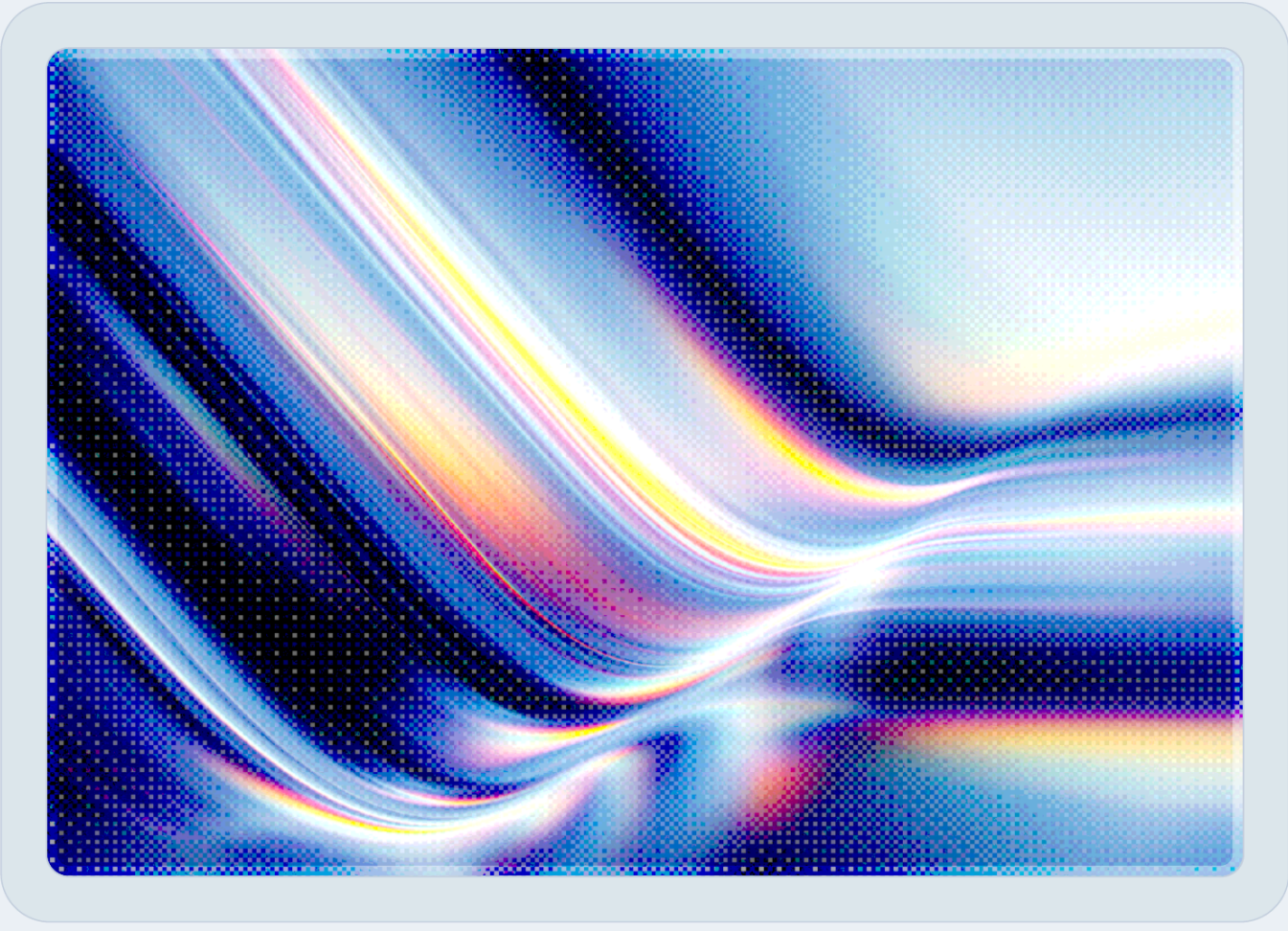
CSAI Foundation | Cloud Security Alliance

Project Granite: SAP Source-Code Incident and Vendor Code Integrity

Security Implications and Guidance for ERP Customers

2026-11-29

AI-assisted Rapid Research



CSAI

CSA 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

SAP confirmed on November 27, 2026 that it is investigating a security incident. According to press reports, attackers used stolen authentication tokens belonging to SAP employees to reach the company's GitHub environment and copy and alter source code there [1][2][3]. SAP reportedly runs the investigation internally under the name "Project Granite" [2]. SAP has said only that its investigation "to date" has found "no evidence of unauthorized access to customer business data" [1]. As of this writing, it has not said which repositories were affected, whether proprietary code was accessed, or whether any altered code reached a shipped build [1][2].

