CSAI Foundation

Cloud Security Alliance AI Safety Initiative

Reflexive Supply Chain: When Security Tools Become Attack Vectors

The TeamPCP Campaign, Toolchain Compromise, and the Limits of Defense-in-Depth

Unofficial AI-assisted Research

2026-04-08

Table of Contents

Executive Summary

A reflexive supply chain attack occurs when an adversary compromises the security or development tools that organizations use for defense—scanners, static analysis engines, package managers, AI gateways—thereby turning the act of self-protection into a delivery mechanism for compromise. Unlike conventional supply chain attacks that target the software an organization ships or consumes, reflexive attacks target the tools standing guard over that software. The organizations most exposed are precisely those exercising the greatest security diligence: the enterprises that scan every container image, check every build artifact, and validate every infrastructure configuration file.

Between March 19 and March 27, 2026, the financially motivated threat actor TeamPCP executed a cascading campaign that demonstrated the full operational potential of the reflexive supply chain threat. Over the course of eight days, the group force-pushed malicious code to the widely used Trivy vulnerability scanner, then leveraged credentials harvested from that compromise to pivot into Checkmarx's KICS infrastructure-as-code scanner and into fifty-plus npm packages. A parallel branch of the campaign targeted LiteLLM, the open-source AI gateway library present in over a third of monitored enterprise cloud environments, which manages API keys for more than one hundred large language model providers. The campaign was assigned CVE-2026-33634 and attributed a CVSS score of 9.4 [18]. Its confirmed downstream consequences included the exfiltration of 91.7 gigabytes of compressed data—approximately 340 gigabytes uncompressed—from 71 entities across the European Commission.

This whitepaper examines the TeamPCP campaign in detail, analyzes the structural conditions that enable reflexive supply chain attacks, situates the threat within the broader pattern of AI and developer toolchain compromise, and provides actionable guidance for organizations seeking to reduce their exposure. The central finding is that standard security controls—particularly automated vulnerability scanning integrated into CI/CD pipelines—create a new and underappreciated trust relationship: organizations implicitly trust the scanning infrastructure itself, and attackers have learned to exploit that trust.

Introduction and Background

Supply chain security has been a central focus of enterprise security investment and discourse since the SolarWinds compromise of 2020 and the Log4Shell vulnerability of 2021 revealed how profoundly a single trusted component could undermine thousands of downstream organizations. The past five years have produced an increasingly sophisticated body of research, regulation, and tooling designed to address this

threat. Software bills of materials, code signing, artifact provenance verification, and continuous dependency monitoring have moved from niche practices to mainstream expectations. Many organizations have responded by investing heavily in the very tooling now at the center of this report.

What the security community has been slower to examine is the recursive dimension of supply chain risk. When an organization deploys a vulnerability scanner, a static analysis framework, or an AI coding assistant to protect its software supply chain, it acquires a new supply chain dependency—the security tool itself. If that tool is compromised, the credential-rich, highly privileged context in which it operates becomes an asset available to an attacker rather than a safeguard against one. This is the essential logic of the reflexive supply chain attack: the more diligently an organization monitors its software supply chain, the more valuable and reachable its monitoring infrastructure becomes as an attack target.

The concept, though not previously named in security literature, has antecedents in documented incidents. In 2025, researchers identified npm packages that used legitimate secret scanning libraries to locate and exfiltrate credentials—weaponizing the same code patterns used by security tools to find exposed secrets [9]. CrowdStrike npm packages, used in development workflows, were among packages pulled into a self-replicating supply chain worm later that year [10]. The NX build system was compromised for approximately five hours in August 2025, with the malware specifically targeting AI CLI tools including Claude, Gemini, and Google's q CLI as reconnaissance and exfiltration instruments—the first documented case, according to StepSecurity researchers, of an attacker turning developer AI assistants into supply chain attack instruments [11]. The TeamPCP campaign of March 2026 represents the most operationally sophisticated and consequential expression of this pattern to date.

Understanding how TeamPCP developed, what it attacked, and how its cascade propagated requires examining both the threat actor's evolution and the structural characteristics of the tooling it targeted.

