INTELLIGENCE BRIEFING
Security Command Center
TLP:CLEAR
2026-05-27 06:38 UTC

Critical Integer Overflow in golang.org/x/sys/windows NewNTUnicodeString Affects Azure Linux Ingress Controller

CVE VULNERABILITY | **CRITICAL** | CVSS 9.8

SCC Item ID	SCC-CVE-2026-0224
Type	CVE Vulnerability
CVE ID	CVE-2026-39824
Severity	CRITICAL
CVSS Base Score	9.8
EPSS Score	0.0002 (5th percentile)
Affected Products	Microsoft azl3 application-gateway-kubernetes-ingress 1.7.7-3 on Azure Linux 3.0; golang.org/x/sys/windows (upstream Go system library)
Published	2026-05-27T01:40:27
Discovery Source	Msrc Patch Tuesday

Executive Summary

A critical memory corruption vulnerability (CVE-2026-39824, CVSS 9.8) was disclosed in Microsoft's May 2026 Patch Tuesday affecting the Azure Linux 3.0 application-gateway-kubernetes-ingress package version 1.7.7-3. The flaw resides in a core Go system library function and can be triggered by a specially crafted input string, potentially enabling remote code execution against Kubernetes ingress controllers running on Azure Linux. Organizations using this specific package to route traffic into Azure-hosted Kubernetes clusters should treat this as an emergency patching event.

Technical Analysis

CVE-2026-39824 is an integer overflow (CWE-190) in the NewNTUnicodeString function of the golang.org/x/sys/windows package. During string length calculation, a malformed input can cause the integer to wrap, producing an undersized heap allocation. A subsequent write to that buffer constitutes a heap-based buffer overflow (CWE-122), which can corrupt adjacent memory regions and, under exploitable conditions, enable remote code execution. MITRE ATT&CK techniques T1190 (Exploit Public-Facing Application) and T1211 (Exploitation for Defense Evasion) are mapped to this CVE. The confirmed affected package is Microsoft azl3 application-gateway-kubernetes-ingress 1.7.7-3 on Azure Linux 3.0. The upstream vulnerability is in golang.org/x/sys/windows, so any Go application that imports this package and passes untrusted string input to

NewNTUnicodeString may carry residual risk. CVSS base score is 9.8 with vector CVSS:3.1/AV:N/AC:L/PR:N/UI:N/S:U/C:H/I:H/A:H; EPSS is 0.018% (4.86th percentile) as of disclosure, indicating low observed exploitation activity. The CVE is not currently listed in the CISA Known Exploited Vulnerabilities catalog. Primary advisory: MSRC (https://msrc.microsoft.com/update-guide/vulnerability/CVE-2026-39824) and NVD (https://nvd.nist.gov/vuln/detail/CVE-2026-39824).

Action Checklist

1. Step 1: Containment. Identify all Azure Linux 3.0 nodes running application-gateway-kubernetes-ingress 1.7.7-3. Run 'rpm -q application-gateway-kubernetes-ingress' or query your Azure Kubernetes Service (AKS) node inventory. Coordinate with application owners before restricting traffic, as this may impact service availability. Consider restricting inbound traffic to affected ingress controllers via NSG rules or Azure Firewall only if exploitation risk is deemed higher than service disruption risk, or implement a WAF rule to selectively block malformed input. Per NIST AC-3 (Access Control) and SC-7 (Boundary Protection), isolate the affected component from untrusted external input.
2. Step 2: Detection. Check Azure Monitor and container runtime logs for anomalous ingress controller process behavior: unexpected crashes, memory access violations, or out-of-bounds write errors in the AGIC pod logs (kubectl logs -n kube-system -l app=ingress-azure). Query your SIEM for abnormal HTTP request patterns targeting the ingress endpoint, specifically oversized or malformed Host or URL strings that could trigger the overflow. Enable NIST AU-2 (Audit Events) on the ingress controller namespace if not already active. Per CIS 8.2 (Collect Audit Logs), confirm audit logging is enabled across the AKS cluster. No public IOCs have been published for active exploitation as of disclosure.
3. Step 3: Eradication. Apply the updated package from the Microsoft Azure Linux 3.0 repository. The fix addresses the integer overflow in NewNTUnicodeString by correcting length boundary checks. Run 'tdnf update application-gateway-kubernetes-ingress' on affected Azure Linux 3.0 nodes, or redeploy AKS node pools using the patched image. Confirm the upstream golang.org/x/sys/windows dependency has been updated in the rebuilt package. Reference: MSRC advisory CVE-2026-39824 and the May 2026 CVRF feed (https://msrc.microsoft.com/update-guide/vulnerability/CVE-2026-39824). Apply CIS 7.3 (Perform Automated OS Patch Management) and CIS 7.4 (Perform Automated Application Patch Management).
4. Step 4: Recovery. After patching, verify the installed package version with 'rpm -q application-gateway-kubernetes-ingress' and confirm it is no longer 1.7.7-3. Validate ingress controller health via AKS pod status and Azure Application Gateway health probes. Re-enable any traffic restrictions lifted during containment only after version confirmation. Monitor ingress pod logs for 24-48 hours post-patch for residual instability. Apply NIST AU-6 (Audit Record Review, Analysis, and Reporting) to confirm no anomalous behavior occurred during the exposure window.
5. Step 5: Post-Incident. Review your Go dependency inventory across all internal applications that import golang.org/x/sys/windows; any that pass user-controlled strings to NewNTUnicodeString carry the same upstream risk (CIS 2.1, Establish and Maintain a Software Inventory). Evaluate whether your software composition analysis tooling would have flagged this upstream library before the vendor advisory. Assess whether ingress controller components are included in your vulnerability scanning scope (CIS 7.1, Establish and Maintain a Vulnerability Management Process). Document the detection gap if this component was not previously tracked in the asset inventory (CIS 1.1).

