INTELLIGENCE BRIEFING
Security Command Center
TLP:CLEAR
2026-05-27 06:40 UTC

CVE-2026-9356: A vulnerability has been found in SourceCodester Hospitals Patient Records Management System 1.0. Th...

CVE VULNERABILITY | HIGH | CVSS 7.3

SCC Item ID	SCC-CVE-2026-0230
Type	CVE Vulnerability
CVE ID	CVE-2026-9356
Severity	HIGH
CVSS Base Score	7.3
EPSS Score	0.0003 (9th percentile)
Affected Products	SourceCodester Hospitals Patient Records Management System 1.0
Published	2026-05-24T06:16:36.150
Discovery Source	Nvd

Executive Summary

A SQL injection vulnerability (CVE-2026-9356, CVSS 7.3) has been identified in SourceCodester Hospitals Patient Records Management System 1.0, affecting the patient history management interface. An unauthenticated remote attacker can manipulate the 'ID' parameter to extract, modify, or delete patient records from the underlying database. A public proof-of-concept exploit is available, meaning exploitation requires minimal technical skill and no prior access.

Technical Analysis

CVE-2026-9356 is a SQL injection vulnerability (CWE-89, CWE-74) in the file /admin/patients/manage_history.php of SourceCodester Hospitals Patient Records Management System 1.0. The 'ID' GET/POST parameter is passed unsanitized to the backend SQL query, allowing an attacker to inject arbitrary SQL statements. Attack vector is network-based with no authentication required (CVSS 7.3). A public proof-of-concept has been disclosed, lowering the barrier to exploitation. EPSS score is 0.03% (0.9th percentile) as of the configuration date, indicating low observed exploitation volume currently, though PoC availability elevates near-term risk. MITRE ATT&CK technique T1190 (Exploit Public-Facing Application) applies. No vendor patch has been confirmed in the source data. The affected software is a PHP-based open-source hospital records application commonly deployed in small and mid-tier healthcare environments. Review all user-supplied parameters in the application for similar injection flaws during remediation.

Action Checklist

1. Step 1: Containment, Immediately restrict network access to /admin/patients/manage_history.php at the web server or firewall level. If the application is internet-facing, block external access to the /admin/ path entirely until patched. Apply network-layer controls (NIST AC-4) to prevent unauthorized inbound traffic to this endpoint.
2. Step 2: Detection, Review web server access logs for anomalous requests to /admin/patients/manage_history.php containing SQL metacharacters (single quotes, double dashes, UNION, SELECT, OR 1=1 patterns) in the 'ID' parameter. Query your SIEM or log aggregator for HTTP requests with URL-encoded payloads (%27, %22, %2D%2D) targeting this path. Flag any 500-series error spikes or unusual response sizes from this endpoint as potential active exploitation (NIST AU-6, CIS 8.2).
3. Step 3: Eradication, Apply input validation and parameterized queries to the 'ID' parameter in manage_history.php if your team maintains the codebase. If running an unmodified SourceCodester deployment, check the vendor's GitHub repository for a patched release and upgrade immediately. If no patch exists, implement a WAF rule to block SQL injection patterns against this path as a compensating control (NIST SI-10).
4. Step 4: Recovery, After applying the patch or compensating control, validate the fix by submitting test SQL injection payloads (in a non-production clone) to confirm the parameter is no longer injectable. Review database audit logs for signs of unauthorized reads, modifications, or deletions to patient_history tables during the exposure window. Restore any altered records from verified backups. Monitor the endpoint for continued anomalous traffic (NIST SI-4).
5. Step 5: Post-Incident, Document the exposure window and any confirmed or suspected data access. Conduct a broader code review of all user-supplied parameters in the application for similar injection flaws. Implement a web application firewall policy covering the full /admin/ path (CIS 4.4). Establish a vulnerability management process per CIS 7.1 to track open-source application CVEs relevant to your deployed software inventory.

IR / Forensic Enrichment

Triage Priority	IMMEDIATE
Escalation Criteria	Escalate immediately to legal/compliance and executive leadership if MySQL binlogs, application logs, or database row-count anomalies confirm unauthorized SELECT, UPDATE, or DELETE operations against patient_history or patients tables during the exposure window, as this constitutes a potential HIPAA reportable breach of PHI requiring breach notification assessment within 60 days under 45 CFR §164.400-414; also escalate if a public PoC has been weaponized into active scanning campaigns targeting SourceCodester deployments (monitor CISA KEV and threat intel feeds for addition of CVE-2026-9356).

