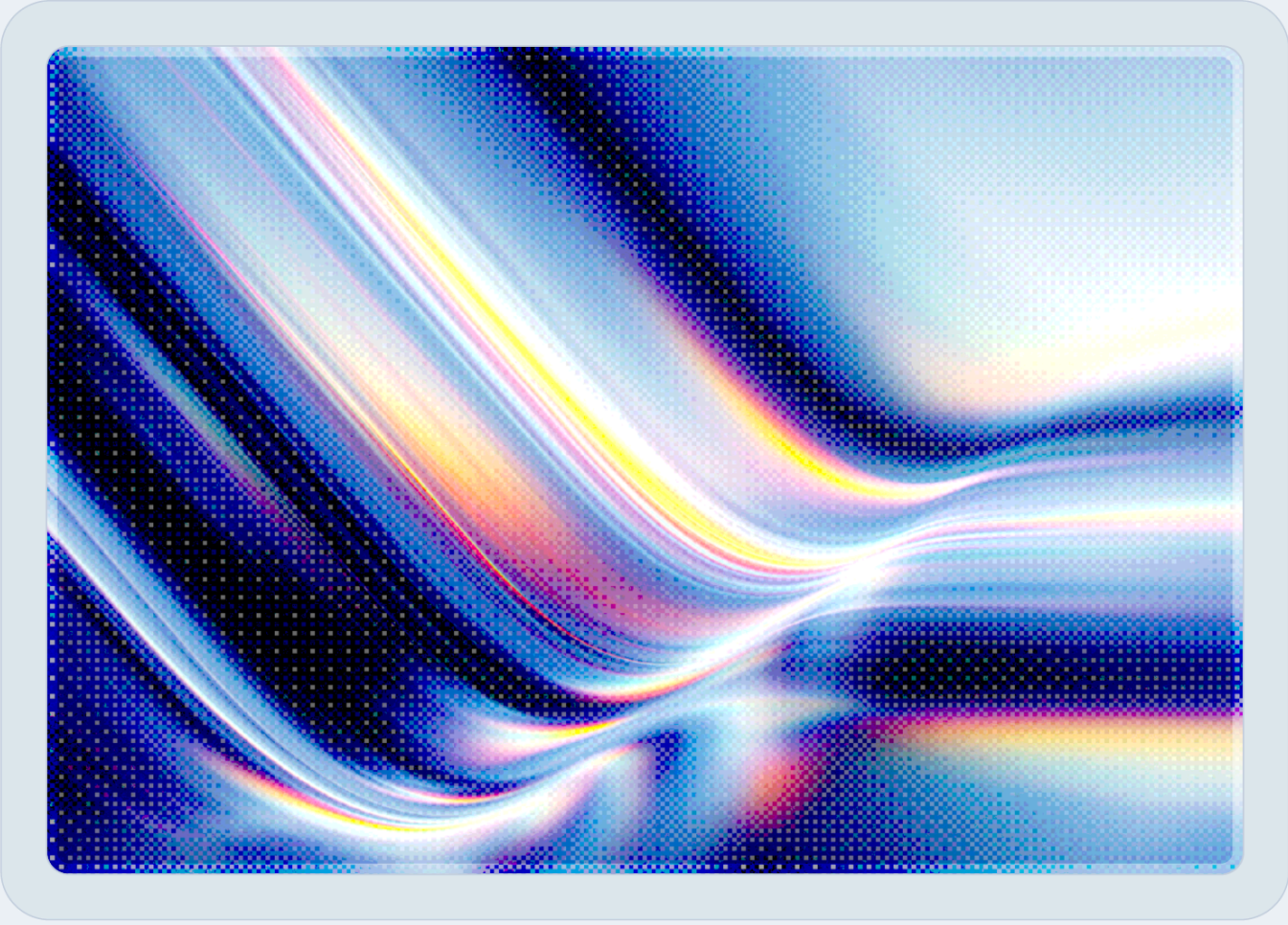
CSAI Foundation | Cloud Security Alliance

Hijacked agent-framework on PyPI: Startup-Hook Credential Theft Hits Agent Builders

Security Implications and Guidance for the Microsoft Agent Framework Package Compromise

2026-11-19

AI-assisted Rapid Research



CSAI | cloud security alliance®

© 2026 Cloud Security Alliance. Some rights reserved.

You may download, store, display, view, print, redistribute, and link to this document in its original, unmodified form, provided that attribution to the Cloud Security Alliance is maintained and all trademark and copyright notices remain intact.

