unosecur

WHITEPAPER

Identity security in developer platforms

Why identity is the common thread through modern software supply chain attacks, and what a modern defense looks like.

Table of contents

Executive summary 02

Section 1: The developer platform identity landscape 03

Section 2: How identity is being attacked 04

Section 3: The emerging frontier: AI agents as developer identities 06

Section 4: Why this is hard to solve 08

Section 5: What an identity-first defense looks like 09

Conclusion 12

Executive summary

Developer platforms have become the software control plane of the modern enterprise. Code is written, reviewed, built, tested, deployed, and increasingly operated by AI inside them. Source code, package registries, CI/CD pipelines, managed databases, cloud environments, SaaS tools, and customer-facing applications now share one operating layer.

That operating layer is dense with identity. A developer account can read source code, approve changes, trigger deployments, access logs, manage databases, and connect third-party tools. A CI/CD runner can pull secrets, assume cloud roles, publish artifacts, and deploy to production. A package maintainer token can distribute code to thousands of downstream environments. An AI coding agent can inspect repositories, call tools, run commands, and interact with cloud services. Attackers have followed the identity. They are targeting the credentials, tokens, agents, and service accounts that create, move, publish, and operate code, because those are the access paths to everything that matters downstream.

In 2025, GitGuardian detected 28,649,024 new secrets in public source-code commits, a 34 percent year-on-year increase. AI-assisted commits leaked secrets at roughly twice the baseline rate across public repositories. Sonatype identified more than 454,600 new malicious open-source packages, bringing the total of known and blocked malware above 1.233 million packages. GitHub reported more than 1 million coding-agent pull requests between May and September 2025, with coding-agent activity skewing toward established repositories.

The pattern across every major incident is consistent. Identity is the attack surface. Stolen credentials, overprivileged CI/CD runners, compromised maintainer tokens, leaked secrets, and ungoverned AI agent access enable the attacker to move from initial compromise to downstream impact across thousands of dependent environments.

Metric	Figure
New secrets detected in public source-code commits in 2025	28.65M
Year-on-year growth in public secret exposure	34%
New malicious open-source packages identified in 2025	454,600+
Coding-agent pull requests created in five months in 2025	1M+

This paper maps the identity threat landscape specific to developer platforms, examines the incidents that define it, and shows what an identity-first security posture looks like across source code, CI/CD, package registries, cloud, SaaS, and AI environments.

Section 1: The developer platform identity landscape

Most enterprises have complex identity environments. Developer platforms have a uniquely concentrated one. Three identity types coexist in every modern software organization, each with a different risk profile.

Human identities

Developers, contractors, maintainers, platform engineers, DevOps teams, security engineers, data engineers, and external contributors all operate close to the production path. They access code, review changes, trigger builds, approve releases, manage secrets, inspect logs, and connect external tools. A compromised developer identity rarely stays confined to one repository. It reaches packages, pipelines, cloud roles, and connected SaaS in the same hour.

Non-human identities (NHIs)

Non-human identities make up the largest and least-governed population inside the software factory. They include API keys, service accounts, webhook tokens, deploy keys, OAuth apps, CI/CD runners, cloud roles, database credentials, bot accounts, package publishing tokens, and automation credentials.
They multiply continuously as teams automate the software lifecycle. Most persist long after the workflow they served, with unclear ownership, infrequent rotation, and a scope that is rarely tightened once a deployment works.

AI Agents

AI agents are the newest and least-governed identity class. They generate code, inspect repositories, summarize pull requests, invoke tools, query documentation, write tests, open tickets, run terminal commands, and assist with infrastructure changes.
An AI agent interprets instructions, selects tools, chains actions, and operates across systems. Unosecur describes agentic AI as "an identity expansion event," and that framing applies directly to developer platforms. AI agents add new actors that hold credentials, inherit permissions, and make decisions inside the production path.

Developer platforms carry an identity governance problem inside the software factory. The convergence of human users, non-human identities, and AI agents across source code, CI/CD, package registries, cloud services, SaaS tools, internal systems, and data stores creates an identity perimeter that most organizations have only partially mapped.

Section 2: How identity is being attacked

The following incidents represent recurring patterns of identity exploitation across developer platforms. Each reflects a current pattern in the software supply chain.

