INTELLIGENCE BRIEFING
Security Command Center

TLP:CLEAR
2026-05-06 08:39 UTC

Quasar Linux (QLNX): Dual-Layer Rootkit Implant Targeting Developer Pipelines and Software Supply Chains

THREAT CAMPAIGN | **CRITICAL** | CVSS 9.5

SCC Item ID	SCC-CAM-2026-0276
Type	Threat Campaign
Severity	CRITICAL
CVSS Base Score	9.5
Affected Products	Linux systems; developer and DevOps workstations; npm, PyPI, GitHub, AWS, Docker, and Kubernetes environments
Published	2026-05-05T18:01:39
Discovery Source	Rss

Executive Summary

A newly documented Linux malware called Quasar Linux (QLNX) has been identified targeting software developers and DevOps engineers, combining a stealthy rootkit, credential harvester, and authentication backdoor in a single implant. The malware specifically harvests credentials for npm, PyPI, GitHub, AWS, Docker, and Kubernetes, giving attackers a direct path to poison software packages and cloud infrastructure used across an organization's entire software supply chain. With fewer than four antivirus engines detecting it at time of publication, most organizations have no existing defense coverage and must treat any developer workstation as a potential staging ground for downstream supply chain compromise.

Technical Analysis

QLNX is a modular Linux implant documented by Trend Micro combining four functional components: a dual-layer rootkit (user-space and kernel-space), a credential harvesting module, a remote access trojan (RAT), and a PAM (Pluggable Authentication Module) backdoor. The dual-layer rootkit architecture is specifically engineered to evade detection at both the OS user-space level and the kernel level, making traditional AV and EDR signature detection unreliable. At time of publication, only 4 of ~70 antivirus engines on VirusTotal detected QLNX. The PAM backdoor (T1556.003) enables persistent authentication bypass, surviving password resets and user remediation attempts. The credential harvester targets secrets stored on developer workstations, including npm authentication tokens, PyPI credentials, GitHub personal access tokens, AWS IAM keys, Docker registry credentials, and Kubernetes service account tokens (T1552.001, T1552.004). These credentials provide direct supply chain injection capability via T1195.001 (Compromise Software Dependencies and Development Tools) and T1195.002 (Compromise Software Supply Chain). The implant uses process injection (T1055), kernel module loading for persistence (T1547.006), command-and-control via standard application-layer protocols (T1071), and SSH for lateral movement (T1021.004). Privilege escalation (T1068) is used to establish kernel-level persistence. Log tampering (T1070.003, T1070.004) is employed to remove forensic artifacts. No CVE is assigned; this is a malware campaign. CVSS 9.5 reflects attack vector network, attack complexity low, privileges required none, user interaction none, scope changed, and confidentiality/integrity/availability all high, based on dual-layer rootkit, credential harvesting, and supply chain injection capability. Relevant CWEs: CWE-494 (Download of Code Without Integrity Check), CWE-287 (Improper Authentication), CWE-506 (Embedded Malicious Code), CWE-522 (Insufficiently Protected Credentials). Attribution is unconfirmed. Source: Trend Micro research, BleepingComputer reporting.

