INTELLIGENCE BRIEFING
Security Command Center
TLP:CLEAR
2026-06-06 14:03 UTC

# Miasma Worm Compromises 73 Microsoft GitHub Repositories via Stolen Publisher Credentials, Targets AI Developer Toolchains

**THREAT CAMPAIGN** | **CRITICAL** | CVSS 9.5

| | |
|---|---|
| **SCC Item ID** | SCC-CAM-2026-0419 |
| **Type** | Threat Campaign |
| **Severity** | CRITICAL |
| **CVSS Base Score** | 9.5 |
| **Affected Products** | Microsoft GitHub repositories (Azure, Azure-Samples, Microsoft, MicrosoftDocs orgs); Azure Durable Task ecosystem (durabletask, durabletask-dotnet, durabletask-go, durabletask-js, durabletask-mssql); durabletask PyPI package; icflorescu/mantine-datatable and related npm packages; AI coding environments including Claude Code, Gemini CLI, Cursor, VS Code |
| **Published** | 2026-06-06T02:58:04 |
| **Discovery Source** | Rss |

## Executive Summary

A self-replicating worm designated Miasma has compromised 73 Microsoft GitHub repositories across the Azure, Azure-Samples, Microsoft, and MicrosoftDocs organizations by exploiting stolen publisher credentials. The attack vector exploits stolen publisher credentials, meaning malicious commits pass standard integrity checks and appear to originate from authorized maintainer accounts. The worm targets the Azure Durable Task ecosystem and propagates downstream through PyPI and npm package registries, meaning any organization consuming these packages in production pipelines may have ingested compromised code. A secondary payload component specifically targets AI-assisted coding environments (Claude Code, Gemini CLI, Cursor, VS Code), meaning developers who clone affected repositories in these environments may trigger autonomous payload execution on their local workstations, extending the blast radius into enterprise development environments.

## Technical Analysis

Miasma is a mutating variant of the Mini Shai-Hulud worm. A threat actor group designated TeamPCP has been associated with similar activity (first observed mid-May 2026), though this attribution is not yet corroborated by a primary source and should be treated as medium-confidence intelligence pending further investigation. The attack vector is credential theft against repository publishers (CWE-798: Use of Hard-coded Credentials / stolen

credentials; CWE-284: Improper Access Control), not a platform-level GitHub vulnerability. Malicious commits pass standard integrity checks because they originate from authenticated, authorized publisher accounts (MITRE T1078: Valid Accounts; T1554: Compromise Software Supply Chain; T1195.001: Compromise Software Dependencies and Development Tools). Affected repositories span the Azure Durable Task ecosystem: durabletask, durabletask-dotnet, durabletask-go, durabletask-js, durabletask-mssql, and the durabletask PyPI package, as well as icflorescu/mantine-datatable and related npm packages. The worm self-replicates across repositories (T1608.003: Stage Capabilities, Install Digital Certificate; T1195.002: Compromise Software Supply Chain). A secondary payload targets AI coding agents: opening a compromised repository in Claude Code, Gemini CLI, Cursor, or VS Code may trigger autonomous payload execution within the agent's context (T1176: Browser Extensions analog for agent plugins; T1072: Software Deployment Tools). CWE-829 (Inclusion of Functionality from Untrusted Control Sphere) and CWE-494 (Download of Code Without Integrity Check) describe the downstream consumer exposure. No CVE ID has been assigned. No CISA KEV entry exists as of the data submission date. GitHub has disabled access to all 73 confirmed affected repositories. Source quality score is 0.712 (T3 press reporting with one T1 Microsoft Learn URL); attribution to TeamPCP is not yet corroborated by a primary source, treat as medium-confidence intelligence.

