INTELLIGENCE BRIEFING
Security Command Center
TLP:CLEAR
2026-04-22 06:47 UTC

CVE-2026-6562: A flaw has been found in dameng100 muucmf 1.9.5.20260309. Impacted is the function getListByPage of ...

CVE VULNERABILITY | HIGH | CVSS 7.3

SCC Item ID	SCC-CVE-2026-0066
Type	CVE Vulnerability
CVE ID	CVE-2026-6562
Severity	HIGH
CVSS Base Score	7.3
EPSS Score	0.0003 (8th percentile)
Affected Products	dameng100 muucmf 1.9.5.20260309
Published	2026-04-19T09:16:10.100
Discovery Source	Nvd

Executive Summary

CVE-2026-6562 is a SQL injection vulnerability in dameng100 muucmf version 1.9.5.20260309, a PHP-based content management framework. An unauthenticated remote attacker can inject malicious SQL commands through the search function, potentially accessing or modifying the underlying database. No vendor patch exists; the vendor did not respond to disclosure, and a public exploit is available.

Technical Analysis

CVE-2026-6562 affects the getListByPage function in /index/Search/index.html of dameng100 muucmf version 1.9.5.20260309. The 'keyword' parameter is not properly sanitized before being passed to a SQL query, satisfying CWE-89 (SQL Injection) and the broader CWE-74 (Injection) classification. Attack vector is network-based, requires no authentication, and a public proof-of-concept exploit has been published. CVSS base score is 7.3 (High). EPSS score is 0.00028 (8th percentile), indicating currently low observed exploitation activity. MITRE ATT&CK technique T1190 (Exploit Public-Facing Application) applies. No vendor patch or acknowledgment exists as of configuration date 2026-03-04. Source quality score is 0.712; primary reference is NVD (nvd.nist.gov/vuln/detail/CVE-2026-6562).

Action Checklist

1. Step 1: Containment. Identify all instances of dameng100 muucmf version 1.9.5.20260309 in your environment. If internet-facing, immediately restrict access to the /index/Search/index.html endpoint via WAF rule or network ACL blocking external requests to that path.
2. Step 2: Detection. Query web server access logs for requests to /index/Search/index.html containing SQL metacharacters in the 'keyword' parameter: look for patterns such as single quotes, double dashes, UNION, SELECT, or encoded equivalents (%27, %2D%2D, %55%4E%49%4F%4E). Flag any anomalous database error responses (HTTP 500) following search requests.
3. Step 3: Eradication. No vendor patch is available. Options: remove or disable the search endpoint entirely if not required; implement input validation and parameterized queries at the application layer if source access exists; replace the application with a patched alternative if business requirements allow. Do not leave the unpatched endpoint internet-facing.
4. Step 4: Recovery. After applying compensating controls, retest the /index/Search/index.html endpoint with SQL injection payloads to confirm the mitigation is effective. Review database audit logs for signs of unauthorized queries or data access during the exposure window. Restore service only after validation is complete.
5. Step 5: Post-Incident. Evaluate whether vendor software with no established security response process meets your third-party risk acceptance criteria. Implement a software inventory control to flag use of unsupported or unresponsive-vendor components. Add WAF rules for SQL injection patterns as a baseline control across all web-facing applications.

IR / Forensic Enrichment

Triage Priority	IMMEDIATE
Escalation Criteria	Escalate to senior IR leadership and legal/compliance if MySQL binary log review reveals successful data exfiltration (e.g., SELECT queries returning user, credentials, or PII tables, or INTO OUTFILE activity), as this triggers breach notification obligations under applicable data protection regulations; also escalate if a webshell is discovered in the muucmf web root, indicating post-exploitation and a wider compromise scope beyond the SQL injection.
Recovery Notes	After compensating controls are confirmed effective via sqlmap validation, monitor web server access logs and MySQL slow query logs daily for a minimum of 30 days for resumed scanning or exploitation attempts against the muucmf application or adjacent PHP endpoints from the same source IPs identified during the exposure window. Verify the database schema and all user accounts against a known-good baseline to confirm no persistence mechanisms (rogue DB users, INTO OUTFILE webshells) were established before containment. Do not restore unrestricted internet access to any muucmf endpoint until either a code-level fix is in place or the application is replaced, given the availability of a public exploit and the absence of a vendor patch.

