?id=CVE-2025-11993	T3
CVE-2025-11993: WooCommerce Infinite Scroll and Deserialization	https://www.sherlockforensics.com/blog/2026-05-29-cve-2025-11993.html	T3
CVE-2025-11993 - INCIBE	https://www.incibe.es/index.php/en/incibe-cert/early-warning/vulner...	T3
WooCommerce Infinite Scroll and Ajax Pagination Plugin ...	https://freshysites.com/security-bulletins/woocommerce-infinite-scr...	T3

Generated 2026-06-01 06:14 UTC by TJS Security Command Center