2026 Software Supply Chain Security State of the Union

AI Adoption and Governance Concerns Reshape Software Security Landscape

Table of Contents

About This Report

The software supply chain looks drastically different today than it did a year ago. AI has moved from experimentation to a structural force, changing how code is written, how dependencies are assembled, and how quickly software reaches production. In a post-AI world these challenges are only accelerating, leaving most technical teams wondering: how do we keep up with all the change?

The JFrog 2026 Software Supply Chain Security State of the Union helps ground the discussion by identifying where risk is expanding (i.e., dependencies, binaries, AI artifacts), where controls are failing, and what attackers are exploiting right now. More importantly, it translates noise into actionable priorities:

Risk Prioritization

What to fix first based on where persistent and new risks reside

Strategic Investment

Where to focus investment based on where governance efforts are, and are not, keeping pace

Board Justification

How to use the context of the broader software supply chain and development landscape to justify investment to the board

This report aims to help engineering and security professionals move from reactive patching to systemic control of software risk, with the data to justify every decision.

As the System of Record for thousands of global organizations and more than 80% of the Fortune 100, the JFrog Software Supply Chain Platform has insights across billions of software artifacts.

This report is the resulting analysis of that data combined with independent vulnerability research by the JFrog Security Research team and commissioned third-party survey responses from 1,508 security, development, and operations professionals across eight countries.

We hope you find value in this report and welcome any feedback, which can be shared with us at data_report@jfrog.com.

The Study in Brief

AI is reshaping the software supply chain at an unprecedented rate, but governance frameworks are struggling to keep pace. A new wave of risk is forming faster than it can be addressed, fueled by AI-generated code, model artifacts, and agentic tools. This trend is creating a widening gap between reported security confidence and actual, measured exposure, or an "illusion of mastery".

The tsunami of binaries is here

Your software supply chain isn't just growing; it's multiplying. Driven by AI-assisted development, artifact creation has hit record-shattering levels, and its composition is shifting in ways that reflect a fundamental change in how software is built.

- The JFrog Platform held **18.2 billion** artifacts at year-end 2025 across JFrog SaaS customers, up 136% from 2024.
- Hugging Face published **1.4 million new packages** in 2025, making it the second largest source of new packages, behind only Docker Hub.
- **npm overtook Maven** as the most-used package ecosystem by traffic. In a parallel shift, **PyPI passed YUM**, indicating that AI/ML workloads are displacing legacy infrastructure. This is more than a traffic shift; agentic workflows and AI-assisted development favor scripting languages, and the ecosystems carrying the most traffic today reflect that shift.
- **AI coding agent extensions on OpenVSX grew 262%** YoY, from approximately 1,000 to 3,803 new packages published.

Risk is accelerating across traditional and new attack surfaces

The threat landscape is expanding in all directions. While 2025 was the most dangerous year on record for npm users, a massive new attack surface of AI models and agentic developer tools emerged in parallel.

- Over **48,000 new CVEs** were disclosed in 2025, a 20% increase over 2024. The JFrog Security Research team attributes part of this growth to AI-generated code that does not apply secure coding practices, driving a surge in decades-old weaknesses like XSS, SQL Injection, and other Injection flaws.
- **Malicious npm packages skyrocketed by 451%**, reaching 171,592 unique instances, fueled by three major hijack campaigns that resulted in more than two million compromised downloads.
- The JFrog Security Research team identified **495 malicious models** on Hugging Face carrying live payloads including reverse shells, credential harvesting, and system command execution.
- JFrog also identified **969 malicious AI agent skills** in the ClawHub and Skills.sh repositories, filtering out many false positive malicious “findings”. These findings show how attackers often beat defenders when examining new technologies for security issues.
- **56 malicious extensions were detected on OpenVSX** the first time this attack surface has been tracked in this report. As a rapidly growing ecosystem, OpenVSX is an increasingly attractive target.

These patterns did not end in 2025. In early 2026, JFrog's AI-powered RepoHunter bot identified 13 critical CI/CD vulnerabilities across widely used open-source projects, including Ansible, CNCF Telepresence, and JavaScript standards repositories, before attackers could exploit them.

The attack mechanics were identical to those used in the Shai-Hulud campaign: CI/CD pipeline misconfigurations that give attackers access to secrets, publishing tokens, and cloud credentials. The same mechanics that powered Shai-Hulud are still present across the open-source ecosystem.

Not all reported vulnerabilities are what they appear

Volume-based triage is failing security teams. Having more CVEs to chase doesn't mean more actual risk, it means more noise.

- **66% of CVEs analyzed had a low applicability rate (0–20%)**, meaning that even when a vulnerability exists in code, the conditions required to exploit it are rarely met in practice. Only 12% were found to be highly exploitable in real enterprise environments.
- **Only 40% of survey respondents adopted malicious package detection** (unchanged from last year) and **secrets detection is in active use at only 28% of organizations**. The threat grew, but the coverage didn't. In fact, the categories growing fastest in threat volume are the least covered by security tooling.

Even governance needs governance

While organizations are rapidly adopting AI development tools, models, and protocols, governance frameworks for these new surfaces remain nascent or aspirational. AI is no longer an emerging consideration in the software supply chain. It is the supply chain.