Forensic Artifacts	Web server access logs (Apache: /var/log/apache2/access.log; nginx: /var/log/nginx/access.log) — will contain the raw HTTP GET/POST requests to /index/Search/index.html with SQL metacharacters or encoded payloads in the 'keyword' parameter, sourced directly from the CVE-2026-6562 exploit vector | MySQL/MariaDB general query log and binary logs (mysql-bin.*) — will capture the injected SQL statements constructed by the vulnerable getListByPage function, including any UNION SELECT, information_schema enumeration, or INTO OUTFILE commands executed during the attack | muucmf application error/runtime logs (typically runtime/logs/ under the framework root) — will contain PHP stack traces and database error output generated by malformed SQL injection payloads, confirming successful injection point interaction | Web root filesystem — check for unexpected .php files written via MySQL INTO OUTFILE (e.g., 'find /var/www -name "*.php" -newer [deployment_date] -ls') as this exploit class commonly enables webshell deployment when the MySQL user has FILE privilege | MySQL user and privilege tables (SELECT user, host, plugin, authentication_string FROM mysql.user;) — an attacker with sufficient injection privileges may have created a backdoor database account; compare against a known-good baseline captured before the exposure window

Per-Action IR Details

Step 1: Containment — Identify all instances of dameng100 muucmf version 1.9.5.20260309 in your environment. If internet-facing, immediately restrict access to the /index/Search/index.html endpoint via WAF rule or network ACL blocking external requests to that path.

NIST Phase: Containment

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

Controls: NIST IR-4 (Incident Handling), NIST SC-7 (Boundary Protection), CIS 1.1 (Establish and Maintain Detailed Enterprise Asset Inventory), CIS 4.4 (Implement and Manage a Firewall on Servers), CIS 7.1 (Establish and Maintain a Vulnerability Management Process)

Compensating: Run 'grep -r "muucmf" /var/www/ --include="*.php" -l' to locate all muucmf deployments on Linux web hosts; on Windows use 'Get-ChildItem -Recurse -Filter *.php | Select-String "muucmf"'. Block the endpoint immediately with an nginx rule: 'location ~* ^/index/Search/index\.html { deny all; }' or equivalent Apache directive 'Deny from all' in a Location block. If no WAF is available, add an iptables rule: 'iptables -I INPUT -p tcp --dport 80 -m string --string "/index/Search/index.html" --algo bm -j DROP' as an interim measure.

Evidence: Before blocking, capture a full snapshot of current web server access logs (Apache: /var/log/apache2/access.log; nginx: /var/log/nginx/access.log) and error logs to preserve pre-containment attack evidence. Export a list of all muucmf PHP files and their last-modified timestamps ('find /var/www -name "*.php" -newer /tmp/baseline -ls') to detect any webshells already dropped via prior exploitation. Record current database user session data from the muucmf database ('SELECT user, host, db, command, time, state, info FROM information_schema.processlist;') to identify any active unauthorized sessions before access is cut.

Step 2: Detection — Query web server access logs for requests to /index/Search/index.html containing SQL metacharacters in the 'keyword' parameter: look for patterns such as single quotes, double dashes, UNION, SELECT, or encoded equivalents (%27, %2D%2D, %55%4E%49%4F%4E). Flag any anomalous database error responses (HTTP 500) following search requests.

