INTELLIGENCE BRIEFING
Security Command Center
TLP:CLEAR
2026-05-20 06:43 UTC

TeamPCP Breaches GitHub via Malicious VS Code Extension; Mini Shai-Hulud Worm Compromises durabletask PyPI Package

THREAT CAMPAIGN | **CRITICAL** | CVSS 9.5

SCC Item ID	SCC-CAM-2026-0341
Type	Threat Campaign
Severity	CRITICAL
CVSS Base Score	9.5
Affected Products	GitHub internal repositories (Microsoft); durabletask PyPI package versions 1.4.1-1.4.3 (Microsoft/Dapr, ~417,000 downloads/month); Nx Console VS Code extension; guardrails-ai PyPI package; AWS EC2/SSM, Kubernetes, HashiCorp Vault, 1Password, Bitwarden, Docker, SSH environments
Published	2026-05-20T00:01:15
Discovery Source	Rss

Executive Summary

Threat actor TeamPCP breached a GitHub employee device via a trojanized VS Code extension, exfiltrating approximately 3,800 internal Microsoft/GitHub repositories and forcing emergency rotation of critical secrets across Microsoft infrastructure. Simultaneously, TeamPCP deployed a self-replicating worm called Mini Shai-Hulud inside Microsoft-affiliated durabletask PyPI package versions 1.4.1 through 1.4.3, which has approximately 417,000 downloads per month and is used across Dapr and Azure-connected Python workloads. Organizations building on GitHub or running Python environments with these packages face cascading risk: stolen source code, harvested cloud credentials, and active lateral movement across AWS, Kubernetes, HashiCorp Vault, and secrets management platforms.

Technical Analysis

TeamPCP executed a dual-vector attack. In the platform breach vector, a trojanized VS Code extension compromised a GitHub employee device (MITRE T1176, T1554), enabling exfiltration of approximately 3,800 internal repositories and triggering emergency secret rotation across Microsoft infrastructure. The Nx Console VS Code extension was also implicated. In the supply chain vector, TeamPCP injected malicious code into durabletask PyPI package versions 1.4.1, 1.4.2, and 1.4.3 and the guardrails-ai PyPI package. The embedded Mini Shai-Hulud worm (T1195.001) harvests credentials from AWS EC2 instance metadata service (IMDS)

(T1552.005), Kubernetes service account tokens, HashiCorp Vault tokens, 1Password, Bitwarden vault data, SSH key material (T1552.004), and major cloud provider credential stores (T1552, T1552.001). Propagation logic targets AWS EC2 and Kubernetes environments (T1021, T1021.001, T1021.004) with exfiltration over HTTP/S (T1071.001, T1567, T1567.001). Persistence mechanisms (T1543) and valid account abuse (T1078, T1078.004) facilitate lateral movement. Data destruction capability (T1485) is also indicated. Relevant CWEs: CWE-732 (incorrect permission assignment), CWE-522 (insufficiently protected credentials), CWE-312 (cleartext storage of sensitive information), CWE-494 (download of code without integrity check). No CVE has been assigned. Affected packages have been removed from PyPI; safe versions are 1.4.0 and below or 1.4.4 and above if released.