Action Checklist

1. Step 1: Containment. Isolate developer and DevOps workstations from production pipelines and package registries immediately. Revoke all npm tokens, PyPI API keys, GitHub PATs, AWS IAM access keys, Docker registry credentials, and Kubernetes service account tokens issued from any Linux developer workstation pending investigation. Do not wait for confirmed infection on high-value credential sets; revocation is low-cost and high-value given the detection gap.
2. Step 2: Detection. Scan Linux developer workstations for unauthorized kernel modules (lsmod, /proc/modules), unexpected PAM module entries in /etc/pam.d/, unusual SSH authorized_keys additions, and hidden processes detectable via discrepancies between /proc listings and ps output. Check for credential file access in ~/.npmrc, ~/.pypirc, ~/.aws/credentials, ~/.kube/config, and Docker config.json. Review shell history files for tampering (T1070.003). Cross-reference endpoint process trees against known-good baselines. Note: signature-based AV will miss QLNX in most configurations, behavioral and memory-based detection is required.
3. Step 3: Eradication. For any confirmed or suspected infected system, treat the host as fully compromised and reimage from a known-clean base image. Do not attempt in-place removal of a dual-layer rootkit, kernel-level persistence survives most standard removal procedures. Verify kernel module integrity post-reimage. Audit all PAM configuration files on any Linux host accessible to development teams.
4. Step 4: Recovery. After reimaging, rotate all credentials that were stored on or accessible from affected workstations, including downstream service accounts. Audit GitHub, npm, PyPI, AWS CloudTrail, and Kubernetes audit logs for any anomalous package publishes, IAM actions, or registry pushes from the affected credential set. Validate software packages published during the exposure window via code signing verification and dependency hash checks. Monitor for unauthorized pipeline executions.
5. Step 5: Post-Incident. Implement secrets management solutions (e.g., HashiCorp Vault, AWS Secrets Manager) to eliminate long-lived credential files on developer workstations. Enforce mandatory code signing for all internal package publishes to npm and PyPI. Implement kernel integrity monitoring (e.g., Falco, auditd rules for kernel module loads) as a standing detection control. Review PAM configurations enterprise-wide and enforce change alerting. Map existing EDR coverage against T1547.006 and T1556.003 to quantify detection gaps.

IR / Forensic Enrichment

Triage Priority	IMMEDIATE
Escalation Criteria	Escalate to executive leadership, legal counsel, and potentially CISA (per CISA reporting guidelines for significant cyber incidents) if: any evidence of unauthorized package versions published to npm or PyPI during the exposure window is found; any AWS IAM actions show data exfiltration (S3 GetObject at scale, RDS snapshots, Secrets Manager reads) from production accounts; or if affected developer workstations had access to codebases used by external customers, triggering potential software supply chain breach notification obligations under applicable state breach notification laws or contractual SLA requirements.
Recovery Notes	After reimaging and credential rotation, maintain a 90-day enhanced monitoring posture on all newly issued credentials (npm tokens, GitHub PATs, AWS IAM keys, Kubernetes service accounts) using automated alerts on any package publish, IAM privilege escalation, or cross-account AssumeRole activity — QLNX's supply chain targeting means delayed use of harvested credentials is a realistic threat model. Validate the integrity of every npm and PyPI package published during the exposure window by comparing published tarball SHAs against pre-compromise checksums stored in your CI/CD pipeline; any version without a verifiable pre-compromise hash must be yanked and republished from a clean environment. Continue monitoring downstream consumers of your packages via GitHub dependency graph or npm package dependents for any reports of anomalous behavior that could indicate a successful supply chain injection went undetected during your exposure window.
Forensic Artifacts	Kernel module artifacts: unsigned or anomalous .ko files in /lib/modules/$(uname -r)/kernel/ or loaded modules present in /proc/modules but absent from lsmod output — the discrepancy is QLNX's rootkit hiding mechanism (MITRE T1547.006); preserve with \`modinfo\` output and SHA-256 hashes before reimaging | PAM backdoor artifacts: modified or injected .so files in /lib/security/ or /lib64/security/ with timestamps newer than the last known-good system update, and corresponding entries in /etc/pam.d/sshd or /etc/pam.d/sudo granting access to unauthorized users (MITRE T1556.003); verify against distro package manifest checksums | Credential access audit records: auditd SYSCALL records of type open/read against ~/.npmrc, ~/.pypirc, ~/.aws/credentials, ~/.kube/config, and ~/.docker/config.json by processes other than their owner applications — the harvester component of QLNX will appear here as anomalous process names or PIDs with no corresponding /proc entry | Shell history tampering evidence: missing or truncated ~/.bash_history / ~/.zsh_history files, HISTFILE set to /dev/null in ~/.bashrc or ~/.bash_profile, or gaps in history timestamps — QLNX clears history to hinder forensic reconstruction (MITRE T1070.003); compare file modification timestamps against last login records in /var/log/wtmp | Supply chain artifact trail: npm registry publish timestamps and source IPs for all package versions published during the exposure window (retrievable via \`npm view time --json\` and npm audit log API), combined with AWS CloudTrail records of IAM key usage showing AssumeRole or CreateAccessKey events — these cross-correlated artifacts establish whether credential theft translated into active supply chain compromise

