INTELLIGENCE BRIEFING
Security Command Center
TLP:CLEAR
2026-05-04 13:27 UTC

CVE-2026-3296: Everest Forms WordPress Plugin PHP Object Injection via Deserialization

CVE VULNERABILITY | CRITICAL | CVSS 9.8 | CISA KEV

SCC Item ID	SCC-CVE-2026-0118
Type	CVE Vulnerability
CVE ID	CVE-2026-3296
Severity	CRITICAL
CVSS Base Score	9.8
EPSS Score	0.0003 (7th percentile)
KEV Status	Yes — CISA Known Exploited Vulnerability
Affected Products	wpeverest Everest Forms plugin for WordPress, all versions up to and including 3.4.3
Published	2026-05-04T00:00:00Z
Discovery Source	Vulncheck Kev

Executive Summary

A critical vulnerability in the Everest Forms WordPress plugin (versions up to 3.4.3) allows unauthenticated attackers to inject malicious code through public contact forms, which executes when an administrator reviews submissions. The attack requires no login and can result in full server compromise, including remote code execution and unauthorized file access. This vulnerability is confirmed actively exploited and listed on both the CISA and VulnCheck Known Exploited Vulnerabilities catalogs, making immediate remediation a priority. Fixed in version 3.4.4 and later.

Technical Analysis

CVE-2026-3296 (CVSS 9.8, Critical) is a PHP Object Injection vulnerability (CWE-502) affecting the wpeverest Everest Forms WordPress plugin, all versions up to and including 3.4.3. The vulnerability resides in html-admin-page-entries-view.php, which calls PHP's native unserialize() on entry metadata stored in the wp_evf_entrymeta database table without restricting allowed classes via the allowed_classes parameter. Attackers submit serialized PHP object payloads via any public Everest Forms field; sanitize_text_field() does not strip serialization control characters, allowing the payload to persist in the database. Deserialization is triggered server-side when an administrator views form entries, a stored, two-stage attack chain (unauthenticated write, privileged trigger). Exploitation impact depends on available PHP gadget chains in the target environment. In environments with exploitable chains, outcomes include remote code execution

(T1505.003, T1059.004) and exploitation of public-facing services (T1190). Confirmed active exploitation per CISA KEV and VulnCheck KEV. Patch to Everest Forms 3.4.4 or later immediately.

Action Checklist

1. Step 1: Containment. Identify all WordPress instances running Everest Forms 3.0.0-3.4.3 across your environment. If immediate patching is not possible, disable the Everest Forms plugin via the WordPress admin panel or by renaming the plugin directory on the server. Apply WAF rules to block serialized PHP payloads (containing 'O:' patterns) submitted via POST requests to Everest Forms endpoints.
2. Step 2: Detection. Query your database for anomalous entries in wp_evf_entrymeta: look for values beginning with 'O:' or containing serialization patterns (e.g., '{s:', 'a:'). Review web server access logs for POST requests to Everest Forms submission endpoints from unusual or high-volume sources. Check WordPress admin logs or audit plugins for unexpected administrator-triggered page loads of form entry views. Search SIEM for PHP error logs referencing unserialize() or class instantiation errors, which may indicate failed or probing payloads.
3. Step 3: Eradication. Update Everest Forms to version 3.4.4 or the latest available release via the WordPress plugin repository or admin dashboard. Purge existing wp_evf_entrymeta records if any anomalous serialized values are found; do not re-process them. If compromise is suspected, conduct a full file integrity check of the WordPress installation against known-good hashes.
4. Step 4: Recovery. After patching, confirm the installed plugin version in the WordPress admin (Plugins > Installed Plugins). Re-enable the plugin only after verification. Monitor PHP error logs and web server logs for 48-72 hours post-patch for continued exploitation attempts. Validate that no unauthorized administrator accounts, web shells, or modified core files were introduced before the patch was applied.
5. Step 5: Post-Incident. Review your WordPress plugin inventory and establish a policy for patching critical plugins within 24 hours of KEV listing. Evaluate whether WAF rules for serialized PHP payload patterns are part of your standard WordPress hardening baseline. Assess whether PHP's unserialize() usage in other plugins follows safe practices (allowed_classes parameter). This vulnerability exposed a gap in input validation controls at the plugin layer; map it to NIST SP 800-53 SI-10 (Information Input Validation) and SI-3 (Malicious Code Protection) for control improvement tracking.

