INTELLIGENCE BRIEFING
Security Command Center
TLP:CLEAR
2026-03-31 06:19 UTC

RoadK1ll Implant Turns Compromised Hosts Into Network Relay Points via WebSocket Tunneling

THREAT CAMPAIGN | **HIGH** | CVSS 7.5

SCC Item ID	SCC-CAM-2026-0127
Type	Threat Campaign
Severity	HIGH
CVSS Base Score	7.5
Affected Products	Compromised hosts running Node.js runtime environment; no specific vendor product targeted
Published	2026-03-30T16:49:22
Discovery Source	Rss

Executive Summary

Blackpoint Cyber discovered RoadK1ll, a Node.js-based implant, during an active incident response engagement. The implant converts compromised hosts into silent network relay points, allowing attackers to reach internal systems that are otherwise isolated from the internet. Organizations with Node.js present on internal hosts face elevated risk of lateral movement that bypasses perimeter controls and limits forensic footprint by avoiding traditional persistence mechanisms.

Technical Analysis

RoadK1ll is a Node.js implant that establishes outbound WebSocket tunnels (T1572, T1071.001) from a compromised host to attacker-controlled infrastructure. It multiplexes multiple TCP connections over a single tunnel (T1090, T1090.001), enabling lateral movement (T1021) to internal network segments unreachable from the perimeter. The implant leverages the compromised host's internal trust relationships to reach otherwise isolated systems. Execution relies on Node.js runtime (T1059.007) present on the host. File delivery likely occurs via ingress tool transfer (T1105). The implant does not use traditional persistence mechanisms such as registry keys, scheduled tasks, or startup entries; it relies on process-based execution and built-in reconnection logic, reducing detection via persistence-focused forensics. No CVE is associated with this campaign. Relevant CWEs are CWE-441 (Unintended Proxy or Intermediary) and CWE-923 (Improper Restriction of Communication Channel to Intended Endpoints). No patch applies; this is a capability, not a software vulnerability. Attribution is unknown as of Blackpoint's disclosure. Source: Blackpoint Cyber via BleepingComputer.

Action Checklist

1. Step 1: Containment. After confirming compromise, prioritize isolation of any host exhibiting suspicious outbound WebSocket connections. [Note: Coordinate isolation with business stakeholders to assess production impact.] Identify hosts with Node.js runtime installed across your environment, particularly on internal servers and workstations not requiring it. Block outbound WebSocket traffic from hosts where it has no business justification, at the network layer.
2. Step 2: Detection. Query firewall and proxy logs for outbound HTTP Upgrade requests (Connection: Upgrade, Upgrade: websocket) originating from internal hosts. Correlate with process telemetry to identify node.exe or node processes making network connections. Review EDR telemetry for Node.js processes spawned outside expected application contexts. Look for sustained, low-volume outbound connections to unfamiliar external IPs on ports 80 or 443 with atypical connection durations. Check for multiplexed TCP traffic patterns over single long-lived connections.
3. Step 3: Eradication. Terminate identified node.exe processes associated with the implant. Remove the implant script from disk. Because RoadK1ll does not use registry-based or scheduled task persistence, killing the process and removing the file eliminates the execution path. Audit Node.js installation necessity on affected hosts and uninstall where not required. Rotate credentials for any accounts or services accessible from compromised hosts, as internal trust relationships may have been exploited.
4. Step 4: Recovery. Validate that no residual node.exe processes with external WebSocket connections remain. Confirm firewall rules blocking unauthorized outbound WebSocket traffic are active and logging. Conduct a network review of internal segments the compromised host could reach to assess lateral movement scope. Monitor for reconnection attempts from the same host or peer hosts for at least 30 days post-remediation.
5. Step 5: Post-Incident. Assess the control gap that permitted Node.js runtime to exist on hosts where it had no justified business function (application whitelisting gap). Review network segmentation effectiveness; the implant's value depends on the compromised host having broad internal reach. Reduce that reach via micro-segmentation or host-based firewall policy. Evaluate whether outbound WebSocket traffic is inspected or restricted by existing proxy and firewall policy. Update detection rules in SIEM and EDR to flag node.exe initiating outbound network connections outside approved application paths.