Per-Action IR Details

Step 1: Containment — Isolate developer and DevOps workstations from production pipelines and package registries immediately. Revoke all npm tokens, PyPI API keys, GitHub PATs, AWS IAM access keys, Docker registry credentials, and Kubernetes service account tokens issued from any Linux developer workstation pending investigation. Do not wait for confirmed infection — credential revocation is low-cost and high-value given the detection gap.

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

Controls: NIST IR-4 (Incident Handling), NIST AC-2 (Account Management), NIST AC-17 (Remote Access), CIS 6.2 (Establish an Access Revoking Process), CIS 5.4 (Restrict Administrator Privileges to Dedicated Administrator Accounts)

Compensating: For teams without NAC or enterprise EDR: immediately apply host-based firewall rules via iptables or ufw to block outbound connections to npm registry (registry.npmjs.org:443), PyPI (pypi.org:443), GitHub (api.github.com:443), and AWS endpoints (*.amazonaws.com:443) from affected workstations. Run: `sudo iptables -I OUTPUT -d registry.npmjs.org -j DROP` for each registry. Simultaneously execute GitHub PAT revocation via GitHub API: `curl -X DELETE -H 'Authorization: token ' https://api.github.com/applications//token` and AWS key deactivation via CLI: `aws iam update-access-key --access-key-id --status Inactive --user-name `.

Evidence: BEFORE isolating: capture full memory dump using `sudo avml /tmp/memory.lime` or `sudo dd if=/dev/mem` for offline analysis; dump active network connections with `ss -tulnp > /tmp/netstat_pre_isolation.txt` and `sudo netstat -antp >> /tmp/netstat_pre_isolation.txt`; record running processes including PPID with `ps auxef > /tmp/ps_tree_pre_isolation.txt`; capture all active kernel modules with `lsmod > /tmp/lsmod_pre_isolation.txt`; snapshot /proc/modules and /proc/net/tcp for comparison; export AWS CloudTrail events for affected IAM keys from the past 30 days using `aws cloudtrail lookup-events --lookup-attributes AttributeKey=AccessKeyId,AttributeValue=` before revoking.

Step 2: Detection — Scan Linux developer workstations for unauthorized kernel modules (lsmod, /proc/modules), unexpected PAM module entries in /etc/pam.d/, unusual SSH authorized_keys additions, and hidden processes detectable via discrepancies between /proc listings and ps output. Check for credential file access in ~/.npmrc, ~/.pypirc, ~/.aws/credentials, ~/.kube/config, and Docker config.json. Review shell history files for tampering (T1070.003). Cross-reference endpoint process trees against known-good baselines. Note: signature-based AV will miss QLNX in most configurations — behavioral and memory-based detection is required.

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-12 (Audit Record Generation), NIST IR-5 (Incident Monitoring), CIS 8.2 (Collect Audit Logs), CIS 1.1 (Establish and Maintain Detailed Enterprise Asset Inventory)

