INTELLIGENCE BRIEFING
Security Command Center
TLP:CLEAR
2026-05-27 14:06 UTC

14 npm/PyPI/AI Supply-Chain Threats Today (2026-05-26): Critical Worms, Parse Server DoS, and AI RCEs

THREAT CAMPAIGN | **CRITICAL** | CVSS 9.0

SCC Item ID	SCC-CAM-2026-0371
Type	Threat Campaign
CVE ID	CVE-2026-46421, CVE-2026-46412, CVE-2026-45758, CVE-2026-47138, CVE-2026-8723, CVE-2026-46679, CVE-2026-45783, CVE-2026-46374, CVE-2026-45804, CVE-2026-46517, CVE-2026-46497, CVE-2026-46372, CVE-2026-46490, CVE-2026-46625
Severity	CRITICAL
CVSS Base Score	9.0
Affected Products	npm: @cap-js/sqlite, @cap-js/postgres, @cap-js/db-service, @beproduct/nestjs-auth, qs, @libp2p/gossipsub, @libp2p/kad-dht, samlify, js-cookie; PyPI: guardrails-ai, SQLFluff, Diffusers, Crawlee for Python; AI/ML: lmdeploy, SillyTavern; Server: Parse Server
Published	2026-05-26
Discovery Source	Gemini

Executive Summary

On May 26, 2026, fourteen vulnerabilities were disclosed across npm, PyPI, and AI/ML package ecosystems, affecting components used in enterprise application development, authentication, networking, and AI model deployment. The most severe issues include worm-capable credential-harvesting flaws in SAP CAP framework libraries, remote code execution in Hugging Face Diffusers and the lmdeploy AI deployment framework, and denial-of-service vulnerabilities in the widely-used qs query-string library and Parse Server. Organizations building software with these dependencies, running AI/ML pipelines, or deploying Node.js and Python applications are at elevated risk of supply-chain compromise, credential theft, and unauthorized code execution. Note: individual CVE-to-package mappings carry low confidence pending NVD or OSV confirmation as of analysis time. Verify all package-to-CVE mappings against upstream security advisories (npmjs.com, PyPI.org, vendor GitHub repositories) before initiating emergency patching.

Technical Analysis

Fourteen CVEs (CVE-2026-46421, CVE-2026-46412, CVE-2026-45758, CVE-2026-47138, CVE-2026-8723, CVE-2026-46679, CVE-2026-45783, CVE-2026-46374, CVE-2026-45804, CVE-2026-46517, CVE-2026-46497, CVE-2026-46372, CVE-2026-46490, CVE-2026-46625) were disclosed 2026-05-26 across three ecosystems. CVE-to-package mappings are unconfirmed in NVD/OSV as of analysis time, treat all specific mappings as LOW confidence pending authoritative confirmation.

Cluster 1, Worm-Capable Credential Harvesting (CWE-287, CWE-1104): @cap-js/sqlite, @cap-js/postgres, @cap-js/db-service (SAP CAP framework), and @beproduct/nestjs-auth. Attack vector likely involves dependency confusion or malicious package substitution enabling lateral propagation across dependent build environments. MITRE: T1195.002 (Supply Chain Compromise: Compromise Software Dependencies), T1552 (Unsecured Credentials), T1078 (Valid Accounts).

Cluster 2, Denial of Service (CWE-400): qs (widely deployed query-string parser, npm) and Parse Server. Attack vector is malformed input triggering unbounded resource consumption. MITRE: T1499 (Endpoint Denial of Service).

Cluster 3, Remote Code Execution (CWE-94, CWE-502): Diffusers (Hugging Face, PyPI) and lmdeploy. Likely attack pathway is unsafe deserialization or model-loading via pickle or equivalent during model artifact ingestion. MITRE: T1059 (Command and Scripting Interpreter), T1190 (Exploit Public-Facing Application).

Cluster 4, Additional vulnerabilities: @libp2p/gossipsub and @libp2p/kad-dht (networking layer, CWE-400/CWE-918 plausible), samlify (SAML authentication, CWE-287), js-cookie (CWE-94 plausible), SQLFluff (SQL linter, PyPI), Crawlee for Python (web-scraping, CWE-918 plausible), SillyTavern (LLM UI, CWE-94/CWE-502 plausible).