## Action Checklist

1. Step 1: Containment. Audit your dependency manifests (requirements.txt, package.json, go.mod, .csproj) for any references to durabletask (PyPI), durabletask-dotnet, durabletask-go, durabletask-js, durabletask-mssql (npm/NuGet), or mantine-datatable and its related npm packages. Pin all affected dependencies to the last known-clean commit hash or version predating mid-May 2026 (NIST SI-7: Software, Firmware, and Information Integrity). Restrict or disable pipeline access to the affected GitHub repository URLs pending GitHub's confirmation of repository integrity. Do not merge or deploy any code pulled from the 73 disabled repositories during the compromise window.
2. Step 2: Detection. Query CI/CD pipeline logs for clone, pull, or install events referencing the affected repositories (Azure/durabletask, microsoft/durabletask-mssql, MicrosoftDocs/azure-docs and siblings) between mid-May 2026 and present. In SIEM, search for process execution events spawned by Claude Code, Gemini CLI, Cursor, or VS Code processes that follow a repository clone event; unexpected child processes, network connections, or file writes from agent processes are behavioral indicators of secondary payload execution. Review NIST AU-2 (Event Logging) and NIST AU-6 (Audit Record Review, Analysis, and Reporting) coverage across developer workstations and build agents. Cross-reference CIS Controls v8 2.3 (Address Unauthorized Software) compliance on those endpoints.
3. Step 3: Eradication. Rotate all publisher credentials (tokens, SSH keys, API keys) for any account that had write access to the affected repositories (NIST IA-4: Identifier Management; NIST IA-5: Authentication). Revoke and reissue any tokens stored in CI/CD secrets managers that had access to these repositories (NIST AC-2: Account Management; NIST AC-6: Least Privilege). Remove and reinstall all affected packages from clean, verified sources; do not restore from cached or locally stored copies pulled during the compromise window. Enforce NIST AC-2 (Account Management) to restrict which pipeline service accounts can pull from external registries.
4. Step 4: Recovery. After reinstalling from clean sources, verify package integrity against published checksums or digests from the registry maintainer before reintroducing into build pipelines. Enable NIST SI-7 (Software, Firmware, and Information Integrity) on build agent filesystems to detect and prevent installation of unsigned or tampered artifacts. Monitor NIST AU-6 (Audit Record Review, Analysis, and Reporting) for anomalous process behavior on developer workstations for a minimum of 30 days

post-remediation, with particular attention to processes spawned by AI coding agent tools. Validate that Azure Durable Functions deployments in production are running confirmed-clean SDK versions.

5. Step 5: Post-Incident. This campaign exposed a critical control gap: publisher credential compromise bypasses code-signing and commit verification controls entirely. Implement NIST AC-5 (Separation of Duties) for repository publish rights; no single account should hold both write and publish permissions. Enforce multi-factor authentication (MFA) on all GitHub publisher accounts without exception (NIST IA-2: Authentication; CIS Controls v8 6.2: Use Multi-Factor Authentication). Establish a software composition analysis (SCA) gate in CI/CD that verifies commit provenance and publisher identity (NIST IA-8: Identification and Authentication for Organizational Users). Review AI coding agent permissions and apply least-privilege principles (NIST AC-6) to agent tool access, restricting agent file-system and network permissions to project scope only.

## IR / Forensic Enrichment

| | |
|---|---|
| **Triage Priority** | IMMEDIATE |
| **Escalation Criteria** | Escalate to senior leadership, legal counsel, and potentially regulatory bodies immediately if forensic analysis confirms that compromised durabletask SDK versions were executed within production Azure Durable Functions workloads, particularly any handling PII, PHI, or financial data, as active payload execution in production constitutes a breach event triggering notification obligations under GDPR, HIPAA, and state breach notification laws. |
| **Recovery Notes** | Do not return any build agent or developer workstation to production status without a clean-state reimaging or a verified filesystem scan confirming no Miasma worm artifacts remain — restored cached packages are not sufficient, as the worm's self-replication design means malicious code may persist in locations outside the package install path. Monitor all Azure Durable Functions orchestration workflows in production for anomalous task scheduling, unexpected external HTTP calls from orchestrator code, or unusually elevated execution counts for a minimum of 30 days post-remediation, as the worm may have injected persistent orchestration logic into deployed function state stores. Verify that all GitHub publisher accounts have MFA enforced and PAT scopes are re-evaluated before any new commits are pushed to packages published to PyPI or npm, as the stolen-credential attack vector remains viable until access hygiene is fully restored. |