Compensating: Deploy osquery on each workstation and run: `SELECT name, path, used_by FROM kernel_modules;` to identify unauthorized modules and compare against a known-good baseline export. Use `sudo diff <(cat /tmp/lsmod_pre_isolation.txt | awk '{print $1}') <(cat /proc/modules | awk '{print $1}')` to detect hidden modules. For PAM backdoor detection (MITRE T1556.003), run: `find /etc/pam.d/ -newer /etc/passwd -ls` and `find /lib/security/ /lib64/security/ -newer /etc/passwd -ls` to identify recently modified or added PAM modules. Detect hidden processes via: `diff <(ls /proc | grep '^[0-9]' | sort -n) <(ps -e --no-headers -o pid | sort -n)` — any PID in /proc not shown by ps indicates rootkit-level hiding. For credential file exfiltration detection, use auditd: `auditctl -w /home -p r -k credential_access` then review with `ausearch -k credential_access | grep -E 'npmrc|pypirc|aws/credentials|kube/config|docker'`.

Evidence: Collect auditd logs from /var/log/audit/audit.log filtering for SYSCALL records touching ~/.aws/credentials, ~/.npmrc, ~/.pypirc, ~/.kube/config, and ~/.docker/config.json — QLNX credential harvesting will appear as read() syscalls against these files by a non-owner process. Capture PAM module hashes: `md5sum /lib/security/*.so /lib64/security/*.so > /tmp/pam_hashes.txt` and compare against vendor-supplied checksums. Export sshd logs from /var/log/auth.log or /var/log/secure filtering for 'Accepted publickey' events from unexpected source IPs. Dump bash/zsh history from all developer user home directories before potential QLNX tampering wipes them: `find /home /root -name '.*_history' -exec cp --parents {} /tmp/history_backup/ \;`. Capture /proc/[pid]/maps and /proc/[pid]/exe for all processes with open network sockets to identify injected or hidden implant memory regions.

Step 3: Eradication — For any confirmed or suspected infected system, treat the host as fully compromised and reimage from a known-clean base image. Do not attempt in-place removal of a dual-layer rootkit — kernel-level persistence survives most standard removal procedures. Verify kernel module integrity post-reimage. Audit all PAM configuration files on any Linux host accessible to development teams.

NIST Phase: Eradication

Reference: NIST 800-61r3 §3.4 — Eradication

Controls: NIST IR-4 (Incident Handling), NIST SI-2 (Flaw Remediation), NIST SI-7 (Software, Firmware, and Information Integrity), NIST CM-2 (Baseline Configuration), CIS 4.6 (Securely Manage Enterprise Assets and Software), CIS 2.1 (Establish and Maintain a Software Inventory)

Compensating: For teams without automated imaging pipelines: boot replacement systems from a verified, offline golden image (validated against a stored SHA-256 checksum of the image file) rather than attempting network-based recovery. Post-reimage, verify kernel module integrity using: `modinfo ` for all loaded modules and cross-reference signatures with `sudo modprobe --dry-run --verbose ` — unsigned or self-signed modules not in the distribution's trusted keyring are suspect. Audit PAM configurations enterprise-wide using a one-liner: `find /etc/pam.d/ /lib/security/ /lib64/security/ -type f | xargs md5sum | tee /tmp/pam_audit_$(hostname).txt` then compare checksums across hosts and against distribution package manifests via `rpm -V pam` (RHEL) or `dpkg --verify libpam-modules` (Debian). Do NOT reuse the same base image if it was stored on a system that had developer workstation access.

Evidence: Before reimaging, preserve a full forensic disk image using: `sudo dc3dd if=/dev/sda hash=sha256 log=/tmp/dc3dd_$(hostname).log | gzip > /external/$(hostname)_disk.img.gz` to maintain chain of custody for any downstream legal or regulatory review. Capture the full /etc/pam.d/ directory tree and all PAM .so files: `tar czf /tmp/pam_evidence_$(hostname).tar.gz /etc/pam.d/ /lib/security/ /lib64/security/`. Extract and preserve any unknown or unsigned kernel modules from /lib/modules/$(uname -r)/ before the reimage destroys them — these are the primary QLNX persistence artifacts. Document the exact kernel version, distribution, and patch level of the compromised host for correlation with any future QLNX intelligence reporting.

