INTELLIGENCE BRIEFING
Security Command Center
TLP:CLEAR
2026-05-29 14:01 UTC

Unpatched Gogs RCE: Authenticated Users Can Compromise Any Instance via Git Rebase Injection

CVE VULNERABILITY | **CRITICAL** | CVSS 9.5

SCC Item ID	SCC-CVE-2026-0237
Type	CVE Vulnerability
Severity	CRITICAL
CVSS Base Score	9.5
Affected Products	Gogs (all versions as of late May 2026; Windows, Linux, macOS)
Published	2026-05-28T13:24:44
Discovery Source	Rss

Executive Summary

A critical, unpatched remote code execution vulnerability in Gogs, a widely used self-hosted Git service, allows any authenticated user to execute OS commands with the privileges of the Gogs process user, potentially leading to full control of the host server. The attack is fully automated via a publicly available exploit tool, meaning the barrier to exploitation is low and active compromise attempts are likely. Organizations running Gogs for source code management face immediate risk of intellectual property theft, supply chain compromise, and full system takeover with no vendor patch available.

Technical Analysis

An argument injection vulnerability (CWE-88) in Gogs' git rebase operation allows any authenticated user to inject OS commands through a maliciously crafted branch name containing the --exec flag. The injected argument passes unsanitized into a shell-level git call due to improper output encoding (CWE-116), resulting in OS command injection (CWE-78). All Gogs versions on Windows, Linux, and macOS are affected as of late May 2026. A CVE identifier has not been publicly assigned as of May 2026; monitoring NVD for formal publication. CVSS base score is 9.5 (Critical). The vulnerability was disclosed responsibly more than two months prior to the May 2026 reporting window (responsible disclosure: approximately March 2026) and remains fully unpatched by the Gogs project. A public Metasploit module now automates the full exploit chain, requiring only a valid account. MITRE ATT&CK techniques include T1059 (Command and Scripting Interpreter), T1059.004 (Unix Shell), T1190 (Exploit Public-Facing Application), T1203 (Exploitation for Client Execution), T1072 (Software Deployment Tools), T1195/T1195.002 (Supply Chain Compromise), T1078 (Valid Accounts), T1021 (Remote

Services), and T1552/T1552.001 (Unsecured Credentials). No patch is available; mitigation requires disabling or isolating the service.

Action Checklist

1. Step 1: Containment. Immediately assess whether Gogs is internet-facing. If so, restrict access to trusted IP ranges using firewall rules or network ACLs. If Gogs is internal-only, enforce network segmentation so only authorized developer workstations can reach the service. No vendor patch exists; isolation is the primary control. Reference: NIST AC-17 (Remote Access), CIS 4.4 (Firewall on Servers).
2. Step 2: Detection. Query authentication logs for all Gogs login events in the past 90 days. Review git operation logs for branch names containing '--exec' or other flag-formatted strings (e.g., strings beginning with '--'). Search host-level process logs for unexpected child processes spawned by the Gogs service binary. On Linux, query auditd or syslog for execve calls with gogs as parent process. On Windows, review Event ID 4688 (process creation) for processes with gogs.exe as parent. Alert on branch names matching the pattern '.*--exec.*' or similar flag syntax. Reference: NIST AU-6 (Audit Record Review), NIST SI-4 (System Monitoring), CIS 8.2 (Collect Audit Logs), D3-SFA (System File Analysis), D3-LAM (Local Account Monitoring).
3. Step 3: Eradication. No vendor patch has been released as of May 2026. Immediate mitigation (hours 0-24): Isolate Gogs from the network or restrict access to administrator-only. Medium-term mitigation (days 1-30): Plan migration to Gitea, Forgejo, or a hosted Git service. If replacement is not immediately feasible, evaluate disabling the rebase feature at the application or proxy level if configurable. Revoke all Gogs user credentials and rotate any secrets stored in Gogs-hosted repositories as a precaution. Reference: NIST CM-7 (Least Functionality), NIST SI-2 (Flaw Remediation), CIS 7.1 (Vulnerability Management Process), D3-CRO (Credential Rotation).
4. Step 4: Recovery. Before restoring access, verify no unauthorized accounts were added to the Gogs instance or the host OS. Audit all repositories for unauthorized commits, webhook additions, or CI/CD pipeline modifications since approximately March 2026 (disclosure date). Validate that any secrets, keys, or credentials stored in repositories have been rotated. Restore from a known-good backup only after confirming the host OS has not been backdoored. Monitor process creation logs on the Gogs host for 30 days post-remediation. Reference: NIST IR-4 (Incident Handling), NIST AU-11 (Audit Record Retention), D3-LAM (Local Account Monitoring).
5. Step 5: Post-Incident. This vulnerability exposes a control gap in third-party open-source software lifecycle management. Review your inventory of self-hosted developer tools for patch currency and active maintenance status (CIS 1.1, CIS 2.2). Establish a process for monitoring CVE disclosures and vendor advisories for all self-hosted tools, including those without active patch programs. Evaluate whether self-hosted Git services should be replaced with maintained alternatives or hosted solutions where vendor patch SLAs are contractually guaranteed. Reference: NIST SA-22 (Unsupported System Components), CIS 2.2 (Ensure Authorized Software is Currently Supported), CIS 7.2 (Remediation Process).

