INTELLIGENCE BRIEFING
Security Command Center
TLP:CLEAR
2026-06-10 07:26 UTC

Proto6: Six protobuf.js Vulnerabilities Enable RCE and Supply Chain Compromise in Node.js Ecosystems

CVE VULNERABILITY | HIGH | CVSS 7.5

SCC Item ID	SCC-CVE-2026-0284
Type	CVE Vulnerability
CVE ID	CVE-2026-44289, CVE-2026-44290, CVE-2026-44291, CVE-2026-44292, CVE-2026-44294, CVE-2026-44295
Severity	HIGH
CVSS Base Score	7.5
EPSS Score	0.0006 (19th percentile)
Affected Products	protobufjs <=7.5.5 and 8.0.0-8.0.1; protobufjs-cli <=1.2.0 and 2.0.0-2.0.1; Google Cloud client libraries dependent on protobuf.js; Baileys (WhatsApp Web API library); Node.js applications using protobuf deserialization or code generation. Patched in protobufjs 7.5.6, 8.0.2 and protobufjs-cli 1.2.1, 2.0.2.
Published	2026-06-10T01:08:35
Discovery Source	Rss

Executive Summary

Six vulnerabilities in protobuf.js, a JavaScript serialization library with tens of millions of weekly downloads, enable remote code execution and supply chain compromise across Node.js applications, Google Cloud SDKs, and CI/CD pipelines. Organizations running unpatched versions (protobufjs ≤7.5.5 or 8.0.0-8.0.1) face risk of arbitrary code execution through prototype pollution and static code injection, with effects affecting AI/ML infrastructure and automated build systems via transitive dependencies. Patches are available in protobufjs 7.5.6 and 8.0.2; immediate dependency audits and upgrades are the priority action.

Technical Analysis

The Proto6 vulnerability cluster affects protobufjs ≤7.5.5 and 8.0.0-8.0.1, plus protobufjs-cli ≤1.2.0 and 2.0.0-2.0.1. The aggregate CVSS base score is 7.5 (high severity). Individual CVE CVSS scores have not been independently verified at analysis time; reported ranges suggest some may exceed 8.0, but confirmation against NVD is required before operationalizing severity claims for individual identifiers. The highest-priority items are:

prototype pollution (CWE-1321) enabling manipulation of the JavaScript object prototype chain during deserialization, and static code injection (CWE-94) exploitable during schema-driven code generation. Supporting weaknesses include uncontrolled resource consumption (CWE-400), improper input validation (CWE-20), and protection mechanism failure (CWE-693). MITRE ATT&CK coverage includes T1195.001 and T1195.002 (supply chain compromise via dependency and software), T1059.007 (JavaScript execution), T1190 (exploit public-facing application), T1574 (hijack execution flow), T1499.004 (application exhaustion via CWE-400), and T1552.004 (private key exposure in affected environments). Attack surface spans any Node.js service performing protobuf deserialization or invoking protobufjs-cli code generation, Google Cloud client libraries that depend on protobuf.js, the Baileys WhatsApp Web API library, and CI/CD pipelines consuming generated protobuf schemas. Patched versions: protobufjs 7.5.6 and 8.0.2; protobufjs-cli 1.2.1 and 2.0.2. Confidence in specific CVE number accuracy is medium; NVD and CISA KEV confirmation was not available at analysis time. Verify against NVD and the official protobufjs GitHub advisory before operationalizing CVE references.