IR / Forensic Enrichment

Triage Priority	URGENT
Escalation Criteria	Escalate to senior IR leadership and legal/compliance if NetFlow or proxy analysis confirms the RoadK1ll relay was used to access systems containing PII, PHI, PCI-scoped data, or privileged credential stores, as this may trigger breach notification obligations under HIPAA, PCI DSS, or applicable state privacy law, or if lateral movement evidence shows the attacker pivoted beyond the initial compromised host to additional internal systems.

Recovery Notes	After eradication, re-image any host where the initial access vector that delivered or executed the RoadK1ll script cannot be conclusively identified, as an unresolved entry point means the implant can be redeployed. Maintain continuous Sysmon Event ID 3 monitoring for node.exe on all internal hosts for a minimum of 30 days, and extend to 60 days if the attacker's dwell time prior to discovery exceeded two weeks. Validate that all internal services reachable through the relay (identified from the network segment access map) show no anomalous authentication events in their own logs during the implant's active period.
Forensic Artifacts	In-memory process dump of the node.exe implant process (via ProcDump or gcore): contains the in-memory RoadK1ll JavaScript source, active WebSocket connection objects, and any buffered relay data that was never written to disk — the highest-value artifact for this specific implant. | Filesystem copy and SHA-256 hash of the RoadK1ll .js implant script with preserved MAC timestamps: 'stat' output or a forensic image of the file's inode captures created/modified/accessed times to establish when the implant was staged, which correlates to the initial access timeline. | Proxy or web gateway logs containing raw HTTP request headers for the WebSocket upgrade handshake: specifically the 'Upgrade: websocket', 'Connection: Upgrade', and 'Sec-WebSocket-Key' header values from the internal host IP — these uniquely identify RoadK1ll's tunnel establishment versus generic HTTPS traffic. | Sysmon Event ID 1 (ProcessCreate) records for node.exe: the CommandLine field will contain the path to the implant script and any arguments passed to it, and the ParentImage and ParentCommandLine fields identify the process that launched the implant, directly revealing the initial access or execution vector. | NetFlow or firewall session logs for the compromised host's IP covering the implant's active period: long-lived single TCP sessions to an external IP on port 443 with low symmetric byte counts and no HTTP response body patterns are the network-layer signature of the WebSocket relay tunnel, and the destination IP provides attacker infrastructure for threat intelligence enrichment and blocking.

Per-Action IR Details

Step 1: Containment — Identify hosts with Node.js runtime installed across your environment, particularly on internal servers and workstations not requiring it. Isolate any host exhibiting suspicious outbound WebSocket connections (port 443 or 80 with ws:// or wss:// upgrade headers) to attacker-controlled IPs. Block outbound WebSocket traffic from hosts where it has no business justification, at the network layer.

NIST Phase: Containment

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

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

Compensating: Run 'Get-Command node -ErrorAction SilentlyContinue' across Windows hosts via PowerShell remoting, or 'which node; find / -name node -type f 2>/dev/null' on Linux, to enumerate Node.js presence without an asset management platform. For network isolation without an enterprise NAC, apply an inbound/outbound host-based firewall rule blocking TCP 80/443 on the suspect host using 'netsh advfirewall firewall add rule name=BLOCK_NODE dir=out program=node.exe action=block' on Windows, or an iptables OUTPUT rule scoped to the node process UID on Linux. Capture a netstat snapshot ('netstat -anop tcp') before isolating to preserve connection state.

Evidence: Before isolating, capture: (1) full 'netstat -anop tcp' or 'ss -tupn' output to document active WebSocket connections from node.exe, including remote IP, port, and local PID; (2) process tree via 'wmic process get ProcessId,ParentProcessId,CommandLine' (Windows) or 'ps auxf' (Linux) to identify node.exe parent process and full command line including the implant script path; (3) network flow data from firewall/proxy logs showing HTTP Upgrade handshake (Connection: Upgrade, Upgrade: websocket headers) to the suspected attacker-controlled IP. This evidence establishes the relay channel before it is severed.

Step 2: Detection — Query firewall and proxy logs for outbound HTTP Upgrade requests (Connection: Upgrade, Upgrade: websocket) originating from internal hosts. Correlate with process telemetry to identify

node.exe or node processes making network connections. Review EDR telemetry for Node.js processes spawned outside expected application contexts. Look for sustained, low-volume outbound connections to unfamiliar external IPs on ports 80 or 443 with atypical connection durations. Check for multiplexed TCP traffic patterns over single long-lived connections.