Action Checklist

1. Step 1: Containment, Immediately audit all Python environments for durabletask versions 1.4.1, 1.4.2, and 1.4.3 and guardrails-ai packages; isolate any host where these versions are installed from network access. Suspend GitHub Actions workflows and CI/CD pipelines that pull these packages until clean versions are confirmed. Revoke and rotate all cloud credentials accessible from affected hosts, including AWS IAM roles associated with EC2 instance metadata, Kubernetes service account tokens, HashiCorp Vault tokens, 1Password and Bitwarden vault credentials, and SSH keys present on affected systems.
2. Step 2: Detection, Query package manifests, requirements.txt, pyproject.toml, and lock files across all environments for durabletask==1.4.1, 1.4.2, or 1.4.3 and any pinned guardrails-ai version deployed in the same timeframe. In AWS CloudTrail, look for unusual IMDS calls (169.254.169.254) from EC2 instances running Python workloads, unexpected AssumeRole events, and data exfiltration to unknown endpoints. In Kubernetes audit logs, look for service account token reads outside normal application behavior. Review VS Code extension inventories for Nx Console versions installed via unofficial sources. Check DNS and HTTP proxy logs for C2 beacon patterns consistent with T1071.001 outbound from developer or CI/CD hosts.
3. Step 3: Eradication, Downgrade or remove durabletask to version 1.4.0 or upgrade to a verified clean release if available from the official Dapr GitHub repository (https://github.com/dapr/durabletask-python). Remove any installed guardrails-ai versions flagged in vendor advisories. Uninstall Nx Console extensions sourced outside the official VS Code Marketplace and re-install from the verified publisher. Re-image any host confirmed to have executed malicious package code. Rotate all credentials identified in Step 1 containment, even if compromise is unconfirmed on that host.
4. Step 4: Recovery, After credential rotation, verify new credentials are functional and access policies are scoped to least privilege. Confirm no unauthorized IAM users, roles, or Kubernetes cluster role bindings were created during the exposure window. Monitor AWS CloudTrail, Kubernetes audit logs, and Vault audit logs for continued anomalous access patterns for a minimum of 30 days. Validate that CI/CD pipelines are pulling from verified package sources with hash-pinned dependencies before restoring automated build workflows.
5. Step 5: Post-Incident, This campaign exposed three control gaps: absence of package integrity verification (CWE-494) in CI/CD pipelines, insufficient secrets isolation on developer endpoints (CWE-522), and lack of VS Code extension allowlisting (T1176). Implement hash-pinned dependencies with automated integrity checks (e.g., pip-audit, Dependabot, or Endor Labs) for all PyPI packages. Enforce developer endpoint controls that restrict cloud credential access to dedicated CI/CD service identities. Establish a VS Code extension allowlist policy enforced via MDM or developer workstation policy. Review GitHub repository access controls and rotate secrets on a defined schedule independent of

incident triggers.

IR / Forensic Enrichment

Triage Priority	IMMEDIATE
Escalation Criteria	Escalate to CISO, legal counsel, and external IR retainer immediately if CloudTrail or Kubernetes audit logs confirm credential use outside the organization's AWS regions or Kubernetes clusters during the exposure window, if any of the ~3,800 exfiltrated GitHub repositories contain customer PII/PHI (triggering breach notification obligations under GDPR, CCPA, or HIPAA), or if the Mini Shai-Hulud worm is confirmed to have propagated beyond initially identified hosts to production systems.
Recovery Notes	After credential rotation and pipeline restoration, maintain active monitoring of all newly issued AWS IAM role ARNs, Kubernetes service account tokens, and Vault token accessors for 30 days minimum, specifically alerting on any `AssumeRole` or `TokenReview` events from IP ranges not matching your known CI/CD or developer egress addresses — residual attacker-held credential copies from TeamPCP's exfiltration are a realistic persistence risk. Validate the integrity of all GitHub repositories accessible to the compromised employee account by comparing current repository contents against pre-incident commit hashes, checking for any unauthorized commits, branch protection rule changes, or webhook additions that may have been introduced during the breach window. Do not restore automated GitHub Actions workflows until all dependency files across all repositories have been audited for durabletask 1.4.1–1.4.3 references and updated to hash-pinned clean versions, as the worm's self-replication mechanism specifically targets CI/CD environments to propagate.
Forensic Artifacts	Python site-packages directory artifacts: malicious durabletask 1.4.1–1.4.3 __init__.py and setup.py files containing Mini Shai-Hulud worm payload code, preserved with original file timestamps and SHA-256 hashes, located at /usr/lib/python3/dist-packages/durabletask/ or within virtualenv lib/python*/site-packages/durabletask/ on affected hosts. | AWS CloudTrail logs for the exposure window filtered on EventNames: GetCredentials (IMDS), AssumeRole, GetSecretValue, ListBuckets, GetObject — specifically events where the sourceIPAddress field matches EC2 instance private IPs running durabletask workloads, evidencing the worm's automated credential harvesting from 169.254.169.254/latest/meta-data/iam/security-credentials/. | Linux process accounting logs (/var/log/psacct or auditd logs at /var/log/audit/audit.log) filtered for SYSCALL records with comm=python3 and EXECVE records showing child process execution (e.g., curl, wget, or base64) spawned by the durabletask package during its malicious initialization, consistent with worm staging and C2 callback behavior. | DNS resolver query logs from the exposure window for all queries originating from developer workstations or CI/CD runner hosts, specifically non-baseline fully qualified domain names resolved by Python interpreter processes — these represent the Mini Shai-Hulud worm's C2 beacon destinations via MITRE ATT&CK T1071.001 and are critical for IOC extraction and threat intelligence sharing. | GitHub audit log export (available via GitHub Enterprise audit log API or GitHub.com organization audit log) covering the breach window, specifically `git.clone`, `repo.download`, `repo.create_fork`, and `org.add_member` event types tied to the compromised employee account, bounding the scope of TeamPCP's ~3,800 repository exfiltration and identifying any secondary access or persistence mechanisms established within the GitHub organization.