Action Checklist

1. Step 1: Discovery and Assessment. Run 'npm ls protobufjs' and 'npm ls protobufjs-cli' across all Node.js application repositories and CI/CD pipeline configurations to identify deployments of protobufjs ≤7.5.5, 8.0.0-8.0.1, protobufjs-cli ≤1.2.0, or 2.0.0-2.0.1. Prioritize internet-facing services and pipelines that accept external protobuf-serialized input or invoke schema-based code generation. Isolate or disable code generation workflows using protobufjs-cli until patched. (Supports NIST AC-4, Information Flow Enforcement; CIS 1.1, Establish and Maintain Detailed Enterprise Asset Inventory)
2. Step 2: Detection. Query SIEM and application logs for anomalous Node.js process spawning, unexpected child processes forked from protobuf deserialization paths, and schema ingestion from external or untrusted sources. Hunt for prototype pollution indicators: log entries showing '__proto__', 'constructor', or 'prototype' keys in deserialized input payloads. Review dependency-track or SCA tooling alerts for the affected version ranges. Check Google Cloud SDK dependency trees for transitive protobuf.js inclusion. (Supports NIST AU-6, Audit Record Review, Analysis, and Reporting; CIS 8.2, Collect Audit Logs; D3-SFA, System File Analysis)
3. Step 3: Eradication. Upgrade protobufjs to 7.5.6 (for 7.x deployments) or 8.0.2 (for 8.x deployments) and protobufjs-cli to 1.2.1 or 2.0.2 respectively via 'npm update protobufjs protobufjs-cli'. For transitive dependencies through Google Cloud client libraries or Baileys, update the parent packages and confirm the resolved protobufjs version with 'npm ls protobufjs'. Rebuild and redeploy all artifacts that bundled vulnerable versions. Invalidate and regenerate any protobuf-generated code produced by vulnerable protobufjs-cli versions. (Supports NIST SI-2, Software Updates; CIS 7.3, Perform Automated Operating System Patch Management; CIS 7.4, Perform Automated Application Patch Management)
4. Step 4: Recovery. After patching, re-run SCA scans to confirm no remaining vulnerable versions in the dependency tree, including nested transitive dependencies. Validate that regenerated protobuf schema code contains no injected payloads by diffing against known-good output or reviewing generated files for unexpected executable content. Re-enable CI/CD pipeline code generation steps only after confirming patched CLI versions are in use. Monitor application logs for 72 hours post-deployment for anomalous deserialization behavior or unexpected process execution. (Supports NIST AU-6, Audit Record Review, Analysis, and Reporting; D3-SFA, System File Analysis)
5. Step 5: Post-Incident. This vulnerability cluster exposes gaps in third-party dependency governance and supply chain visibility. Implement software composition analysis (SCA) tooling in CI/CD pipelines to block

builds when critical vulnerabilities are detected in direct or transitive dependencies. Establish a documented policy for protobuf schema ingestion: restrict schema sources to trusted, version-controlled inputs and reject schemas from untrusted or external origins at the application layer. Review access controls on code generation pipeline steps to enforce least privilege. (Supports NIST AC-6, Least Privilege; CIS 2.1, Establish and Maintain a Software Inventory; 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 CISO and legal/compliance if forensic analysis of protobufjs-cli-generated code files reveals injected payloads (confirming supply chain compromise via CVE-2026-44294 or CVE-2026-44295), if process execution logs show confirmed child process spawning from Node.js deserialization paths indicating active RCE, or if affected applications process PII, PHI, or payment card data triggering breach notification obligations under applicable regulations.
Recovery Notes	After patching to protobufjs 7.5.6 or 8.0.2 and protobufjs-cli 1.2.1 or 2.0.2, validate that 'npm ls protobufjs --all' shows no residual vulnerable version resolutions anywhere in the dependency tree, including transitive paths through Google Cloud client libraries and Baileys. All protobuf-generated '.js' and '.ts' files previously produced by vulnerable CLI versions must be deleted and regenerated from scratch — do not retain or redeploy tainted generated code even if no injected content is visually apparent, as static injection from CVE-2026-44294 or CVE-2026-44295 may be subtle. Monitor application logs and OS process execution events for a minimum of 72 hours post-redeployment, specifically watching for Node.js spawning unexpected child processes or deserialization exceptions referencing protobuf object parsing paths, before declaring the incident closed.
Forensic Artifacts	npm package-lock.json and yarn.lock files from all affected repositories: these establish which exact resolved version of protobufjs was in use at time of exposure, including transitive resolution paths through Google Cloud client libraries or Baileys, and serve as the primary evidence record for scope determination | protobufjs-cli generated output files (pbjs/pbts .js and .ts artifacts): any files generated by protobufjs-cli ≤1.2.0 or 2.0.0–2.0.1 are potential carriers of statically injected malicious code (CVE-2026-44294, CVE-2026-44295); preserve originals with SHA-256 hashes before overwriting and diff against clean regenerated output for 'eval', 'require("child_process")', 'spawn', or dynamic Function() constructor patterns | OS process execution logs showing Node.js child process chains: on Linux, auditd EXECVE records or /var/log/syslog entries; on Windows, Security Event Log Event ID 4688 (Process Creation) filtered for node.exe or node as parent process — unexpected children such as sh, bash, cmd.exe, or powershell.exe spawned from protobuf deserialization code paths are the primary RCE indicator for CVE-2026-44289 through CVE-2026-44292 | Web server and application access logs (nginx access.log, Apache access_log, Node.js application JSON logs) covering the full exposure window: specifically filter for POST requests to protobuf deserialization endpoints containing payload bodies with '__proto__', 'constructor', or 'prototype' keys, which are the direct indicators of prototype pollution exploitation attempts against the affected protobufjs versions | CI/CD pipeline build logs (GitHub Actions run logs, Jenkins console output, GitLab CI job traces) for all builds that executed 'pbjs' or 'pbts' during the exposure window: these establish whether potentially compromised code generation ran in automated pipelines and whether tainted generated artifacts were deployed to production or distributed as published packages

