INTELLIGENCE BRIEFING
Security Command Center
TLP:CLEAR
2026-06-10 07:28 UTC

CVE-2026-11460: A flaw has been found in Boost Serialization up to 1.91. The impacted element is an unknown function...

CVE VULNERABILITY | HIGH | CVSS 7.3

SCC Item ID	SCC-CVE-2026-0289
Type	CVE Vulnerability
CVE ID	CVE-2026-11460
Severity	HIGH
CVSS Base Score	7.3
EPSS Score	0.0007 (21th percentile)
Affected Products	Boost Serialization library, versions up to and including 1.91
Published	2026-06-07T20:16:39.993
Discovery Source	Nvd

Executive Summary

A publicly exploitable input validation flaw in the Boost Serialization library (versions up to 1.91) has no patch available. The library is widely embedded in C++ applications across enterprise software, meaning the actual attack surface depends on how extensively your vendors and internal developers have adopted it. No official fix has been announced as of 2026-03-04, leaving all affected deployments exposed until vendors or internal teams apply independent mitigations or upgrade to unaffected versions.

Technical Analysis

CVE-2026-11460 affects the Boost Serialization library through version 1.91. The flaw involves improper input validation (CWE-20: Improper Input Validation; CWE-1287: Improper Validation of Specified Type of Input) within an unspecified function in the library. The vulnerability is remotely exploitable and maps to MITRE ATT&CK T1190 (Exploit Public-Facing Application). CVSS base score is 7.3 (High). Public exploitation code may be in circulation, lowering the bar for exploitation. EPSS score is 0.00069 (21st percentile), indicating currently low observed exploitation activity, though the presence of publicly available exploits elevates near-term risk. No official patch has been released as of 2026-03-04. Affected deployments are any C++ application statically or dynamically linking Boost Serialization <= 1.91 that processes externally supplied serialized data.

Action Checklist

1. Step 1: Containment, Inventory all internal applications and third-party software that statically or dynamically link Boost Serialization <= 1.91. Prioritize internet-facing services that accept serialized input. Block or restrict external input paths to affected services at the network perimeter where operationally feasible. Reference CIS 1.1 (Establish and Maintain Detailed Enterprise Asset Inventory) and CIS 2.1 (Establish and Maintain a Software Inventory) to support this inventory effort.
2. Step 2: Detection, Query your software composition analysis (SCA) tool, SBOM, or package manifests for Boost Serialization <= 1.91. In the absence of SCA tooling, scan build artifacts and dependency trees manually. Monitor application and WAF logs for anomalous or malformed serialized data inputs against affected endpoints. Reference NIST AU-2 (Event Logging) and AU-6 (Audit Record Review, Analysis, and Reporting) to ensure relevant log sources are enabled and reviewed. No confirmed IOC patterns are available from published sources at this time.
3. Step 3: Eradication, No official patch exists. Options are: (1) replace or disable the Boost Serialization dependency where technically feasible; (2) implement strict input validation and size/type enforcement at the application layer before data reaches the library; (3) apply virtual patching via WAF or IPS rules with heuristic or behavioral detection for malformed serialized input (test in non-blocking mode first, as no known payload signatures are available). Reference NIST AC-4 (Information Flow Enforcement) for enforcing input boundary controls. Document the no-patch status and the compensating controls applied.
4. Step 4: Recovery, After compensating controls are in place, validate that affected services continue to function correctly and that input filtering has not broken legitimate serialization workflows. Enable enhanced logging on affected services per NIST AU-12 (Audit Record Generation) and monitor for exploitation indicators. Re-run asset and software inventory checks (CIS 1.1, CIS 2.1) to confirm no additional instances were missed. Establish a watch process to detect when an upstream patch is released.
5. Step 5: Post-Incident, This vulnerability exposes a control gap in software composition visibility and third-party library lifecycle management. Implement or mature your SBOM process to enable rapid identification of vulnerable transitive dependencies. Establish a policy for handling unpatched upstream libraries: escalation path, compensating control requirements, and exception documentation. Reference CIS 7.1 (Establish and Maintain a Vulnerability Management Process) and CIS 7.2 (Establish and Maintain a Remediation Process) to formalize this. Where feasible, evaluate dependency risk during procurement and development to reduce future exposure from unmaintained libraries.

IR / Forensic Enrichment

Triage Priority	URGENT
Escalation Criteria	Escalate immediately to senior leadership and legal/compliance if application logs or WAF telemetry show exploitation attempts (boost::archive::archive_exception errors correlated with anomalous external input), if any affected service processes PII, PHI, or financial data triggering breach notification obligations, or if the security team lacks the build-system access needed to complete the Boost Serialization dependency inventory within 48 hours.