NIST Phase: Detection Analysis

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

Controls: NIST SI-4 (System Monitoring), NIST AU-2 (Event Logging), NIST AU-6 (Audit Record Review, Analysis, and Reporting), NIST IR-5 (Incident Monitoring), CIS 8.2 (Collect Audit Logs), CIS 7.1 (Establish and Maintain a Vulnerability Management Process)

Compensating: Deploy Sysmon with a config that includes NetworkConnect events (Event ID 3) and ProcessCreate events (Event ID 1); filter Event ID 3 where Image ends in 'node.exe' and DestinationPort is 80 or 443. Use this PowerShell one-liner against Windows Event Logs: 'Get-WinEvent -LogName "Microsoft-Windows-Sysmon/Operational" | Where-Object {$_.Id -eq 3 -and $_.Message -match "node.exe"}'. On Linux, use osquery: 'SELECT pid, name, remote_address, remote_port, local_address FROM process_open_sockets WHERE name = "node" AND remote_port IN (80, 443);'. For network-layer detection without a proxy, run a Wireshark capture filtered on 'tcp.port == 443 && http.upgrade == "websocket"' on a network tap or span port to identify long-lived WebSocket sessions from internal hosts. Apply the publicly available Sigma rule for suspicious Node.js network connections (community rules under process_network or proxy categories) against any locally parsed logs.

Evidence: Capture before performing analysis: (1) Sysmon Event ID 1 (ProcessCreate) logs for node.exe showing full CommandLine including the RoadK1ll script filename and parent process (look for unusual parents such as cmd.exe, powershell.exe, or a compromised application process); (2) Sysmon Event ID 3 (NetworkConnect) logs for node.exe showing DestinationIp and DestinationPort 80/443 with Initiated=true; (3) proxy or firewall logs containing raw HTTP headers for the Upgrade handshake — specifically 'Upgrade: websocket' and 'Connection: Upgrade' in the request from the internal host IP; (4) DNS query logs for the hostname the implant connected to, to support attacker infrastructure attribution; (5) connection duration and byte-count fields from NetFlow or firewall session logs showing a single TCP session remaining open for minutes to hours with low symmetric traffic volume, characteristic of a WebSocket tunnel keepalive pattern.

Step 3: Eradication — Terminate identified node.exe processes associated with the implant. Remove the implant script from disk. Because RoadK1ll does not use registry-based or scheduled task persistence, killing the process and removing the file eliminates the execution path. Audit Node.js installation necessity on affected hosts and uninstall where not required. Rotate credentials for any accounts or services accessible from compromised hosts, as internal trust relationships may have been exploited.

NIST Phase: Eradication

Reference: NIST 800-61r3 §3.4 — Eradication

Controls: NIST IR-4 (Incident Handling), NIST SI-2 (Flaw Remediation), NIST CM-7 (Least Functionality), NIST IA-5 (Authenticator Management), CIS 2.3 (Address Unauthorized Software), CIS 5.2 (Use Unique Passwords), CIS 4.6 (Securely Manage Enterprise Assets and Software)

Compensating: Before terminating, collect a full process memory dump for forensic retention: use ProcDump ('procdump.exe -ma roadkill_implant.dmp') on Windows or 'gcore ' on Linux. Then terminate: 'Stop-Process -Id -Force' (Windows) or 'kill -9 ' (Linux). Locate and hash the implant script before deletion: 'Get-FileHash -Algorithm SHA256' or 'sha256sum ' — retain the hash in your incident record. Verify no additional copies exist: 'Get-ChildItem -Path C:\ -Recurse -Filter *.js | Select-String -Pattern "websocket" -List' (Windows) or 'grep -rl "websocket" /home /tmp /var /opt 2>/dev/null' (Linux). For credential rotation, prioritize any service accounts or API keys whose credentials were stored on the compromised host or accessible from it, as the relay nature of RoadK1ll means attackers may have proxied authentication attempts to internal services.