CVSS base 9.0 represents the highest base score among the 14 disclosed CVEs; individual vulnerability CVSS scores vary. EPSS scores not yet available pending NVD publication. No CISA KEV listing as of analysis time. Patch status unconfirmed, check respective package registries (npmjs.com, PyPI) and upstream GitHub repositories for patched versions before remediation.

Action Checklist

1. Step 1: Containment, Audit your dependency manifests (package.json, requirements.txt, pyproject.toml) immediately for presence of @cap-js/sqlite, @cap-js/postgres, @cap-js/db-service, @beproduct/nestjs-auth, qs, @libp2p/gossipsub, @libp2p/kad-dht, samlify, js-cookie, SQLFluff, Diffusers, Crawlee, lmdeploy, or SillyTavern. Freeze dependency installation in CI/CD pipelines (lock files, private registry mirrors) until patched versions are confirmed. If Parse Server is internet-facing, place it behind a WAF with input validation rules immediately, aligns with NIST SC-7 (Boundary Protection) and CIS 4.4 (Implement and Manage a Firewall on Servers).
2. Step 2: Detection, Query SIEM/log aggregator for anomalous outbound connections from build agents or application servers that use affected packages, focusing on credential exfiltration indicators (T1552). For qs and Parse Server DoS exposure, check web server and application logs for requests with deeply nested or malformed query strings (e.g., repeated bracket notation, excessive key counts). For Diffusers and lmdeploy RCE risk, audit model-loading operations in ML pipeline logs for unexpected subprocess spawning or file writes outside designated model directories. Enable audit logging per NIST AU-2 (Event Logging) and AU-6 (Audit Record Review, Analysis, and Reporting) if not already active on build and inference infrastructure. Use NIST SI-7 (Software, Firmware, and Information Integrity) monitoring to track for unexpected modification of package lock files, site-packages directories, or node_modules trees.

3. Step 3: Eradication, Upgrade affected packages to patched versions once confirmed by upstream maintainers via npmjs.com advisories, PyPI security advisories, or OSV.dev. Until patched versions are published, pin dependencies to the last known-clean version in lock files and validate integrity via checksum (NIST SI-7, Software, Firmware, and Information Integrity). For samlify, audit all SAML assertion validation logic and enforce strict schema validation. For Diffusers and lmdeploy, disable pickle-based model loading where feasible and restrict model sources to trusted, signed repositories. Apply credential rotation (NIST IA-4) for any secrets accessible to services using @cap-js or @beproduct/nestjs-auth, as credential harvesting is the stated worm vector. Reference CIS 7.3 and CIS 7.4 for automated patch management process.

4. Step 4: Recovery, After upgrading, re-run dependency integrity checks using npm audit, pip-audit, or equivalent tooling and confirm zero findings for the listed packages. Re-enable CI/CD pipelines incrementally, starting with lowest-risk build environments. Monitor application and infrastructure logs for 72 hours post-remediation for any residual anomalous behavior consistent with post-exploitation (unexpected account creation, lateral movement, new scheduled tasks). Validate that Parse Server and qs-dependent endpoints return expected responses to edge-case inputs without resource exhaustion. Apply account monitoring (NIST AU-2) to verify no new local accounts were created during the exposure window. Reference NIST IR-4 (Incident Handling) and NIST AU-11 (Audit Record Retention) to preserve logs for post-incident review.

5. Step 5: Post-Incident, This cluster exposes gaps in software supply-chain visibility and AI/ML dependency governance. Implement or validate: (a) software composition analysis (SCA) in CI/CD pipelines per NIST SA-12 (Supply Chain Risk Management) and CIS 2.1 (Establish and Maintain a Software Inventory); (b) a private package mirror or registry with allowlist enforcement to prevent dependency confusion attacks per CIS 2.3 (Address Unauthorized Software); (c) formal model artifact signing and provenance validation for AI/ML pipelines using frameworks such as SLSA or Sigstore, given the RCE vectors in Diffusers and lmdeploy; (d) MFA on all package registry publishing accounts per CIS 6.5 (Require MFA for Administrative Access), a prerequisite to account takeover supply-chain attacks; (e) periodic review of third-party authentication libraries (samlify, nestjs-auth) against NIST IA-8 (Identification and Authentication, Non-Organizational Users). Document findings in your risk register per NIST RA-3 (Risk Assessment).