| | |
|---|---|
| **Forensic Artifacts** | GitHub organization audit log (JSON export via REST API for Azure, Azure-Samples, Microsoft, MicrosoftDocs orgs): records every git.push, oauth_access.create, and repo.add_member event associated with the stolen publisher credentials during the compromise window — the primary artifact for establishing credential misuse timeline and account scope. \| CI/CD pipeline build logs with dependency resolution steps (GitHub Actions .log files, Jenkins build console output, Azure DevOps timeline records): contain the exact resolved commit SHA and download URL for every durabletask and mantine-datatable package pulled during the exposure window, establishing whether your pipeline consumed a malicious version. \| Sysmon Event ID 1 (Process Create) and Event ID 3 (Network Connection) logs from developer workstations: capture process lineage from Claude Code, Gemini CLI, Cursor, or VS Code parent processes to any unexpected child processes spawned post-repository-clone, which is the behavioral fingerprint of secondary payload detonation by the AI coding agent attack vector. \| Package cache directories with original file timestamps preserved (/root/.cache/pip, %LOCALAPPDATA%\NuGet\v3-cache, node_modules/.cache on build agents): contain the original malicious package artifacts as installed during the compromise window and are the primary source for malware analysis and YARA rule development against the Miasma worm payload. \| Azure Monitor Log Analytics tables AppDependencies and AppRequests for production Durable Functions workloads: reveal whether compromised SDK versions made outbound calls to attacker-controlled infrastructure during orchestration execution, establishing whether the worm's payload detonated in production and what data or credentials it may have accessed. |

### Per-Action IR Details

**Step 1: Containment — Audit your dependency manifests (requirements.txt, package.json, go.mod, .csproj) for any references to durabletask (PyPI), durabletask-dotnet, durabletask-go, durabletask-js, durabletask-mssql (npm/NuGet), or mantine-datatable and its related npm packages. Immediately pin all affected dependencies to the last known-clean commit hash or version predating mid-May 2026, and block pipeline pulls from the affected GitHub repository URLs until GitHub confirms repository integrity. Do not merge or deploy any code pulled from the 73 disabled repositories.**

**NIST Phase:** Containment

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

**Controls:** NIST AC-4 (Information Flow Enforcement), CIS 2.2 (Ensure Authorized Software is Currently Supported), CIS 7.1 (Establish and Maintain a Vulnerability Management Process)

**Compensating:** Run `pip show durabletask`, `npm list durabletask-js mantine-datatable`, and `grep -r 'durabletask' go.mod *.csproj requirements.txt package.json` across all developer workstations and build agents. Add firewall rules or /etc/hosts null-routing for github.com/Azure/durabletask, github.com/microsoft/durabletask-mssql, and github.com/MicrosoftDocs/azure-docs until integrity is confirmed. Use `pip hash` or `npm pack --dry-run` to compare local cache hashes against pre-compromise registry digests for any packages installed between mid-May 2026 and present.

**Evidence:** Before pinning or removing packages, snapshot the following: (1) the exact installed version and install timestamp from `pip show durabletask` or `npm list --depth=0` output; (2) the resolved commit SHA recorded in package-lock.json, go.sum, or packages.lock.json for each affected package; (3) CI/CD artifact cache directories (e.g., ~/.cache/pip, node_modules/.cache, %LOCALAPPDATA%\NuGet\v3-cache) — these may contain the malicious build artifact and are primary forensic evidence of the exposure window. Do NOT purge caches before imaging.