IR / Forensic Enrichment

Triage Priority	IMMEDIATE

Escalation Criteria	Escalate to CISO and legal counsel immediately if forensic evidence confirms RCE was achieved (child processes under gogs/gogs.exe, unauthorized OS accounts, or modified repository pipeline files) or if any compromised Gogs repository fed a production build pipeline, triggering potential supply chain breach notification obligations under applicable data protection regulations.
Recovery Notes	Restoration should not begin until host OS integrity is confirmed via memory forensics and file system timeline analysis — an authenticated RCE at CVSS 9.5 with a public exploit tool means backdoor implantation must be assumed, not ruled out by log absence alone. All repositories hosted during the exposure window (from approximately March 2026 disclosure date) must be treated as potentially tampered; any CI/CD pipelines consuming those repositories should be paused and their build artifacts re-validated from source before redeployment. Maintain elevated process creation monitoring on the replacement Git host (Forgejo or equivalent) for 30 days post-cutover, with specific alerting on any child process spawned by the Git service binary.
Forensic Artifacts	Gogs HTTP access log (`$GOGS_HOME/log/http.log` or as configured in `app.ini`) — the rebase injection exploit delivers its payload via authenticated HTTP requests containing flag-formatted strings (e.g., '--exec') in git branch name parameters; these requests appear as anomalous POST/GET entries to git operation API endpoints and are the primary exploitation fingerprint. \| Gogs SQLite database (`$GOGS_HOME/gogs.db`) or PostgreSQL equivalent — query the `branch` or `git_ref` tables for any branch name matching `^--[a-z]` and the `webhook` table for entries created after March 2026 pointing to external URLs; these are direct artifacts of the injection technique and potential supply chain backdoor establishment. | Host OS process creation logs — on Linux, auditd `execve` syscall records (type=EXECVE in `/var/log/audit/audit.log`) with `ppid` matching the Gogs service PID will show injected commands executed via the rebase vulnerability; on Windows, Security Event Log Event ID 4688 with `ParentProcessName` = `gogs.exe` captures the same exploitation artifact. | Repository commit history across all Gogs-hosted repos — run `git log --all --format='%ai %H %an %ae %s' --since='2026-03-01'` per repository to identify unauthorized commits introduced during the exposure window, particularly to CI/CD pipeline definition files (`.github/workflows/*.yml`, `Jenkinsfile`, `.gitlab-ci.yml`, `Makefile`) which are the highest-value supply chain tampering targets. | Host OS persistence mechanism artifacts — `/etc/cron.d/`, `/etc/cron.daily/`, `~/.ssh/authorized_keys` for all OS users, `/etc/systemd/system/` for new unit files (Linux), or Windows Scheduled Tasks (`schtasks /query /fo LIST /v`) and new service registrations (Event ID 7045 in System log) created after March 2026; a successful RCE via this vulnerability gives the attacker full OS access and these are the most likely persistence paths.

Per-Action IR Details

Step 1: Containment — Immediately assess whether Gogs is internet-facing. If so, restrict access to trusted IP ranges using firewall rules or network ACLs. If Gogs is internal-only, enforce network segmentation so only authorized developer workstations can reach the service. No vendor patch exists; isolation is the primary control. Reference: NIST AC-17 (Remote Access), CIS 4.4 (Firewall on Servers).