Recovery Notes	After patching or WAF deployment, revalidate the manage_history.php endpoint using sqlmap against a non-production clone to confirm the 'id' parameter is no longer injectable before restoring public access. Conduct a 30-day heightened monitoring window on the /admin/ path access logs and MySQL general query log, specifically watching for anomalous query patterns against patient_history, patients, and users tables that could indicate a persistent access mechanism (e.g., a backdoor account created via SQLi during the exposure window). Cross-reference the user accounts table in the application database against known legitimate staff accounts to detect any accounts injected by the attacker using INSERT statements during exploitation.
Forensic Artifacts	Web server access log (/var/log/apache2/access.log or /var/log/nginx/access.log): contains the raw HTTP requests to manage_history.php with the injected 'id' parameter values, source IPs, timestamps, HTTP response codes, and response sizes — the primary evidence source for exploitation scope and attacker IP attribution | MySQL/MariaDB binary logs (binlogs at path configured in my.cnf log_bin directive): contain a forensic record of every SQL statement executed against the database including any injected UNION SELECT, UPDATE, DELETE, or INSERT statements that modified or exfiltrated patient_history and patients table data | PHP application error log (path defined in php.ini error_log directive, often /var/log/php_errors.log): will contain MySQL syntax error messages generated by malformed injection probes, providing a secondary timeline of failed and successful injection attempts against the patient_history query in manage_history.php | MySQL general query log (enabled via 'SET GLOBAL general_log = ON' with log output to FILE): if enabled during or retroactively after the incident, captures full SQL statements passed to the database engine including any FILE() or INTO OUTFILE commands that may have been used to write a PHP webshell to the web root | Filesystem modification timestamps on /var/www/html/ PHP files: exploitation of MySQL FILE privilege via SQLi can deposit webshells; 'find /var/www/html -name "*.php" -newer [incident_start_timestamp_reference_file]' will identify any PHP files written to the web root during the attack window as evidence of post-exploitation persistence

Per-Action IR Details

Step 1: Containment — Immediately restrict network access to /admin/patients/manage_history.php at the web server or firewall level. If the application is internet-facing, block external access to the /admin/ path entirely until patched. Per NIST AC-4 (Information Flow Enforcement), enforce network-layer controls to prevent unauthorized inbound traffic to this endpoint.

NIST Phase: Containment

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

Controls: NIST AC-4 (Information Flow Enforcement), 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)

Compensating: On Apache: add 'Require all denied' inside a block in httpd.conf and reload. On Nginx: add 'deny all;' in a location block for that path. At the OS firewall level (iptables): run 'iptables -I INPUT -p tcp --dport 80 -m string --string "/admin/" --algo bm -j DROP' as an emergency rule. For a 2-person team with no WAF appliance, ModSecurity (free, Apache/Nginx module) with the OWASP Core Rule Set can be installed and activated within 30 minutes to block SQLi patterns against the entire /admin/ path.

Evidence: Before implementing block rules, capture a full snapshot of the current Apache/Nginx access log (e.g., /var/log/apache2/access.log or /var/log/nginx/access.log) to preserve the pre-containment exploitation record. Also capture active connection state via 'netstat -antp' or 'ss -tp' to identify any live sessions to the manage_history.php endpoint. Take a timestamped copy of the web server's running configuration to document the unpatched exposure state for the incident record.

Step 2: Detection — Review web server access logs for anomalous requests to /admin/patients/manage_history.php containing SQL metacharacters (single quotes, double dashes, UNION, SELECT, OR 1=1 patterns) in the 'ID' parameter. Query your SIEM or log aggregator (aligned with NIST AU-6, CIS 8.2) for HTTP requests with URL-encoded payloads (%27, %22, %2D%2D) targeting this path. Flag any 500-series error spikes or unusual response sizes from this endpoint as potential active exploitation.

NIST Phase: Detection Analysis

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

Controls: NIST AU-6 (Audit Record Review, Analysis, and Reporting), NIST AU-2 (Event Logging), NIST SI-4 (System Monitoring), CIS 8.2 (Collect Audit Logs)

Compensating: Without a SIEM, run this grep against Apache/Nginx access logs to detect SQLi attempts against the vulnerable endpoint: grep -iE "manage_history\.php.*['\"%27%22%2D%2D]|UNION|SELECT|OR\s+1=1|--" /var/log/apache2/access.log. To find HTTP 500 error spikes that may indicate successful backend database errors triggered by the injection: awk '$9==500' /var/log/apache2/access.log | grep manage_history. For URL-encoded variants: grep -iE "%27|%22|%2D%2D|%20UNION%20|%20SELECT%20" /var/log/apache2/access.log | grep manage_history. If the application uses PHP error logging, also check /var/log/php_errors.log or the path defined in php.ini for SQL syntax error messages referencing the patient_history table, which would confirm backend injection reached the database layer.