The TeamPCP Threat Actor

TeamPCP, also tracked by Google's Threat Intelligence Group under the designation UNC6780 and known by aliases including PCPcat and ShellForce, emerged as a financially motivated cloud threat actor in 2024 [1][2][13][20]. The group's early operations were characteristic of opportunistic cloud attackers: exploiting misconfigured Docker APIs, exposed Kubernetes dashboards, and unauthenticated Redis servers to steal credentials and deploy cryptocurrency miners. By the standards of 2024 cloud threat actors, TeamPCP's distinguishing characteristics were scale and automation—the group operated purpose-built scanning infrastructure capable of sweeping internet address ranges continuously and processing credential material at high throughput.

The group's technical ambitions grew substantially in late 2025. A December 2025 campaign used CVE-2025-55182, a remote code execution flaw in certain cloud management endpoints, to seed what TeamPCP called the "React2Shell" campaign—a reference to the automated chaining of initial access into shell execution across compromised environments [13]. More significant was the development of CanisterWorm, a self-propagating worm that used Internet Computer Protocol blockchain canisters as command-and-control infrastructure [1][12]. The ICP blockchain's decentralized architecture made the C2 channel censorship-resistant and effectively immune to traditional takedown techniques: there was no hosting provider to contact, no domain registrar to pressure, and no IP address to block. Security researchers tracking this technique in early 2026 characterized it as the first documented use of ICP canisters for active malware command-and-control in the wild [12].

By early 2026, TeamPCP had demonstrated the key attributes that would define its March campaign: automated, internet-scale reconnaissance; an ability to rapidly pivot from initial access to credential harvesting; novel C2 infrastructure resistant to standard countermeasures; and operational patience to build capability before deploying it at scale. Flare Research documented the group's expanding activity in February 2026, noting a pattern of targeting cloud-native infrastructure with increasingly sophisticated tools [13]. What researchers did not yet anticipate was that the group's next major campaign would target the security infrastructure used to protect cloud environments rather than the environments themselves.

The March 2026 Campaign: A Cascade Anatomy

The campaign assigned CVE-2026-33634 unfolded in five distinct phases across eight days, with each phase exploiting credentials or access obtained in the previous one [1][2][3][4][5].

Phase One: The Trivy Breach (March 19, 2026)

Trivy, developed and maintained by Aqua Security, is one of the most widely deployed open-source container and infrastructure vulnerability scanners available [19]. It is integrated into CI/CD pipelines across thousands of organizations and used to scan container images, filesystem artifacts, and infrastructure-as-code configurations for known vulnerabilities, misconfigurations, and exposed secrets. The `trivy-action` GitHub Action and the companion `setup-trivy` action enable Trivy to be run automatically as part of GitHub Actions workflows, typically on every pull request and merge commit.

On March 19, 2026, TeamPCP force-pushed malicious code to 76 of 77 version tags of `trivy-action` and to all 7 version tags of `setup-trivy` [1][2][6]. A force-push in this context overwrites what a given version tag points to in the Git repository without creating a new release or triggering notifications to users who have pinned their workflows to that tag. Organizations that had configured their pipelines to reference, for example, `aquasecurity/trivy-action@v0.30.0` continued to receive what they believed was

the trusted, version-pinned action—but the code behind that tag had been silently replaced with malware. The malicious code executed a legitimate Trivy scan to avoid raising immediate suspicion while simultaneously harvesting all secrets available in the GitHub Actions environment: SSH keys, API tokens, cloud provider credentials, and any other sensitive variables exposed during the workflow run. Harvested credentials were exfiltrated before the scan results were returned.

The attack's design is notable for its invisibility within normal pipeline output. Organizations running Trivy in their CI/CD pipelines were doing so specifically to improve their security posture. The scan would complete normally and return results that looked correct. Developers and security teams reviewing pipeline output would see a vulnerability scan that ran, found or did not find issues, and completed successfully. The exfiltration was invisible within that workflow. SANS Institute researchers analyzing the campaign noted that the most security-conscious organizations—those scanning every pull request and every build—had the greatest exposure, because their pipelines contained the broadest set of credentials and ran scans most frequently [3][14]. The safe versions of the affected actions were Trivy v0.69.3 and earlier, `trivy-action` tag 0.35.0, and `setup-trivy` tag 0.2.6; all tags released after those points were compromised until Aqua Security rotated control and republished clean versions.