IR / Forensic Enrichment

Triage Priority	IMMEDIATE
Escalation Criteria	Escalate to senior IR leadership and legal/privacy counsel immediately if the wp_evf_entrymeta forensic query returns confirmed malicious serialized payloads with timestamps predating detection, if any web shell is discovered in /wp-content/uploads/, if unauthorized administrator accounts are found in wp_users, or if the WordPress site processes contact form submissions containing PII subject to GDPR, CCPA, or HIPAA — any of these conditions indicate active compromise with potential regulatory breach notification obligations.

Recovery Notes	Before restoring full plugin operation, confirm via WordPress CLI (`wp plugin get everest-forms --field=version`) that version 3.4.4 or later is installed, and run a complete file integrity check of /wp-content/plugins/everest-forms/ and /wp-content/uploads/ to rule out web shells or backdoors dropped during the exploitation window. Monitor web server access logs and PHP error logs continuously for 72 hours post-patch, specifically alerting on any POST requests to Everest Forms endpoints that return HTTP 200 and any PHP execution originating from the uploads directory. If PII was collected through affected contact forms during the exploitation window, initiate a data exposure assessment in parallel with technical recovery — do not close the incident until both tracks are resolved.
Forensic Artifacts	wp_evf_entrymeta database table — contains the raw serialized PHP object payloads submitted by attackers via Everest Forms public-facing contact forms; records with meta_value matching regex '^O:[0-9]+:' are the attack payload itself and must be preserved as primary forensic evidence before any purge | Web server access logs (Apache /var/log/apache2/access.log or Nginx /var/log/nginx/access.log) — POST requests to wp-admin/admin-ajax.php with evf action parameters represent the submission phase; subsequent GET requests to wp-admin/admin.php?page=evf-entries by an administrator account represent the deserialization trigger moment and are equally critical to the attack timeline | PHP error log (/var/log/php/error.log or path defined in php.ini error_log directive) — failed or probing deserialization attempts from CVE-2026-3296 exploitation appear as 'unserialize(): Error at offset', '__wakeup() not found', or fatal errors referencing unexpected class instantiation tied to Everest Forms class names | Filesystem artifacts in /wp-content/uploads/ — successful PHP object injection leading to RCE commonly results in web shell deployment to the uploads directory (the only world-writable directory in a default WordPress installation); any .php, .phtml, or .php5 file in this path post-dating the first malicious form submission is a high-confidence indicator of successful exploitation | WordPress wp_users and wp_usermeta tables — post-exploitation persistence via CVE-2026-3296 RCE frequently manifests as creation of a rogue administrator account; query wp_usermeta for records where meta_key='wp_capabilities' and meta_value contains 'administrator', joined to wp_users ordered by user_registered, to identify accounts created during or after the attack window

Per-Action IR Details

Step 1: Containment — Identify all WordPress instances running Everest Forms 3.0.0–3.4.3 across your environment. If immediate patching is not possible, disable the Everest Forms plugin via the WordPress admin panel or by renaming the plugin directory on the server. Apply WAF rules to block serialized PHP payloads (containing 'O:' patterns) submitted via POST requests to Everest Forms endpoints.

NIST Phase: Containment

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

Controls: NIST IR-4 (Incident Handling), NIST SI-3 (Malicious Code Protection), NIST SC-7 (Boundary Protection), CIS 4.4 (Implement and Manage a Firewall on Servers), CIS 7.1 (Establish and Maintain a Vulnerability Management Process)

Compensating: For teams without a WAF, use ModSecurity (free, open-source) with the OWASP Core Rule Set — enable rule 933160 (PHP Injection Attack) and add a custom rule: `SecRule ARGS|REQUEST_BODY "@rx O:\d+:" "id:9001,phase:2,deny,log,msg:'Serialized PHP Object in Everest Forms POST'"`. To locate all affected plugin instances across a multi-site environment, run: `find /var/www -path '*/wp-content/plugins/everest-forms/class-evf-install.php' | xargs grep -l 'Version' | xargs grep 'Version'` to extract installed versions. Rename the plugin directory as an emergency disable: `mv /wp-content/plugins/everest-forms /wp-content/plugins/everest-forms.DISABLED`.