IR / Forensic Enrichment

Triage Priority	IMMEDIATE
Escalation Criteria	Escalate to CISO and legal/privacy counsel immediately if forensic review of outbound network logs confirms credential exfiltration from @cap-js or @beproduct/nestjs-auth components, or if samlify SAML assertion logs show anomalous authentication successes during the exposure window, as either condition may trigger breach notification obligations under GDPR, CCPA, or HIPAA depending on data classification of affected services.

Recovery Notes	After patching all 14 packages, maintain elevated monitoring for a minimum of 72 hours on build agents, inference servers running lmdeploy/Diffusers, and any Parse Server instances previously internet-facing, watching specifically for new outbound connections to non-approved IPs (residual @cap-js worm C2), unexpected Python subprocess chains (post-RCE persistence), and anomalous SAML authentication events (post-samlify exploitation). Rotate all database credentials referenced by @cap-js/postgres and @cap-js/sqlite service configurations regardless of whether confirmed compromise is established — the worm-capable credential-harvesting design of these packages means the exposure window itself justifies rotation as a precautionary measure. Validate recovery completeness by confirming 'npm audit' and 'pip-audit' return zero findings for all 14 named packages across every affected repository before restoring full CI/CD pipeline operations.
Forensic Artifacts	Outbound network connection logs from build agents during the period @cap-js/db-service, @cap-js/sqlite, @cap-js/postgres, or @beproduct/nestjs-auth were installed — filter on Sysmon EventID 3 or firewall egress logs for POST requests to non-approved external IPs during npm install or application startup, as the worm-capable credential-harvesting mechanism activates at module load time | Web server access logs (nginx/Apache) for Parse Server endpoints, filtered for requests with URI query strings matching the pattern '\[.*\]\[.*\]\[.*\]' repeated more than 10 levels deep or containing more than 50 unique keys — the specific input pattern that triggers CVE-2026-46497 qs prototype pollution DoS and CVE-2026-46679 Parse Server DoS | ML inference server process trees and file creation events for lmdeploy and Hugging Face Diffusers processes — specifically auditd or Sysmon EventID 11 (FileCreate) records showing writes to paths outside '/root/.cache/huggingface/' or the configured model directory, and EventID 1 (ProcessCreate) records showing python spawning bash, sh, or curl as a child process during model loading (CVE-2026-45758 RCE vector) | IdP/SP SAML authentication logs for the 30-day window prior to 2026-05-26 covering all applications using samlify — specifically authentication events where the assertion NameID or attribute values contain XML comment nodes or namespace prefix manipulations, which are the signature-wrapping bypass artifacts left by exploitation of CVE-2026-46421 | Installed package contents and integrity hashes for all 14 affected packages across all environments — preserve 'sha256sum' of each package's main entry-point file (e.g., node_modules/@cap-js/db-service/lib/index.js, site-packages/diffusers/__init__.py) compared against the official npmjs.com and PyPI published checksums to detect if a compromised or typosquatted version was installed rather than the legitimate vulnerable release

Per-Action IR Details

Step 1: Containment — Audit your dependency manifests (package.json, requirements.txt, pyproject.toml) immediately for presence of @cap-js/sqlite, @cap-js/postgres, @cap-js/db-service, @beproduct/nestjs-auth, qs, @libp2p/gossipsub, @libp2p/kad-dht, samlify, js-cookie, guardrails-ai, SQLFluff, Diffusers, Crawlee, lmdeploy, or SillyTavern. Freeze dependency installation in CI/CD pipelines (lock files, private registry mirrors) until patched versions are confirmed. If Parse Server is internet-facing, place it behind a WAF with input validation rules immediately — aligns with NIST SC-7 (Boundary Protection) and CIS 4.4 (Implement and Manage a Firewall on Servers).

NIST Phase: Containment

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

Controls: NIST SC-7 (Boundary Protection), NIST CM-3 (Configuration Change Control), NIST SI-2 (Flaw Remediation), CIS 4.4 (Implement and Manage a Firewall on Servers), CIS 2.3 (Address Unauthorized Software)

Compensating: Run 'grep -r "@cap-js\|samlify\|js-cookie\|guardrails-ai\|lmdeploy\|diffusers\|crawlee\|sqlfluff" /path/to/repo --include="*.json" --include="*.txt" --include="*.toml"' across all repos. Freeze npm installs by setting 'npm config set ignore-scripts true' and committing package-lock.json with 'npm ci' enforced. For Parse Server, deploy

ModSecurity CRS with rule 941100 (SQL injection) and 942100 (deep nesting) via Apache/nginx as a zero-cost WAF layer. Two-person teams should split: one audits manifests, one locks CI/CD pipeline config files immediately.