Microsoft's Defender Security Research Team published guidance for detecting the Trivy compromise on March 25, 2026, noting that the affected GitHub Actions had been used in pipelines containing AWS API keys, Azure service principal credentials, GCP service account keys, and GitHub personal access tokens [7]. The breadth of credential types reflected Trivy's typical deployment context: organizations use it to scan the artifacts they ultimately deploy into cloud infrastructure, so the pipeline environment where Trivy runs is often richly populated with the credentials needed to deploy and operate that infrastructure.

Phase Two: The npm Cascade (March 20, 2026)

Within twenty-four hours of the Trivy breach, TeamPCP leveraged credentials harvested from affected pipelines to compromise more than fifty npm packages via CanisterWorm's automated credential replay capability [1][5]. The worm's design specifically prioritized npm publishing tokens among harvested secrets, and the automation reportedly cycled through stolen tokens at approximately sixty seconds per package—fast enough to compromise dozens of packages before any individual package owner received an alert or could intervene.

The npm cascade illustrated a second-order dynamic inherent in the reflexive supply chain approach. Because CI/CD pipelines for software projects of all types routinely run Trivy scans, the credentials stolen in Phase One included publishing tokens for npm packages that had no relationship to security scanning. A JavaScript library author whose Trivy pipeline ran on March 19 might find their npm publishing token compromised and their library backdoored on March 20. The security scanner had become a vector into an entirely different software ecosystem.

Phase Three: Checkmarx KICS (March 21, 2026)

On March 21, TeamPCP used GitHub personal access tokens (PATs) obtained through the Trivy breach to compromise Checkmarx's KICS (Keeping Infrastructure as Code Secure) GitHub Actions [1][4]. KICS is a widely used open-source static analysis tool for detecting security misconfigurations in infrastructure-as-code templates, supporting Terraform, CloudFormation, Kubernetes manifests, Dockerfile configurations, and dozens of other formats. Its GitHub Actions– `checkmarx/kics-github-action` and `ast-github-action` –are integrated into CI/CD pipelines in a manner directly analogous to Trivy: automatically scanning infrastructure configurations on every commit.

All 35 version tags of the affected KICS actions were force-pushed with malicious commits [1][4]. The targeting logic is consistent with TeamPCP's methodology: KICS pipelines, like Trivy pipelines, run in highly privileged environments with access to cloud provider credentials, because the infrastructure-as-code files they analyze are the same templates used to provision and modify cloud environments. The organizations most likely to have KICS integrated into their pipelines are also the organizations most likely to maintain mature cloud security programs—and therefore the most likely to have consequential cloud credentials available within the pipeline environment.

Phase Four: The LiteLLM Compromise (March 23–24, 2026)

The most technically sophisticated phase of the campaign targeted LiteLLM, an open-source Python library that functions as a unified API gateway for more than one hundred large language model providers including OpenAI, Anthropic, Google, Cohere, and numerous others [1][4][8]. LiteLLM's core value proposition is simplifying LLM integration: organizations configure their various provider API keys in a single LiteLLM configuration file or environment, and LiteLLM handles routing, retries, fallbacks, and cost tracking across providers. This means that a LiteLLM deployment typically contains a highly concentrated collection of AI provider API keys—credentials that can be used to run inference workloads at the holder's expense.

At the time of the compromise, LiteLLM was estimated to be present in approximately 36% of monitored enterprise cloud environments, according to Wiz research cited across industry coverage of the incident [8], and had accumulated roughly 97 million monthly downloads from PyPI. TeamPCP poisoned LiteLLM versions 1.82.7 and 1.82.8, inserting a `.pth` file persistence mechanism that survived standard package removal—because `.pth` files in a Python environment's site-packages directory are evaluated on every Python interpreter invocation, the malicious code continued executing even after an `uninstall` of the LiteLLM package itself. Trend Micro researchers analyzing the compromise characterized the use of a `.pth` file as deliberately chosen for its persistence properties: removing it required not just uninstalling LiteLLM but auditing and cleaning the entire Python environment's site-packages [8].