1. **2025 | tj-actions/changed-files supply chain compromise**
   Pattern: CI/CD identity abuse. A trusted GitHub Action was compromised, exposing CI/CD secrets across more than 23,000 repositories through workflow logs.

   In March 2025, attackers compromised the widely used tj-actions/changed-files GitHub Action, retroactively modifying multiple version tags to point to a malicious commit. The injected code caused CI/CD secrets, including cloud credentials, signing keys, and deployment tokens, to be printed into GitHub Actions workflow logs. The advisory affected more than 23,000 repositories, many of which had inherited the malicious version through pinned tags.
   CI/CD runners are privileged non-human identities. They carry tokens, cloud credentials, signing keys, and deployment permissions. When a trusted CI component is compromised, the attacker reaches the secrets and cloud access of every repository that uses it, with the build environment providing the access.
   Developer platform identity extends beyond developer login security. It covers build systems, actions, runners, and workflow tokens.

2. **2025 | Nx s1ngularity attack**
   Pattern: Build workflow compromise. A CI injection vulnerability led to the theft of an npm publishing token and the distribution of malicious Nx package versions through the registry.

   In August 2025, malicious versions of Nx, a widely used JavaScript build system, were published to npm. The official Nx postmortem confirmed that attackers exploited a CI injection vulnerability, stole the npm publishing token, and pushed malicious package versions for approximately four hours. The malicious code scanned user systems for sensitive data, including SSH keys, npm tokens, and cryptocurrency wallet files, and uploaded harvested data to public GitHub repositories created on the victims' own accounts.
   The attack chained multiple identity layers in a single sequence: CI workflow, publishing token, package registry, developer machines, and source code repositories. The stolen publishing identity became the distribution mechanism, carrying the attack outward into every downstream environment that trusted the package.

3. **2025 | Shai-Hulud self-propagating npm worm**
   Pattern: Compromised maintainer identity. A self-propagating worm spread across the npm ecosystem by authenticating as a compromised developer and injecting malicious code into other packages.

   In September 2025, CISA warned of a widespread compromise of the npm ecosystem that used an automated process to authenticate with the registry as a compromised developer and inject code into other packages owned by that developer. CERT/CC described the same campaign as a self-propagating malware variant that spread through credential theft and automated package publishing, ultimately compromising hundreds of npm packages with downstream reach across countless dependent projects.
   This is the developer platform threat model at ecosystem scale. A maintainer identity is compromised, tokens are harvested, packages are altered, and the worm propagates into downstream environments that trust the publisher. In this pattern, identity is the payload carrier.

4. **2026 | Secrets sprawl as the persistent condition**
   Pattern: Long-lived credentials scattered across code, collaboration systems, developer machines, and CI/CD runners.

   Secret sprawl is the condition that makes developer platform attacks repeatable. GitGuardian's State of Secrets Sprawl 2026 detected 28,649,024 new secrets in public source-code commits and 1,275,105 exposed AI-service secrets, an 81 percent year-on-year increase for AI-related service leaks.
   The same report found that 59 percent of compromised machines analyzed in the Shai-Hulud 2 supply chain attack were CI/CD runners rather than personal developer workstations. Secrets now leak from shared build infrastructure, automation environments, collaboration tools, and agentic development workflows, well beyond careless commits. The compromise surface lives inside the platforms organizations rely on to ship software.

5. **2026 | AI-assisted development as an identity accelerant**
   Pattern: AI tools increase software velocity while introducing new credential and governance paths.

   AI-assisted development is part of normal engineering activity. GitHub's 2025 Octoverse report found that more than 1.1 million public repositories use an LLM SDK and that nearly 80 percent of new developers on GitHub use Copilot within their first week. GitGuardian reported that AI-assisted commits leak secrets at roughly twice the baseline rate, with credentials for AI services leaking at a faster rate than any other secret category.

AI changes the rate at which identities, credentials, integrations, and tool connections appear inside developer workflows. The software factory now operates as a privilege graph in which human users, automation tokens, publishing credentials, cloud roles, package workflows, AI tools, and CI/CD runners combine to form access paths that attackers chain together.