Evidence: Before disabling the plugin, preserve the following: (1) Full database dump of the wp_evf_entrymeta and wp_evf_entries tables — `mysqldump -u root -p wordpress wp_evf_entrymeta wp_evf_entries > evf_entries_$(date +%Y%m%d).sql` — to retain serialized payloads for later analysis without re-triggering deserialization. (2) Snapshot of /wp-content/plugins/everest-forms/ directory including all PHP files and timestamps — `tar czf evf_plugin_$(date +%Y%m%d).tar.gz /wp-content/plugins/everest-forms/`. (3) Current web server access logs (Apache: /var/log/apache2/access.log; Nginx: /var/log/nginx/access.log) before any log rotation occurs, specifically capturing POST requests to wp-admin/admin-ajax.php or the Everest Forms REST endpoint (/wp-json/evf/v1/).

Step 2: Detection — Query your database for anomalous entries in wp_evf_entrymeta: look for values beginning with 'O:' or containing serialization patterns (e.g., '{s:', 'a:'). Review web server access logs for POST requests to Everest Forms submission endpoints from unusual or high-volume sources. Check WordPress admin logs or audit plugins for unexpected administrator-triggered page loads of form entry views. Search SIEM for PHP error logs referencing unserialize() or class instantiation errors, which may indicate failed or probing payloads.

NIST Phase: Detection Analysis

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

Controls: NIST IR-5 (Incident Monitoring), NIST AU-6 (Audit Record Review, Analysis, and Reporting), NIST AU-12 (Audit Record Generation), NIST SI-4 (System Monitoring), CIS 8.2 (Collect Audit Logs)

Compensating: Run the following MySQL query directly against the WordPress database to detect serialized PHP object payloads in form submissions: `SELECT entry_id, meta_key, meta_value FROM wp_evf_entrymeta WHERE meta_value REGEXP '^O:[0-9]+:' OR meta_value LIKE '%{s:%' OR meta_value LIKE '%unserialize%';`. For web server log analysis without a SIEM, use GoAccess (free) or run: `grep -E '"POST.*(evf|everest-forms|admin-ajax).*" (200|302)' /var/log/nginx/access.log | awk '{print $1}' | sort | uniq -c | sort -rn | head -30` to identify high-frequency submitting IPs. For PHP error log scanning: `grep -E 'unserialize|class.*not found|__wakeup|__destruct' /var/log/php/error.log | grep -i 'evf\|everest'` to surface failed deserialization attempts indicating attacker probing. Install the WP Activity Log plugin (free tier) if not already present to capture admin-side entry view events.

Evidence: Before querying or alerting, preserve: (1) PHP error log (/var/log/php/error.log or as configured in php.ini) — failed deserialization of malformed objects from probing attempts will appear as 'unserialize(): Error at offset' or fatal errors referencing unexpected class instantiation. (2) WordPress database: full contents of wp_evf_entrymeta where meta_value matches serialization regex — these records ARE the attack payload and must be preserved as forensic evidence before any purge. (3) Web server access logs filtered for POST requests to wp-admin/admin-ajax.php with action=evf_* parameters, and to /index.php?evf-entries — the administrator review action that triggers deserialization is the precise moment of code execution, so admin-side GET/POST requests to entry view pages are equally important to capture as the initial submission POSTs.

Step 3: Eradication — Update Everest Forms to version 3.4.4 or the latest available release via the WordPress plugin repository or admin dashboard. Purge existing wp_evf_entrymeta records if any anomalous serialized values are found — do not re-process them. If compromise is suspected, conduct a full file integrity check of the WordPress installation against known-good hashes.

NIST Phase: Eradication

Reference: NIST 800-61r3 §3.4 — Eradication

Controls: NIST SI-2 (Flaw Remediation), NIST SI-7 (Software, Firmware, and Information Integrity), NIST CM-6 (Configuration Settings), CIS 7.3 (Perform Automated Operating System Patch Management), CIS 7.4 (Perform Automated Application Patch Management)

Compensating: Use WPScan (free, open-source) to verify the post-patch plugin version and detect any additional vulnerable plugins: `wpscan --url https://yoursite.com --enumerate p --plugins-detection aggressive`. For file integrity verification without a commercial tool, use the WordPress CLI: `wp core verify-checksums && wp plugin verify-checksums everest-forms` — this compares installed files against WordPress.org SVN checksums. For broader file integrity, generate a baseline hash manifest post-patch: `find /var/www/html/wp-content/ -type f -name '*.php' -exec md5sum {} \; > /root/wp_php_hashes_$(date +%Y%m%d).txt` and diff against any pre-incident baseline. To safely purge malicious wp_evf_entrymeta records identified in Step 2: `DELETE FROM wp_evf_entrymeta WHERE

meta_value REGEXP '^O:[0-9]+:';` — execute only after the forensic dump from Step 1 is confirmed intact.