The LiteLLM compromise carried a qualitatively different risk profile from the scanner compromises. Stolen API keys for LLM providers can be used to run inference workloads that generate significant unauthorized charges—with incident reports describing costs reaching tens of thousands of dollars within hours—to extract data from AI applications built on those APIs, or to conduct AI-assisted attacks at the credential owner's expense. An organization that had adopted LiteLLM as part of an AI security monitoring stack—using LLMs to analyze logs, flag anomalies, or correlate threat intelligence—would have faced a particular irony: its AI security infrastructure becoming the vector through which credentials for those very AI services were stolen.

The Downstream Cascade: The European Commission Breach

The confirmed highest-consequence downstream impact of the March 2026 campaign was the exfiltration of data from European Commission entities. AWS API keys stolen via the Trivy compromise enabled TeamPCP to access accounts associated with European Commission cloud deployments [1][5][13]. Over five days following the initial breach, the group exfiltrated 91.7 gigabytes of compressed data—equivalent to approximately 340 gigabytes uncompressed—from systems across 71 EU entities before the compromise was detected [1]. By late March 2026, reporting indicated that TeamPCP had launched CipherForce, a ransomware-as-a-service affiliate program, alongside a broader Vect ransomware affiliate network, signaling that the campaign was transitioning from pure credential theft and data exfiltration toward double-extortion ransomware operations [17].

Structural Vulnerabilities Enabling the Reflexive Attack

The March 2026 campaign succeeded not because it discovered novel technical vulnerabilities in the targeted products—the version tag manipulation it exploited was not a software bug but an artifact of how Git tag mutability interacts with CI/CD trust models. The campaign's success derived from structural characteristics of how security tooling is deployed and trusted.

Version Tag Mutability and the Illusion of Pinning

When an organization configures a CI/CD workflow to reference `aquasecurity/trivy-action@v0.30.0`, the intent is to pin the dependency to a known, audited version. In a supply chain security context, pinning is taught as a best practice precisely because it protects against silently updated dependencies. However, Git version tags are mutable references: a repository owner (or an attacker with repository write access) can force-push a tag to point at a different commit without creating a new tag

name or generating notifications to any downstream users [2][6]. The result is that workflows pinned to a named version tag have an implicit trust relationship with whoever controls the repository—not with the specific commit that tag referenced when the workflow was configured.

The most direct protection against the force-push vector itself is pinning dependencies to their full commit SHA rather than to a named version tag. A commit SHA is immutable: if the repository is updated, the SHA still refers to the original commit. Many CI/CD platforms offer tooling to convert tag references to SHA pins, but adoption has not been widespread in practice, in part because SHA pins are opaque to human review and more cumbersome to update when legitimate new versions are released. Other controls—including allowlisting approved actions, eliminating static pipeline secrets via OIDC federation, and maintaining private forks of critical actions—address related attack surfaces and reduce the credential yield even if a compromised action runs.

Trust Inheritance in the CI/CD Environment

CI/CD pipelines are privileged environments by design. A pipeline that builds software, runs tests, scans for vulnerabilities, and deploys to production infrastructure must be able to authenticate to code repositories, artifact registries, cloud providers, and deployment targets. The concentration of credentials in this environment is not a misconfiguration—it is a necessary consequence of what CI/CD pipelines do.

When a security scanning tool is integrated into this environment, it inherits access to all credentials that the pipeline exposes. In the Trivy and KICS compromises, the scanning actions ran in the same GitHub Actions environment as the surrounding pipeline, with full access to all secrets injected into that environment. There was no technical mechanism to restrict a GitHub Action's access to only the secrets it nominally needed—GitHub's secrets model at the time provided scoping at the workflow level but not at the individual action level [6][7].

This architecture means that compromising a security scanner produces a disproportionate return relative to compromising other pipeline components. A tool that claims credentials to deploy an application accesses only the credentials for that deployment target. A tool that scans the entire pipeline environment for security issues often has legitimate reasons to examine all secrets present in that environment. The reflexive supply chain attack exploits this asymmetry: the security scanner's broad visibility becomes the attacker's broad harvest.