NIST Phase: Containment

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

Controls: NIST AC-17 (Remote Access), NIST AC-4 (Information Flow Enforcement), NIST SC-7 (Boundary Protection), CIS 4.4 (Implement and Manage a Firewall on Servers), CIS 4.2 (Establish and Maintain a Secure Configuration Process for Network Infrastructure)

Compensating: On Linux hosts: immediately apply iptables rules to whitelist only authorized developer CIDR ranges to Gogs HTTP/HTTPS/SSH ports (default TCP 3000 and 22): `iptables -I INPUT -p tcp --dport 3000 -j DROP` followed

by `iptables -I INPUT -s -p tcp --dport 3000 -j ACCEPT`. On Windows Server: use Windows Firewall with Advanced Security (wf.msc) to create inbound rules blocking TCP 3000 and scoped SSH. If a reverse proxy (nginx, Caddy) fronts Gogs, add `allow ; deny all;` to the Gogs location block immediately. Confirm the public exploit tool (which automates the git rebase injection via authenticated HTTP) cannot reach the service by attempting a curl from an untrusted IP after applying rules.

Evidence: Before applying firewall rules, capture current network state: run `ss -tulpn | grep gogs` (Linux) or `netstat -ano | findstr :3000` (Windows) to document all active connections to the Gogs service port. Export existing firewall rules with `iptables-save > /tmp/iptables_pre_gogs_isolation_$(date +%F).txt` (Linux) or `netsh advfirewall export` (Windows). Pull Gogs access logs at `/path/to/gogs/log/gogs.log` and the embedded Gitea-format HTTP access log for the 90-day window before isolation — these capture every authenticated HTTP request including the rebase API endpoint calls that constitute the exploit delivery path.

Step 2: Detection — Query authentication logs for all Gogs login events in the past 90 days. Review git operation logs for branch names containing '--exec', '--upload-pack', '--receive-pack', or any flag-formatted strings (e.g., strings beginning with '--'). Search host-level process logs for unexpected child processes spawned by the Gogs service binary. On Linux, query auditd or syslog for execve calls with gogs as parent process. On Windows, review Event ID 4688 (process creation) for processes with gogs.exe as parent. Alert on any branch names matching the pattern: .*--[a-z].*. Reference: NIST AU-6 (Audit Record Review), NIST SI-4 (System Monitoring), CIS 8.2 (Collect Audit Logs), D3-SFA (System File Analysis), D3-LAM (Local Account Monitoring).

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-12 (Audit Record Generation), NIST SI-4 (System and Information Integrity — System Monitoring), CIS 8.2 (Collect Audit Logs)

Compensating: On Linux without SIEM: run `grep -E '(--exec|--upload-pack|--receive-pack|--[a-z])' /path/to/gogs/log/*.log` to surface injected flag-formatted branch names in Gogs HTTP logs. Deploy Sysmon (Linux: sysmon-linux) with a config that logs ProcessCreate events where ParentImage contains 'gogs' — any child process (bash, sh, cmd.exe, powershell.exe, curl, wget, nc) is an immediate IOC. On Windows without EDR, enable audit process creation via Group Policy (Computer Configuration → Windows Settings → Security Settings → Advanced Audit Policy → Detailed Tracking → Audit Process Creation = Success) and then query with: `Get-WinEvent -FilterHashtable @{LogName='Security'; Id=4688} | Where-Object {$_.Message -match 'gogs.exe'} | Select-Object TimeCreated, Message`. Write a Sigma rule targeting ParentImage='*gogs*' with ChildImage in (cmd.exe, powershell.exe, sh, bash, python*) for offline log analysis with sigma-cli + Chainsaw.