Evidence: Before freezing CI/CD, snapshot current package-lock.json, yarn.lock, requirements.txt, and pyproject.toml with 'sha256sum' checksums — these are your baseline for detecting any tampering the worm-capable @cap-js components may have already introduced. Capture npm cache directory (~/.npm) and pip cache (~/.cache/pip) contents, as a compromised @cap-js/db-service or guardrails-ai package may have written malicious artifacts to the local cache during prior installs. Document all environment variables accessible to build agents, as the @cap-js credential-harvesting vector specifically targets secrets exposed in the build environment.

Step 2: Detection — Query SIEM/log aggregator for anomalous outbound connections from build agents or application servers that use affected packages, focusing on credential exfiltration indicators (T1552). For qs and Parse Server DoS exposure, check web server and application logs for requests with deeply nested or malformed query strings (e.g., repeated bracket notation, excessive key counts). For Diffusers and lmdeploy RCE risk, audit model-loading operations in ML pipeline logs for unexpected subprocess spawning or file writes outside designated model directories. Enable audit logging per NIST AU-2 (Event Logging) and AU-6 (Audit Record Review, Analysis, and Reporting) if not already active on build and inference infrastructure. Use D3-SFA (System File Analysis) to monitor for unexpected modification of package lock files, site-packages directories, or node_modules trees.

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), NIST SI-4 (System Monitoring), CIS 8.2 (Collect Audit Logs)

Compensating: Without a SIEM, deploy Sysmon with EventID 3 (NetworkConnect) filtering on build agent PIDs to catch @cap-js credential exfiltration outbound to non-approved IPs; use SwiftOnSecurity's Sysmon config as a baseline. For qs/Parse Server DoS detection, run 'grep -E "(\[.*\]){10,}|%5B.*%5D.*%5B" /var/log/nginx/access.log' to find deeply nested bracket notation requests. For Diffusers/lmdeploy RCE, add a Falco rule triggering on python or python3 processes spawning bash/sh children outside '/opt/model-runner' or equivalent designated inference directories. Monitor node_modules and site-packages with 'find /app/node_modules -newer /app/package-lock.json -type f' on a 15-minute cron to catch unexpected file modifications.

Evidence: Capture nginx/Apache access logs covering the 30 days prior to disclosure (2026-04-26 through 2026-05-26) and filter for Parse Server endpoints receiving POST bodies with query strings containing more than 20 nested bracket pairs — this is the specific qs prototype pollution/DoS fingerprint. Export Sysmon EventID 11 (FileCreate) and EventID 3 (NetworkConnect) records for any process whose executable path resolves under node_modules/@cap-js or site-packages/guardrails — these packages' worm mechanism would manifest as network callbacks during module load. For lmdeploy/Diffusers RCE, collect ML inference server logs for pickle.load() or torch.load() calls referencing model files not in the approved model registry, and any os.system() or subprocess.Popen() invocations in the inference process tree (visible via 'auditd' rule: '-a always,exit -F arch=b64 -S execve -F ppid=$(pgrep lmdeploy)').

Step 3: Eradication — Upgrade affected packages to patched versions once confirmed by upstream maintainers via npmjs.com advisories, PyPI security advisories, or OSV.dev. Until patched versions are published, pin dependencies to the last known-clean version in lock files and validate integrity via checksum (NIST SI-7, Software, Firmware, and Information Integrity). For samlify, audit all SAML assertion validation logic and enforce strict schema validation. For Diffusers and lmdeploy, disable pickle-based model loading where feasible and restrict model sources to trusted, signed repositories. Apply D3-CRO (Credential Rotation) for any secrets accessible to services using @cap-js or @beproduct/nestjs-auth, as credential harvesting is the stated worm vector. Reference CIS 7.3 and CIS 7.4 for automated patch management process.

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 IA-5 (Authenticator Management), CIS 7.3 (Perform Automated Operating System Patch Management), CIS 7.4 (Perform Automated Application Patch Management)

Compensating: Validate package integrity without a commercial SCA tool using 'npm audit --json > audit_$(date +%F).json' and 'pip-audit --output json -o pyaudit_$(date +%F).json'; compare hashes of installed packages against OSV.dev API ('curl https://api.osv.dev/v1/query -d {"package":{"name":"samlify","ecosystem":"npm"}}') for each affected package. For samlify XML signature bypass, run 'grep -r "validatePostResponse\|validateRedirectResponse" /app/src' to locate all SAML validation call sites and confirm each passes the strict schema option. Rotate all database credentials (PostgreSQL, SQLite connection strings) referenced in @cap-js/postgres and @cap-js/sqlite service configs — enumerate them via 'grep -r "connectionString\|DATABASE_URL\|DB_PASS" /app --include="*.env" --include="*.json"' and treat all as compromised.