Most of the operational detail has not been confirmed by SAP. Handelsblatt reported that SAP replaced signing keys, limited access to GitHub, and suspended automated development operations [1][4]. Sources cited by Der Spiegel suggested the attackers may have had help from AI systems, which Cybersecurity News described as AI agents, but neither SAP nor any other source reviewed for this note has confirmed this [1][2][3]. No threat actor has been publicly named, and the way the tokens were stolen has not been disclosed [2].

For ERP customers, CSA assesses that the more consequential question is code integrity: whether software and tooling received from SAP is exactly what SAP intended to ship. Confidentiality still matters, because copied source code may speed up vulnerability discovery, but SAP has stated that its investigation to date found no evidence of access to customer business data [1]. The reported rotation of signing keys means verification must anchor on the new keys and not trust artifacts signed with superseded material without question. Customers should inventory every point where SAP-originated code enters their environment: transports, support packages, SAP Notes, BTP and CAP libraries, npm and Maven packages, and code cloned from SAP's public GitHub organizations.

This is the second code supply-chain event at SAP in 2026. In April, four official SAP npm packages, three in the Cloud Application Programming (`@cap-js`) namespace plus the `mbt` build tool, were poisoned with a credential-stealing payload in the "Mini Shai-Hulud" campaign [5][6]. Taken together, the two incidents show that SAP's developer credentials and source-hosting environment have been targeted twice in 2026. We recommend that customer controls be designed on the assumption that such incidents will recur.

Background

SAP is among the most deeply embedded enterprise software vendors. Its ERP, finance, supply-chain, and HR systems often sit at the center of an organization's regulated business processes. Because SAP systems are so heavily customized, SAP-originated code reaches customers through many channels. These include the classic ABAP transport and SAP Note correction mechanisms, cloud services on SAP Business Technology Platform (BTP), and open-source libraries, SDKs, and sample applications that SAP publishes on GitHub and package registries such as npm. Each channel carries its own integrity assumptions and its own verification controls.

The current incident first became public on Thursday, November 26, 2026, when Der Spiegel reported, citing SAP insiders and an internal notice to employees, that the company was dealing with an active security incident [1][3]. The German news agency DPA separately reported seeing internal SAP communications that located the suspicious activity within GitHub [1][7]. In a statement to SecurityWeek, SAP said it had identified suspicious activity, implemented measures to protect customers and secure its services, and found no evidence of unauthorized access to customer business data [1]. The dpa report described these measures as having been taken immediately [7]. Cybersecurity News reported that SAP declined to answer detailed questions [2].

Several facts that matter for customer risk are still unknown. SAP maintains public GitHub organizations for developer tools, sample applications, and open-source projects. SecurityWeek noted that it is unclear whether any proprietary, non-public code was accessed, and that none of the reports indicate tampered code reached software delivered to customers [1]. Cybersecurity News observed that SAP's statement addresses customer business data, which is a different question from whether unauthorized code changes reached customer-facing software [2]. Both questions matter, but the second one determines what customers need to verify.

Two other events provide context. In late October 2026, the Gentlemen ransomware group listed SAP on its leak site, referencing SAP's Latin American operations. SecurityWeek reported that the incidents "do not appear to be connected" [1]. Separately, on April 29, 2026, attackers attributed by Wiz and Socket to the TeamPCP group published malicious versions of `@cap-js/sqlite`, `@cap-js/postgres`, `@cap-js/db-service`, and `mbt` to npm. These versions carried a preinstall hook that harvested developer and CI/CD credentials, including GitHub and npm tokens and cloud secrets [5][6]. Aikido reported that SAP was likely compromised through an npm token exposed to pull-request builds in its CircleCI pipeline [5]. SAP published an FAQ and SAP Note 3747787 listing affected and clean package versions [8]. Nothing public links the April campaign to Project Granite. The April incident is nonetheless relevant as an illustration of how harvested developer tokens can extend the reach of a single compromise, and customers should confirm their April remediation is complete regardless of any link.

