INTELLIGENCE BRIEFING
Security Command Center
TLP:CLEAR
2026-04-28 06:34 UTC

CVE-2026-6977: A security vulnerability has been detected in vanna-ai vanna up to 2.0.2. The affected element is an...

CVE VULNERABILITY | HIGH | CVSS 7.3

SCC Item ID	SCC-CVE-2026-0083
Type	CVE Vulnerability
CVE ID	CVE-2026-6977
Severity	HIGH
CVSS Base Score	7.3
EPSS Score	0.0004 (11th percentile)
Affected Products	vanna-ai/vanna up to version 2.0.2 (Legacy Flask API component)
Published	2026-04-25T11:16:19.103
Discovery Source	Nvd

Executive Summary

CVE-2026-6977 is a high-severity authorization bypass vulnerability in the Legacy Flask API component of vanna-ai/vanna, affecting all versions through 2.0.2. Unauthenticated remote attackers can bypass access controls to interact with protected API endpoints, potentially accessing or manipulating AI query functionality and underlying data. A public exploit has been disclosed. No official patch exists at this time.

Technical Analysis

CVE-2026-6977 (CVSS 7.3, High) affects vanna-ai/vanna through version 2.0.2, specifically the Legacy Flask API component. The vulnerability combines CWE-266 (Incorrect Privilege Assignment) and CWE-285 (Improper Authorization), allowing unauthenticated or low-privileged remote attackers to bypass access controls on protected API routes. Exploitation maps to MITRE ATT&CK T1190 (Exploit Public-Facing Application) and T1078 (Valid Accounts, abuse of improperly enforced authorization). A public exploit has been disclosed prior to any vendor-issued patch. EPSS score is 0.038%, indicating low current exploitation activity, but the public exploit disclosure elevates practical risk. No CISA KEV listing at this time. No CVSSv3 vector string was published by the vendor.

Action Checklist

1. Containment: Identify all production deployments of vanna-ai/vanna at or below version 2.0.2. If the Legacy Flask API component is enabled and internet-facing, isolate it immediately behind network controls or take it offline until a fix is available. Block external access to the Flask API endpoints at the perimeter or WAF layer.
2. Detection: Query package inventory and SBOM records for vanna-ai/vanna <= 2.0.2. Review web server and application logs for unexpected or unauthenticated requests to Flask API routes, particularly those that should require authentication. Look for T1190 indicators: anomalous 200-series responses to endpoints that should return 401/403 for unauthenticated callers. Check for unusual query patterns consistent with T1078 access abuse.
3. Eradication: No vendor-issued patch exists at time of publication. Disable or remove the Legacy Flask API component if it is not operationally required. If the component is required, enforce authentication at the reverse proxy or gateway layer as a compensating control. Monitor the official vanna-ai/vanna repository for any patch release.
4. Recovery: After applying compensating controls or a future patch, verify that all API endpoints return appropriate 401/403 responses to unauthenticated test requests. Confirm access control enforcement through authenticated and unauthenticated test cases. Monitor application logs for continued anomalous access attempts for at least 30 days post-remediation.
5. Post-Incident: This vulnerability exposes a gap in authorization enforcement within AI/ML tooling that may lack the same security review maturity as production web services. Review all AI/ML libraries and frameworks in your environment for similar Legacy API components with insufficient access controls. Add vanna-ai/vanna to your software composition analysis (SCA) scanning pipeline and establish a process for tracking vendor responsiveness during coordinated disclosure.

IR / Forensic Enrichment

Triage Priority	IMMEDIATE
Escalation Criteria	Escalate to incident commander and legal/compliance if web server logs confirm any unauthenticated HTTP 200 responses to vanna Flask API endpoints prior to containment, as successful exploitation may constitute unauthorized access to AI query functionality and connected data sources, triggering breach notification obligations under GDPR, CCPA, or HIPAA depending on the data classifications reachable by the vanna-connected database.
Recovery Notes	After enforcing authentication at the reverse proxy or gateway layer, conduct HTTP-level validation by replaying unauthenticated requests to every documented vanna Flask API route (particularly `/api/v0/generate_sql`, `/api/v0/run_sql`, and `/api/v0/train`) and confirming 401/403 responses before restoring internet-facing access. Review the connected database query logs for the full exploitation window to determine whether any attacker-driven SQL queries were executed against sensitive data, as the AI query generation capability means exploitation could have resulted in arbitrary data retrieval beyond the API layer itself. Maintain enhanced access log monitoring on the vanna service for a minimum of 30 days post-remediation, watching specifically for resumed probing from source IPs identified in the pre-containment anomaly review.