Evidence: Capture Gogs application logs at the path configured in `app.ini` (default `$GOGS_HOME/log/gogs.log` and `$GOGS_HOME/log/http.log`) covering the full 90-day window — the exploit's rebase injection will appear as authenticated POST requests to `/api/v1/repos///git/refs` or similar git operation endpoints with branch name parameters containing flag-formatted strings. Export auditd logs (`ausearch -sc execve --start 90 days ago > /tmp/execve_audit.log`) filtered for ppid matching the Gogs process PID. On Windows, export Security Event Log filtered on Event ID 4688 with ProcessName matching `gogs.exe` as parent. Pull the Gogs database (SQLite: `gogs.db` or PostgreSQL dump) to enumerate all branch names ever created — flag any matching `^--[a-z]` as high-confidence exploitation indicators. Capture `/proc//fd` directory listing (Linux) to identify any unexpected open file descriptors or network sockets opened by the Gogs process.

Step 3: Eradication — No vendor patch is available as of the May 2026 reporting window. If you cannot isolate the service, consider replacing Gogs with a patched alternative (Gitea, Forgejo, or a hosted Git service). If replacement is not immediately feasible, disable the rebase feature at the application or proxy level if configurable. Revoke all Gogs user credentials and rotate any secrets stored in Gogs-hosted repositories as a precaution. Reference: NIST CM-7 (Least Functionality), NIST SI-2 (Flaw Remediation), CIS 7.1 (Vulnerability Management Process), D3-CRO (Credential Rotation).

NIST Phase: Eradication

Reference: NIST 800-61r3 §3.4 — Eradication

Controls: NIST CM-7 (Least Functionality), NIST SI-2 (Flaw Remediation), NIST AC-2 (Account Management), NIST AC-3 (Access Enforcement), CIS 7.1 (Establish and Maintain a Vulnerability Management Process), CIS 7.2 (Establish and Maintain a Remediation Process), CIS 5.3 (Disable Dormant Accounts)

Compensating: Because no Gogs patch exists, migration to Forgejo (a hard fork of Gitea with active security maintenance) is the operationally sound eradication path for a 2-person team. Use `gogs dump` to export all repositories and user data, then import using Forgejo's built-in Gogs migration tool. For secret rotation without a secrets manager: run `git log --all --full-history -p -- '**/*.env' '**/*.key' '**/*.pem' '**/*.conf'` across all cloned repositories to enumerate committed secrets, then rotate each discovered credential individually. Use `truffleHog3` (free, CLI) against the repository dump to automate secret detection before migration: `trufflehog filesystem /path/to/gogs/repositories/ --json > secrets_found.json`. Block the git rebase API endpoint at the nginx/Caddy reverse proxy layer as an interim measure: add `location ~* /git/rebase { return 403; }` to the proxy config.

Evidence: Before eradicating, preserve the Gogs data directory (`$GOGS_HOME/repositories/`, `$GOGS_HOME/data/`) and database as forensic images — do not wipe until post-incident review is complete. Run `git log --all --oneline --format='%H %s %d' --branches='*--*'` in each repository to catalog any commits made from injected branch names, which would confirm exploitation occurred rather than mere attempt. Document all Gogs user accounts with admin flags via direct DB query: `sqlite3 gogs.db 'SELECT name, email, is_admin, created FROM user;'` — any admin account created or promoted after the March 2026 disclosure date is a red flag. Hash and timestamp all preserved evidence with `sha256sum` before deletion.

Step 4: Recovery — Before restoring access, verify no unauthorized accounts were added to the Gogs instance or the host OS. Audit all repositories for unauthorized commits, webhook additions, or CI/CD pipeline modifications since the disclosure date (approximately March 2026). Validate that any secrets, keys, or credentials stored in repositories have been rotated. Restore from a known-good backup only after confirming the host OS has not been backdoored. Monitor process creation logs on the Gogs host for 30 days post-remediation. Reference: NIST IR-4 (Incident Handling), NIST AU-11 (Audit Record Retention), D3-LAM (Local Account Monitoring).