Per-Action IR Details

Step 1: Containment — Immediately audit all Python environments for durabletask versions 1.4.1, 1.4.2, and 1.4.3 and guardrails-ai packages; isolate any host where these versions are installed from network access. Suspend GitHub Actions workflows and CI/CD pipelines that pull these packages until clean versions are confirmed. Revoke and rotate all cloud credentials accessible from affected hosts, including AWS IAM roles associated with EC2 instance metadata, Kubernetes service account tokens, HashiCorp Vault tokens, 1Password and Bitwarden vault credentials, and SSH keys present on affected systems.

NIST Phase: Containment

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

Controls: NIST IR-4 (Incident Handling), NIST AC-17 (Remote Access), NIST SC-7 (Boundary Protection), CIS 7.1 (Establish and Maintain a Vulnerability Management Process), CIS 7.2 (Establish and Maintain a Remediation Process)

Compensating: Run `pip list --format=columns | grep -E 'durabletask|guardrails'` across all Python environments via SSH batch (pdsh or pssh) or Ansible ad-hoc command. For CI/CD, immediately set GitHub Actions environment protection rules to require manual approval, blocking any workflow referencing durabletask. Block outbound traffic from affected hosts using iptables: `iptables -I OUTPUT -m owner --uid-owner -j DROP` until package audit completes. Use `aws iam list-attached-role-policies` and `aws iam delete-access-key` via AWS CLI to revoke compromised IAM keys. Rotate SSH keys by removing suspect public keys from ~/.ssh/authorized_keys on affected hosts and regenerating with `ssh-keygen -t ed25519`.

Evidence: Before isolating, capture full network connection state from affected hosts: `ss -tunap > /evidence/netstat_$(hostname)_$(date +%s).txt` to document active C2 or exfiltration sessions originating from the Mini Shai-Hulud worm's self-propagation mechanism. Capture running process tree: `ps auxf > /evidence/proctree_$(hostname)_$(date +%s).txt` to identify any child processes spawned by durabletask's malicious __init__.py or setup.py payload. Dump environment variables of the Python interpreter process: `cat /proc//environ | tr '\0' '\n' > /evidence/env_$(hostname)_$(date +%s).txt` to capture any AWS_ACCESS_KEY_ID, AWS_SECRET_ACCESS_KEY, VAULT_TOKEN, or KUBECONFIG values resident in memory. Preserve pip cache directory at ~/.cache/pip/wheels/ and site-packages/durabletask* before removal to retain the malicious package artifacts for forensic analysis.

Step 2: Detection — Query package manifests, requirements.txt, pyproject.toml, and lock files across all environments for durabletask==1.4.1, 1.4.2, or 1.4.3 and any pinned guardrails-ai version deployed in the same timeframe. In AWS CloudTrail, look for unusual IMDS calls (169.254.169.254) from EC2 instances running Python workloads, unexpected AssumeRole events, and data exfiltration to unknown endpoints. In Kubernetes audit logs, look for service account token reads outside normal application behavior. Review VS Code extension inventories for Nx Console versions installed via unofficial sources. Check DNS and HTTP proxy logs for C2 beacon patterns consistent with T1071.001 outbound from developer or CI/CD hosts.