Forensic Artifacts	Web server access logs (nginx /var/log/nginx/access.log or Apache /var/log/apache2/access.log): filter for HTTP 200 responses on vanna Flask API routes (e.g., /api/v0/generate_sql, /api/v0/run_sql, /api/v0/train) from requests carrying no Authorization header or valid session cookie — these represent confirmed authorization bypass exploitation attempts. | vanna application-level logs (Python werkzeug/Flask request logs if info/debug logging enabled): these may contain the full SQL query strings that the vanna AI model generated in response to attacker-supplied natural language inputs, revealing the scope of data the attacker caused to be queried from the connected database. | Connected database query logs: PostgreSQL pg_stat_statements or general query log, MySQL general_log, or equivalent — cross-reference query timestamps against anomalous vanna API access windows to identify AI-generated SQL queries executed on behalf of unauthenticated callers during the exploitation window. | Python package installation records on affected hosts: pip freeze output, requirements.txt, pyproject.toml, and any container image layer history confirming vanna-ai/vanna version <=2.0.2 was present and when it was deployed — establishes exposure window for impact scoping. | Network flow data or WAF logs covering the Flask service port during the exposure window: source IP addresses, request volumes, and URI paths targeting vanna API routes — identifies whether exploitation was opportunistic scanning or targeted, and whether data exfiltration volumes are consistent with bulk query responses being returned to the attacker.

Per-Action IR Details

Containment — Identify all production deployments of vanna-ai/vanna at or below version 2.0.2. If the Legacy Flask API component is enabled and internet-facing, isolate it immediately behind network controls or take it offline until a fix is available. Block external access to the Flask API endpoints at the perimeter or WAF layer.

NIST Phase: Containment

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

Controls: NIST IR-4 (Incident Handling), NIST SC-7 (Boundary Protection), CIS 4.4 (Implement and Manage a Firewall on Servers), CIS 4.5 (Implement and Manage a Firewall on End-User Devices), CIS 2.2 (Ensure Authorized Software is Currently Supported)

Compensating: Run `pip show vanna` or `pip list | grep vanna` on all application hosts to confirm installed version. For container environments: `docker inspect | grep -i vanna` or review requirements.txt/pyproject.toml. Block Flask API routes at the host firewall using `iptables -I INPUT -p tcp --dport -j DROP` or equivalent `ufw deny `. If behind nginx/Apache, add a `deny all` directive to the location block matching the vanna API route prefix (commonly `/api/` or `/vanna/`). For teams using Cloudflare or similar WAF, create a firewall rule matching the URI path to the Flask endpoint and set action to Block.

Evidence: Before isolating, capture a full snapshot of active network connections to the Flask service: `ss -tnp | grep ` and `netstat -an | grep `. Export web server access logs (nginx: `/var/log/nginx/access.log`; Apache: `/var/log/apache2/access.log`) covering at minimum 30 days prior. Capture all HTTP requests targeting the vanna Flask API routes with response codes — specifically filter for 200-series responses on routes that should be authentication-gated. Preserve these logs to immutable storage before any network change disrupts the flow.

Detection — Query package inventory and SBOM records for vanna-ai/vanna <= 2.0.2. Review web server and application logs for unexpected or unauthenticated requests to Flask API routes, particularly those that should require authentication. Look for T1190 indicators: anomalous 200-series responses to endpoints that should return 401/403 for unauthenticated callers. Check for unusual query patterns consistent with T1078 access abuse.

NIST Phase: Detection Analysis

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

Controls: NIST SI-4 (System Monitoring), NIST AU-6 (Audit Record Review, Analysis, and Reporting), NIST AU-2 (Event Logging), NIST AU-3 (Content of Audit Records), CIS 8.2 (Collect Audit Logs), CIS 7.1 (Establish and Maintain a Vulnerability Management Process)

Compensating: Query SBOM or package manifests with: `grep -r 'vanna' /path/to/requirements*.txt /path/to/pyproject.toml /path/to/setup.cfg 2>/dev/null`. For running containers: `docker exec pip list | grep vanna`. For T1190 detection in nginx logs, run: `awk '$9 ~ /^2[0-9]{2}$/ {print $0}' /var/log/nginx/access.log | grep -E '/(api|query|ask|train|connect)' > unauthenticated_hits.txt` — adjust route patterns to match actual vanna Flask endpoint paths (typically `/api/v0/` prefixed routes). For T1078 pattern detection, use the free Sigma rule framework with `sigma convert` targeting the access log source; alternatively, manually grep for repeated authenticated-session-less requests: `awk '{print $1}' access.log | sort | uniq -c | sort -rn | head -20` to surface high-frequency source IPs. Use osquery `SELECT * FROM python_packages WHERE name='vanna';` if osquery is deployed.