This document may not be modified or altered. You may quote portions of the document as permitted by the Fair Use provisions of the United States Copyright Act, provided that attribution is given to the Cloud Security Alliance.

This document may be shared on professional and social media platforms in its original form with attribution.

This document was generated with AI assistance and has not undergone official CSA review and approval processes.

Key Takeaways

On November 17, 2026, an unknown actor published two malicious releases of `agent-framework`, the PyPI meta-package for Microsoft Agent Framework, under the project's legitimate name on PyPI, so they appeared as ordinary new releases. Versions 1.26.1 and 1.26.2 shipped a credential stealer in their wheels. PyPI quarantined both within about 45 minutes, and version 1.26.0 is the last known clean release [1][2][3]. Because the payload is delivered through a Python `.pth` startup hook, it runs on every Python interpreter launch in an affected environment. That includes `pip`, `pytest`, IDE language servers and container entrypoints, even when no application code ever imports the framework [2]. The stealer collected secrets from environment variables, credential files and the environments of other processes, and it tried to mint live Azure managed-identity, Azure DevOps and GitHub Actions OIDC tokens before sending everything to a `webhook.site` endpoint [1][2].

The first practical conclusion for security teams concerns scope. Any host, container image, CI runner or cache that installed 1.26.1 or 1.26.2 should be treated as compromised, every credential reachable from it should be rotated, and the environment should be rebuilt from a clean base rather than cleaned in place [1][2]. Remediation itself carries a trap: running `pip uninstall` in an affected environment re-triggers the payload, so responders should set `_AF_TEL=1` first or use tooling that does not start the infected interpreter [2].

Attempted token minting is the most serious potential exposure, although researchers have not confirmed that any token was actually issued [1][5]. Access tokens obtained this way generally cannot be revoked by the victim and stay valid until they expire, which can be up to about 24 hours for Azure managed-identity tokens. For that reason, identity-plane review matters as much as secret rotation [1][2]. Finally, the initial access vector remains unconfirmed. Researchers agree only that the attacker kept publishing access after the first quarantine, which shows that quarantining a release did not revoke the attacker's ability to publish [1][2][3].

Background

Microsoft Agent Framework is Microsoft's open-source SDK for building AI agents and multi-agent workflows in Python and .NET. Microsoft introduced it as the successor to Semantic Kernel and AutoGen, combining AutoGen's agent abstractions with Semantic Kernel's enterprise state management and telemetry [9]. On PyPI, the `agent-framework` package is a thin meta-package. The working code lives in `agent-framework-core` and a family of satellite packages, and the meta-package pins those dependencies and exposes a small stub module [2]. Reporting puts the meta-package at roughly 190,000–200,000 downloads per month [3][4][5], while JFrog cites about 40,000 per week [2]; the sources differ slightly, but either figure indicates broad adoption. The likely audience is developers and platform teams building agents against Azure OpenAI, Azure AI Foundry and other model providers. Those environments typically hold model-provider API keys, cloud credentials and CI/CD secrets at the same time.

The incident fits a pattern that CSA has tracked through 2026. In March, the TeamPCP campaign poisoned the `litellm` PyPI package with a `.pth`-based credential stealer aimed at AI middleware, which aggregates credentials for many model providers [10]. In April, the `elementary-data` compromise used a GitHub Actions script injection to push a `.pth` infostealer through the project's own release pipeline [12]. In June, a takeover of a dormant contributor account backdoored roughly 144 packages in the `@mastra` npm scope, another AI agent framework [11]. The Hacker News also reports that this is the second Microsoft-related PyPI compromise of 2026, after TeamPCP pushed malicious versions of Microsoft's `durabletask` client in May [4]. Taken together, these cases suggest that AI agent frameworks and their developer tooling have become a recurring target.

Timeline

The following timeline is reconstructed from registry timestamps and researcher reporting. Times are UTC on November 17, 2026 unless noted, and are shown at minute resolution; the sources give seconds where relevant.