IR / Forensic Enrichment

Triage Priority	IMMEDIATE
Escalation Criteria	Escalate immediately to CISO and cloud platform owner if any AGIC pod in the affected AKS cluster shows a crash exit code of 139 (SIGSEGV), if AKS audit logs reveal unauthorized ServiceAccount creation or ClusterRoleBinding modifications in kube-system during the exposure window (indicating post-exploitation persistence), or if the organization routes PII, PHI, or payment card data through the affected ingress controller — triggering breach notification assessment under applicable data protection regulations.
Recovery Notes	After confirming the patched package version via 'rpm -q application-gateway-kubernetes-ingress' on all nodes, restore NSG and Azure Firewall rules in a staged sequence — re-enable internal load balancer access first, then external traffic — to allow validation of ingress routing health before full exposure. Monitor AGIC pod restartCount and Azure Application Gateway backend health probe status continuously for a minimum of 48 hours post-patch, as memory corruption vulnerabilities can cause delayed instability if heap state was partially corrupted prior to patching. If any pod exhibits a restart or non-zero exit code during this window, treat the node as potentially compromised and escalate to full forensic triage before returning it to production.
Forensic Artifacts	Azure Application Gateway access logs (storage account blob: 'insights-logs-applicationgatewayaccesslog') — search for requests with Host header length exceeding 32,767 bytes or URL paths containing null bytes or non-ASCII sequences designed to manipulate the uint16 length field in NewNTUnicodeString, which is the specific trigger mechanism for CVE-2026-39824. | AGIC pod crash logs captured via 'kubectl logs -n kube-system -l app=ingress-azure --previous' — a successful or attempted exploit of this integer overflow will produce a Go runtime panic stacktrace referencing 'golang.org/x/sys/windows.NewNTUnicodeString' or 'windows.UTF16PtrFromString', with a SIGSEGV or heap corruption signal that is distinct from normal operational errors. | AKS control plane audit log (kube-audit diagnostic category) — filter for CREATE and PATCH operations on ServiceAccounts, ClusterRoles, ClusterRoleBindings, and Pods in the kube-system namespace during the CVE exposure window, as RCE against the AGIC process running with kube-system privileges would most likely be leveraged to escalate within the cluster via the Kubernetes API. | Pre-patch AGIC binary file hash and embedded Go module metadata ('go version -m /usr/bin/application-gateway-ingress-controller') from affected nodes — preserves evidence of the vulnerable golang.org/x/sys/windows version string and confirms whether the binary was the unmodified vendor-distributed package or had been replaced, which would indicate tampering post-exploitation. | Azure Monitor container insights memory RSS time-series for AGIC pods (Metric: 'memoryRssBytes', namespace: 'kube-system', pod label: 'app=ingress-azure') over the 30-day exposure window — the integer overflow in NewNTUnicodeString causes undersized buffer allocation followed by out-of-bounds writes, which will manifest as anomalous memory growth spikes or sawtooth patterns immediately preceding any recorded pod crashes.

Per-Action IR Details

Step 1: Containment — Identify all Azure Linux 3.0 nodes running application-gateway-kubernetes-ingress 1.7.7-3. Run 'rpm -q application-gateway-kubernetes-ingress' or query your Azure Kubernetes Service (AKS) node inventory. Restrict inbound traffic to affected ingress controllers via NSG rules or Azure Firewall until the patch is applied, per MSRC advisory CVE-2026-39824. Apply NIST SI-3 (Malicious Code Protection) and AC-4 (Information Flow Enforcement) by isolating the affected component from untrusted external input.