Security Analysis

Token Compromise as the Entry Point

The reported entry point was stolen employee authentication tokens, not an exploited vulnerability [1][2]. Token theft is a recurring intrusion technique. Personal access tokens, OAuth tokens, and CI/CD secrets are bearer credentials. Whoever holds one can act with its full privileges, usually without a second authentication factor and often from any network location. A token issued to a developer with write access to many repositories gives an attacker that same reach, and a long-lived token remains useful until someone notices and revokes it. CSA's work on non-human identity has noted that such credentials are often over-scoped, under-inventoried, and rarely rotated, which makes them attractive targets [15].

GitHub's audit log can tie actions to the specific token that performed them, which lets a defender reconstruct what a compromised credential touched [9]. That capability matters most to SAP's own responders. It also suggests what customers should ask for: an evidence-based statement of which repositories and branches the compromised tokens could write to, and which they actually wrote to. That is far more useful to a customer than a general assurance about customer data.

Why Source Alteration Is the Higher-Order Risk

Copying source code is a confidentiality loss. It can speed up vulnerability discovery against SAP products, because attackers can study code paths offline, but it does not directly put malicious code into customer systems. Altering source code is an integrity event. Its impact depends on whether altered code passed through build, signing, and release pipelines into artifacts that customers consume. The 2020 SolarWinds SUNBURST compromise showed how a build-pipeline intrusion at a trusted vendor can deliver signed, legitimate-looking updates to thousands of downstream organizations [17]. Nothing reported to date suggests that happened here, and readers should not infer that it did [1][2]. Even so, it is the scenario that customer controls must be able to rule out.

SAP has not explained the rationale for its reported response actions, but each is consistent with this concern. Pausing automated development operations stops possibly tainted commits from flowing into builds. Restricting GitHub access limits further use of stolen or unknown credentials. Replacing signing keys guards against the possibility that signing material, or pipelines able to invoke it, were exposed [1][4], although key rotation is a common precautionary measure that does not by itself indicate key compromise. Rotation has a practical consequence for customers. Verification tooling and trust stores that pin SAP's previous keys will need updating once SAP publishes the new ones, and customers should be cautious about artifacts signed close to the incident window until SAP describes the scope.

The Reported AI-Agent Dimension

Der Spiegel's sources suggested the attackers may have been assisted by AI systems, reported by Cybersecurity News as AI agents, which SAP has not confirmed [1][2][3]. If accurate, the most plausible implication is speed and scale. An agent operating with a stolen token can enumerate repositories, read large codebases, find build and release configuration, and make targeted, plausible-looking edits faster than a human operator could. This would shorten the time between credential theft and meaningful impact, which raises the value of detection controls that act within minutes. Without confirmation, this remains an inference, and defenders should treat it as a planning assumption, not an established fact about Project Granite.

Distribution Channels and Customer Exposure

Customer exposure depends on which SAP channels an organization consumes. The table below summarizes the main channels, how each could in principle be affected by source tampering at the vendor, and the verification controls available to customers. It is a planning aid, not a statement about what was affected in this incident.

Distribution channel	Potential exposure from vendor-side source tampering	Customer-side verification controls
Open-source libraries and tools on npm, Maven, PyPI (e.g., CAP, `mbt`)	Malicious release published through a compromised pipeline or maintainer credential, as in April 2026 [5]	Lockfiles with integrity hashes, version pinning, provenance attestation checks, disabling install-time scripts in CI
Code cloned or forked from SAP public GitHub organizations	Tampered commits in samples, SDKs, or tooling pulled directly into customer repositories	Pin to reviewed commit SHAs, review diffs against prior known-good versions, signed-commit checks
SAP Notes, support packages, and ABAP transports	Altered corrections delivered through standard maintenance channels	Signature verification of downloaded archives, transport review before import, change monitoring in production
BTP and SaaS services operated by SAP	Altered code deployed by SAP into its own managed services	Reliance on vendor assurance and attestations, tenant-level