Evidence: The primary forensic evidence for active exploitation of CVE-2026-9356 will be in the web server access log: look for GET or POST requests to /admin/patients/manage_history.php where the 'id' parameter contains SQL syntax. Successful UNION-based extraction attacks produce responses with anomalously large Content-Length values compared to normal record lookups. Also pull the PHP/application error log for MySQL/MariaDB syntax errors (e.g., 'You have an error in your SQL syntax') timestamped against suspicious access log entries. If the database server has its own general query log enabled (MySQL: /var/lib/mysql/general.log or configured path), capture it immediately — it will contain the raw injected SQL statements executed against the patients/patient_history tables.

Step 3: Eradication — Apply input validation and parameterized queries to the 'ID' parameter in manage_history.php if your team maintains the codebase. If running an unmodified SourceCodester deployment, check the vendor's GitHub repository for a patched release and upgrade immediately. If no patch exists, implement a WAF rule to block SQL injection patterns against this path as a compensating control. Per NIST SI-10 (Information Input Validation), all user-supplied inputs must be validated before processing.

NIST Phase: Eradication

Reference: NIST 800-61r3 §3.4 — Eradication

Controls: NIST SI-2 (Flaw Remediation), NIST SI-10 (Information Input Validation), NIST CM-6 (Configuration Settings), CIS 7.3 (Perform Automated Operating System Patch Management), CIS 7.4 (Perform Automated Application Patch Management)

Compensating: If a vendor patch is unavailable for SourceCodester Hospitals Patient Records Management System 1.0, apply a ModSecurity rule scoped to the vulnerable endpoint: 'SecRule ARGS:id "@detectSQLi" "id:1001,phase:2,deny,status:403,msg:SQLi attempt on manage_history"'. As a code-level fix if you have PHP source access, replace any direct concatenation in manage_history.php (e.g., '$query = "SELECT * FROM patient_history WHERE id=" . $_GET["id"]') with a PDO prepared statement: '$stmt = $pdo->prepare("SELECT * FROM patient_history WHERE id = ?"); $stmt->execute([$_GET["id"]]);'. Validate that the 'id' parameter is strictly integer before the query using PHP's 'intval()' or a regex whitelist as a defense-in-depth measure.

Evidence: Before applying code or configuration changes, preserve a full file-level hash inventory of the manage_history.php file and any included database abstraction files using 'sha256sum /var/www/html/admin/patients/manage_history.php > pre_patch_hashes.txt'. If exploitation is confirmed, check the patient_history database table for unexpected schema changes, new admin accounts in the users table, or stored web shells written via SQL FILE() functions (MySQL: check for files in the web root created around the exploitation timestamp). Run 'find /var/www/html -newer /var/log/apache2/access.log -name "*.php"' to detect any PHP files dropped during the attack window.

Step 4: Recovery — After applying the patch or compensating control, validate the fix by submitting test SQL injection payloads (in a non-production clone) to confirm the parameter is no longer injectable. Review database audit logs for signs of unauthorized reads, modifications, or deletions to patient_history tables during the exposure window. Restore any altered records from verified backups. Monitor the endpoint for continued anomalous traffic per NIST SI-4 (System Monitoring).

NIST Phase: Recovery

Reference: NIST 800-61r3 §3.5 — Recovery

Controls: NIST SI-4 (System Monitoring), NIST CP-9 (System Backup), NIST CP-10 (System Recovery and Reconstitution), NIST AU-11 (Audit Record Retention), CIS 7.1 (Establish and Maintain a Vulnerability Management Process)

Compensating: To validate the fix without a commercial scanner, use sqlmap in a non-production clone: 'sqlmap -u "http://[clone-host]/admin/patients/manage_history.php?id=1" --level=3 --risk=2 --dbms=mysql --batch' — a 'not injectable' result confirms remediation. For ongoing monitoring without a SIEM, deploy a cron-based log watcher: 'crontab -e' with '*/5 * * * * grep -c "manage_history" /var/log/apache2/access.log >> /var/log/manage_history_hit_count.log' to track request volume, and set an alert threshold alert via a simple bash script comparing counts against a baseline. Use MySQL's built-in audit via the general_log (SET GLOBAL general_log = 'ON';) temporarily during the monitoring window to capture any residual unauthorized queries against patient_history.

Evidence: Pull a full row-count and checksum audit of the patient_history, patients, and users tables in MySQL/MariaDB before and after restoration to quantify data integrity impact: 'SELECT COUNT(*), CHECKSUM TABLE patient_history;'. Check MySQL binary logs (binlogs) at the configured log_bin path for DML statements (INSERT, UPDATE, DELETE) against patient_history during the exploitation window — these provide a forensic record of every database modification made by the injected queries. Preserve these binlogs as legal evidence if PHI exposure is confirmed, as they may be required for HIPAA breach notification documentation.