Evidence: Before removing any affected package version, preserve a forensic copy of the installed package directory (e.g., '/app/node_modules/@cap-js/db-service') using 'tar czf cap-js-db-service-forensic-$(date +%F).tar.gz /app/node_modules/@cap-js/db-service' — the worm mechanism is embedded in the package code itself and this copy is needed for malware analysis. For samlify, export all SAML authentication events from your IdP/SP logs for the 30 days prior to 2026-05-26, filtering on assertion validation responses, to detect any XML signature wrapping attacks that may have succeeded before eradication. For @beproduct/nestjs-auth credential harvesting, retrieve all outbound HTTP/HTTPS requests made by the application during the exposure window from proxy or firewall logs, filtering on destinations not in your approved vendor list — exfiltrated credentials would appear as POST requests to attacker-controlled endpoints.

Step 4: Recovery — After upgrading, re-run dependency integrity checks using npm audit, pip-audit, or equivalent tooling and confirm zero findings for the listed packages. Re-enable CI/CD pipelines incrementally, starting with lowest-risk build environments. Monitor application and infrastructure logs for 72 hours post-remediation for any residual anomalous behavior consistent with post-exploitation (unexpected account creation, lateral movement, new scheduled tasks). Validate that Parse Server and qs-dependent endpoints return expected responses to edge-case inputs without resource exhaustion. Apply D3-LAM (Local Account Monitoring) to verify no new local accounts were created during the exposure window. Reference NIST IR-4 (Incident Handling) and NIST AU-11 (Audit Record Retention) to preserve logs for post-incident review.

NIST Phase: Recovery

Reference: NIST 800-61r3 §3.5 — Recovery

Controls: NIST IR-4 (Incident Handling), NIST AU-11 (Audit Record Retention), NIST CP-10 (System Recovery and Reconstitution), NIST SI-2 (Flaw Remediation), CIS 7.1 (Establish and Maintain a Vulnerability Management Process)

Compensating: For local account creation monitoring without EDR, run on all affected hosts: 'getent passwd | awk -F: "$3 >= 1000 {print}" > /tmp/accounts_post_$(date +%F).txt' and diff against a pre-incident baseline. For scheduled task persistence (a likely @cap-js worm follow-on), run 'crontab -l -u $(whoami); cat /etc/cron* /var/spool/cron/crontabs/*' and 'schtasks /query /fo LIST /v > schtasks_post.txt' on Windows build agents. Validate Parse Server recovery by replaying the malformed qs input pattern ('curl -X GET "http://parseserver/parse/classes/TestClass?where=%7B%22a%22%3A%7B%22b%22%3A%7B%22c%22%3A...%7D%7D%7D"' with 50-level nesting) and confirming the server returns a 400 error without hanging — this directly tests that CVE-2026-46497 and the qs DoS are resolved.

Evidence: Before re-enabling CI/CD pipelines, collect a full snapshot of all Linux scheduled jobs ('crontab -l', '/etc/cron.d/', '/etc/cron.daily/', '/var/spool/cron/') and Windows Task Scheduler exports ('schtasks /query /xml') on all build agents — the @cap-js worm's persistence mechanism would most likely manifest here. Archive all application server logs (nginx, node.js stdout, gunicorn/uvicorn for Python ML services) with 'logrotate -f' before re-enabling pipelines to preserve evidence of any lmdeploy or Diffusers RCE activity that occurred during the exposure window. Capture 'netstat -anp' and 'ss -tulnp' output from inference servers running lmdeploy or SillyTavern immediately before cutover to document any unexpected listening ports opened by a successful RCE.