Evidence: Preserve before eradication: (1) full memory dump of the node.exe process containing the in-memory implant script, WebSocket connection state, and any buffered tunnel data — this is the primary forensic artifact for RoadK1ll since the implant lives largely in process memory; (2) SHA-256 hash and a forensic copy of the RoadK1ll JavaScript file from disk, preserving filesystem metadata (created, modified, accessed timestamps via 'stat' or forensic

imaging); (3) the parent process that launched node.exe and the full command-line arguments used to invoke the implant script, from Sysmon Event ID 1 or EDR telemetry — this identifies the initial access vector that dropped or executed the implant; (4) a list of internal IPs and services contacted through the relay tunnel, recoverable from NetFlow data or from the memory dump, to bound lateral movement scope.

Step 4: Recovery — Validate that no residual node.exe processes with external WebSocket connections remain. Confirm firewall rules blocking unauthorized outbound WebSocket traffic are active and logging. Conduct a network review of internal segments the compromised host could reach to assess lateral movement scope. Monitor for reconnection attempts from the same host or peer hosts for at least 30 days post-remediation.

NIST Phase: Recovery

Reference: NIST 800-61r3 §3.5 — Recovery

Controls: NIST IR-4 (Incident Handling), NIST SI-4 (System Monitoring), NIST AU-6 (Audit Record Review, Analysis, and Reporting), NIST SC-7 (Boundary Protection), CIS 8.2 (Collect Audit Logs), CIS 4.4 (Implement and Manage a Firewall on Servers)

Compensating: Automate residual-process validation with a scheduled PowerShell task running every 15 minutes: 'Get-NetTCPConnection -State Established | Where-Object {$_.OwningProcess -in (Get-Process node -ErrorAction SilentlyContinue).Id}' — alert on any output. For 30-day reconnection monitoring without a SIEM, configure a Windows Event Log subscription or a cron-driven osquery query ('SELECT * FROM process_open_sockets WHERE name = "node" AND remote_port IN (80, 443);') that writes results to a local log file and emails on non-empty output. Verify firewall block rules are logging by generating a test outbound connection attempt from the host and confirming a deny entry appears in the firewall log. Use Wireshark or tcpdump on a network tap to spot-check for WebSocket Upgrade headers originating from the remediated host or its network neighbors during the monitoring window.

Evidence: Document for recovery validation: (1) timestamped 'netstat -anop tcp' and process list output from the remediated host confirming absence of node.exe with external connections — this serves as the post-remediation clean-state baseline; (2) firewall rule export or screenshot showing the blocking rule for outbound WebSocket traffic (port 80/443 with Upgrade header) is active and the logging flag is enabled; (3) network segment access map for the compromised host — enumerate all VLANs, subnets, and internal services reachable from that host's IP prior to remediation, to define the lateral movement blast radius that must be reviewed; (4) any new outbound connection attempts to the same attacker-controlled IP or infrastructure block observed in firewall deny logs during the 30-day monitoring window, which would indicate reinfection or a second implant on a peer host.

Step 5: Post-Incident — Assess the control gap that permitted Node.js runtime to exist on hosts where it had no justified business function (application whitelisting gap). Review network segmentation effectiveness — the implant's value depends on the compromised host having broad internal reach; reduce that reach via micro-segmentation or host-based firewall policy. Evaluate whether outbound WebSocket traffic is inspected or restricted by existing proxy and firewall policy. Update detection rules in SIEM and EDR to flag node.exe initiating outbound network connections outside approved application paths.

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 CM-7 (Least Functionality), NIST SC-7 (Boundary Protection), NIST SI-4 (System Monitoring), 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 4.2 (Establish and Maintain a Secure Configuration Process for Network Infrastructure)

Compensating: For application whitelisting without enterprise tooling, deploy Windows AppLocker or WDAC (Windows Defender Application Control) policies that deny execution of node.exe outside approved installation paths (e.g., block 'C:\Users*\node.exe' and 'C:\Temp*\node.exe' while permitting only 'C:\Program Files\nodejs\node.exe' for hosts with a business justification). On Linux, use auditd rules: 'auditctl -a always,exit -F arch=b64 -S execve -F exe=/usr/bin/node -k node_exec' to log all Node.js executions for review. For detection rule uplift, author a Sigma rule targeting Sysmon Event ID 3 where Image='*\node.exe', Initiated=true, and DestinationPort IN (80, 443), and apply it against your log pipeline. For segmentation assessment, run a network reachability audit from the compromised host's

IP using nmap ('nmap -sn 10.0.0.0/8') to document actual internal reach versus intended reach, and use that gap to prioritize host-based firewall policy additions.