**Step 2: Detection — Query CI/CD pipeline logs for clone, pull, or install events referencing the affected repositories (Azure/durabletask, microsoft/durabletask-mssql, MicrosoftDocs/azure-docs and siblings) between mid-May 2026 and present. In SIEM, search for process execution events spawned by Claude Code, Gemini CLI, Cursor, or VS Code processes that follow a repository clone event — unexpected child**

**processes, network connections, or file writes from agent processes are behavioral indicators of secondary payload detonation. Review AU-2 (Event Logging) and AU-6 (Audit Record Review, Analysis, and Reporting) coverage across developer workstations and build agents. Cross-reference CIS 8.2 (Collect Audit Logs) compliance on those endpoints.**

**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), CIS 8.2 (Collect Audit Logs)

**Compensating:** Deploy Sysmon (config targeting Event ID 1 — Process Create, Event ID 3 — Network Connection, Event ID 11 — FileCreate) on developer workstations; filter on parent processes matching `claude`, `gemini`, `cursor`, or `code` (VS Code) spawning unexpected children such as `cmd.exe`, `powershell.exe`, `sh`, `python`, or `node` outside the project working directory. Query GitHub audit log export (available free via GitHub REST API: `GET /orgs/{org}/audit-log`) for `git.clone` and `git.push` events referencing the 73 affected repositories from your org's runner IPs between mid-May 2026 and present. On Linux build agents, run `ausearch -c 'git' --start 05/01/2026` to extract git operations from auditd records.

**Evidence:** Capture before concluding detection scope: (1) Sysmon Event ID 1 logs from developer workstations showing process lineage from VS Code, Cursor, Claude Code, or Gemini CLI processes — specifically any child process spawned within 60 seconds of a `git clone` or `npm install` targeting affected repos; (2) GitHub Actions workflow run logs (.log files retained in the Actions tab) for any job that references `uses: Azure/durabletask*` or installs affected PyPI/npm packages — these logs record exact package resolution steps; (3) Network egress logs (firewall or proxy) for outbound connections from build agents to non-GitHub, non-PyPI, non-npm endpoints occurring immediately after install steps — the worm's self-replication mechanism may beacon to attacker infrastructure; (4) VS Code extension host logs at `%APPDATA%\Code\logs` (Windows) or `~/.config/Code/logs` (Linux) for any unexpected extension activation or file writes following repository operations.

**Step 3: Eradication — Rotate all publisher credentials (tokens, SSH keys, API keys) for any account that had write access to the affected repositories, consistent with D3-CRO (Credential Rotation) and D3-CH (Credential Hardening). Revoke and reissue any tokens stored in CI/CD secrets managers that had access to these repositories (NIST AC-2: Account Management; NIST AC-6: Least Privilege). Remove and reinstall all affected packages from clean, verified sources — do not restore from cached or locally stored copies pulled during the compromise window. Enforce D3-UAP (User Account Permissions) to restrict which pipeline service accounts can pull from external registries.**

**NIST Phase:** Eradication

**Reference:** NIST 800-61r3 §3.4 — Eradication

**Controls:** NIST AC-2 (Account Management), NIST AC-6 (Least Privilege), CIS 5.1 (Establish and Maintain an Inventory of Accounts), CIS 6.2 (Establish an Access Revoking Process)

**Compensating:** Use the GitHub REST API (`GET /repos/{owner}/{repo}/collaborators` and `GET /orgs/{org}/members`) to enumerate all accounts with write or admin access to the 73 affected repositories; cross-reference against your account inventory to identify any accounts not recognized or provisioned through your standard access-granting process. Revoke GitHub Personal Access Tokens (PATs) via `gh auth token --revoke` or the GitHub Settings > Developer Settings UI for all affected publisher accounts. For CI/CD secrets, use `printenv | grep -i token` on build agents to surface any in-memory token references that may survive a secrets manager rotation; purge and re-provision. Run `ssh-keygen -lf ~/.ssh/known_hosts` to audit SSH key fingerprints on build agents and remove any keys associated with accounts that touched the affected repositories.