Recovery Notes	Following compensating control deployment, conduct functional regression testing using known-good serialized payloads specific to each affected application to confirm that input boundary enforcement has not broken legitimate workflows dependent on Boost Serialization. Maintain enhanced logging (application error logs, WAF logs, network captures on affected service ports) for a minimum of 30 days post-containment, as CVE-2026-11460 has no upstream patch and the exposure window is indefinite. Establish a recurring 30-day review cadence to re-assess compensating control effectiveness, check for newly discovered Boost Serialization instances introduced by software updates, and monitor for any upstream Boost patch release that would enable full eradication.
Forensic Artifacts	Application crash logs and coredumps containing boost::archive::archive_exception stack traces — located at /var/crash/, journald output for the affected service, or Windows Application Event Log (Event ID 1000/1001) — these directly evidence malformed serialized input reaching the vulnerable Boost Serialization parser | Web server or application access logs (Apache /var/log/apache2/access.log, nginx /var/log/nginx/access.log, IIS LogFiles) showing POST or PUT requests to endpoints that consume serialized data, particularly requests with anomalous Content-Length values or non-standard Content-Type headers that may indicate crafted Boost archive payloads | Network packet captures (tcpdump .pcap files) on the service port of affected applications, preserving the raw byte stream of suspicious requests for offline analysis of Boost binary archive header structure and payload anomalies — critical given no confirmed IOC signatures exist yet for CVE-2026-11460 | File system artifacts: sha256 hashes and archived copies of the vulnerable service binaries and their linked Boost Serialization shared objects (libboost_serialization.so.x.xx or boost_serialization-vc*.lib on Windows) — establishes the pre-remediation evidentiary baseline and confirms which library version was in use at incident time | Software composition artifacts: output of ldd, nm, or strings analysis against affected binaries, plus dpkg/rpm query results or Windows installed programs list — documents the confirmed presence of Boost Serialization <= 1.91 linkage as evidentiary support for the scope of impact and the no-patch exception record

Per-Action IR Details

Step 1: Containment — Inventory all internal applications and third-party software that statically or dynamically link Boost Serialization <= 1.91. Prioritize internet-facing services that accept serialized input. Block or restrict external input paths to affected services at the network perimeter where operationally feasible. Reference CIS 1.1 (Establish and Maintain Detailed Enterprise Asset Inventory) and CIS 2.1 (Establish and Maintain a Software Inventory) to drive this discovery.

NIST Phase: Containment

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

Controls: CIS 1.1 (Establish and Maintain Detailed Enterprise Asset Inventory), CIS 2.1 (Establish and Maintain a Software Inventory), NIST AC-4 (Information Flow Enforcement)

Compensating: On Linux build hosts, run: `find /usr/lib /usr/local/lib /opt -name 'libboost_serialization*' 2>/dev/null` and `ldd /path/to/binary | grep boost_serialization` against each service binary to detect dynamic linkage. For static linkage, run `nm -A /path/to/binary | grep -i 'boost.*serial'` or `strings /path/to/binary | grep 'boost::serialization'`. On Windows, use Sysinternals ListDLLs (`listdlls.exe -d boost_serialization`) for running processes. Firewall rules to block external serialized input can be applied immediately via `iptables` or Windows Firewall targeting the specific service port.

Evidence: Before restricting input paths, capture current network connection state to affected services: `ss -tnp` or `netstat -anp` to identify active sessions that may indicate in-progress exploitation. Preserve web server access logs (e.g., Apache `/var/log/apache2/access.log`, nginx `/var/log/nginx/access.log`, IIS `%SystemDrive%\inetpub\logs\LogFiles\`) showing recent POST requests to endpoints that consume serialized input

— these establish pre-containment baseline traffic for later comparison. Capture a process list (`ps auxf` or `tasklist /v`) to document what was running at containment time.

Step 2: Detection — Query your software composition analysis (SCA) tool, SBOM, or package manifests for Boost Serialization <= 1.91. In the absence of SCA tooling, scan build artifacts and dependency trees manually. Monitor application and WAF logs for anomalous or malformed serialized data inputs against affected endpoints. Reference NIST AU-2 (Event Logging) and AU-6 (Audit Record Review, Analysis, and Reporting) to ensure relevant log sources are enabled and reviewed. No confirmed IOC patterns are available from the provided sources at this time.