NIST Phase: Containment

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

Controls: NIST AC-4 (Information Flow Enforcement), NIST SI-3 (Malicious Code Protection), NIST IR-4 (Incident Handling), CIS 4.4 (Implement and Manage a Firewall on Servers)

Compensating: For teams without Azure Firewall or centralized NSG management, use kubectl to cordon affected nodes ('kubectl cordon ') to prevent new pod scheduling, and apply a Kubernetes NetworkPolicy to the kube-system namespace that drops all ingress traffic to pods labeled 'app=ingress-azure' except from known internal load balancer CIDRs. Enumerate all affected nodes in one pass: 'for node in $(kubectl get nodes -o jsonpath="{.items[*].metadata.name}"); do kubectl debug node/$node -it --image=mcr.microsoft.com/cbl-mariner/base/core:2.0 -- rpm -q application-gateway-kubernetes-ingress; done'. Two-person team: one applies NetworkPolicy, one validates node list.

Evidence: Before isolating, capture a point-in-time snapshot of AGIC pod state: 'kubectl describe pod -n kube-system -l app=ingress-azure' (record restartCount, LastState, OOMKilled flag, and exitCode — an exitCode of 139 indicates SIGSEGV consistent with memory corruption exploitation). Export current Azure NSG flow logs from the Application Gateway subnet for the 72-hour window preceding containment. Pull Azure Monitor container insights metrics for AGIC pod memory RSS spikes. These baselines are lost after node restart or pod eviction.

Step 2: Detection — Check Azure Monitor and container runtime logs for anomalous ingress controller process behavior: unexpected crashes, memory access violations, or out-of-bounds write errors in the AGIC pod logs (kubectl logs -n kube-system -l app=ingress-azure). Query your SIEM for abnormal HTTP request patterns targeting the ingress endpoint — specifically oversized or malformed Host or URL strings that could trigger the overflow. Enable AU-2 (Event Logging) on the ingress controller namespace if not already active. Per CIS 8.2 (Collect Audit Logs), confirm audit logging is enabled across the AKS cluster. No public IOCs have been published for active exploitation as of disclosure.

NIST Phase: Detection Analysis

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

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

Compensating: Without a SIEM, use the following CLI pipeline to extract crash signals from AGIC pod logs: 'kubectl logs -n kube-system -l app=ingress-azure --previous 2>/dev/null | grep -iE "signal|fault|panic|overflow|bounds|nil pointer|runtime error"'. To detect malformed Host header exploitation attempts targeting the NewNTUnicodeString integer overflow, parse Azure Application Gateway access logs (stored in the configured storage account or Log Analytics workspace) with: 'az monitor log-analytics query --workspace --analytics-query "AzureDiagnostics | where Category == \"ApplicationGatewayAccessLog\" | where httpStatus_d >= 500 or requestUri_s contains \"%00\" or strlen(host_s) > 253"'. For teams with no Log Analytics, download raw AGIC container logs via 'kubectl cp kube-system/:/var/log ./agic-logs' and grep for Go runtime panic stacktraces referencing 'golang.org/x/sys/windows.NewNTUnicodeString'.

Evidence: The integer overflow in NewNTUnicodeString is triggered by a string whose length, when converted to a uint16 for the UNICODE_STRING Length field, wraps around to a smaller value — causing the downstream buffer allocation to be undersized. Exploitation attempts will appear in Azure Application Gateway access logs as requests with anomalously long Host headers (>32,767 bytes, the uint16 max) or URL paths with non-ASCII or null-byte sequences designed to manipulate length calculations. Capture: (1) raw Application Gateway access log entries from the storage account blob container 'insights-logs-applicationgatewayaccesslog' for the 30 days preceding detection; (2) AGIC pod crash logs via 'kubectl logs --previous' showing Go runtime panic output with a stack trace referencing 'windows.UTF16PtrFromString' or 'NewNTUnicodeString'; (3) Azure Activity Log entries for any unexpected AGIC pod restarts (OOMKilled or Error exit codes) in the AKS control plane audit log.

Step 3: Eradication — Apply the updated package from the Microsoft Azure Linux 3.0 repository. The fix addresses the integer overflow in NewNTUnicodeString by correcting length boundary checks. Run 'tdnf update application-gateway-kubernetes-ingress' on affected Azure Linux 3.0 nodes, or redeploy AKS node pools using the patched image. Confirm the upstream golang.org/x/sys/windows dependency has been

updated in the rebuilt package. Reference: MSRC advisory CVE-2026-39824 and the May 2026 CVRF feed (https://api.msrc.microsoft.com/cvrf/v3.0/cvrf/2026-May). Apply CIS 7.3 (Perform Automated OS Patch Management) and CIS 7.4 (Perform Automated Application Patch Management).