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)

Compensating: Run this grep against Apache/nginx access logs to surface CVE-2026-6562 exploitation attempts: 'grep -i "/index/Search/index.html" /var/log/apache2/access.log | grep -iE "(%27|%2D%2D|%55%4E%49%4F%4E|UNION|SELECT|\x27|--|\bOR\b|\bAND\b)"'. Follow up with: 'grep

"/index/Search/index.html" /var/log/apache2/access.log | awk "{print $9}" | sort | uniq -c | sort -rn' to surface HTTP 500 response spikes indicating successful error-based injection. Deploy a free ModSecurity CRS rule targeting the keyword parameter on this path; the OWASP CRS rule 942100 (SQL Injection Attack Detected via libinjection) will catch most payloads including UNION-based and boolean-blind variants.

Evidence: Preserve the full web server access log for the muucmf host covering the exposure window (from public exploit availability date forward). Extract all unique source IPs hitting /index/Search/index.html and cross-reference against known scanner ASNs. Capture MySQL/MariaDB general query log (enable temporarily if off: 'SET GLOBAL general_log = ON; SET GLOBAL general_log_file = "/tmp/mysql_general.log";') to record any SQL statements injected through the getListByPage function. Check muucmf application error logs (typically under the framework's runtime/logs/ directory) for stack traces exposing database query construction, which confirms injection point reachability.

Step 3: Eradication — No vendor patch is available. Options: remove or disable the search endpoint entirely if not required; implement input validation and parameterized queries at the application layer if source access exists; replace the application with a patched alternative if business requirements allow. Do not leave the unpatched endpoint internet-facing.

NIST Phase: Eradication

Reference: NIST 800-61r3 §3.4 — Eradication

Controls: NIST SI-2 (Flaw Remediation), NIST SI-10 (Information Input Validation), NIST CM-7 (Least Functionality), CIS 2.2 (Ensure Authorized Software is Currently Supported), CIS 2.3 (Address Unauthorized Software), CIS 7.2 (Establish and Maintain a Remediation Process)

Compensating: If source access exists, locate the getListByPage function in the muucmf PHP source and replace direct string concatenation into the SQL query with PDO prepared statements: replace 'WHERE keyword LIKE \'%'.$keyword.'%\'' with a parameterized binding ('WHERE keyword LIKE ?', [$keyword]). If source modification is not feasible, rename or remove the Search controller file entirely ('mv /var/www/muucmf/application/index/controller/Search.php /var/www/muucmf/application/index/controller/Search.php.disabled') and verify the endpoint returns 404. Document the absence of a vendor patch and the business justification for any continued operation under compensating controls per NIST SI-2 requirements.

Evidence: Before removing or modifying the Search controller, take a file hash of the original PHP source ('sha256sum /var/www/muucmf/application/index/controller/Search.php') and preserve a copy for forensic reference. Audit the muucmf database for signs of data exfiltration that occurred prior to eradication: check for unexpected new database users ('SELECT user, host FROM mysql.user;'), recently modified tables ('SELECT table_name, update_time FROM information_schema.tables WHERE table_schema = DATABASE() ORDER BY update_time DESC;'), and any outbound file-write artifacts (MySQL INTO OUTFILE webshells) in the web root ('find /var/www -name "*.php" -newer /var/www/muucmf/index.php -ls').

Step 4: Recovery — After applying compensating controls, retest the /index/Search/index.html endpoint with SQL injection payloads to confirm the mitigation is effective. Review database audit logs for signs of unauthorized queries or data access during the exposure window. Restore service only after validation is complete.

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 SI-7 (Software, Firmware, and Information Integrity), NIST AU-11 (Audit Record Retention), CIS 7.1 (Establish and Maintain a Vulnerability Management Process)

Compensating: Use sqlmap in safe, read-only mode to validate the compensating control is effective: 'sqlmap -u "http://[target]/index/Search/index.html?keyword=test" --level=3 --risk=1 --technique=BEU --batch --no-cast' — a clean result confirms the injection is blocked. Separately, run a manual curl test: 'curl -v "http://[target]/index/Search/index.html?keyword=1%27+OR+1%3D1--'" and verify the response is a block page or 404, not application output. Review MySQL binary logs for the exposure window ('mysqlbinlog --start-datetime="2026-03-04

00:00:00" /var/lib/mysql/mysql-bin.* | grep -iE "(DROP|DELETE|UPDATE|INSERT|INTO OUTFILE)"') to identify unauthorized data manipulation before restoring full service.