Distribution channel	Potential exposure from vendor-side source tampering	Customer-side verification controls
		monitoring of anomalous behavior and data egress
Developer tooling and IDE extensions from SAP	Compromised tooling running on developer workstations with access to customer credentials	Extension allowlisting, workstation EDR, short-lived developer credentials

For managed cloud services, customers have little direct means of verifying code integrity and must depend on vendor transparency and independent assurance. For artifacts customers download and run themselves, much stronger technical verification is possible. Frameworks such as SLSA [10] and the NIST Secure Software Development Framework [11] describe how build provenance and artifact signing let consumers check that a binary came from a specific source revision through a specific build process. Signing ecosystems such as Sigstore [12] make that verification practical for open-source packages.

Recommendations

Immediate Actions

Organizations running SAP should first establish an inventory of SAP-originated code entering their environment. Because SAP has not disclosed when the unauthorized access began, that inventory should cover a conservative look-back window, at minimum the past several months and ideally back to the April 2026 npm incident, and can be narrowed once SAP publishes a timeline. The inventory includes recently imported transports and SAP Notes, npm and Maven dependencies under SAP namespaces, and any code cloned from SAP GitHub organizations. That inventory determines the scope of any later verification once SAP discloses which repositories were affected. Teams should hold non-urgent imports of SAP corrections and avoid floating dependency versions for SAP open-source packages until SAP clarifies scope. Security-critical patches should still be applied after verification, because delaying them creates its own risk.

Second, customers should rotate any credentials shared with SAP-operated GitHub environments or SAP-hosted developer tooling. This includes tokens used by integrations and CI jobs that pull from SAP repositories. As a precaution, organizations may also review and rotate support access granted to SAP personnel, although nothing reported to date indicates that customer-side support accounts were involved. Third, teams should confirm that remediation from the April 2026 npm incident is complete by

checking dependency trees against SAP Note 3747787 and searching repositories for unexpected workflow branches, as SAP's FAQ advised [8]. Any developer token exposed in April and still valid is a live risk.

Short-Term Mitigations

Over the following weeks, customers should press SAP, through account teams and formal vendor-risk channels, for specific disclosures. These are the affected repositories, whether those repositories feed shipped products or customer-installable artifacts, whether any released build contained unauthorized changes, and the new signing key fingerprints with guidance on how to validate artifacts signed before and after rotation. These questions map directly to customer action. Without answers, organizations cannot close their own assessments.

In parallel, organizations should harden their own consumption of vendor code in ways that will outlast this incident. Lockfiles with integrity hashes, pinning to reviewed versions or commit SHAs, and disabling install-time lifecycle scripts in CI all reduce exposure to a poisoned package release. Where vendors publish provenance attestations, pipelines should verify them before admitting artifacts. For ABAP environments, transport review and change monitoring in production systems give a second line of detection if altered vendor code is ever imported. Organizations should also review their own source-hosting posture, since the same token-theft technique applies to them. Useful measures include replacing long-lived personal access tokens with fine-grained, short-lived credentials, enforcing SSO-bound tokens, and alerting on unusual clone volume or token use from new locations [9].

Strategic Considerations

In CSA's view, Project Granite highlights a gap in how enterprises assess critical software vendors. Many third-party risk programs focus on how a vendor protects customer data and give less weight to how a vendor protects the integrity of the code customers run, even though for ERP platforms that code effectively becomes part of the customer's own control environment. Assessments of strategic vendors should explicitly cover source-hosting access controls, developer credential lifecycle, build-pipeline isolation, signing key custody, provenance publication, and incident disclosure commitments regarding code integrity. Contracts can reasonably require timely notification when unauthorized changes to customer-delivered code are suspected, separate from data-breach notification clauses.

Organizations should also plan for AI-accelerated intrusion as a general condition, whether or not the AI-agent element of this incident is confirmed. If attackers can move from a stolen token to meaningful source manipulation within minutes, controls based on periodic review are likely to be too slow.