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-2 (Event Logging), NIST IR-5 (Incident Monitoring), CIS 8.2 (Collect Audit Logs), CIS 7.1 (Establish and Maintain a Vulnerability Management Process)

Compensating: Run `find / -name 'requirements*.txt' -o -name 'pyproject.toml' -o -name 'Pipfile.lock' 2>/dev/null | xargs grep -l 'durabletask'` to locate all dependency files referencing the package. For AWS CloudTrail without a SIEM, use AWS CLI: `aws cloudtrail lookup-events --lookup-attributes AttributeKey=EventName,AttributeValue=AssumeRole --start-time ` and separately query for IMDS abuse using VPC Flow Logs filtered to destination 169.254.169.254. For Kubernetes, run `kubectl get events --all-namespaces --field-selector reason=TokenReview` and review API server audit log at /var/log/kubernetes/audit.log filtering on `verb=create` and `resource=serviceaccounts/token`. For VS Code extension detection, run `ls ~/.vscode/extensions/ | grep -i nx-console` on developer workstations and compare install path metadata against the official publisher ID `nrwl.angular-console`. Deploy the public Sigma rule for T1071.001 (Application Layer Protocol: Web Protocols) against proxy/DNS logs using `sigma convert` with grep backend as a no-SIEM fallback. Use osquery: `SELECT name, version, path FROM python_packages WHERE name IN

('durabletask', 'guardrails-ai')` to enumerate affected hosts at scale.

Evidence: AWS CloudTrail logs filtered for the exposure window: specifically `GetCallerIdentity`, `AssumeRole`, `GetSecretValue` (Secrets Manager), and `DescribeInstances` API calls originating from EC2 instance profile ARNs associated with durabletask-running workloads — the Mini Shai-Hulud worm targets IMDS at 169.254.169.254/latest/meta-data/iam/security-credentials/ to harvest temporary credentials. Kubernetes API server audit logs at /var/log/kubernetes/audit.log for `TokenRequest` and `create/serviceaccounts/token` events outside normal application namespaces, indicating the worm's lateral movement toward cluster credentials. DNS query logs from Route 53 Resolver or on-prem DNS showing repeated resolution of domains not in the application's normal baseline from Python runtime processes — the worm's C2 beacon pattern via T1071.001 (MITRE ATT&CK T1071.001). GitHub audit log (if accessible) for bulk repository clone or download events from the compromised employee account during the breach window, specifically `git.clone` or `repos.download` event types covering the ~3,800 exfiltrated repositories. VS Code extension directory on developer workstations at `%APPDATA%\Code\User\extensions\` (Windows) or `~/.vscode/extensions/` (Linux/macOS) for Nx Console extension folders with modified timestamps inconsistent with official Marketplace release dates, indicating trojanized installation.

Step 3: Eradication — Downgrade or remove durabletask to version 1.4.0 or upgrade to a verified clean release if available from the official Dapr GitHub repository (https://github.com/dapr/durabletask-python). Remove any installed guardrails-ai versions flagged in vendor advisories. Uninstall Nx Console extensions sourced outside the official VS Code Marketplace and re-install from the verified publisher. Re-image any host confirmed to have executed malicious package code. Rotate all credentials identified in Step 1 containment, even if compromise is unconfirmed on that host.

NIST Phase: Eradication

Reference: NIST 800-61r3 §3.4 — Eradication

Controls: NIST SI-2 (Flaw Remediation), NIST SI-3 (Malicious Code Protection), NIST SI-7 (Software, Firmware, and Information Integrity), NIST CM-3 (Configuration Change Control), CIS 7.3 (Perform Automated Operating System Patch Management), CIS 7.4 (Perform Automated Application Patch Management), CIS 2.2 (Ensure Authorized Software is Currently Supported)

Compensating: Verify durabletask 1.4.0 package integrity before installation by comparing SHA-256 hash against the value published on PyPI: `pip download durabletask==1.4.0 --no-deps -d /tmp/pkg_verify && sha256sum /tmp/pkg_verify/durabletask*.whl`. For Nx Console, verify the reinstalled extension's VSIX hash matches the official Marketplace release by downloading directly: `code --install-extension nrwl.angular-console` and confirming the extension ID in `~/.vscode/extensions/nrwl.angular-console-*/package.json`. Write a YARA rule targeting Mini Shai-Hulud's known self-replication behavior (scanning for durabletask package directory traversal and IMDS HTTP request strings in Python bytecode) and run against site-packages: `yara -r /path/to/mini_shai_hulud.yar /usr/lib/python3/dist-packages/`. For confirmed-execution hosts, do not attempt in-place remediation — re-image from a known-good golden AMI or VM snapshot predating the durabletask 1.4.1 release date. Use `pip-audit` post-reinstall to confirm no remaining vulnerable packages: `pip-audit --require-hashes -r requirements.txt`.