Step 4: Recovery — After reimaging, rotate all credentials that were stored on or accessible from affected workstations, including downstream service accounts. Audit GitHub, npm, PyPI, AWS CloudTrail, and Kubernetes audit logs for any anomalous package publishes, IAM actions, or registry pushes from the affected credential set. Validate software packages published during the exposure window via code signing verification and dependency hash checks. Monitor for unauthorized pipeline executions.

NIST Phase: Recovery

Reference: NIST 800-61r3 §3.5 — Recovery

Controls: NIST IR-4 (Incident Handling), NIST AC-2 (Account Management), 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 5.1 (Establish and Maintain an Inventory of Accounts)

Compensating: For npm supply chain validation: query npm registry for all package versions published under compromised accounts during the exposure window using `npm view time --json` and compare publish timestamps against known developer activity. Verify package integrity: `npm pack @ && shasum -a 256 ` then compare against previously recorded checksums. For AWS CloudTrail: run `aws cloudtrail lookup-events --lookup-attributes AttributeKey=Username,AttributeValue= --start-time --query 'Events[*].[EventTime,EventName,Resources]'` filtering specifically for CreateAccessKey, PutRolePolicy, AssumeRole, and ec2:RunInstances events — these indicate lateral movement from harvested credentials. For Kubernetes: retrieve the audit log from the API server (typically /var/log/kube-apiserver-audit.log) and filter for service account token usage with verbs create, update, delete targeting namespace-scoped resources during the exposure window.

Evidence: AWS CloudTrail logs filtered to the harvested IAM key's access key ID showing all API calls during exposure window, with particular attention to sts:AssumeRole (lateral movement), iam:CreateAccessKey (persistence), s3:PutObject to unusual buckets (exfiltration), and lambda:CreateFunction (supply chain backdoor). GitHub audit log export (available via GitHub Enterprise audit log API or GitHub.com audit log export) filtered for the compromised PAT showing package:publish, repository:push, and workflow_run events. npm registry publish logs for all packages owned by the compromised account — specifically request logs showing the source IP of any publishes, which can confirm whether QLNX's credential theft resulted in unauthorized package versions. Kubernetes API server audit logs showing service account token usage with kubectl auth can-i calls or RBAC escalation attempts during the exposure window.

Step 5: Post-Incident — Implement secrets management solutions (e.g., HashiCorp Vault, AWS Secrets Manager) to eliminate long-lived credential files on developer workstations. Enforce mandatory code signing

for all internal package publishes to npm and PyPI. Implement kernel integrity monitoring (e.g., Falco, auditd rules for kernel module loads) to quantify detection gaps.

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-7 (Software, Firmware, and Information Integrity), NIST SI-4 (System Monitoring), NIST AU-2 (Event Logging), CIS 7.1 (Establish and Maintain a Vulnerability Management Process), CIS 7.2 (Establish and Maintain a Remediation Process), CIS 8.2 (Collect Audit Logs)

Compensating: For teams without commercial secrets management: deploy Vault Community Edition (free, open source) with short-lived dynamic credentials for AWS via the AWS secrets engine — this eliminates ~/.aws/credentials files that QLNX explicitly targets. For kernel integrity monitoring without commercial EDR: deploy Falco (free, CNCF project) with the default ruleset plus custom rules targeting QLNX-specific behaviors: `- rule: Unexpected Kernel Module Load triggered by: evt.type = init_module and not proc.name in (allowed_module_loaders)`. Supplement with auditd rules: `auditctl -a always,exit -F arch=b64 -S init_module -S finit_module -k kernel_module_load` to alert on any kernel module load event — QLNX's rootkit layer requires init_module or finit_module syscalls to install. For PAM change alerting, use auditd: `auditctl -w /etc/pam.d/ -p wa -k pam_changes` and ship alerts to a centralized syslog server.