Per-Action IR Details

Step 1: Containment — Run 'npm ls protobufjs' and 'npm ls protobufjs-cli' across all Node.js application repositories and CI/CD pipeline configurations to identify deployments of protobufjs ≤7.5.5, 8.0.0–8.0.1, protobufjs-cli ≤1.2.0, or 2.0.0–2.0.1. Prioritize internet-facing services and pipelines that accept external protobuf-serialized input or invoke schema-based code generation. Isolate or disable code generation workflows using protobufjs-cli until patched. (Supports NIST AC-4 — Information Flow Enforcement; CIS 1.1 — Establish and Maintain Detailed Enterprise Asset Inventory)

NIST Phase: Containment

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

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

Compensating: For teams without a centralized SBOM or SCA platform, run 'find . -name package-lock.json -exec grep -l protobufjs {} \;' recursively across all code repositories to enumerate affected projects. Follow with 'cat package-lock.json | python3 -c "import sys,json; deps=json.load(sys.stdin).get(\"packages\",{}); [print(k,v[\"version\"]) for k,v in deps.items() if \"protobufjs\" in k]"' to extract resolved versions. Immediately disable or comment out CI/CD pipeline stages that invoke 'pbjs' or 'pbts' (protobufjs-cli binaries) by setting them to a no-op command until patched versions are confirmed.

Evidence: Before isolating any pipeline or service, preserve: (1) the full 'npm ls protobufjs --all' output for each affected repository showing the complete resolved dependency tree including transitive paths through Google Cloud client libraries or Baileys; (2) a snapshot of 'package.json' and 'package-lock.json' (or 'yarn.lock') files from all affected projects to establish the vulnerable version baseline for post-incident review; (3) CI/CD pipeline job logs (GitHub Actions workflow run logs, Jenkins build console output, or GitLab CI job traces) showing which pipeline stages executed 'pbjs' or 'pbts' code generation commands and when.

Step 2: Detection — Query SIEM and application logs for anomalous Node.js process spawning, unexpected child processes forked from protobuf deserialization paths, and schema ingestion from external or untrusted sources. Hunt for prototype pollution indicators: log entries showing '__proto__', 'constructor', or 'prototype' keys in deserialized input payloads. Review dependency-track or SCA tooling alerts for the affected version ranges. Check Google Cloud SDK dependency trees for transitive protobuf.js inclusion. (Supports NIST AU-6 — Audit Record Review, Analysis, and Reporting; CIS 8.2 — Collect Audit Logs; D3-SFA — System File Analysis)

NIST Phase: Detection Analysis

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

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

Compensating: Without a SIEM, deploy Sysmon on Windows Node.js hosts with Event ID 1 (Process Create) and filter for node.exe spawning unexpected child processes (cmd.exe, sh, powershell.exe, bash) — these indicate successful RCE via prototype pollution or code injection through protobuf deserialization. On Linux, use auditd with '-a always,exit -F arch=b64 -S execve -F ppid=$(pgrep -f node)' to capture child process execution from Node.js parent processes. For prototype pollution payload detection in HTTP request bodies, use a Sigma rule targeting application access logs: grep -E '(__proto__|constructor\.prototype|Object\.prototype)' /var/log/app/*.log or equivalent web server access logs where protobuf-serialized input is ingested via REST or gRPC endpoints.