Evidence: Primary artifact: web server access logs filtered for vanna Flask API routes (e.g., `/api/v0/generate_sql`, `/api/v0/run_sql`, `/api/v0/train`) showing HTTP 200 responses with no Authorization header or session cookie present in the request. Secondary artifact: application-level logs from vanna itself if debug/info logging is enabled — these may record SQL query strings generated by the AI component, revealing what data an attacker caused the model to query. Capture Python process logs if Flask debug mode was enabled, as these may contain full request/response bodies. Preserve Flask werkzeug request logs if available at the Python application layer.

Eradication — No vendor-issued patch exists at time of publication. Disable or remove the Legacy Flask API component if it is not operationally required. If the component is required, enforce authentication at the reverse proxy or gateway layer as a compensating control. Monitor the vanna-ai GitHub repository (https://github.com/vanna-ai/vanna) for any patch release.

NIST Phase: Eradication

Reference: NIST 800-61r3 §3.4 — Eradication

Controls: NIST SI-2 (Flaw Remediation), NIST CM-7 (Least Functionality), NIST SI-5 (Security Alerts, Advisories, and Directives), CIS 2.3 (Address Unauthorized Software), CIS 4.6 (Securely Manage Enterprise Assets and Software), CIS 7.2 (Establish and Maintain a Remediation Process)

Compensating: To disable the Legacy Flask API component without full removal: set `VANNA_DISABLE_FLASK_API=true` in the application environment if supported, or comment out the Flask app instantiation in the vanna configuration. If the component must remain active, enforce authentication at the nginx reverse proxy layer by adding `auth_basic` or `auth_request` directives to the location block serving vanna Flask routes — example: `location /api/ { auth_request /auth; proxy_pass http://127.0.0.1:; }`. Alternatively, use an API gateway such as the open-source Kong (free tier) to require JWT or API key validation before requests reach the Flask service. To track vendor patch release without manual monitoring, set a GitHub RSS feed watch on `https://github.com/vanna-ai/vanna/releases.atom` or use a free service like GitHub's native 'Watch > Releases' notification to alert on any new release of vanna-ai/vanna.

Evidence: Before disabling or modifying the Flask component, capture the installed package state: `pip freeze > installed_packages_pre_eradication.txt`. If the component is being removed, document the configuration files that enabled it (typically `app.py`, `server.py`, or vanna's Flask app factory invocation). Preserve any vanna training data artifacts (`.pkl` files, vector store data, or connected database credentials stored in vanna config) as these may represent data that was exposed or exfiltrated during the exploitation window. Hash all captured files with SHA-256 before modification.

Recovery — After applying compensating controls or a future patch, verify that all API endpoints return appropriate 401/403 responses to unauthenticated test requests. Confirm access control enforcement through authenticated and unauthenticated test cases. Monitor application logs for continued anomalous access attempts for at least 30 days post-remediation.

NIST Phase: Recovery

Reference: NIST 800-61r3 §3.5 — Recovery

Controls: NIST IR-4 (Incident Handling), NIST SI-6 (Security and Privacy Function Verification), NIST AU-6 (Audit Record Review, Analysis, and Reporting), NIST SI-7 (Software, Firmware, and Information Integrity), CIS 7.1 (Establish and Maintain a Vulnerability Management Process), CIS 7.4 (Perform Automated Application Patch Management)

Compensating: Validate access control enforcement using curl from an external or non-privileged context: `curl -v -X GET https:///api/v0/generate_sql` — expected result is HTTP 401 or 403 with no query execution. Test authenticated path separately to confirm the reverse proxy auth layer passes legitimate requests through correctly. For ongoing 30-day monitoring without a SIEM, configure a cron job to parse nginx access logs nightly and email a summary of any 200-series responses on vanna API routes: `grep -E 'POST|GET.*/api/v0/' /var/log/nginx/access.log | awk '$9 ~ /^2/' | mail -s 'vanna API anomaly report' soc@yourdomain.com`. Use Wireshark or `tcpdump -i eth0 -w vanna_traffic.pcap port ` for periodic traffic capture during the monitoring window.