Secret Sprawl and Credential Density

The scale of the European Commission exfiltration is consistent with a broader industry pattern observed in multiple threat reports: organizational secrets have proliferated faster than organizations' ability to manage their lifecycle, rotation, and scope. When a CI/CD pipeline harvesting operation can yield AWS API keys

enabling access to 71 institutional cloud environments, it indicates that those keys were long-lived, broadly scoped, and distributed across many pipelines without individual rotation schedules or least-privilege constraints [1][5][7].

This is not a criticism unique to the European Commission. GitGuardian's 2025 State of Secrets Sprawl report documented that 59 percent of compromised machines were CI/CD runners and that 70 percent of leaked secrets remain active two years after their initial exposure, reflecting how broadly organizations accumulate long-lived, overly permissioned credentials across pipeline environments without proportionate lifecycle management [22]. The TeamPCP campaign is partly an indictment of this pattern: the yield from a single scanner compromise should not be sufficient to access 71 separate institutional environments.

The Security-Visibility Blind Spot

A central operational reality of the Trivy compromise was that affected organizations saw nothing anomalous in their pipeline output. The malicious action completed the vulnerability scan as expected and returned results that appeared correct. The exfiltration occurred over a side channel, completing before the scan results were returned. For security teams relying on pipeline logs, scan results, and conventional monitoring to detect supply chain compromise, this attack pattern is effectively invisible: it looks exactly like a successful security control operating as designed.

This represents a fundamental limitation of detection approaches that rely on behavioral deviation from baseline. When the compromised component is the baseline monitoring tool itself, deviation detection cannot function. The adversary's actions are indistinguishable from legitimate scanning activity because they are wrapped in legitimate scanning activity.

The AI/ML Toolchain as a Developing Attack Surface

The LiteLLM compromise in the March 2026 campaign reflects a broader expansion of supply chain attacker interest into the AI and machine learning toolchain. Organizations integrating AI capabilities into their applications and security operations have necessarily expanded their dependency surface, and the early security maturity of many AI tooling components creates a favorable risk calculus for attackers.

The AI Gateway Attack Pattern

AI gateways like LiteLLM occupy a uniquely sensitive position in the AI application stack. They centralize API key management for multiple providers, handle all inference traffic flowing through the application, and typically have access to request and response content for logging, monitoring, and cost attribution purposes. Compromising an AI gateway yields not only the API credentials needed to run inference at the

victim's expense but also the ability to intercept, modify, or log all AI inference traffic passing through the gateway. For organizations using AI in security-sensitive contexts—threat analysis, incident response automation, security data enrichment—this represents access to some of the most sensitive analytical workflows in the enterprise.

The `.pth` file persistence technique used in the LiteLLM compromise is particularly notable. It reflects an attacker capability designed specifically for Python-heavy AI/ML environments, where pip install and pip uninstall are standard operational tools and where security teams may not be familiar with the Python site-packages mechanism's capacity to execute code outside the installed package hierarchy. Trend Micro researchers described the technique as specifically chosen to survive the remediation steps that a developer or security engineer would most naturally take upon learning of a compromised package [8].

Model Context Protocol Servers as Supply Chain Vectors

Beyond the March 2026 campaign, the broader MCP (Model Context Protocol) ecosystem has emerged as a parallel supply chain risk. MCP servers function as extension points for AI agents, providing access to tools, databases, file systems, and external services. The Kaspersky GERT team documented a proof-of-concept malicious MCP server in September 2025 that, while presented as a legitimate developer tool through PyPI, systematically exfiltrated environment files, SSH keys, and AWS credentials via API calls disguised as GitHub traffic [15]. Check Point Research disclosed a remote code execution vulnerability in Claude Code via poisoned repository configuration files in February 2026 [16], and Trend Micro found 492 MCP servers exposed to the internet with no authentication required [21].

The MCP ecosystem exhibits the same structural conditions that enabled the TeamPCP campaign. MCP servers are extended trust relationships—an AI agent using an MCP server implicitly trusts that the server's tools will behave as advertised. An MCP server that presents as a legitimate capability but harvests credentials or manipulates agent behavior has the same reflexive character as a security scanner that looks correct while exfiltrating secrets. As MCP adoption accelerates, the security community should expect adversaries to invest in malicious MCP server distribution, particularly through registries and marketplaces where discovery friction is low and vetting processes are immature.