Evidence: Before patching, capture: (1) A full listing of all PHP files in /wp-content/plugins/everest-forms/ with modification timestamps — `find /wp-content/plugins/everest-forms/ -name '*.php' -printf '%TY-%Tm-%Td %TH:%TM %p\n' | sort` — to identify any backdoor files written by the exploit's payload during the deserialization attack window. (2) Contents of /wp-content/uploads/ subdirectories for anomalous .php files — a common post-exploitation artifact of PHP object injection leading to RCE is a web shell dropped to the uploads directory (e.g., /wp-content/uploads/2026/03/shell.php). (3) WordPress user table snapshot — `SELECT user_login, user_registered, user_email FROM wp_users ORDER BY user_registered DESC LIMIT 20;` — to detect unauthorized administrator accounts created as a persistence mechanism following successful deserialization exploitation.

Step 4: Recovery — After patching, confirm the installed plugin version in the WordPress admin (Plugins > Installed Plugins). Re-enable the plugin only after verification. Monitor PHP error logs and web server logs for 48–72 hours post-patch for continued exploitation attempts. Validate that no unauthorized administrator accounts, web shells, or modified core files were introduced before the patch was applied.

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), NIST CM-6 (Configuration Settings), CIS 5.1 (Establish and Maintain an Inventory of Accounts), CIS 5.3 (Disable Dormant Accounts)

Compensating: Deploy a free Sigma rule targeting post-exploitation indicators specific to this vulnerability: search web server logs for requests to any .php file within /wp-content/uploads/ (web shell access pattern) using: `grep -E '\.php' /var/log/nginx/access.log | grep 'uploads'`. Monitor for new WordPress admin accounts created after your earliest evidence of malicious form submissions: `SELECT user_login, user_registered FROM wp_users u JOIN wp_usermeta m ON u.ID=m.user_id WHERE m.meta_key='wp_capabilities' AND m.meta_value LIKE '%administrator%' AND u.user_registered > '2026-[earliest_log_date]';`. Use AIDE (free host-based IDS) configured against the WordPress document root to alert on new or modified PHP files during the 72-hour monitoring window: `aide --check`.

Evidence: Before re-enabling the plugin, verify and document: (1) Output of `wp plugin get everest-forms --field=version` (WordPress CLI) confirming version 3.4.4 or later is installed — screenshot or log the command output as evidence of remediation. (2) Results of web shell scan using the free tool NeoPI or a recursive grep: `find /var/www/html/wp-content/uploads/ -name '*.php' -o -name '*.phtml' -o -name '*.php5'` — any PHP file in the uploads directory is a critical finding requiring immediate isolation. (3) WordPress admin audit log (via WP Activity Log or equivalent) showing the administrator account list and any logins that occurred during the exploitation window — specifically, any admin-role accounts created between the first malicious POST submission and the plugin disable timestamp represent confirmed post-exploitation persistence.

Step 5: Post-Incident — Review your WordPress plugin inventory and establish a policy for patching critical plugins within 24 hours of KEV listing. Evaluate whether WAF rules for serialized PHP payload patterns are part of your standard WordPress hardening baseline. Assess whether PHP's unserialize() usage in other plugins follows safe practices (allowed_classes parameter). This vulnerability exposed a gap in input validation controls at the plugin layer — map it to NIST SP 800-53 SI-10 (Information Input Validation) and SI-3 (Malicious Code Protection) for control improvement tracking.

NIST Phase: Post Incident

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

Controls: NIST SI-10 (Information Input Validation), NIST SI-3 (Malicious Code Protection), NIST IR-4 (Incident Handling), NIST IR-8 (Incident Response Plan), NIST RA-3 (Risk Assessment), CIS 7.1 (Establish and Maintain a Vulnerability Management Process), CIS 7.2 (Establish and Maintain a Remediation Process), CIS 2.2 (Ensure Authorized Software is Currently Supported)

Compensating: Subscribe to the CISA KEV RSS feed (https://www.cisa.gov/sites/default/files/feeds/known_exploited_vulnerabilities.json) and automate a daily diff check against your installed WordPress plugin inventory using WP-CLI: `wp plugin list --format=json > installed_plugins.json` then cross-reference against the KEV JSON feed with a simple Python script filtering for 'wordpress' or 'everest' in the

product field. For ongoing PHP deserialization risk across your plugin portfolio, run a static scan using a free tool such as Psalm or grep-based search: `grep -rn 'unserialize(' /var/www/html/wp-content/plugins/ | grep -v 'allowed_classes'` to identify all unserialize() calls lacking the safe allowed_classes parameter — each result is a potential PHP object injection surface requiring code review.