NIST Phase: Recovery

Reference: NIST 800-61r3 §3.5 — Recovery

Controls: NIST IR-4 (Incident Handling), NIST AU-11 (Audit Record Retention), NIST AC-2 (Account Management), NIST CM-7 (Least Functionality), CIS 5.1 (Establish and Maintain an Inventory of Accounts), CIS 6.2 (Establish an Access Revoking Process)

Compensating: To verify host OS integrity without a commercial integrity monitoring tool: on Linux, run `find / -newer /var/log/gogs_install_date.marker -type f -name '*.sh' -o -name 'cron*' 2>/dev/null` to surface recently added scripts, and `crontab -l; cat /etc/cron.*/*; systemctl list-units --type=service --state=enabled` to enumerate persistence mechanisms added post-exploitation. On Windows: `Get-ScheduledTask | Where-Object {$_.Date -gt (Get-Date).AddDays(-90)}` and `Get-Service | Where-Object {$_.StartType -eq 'Automatic'} | Select-Object Name, DisplayName, BinaryPathName`. Audit Gogs webhooks directly in the database: `sqlite3 gogs.db 'SELECT repo_id, url, content_type, created FROM webhook WHERE created > datetime("2026-03-01");'` — any webhook added after March 2026 pointing to an external URL is a high-confidence supply chain backdoor indicator. Use `git log --all --author-date-is-committer-date --format='%ai %H %an %s' --since='2026-03-01'` per repository to identify unauthorized commits.

Evidence: Before restoration, capture a full memory image of the Gogs host using `avml` (Linux) or WinPmem (Windows) — if RCE was achieved via the rebase injection, the attacker payload may only exist in memory as a reverse shell or injected thread. Collect `/etc/passwd`, `/etc/shadow`, and `last`/`lastb` output (Linux) or `net user /domain` and Security Event Log Event ID 4720 (account created), 4728 (member added to security group) entries (Windows) to verify no OS-level backdoor accounts were established. Export all Gogs repository webhook configurations and CI/CD pipeline files (`.github/workflows/`, `.gitlab-ci.yml`, or equivalent) created or modified since March 2026 as timestamped artifacts for legal hold.

Step 5: Post-Incident — This vulnerability exposes a control gap in third-party open-source software lifecycle management. Review your inventory of self-hosted developer tools for patch currency and active maintenance status (CIS 1.1, CIS 2.2). Establish a process for monitoring CVE disclosures and vendor advisories for all self-hosted tools, including those without active patch programs. Evaluate whether self-hosted Git services should be replaced with maintained alternatives or hosted solutions where vendor patch SLAs are contractually guaranteed. Reference: NIST SA-22 (Unsupported System Components), CIS 2.2 (Ensure Authorized Software is Currently Supported), CIS 7.2 (Remediation Process).

NIST Phase: Post Incident

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

Controls: NIST SA-22 (Unsupported System Components), NIST SI-2 (Flaw Remediation), NIST RA-3 (Risk Assessment), CIS 1.1 (Establish and Maintain Detailed Enterprise Asset Inventory), 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 7.2 (Establish and Maintain a Remediation Process)

Compensating: Implement a lightweight OSS risk monitoring process using free tooling: subscribe to the Gogs GitHub repository's security advisories (`github.com/gogs/gogs/security/advisories`) via RSS and route to a team email alias. Use `osv-scanner` (free, by Google) against your software inventory to detect future unpatched CVEs across all self-hosted developer tools: `osv-scanner --lockfile=/path/to/requirements.txt` or point at Docker image SBOMs. Document Gogs (and any other self-hosted tools lacking a security response program) formally under SA-22 as unsupported components, requiring a documented risk acceptance or replacement timeline signed by a system owner. Schedule a quarterly review using `cve-search` or CISA's KEV catalog RSS feed to catch exploitation-in-the-wild escalations for any tool in the developer toolchain inventory.

Evidence: Retain all forensic artifacts from this incident — Gogs logs, database snapshots, memory images, and process creation logs — for a minimum of 12 months per NIST AU-11 (Audit Record Retention) to support any downstream supply chain compromise investigation. If any repository hosted on the compromised Gogs instance fed a production CI/CD pipeline, treat all artifacts built from those repositories during the exposure window (March 2026 onward) as potentially tampered and initiate a build integrity review. Document the timeline from vulnerability disclosure to containment as a lessons-learned metric to calibrate your patch SLA for self-hosted developer tooling going forward.