Evidence: Before re-imaging, acquire a full memory dump of confirmed-execution hosts using LiME (Linux Memory Extractor) kernel module to preserve in-memory worm state, harvested credentials, and C2 communication buffers: `insmod lime.ko 'path=/evidence/mem_$(hostname).lime format=lime'`. Preserve the malicious durabletask package files intact at /usr/lib/python3/dist-packages/durabletask/ or the virtualenv equivalent — specifically __init__.py, setup.py, and any .pyc bytecode files that encode the worm's replication logic. Capture pip installation logs at ~/.local/share/pip/log/ or /var/log/pip.log to establish the exact timestamp durabletask 1.4.1–1.4.3 was installed, correlating with CloudTrail credential theft events. For developer workstations, export VS Code extension metadata: `cat ~/.vscode/extensions/*/package.json | python3 -c 'import sys,json; [print(d.get("publisher"),d.get("name"),d.get("version")) for d in [json.load(open(f)) for f in sys.argv[1:]]]'` to document the trojanized Nx Console version and installation source.

Step 4: Recovery — After credential rotation, verify new credentials are functional and access policies are scoped to least privilege. Confirm no unauthorized IAM users, roles, or Kubernetes cluster role bindings were created during the exposure window. Monitor AWS CloudTrail, Kubernetes audit logs, and Vault audit logs for

continued anomalous access patterns for a minimum of 30 days. Validate that CI/CD pipelines are pulling from verified package sources with hash-pinned dependencies before restoring automated build workflows.

NIST Phase: Recovery

Reference: NIST 800-61r3 §3.5 — Recovery

Controls: NIST IR-4 (Incident Handling), NIST AC-6 (Least Privilege), NIST AC-2 (Account Management), NIST AU-6 (Audit Record Review, Analysis, and Reporting), NIST SI-7 (Software, Firmware, and Information Integrity), CIS 5.1 (Establish and Maintain an Inventory of Accounts), CIS 6.1 (Establish an Access Granting Process), CIS 6.2 (Establish an Access Revoking Process)

Compensating: Enumerate all IAM users and roles created during the exposure window: `aws iam list-users --query 'Users[?CreateDate>=``]'` and `aws iam list-roles --query 'Roles[?CreateDate>=``]'`; cross-reference against your pre-incident IAM baseline. For Kubernetes, audit all ClusterRoleBindings and RoleBindings created during the window: `kubectl get clusterrolebindings,rolebindings --all-namespaces -o json | jq '.items[] | select(.metadata.creationTimestamp >= "")'`. Set up a 30-day CloudTrail event history alert for the newly rotated IAM role ARNs using CloudWatch Metric Filter targeting `errorCode=AccessDenied` and `eventName=AssumeRole` to catch any attacker-held credential copies still attempting access. Validate CI/CD pipeline integrity by auditing GitHub Actions workflow YAML files for hardcoded package versions and replacing with hash-pinned references: `pip download && pip hash ` to generate SHA-256 hashes for requirements.txt `--hash=sha256:` syntax.

Evidence: AWS CloudTrail `ListBuckets`, `GetObject`, and `PutObject` events from the exposure window to assess whether the worm or the TeamPCP actor exfiltrated data from S3 buckets accessible via the harvested EC2 instance profile credentials. HashiCorp Vault audit log (enabled via `vault audit enable file file_path=/var/log/vault_audit.log`) reviewed for secret reads on paths accessible to the compromised Vault tokens during the exposure window — specifically any `kv/data/` path reads that are anomalous relative to the application's normal access pattern. Kubernetes API server audit log entries for `create` verbs against `clusterrolebindings` or `secrets` resources within the exposure window, indicating privilege escalation attempts by the worm using harvested service account tokens. GitHub audit log export covering the breach window for `repo.download`, `git.clone`, and `org.invite` events tied to the compromised employee account to bound the scope of the ~3,800 repository exfiltration.

Step 5: Post-Incident — This campaign exposed three control gaps: absence of package integrity verification (CWE-494) in CI/CD pipelines, insufficient secrets isolation on developer endpoints (CWE-522), and lack of VS Code extension allowlisting (T1176). Implement hash-pinned dependencies with automated integrity checks (e.g., pip-audit, Dependabot, or Endor Labs) for all PyPI packages. Enforce developer endpoint controls that restrict cloud credential access to dedicated CI/CD service identities. Establish a VS Code extension allowlist policy enforced via MDM or developer workstation policy. Review GitHub repository access controls and rotate secrets on a defined schedule independent of incident triggers.