Evidence: Capture before pivoting to eradication: (1) Node.js application stdout/stderr logs and any structured JSON application logs from the period of exposure, specifically entries containing deserialized protobuf object keys '__proto__', 'constructor', or 'prototype' which indicate a prototype pollution attempt against CVE-2026-44291 or related vulns; (2) OS-level process execution logs (Linux: /var/log/audit/audit.log EXECVE records; Windows: Security Event Log Event ID 4688 Process Creation) filtered for node.exe or node as parent process with unexpected child processes; (3) network captures (Wireshark/tcpdump) or web server access logs (nginx access.log, Apache access_log) showing POST requests to protobuf deserialization endpoints with anomalous payload sizes or Content-Type: application/x-protobuf from untrusted source IPs; (4) Google Cloud SDK and Baileys dependency resolution logs if

those libraries are in use, to establish whether the vulnerable protobufjs version was reached transitively.

Step 3: Eradication — Upgrade protobufjs to 7.5.6 (for 7.x deployments) or 8.0.2 (for 8.x deployments) and protobufjs-cli to 1.2.1 or 2.0.2 respectively via 'npm update protobufjs protobufjs-cli'. For transitive dependencies through Google Cloud client libraries or Baileys, update the parent packages and confirm the resolved protobufjs version with 'npm ls protobufjs'. Rebuild and redeploy all artifacts that bundled vulnerable versions. Invalidate and regenerate any protobuf-generated code produced by vulnerable protobufjs-cli versions. (Supports NIST SI-4 — no mapped control from knowledge base for patch management specifically; 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: CIS 7.3 (Perform Automated Operating System Patch Management), CIS 7.4 (Perform Automated Application Patch Management)

Compensating: For teams without automated patch management tooling, perform a controlled manual upgrade: run 'npm install protobufjs@7.5.6' or 'npm install protobufjs@8.0.2' (version-pinned, not range-resolved) and commit the updated 'package-lock.json' to version control for audit trail. For transitive exposure through Google Cloud client libraries, run 'npm update @google-cloud/[package-name]' and immediately validate with 'npm ls protobufjs' that no 7.x ≤7.5.5 or 8.0.0–8.0.1 resolution remains in the tree. Treat all '.js' files previously generated by 'pbjs' or 'pbts' from protobufjs-cli ≤1.2.0 or 2.0.0–2.0.1 as potentially tainted — delete and regenerate them from scratch using the patched CLI, then diff the output against a clean reference to detect static code injection artifacts from CVE-2026-44294 or CVE-2026-44295.

Evidence: Before executing the upgrade, preserve: (1) a copy of all '.js' and '.ts' files previously generated by the vulnerable protobufjs-cli (pbjs/pbts output), stored in an isolated directory or forensic image, as these may contain injected code payloads attributable to CVE-2026-44294 or CVE-2026-44295 and constitute primary forensic evidence of supply chain compromise; (2) the full npm audit output ('npm audit --json > pre-patch-audit.json') capturing the vulnerability state before remediation for regulatory and post-incident documentation; (3) container image digests or build artifact hashes (SHA-256) of any Docker images or deployment packages that included the vulnerable protobufjs version, to support downstream notification if those artifacts were distributed externally.