Detection Guidance

Primary detection targets: (1) Gogs git operation logs - search for branch names containing '--exec' or other flag syntax matching '--[a-z]+=.*'. (2) Host process auditing - on Linux, use auditd syscall logging (execve) filtered to gogs process as parent; on macOS, use Endpoint Security Framework or EDR telemetry; on Windows, enable and query Event ID 4688 for child processes of gogs.exe. Look for unexpected interpreters (sh, bash, cmd.exe, powershell.exe, python, perl) spawned by the Gogs service. (3) Network egress - alert on outbound connections from the Gogs host to non-whitelisted external IPs, which may indicate reverse shell or C2 callback. (4) Repository integrity - diff repository state and webhook configurations against a known-good baseline from before March 1, 2026 (approximate disclosure date). (5) Credential reuse - if Gogs user credentials are shared with other internal systems, cross-reference Gogs account list against authentication logs on adjacent systems for anomalous access. D3FEND countermeasures: D3-SFA (System File Analysis), D3-LAM (Local Account Monitoring). NIST references: SI-4 (System Monitoring), AU-6 (Audit Record Review, Analysis, and Reporting), AU-12 (Audit Record Generation).

Indicators of Compromise

Type	Value	Context	Confidence
URL	`Pattern: branch names containing '--exec' or other flag-formatted arguments (e.g., --upload-pack, --receive-pack) in Gogs git operation logs`	The argument injection exploit uses a crafted branch name with --exec flag to trigger OS command execution during git rebase operations	HIGH
URL	`Metasploit module reference: Rapid7 blog post documents the module — search internal EDR/NDR for Metasploit framework signatures on traffic to Gogs HTTP endpoints`	A public Metasploit module automates the full exploit chain; detection of Metasploit user-agent strings or exploit patterns against Gogs HTTP API endpoints may indicate active exploitation attempts	MEDIUM

Framework Mappings

MITRE-ATTACK

- **T1059.004** — Unix Shell
- **T1195** — Supply Chain Compromise
- **T1552.001** — Credentials In Files
- **T1078** — Valid Accounts
- **T1552** — Unsecured Credentials
- **T1195.002** — Compromise Software Supply Chain
- **T1021** — Remote Services
- **T1203** — Exploitation for Client Execution
- **T1072** — Software Deployment Tools
- **T1059** — Command and Scripting Interpreter
- **T1190** — Exploit Public-Facing Application

NIST-800-53R5

- **CM-7** — Least Functionality
- **SI-3** — Malicious Code Protection
- **SI-4** — System Monitoring
- **SA-9** — External System Services
- **SR-2** — Supply Chain Risk Management Plan
- **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

- **AC-17** — Remote Access
- **AC-3** — Access Enforcement
- **SC-7** — Boundary Protection
- **SI-2** — Flaw Remediation
- **CA-8** — Penetration Testing
- **RA-5** — Vulnerability Monitoring and Scanning
- **SI-10** — Information Input Validation

OWASP-TOP10-2021

- **A03:2021** — Injection

CIS-V8

- **2.5** — Allowlist Authorized Software
- **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.8.8** — Management of technical vulnerabilities

MITRE ATT&CK Mapping

Technique ID	Technique Name	Tactic
T1059.004	Unix Shell	Execution
T1195	Supply Chain Compromise	Initial-Access
T1552.001	Credentials In Files	Credential-Access
T1078	Valid Accounts	Defense-Evasion
T1552	Unsecured Credentials	Credential-Access
T1195.002	Compromise Software Supply Chain	Initial-Access
T1021	Remote Services	Lateral-Movement
T1203	Exploitation for Client Execution	Execution
T1072	Software Deployment Tools	Execution
T1059	Command and Scripting Interpreter	Execution
T1190	Exploit Public-Facing Application	Initial-Access

Sources

Source	URL	Tier
Security News	https://thehackernews.com/2026/05/critical-gogs-rce-vulnerability-l...	T3
Authenticated RCE via Argument Injection in Gogs (NOT FIXED)	https://www.rapid7.com/blog/post/ve-authenticated-rce-via-argument-...	T3
Gogs.File.Upload.tree_path.CVE-2022-2024.Command.Injection	https://www.fortiguard.com/encyclopedia/ips/52746	T3
The Gogs Blind Spot: A Zero-Day Fueled Mass Compromise - Hive Pro	https://hivepro.com/threat-advisory/the-gogs-blind-spot-a-zero-day-...	T3
gogs/README.md at main - GitHub	https://github.com/gogs/gogs/blob/main/README.md	T3

Generated 2026-05-29 14:01 UTC by TJS Security Command Center