The Five Identity Attack Patterns

Across these incidents, five patterns recur:

- **Developer credential theft:** phishing, infostealers, token theft, and social engineering are used to access source code, packages, and platform consoles.
- **Automation identity abuse:** CI/CD runners, workflow tokens, deploy keys, and publishing credentials used to reach secrets, cloud infrastructure, and downstream users.
- **Secrets sprawl:** API keys, tokens, cloud credentials, database passwords, and AI-service credentials exposed across code, logs, build systems, collaboration tools, and developer machines.
- **Package identity compromise:** maintainer accounts and publishing tokens used to push malicious packages into trusted dependency paths.
- **AI-agent access drift:** AI agents and coding tools are accumulating access to repositories, data, tools, commands, and cloud systems without equivalent ownership or governance.

In every case, the access was the target.

Section 3: The emerging frontier: AI agents as developer identities

AI agents are already operating inside developer environments. They generate code, inspect repositories, summarize pull requests, call internal tools, query documentation, write tests, open tickets, run commands, and assist with infrastructure work. Many also connect to cloud services, knowledge bases, databases, issue trackers, support tools, and observability systems.
In most deployments, agents inherit broad access because restrictions slow adoption. A coding agent receives repository access, a platform automation agent receives ticketing, build, and deployment access, and a support engineering agent receives log,

customer metadata, and knowledge-base access. Each connection adds another identity path that the security team rarely owns.

Most security teams cannot answer basic questions about which agents exist, which data sources they can access, what permissions they hold, and through which execution paths they access sensitive cloud services. That visibility gap is the central risk for developer platforms.

An AI agent serves as a developer assistant, a non-human identity, and an autonomous decision-maker simultaneously. It holds credentials, interprets instructions, calls tools, and acts across systems. That combination produces risk patterns that traditional access review and service account management were never built to handle.

- **AI coding agent credential leak:** an agent with repository and terminal access is manipulated into exposing tokens from local configuration, environment files, or build outputs. The agent retrieves the sensitive material through access it already holds.
- **Build automation agent overreach:** an agent designed to manage deployment tasks holds write access to production infrastructure. A poisoned instruction or compromised tool causes it to trigger changes outside its intended scope.
- **Internal documentation agent exposed:** an agent connected to engineering documentation and support systems retrieves sensitive implementation details, customer metadata, credential references, or infrastructure instructions because its knowledge sources were not properly scoped.
- **Developer toolchain agent compromised:** an agent operating via local developer tooling or a CI runner enumerates files, inspects credentials, and exfiltrates secrets through approved outbound channels.
- **Confused deputy:** the agent has legitimate permission to access a system, and the attacker tricks the agent into using that permission on their behalf.
- **Cross-agent propagation:** one compromised agent passes instructions, files, credentials, or tool outputs to another agent, spreading access across the developer environment.
- **Orphaned credentials:** agents created for experiments, prototypes, or temporary automations retain tokens and permissions long after their original use cases end.
- **Audit gap:** When an action is taken by an agent on behalf of a developer via an interface, attribution becomes difficult.

The Model Context Protocol makes this more urgent by standardizing how agents connect to tools. Unosecur frames MCP security as a lifecycle problem covering how identities are created, trusted, inherited, constrained, monitored, and decommissioned. AI agents now participate in the same identity paths that build, deploy, publish, and operate software, and governance has to participate alongside them.

Section 4: Why this is hard to solve

Understanding the attack patterns is necessary. Acting on them is harder because security leaders responsible for developer platforms work against structural conditions that complicate identity governance even with mature teams and strong intent.

Developer platforms are built for speed. Teams continuously create repositories, branches, packages, build jobs, preview environments, functions, database instances, secrets, service accounts, and integrations. Each action can create a new identity, permission, or trust relationship. Security teams rarely see these changes as quickly as developers create them. Non-human identities multiply faster than ownership records. A deploy key may belong to a project that no longer exists. A package publishing token may be held by a maintainer who changed teams. A CI/CD service account may retain access after a workflow was deprecated. The platform still accepts the credentials, while the business context around it has disappeared.