**Evidence:** Before rotating credentials, preserve: (1) GitHub audit log entries for all `git.push`, `repo.add_member`, and `oauth_access.create` events associated with publisher accounts active on the 73 repositories — this establishes the timeline of credential misuse and scope of the stolen publisher credential abuse; (2) CI/CD secrets manager audit trail (e.g., GitHub Actions secrets access log, HashiCorp Vault audit log) showing which pipelines consumed tokens that had write access to affected repos — this determines secondary exposure surface; (3) SSH `authorized_keys` files and `~/.ssh/id_*` key pairs on build agent hosts — the Miasma worm may have harvested or implanted SSH keys to

sustain access or facilitate lateral movement to other repositories.

**Step 4: Recovery — After reinstalling from clean sources, verify package integrity against published checksums or digests from the registry maintainer before reintroducing into build pipelines. Enable D3-SFA (System File Analysis) and D3-FMBV (File Magic Byte Verification) on build agent filesystems to detect residual malicious artifacts. Monitor AU-6 (Audit Record Review, Analysis, and Reporting) for anomalous process behavior on developer workstations for a minimum of 30 days post-remediation, with particular attention to AI coding agent processes. Validate that Azure Durable Functions deployments in production are running confirmed-clean SDK versions.**

**NIST Phase:** Recovery

**Reference:** NIST 800-61r3 §3.5 — Recovery

**Controls:** NIST AU-6 (Audit Record Review, Analysis, And Reporting), CIS 7.3 (Perform Automated Operating System Patch Management), CIS 7.4 (Perform Automated Application Patch Management)

**Compensating:** Verify PyPI package integrity with `pip download durabletask --no-deps -d /tmp/verify && pip hash /tmp/verify/durabletask-*.whl` and compare SHA-256 output against the digest published on pypi.org for the specific clean version. For npm packages, run `npm audit signatures` (available in npm CLI v8.15+, free) to verify registry-published package signatures. Write a YARA rule targeting the malicious worm payload characteristics (self-replication logic referencing GitHub API endpoints and credential stores) and scan build agent filesystems with `yara -r worm_rule.yar /home /opt /var/lib/jenkins` before returning agents to service. For Azure Durable Functions production deployments, query the Azure Resource Graph (`az graph query -q "Resources | where type == 'microsoft.web/sites"'`) to enumerate all Function Apps, then verify the durabletask SDK version in each deployment package against the confirmed-clean version hash.

**Evidence:** Before restoring build agents to production: (1) Capture a full filesystem listing (`find / -newer /tmp/compromise_start_marker -ls 2>/dev/null`) of build agent hosts using the mid-May 2026 compromise window start as the reference timestamp — any files created or modified by the worm during the exposure window will appear; (2) Collect Azure Durable Functions application logs from Azure Monitor (Log Analytics workspace, table `AppDependencies` and `AppRequests`) for the production environment to identify any orchestration workflows that invoked a compromised SDK version — these may represent already-detonated payloads in production workloads; (3) Record the exact package digests (SHA-256) of every clean replacement package installed, to serve as the integrity baseline for the 30-day monitoring period.

**Step 5: Post-Incident — This campaign exposed a specific control gap: publisher credential compromise bypasses code-signing and commit verification controls entirely. Implement NIST AC-5 (Separation of Duties) for repository publish rights — no single account should hold both write and publish permissions. Enforce CIS 6.3 (Require MFA for Externally-Exposed Applications) and CIS 6.5 (Require MFA for Administrative Access) on all GitHub publisher accounts without exception. Establish a software composition analysis (SCA) gate in CI/CD that verifies commit provenance, not just package version. Review AI coding agent permissions — apply least-privilege principles (NIST AC-6) to agent tool access, and restrict agent file-system and network permissions to project scope only.**