Step 5: Post-Incident — Document the exposure window and any confirmed or suspected data access. Conduct a broader review of all user-supplied parameters in the application for similar injection flaws, as SourceCodester applications have a history of CWE-89 findings. Implement a web application firewall policy covering the full /admin/ path (CIS 4.4). Establish a vulnerability management process per CIS 7.1 to track open-source application CVEs relevant to your deployed software inventory.

NIST Phase: Post Incident

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

Controls: NIST IR-4 (Incident Handling), NIST RA-3 (Risk Assessment), NIST SI-2 (Flaw Remediation), NIST CA-7 (Continuous Monitoring), CIS 4.4 (Implement and Manage a Firewall on Servers), CIS 7.1 (Establish and Maintain a Vulnerability Management Process), CIS 7.2 (Establish and Maintain a Remediation Process)

Compensating: For the broader CWE-89 audit of the SourceCodester application without a commercial DAST tool, run grep recursively across all PHP source files to identify unparameterized query patterns: 'grep -rn "\$_GET\|\$_POST\|\$_REQUEST" /var/www/html/ | grep -i "query\|sql\|SELECT\|WHERE"'. Feed results into a manual review prioritizing any file that constructs SQL strings from user input without 'prepare()' or 'real_escape_string()'. To track future SourceCodester CVEs, subscribe to the NVD CPE feed for 'cpe:2.3:a:sourcecodester:hospital_patient_record_system' via the NVD API (free) or create a keyword alert in CISA's Known Exploited Vulnerabilities catalog for 'SourceCodester'. Document findings in a risk register tied to CIS 2.1 (Establish and Maintain a Software Inventory) so the application is tracked as a monitored asset.

Evidence: Compile the final incident timeline using: (1) web server access log timestamps showing first and last observed SQLi attempt against manage_history.php, (2) MySQL binlog timestamps showing first unauthorized DML against patient_history, and (3) the file modification timestamps of any dropped webshells or altered PHP files. If PHI records were accessed or exfiltrated, this timeline directly supports HIPAA Breach Notification Rule (45 CFR §164.400-414) assessment of the 60-day notification window. Preserve all logs, binlogs, and file artifacts in a write-protected evidence archive (tar.gz with sha256sum manifest) before the standard log rotation cycle overwrites them.

Detection Guidance

Query web server access logs (Apache/Nginx) for requests to /admin/patients/manage_history.php where the 'id' parameter contains SQL injection indicators: single quotes ('), double dashes (--), URL-encoded equivalents (%27, %2D%2D), or keywords such as UNION, SELECT, OR, AND followed by numeric comparisons. Look for HTTP 500 responses or abnormally large response bodies from this endpoint, which may indicate successful data extraction. In your SIEM, create a detection rule (aligned with NIST AU-6, AU-12, CIS 8.2) filtering on: source_file = 'manage_history.php' AND (uri_query CONTAINS '%27' OR uri_query CONTAINS 'UNION' OR uri_query CONTAINS '--'). Also alert on database query errors logged by the PHP application. If a WAF is in place, review blocked-request logs for repeated SQL injection attempts against this path from single source IPs. No confirmed IOCs (IPs, domains, hashes) are available from the provided source data.

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

ISO-27001-2022

- **A.8.28** — Secure coding
- **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-2026-9356	T1
CVE-2026-9356 - CVE Details, Severity, and Analysis | Strobes VI	https://strobes.co/vi/cve/CVE-2026-9356/	T3
CVE-2026-7356 Detail - NVD	https://nvd.nist.gov/vuln/detail/CVE-2026-7356	T1
CVE-2026-9356 | Tenable®	https://www.tenable.com/cve/CVE-2026-9356	T3
CVE-2026-36356: MeiG Smart FORGE_SLT711 RCE Vulnerability	https://www.sentinelone.com/vulnerability-database/cve-2026-36356/	T3

DISCLAIMER

This intelligence report is produced by Tech Jacks Solutions Security Command Center (SCC) for informational purposes only. It does not constitute professional security advice, legal counsel, or an incident response engagement. The information herein is derived from publicly available sources and AI-assisted analysis; while every effort is made to ensure accuracy, Tech Jacks Solutions makes no warranties regarding completeness or timeliness. Organizations should conduct their own validation and consult qualified security professionals before taking action based on this report. Tech Jacks Solutions is not liable for any damages resulting from the use of this information.

Generated 2026-05-27 06:40 UTC by TJS Security Command Center