NIST Phase: Eradication

Reference: NIST 800-61r3 §3.4 — Eradication

Controls: NIST SI-2 (Flaw Remediation), NIST CM-3 (Configuration Change Control), CIS 7.3 (Perform Automated Operating System Patch Management), CIS 7.4 (Perform Automated Application Patch Management)

Compensating: For teams managing AKS node pools manually without automated patch pipelines: (1) Use 'tdnf update application-gateway-kubernetes-ingress' on each Azure Linux 3.0 node, then validate the embedded golang.org/x/sys/windows version by extracting the Go binary and running 'go version -m /usr/bin/application-gateway-ingress-controller | grep golang.org/x/sys' — the patched version must show a golang.org/x/sys release that post-dates the CVE-2026-39824 fix. (2) If node-level package update is not feasible, trigger an AKS node pool reimage via 'az aks nodepool upgrade --resource-group --cluster-name --name --node-image-only' which will pull the latest Azure Linux 3.0 node image containing the patched package. Document the pre- and post-patch RPM version ('rpm -q --queryformat "%{VERSION}-%{RELEASE}\n" application-gateway-kubernetes-ingress') as change control evidence.

Evidence: Before applying the patch, preserve a forensic image of the vulnerable AGIC binary for post-incident analysis: 'kubectl cp kube-system/:/usr/bin/application-gateway-ingress-controller ./agic-binary-1.7.7-3.bin' and record its SHA-256 hash. Also collect the full RPM package manifest ('rpm -qa --queryformat "%{NAME} %{VERSION} %{RELEASE}\n"') from each affected node as a pre-patch baseline. If any node showed crash indicators during detection, pull a Go goroutine dump if the process is still alive: 'kill -SIGABRT ' after redirecting stderr to a file — this captures the in-memory stack state referencing the vulnerable NewNTUnicodeString call path before eradication removes the evidence.

Step 4: Recovery — After patching, verify the installed package version with 'rpm -q application-gateway-kubernetes-ingress' and confirm it is no longer 1.7.7-3. Validate ingress controller health via AKS pod status and Azure Application Gateway health probes. Re-enable any traffic restrictions lifted during containment only after version confirmation. Monitor ingress pod logs for 24-48 hours post-patch for residual instability. Apply AU-6 (Audit Record Review, Analysis, and Reporting) to confirm no anomalous behavior occurred during the exposure window.

NIST Phase: Recovery

Reference: NIST 800-61r3 §3.5 — Recovery

Controls: NIST AU-6 (Audit Record Review, Analysis, and Reporting), NIST CP-10 (System Recovery and Reconstitution), NIST CM-6 (Configuration Settings), CIS 4.2 (Establish and Maintain a Secure Configuration Process for Network Infrastructure)

Compensating: Automate post-patch version verification across all AKS nodes with: 'kubectl get nodes -o name | sed "s|node/||" | xargs -I{} kubectl debug node/{} -it --image=mcr.microsoft.com/cbl-mariner/base/core:2.0 -- rpm -q application-gateway-kubernetes-ingress 2>/dev/null'. To validate that the golang.org/x/sys/windows dependency is also at the fixed version (not just the RPM wrapper), run 'go version -m | grep golang.org/x/sys' on a representative patched node. For the 24-48 hour monitoring window, set up a lightweight watch without SIEM: 'watch -n 60 kubectl get pods -n kube-system -l app=ingress-azure' on a terminal pane, and tail AGIC logs to a file ('kubectl logs -n kube-system -l app=ingress-azure -f >> /tmp/agic-recovery-watch.log 2>&1 &') for offline review.

Evidence: During the recovery monitoring window, capture Application Gateway backend health probe results via 'az network application-gateway show-backend-health --resource-group --name ' at 15-minute intervals — degraded backend health immediately post-patch could indicate residual memory corruption from a pre-patch exploit that persisted across the pod restart. Collect AKS audit logs (enabled via Diagnostic Settings → kube-audit) for any API server calls that created new pods, ServiceAccounts, or ClusterRoleBindings in the kube-system namespace during the exposure window, as successful RCE via this vulnerability could have been used to establish persistence within the cluster before patching.