NIST Phase: Detection Analysis

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

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

Compensating: Without SCA tooling, use `grep -r 'boost/serialization' /path/to/source` across source trees to identify direct includes. For built artifacts, use `dpkg -l | grep -i libboost` (Debian/Ubuntu) or `rpm -qa | grep -i boost-serialization` (RHEL/CentOS) for system-installed packages. For bundled/vendored copies, use `find / -name 'serialization.hpp' -path '*/boost/*' 2>/dev/null`. To detect malformed serialized input at the network layer without a WAF, run Wireshark or `tcpdump -w capture.pcap -i eth0 port ` and inspect for oversized or structurally anomalous Boost archive headers (binary archives begin with a recognizable header byte sequence). Write a Sigma rule targeting application error logs for `boost::archive::archive_exception` stack traces, which surface when malformed input triggers the flaw.

Evidence: Capture application stderr and crash logs for `boost::archive::archive_exception` or `std::bad_alloc` entries, which indicate the serialization parser encountered malformed input — on Linux these appear in `/var/log/syslog`, journald (`journalctl -u `), or application-specific log paths. Preserve WAF logs if present, filtering on HTTP 400/500 responses to endpoints accepting serialized payloads. Capture coredumps if the service has crashed (`/var/crash/`, `/cores/`, or configured `kernel.core_pattern` path) as these may contain the malicious payload in memory at the point of fault. For Windows services, collect Application Event Log entries (Event ID 1000/1001 — Application Error/Fault Bucket) tied to the affected process.

Step 3: Eradication — No official patch exists. Options are: (1) replace or disable the Boost Serialization dependency where technically feasible; (2) implement strict input validation and size/type enforcement at the application layer before data reaches the library; (3) apply virtual patching via WAF or IPS rules targeting malformed serialized payloads at ingress points. Reference NIST AC-4 (Information Flow Enforcement) for enforcing input boundary controls. No patch ID is available to cite; document the no-fix status and the compensating controls applied.

NIST Phase: Eradication

Reference: NIST 800-61r3 §3.4 — Eradication

Controls: NIST AC-4 (Information Flow Enforcement), CIS 7.2 (Establish and Maintain a Remediation Process), CIS 4.4 (Implement and Manage a Firewall on Servers)

Compensating: For option (2) without commercial WAF: implement input size limits and content-type enforcement at the reverse proxy layer using nginx (`client_max_body_size`, `if ($content_type !~ 'expected/type')`) or Apache (`LimitRequestBody`, `mod_security` with a free CRS ruleset). For option (3), write a YARA rule matching known Boost binary archive magic bytes followed by anomalously large type-length fields and deploy via a network IDS (Suricata with `file.data` matching on the service port). Document the compensating control, the CVE-2026-11460 no-fix status, and the interim risk acceptance in writing — this satisfies the exception documentation requirement under CIS 7.2.

Evidence: Before modifying the application or applying virtual patches, preserve a copy of the vulnerable binary (`sha256sum` the binary and archive it with its linked libraries) to establish a pre-eradication forensic baseline. Capture the full dependency manifest (`ldd -v /path/to/binary` or `dumpbin /dependents binary.exe`) so post-eradication verification can confirm the Boost Serialization linkage has been severed or version-replaced. If the service has processed any suspicious input, extract and preserve those payloads from WAF logs or packet captures before

patching — they constitute evidence of potential exploitation attempts.

Step 4: Recovery — After compensating controls are in place, validate that affected services continue to function correctly and that input filtering has not broken legitimate serialization workflows. Enable enhanced logging on affected services per NIST AU-12 (Audit Record Generation) and monitor for exploitation indicators. Re-run asset and software inventory checks (CIS 1.1, CIS 2.1) to confirm no additional instances were missed. Establish a watch process to detect when an upstream patch is released.

NIST Phase: Recovery

Reference: NIST 800-61r3 §3.5 — Recovery

Controls: NIST AU-12 (Audit Record Generation), CIS 1.1 (Establish and Maintain Detailed Enterprise Asset Inventory), CIS 2.1 (Establish and Maintain a Software Inventory), NIST AU-2 (Event Logging)

Compensating: Validate legitimate serialization workflows by replaying a set of known-good serialized payloads (sourced from integration tests or captured from pre-incident traffic) against the now-protected endpoint and verifying expected HTTP 200 responses and correct application behavior — document the test cases and results. Set up a lightweight watch process for upstream Boost patch releases using a cron job that queries the Boost GitHub releases API (`curl -s https://api.github.com/repos/boostorg/boost/releases/latest`) daily and alerts via email or Slack webhook when a new release tag appears. Note: this URL should be validated by the operator — it is provided as a structural reference, not a verified canonical endpoint.