**NIST Phase:** Post Incident

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

**Controls:** NIST AC-5 (Separation Of Duties), NIST AC-6 (Least Privilege), CIS 6.3 (Require MFA for Externally-Exposed Applications), CIS 6.5 (Require MFA for Administrative Access), CIS 7.2 (Establish and Maintain a Remediation Process)

**Compensating:** Implement GitHub branch protection rules (free on all GitHub tiers) requiring at minimum two reviewer approvals and requiring signed commits (`git config --global commit.gpgsign true` with GPG or SSH signing) for any push to the default branch of repositories holding published package source — this ensures stolen credentials alone cannot produce a trusted commit. Use the free `deps.dev` API or `pip-audit` / `npm audit` CLI tools as a lightweight SCA gate in CI/CD pipelines by adding a pre-build step that queries known-compromised package versions. For AI coding agent sandboxing, configure VS Code's `settings.json` to restrict Claude Code and Cursor extension filesystem

access using the workspace trust feature (`"security.workspace.trust.enabled": true`) and block outbound network from agent processes using a host-based firewall rule (`ufw deny out from any to any app GitHub` on Ubuntu build agents, scoped to the agent service account UID).

**Evidence:** For lessons-learned documentation, preserve: (1) The complete GitHub audit log export covering the full compromise window for all four affected organizations (Azure, Azure-Samples, Microsoft, MicrosoftDocs) — this is the authoritative record of how stolen publisher credentials were used and which accounts were abused, and it supports any regulatory notification obligations; (2) A diff of every malicious commit made to the 73 repositories (retrievable via `git log --all --oneline` on a pre-disable mirror) documenting exactly what code the Miasma worm injected — this serves as the threat intelligence input for future YARA/Sigma detection rules; (3) Incident timeline records mapping the first malicious commit timestamp to the first pipeline pull in your environment, establishing dwell time for the post-incident report and any required breach notification window calculations.

## Detection Guidance

Primary detection surface is CI/CD pipeline and developer workstation logs. Query build system logs for git clone, pip install, npm install, or NuGet restore operations referencing the affected repositories or package names (durabletask PyPI; durabletask-dotnet, durabletask-go, durabletask-js, durabletask-mssql on npm/NuGet; mantine-datatable npm) with timestamps between mid-May 2026 and the date GitHub disabled repository access. In endpoint telemetry, alert on child processes spawned by AI agent processes (claude, gemini-cli, cursor, code) that execute shell commands, write files outside the project directory, or initiate outbound network connections; these are behavioral indicators consistent with secondary payload execution described in the campaign. For SIEM correlation: chain a repository clone event to an anomalous agent child process within a 5-minute window as a high-fidelity detection rule. On the identity side, review GitHub audit logs (if available via GitHub Enterprise) for unexpected publish events or credential use from unusual source IPs against the affected organizations (Azure, Azure-Samples, Microsoft, MicrosoftDocs) during the compromise window. NIST AU-2 (Event Logging) and NIST SI-7 (Software, Firmware, and Information Integrity) apply to developer workstations where affected repositories were opened. Note: The available source data does not include confirmed indicators of compromise (hashes, IPs, domains, file names). If additional IOC data becomes available from Microsoft security advisories, CISA alerts, or threat intelligence platforms, incorporate those into detection rules. Absence of IOCs in this analysis should not be interpreted as absence of detectable artifacts in your environment.