Developer access often spans multiple planes at once. A single engineer may hold access to source code, package registries, build pipelines, cloud projects, databases, logging tools, support systems, and AI coding environments. Each platform sees its own slice. None of them sees the complete identity path. Secrets scatter across places that were never meant to be secret stores. They appear in local configuration files, public commits, private repositories, CI/CD variables, build logs, container images, package scripts, test fixtures, environment files, support transcripts, and AI tool outputs. Once exposed, they can be copied, cached, forked, indexed, or embedded downstream.

Package ecosystems turn identity compromise into distribution. A maintainer account, publishing token, or build pipeline can push malicious code into downstream environments that trust the package. The identity blast radius extends through every project that installs, builds, mirrors, or deploys the dependency. AI agents make attribution harder. A developer approves a tool. The tool calls an agent. The agent invokes another tool. That tool uses a service account. By the time the action reaches cloud infrastructure or customer data, the original intent, the acting identity, and the effective permission chain may be separated across five systems.

Compliance does not close this gap. A team can pass periodic access reviews while dormant tokens, unowned service accounts, overprivileged build runners, shadow OAuth apps, and AI agents operate outside the approved inventory. Developer platforms do not pause for quarterly review cycles. Identity risk changes at runtime, and governance needs to operate at the same speed.

Section 5: What an identity-first defense looks like

Attack pattern	Identity-first control required
Developer credential theft	Runtime behavioral monitoring across developer identities
Automation identity abuse	Non-human identity discovery, ownership, and activity baselining
Secrets sprawl	Credential lifecycle visibility and right-sizing based on actual use
Package identity compromise	Publishing-token governance and anomalous maintainer activity detection
Valid-account lateral movement	Cross-system identity timeline and access-path reconstruction
AI-agent access drift	AI agent discovery, permission scoping, MCP governance, and agent behavior history

An effective response covers all three identity types: human, non-human, and AI agent. Coverage extends across source-code systems, cloud environments, SaaS applications, on-prem systems, CI/CD pipelines, package workflows, and AI environments. Runtime behavior analysis is required because configuration alone cannot show whether access is being used safely.

Unosecur's Unified Identity Fabric is built for this identity reality. The platform helps enterprises discover human, non-human, and AI agent identities, understand what they can access, monitor what they actually do, and act when identity risk appears across cloud, on-prem, SaaS, identity providers, CI/CD pipelines, and AI environments. Integration is read-only and agentless. Native API integrations reach roughly six times as deep as standardized protocols such as SCIM, SAML, and OIDC. Tenant provisioning completes in 15 minutes.

Unified Identity Fabric

Developer platform security fails when identity data stays fragmented. Source-code systems know repository access. CI/CD tools know build activity. Cloud providers know the role use. SaaS tools know sessions. Package registries know publishing events. AI systems know tool calls. Attackers move across all of them while each tool sees only its own surface.
The Unified Identity Fabric connects those signals into one operating view. It surfaces the relationships, access paths, unused privileges, risky behaviors, and identity dependencies that fragmented tooling leaves invisible across the software factory.

Non-human identity governance

The developer platform is full of non-human identities. Service accounts, API keys, OAuth apps, deploy keys, webhook tokens, CI/CD runners, package publishing tokens, database credentials, and automation accounts all operate continuously and rarely receive the same level of governance as human users.
Unosecur discovers these identities, connects them to ownership, baselines their behavior, and reduces standing privileges through activity-based right-sizing. Every non-human identity should carry an owner, a purpose, a scope, a lifecycle, and a behavior history. Findings include locally managed accounts within package registries and forked tools, partially offboarded service credentials that remain active after a team change, and orphaned deploy keys.

Runtime behavioral monitoring

Developer identity risk cannot be measured solely from permissions. A publishing token used after months of inactivity matters. A build runner accessing a new class of secrets matters. A developer account touching sensitive repositories at unusual hours matters. A service account suddenly exercising unused privileges matters.
Unosecur monitors identity behavior at runtime across connected systems, so teams can distinguish expected access from risky access before it becomes lateral movement. Findings include Shadow Admins inside source code and cloud platforms, identity drift across OAuth apps and integration credentials, and MFA blind spots in personal access tokens and partner connections.