Developer AI Assistants as Reconnaissance Tools

The August 2025 NX build system compromise offered an early demonstration of AI coding assistants being turned against the organizations using them. The malware deployed in that compromise specifically targeted Claude, Gemini, and Google's q CLI, using their filesystem access and API connectivity to conduct reconnaissance and exfiltrate data [11]. This technique exploits the broad permissions that AI coding tools typically receive—access to source code, configuration files, and environment variables across a project directory—and the implicit trust developers extend to outputs from those tools. As AI coding assistants

accumulate more persistent context about development environments and broader filesystem access, they will become increasingly attractive targets for supply chain attackers seeking reconnaissance or exfiltration capabilities.

Recommendations

Immediate Actions

Organizations that use Trivy, Checkmarx KICS, or LiteLLM in their environments should begin remediation with an immediate audit of which versions of these tools are present in their CI/CD pipelines and Python environments. For GitHub Actions pipelines, workflows should be reviewed to identify any references to `aquasecurity/trivy-action`, `setup-trivy`, `checkmarx/kics-github-action`, or `ast-github-action`, and each reference should be cross-referenced against the known safe version thresholds: Trivy v0.69.3 or earlier, `trivy-action` tag 0.35.0 or earlier, and `setup-trivy` tag 0.2.6 or earlier [1][6]. Any pipeline that ran a vulnerable version of these actions between March 19 and March 27, 2026, should be treated as having had its full secrets inventory exposed, and all secrets available in those pipeline environments should be rotated immediately.

For LiteLLM specifically, organizations should not rely on package uninstallation as a remediation step. Because the compromise used a `.pth` file persistence mechanism, LiteLLM versions 1.82.7 and 1.82.8 may have left behind persistent code that executes on Python interpreter invocation even after the package is removed [8]. Affected Python environments should be fully audited for unexpected entries in site-packages directories, and all LLM provider API keys configured in LiteLLM should be rotated and reissued with minimum-necessary scope.

Beyond the specific TeamPCP remediation, organizations should immediately replace all GitHub Actions references to named version tags with commit SHA pins. GitHub provides documentation for converting tag references to SHA references, and tools such as StepSecurity's Harden-Runner and similar open-source utilities can automate this conversion across a repository's workflow files. This change eliminates the force-push attack vector that enabled the Trivy and KICS compromises. It does not eliminate all supply chain risks in GitHub Actions, but it closes the specific mechanism TeamPCP exploited in the March 2026 campaign.

Short-Term Mitigations

Over the following thirty to ninety days, organizations should implement structural controls that reduce the credential density available within any individual pipeline execution. The least-privilege principle applies to CI/CD environments as it does to any other access context: a pipeline that runs a vulnerability scan should

not, as a general practice, have access to credentials for unrelated production deployments. Separating scanning pipelines from deployment pipelines, using distinct service accounts with narrowly scoped permissions for each stage, and implementing just-in-time credential provisioning through tools like HashiCorp Vault or cloud-native secrets management services substantially reduces the yield available to a compromised scanner action.

Organizations should also implement monitoring for anomalous credential usage that might indicate post-compromise activity. While detecting exfiltration during the initial scanner execution is difficult, many attacks that follow from stolen credentials have detectable signatures: unusual geographic origins, access to resources outside normal operational patterns, API call volumes inconsistent with legitimate use. Integrating stolen-credential detection into cloud provider activity monitoring—through AWS GuardDuty, Azure Defender for Cloud, or GCP Security Command Center—provides a second detection layer that does not depend on observing the initial compromise.

For AI/ML tooling specifically, organizations should maintain a software bill of materials for their AI application dependencies with the same rigor applied to other production software. LiteLLM's presence in 36% of monitored enterprise cloud environments at the time of its compromise suggests many organizations were running it without formal inventory tracking [8]. An untracked dependency cannot be patched promptly or audited systematically when a compromise is discovered.