## Indicators of Compromise

| Type | Value | Context | Confidence |
|---|---|---|---|
| URL | `https://github.com/Azure/durabletask` | Affected GitHub repository — one of 73 disabled by GitHub; do not pull from this URL until integrity is confirmed restored | HIGH |
| URL | `https://github.com/microsoft/durabletask-mssql` | Affected GitHub repository — Microsoft SQL storage provider for Durable Functions, listed as compromised | HIGH |

| Type | Value | Context | Confidence |
|---|---|---|---|
| URL | `https://github.com/MicrosoftDocs/azure-docs/blob/main/articles/durable-task/sdks/durable-task-overview.md` | Affected MicrosoftDocs repository reference — treat any content pulled from this path during the compromise window as untrusted | MEDIUM |

## Framework Mappings

### MITRE-ATTACK

- **T1059** — Command and Scripting Interpreter
- **T1543** — Create or Modify System Process
- **T1553.002** — Code Signing
- **T1608.003** — Install Digital Certificate
- **T1195.001** — Compromise Software Dependencies and Development Tools
- **T1176** — Software Extensions
- **T1072** — Software Deployment Tools
- **T1195.002** — Compromise Software Supply Chain
- **T1554** — Compromise Host Software Binary
- **T1078** — Valid Accounts

### NIST-800-53R5

- **CM-7** — Least Functionality
- **SI-3** — Malicious Code Protection
- **SI-4** — System Monitoring
- **SI-7** — Software, Firmware, and Information Integrity
- **SA-9** — External System Services
- **SR-3** — Supply Chain Controls and Processes
- **AC-2** — Account Management
- **AC-6** — Least Privilege
- **IA-2** — Identification and Authentication (Organizational Users)
- **IA-5** — Authenticator Management
- **AC-3** — Access Enforcement
- **CM-3** — Configuration Change Control
- **SR-2** — Supply Chain Risk Management Plan

### OWASP-TOP10-2021

- **A01:2021** — Broken Access Control
- **A07:2021** — Identification and Authentication Failures
- **A08:2021** — Software and Data Integrity Failures

### CIS-V8

- **6.1** — Establish an Access Granting Process
- **6.2** — Establish an Access Revoking Process
- **16.10** — Apply Secure Design Principles in Application Architectures
- **2.5** — Allowlist Authorized Software
- **2.6** — Allowlist Authorized Libraries
- **6.3** — Require MFA for Externally-Exposed Applications
- **15.1** — Establish and Maintain an Inventory of Service Providers

### 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(a)(1)** — Access Control
- **164.312(d)** — Person or Entity Authentication

### ISO-27001-2022

- **A.8.28** — Secure coding
- **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

## MITRE ATT&CK Mapping

| Technique ID | Technique Name | Tactic |
|---|---|---|
| T1059 | Command and Scripting Interpreter | Execution |
| T1543 | Create or Modify System Process | Persistence |
| T1553.002 | Code Signing | Defense-Evasion |
| T1608.003 | Install Digital Certificate | Resource-Development |
| T1195.001 | Compromise Software Dependencies and Development Tools | Initial-Access |
| T1176 | Software Extensions | Persistence |
| T1072 | Software Deployment Tools | Execution |
| T1195.002 | Compromise Software Supply Chain | Initial-Access |

| Technique ID | Technique Name | Tactic |
|---|---|---|
| T1554 | Compromise Host Software Binary | Persistence |
| T1078 | Valid Accounts | Defense-Evasion |

## Sources

| Source | URL | Tier |
|---|---|---|
| **Security News** | https://thehackernews.com/2026/06/miasma-worm-hits-73-microsoft-git... | T3 |
| **GitHub - Azure/durabletask: Durable Task Framework allows users ...** | https://github.com/Azure/durabletask | T3 |
| **Durable Functions Packages, Extensions, and SDKs: Overview** | https://learn.microsoft.com/en-us/azure/azure-functions/durable-fun... | T1 |
| **Microsoft SQL storage provider for Durable Functions and ... - GitHub** | https://github.com/microsoft/durabletask-mssql | T3 |
| **azure-docs/articles/durable-task/sdks/durable-task-overview.md at ...** | https://github.com/MicrosoftDocs/azure-docs/blob/main/articles/dura... | T3 |



Generated 2026-06-06 14:03 UTC by TJS Security Command Center