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-2 (Flaw Remediation), NIST SI-7 (Software, Firmware, and Information Integrity), NIST SA-12 (Supply Chain Protection), NIST CM-7 (Least Functionality), CIS 7.1 (Establish and Maintain a Vulnerability Management Process), CIS 7.2 (Establish and Maintain a Remediation Process), CIS 2.1 (Establish and Maintain a Software Inventory), CIS 2.3 (Address Unauthorized Software)

Compensating: For hash-pinned PyPI dependency enforcement without enterprise tooling, implement a pre-commit hook using `pip-audit` and `hashin`: install hashin (`pip install hashin`) and run `hashin durabletask==1.4.0 requirements.txt` to auto-populate SHA-256 hashes; enforce via `.pre-commit-config.yaml` with `pip-audit` as a hook. For VS Code extension allowlisting without MDM, deploy a startup script on developer workstations that compares installed extensions against an approved list: `code --list-extensions | grep -vFf /etc/vscode/approved_extensions.txt` and alerts or removes unapproved entries. For secrets isolation on developer endpoints, configure AWS credential profiles in ~/.aws/config to use `credential_process` pointing to a script that requires MFA or short-lived STS tokens, preventing long-lived credentials from residing on disk: `aws sts get-session-token --serial-number --token-code `. Implement a GitHub Actions workflow that runs `pip-audit --require-hashes` as a required status check before any build job executes, blocking pipelines that reference unverified package versions.

Evidence: Lessons-learned documentation should include a full timeline reconstructed from: pip installation logs, CloudTrail credential usage events, Kubernetes audit logs, GitHub audit logs, and DNS/proxy logs — correlated to establish when durabletask 1.4.1–1.4.3 first entered each environment, when the worm first made IMDS calls, and when TeamPCP first accessed exfiltrated GitHub repository data. Produce an inventory of all secrets confirmed or suspected to have been in scope during the exposure window, categorized by secret type (AWS IAM, Kubernetes SA token, Vault token, SSH key, password manager entry) and rotation status, to serve as evidence of breach scope for any required regulatory notification assessment. Retain the malicious durabletask package artifacts, memory dumps, and all log exports in write-protected, chain-of-custody storage for a minimum of 12 months per NIST AU-11 (Audit Record Retention) to support any future law enforcement referral or civil litigation related to TeamPCP's campaign.

Detection Guidance

1. Package presence: Scan all environments for durabletask 1.4.1-1.4.3 using 'pip list', SBOM tooling, or dependency scanning (pip-audit, Endor Labs, StepSecurity). Search requirements.txt, Pipfile.lock, pyproject.toml, and Poetry lock files. 2. AWS IMDS abuse (T1552.005): In CloudTrail, query for GetMetadata or calls to 169.254.169.254 originating from EC2 instances running Python workloads, especially outside normal application call patterns. Look for unexpected AssumeRole, GetSessionToken, or ListBuckets calls following IMDS access. 3. Kubernetes token reads: In Kubernetes audit logs, filter for GET requests to /var/run/secrets/kubernetes.io/serviceaccount/token or API calls using service account tokens from unexpected source IPs or processes. 4. Exfiltration (T1567, T1071.001): Review HTTP/S proxy and DNS logs for outbound connections from developer or CI/CD hosts to domains not in your known-good allowlist, particularly shortly after package installation or build execution. 5. VS Code extension: Audit installed VS Code extensions across developer endpoints for Nx Console; verify publisher identity and version hash against the official VS Code Marketplace listing. 6. Behavioral: Look for Python processes spawning shell commands, reading SSH key directories (~/.ssh/), reading cloud credential files (~/.aws/credentials, ~/.config/gcloud/), or accessing secrets manager APIs outside expected application workflows.