Step 4: Recovery — After patching, re-run SCA scans to confirm no remaining vulnerable versions in the dependency tree, including nested transitive dependencies. Validate that regenerated protobuf schema code contains no injected payloads by diffing against known-good output or reviewing generated files for unexpected executable content. Re-enable CI/CD pipeline code generation steps only after confirming patched CLI versions are in use. Monitor application logs for 72 hours post-deployment for anomalous deserialization behavior or unexpected process execution. (Supports NIST AU-6 — Audit Record Review, Analysis, and Reporting; D3-SFA — System File Analysis)

NIST Phase: Recovery

Reference: NIST 800-61r3 §3.5 — Recovery

Controls: NIST AU-6 (Audit Record Review, Analysis, and Reporting)

Compensating: Without a commercial SCA platform, use 'npm audit --audit-level=high' post-patch to verify clean status and pipe output to a file for documentation ('npm audit --json > post-patch-audit.json'). For validating regenerated protobuf schema files, run 'diff -u old-generated/ new-generated/' between the pre-patch and post-patch pbjs/pbts output directories and grep the diff for patterns consistent with injected code: 'grep -E "(eval|require|child_process|exec|spawn|Function\(|setTimeout.*Function)" new-generated/*.js'. For 72-hour post-deployment monitoring without a SIEM, configure application-level logging to capture all deserialization exceptions and use a cron job to alert on node.exe/node child process spawning: 'watch -n 60 "ps auxf | grep node"' on Linux, or Sysmon Event ID 1 with a filter on node parent PID on Windows.

Evidence: Before re-enabling CI/CD code generation: (1) retain the diff output between pre-patch and post-patch pbjs/pbts generated files as evidence of what the patched CLI produces versus potentially tainted prior output — any

unexpected 'eval', 'require("child_process")', or dynamic function construction patterns in the pre-patch files should be escalated as confirmed supply chain injection from CVE-2026-44294 or CVE-2026-44295; (2) capture 'npm ls protobufjs --all' output post-patch as a clean baseline artifact for audit documentation; (3) collect application performance and error logs from the 72-hour monitoring window, specifically Node.js unhandled exception stack traces referencing protobuf deserialization paths, as these may indicate lingering exploitation attempts against the patched surface.

Step 5: Post-Incident — This vulnerability cluster exposes gaps in third-party dependency governance and supply chain visibility. Implement software composition analysis (SCA) tooling in CI/CD pipelines to block builds when critical vulnerabilities are detected in direct or transitive dependencies. Establish a documented policy for protobuf schema ingestion: restrict schema sources to trusted, version-controlled inputs and reject schemas from untrusted or external origins at the application layer. Review access controls on code generation pipeline steps to enforce least privilege. (Supports NIST AC-6 — Least Privilege; CIS 2.1 — Establish and Maintain a Software Inventory; 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), CIS 2.1 (Establish and Maintain a Software Inventory), 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 teams without commercial SCA tooling, integrate the free 'npm audit' command as a mandatory CI/CD gate using a pre-commit hook or pipeline step that fails the build on high/critical findings: 'npm audit --audit-level=high || exit 1'. Add OWASP Dependency-Check (free, open source) to the pipeline to catch transitive vulnerabilities in protobufjs and similar serialization libraries. For schema ingestion policy enforcement without an enterprise WAF, implement a Node.js middleware layer that validates all incoming protobuf schema sources against a static allowlist of trusted file paths or registry package names before invoking protobufjs parsing — reject and log any schema referencing external URLs or paths outside the project's version-controlled schema directory. Restrict CI/CD pipeline service accounts that execute 'pbjs'/'pbts' to read-only access on schema source directories using OS-level file permissions (chmod 444 on schema .proto files).

Evidence: For post-incident lessons learned documentation: (1) compile the full timeline of vulnerable protobufjs versions present in production, derived from package-lock.json commit history in version control ('git log --all -p -- package-lock.json | grep protobufjs'), to determine how long the organization was exposed before detection; (2) collect the pre-patch npm audit JSON files from all affected repositories as the authoritative record of the vulnerability scope; (3) document any protobuf-generated code files that showed anomalous content in the diff analysis from Step 4, with file hashes, as evidence of whether supply chain compromise via CVE-2026-44294 or CVE-2026-44295 was confirmed or ruled out.