Evidence: Lessons-learned documentation should capture: the specific credential file paths QLNX accessed on each compromised workstation (from auditd records), the time delta between initial compromise and first credential use by attackers (from CloudTrail/GitHub audit logs), and whether any EDR or AV tools generated alerts during the dwell period — this gap analysis directly informs MITRE ATT&CK coverage mapping for T1547.006 (Kernel Modules and Extensions) and T1556.003 (Pluggable Authentication Modules). Produce a supply chain exposure report listing every npm/PyPI package version, GitHub Actions workflow, AWS IAM role, and Kubernetes service account that was accessible from compromised credentials during the exposure window — this becomes the standing watchlist for any delayed supply chain compromise indicators.

Detection Guidance

Primary detection targets for QLNX are behavioral, not signature-based. Focus on: (1) Kernel module anomalies, compare lsmod output against a known-good baseline; look for modules not present in /lib/modules or loaded from non-standard paths. (2) PAM backdoor, audit /etc/pam.d/ and /lib/security/ for unexpected or recently modified PAM modules; alert on any PAM configuration change outside change management windows. (3) Credential file access, use auditd rules to alert on reads of ~/.aws/credentials, ~/.kube/config, ~/.npmrc, ~/.pypirc, and Docker config.json by any process other than the expected CLI tools. (4) Process hiding, compare /proc/ enumeration against ps and top output; discrepancies indicate user-space rootkit hooks. (5) Log tampering, monitor shell history files (~/.bash_history, ~/.zsh_history) for truncation or unexpected modification timestamps (T1070.003); alert on HISTFILE unset or HISTSIZE=0 in running processes. (6) Outbound C2, inspect outbound connections from developer workstations for non-standard intervals or beaconing patterns on ports 80/443/22 to unfamiliar destinations (T1071, T1090.001). Tools: Falco rules for kernel module loads and PAM changes, auditd for file access monitoring, osquery for process and module inventory. QLNX source research: Trend Micro (https://www.trendmicro.com/en_us/research/26/e/quasar-linux-qlnx-a-silent-foothold-in-the-software-supply-chain.html). Note: The Trend Micro URL is source-provided and has not been independently verified active by this system, validate before use.

Indicators of Compromise

Type	Value	Context	Confidence
HASH	`[not published at time of analysis]`	QLNX malware sample hashes were not included in the source data provided. Refer to the Trend Micro research report for published IOCs.	LOW
DOMAIN	`[not published at time of analysis]`	C2 infrastructure domains were not included in the source data provided. Refer to the Trend Micro research report for published network IOCs.	LOW

Framework Mappings

MITRE-ATTACK

- **T1003** — OS Credential Dumping
- **T1547.006** — Kernel Modules and Extensions
- **T1611** — Escape to Host
- **T1574.006** — Dynamic Linker Hijacking
- **T1552.004** — Private Keys
- **T1547** — Boot or Logon Autostart Execution
- **T1070.003** — Clear Command History
- **T1195.001** — Compromise Software Dependencies and Development Tools
- **T1090.001** — Internal Proxy
- **T1059** — Command and Scripting Interpreter
- **T1055** — Process Injection
- **T1552.001** — Credentials In Files
- **T1610** — Deploy Container
- **T1543.002** — Systemd Service
- **T1036.005** — Match Legitimate Resource Name or Location
- **T1195.002** — Compromise Software Supply Chain
- **T1071** — Application Layer Protocol
- **T1021.004** — SSH
- **T1068** — Exploitation for Privilege Escalation
- **T1543** — Create or Modify System Process
- **T1556.003** — Pluggable Authentication Modules
- **T1070.004** — File Deletion
- **T1059.004** — Unix Shell
- **T1056.001** — Keylogging