Evidence: Document for lessons-learned and control improvement: (1) the software inventory gap report showing which hosts had Node.js installed without a documented business justification — this quantifies the application whitelisting control failure; (2) the network segmentation gap analysis showing internal subnets and services the compromised host could reach versus what was intended per network architecture documentation — this quantifies the blast radius the relay capability provided to the attacker; (3) the proxy/firewall policy review output confirming whether outbound WebSocket Upgrade requests were inspectable or were passed as opaque TLS tunnels on port 443, which explains why the RoadK1ll tunnel evaded perimeter detection; (4) the finalized Sigma or EDR detection rule for node.exe initiating outbound network connections, saved to the organization's detection engineering repository with the RoadK1ll incident as the source justification.

Detection Guidance

Focus detection on three signal types. First, network: query proxy and firewall logs for HTTP 101 Switching Protocols responses or outbound requests containing 'Upgrade: websocket' headers from hosts where WebSocket use is not expected. Flag long-lived connections (duration >5 minutes) on ports 80 or 443 to external IPs from internal servers. Second, process: use EDR telemetry to identify node.exe or node processes making direct outbound TCP connections, especially where the parent process is unusual (cmd.exe, powershell.exe, or an unexpected application). Alert on Node.js processes executing scripts from temp directories, user profile paths, or paths inconsistent with installed applications. Third, behavioral: look for a single external IP receiving multiplexed traffic that correlates with subsequent lateral movement attempts to internal hosts. Detection success depends on process and network telemetry; endpoint-based persistence detection will not fire. Alert tuning should prioritize low-noise, high-confidence signals: node.exe with unexpected parent process + external connection + absence of known app context. MITRE ATT&CK coverage: T1572 (Protocol Tunneling), T1090 (Proxy), T1071.001 (Application Layer Protocol: Web Protocols), T1021 (Remote Services).

Indicators of Compromise

Type	Value	Context	Confidence
NOTE	`No IOCs published`	Blackpoint Cyber has not released specific IOC values (IPs, domains, file hashes) in the public disclosure as of this item's sourcing. Monitor the Blackpoint Cyber blog and threat intelligence feeds for updates. Source quality score for this item is 0.64, reflecting reliance on third-tier sources.	LOW

Framework Mappings

MITRE-ATTACK

- **T1071.001** — Web Protocols
- **T1090** — Proxy

- **T1572** — Protocol Tunneling
- **T1021** — Remote Services
- **T1105** — Ingress Tool Transfer
- **T1059.006** — Python
- **T1090.001** — Internal Proxy
- **T1059.007** — JavaScript

NIST-800-53R5

- **AC-17** — Remote Access
- **AC-3** — Access Enforcement
- **CM-7** — Least Functionality
- **IA-2** — Identification and Authentication (Organizational Users)
- **CA-7** — Continuous Monitoring
- **SC-7** — Boundary Protection
- **SI-3** — Malicious Code Protection
- **SI-4** — System Monitoring

MITRE ATT&CK Mapping

Technique ID	Technique Name	Tactic
T1071.001	Web Protocols	Command-And-Control
T1090	Proxy	Command-And-Control
T1572	Protocol Tunneling	Command-And-Control
T1021	Remote Services	Lateral-Movement
T1105	Ingress Tool Transfer	Command-And-Control
T1059.006	Python	Execution
T1090.001	Internal Proxy	Command-And-Control
T1059.007	JavaScript	Execution

Sources

Source	URL	Tier
Security News	https://www.bleepingcomputer.com/news/security/new-roadk1ll-websock...	T3
Eight for One: Multiple Vulnerabilities Fixed in the Node.js Runtime	https://www.endorlabs.com/learn/eight-for-one-multiple-vulnerabilit...	T3

Source	URL	Tier
TeamPCP deploys CanisterWorm on NPM following Trivy compromise	https://www.aikido.dev/blog/teampcp-deploys-worm-npm-trivy-compromise	T3
Node.js 20.x < 20.20.2 Multiple Vulnerabilities (Tuesday, Marc...	https://www.tenable.com/plugins/nessus/303494	T3
THE KNOWNSEC LEAK: Yet Another Leak of China's Contractor ...	https://dti.domaintools.com/research/the-knownsec-leak-yet-another-...	T3

Generated 2026-03-31 06:19 UTC by TJS Security Command Center