Time	Event	Source
Nov 12	`agent-framework` 1.26.0 (last clean release) published as wheel and sdist	[1]
20:26	ZIP headers show the 1.26.1 wheel was repacked locally	[2]

Time	Event	Source
20:35	1.26.1 uploaded to PyPI (wheel only, no provenance attestation)	[1][2]
~20:41	Automated dependency bots (e.g., Renovate) begin opening upgrade PRs downstream	[2]
21:06	PyPI quarantines 1.26.1	[1]
21:08	1.26.2 uploaded about 2.5 minutes after the quarantine, using a relocated hook	[1][3]
21:16	PyPI quarantines 1.26.2; exposure window closes (sources give ~21:16–21:20)	[1][4]
Nov 18	StepSecurity, JFrog, Socket, OX Security and others publish analyses	[1][2][3][4][5]

Security Analysis

The startup-hook delivery mechanism

The defining feature of this attack is where the malicious code sits, not what the stealer does. Python's `site` module processes every `.pth` file in a `site-packages` directory during interpreter initialization, and any line in such a file that begins with `import` is executed as code [6]. The mechanism exists so that packages can extend `sys.path`, but it also gives anyone who can place a file in `site-packages` a way to run code on every interpreter start. In version 1.26.1, the attacker added a 28-byte `agent_framework_meta.pth` file containing `import agent_framework_meta`. They also appended a call to a new `_telemetry` module in the stub package's `__init__.py` [2]. After the first quarantine, version 1.26.2 restored the original `__init__.py` and moved the hook into a separately named `af_runtime_diag.pth` and `af_runtime_diag.py` pair [1][2]. That change appears intended to make a diff of the framework's own module look clean.

Two design choices kept the compromise quiet. First, the malicious wheels kept the legitimate dependency pin on `agent-framework-core[all]==1.26.0` and did not touch the framework code, so applications kept working with no visible errors [2][3]. Second, the payload detached itself. On

Linux and macOS it forks, calls `setsid()` and runs the collector in a child process while the parent continues normally. On Windows it falls back to a daemon thread. An environment variable, `_AF_TEL=1`, prevents recursive execution [1][2]. Researchers found no persistence beyond the `.pth` file, no second-stage loader and no worm-like propagation [1][2]. The `.pth` file is persistence enough, though, because it fires again on every Python launch for as long as it stays on disk. This has a consequence that responders often miss: running `pip uninstall` with the infected interpreter executes the stealer one more time [2].

What was collected

The collector is broad rather than targeted, but its search patterns show a clear interest in cloud and AI developer environments. It matched environment-variable names against 31 substrings in 1.26.1 and 35 in 1.26.2. These included `TOKEN`, `SECRET`, `KEY`, `AZURE`, `AWS`, `GITHUB`, `OPENAI`, `COSMOS` and `OIDC`, and the generic patterns also capture third-party model keys such as `ANTHROPIC_API_KEY` and `GEMINI_API_KEY` [1][2]. On Linux it read `/proc/*/environ` for up to 300 other processes. That reaches secrets belonging to sibling processes on the same host or in the same container, not only those of the process that triggered the hook [1][2]. It also read up to 60 files in 1.26.1 and up to 80 in 1.26.2 that matched credential-file globs, as summarized below.

Target class	Examples reported	Versions
Cloud CLI caches	`~/.azure/*.json` token caches, Azure AI Foundry configs	Both [2]
Package and source credentials	`pip.conf`, `.npmrc`, `.netrc`, `.git-credentials`	Both [1] [2]
Container and ML platform credentials	`.docker/config.json`, Hugging Face tokens	Both [1] [2]
Application secrets	`/app/.env`, `/workspace/.env`, `/app/*.json`	Both [2]
CI agent credentials	Azure DevOps self-hosted agent `.credentials` stores	Both [1] [2]

Target class	Examples reported	Versions
CI runner directories	GitHub Actions runner paths under `/home/runner`	1.26.2 [1] [2]

Live token minting

The collector did more than read stored secrets: it tried to obtain new credentials. On Azure compute, it queried the App Service identity endpoint or the instance metadata service at `169.254.169.254` and requested OAuth tokens for about twelve audiences. These included Azure Resource Manager, Key Vault, Storage, Microsoft Graph, Azure AI Foundry and Azure DevOps [1][2]. Both versions tried to mint Azure DevOps OIDC tokens using the pipeline access token. Version 1.26.2 added GitHub Actions OIDC requests for audiences used in Azure token exchange [1].

Rotating every stored secret will not undo this part of the attack. Access tokens issued to managed identities and federated workloads have no static secret behind them to rotate, generally cannot be revoked by the victim, and remain valid until expiry, which JFrog puts at up to 24 hours [2]. Lifetimes vary by token type: GitHub Actions OIDC ID tokens are short-lived, and Entra access tokens obtained by exchanging them typically last on the order of an hour, so the 24-hour figure is the worst case rather than the norm. It is also important to separate attempt from outcome. StepSecurity notes that its findings come from static analysis and that it has not confirmed any token was successfully issued [1][5]. JFrog adds that the Azure DevOps OIDC request was malformed and would have failed. Even so, `SYSTEM_ACCESSTOKEN` would still have leaked through environment variables wherever it was mapped into a pipeline step [2].