NIST-800-53R5

- **AC-6** — Least Privilege

- **IA-5** — Authenticator Management
- **SI-4** — System Monitoring
- **CM-7** — Least Functionality
- **SI-3** — Malicious Code Protection
- **SI-7** — Software, Firmware, and Information Integrity
- **SC-7** — Boundary Protection
- **SA-9** — External System Services
- **SR-3** — Supply Chain Controls and Processes
- **CA-7** — Continuous Monitoring
- **SI-2** — Flaw Remediation
- **CM-3** — Configuration Change Control
- **IA-2** — Identification and Authentication (Organizational Users)
- **IA-8** — Identification and Authentication (Non-Organizational Users)
- **SR-2** — Supply Chain Risk Management Plan

OWASP-TOP10-2021

- **A08:2021** — Software and Data Integrity Failures
- **A07:2021** — Identification and Authentication Failures
- **A04:2021** — Insecure Design

CIS-V8

- **2.5** — Allowlist Authorized Software
- **2.6** — Allowlist Authorized Libraries
- **6.3** — Require MFA for Externally-Exposed Applications
- **6.4** — Require MFA for Remote Network Access
- **6.5** — Require MFA for Administrative Access
- **5.2** — Use Unique Passwords
- **15.1** — Establish and Maintain an Inventory of Service Providers
- **8.2** — Collect Audit Logs

SOC2-TSC

- **CC6.1** — The entity implements logical access security software, infrastructure, and architectures over protected information assets
- **CC9.2** — Manages risks associated with vendors and business partners

HIPAA-SECURITY

- **164.312(d)** — Person or Entity Authentication
- **164.308(a)(5)(ii)(D)** — Password Management

ISO-27001-2022

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

- **A.5.23** — Information security for use of cloud services

NIST-CSF-2

- **GV.SC-01** — Cybersecurity supply chain risk management program
- **DE.CM-01** — Networks and network services are monitored

MITRE ATT&CK Mapping

Technique ID	Technique Name	Tactic
T1003	OS Credential Dumping	Credential-Access
T1547.006	Kernel Modules and Extensions	Persistence
T1611	Escape to Host	Privilege-Escalation
T1574.006	Dynamic Linker Hijacking	Persistence
T1552.004	Private Keys	Credential-Access
T1547	Boot or Logon Autostart Execution	Persistence
T1070.003	Clear Command History	Defense-Evasion
T1195.001	Compromise Software Dependencies and Development Tools	Initial-Access
T1090.001	Internal Proxy	Command-And-Control
T1059	Command and Scripting Interpreter	Execution
T1055	Process Injection	Defense-Evasion
T1552.001	Credentials In Files	Credential-Access
T1610	Deploy Container	Defense-Evasion
T1543.002	Systemd Service	Persistence
T1036.005	Match Legitimate Resource Name or Location	Defense-Evasion
T1195.002	Compromise Software Supply Chain	Initial-Access
T1071	Application Layer Protocol	Command-And-Control
T1021.004	SSH	Lateral-Movement
T1068	Exploitation for Privilege Escalation	Privilege-Escalation
T1543	Create or Modify System Process	Persistence
T1556.003	Pluggable Authentication Modules	Credential-Access

Technique ID	Technique Name	Tactic
T1070.004	File Deletion	Defense-Evasion
T1059.004	Unix Shell	Execution
T1056.001	Keylogging	Collection

Sources

Source	URL	Tier
Security News	https://www.bleepingcomputer.com/news/security/new-stealthy-quasar-...	T3
Quasar Linux (QLNX) – A Silent Foothold in the Supply Chain	https://www.trendmicro.com/en_us/research/26/e/quasar-linux-qlnx-a-...	T3
Security is in the Room with You: Protect Your Developer Workstation	https://www.linkedin.com/posts/nathan-rampersaud-9ab907206_ive-been...	T3
pip Install Backdoored Your Kubernetes Cluster (Explained)	https://www.kubeblogs.com/your-pip-install-backdoored-kubernetes-cl...	T3
How TeamPCP turned Aqua Security's own Trivy scanner into a ...	https://thenewstack.io/teampcp-trivy-supply-chain-attack/	T3

Generated 2026-05-06 08:39 UTC by TJS Security Command Center