Indicators of Compromise

Type	Value	Context	Confidence
HASH	`durabletask==1.4.1 / 1.4.2 / 1.4.3 (PyPI)`	Malicious package versions containing Mini Shai-Hulud worm payload; exact file hashes not publicly confirmed at time of report — use version string for detection	HIGH
URL	`https://pypi.org/project/durabletask/`	Official PyPI listing; versions 1.4.1–1.4.3 reported removed; verify current safe version before installing	HIGH
DOMAIN	`169.254.169.254`	AWS EC2 IMDS endpoint targeted by worm for credential harvesting (T1552.005); unexpected calls from Python processes are a behavioral IOC	HIGH

Framework Mappings

MITRE-ATTACK

- **T1078.004** — Cloud Accounts
- **T1021.001** — Remote Desktop Protocol
- **T1552.001** — Credentials In Files
- **T1554** — Compromise Host Software Binary
- **T1567** — Exfiltration Over Web Service
- **T1071.001** — Web Protocols
- **T1552** — Unsecured Credentials
- **T1552.004** — Private Keys
- **T1567.001** — Exfiltration to Code Repository
- **T1078** — Valid Accounts
- **T1485** — Data Destruction
- **T1552.005** — Cloud Instance Metadata API
- **T1021.004** — SSH
- **T1543** — Create or Modify System Process
- **T1021** — Remote Services
- **T1195.001** — Compromise Software Dependencies and Development Tools
- **T1176** — Software Extensions
- **T1059.006** — Python

NIST-800-53R5

- **AC-2** — Account Management
- **AC-6** — Least Privilege
- **IA-2** — Identification and Authentication (Organizational Users)
- **IA-5** — Authenticator Management
- **AC-17** — Remote Access
- **AC-3** — Access Enforcement
- **CM-7** — Least Functionality
- **SI-7** — Software, Firmware, and Information Integrity
- **CM-3** — Configuration Change Control
- **SR-2** — Supply Chain Risk Management Plan

OWASP-TOP10-2021

- **A01:2021** — Broken Access Control
- **A04:2021** — Insecure Design
- **A07:2021** — Identification and Authentication Failures
- **A08:2021** — Software and Data Integrity Failures

CIS-V8

- **3.3** — Configure Data Access Control Lists
- **5.2** — Use Unique Passwords

- **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

HIPAA-SECURITY

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

SOC2-TSC

- **CC6.1** — Logical access security software, infrastructure, and architectures
- **CC9.2** — Manages risks associated with vendors and business partners

NIST-CSF-2

- **GV.SC-01** — Cybersecurity supply chain risk management program

ISO-27001-2022

- **A.5.21** — Managing information security in the ICT supply chain
- **A.5.23** — Information security for use of cloud services

MITRE ATT&CK Mapping

Technique ID	Technique Name	Tactic
T1078.004	Cloud Accounts	Defense-Evasion
T1021.001	Remote Desktop Protocol	Lateral-Movement
T1552.001	Credentials In Files	Credential-Access
T1554	Compromise Host Software Binary	Persistence
T1567	Exfiltration Over Web Service	Exfiltration
T1071.001	Web Protocols	Command-And-Control
T1552	Unsecured Credentials	Credential-Access
T1552.004	Private Keys	Credential-Access
T1567.001	Exfiltration to Code Repository	Exfiltration
T1078	Valid Accounts	Defense-Evasion
T1485	Data Destruction	Impact
T1552.005	Cloud Instance Metadata API	Credential-Access
T1021.004	SSH	Lateral-Movement

Technique ID	Technique Name	Tactic
T1543	Create or Modify System Process	Persistence
T1021	Remote Services	Lateral-Movement
T1195.001	Compromise Software Dependencies and Development Tools	Initial-Access
T1176	Software Extensions	Persistence
T1059.006	Python	Execution

Sources

Source	URL	Tier
Security News	https://thehackernews.com/2026/05/github-investigating-teampcp-clai...	T3
Trojanized Microsoft SDK: durabletask 1.4.1 through 1.4.3 Deliver ...	https://www.endorlabs.com/learn/trojanized-microsoft-sdk-durabletas...	T3
Microsoft's durabletask PyPI Package Compromised in Supply ...	https://www.stepsecurity.io/blog/microsofts-durabletask-pypi-packag...	T3
Microsoft's durabletask package on PyPi Compromised. Mini Shai ...	https://www.aikido.dev/blog/durabletask-package-compromised-mini-sh...	T3
Security - Overview · dapr/durabletask-python - GitHub	https://github.com/dapr/durabletask-python/security	T3

Generated 2026-05-20 06:43 UTC by TJS Security Command Center