Evidence: Document the specific curl or HTTP test commands used for access control verification and preserve their output with timestamps as proof-of-remediation evidence. Capture a post-remediation snapshot of access logs showing the transition from anomalous 200-series responses to consistent 401/403 on unauthenticated requests — this delta is your remediation effectiveness record. If the vanna AI component executed any SQL queries during the exploitation window, attempt to retrieve the query history from the connected database (query logs in PostgreSQL at `pg_stat_statements` or MySQL general query log) to assess what data the attacker may have caused the model to retrieve.

Post-Incident — This vulnerability exposes a gap in authorization enforcement within AI/ML tooling that may lack the same security review maturity as production web services. Review all AI/ML libraries and frameworks in your environment for similar Legacy API components with insufficient access controls. Add vanna-ai/vanna to your software composition analysis (SCA) scanning pipeline and establish a process for tracking vendor responsiveness during coordinated disclosure.

NIST Phase: Post Incident

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

Controls: NIST IR-4 (Incident Handling), NIST IR-5 (Incident Monitoring), NIST IR-8 (Incident Response Plan), NIST RA-3 (Risk Assessment), NIST SI-2 (Flaw Remediation), 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)

Compensating: Integrate vanna-ai/vanna and all AI/ML Python packages into free SCA tooling: run `pip-audit` (free, pip-installable) against all requirements files to surface known CVEs in the Python dependency chain — `pip-audit -r requirements.txt`. For broader AI/ML library review, run `grype sbom:./sbom.json` using the free Anchore Grype tool against a generated SBOM (`syft . -o spdx-json > sbom.json` using free Syft). To systematically identify other AI/ML packages exposing unauthenticated HTTP APIs, grep for Flask, FastAPI, Gradio, Streamlit, or LiteLLM instantiations in your codebase that lack authentication middleware: `grep -r 'Flask(__name__)\|app.run(' --include='*.py' /path/to/project`. Document vendor non-response as a risk item and establish a 90-day SLA for vendor acknowledgment in your coordinated disclosure policy per NIST IR-8 guidance.

Evidence: Produce a full lessons-learned record per NIST 800-61r3 §4 including: the discovery timeline for vanna-ai/vanna <=2.0.2 in your environment, the gap between deployment and detection, the access window during which the unpatched Legacy Flask API was internet-facing, and any SQL queries or data accesses that occurred during that window. Preserve the complete access log archive covering the exposure period to support any future regulatory breach notification analysis. Document the vendor non-response timeline as evidence for risk acceptance decisions and potential upstream supply chain risk escalation.

Detection Guidance

Search package manifests, requirements.txt, pyproject.toml, and container images for vanna-ai/vanna at version 2.0.2 or below. In application and web server logs, look for unauthenticated requests returning HTTP 200 to Flask API routes that handle query or data operations; these should return 401 or 403. Flag requests where no

Authorization header or session token is present but a successful response is returned. If a WAF is in place, review logs for requests matching the Flask API path patterns with no associated authenticated session. No specific IOC hashes or IPs are available at this time given the absence of observed active exploitation campaigns.

Framework Mappings

MITRE-ATTACK

- **T1078** — Valid Accounts
- **T1190** — Exploit Public-Facing Application

NIST-800-53R5

- **AC-2** — Account Management
- **AC-6** — Least Privilege
- **IA-2** — Identification and Authentication (Organizational Users)
- **IA-5** — Authenticator Management
- **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-4** — System Monitoring

ISO-27001-2022

- **A.8.8** — Management of technical vulnerabilities
- **A.5.21** — Managing information security in the ICT supply chain

SOC2-TSC

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

CIS-V8

- **8.2** — Collect Audit Logs

NIST-CSF-2

- **DE.CM-01** — Networks and network services are monitored

MITRE ATT&CK Mapping

Technique ID	Technique Name	Tactic
T1078	Valid Accounts	Defense-Evasion
T1190	Exploit Public-Facing Application	Initial-Access

Sources

Source	URL	Tier
nvd	https://nvd.nist.gov/vuln/detail/CVE-2026-6977	T1
CVE-2026-6977 - CVE Record	https://www.cve.org/CVERecord?id=CVE-2026-6977	T3
CVE-2026-6977 | Tenable®	https://www.tenable.com/cve/CVE-2026-6977	T3
CVE-2026-6977: A security vulnerability has Authorization bypass	https://www.sherlockforensics.com/blog/2026-04-26-cve-2026-6977.html	T3
CVE-2026-6977 | Mondoo Vulnerability Intelligence	https://mondoo.com/vulnerability-intelligence/vulnerability/CVE-202...	T3

Generated 2026-04-28 06:34 UTC by TJS Security Command Center