Step 5: Post-Incident — Review your Go dependency inventory across all internal applications that import golang.org/x/sys/windows — any that pass user-controlled strings to NewNTUnicodeString carry the same upstream risk (CIS 2.1, Establish and Maintain a Software Inventory). Evaluate whether your software composition analysis tooling would have flagged this upstream library before the vendor advisory. Assess whether ingress controller components are included in your vulnerability scanning scope (CIS 7.1, Establish and Maintain a Vulnerability Management Process). Document the detection gap if this component was not previously tracked in the asset inventory (CIS 1.1).

NIST Phase: Post Incident

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

Controls: NIST RA-3 (Risk Assessment), NIST SA-15 (Development Process, Standards, and Tools), CIS 1.1 (Establish and Maintain Detailed Enterprise Asset Inventory), CIS 2.1 (Establish and Maintain a Software Inventory), CIS 7.1 (Establish and Maintain a Vulnerability Management Process), CIS 7.2 (Establish and Maintain a Remediation Process)

Compensating: To enumerate all internal Go binaries and container images that link golang.org/x/sys/windows without a commercial SCA tool: (1) Scan your container registry images with Trivy (free, OSS): 'trivy image --vuln-type library --severity CRITICAL | grep golang.org/x/sys' — Trivy parses Go module metadata embedded in binaries. (2) For source repositories, run 'grep -r "golang.org/x/sys" --include="go.mod" /path/to/repos' to find any go.mod files declaring this dependency, then check each for a 'replace' directive or pinned version that predates the fix. (3) Use 'go list -m -json golang.org/x/sys' inside each module to confirm the resolved version. Document findings in a simple CSV (binary/image name, golang.org/x/sys version, calls NewNTUnicodeString: yes/no) as the lessons-learned artifact.

Evidence: For the lessons-learned record, gather: (1) the timestamp delta between the May 2026 MSRC Patch Tuesday publication and your team's first internal awareness of CVE-2026-39824 — this measures detection lead time for supply chain components; (2) the asset inventory record (or absence thereof) for the application-gateway-kubernetes-ingress package and its node pool assignment prior to this incident, to document the inventory gap; (3) output of 'az security assessment list --subscription ' filtered for container registry or AKS assessments to determine whether Microsoft Defender for Cloud had flagged the vulnerable package version before manual discovery.

Detection Guidance

No public IOCs or active exploit code have been reported as of the May 2026 Patch Tuesday disclosure. Detection focuses on behavioral indicators and configuration state. (1) Asset identification: query your AKS cluster or CMDB for nodes running application-gateway-kubernetes-ingress 1.7.7-3 on Azure Linux 3.0. (2) Crash indicators: search ingress controller pod logs (kubectl logs) for segmentation faults, memory access violations, or unexpected process restarts in the kube-system or ingress namespace; these may indicate failed exploitation attempts. (3) Anomalous input: inspect Azure Application Gateway access logs for requests with abnormally long or malformed Host headers, URL paths, or query strings that could be probing the overflow boundary. (4) Network anomalies: look for unexpected outbound connections from ingress controller pods post-crash, which could indicate successful code execution. (5) Patch state verification: if your SIEM or vulnerability management platform ingests package inventory data, create a detection rule for application-gateway-kubernetes-ingress == 1.7.7-3 on Azure Linux 3.0 as an unpatched asset alert. Per NIST AU-6, maintain audit review cadence on ingress-layer logs until patching is confirmed complete.

Framework Mappings

MITRE-ATTACK

- **T1211** — Exploitation for Defense Evasion
- **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-16** — Memory Protection

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.8.8** — Management of technical vulnerabilities
- **A.5.23** — Information security for use of cloud services

MITRE ATT&CK Mapping

Technique ID	Technique Name	Tactic
T1211	Exploitation for Defense Evasion	Defense-Evasion
T1190	Exploit Public-Facing Application	Initial-Access

Sources

Source	URL	Tier
MSRC Update Guide	https://msrc.microsoft.com/update-guide/vulnerability/CVE-2026-39824	T1
(consolidated)	https://api.msrc.microsoft.com/cvrf/v3.0/cvrf/2026-May	T1
CVE-2026-39824 Detail - NVD	https://nvd.nist.gov/vuln/detail/CVE-2026-39824	T1
CVE-2026-39824: string length overflow vulnerability in sys (Go)	https://www.resolvedsecurity.com/vulnerability-catalog/CVE-2026-39824	T3

Source	URL	Tier
CVE-2026-39824 - Invoking integer overflow in ... - CVEFeed.io	https://cvefeed.io/vuln/detail/CVE-2026-39824	**T3**

DISCLAIMER

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

Generated 2026-05-27 06:38 UTC by TJS Security Command Center