Strategic Considerations

Addressing the reflexive supply chain threat at a strategic level requires reconsidering the trust model applied to security tooling. The organizational intuition that security tools can be trusted because they are security tools is precisely the assumption that TeamPCP exploited. Security scanning tools, static analysis frameworks, AI gateways, and development tooling all represent supply chain dependencies that require the same scrutiny as application dependencies—including assessment of the trust chain behind those tools, the security practices of their maintainers, and the privileges they receive in the environments where they operate.

Vendor-managed and commercially supported security tools may offer additional supply chain integrity assurances through commercial code signing and distribution controls, but these are not reliable guarantees of safety. The CrowdStrike-adjacent npm package compromises in 2025 demonstrate that even tooling from major security vendors can be caught in supply chain cascades when development workflow components are involved [10]. The relevant question is not whether a tool is commercial or open-source but whether the integrity of its delivery pipeline can be verified at installation time.

Longer term, the security community needs to develop and adopt artifact provenance standards specifically tailored to the CI/CD and security tooling context. SLSA (Supply Chain Levels for Software Artifacts) provides a framework for assessing and improving the integrity of the build and delivery pipeline, but

adoption among security tool maintainers remains a work in progress. Organizations should prioritize security tools whose maintainers have achieved or committed to SLSA Level 3 compliance, require signed artifacts for all security tool installations, and establish processes for verifying signature chains against trusted root authorities rather than relying on repository-level trust.

CSA Resource Alignment

The reflexive supply chain threat described in this whitepaper intersects with several established CSA frameworks and guidance publications, and organizations responding to this class of threat should treat those frameworks as complementary to the technical recommendations above.

The CSA AI Controls Matrix (AICM) addresses AI supply chain security explicitly across multiple control domains, including controls governing the vetting of AI model providers, the security of AI application dependencies, and the governance of AI tooling integrated into development and operational workflows. The LiteLLM compromise is an instance of the AI gateway dependency risk that AICM's Application Provider implementation guidelines address: organizations building on LLM APIs have a shared responsibility to validate the integrity of the libraries mediating that access. AICM's Orchestrated Service Provider guidelines similarly address the security of tooling that sits between AI consumers and AI providers.

CSA's MAESTRO framework for agentic AI threat modeling provides a threat taxonomy directly applicable to the MCP server supply chain risk described in this whitepaper. MAESTRO's analysis of trust relationships in agentic systems—specifically, the trust an AI agent extends to its tools and the tools' capacity to manipulate agent behavior—frames the malicious MCP server threat in terms that security architects can use to evaluate specific deployment configurations. Organizations deploying AI agents with MCP tool access should apply MAESTRO's trust boundary analysis to assess which MCP servers have access to credentials, file systems, or network resources that would be valuable to an attacker.

The CSA Software Transparency: Securing the Digital Supply Chain publication provides complementary guidance on SBOM practices, CI/CD pipeline security, and the organizational processes needed to maintain software supply chain integrity. Its recommendations on CI/CD security controls—including access management for pipeline environments, secrets management practices, and verification of build artifacts—directly address the structural conditions that enabled the March 2026 campaign.

Finally, CSA's Zero Trust guidance is directly relevant to the CI/CD trust model reform recommended in this whitepaper. Applying Zero Trust principles to pipeline environments means treating the security tooling integrated into those pipelines as untrusted by default—requiring verified identity, least-privilege credential access, and continuous monitoring rather than extending implicit trust based on the tool's security-oriented purpose. This inversion of the standard assumption—that security tools can be trusted more than other software—is the most important conceptual shift the reflexive supply chain threat demands.

Conclusions

The TeamPCP March 2026 campaign represents a maturation of adversarial supply chain technique. By targeting the security and AI tooling that organizations deploy to protect their software supply chains, the campaign exploited not a technical vulnerability in those tools' code but a structural vulnerability in the trust model that organizations apply to them. The yield from a single scanner compromise—credentials enabling access to 71 European Commission entities—reflects how privileged the CI/CD environment has become and how broadly organizations have distributed consequential secrets across that environment without proportionate attention to the integrity of the tools running there.