Activity-based right-sizing

Developer platforms accumulate privilege quickly. A build role receives broad access for a launch. A token is created for a migration. A service account is granted temporary permissions that become permanent. An AI agent receives access to make something work and keeps that access after the experiment ends.
Unosecur aligns permissions with actual usage and surfaces JEP recommendations through activity-based right-sizing. Standing privileges shrink, JIT access decisions become measurable, and toxic privilege combinations across source code, CI/CD, and cloud are flagged before they are exploited. Cross-cloud identity chains where a single CI/CD runner traverses AWS, Azure, and GCP roles are reconstructed in full.

Identity Timeline

When an incident crosses repository, package, pipeline, cloud, SaaS, and AI tooling, logs alone are not enough. Security teams need to know which identity acted, what access it held at the time, what systems it touched, whether its behavior changed, and what other identities or assets were connected to it.

The Identity Timeline provides security teams with behavioral history for each identity across connected systems. In developer platforms, that means reconstructing what a developer, token, service account, OAuth app, CI/CD runner, package publisher, or AI agent actually did, in sequence, across every connected environment.

Security for AI

AI agents must be governed as identities. The AI Agent Dashboard provides security teams with a dedicated way to view agents, assess risk, review data access, inspect permissions, and trace execution paths. It surfaces account-level metrics, risk findings, knowledge-base access, permissions, and access graphs for agent execution chains.

The MCP Auth Gateway extends governance into agent-tool interactions. It controls which tools agents can call, what data they can access, and how agent activity is audited across the environment. AI agents in developer workflows operate inside the same identity model as every other actor.

AI for Security

ARK AI is Unosecur's AI copilot that plans and executes actions across the Identity Fabric. A security engineer makes a request in natural language. ARK builds an execution plan, waits for approval or modification, executes the approved steps, and automatically writes an audit trail for each step.

ARK reaches across permissions, NHI inventory, AI agent inventory, the identity graph, compliance evidence, ticketing, and threat response. In a developer platform context, ARK can surface dormant service accounts with active cloud access, overprivileged AI agents touching production data, risky users above a defined threshold, or compliance evidence tied to specific identity controls. Response time compresses while approval and traceability remain in place.

Identity-first defense across your stack

Developer platform security cannot be solved one tool at a time. A source-code scanner will not govern a package publishing token. A secrets detector will not show whether a CI/CD runner is overprivileged. A cloud posture tool will not explain why an AI agent can invoke a tool that reaches production data. A SIEM will not automatically reconstruct the full identity chain across repositories, packages, builds, cloud roles, SaaS tools, and agents.

The Unified Identity Fabric connects these surfaces, shows which identities can move across them, and helps reduce the access paths attackers chain before they become a breach.

Conclusion

Developer platforms have become the control room for modern software. They connect code, cloud, data, automation, package ecosystems, customer environments, and AI development workflows. That concentration of access is exactly why attackers target them.

The incidents of the past year show a consistent pattern. The CI action compromise showed that trusted automation can expose secrets across thousands of repositories. The build system package compromise showed that a CI workflow issue can become a stolen publishing token, malicious packages, and downstream credential exposure. The package ecosystem worm showed that compromised developer identities can propagate attacks through authenticated publishing. Secrets sprawl data shows that credentials continue to leak at a software scale. AI-assisted development data shows that agentic workflows are entering production-grade repositories. These patterns share a common root. Identity has expanded faster than governance across code, cloud, pipelines, packages, data, and AI systems. AI agents make this more urgent. As developer platforms add autonomous agents across coding, testing, support, deployment, and infrastructure workflows, they create identities that operate with broad access, make tool decisions, and produce audit trails that are harder to interpret after the fact.

Three things every developer platform security leader should do now.

- **Inventory every identity type across the developer environment:** human, non-human, and AI agent. Without an answer to which identities exist and what each one can access, identity risk cannot be governed.
- **Reduce standing access across developer automation:** CI/CD runners, OAuth apps, deploy keys, package tokens, service accounts, and cloud roles should be scoped to actual usage, owned, rotated, and reviewed continuously.
- **Establish AI agent identity governance before adoption scales further:** every agent should carry ownership, permission scoping, tool visibility, behavioral history, and auditability before it becomes part of production workflows.

The organizations hit by the next wave of developer platform attacks will be those that allowed identities to accumulate access across code, cloud, pipelines, packages, data, and AI systems without unified governance.