Exfiltration and operational flaws

The collected data was serialized, compressed with zlib, base64-encoded and sent in one HTTPS POST to `webhook.site/d381fd80-…?t=telemetry`; the full inbox UUID is published as an indicator of compromise in [1] and [2]. The User-Agent was `af-runtime-diagnostics/1` for 1.26.1 and `runtime-diagnostics/1` for 1.26.2 [1][2]. The payload was not encrypted, and JFrog found that the free webhook inbox was returning HTTP 429 rate-limit responses when checked. Victims whose requests arrived after the limit may not have had their data captured. Earlier victims probably did, and anyone who knows the UUID may be able to read the captured requests [2], which is why this note does not reproduce it in full.

Other weaknesses limit the stealer's reach without removing the risk. On non-Azure Linux hosts that silently drop metadata-service traffic, the IMDS probing adds about 104 seconds of timeouts before exfiltration. Most file globs are absolute Linux paths, so macOS developer machines are only partly covered. On Windows, short-lived commands may exit before the daemon thread finishes [2]. JFrog notes that the operator made several mistakes and that the payload carried no encryption, no actor markers, no systemd persistence and no worm propagation [2]. Defenders should not read that as low impact: one `.pth` line plus an unchanged dependency pin was enough to run code wherever the package was installed.

How the attacker obtained publishing access

The access vector has not been established. Microsoft had not commented publicly as of BleepingComputer's reporting [3], and none of the published reports explain how the attacker obtained publishing rights [5]. The evidence does narrow the possibilities. JFrog found that the wheel's metadata still named the `flit` build backend while its ZIP headers came from Info-ZIP `zip 3.0` under a Unix `uid=1000` account. JFrog reads this as the clean wheel being repacked by a local script rather than produced by Microsoft's release pipeline [2]. StepSecurity found that both malicious releases were wheel-only, unlike 1.26.0, which shipped both a wheel and an sdist. The malicious releases also carried no provenance attestation and were uploaded with a `RestSharp/106.13.0.0` client user-agent, although StepSecurity cautions that the absence of an attestation alone does not identify the upload mechanism or establish a change from the legitimate release process [1].

StepSecurity's timeline also records a maintainer account that was created and added to the project on November 17, though it does not say what role that account played [1]. OX Security, quoted in other coverage, describes the attack as hijacking the "wheel creation flow" [4]. That characterization sits uneasily with JFrog's evidence of out-of-pipeline repackaging and should be treated as unconfirmed until Microsoft or PyPI publishes a root-cause account. JFrog also noted a UTC+8 clock in the archive timestamps [2]. A timestamp like that is easy to manipulate and does not support attribution, and no researcher has linked the activity to a known threat group [1][2].

The available evidence suggests, without confirming, that a long-lived PyPI API token or a maintainer session was compromised rather than a CI workflow. That makes it a different failure mode from `elementary-data`, where the attacker abused the project's own workflow and so inherited its provenance signals [12]. The difference matters for defense. Consumers whose ingestion tooling requires PyPI attestations would have rejected both malicious uploads, which carried none; mainstream `pip` does not perform that check by default, so the protection applies only to organizations that have deployed it [1][8].

Why agent builders are a distinct target

The `litellm` analysis describes AI middleware as a credential aggregation point, and agent-development environments share that property [10]. A developer or CI job building on Microsoft Agent Framework usually holds model-provider keys, Azure or Foundry identities, vector-store and database connection strings, and the repository and registry credentials needed to ship the agent. The stealer's search list, which includes Foundry configurations, Cosmos DB, Hugging Face and OpenAI patterns alongside generic cloud secrets, suggests the attacker knew this [2]. Agent workloads are often deployed onto managed compute with workload identities, which is exactly the setting where the token-minting logic works best. JFrog observed automated dependency bots opening upgrade PRs within about six minutes of the first malicious upload [2]. Organizations that auto-merge such updates or build images from floating version ranges could have been exposed within the 45-minute window, faster than a typical human review cycle.