Step 5: Post-Incident — This cluster exposes gaps in software supply-chain visibility and AI/ML dependency governance. Implement or validate: (a) software composition analysis (SCA) in CI/CD pipelines per NIST SA-12 (Supply Chain Risk Management) and CIS 2.1 (Establish and Maintain a Software Inventory); (b) a

private package mirror or registry with allowlist enforcement to prevent dependency confusion attacks per CIS 2.3 (Address Unauthorized Software); (c) formal model artifact signing and provenance validation for AI/ML pipelines, given the RCE vectors in Diffusers and lmdeploy; (d) MFA on all package registry publishing accounts per CIS 6.5 (Require MFA for Administrative Access), a prerequisite to account takeover supply-chain attacks; (e) periodic review of third-party authentication libraries (samlify, nestjs-auth) against NIST IA-8 (Identification and Authentication, Non-Organizational Users). Document findings in your risk register per NIST RA-3 (Risk Assessment).

NIST Phase: Post Incident

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

Controls: NIST SA-12 (Supply Chain Risk Management), NIST IA-8 (Identification and Authentication, Non-Organizational Users), NIST RA-3 (Risk Assessment), NIST SI-2 (Flaw Remediation), CIS 2.1 (Establish and Maintain a Software Inventory), CIS 2.3 (Address Unauthorized Software), CIS 6.5 (Require MFA for Administrative Access)

Compensating: Implement free SCA by integrating 'pip-audit' and 'npm audit' as mandatory CI/CD pipeline steps using GitHub Actions or GitLab CI YAML — block merges on HIGH/CRITICAL findings. For private registry mirroring on zero budget, deploy Verdaccio (npm) and devpi (PyPI) as Docker containers on an internal host; configure '.npmrc' with 'registry=http://your-verdaccio-host:4873' and 'pip.conf' with 'index-url = http://your-devpi-host/root/pypi/+simple/' to enforce allowlist. For AI/ML model provenance, adopt Sigstore/cosign (free, CNCF-backed) to sign model artifacts: 'cosign sign --key cosign.key ghcr.io/yourorg/model:tag' and verify at load time — this directly addresses the Diffusers/lmdeploy unsigned model RCE vector. Add YARA rules targeting pickle magic bytes in non-approved directories as a detection layer for future Diffusers/lmdeploy-class attacks.

Evidence: Document the full timeline of @cap-js, samlify, and lmdeploy vulnerability exposure windows — from earliest package version in use to patch application — for risk register entry and potential breach notification analysis if PII-touching services used @beproduct/nestjs-auth or samlify during the window. Retain all npm audit, pip-audit, and OSV.dev query outputs generated during this incident as evidence of due diligence per NIST AU-11 (Audit Record Retention). Produce a dependency graph (via 'npm ls --json > dep-graph.json' and 'pipdeptree --json > py-dep-graph.json') showing which production applications transitively depended on the 14 affected packages — this graph is the primary artifact for your supply-chain risk register update per NIST RA-3 (Risk Assessment) and SA-12 (Supply Chain Risk Management).

Detection Guidance

Detection priorities by cluster:

1. Worm/Credential Harvesting (@cap-js/*, @beproduct/nestjs-auth): Monitor build servers and Node.js application hosts for outbound DNS and HTTP/S requests to domains not in an approved allowlist, originating from npm install or application startup processes. Alert on any postinstall script execution in node_modules for the affected packages. Query EDR telemetry for child process spawning from node or npm processes that invoke shell commands, curl, wget, or PowerShell. MITRE T1195.002 and T1552 detection: look for access to environment variable files (.env), credential store directories, or AWS metadata endpoints (169.254.169.254) from build pipeline processes.

2. DoS, qs and Parse Server: In web application firewall or reverse proxy logs (nginx, Apache, cloud load balancer), search for HTTP requests where the query string contains more than 1,000 characters, deeply nested bracket notation (e.g., a[b][c][d]...), or repeated identical keys at high volume from a single source IP. Parse Server: monitor application error logs for stack traces referencing query parsing failures at elevated rates.

3. RCE, Diffusers and lmdeploy (T1059, T1190): In ML inference infrastructure, alert on: (a) Python process spawning unexpected child processes (bash, sh, cmd.exe) during model load operations; (b) file writes to /tmp,

/var/tmp, or user home directories by the inference service process; (c) unexpected outbound network connections from inference workers. If using containerized inference (Docker/Kubernetes), enable seccomp and AppArmor profiles and alert on policy violations during model loading. SIEM query example (validate field names for your specific SIEM platform before deployment; common field names include process_name, child_process_name, parent_process_path, but these vary by SIEM implementation): process_name IN ('python', 'python3') AND child_process_name IN ('bash', 'sh', 'curl', 'wget') AND parent_process_path CONTAINS ('diffusers', 'lmdeploy').