Detection Guidance

Primary detection targets are prototype pollution injection attempts and unauthorized code generation activity. In application logs, search for deserialized input payloads containing the strings '__proto__', 'constructor.prototype', or 'prototype' as object keys; these indicate active prototype pollution exploitation attempts against CWE-1321. In Node.js process telemetry (EDR or eBPF-based), alert on unexpected child process spawning from node processes that handle protobuf deserialization, particularly shells (sh, bash, cmd.exe, powershell) or network-initiating processes. For CI/CD pipelines, monitor for protobufjs-cli invocations against schema files sourced from external URLs or recently modified untrusted paths. In SIEM, correlate: (1) inbound protobuf data from external IPs to Node.js services, (2) subsequent anomalous outbound connections or new process creation within the same application context, (3) file write events in application directories following deserialization activity. For Google Cloud environments, review Cloud Audit Logs for unexpected API

calls following SDK operations that invoke protobuf serialization. SCA and dependency scanning tools (Dependabot, Snyk, OWASP Dependency-Check) should alert on protobufjs versions within the affected ranges. No confirmed public IOC hashes or IPs are available at analysis time; behavioral detection is the primary mechanism. CVE identifiers are medium-confidence pending NVD publication; detection rules should key on version ranges and behavioral patterns rather than CVE string matching.

Indicators of Compromise

Type	Value	Context	Confidence
URL	`no confirmed public IOCs available at analysis time`	No verified IP, domain, hash, or URL indicators have been published for active exploitation of Proto6 vulnerabilities. Detection should rely on behavioral and version-based indicators as described in detection_guidance.	LOW

Framework Mappings

MITRE-ATTACK

- **T1195.001** — Compromise Software Dependencies and Development Tools
- **T1195.002** — Compromise Software Supply Chain
- **T1499.004** — Application or System Exploitation
- **T1552.004** — Private Keys
- **T1574** — Hijack Execution Flow
- **T1059.007** — JavaScript
- **T1190** — Exploit Public-Facing Application

NIST-800-53R5

- **CM-7** — Least Functionality
- **SA-9** — External System Services
- **SR-3** — Supply Chain Controls and Processes
- **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
- **SC-5** — Denial-of-Service Protection
- **SI-10** — Information Input Validation
- **SR-2** — Supply Chain Risk Management Plan

CIS-V8

- **13.8** — Deploy a Network Intrusion Prevention Solution

- **16.10** — Apply Secure Design Principles in Application Architectures
- **7.3** — Perform Automated Operating System Patch Management
- **7.4** — Perform Automated Application Patch Management
- **15.1** — Establish and Maintain an Inventory of Service Providers

OWASP-TOP10-2021

- **A03:2021** — Injection

ISO-27001-2022

- **A.8.26** — Application security requirements
- **A.8.8** — Management of technical vulnerabilities
- **A.5.21** — Managing information security in the ICT supply chain
- **A.5.23** — Information security for use of cloud services

NIST-CSF-2

- **GV.SC-01** — Cybersecurity supply chain risk management program

SOC2-TSC

- **CC9.2** — Manages risks associated with vendors and business partners

MITRE ATT&CK Mapping

Technique ID	Technique Name	Tactic
T1195.001	Compromise Software Dependencies and Development Tools	Initial-Access
T1195.002	Compromise Software Supply Chain	Initial-Access
T1499.004	Application or System Exploitation	Impact
T1552.004	Private Keys	Credential-Access
T1574	Hijack Execution Flow	Persistence
T1059.007	JavaScript	Execution
T1190	Exploit Public-Facing Application	Initial-Access

Sources

Source	URL	Tier
Security News	https://thehackernews.com/2026/06/six-proto6-vulnerabilities-in.html	T3

Source	URL	Tier
May 2026 Patch Tuesday: Updates and Analysis | CrowdStrike	https://www.crowdstrike.com/en-us/blog/patch-tuesday-analysis-may-2...	T3
Known Exploited Vulnerabilities Catalog | CISA	https://www.cisa.gov/known-exploited-vulnerabilities-catalog	T1
CVE-2026-34342 - CVE Record	https://www.cve.org/CVERecord?id=CVE-2026-34342	T3
K000159076: Quarterly Security Notification (February 2026)	https://my.f5.com/manage/s/article/K000159076	T3
NVD	https://nvd.nist.gov/vuln/detail/CVE-2026-44289, CVE-2026-44290, CV...	T1
Google Security Advisory	https://chromereleases.googleblog.com/	T1

Generated 2026-06-10 07:26 UTC by TJS Security Command Center