Recommendations

Immediate Actions

Every environment that could have resolved `agent-framework` between 20:35 and about 21:20 UTC on November 17 needs scoping first. That includes developer workstations, CI runners, internal mirrors and container images built that evening. Responders should search lockfiles, `pip freeze` output, CI logs, Renovate and Dependabot PRs, and artifact-proxy caches for versions 1.26.1 and 1.26.2 and for the published wheel hashes. They should also look for the four malicious filenames (`agent_framework_meta.pth`, `agent_framework_meta/_telemetry.py`, `af_runtime_diag.pth`, `af_runtime_diag.py`) in every `site-packages` directory [1] [2]. On the network side, DNS and proxy logs should be checked for `webhook.site` lookups and for POSTs to the exfiltration path. A successful HTTP 200 response should be treated as confirmed transfer, and blocked requests treated separately [2][3].

Where exposure is confirmed or likely, containment begins with isolating affected hosts and runners. Responders should set `_AF_TEL=1` before running any Python command in them, or use `uv`, which does not execute `.pth` hooks during removal [2]. Because the environment variable works only by exploiting the malware's own recursion guard, it is best combined with `uv` or offline removal rather than relied on alone. Affected systems should then be rebuilt from a known-clean base image pinned to `agent-framework==1.26.0` with hash-pinned lockfiles; deleting the hook does not stop a

collector that is already running, and cleaning in place leaves caches behind [1][2]. Finally, both malicious wheels should be evicted from internal mirrors and artifact proxies, with explicit deny rules added for 1.26.1 and 1.26.2 [1][2].

Credential and identity response runs in parallel with containment, and the table below summarizes it.

Exposure	Response	Rationale
Secrets readable from the environment	Rotate model-provider API keys, cloud keys, registry and Git credentials, Hugging Face tokens and Azure DevOps service connections	The collector read environment variables, other processes' environments and credential files [1][2]
`~/.azure` token caches present	Explicitly revoke Entra ID sessions	A password reset alone does not reliably invalidate all cached refresh tokens [1][2]
Managed-identity and federated tokens	Recycle affected compute so tokens expire; review managed-identity sign-in logs, GitHub OIDC federated credentials and role assignments for activity from unexpected locations	Review should cover at least 24 hours after exposure, the maximum lifetime of the relevant managed-identity tokens [1][2]

Short-Term Mitigations

Once the immediate incident is contained, organizations should close the gaps that made a 45-minute window enough. Dependency cooldown policies, which hold newly published versions for a set period before automated tools may adopt them, would have kept this release out of any pipeline enforcing even a short cooldown, since PyPI quarantined it within about 45 minutes [1][2]. Package-ingestion controls should also verify PyPI attestations wherever a project publishes them and flag releases that lack provenance when earlier releases had it. They should also flag wheel-only releases that break a project's normal wheel-plus-sdist pattern [1][8].

Detection engineering can treat the appearance of new `.pth` files with executable `import` lines as a high-signal event once known-legitimate producers are baselined, such as setuptools' `distutils-precedence.pth`, editable-install finders and virtualenv hooks [6]. The same primitive has appeared in at least three 2026 PyPI compromises affecting AI and data tooling: `litellm` [10],

`elementary-data` [12] and this incident [1][2]. For CI, egress allowlisting on build runners blocks generic exfiltration services such as `webhook.site`. Removing `id-token: write` permissions and IMDS reachability from jobs that do not need them shrinks the token-minting surface [1][2].

Strategic Considerations

The deeper lesson concerns how long publishing credentials live and how the ecosystem responds once a compromise is underway. The attacker's ability to publish again minutes after quarantine shows that registry response stops a single artifact but does not cut off the actor behind it. Project owners should publish exclusively through PyPI Trusted Publishing, so that uploads come only from attested CI workflows with short-lived OIDC credentials, and, critically, should retire every long-lived API token and audit maintainer lists for recent changes. Enabling Trusted Publishing does not by itself disable other upload paths [7][8]. Consumers can encourage this by preferring, and eventually requiring, attested releases for high-privilege dependencies. Trusted Publishing on its own would not have stopped the workflow-injection route seen in `elementary-data` [12], so it should be paired with hardened release workflows.

Organizations building AI agents should also treat the agent development environment as a privileged zone. Model keys, cloud workload identities and deployment credentials should not sit together in a single developer shell or CI job. Scoped, short-lived credentials issued per task limit what any single interpreter-level compromise can harvest. Finally, the repeated use of `.pth` hooks in 2026 gives enterprise users reason to track the deprecation of `.pth` `import` lines, which begins in Python 3.15, and the replacement startup-configuration mechanism specified in PEP 829 [6][15]. Until older interpreters age out, organizations can enforce the same restriction through image-scanning and endpoint policy.