Evidence: For the lessons-learned record, compile: (1) Timeline reconstruction from web server logs showing the first malicious POST to an Everest Forms endpoint, the first administrator page load of form entries (the deserialization trigger), and any subsequent post-exploitation activity — this timeline is required for breach notification assessment if PII was submitted via contact forms. (2) Inventory of all WordPress plugins in your environment with their version histories and KEV match status, generated via `wp plugin list --fields=name,version,status --format=csv` — this documents the asset management gap that allowed a CVSS 9.8 KEV-listed vulnerability to persist. (3) Complete copy of the wp_evf_entrymeta forensic dump from Step 1, retained per your incident record policy — if any submitted form data contained PII (names, emails, messages), this dump may constitute evidence relevant to a data exposure assessment and must be stored securely with chain-of-custody documentation per NIST IR-5 (Incident Monitoring) requirements.

Detection Guidance

Primary indicator: query the WordPress database for wp_evf_entrymeta rows where meta_value begins with 'O:' or contains PHP serialization syntax (e.g., O:[digit]+:, a:[digit]+:{, s:[digit]+:). These patterns in form submission metadata are not expected in legitimate use. Secondary indicator: review web server access logs for POST requests to Everest Forms submission endpoints (typically /wp-json/evf/v1/ or /wp-admin/admin.php?page=evf-entries; consult plugin documentation for exact paths) from external IPs, particularly those with oversized or encoded body content. Tertiary indicator: monitor PHP error logs for unserialize()-related warnings or unexpected class instantiation errors, which may reflect probing or failed gadget chain execution. If a SIEM is in use, build a rule alerting on database writes containing 'O:' to the wp_evf_entrymeta table combined with subsequent admin page reads of form entries. No public IOCs (IPs, hashes, domains) are currently confirmed for this campaign.

Framework Mappings

MITRE-ATTACK

- **T1505.003** — Web Shell
- **T1059.004** — Unix Shell
- **T1190** — Exploit Public-Facing Application

NIST-800-53R5

- **CM-2** — Baseline Configuration
- **CM-7** — Least Functionality
- **SI-3** — Malicious Code Protection
- **SI-4** — System Monitoring
- **SI-7** — Software, Firmware, and Information Integrity
- **CA-8** — Penetration Testing
- **RA-5** — Vulnerability Monitoring and Scanning
- **SC-7** — Boundary Protection

- **SI-2** — Flaw Remediation
- **SI-10** — Information Input Validation
- **IR-5** — Incident Monitoring

OWASP-TOP10-2021

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

CIS-V8

- **16.10** — Apply Secure Design Principles in Application Architectures

ISO-27001-2022

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

NIST-CSF-2

- **DE.AE-08** — Incidents are declared when adverse events meet the defined incident criteria

MITRE ATT&CK Mapping

Technique ID	Technique Name	Tactic
T1505.003	Web Shell	Persistence
T1059.004	Unix Shell	Execution
T1190	Exploit Public-Facing Application	Initial-Access

Sources

Source	URL	Tier
vulncheck_kev	https://nvd.nist.gov/vuln/detail/CVE-2026-3296	T1
CVE-2026-3296 - Tenable	https://www.tenable.com/cve/CVE-2026-3296	T3
CVE-2026-3296: Everest Forms WordPress Plugin RCE Flaw	https://www.sentinelone.com/vulnerability-database/cve-2026-3296/	T3
Everest Forms Plugin Vulnerability (CVE-2026-3296) | Freshy	https://freshysites.com/security-bulletins/everest-forms-plugin-vul...	T3
CVE-2026-33296 Detail - NVD	https://nvd.nist.gov/vuln/detail/CVE-2026-33296	T1
CISA KEV	https://www.cisa.gov/known-exploited-vulnerabilities-catalog	T1

DISCLAIMER

Generated 2026-05-04 13:27 UTC by TJS Security Command Center