4. samlify (CWE-287, authentication bypass): Review IdP and application authentication logs for SAML assertions that were accepted without a valid signature, or for authentication events from users with no corresponding MFA step. Alert on authentication bursts from unexpected geographic sources.

All detections should feed SIEM with alerting per NIST AU-6 and AU-12. NIST SI-7 (System File Analysis) applies to package integrity monitoring; NIST AU-2 applies to post-exploitation account activity detection.

Indicators of Compromise

Type	Value	Context	Confidence
HASH	`not available`	No confirmed malicious package hashes published as of analysis time. Monitor OSV.dev and npm/PyPI security advisories for package-specific SHA digests of compromised versions.	LOW
URL	`not available`	No confirmed C2 or exfiltration URLs associated with the worm-capable packages as of analysis time. Monitor outbound connections from build agents for anomalous destinations.	LOW

Framework Mappings

MITRE-ATTACK

- **T1499** — Endpoint Denial of Service
- **T1195.002** — Compromise Software Supply Chain
- **T1552** — Unsecured Credentials
- **T1078** — Valid Accounts
- **T1059** — Command and Scripting Interpreter
- **T1190** — Exploit Public-Facing Application

NIST-800-53R5

- **SC-5** — Denial-of-Service Protection
- **CM-7** — Least Functionality
- **SA-9** — External System Services
- **SR-3** — Supply Chain Controls and Processes

- **SI-7** — Software, Firmware, and Information Integrity
- **AC-2** — Account Management
- **AC-6** — Least Privilege
- **IA-2** — Identification and Authentication (Organizational Users)
- **IA-5** — Authenticator Management
- **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
- **SA-4** — Acquisition Process
- **SI-10** — Information Input Validation
- **IA-8** — Identification and Authentication (Non-Organizational Users)

CIS-V8

- **13.8** — Deploy a Network Intrusion Prevention Solution
- **16.4** — Establish and Manage an Inventory of Third-Party Software Components
- **16.10** — Apply Secure Design Principles in Application Architectures
- **6.3** — Require MFA for Externally-Exposed Applications
- **6.4** — Require MFA for Remote Network Access
- **6.5** — Require MFA for Administrative Access
- **13.4** — Perform Traffic Filtering Between Network Segments

OWASP-TOP10-2021

- **A06:2021** — Vulnerable and Outdated Components
- **A03:2021** — Injection
- **A07:2021** — Identification and Authentication Failures
- **A08:2021** — Software and Data Integrity Failures
- **A10:2021** — Server-Side Request Forgery (SSRF)

SOC2-TSC

- **CC6.1** — The entity implements logical access security software, infrastructure, and architectures over protected information assets

HIPAA-SECURITY

- **164.312(d)** — Person or Entity Authentication

ISO-27001-2022

- **A.8.8** — Management of technical vulnerabilities
- **A.5.23** — Information security for use of cloud services

MITRE ATT&CK Mapping

Technique ID	Technique Name	Tactic
T1499	Endpoint Denial of Service	Impact
T1195.002	Compromise Software Supply Chain	Initial-Access
T1552	Unsecured Credentials	Credential-Access
T1078	Valid Accounts	Defense-Evasion
T1059	Command and Scripting Interpreter	Execution
T1190	Exploit Public-Facing Application	Initial-Access

Sources

Source	URL	Tier
gemini	https://www.reddit.com/r/cybersecurity/comments/1d4x1y0/14_npm_pypi...	T3
CVE-2026-23000 Detail - NVD	https://nvd.nist.gov/vuln/detail/cve-2026-23000	T1
May 2026 Patch Tuesday: Updates and Analysis - CrowdStrike	https://www.crowdstrike.com/en-us/blog/patch-tuesday-analysis-may-2...	T3
CVE-2026-20926 - Microsoft Security Response Center (MSRC)	https://msrc.microsoft.com/update-guide/en-US/advisory/CVE-2026-20926	T1
CVE-2026-33824: Remote Code Execution in Windows IKEv2 - thezdi	https://www.thezdi.com/blog/2026/4/22/cve-2026-33824-remote-code-ex...	T3
NVD	https://nvd.nist.gov/vuln/detail/CVE-2026-46421, CVE-2026-46412, CV...	T1

Generated 2026-05-27 14:06 UTC by TJS Security Command Center