CSA Resource Alignment

CSA's research note on the TeamPCP campaign, *TeamPCP and the Cascading AI/ML Supply Chain Campaign* [10], is the closest precedent for this incident. That note documented the `litellm_init.pth` startup hook and described AI middleware as a category-level credential aggregation point. It also recommended hunting for unexplained `.pth` files in `site-packages`, which applies directly here. The `agent-framework` compromise uses the same execution primitive against a different layer of the AI stack, the agent-orchestration SDK rather than the model gateway, which is consistent with that note's conclusion that the risk is structural rather than tied to one product.

CSA's *elementary-data PyPI Compromise: Cloud Credential Theft via CI/CD Hijack* [12] offers a useful contrast on the access vector. There, a GitHub Actions script injection let the attacker publish through the project's own pipeline and inherit its provenance signals. Here, the evidence points to an out-of-pipeline upload without attestation. Read together, the two notes show why provenance verification and release-workflow hardening are complementary controls and why neither is sufficient alone.

Mastra npm Scope Takeover: AI Framework Supply Chain Backdoored [11] analyzed a June 2026 takeover of a dormant contributor account that republished an AI agent framework's whole package scope; CSA's follow-up note, *Sapphire Sleet Poisons Mastra AI npm Ecosystem* [16], subsequently attributed that campaign to a North Korean actor. The Mastra findings on publish rights that never expire for inactive or newly added accounts match the open questions here, including StepSecurity's report of a maintainer account added on the day of the attack. Its recommendations on maintainer-list auditing and cooldowns for floating version ranges apply equally to PyPI.

For control mapping, the CSA AI Controls Matrix (AICM) v1.1 [13] provides the governing framework. Its supply chain management, threat and vulnerability management, and identity and access management domains cover dependency provenance verification, rapid revocation of compromised credentials, and least-privilege scoping of workload identities and CI tokens. MAESTRO [14], CSA's agentic AI threat modeling framework, places this compromise in its Deployment and Infrastructure layer and is useful for tracing how credentials stolen from an agent's build environment could lead to compromise of the deployed agent and its tools.

References

[1] StepSecurity (R. Prabhu). "Microsoft agent-framework PyPI Package Compromised." StepSecurity Blog, November 18, 2026.

[2] JFrog Security Research (S. Satou). "Microsoft's agent-framework Hijacked on PyPI via Silent .pth Startup Hook." JFrog, November 18, 2026.

[3] B. Toulas. "Microsoft's Agent Framework PyPI package compromised to steal credentials." BleepingComputer, November 18, 2026.

[4] R. Lakshmanan. "Attackers Compromise Microsoft's Agent Framework PyPI Package to Steal Cloud Credentials." The Hacker News, November 18, 2026.

[5] S. Markovic. "Credential stealer slipped into Microsoft's official Python agent framework." Help Net Security, November 19, 2026.

[6] Python Software Foundation. "site — Site-specific configuration hook." Python 3 Documentation, accessed November 19, 2026.

[7] Python Packaging Authority. "Publishing to PyPI with a Trusted Publisher." PyPI Documentation, accessed November 19, 2026.

[8] Python Packaging Authority. "PyPI Attestations." PyPI Documentation, accessed November 19, 2026.

[9] Microsoft. "Introducing Microsoft Agent Framework." Microsoft Azure Blog, October 2025.

[10] Cloud Security Alliance. "TeamPCP and the Cascading AI/ML Supply Chain Campaign." CSA AI Safety Initiative Research Note, March 2026.

[11] Cloud Security Alliance. "Mastra npm Scope Takeover: AI Framework Supply Chain Backdoored." CSA AI Safety Initiative Research Note, June 2026.

[12] Cloud Security Alliance. "elementary-data PyPI Compromise: Cloud Credential Theft via CI/CD Hijack." CSA AI Safety Initiative Research Note, April 2026.

[13] Cloud Security Alliance. "AI Controls Matrix (AICM) v1.1." Cloud Security Alliance, 2025.

[14] Cloud Security Alliance. "Agentic AI Threat Modeling Framework: MAESTRO." Cloud Security Alliance Blog, February 2025.

[15] Python Software Foundation. "PEP 829 – Package Startup Configuration Files." Python Enhancement Proposals, accessed November 19, 2026.

[16] Cloud Security Alliance. "Sapphire Sleet Poisons Mastra AI npm Ecosystem." CSA AI Safety Initiative Research Note, June 2026.