The reflexive supply chain threat will not diminish as organizations respond to the March 2026 campaign. CanisterWorm's ICP blockchain command-and-control infrastructure, TeamPCP's launch of the CipherForce ransomware-as-a-service program, and the group's demonstrated capacity to pivot from initial access to cascading credential theft and ransomware preparation indicate a threat actor with both the means and the motivation to continue developing this attack pattern. The AI/ML toolchain provides an expanding attack surface: LiteLLM's presence in over a third of monitored enterprise environments at the time of its compromise reflects how rapidly AI infrastructure has accumulated in enterprise environments faster than supply chain scrutiny has followed.

The core recommendation of this whitepaper is straightforward in principle if challenging in execution: treat security tools, AI tools, and development infrastructure as first-class supply chain dependencies subject to the same verification, least-privilege, and integrity controls applied to the application software those tools protect. The organizations that will be most resilient to future TeamPCP campaigns are those that do not exempt their security stack from supply chain scrutiny simply because it is a security stack. In a threat environment where defenders' tools have become a preferred attack target, the consistency of supply chain discipline—applied to all software, without exception—is the most durable protection available.

References

[1] Palo Alto Unit 42. "Weaponizing the Protectors: TeamPCP's Multi-Stage Supply Chain Attack on Security Infrastructure." Palo Alto Networks, March 31, 2026.

[2] Arctic Wolf. "TeamPCP Supply Chain Attack Campaign Targets Trivy, Checkmarx (KICS), and LiteLLM." Arctic Wolf Networks, March 25, 2026.

[3] SANS Institute. "When the Security Scanner Became the Weapon: Inside the TeamPCP Supply Chain Campaign." SANS Institute, March 25, 2026.

[4] Cato Networks. "TeamPCP Supply Chain Attack: Trivy, KICS, LiteLLM." Cato Networks, 2026.

[5] MrCloudBook. "TeamPCP Supply Chain Attack — Telnyx, CanisterWorm Full Analysis (CVE-2026-33634)." MrCloudBook, 2026.

[6] Snyk. "Trivy GitHub Actions Supply Chain Compromise." Snyk Security Research, March 20, 2026.

[7] Microsoft Defender Security Research Team. "Guidance for detecting, investigating, and defending against the Trivy supply chain compromise." Microsoft Security Blog, March 25, 2026.

[8] Trend Micro. "Your AI Gateway Was a Backdoor: Inside the LiteLLM Supply Chain Compromise." Trend Micro Research, 2026.

[9] Semgrep. "Security Advisory: NPM Packages Using Secret Scanning Tools to Steal Credentials." Semgrep, 2025.

[10] Dark Reading. "Supply Chain Worms 2026: What Shai-Hulud Taught Attackers and How to Prepare." Dark Reading, 2026.

[11] StepSecurity. "Supply Chain Security Alert: Popular Nx Build System Package Compromised with Data-Stealing Malware." StepSecurity, August 2025.

[12] Kaspersky. "Trojanization of Trivy, Checkmarx, and LiteLLM solutions." Kaspersky Blog, 2026.

[13] Flare Research. "Threat Alert: TeamPCP, An Emerging Force in the Cloud Native and Ransomware Landscape." Flare, February 5, 2026.

[14] IANS Research. "Trivy Supply Chain Attack Triggers Self-Propagating CI/CD Compromise." IANS Research, March 31, 2026.

[15] Kaspersky Securelist. "Shiny tools, shallow checks: how the AI hype opens the door to malicious MCP servers." Kaspersky Securelist, September 15, 2025.

[16] Check Point Research. "Caught in the Hook: RCE and API Token Exfiltration Through Claude Code Project Files." Check Point Research, February 2026.

[17] Help Net Security. "TeamPCP's attack spree slows, but threat escalates with ransomware pivot." Help Net Security, March 30, 2026.

[18] Tenable. "CVE-2026-33634." Tenable Research, 2026.

[19] InfoQ. "Open Source Security Tool Trivy Hit by Supply Chain Attack." InfoQ, April 2026.

[20] Cyble. "TeamPCP Threat Actor Profile." Cyble Threat Intelligence, 2026.

[21] Trend Micro. "MCP Security: Network-Exposed Servers Are Backdoors to Your Private Data." Trend Micro, 2025.

[22] GitGuardian. "State of Secrets Sprawl 2025." GitGuardian, 2025.