Evidence: After compensating controls are applied, capture a second network baseline (`ss -tnp`, access log snapshot) and compare against the pre-containment baseline to confirm the attack surface reduction is measurable. Archive the final compensating control configuration files (nginx config, WAF ruleset, iptables rules) with timestamps as evidence of the remediation state — these serve as the audit trail required by NIST AU-12 and support post-incident review. Log the re-inventory scan results and diff them against the Step 1 inventory to document any newly discovered Boost Serialization instances found during recovery.

Step 5: Post-Incident — This vulnerability exposes a control gap in software composition visibility and third-party library lifecycle management. Implement or mature your SBOM process to enable rapid identification of vulnerable transitive dependencies. Establish a policy for handling unpatched upstream libraries: escalation path, compensating control requirements, and exception documentation. Reference CIS 7.1 (Establish and Maintain a Vulnerability Management Process) and CIS 7.2 (Establish and Maintain a Remediation Process) to formalize this. Where feasible, evaluate dependency risk during procurement and development to reduce future exposure from unmaintained libraries.

NIST Phase: Post Incident

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

Controls: 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: For SBOM generation without commercial tooling, integrate `syft` (free, open-source) into CI/CD pipelines to auto-generate CycloneDX or SPDX SBOMs for every build artifact — `syft packages dir:/path/to/project -o cyclonedx-json > sbom.json`. Cross-reference generated SBOMs against the OSV (Open Source Vulnerabilities) database using `osv-scanner --sbom sbom.json` to flag vulnerable transitive dependencies including Boost Serialization. Establish a policy document that explicitly addresses the no-upstream-fix scenario (as with CVE-2026-11460) with defined timelines for compensating control review and executive risk acceptance sign-off.

Evidence: Produce a lessons-learned document that records: (1) the count and names of internal applications found to link Boost Serialization <= 1.91, (2) the time elapsed between CVE-2026-11460 disclosure and initial inventory completion, and (3) which applications lacked SBOM coverage that would have accelerated discovery. This gap analysis is the primary post-incident artifact for this event and directly informs the policy and tooling improvements cited above. Retain the full incident timeline, all compensating control configurations, and the no-fix status acknowledgment from the Boost maintainer as the exception documentation record.

Detection Guidance

No confirmed IOCs (IP addresses, domains, file hashes, or network signatures) are available from published sources for CVE-2026-11460 at this time. Detection should focus on asset exposure and anomalous input behavior. First, use software composition analysis tools or SBOM inventory to identify any application linking Boost Serialization <= 1.91; this is the primary exposure signal. Second, monitor application logs on services that process externally supplied serialized objects for unexpected deserialization errors, type mismatch exceptions, or abnormal payload sizes, which may indicate exploit attempts. Third, review WAF and IDS/IPS logs for attempts to send malformed or oversized payloads to endpoints served by affected applications. NIST AU-2 (Event Logging) and AU-6 (Audit Record Review, Analysis, and Reporting) govern the logging posture required to support this detection. CIS 8.2 (Collect Audit Logs) provides the foundational safeguard for ensuring log coverage is in place. Monitor threat intelligence feeds for emerging behavioral signatures or exploitation patterns as attack activity develops.

Framework Mappings

MITRE-ATTACK

- **T1190** — Exploit Public-Facing Application

NIST-800-53R5

- **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
- **IR-5** — Incident Monitoring

OWASP-TOP10-2021

- **A03:2021** — Injection

CIS-V8

- **16.10** — Apply Secure Design Principles in Application Architectures
- **7.3** — Perform Automated Operating System Patch Management
- **7.4** — Perform Automated Application Patch Management

ISO-27001-2022

- **A.8.26** — Application security requirements
- **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
T1190	Exploit Public-Facing Application	Initial-Access

Sources

Source	URL	Tier
nvd	https://nvd.nist.gov/vuln/detail/CVE-2026-11460	T1
CVE-2026-11460 - Exploits & Severity - Feedly	https://feedly.com/cve/CVE-2026-11460	T3
CVE-2026-11460 in Serialization	https://vuldb.com/cve/CVE-2026-11460	T3
CVE-2026-41460 Detail - NVD	https://nvd.nist.gov/vuln/detail/CVE-2026-41460	T1
CVE-2026-37460: FRRouting (FRR) DoS Vulnerability - SentinelOne	https://www.sentinelone.com/vulnerability-database/cve-2026-37460/	T3

Generated 2026-06-10 07:28 UTC by TJS Security Command Center