Continuous verification at the point of consumption, combined with short credential lifetimes on both vendor and customer sides, is a more durable stance than relying on after-the-fact vendor statements.

CSA Resource Alignment

CSA's research supports both the vendor-assessment and the technical-control parts of this guidance. The **AI Controls Matrix (AICM) v1.1** [13] provides the control framework against which customers can structure assessments of SAP and similar vendors. Its Supply Chain Management, Transparency, and Accountability (STA) domain covers third-party assurance and supply-chain risk, its Application and Interface Security (AIS) domain addresses secure development and build integrity, and its Identity and Access Management (IAM) domain covers the credential governance controls most relevant to the reported entry point, stolen developer tokens. Customers can use AICM-aligned questions to turn the disclosure requests in this note into a repeatable assessment instead of a one-off inquiry.

The **STAR program** [14] provides a public mechanism for vendors to document their control posture through self-assessment and third-party certification. Following Project Granite, customers reviewing STAR entries for critical software suppliers should look specifically at how source-code and build-pipeline integrity controls are described, and may request that vendors address code-integrity assurance explicitly in future submissions.

CSA's **Defining Non-Human Identity** [15] is directly relevant to the reported entry point. Developer and automation tokens are non-human credentials whose scope, lifetime, and ownership are often poorly governed. The publication's emphasis on moving from static credentials to short-lived, ephemeral authentication is a key structural mitigation for token-theft intrusions of the kind reported at SAP, and it applies equally to customers' own source-hosting and CI environments. **The Six Pillars of DevSecOps: Automation** [16] complements this with guidance on embedding software composition analysis, secrets management, and automated verification into build pipelines. These are the controls customers need to verify vendor artifacts at the point of consumption.

References

[1] Kovacs, Eduard. "SAP Confirms Cyberattack, Source Code Reportedly Compromised." SecurityWeek, November 27, 2026.

[2] Baran, Guru. "SAP Confirms Security Incident as Hackers Reportedly Steal GitHub Tokens and Alter Code." Cybersecurity News, November 27, 2026.

[3] Rosenbach, Marcel. "SAP wird Opfer von Hackerangriff." Der Spiegel, November 26, 2026.

[4] Handelsblatt. "SAP: IT-Sicherheitsvorfall – Kunden fordern 'lückenlose' Aufklärung." Handelsblatt, November 26, 2026.

[5] Arghire, Ionut. "SAP npm Packages Targeted in Supply Chain Attack." SecurityWeek, April 30, 2026.

[6] The Register. "Ongoing supply chain attacks worm into SAP npm packages." The Register, April 30, 2026.

[7] Süddeutsche Zeitung / dpa. "Softwarekonzern SAP: 'Haben verdächtige Aktivitäten identifiziert'." Süddeutsche Zeitung, November 26, 2026.

[8] SAP. "FAQ: Mini Shai-Hulud npm Developer Package." SAP, May 2026.

[9] GitHub. "Identifying audit log events performed by an access token." GitHub Docs, accessed November 2026.

[10] OpenSSF. "SLSA Specification v1.0." Supply-chain Levels for Software Artifacts, April 2023.

[11] NIST. "SP 800-218: Secure Software Development Framework (SSDF) Version 1.1." National Institute of Standards and Technology, February 2022.

[12] Sigstore. "Sigstore: A new standard for signing, verifying and protecting software." Open Source Security Foundation, accessed November 2026.

[13] Cloud Security Alliance. "AI Controls Matrix (AICM) v1.1." Cloud Security Alliance, 2026.

[14] Cloud Security Alliance. "Security, Trust, Assurance and Risk (STAR) Program." Cloud Security Alliance, accessed November 2026.

[15] Cloud Security Alliance. "Defining Non-Human Identity (NHI)." Cloud Security Alliance, 2026.

© 2026 Cloud Security Alliance. AI-assisted Rapid Research.

[16] Cloud Security Alliance. "The Six Pillars of DevSecOps: Automation." Cloud Security Alliance, 2020.

[17] CISA. "Advanced Persistent Threat Compromise of Government Agencies, Critical Infrastructure, and Private Sector Organizations (AA20-352A)." Cybersecurity and Infrastructure Security Agency, December 17, 2020.