Evidence: Pull the MySQL binary logs and slow query logs for the entire exposure window to reconstruct any unauthorized SQL executed through the getListByPage injection point. Compare current database table row counts and schema against the most recent clean backup to detect exfiltrated or deleted records. Verify integrity of all PHP files in the muucmf web root using stored hashes or a fresh deployment checksum ('find /var/www/muucmf -name "*.php" -exec sha256sum {} \;') to confirm no webshells were written via MySQL INTO OUTFILE during the exposure window.

Step 5: Post-Incident — Evaluate whether vendor software with no established security response process meets your third-party risk acceptance criteria. Implement a software inventory control to flag use of unsupported or unresponsive-vendor components. Add WAF rules for SQL injection patterns as a baseline control across all web-facing applications.

NIST Phase: Post Incident

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

Controls: NIST IR-4 (Incident Handling), NIST IR-8 (Incident Response Plan), NIST SI-5 (Security Alerts, Advisories, and Directives), NIST RA-3 (Risk Assessment), CIS 2.1 (Establish and Maintain a Software Inventory), CIS 2.2 (Ensure Authorized Software is Currently Supported), CIS 7.1 (Establish and Maintain a Vulnerability Management Process), CIS 7.4 (Perform Automated Application Patch Management)

Compensating: Add muucmf and its vendor (dameng100) to a tracked 'no-patch-response' list in your software inventory; flag any future CVEs against this vendor for immediate escalation without waiting for a patch cycle. Create a Sigma rule targeting web server logs for SQL injection patterns against all PHP CMS endpoints (not just muucmf) and deploy it as a scheduled grep cron job if no SIEM is available. Subscribe to NVD CPE watch for 'dameng100' ('https://nvd.nist.gov/products/cpe/search' — search-retrieved, recommend human validation) to receive future CVE notifications for this vendor's components.

Evidence: Compile a lessons-learned report documenting the exposure window (from muucmf deployment date to containment date), all source IPs that targeted the endpoint, whether any successful injection is evidenced in MySQL logs, and the absence of a vendor patch as a contributing risk factor. Preserve all web server access logs, MySQL binary logs, and modified PHP file hashes as an evidence package retained per NIST AU-11 (Audit Record Retention) requirements — minimum 90 days, longer if regulatory obligations apply. Document the third-party risk finding for dameng100 muucmf in your risk register with a risk acceptance decision or replacement timeline.

Detection Guidance

Monitor web server access logs for POST or GET requests to /index/Search/index.html where the 'keyword' parameter contains SQL injection indicators: single quotes ('), double dashes (--), UNION SELECT sequences, or URL-encoded equivalents (%27, %2D%2D, %55%4E%49%4F%4E%20%53%45%4C%45%43%54). Correlate with backend database error logs or HTTP 500 responses immediately following search requests. If a WAF is in place, review its event log for triggered SQL injection signatures against this path. No confirmed IOCs (IPs, domains, hashes) have been attributed to active exploitation campaigns against this CVE as of the data provided; detection should focus on behavioral patterns rather than known-bad indicators.

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

OWASP-TOP10-2021

- **A03:2021** — Injection

CIS-V8

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

ISO-27001-2022

- **A.8.28** — Secure coding
- **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

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-2026-6562	T1
CVE-2026-6562 - CVE Record	https://www.cve.org/CVERecord?id=CVE-2026-6562	T3
CVE-2026-6562 | Mondoo Vulnerability Intelligence	https://mondoo.com/vulnerability-intelligence/vulnerability/CVE-202...	T3
CVE-2026-6562 Security Vulnerability Analysis & Exploit Details	https://cve.akaoma.com/cve-2026-6562	T3

Source	URL	Tier
CVE-2026-6562 - Exploits & Severity - Feedly	https://feedly.com/cve/CVE-2026-6562	T3

Generated 2026-04-22 06:47 UTC by TJS Security Command Center