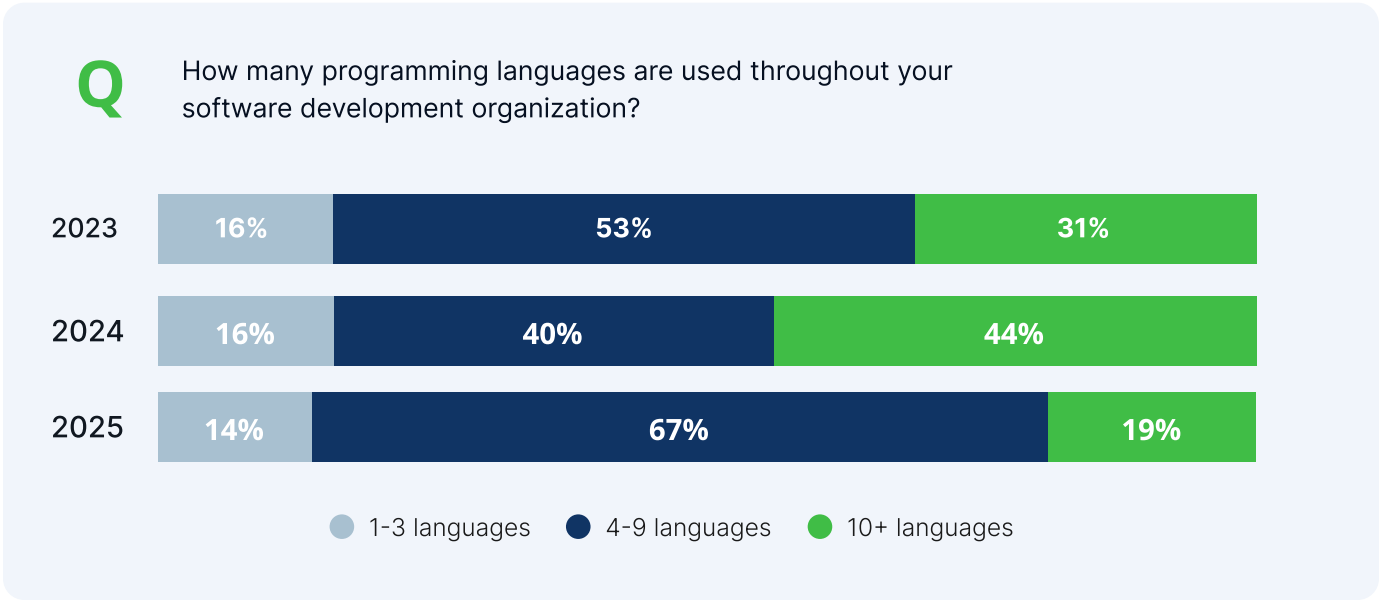
- **41% of organizations are actively using AI and ML libraries** (the software packages that connect applications to AI models and services), up from 34% in 2024. The average organization is now managing **47% more** of these packages than last year, as teams move from relying on a single AI service to multi-AI realities, building across several services simultaneously.
- **53% of organizations self-host AI models** in some form, pulling from Hugging Face and similar public registries where 495 malicious models were detected. Yet **97% claim certified model governance**, a figure the malicious model data calls into question.
- **59% report full production provenance visibility.** Yet 48% still need a week or more to generate compliance audit proof. Visibility without accountability is not governance.
- **23% of developers would treat an AI-suggested security fix as near-definitive** with only a quick review. **18% of organizations have no active governance for the IDE extensions and MCP servers** sitting inside their developers' environments — either no policies at all, or policies that aren't enforced.

What's in Your **Software Supply Chain?**

The enterprise software supply chain's composition shifted in ways that reflect a fundamental change in how software is built. AI is no longer just a topic for which organizations are preparing. It is actively shaping what gets written, what gets assembled, and what enters production pipelines every day.

Number of programming languages used in development organizations

The distribution of language usage across organizations shows a marked shift toward consolidation. Fourteen percent of organizations report using just 1–3 languages, 67% use 4–9, and 19% use 10 or more.



This represents a **sharp reversal of a multi-year expansion trend**. The share of organizations using seven or more languages has declined from 64% in 2024 to 45%. The contraction is even more pronounced at the high end: organizations using 10 or more languages dropped from 44% in 2024 to just 19%.

The most plausible explanation is **intentional consolidation:** organizations rationalizing their technology stacks, reducing the sprawl that accumulated during years of rapid expansion, and accelerated by AI coding assistants that may default to certain languages and frameworks.

Fewer languages may mean a more governable attack surface, but they also concentrate risk. **When enterprise development runs on two or three ecosystems, a single critical vulnerability in a foundational library reaches further than ever before.**

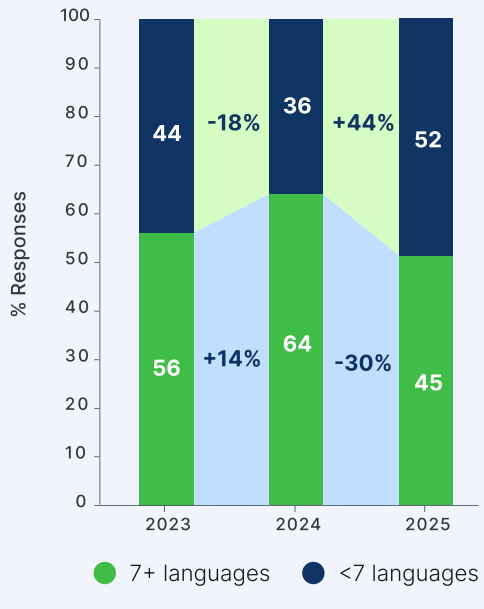


Figure 1.1 YoY change in number of languages used by organizations

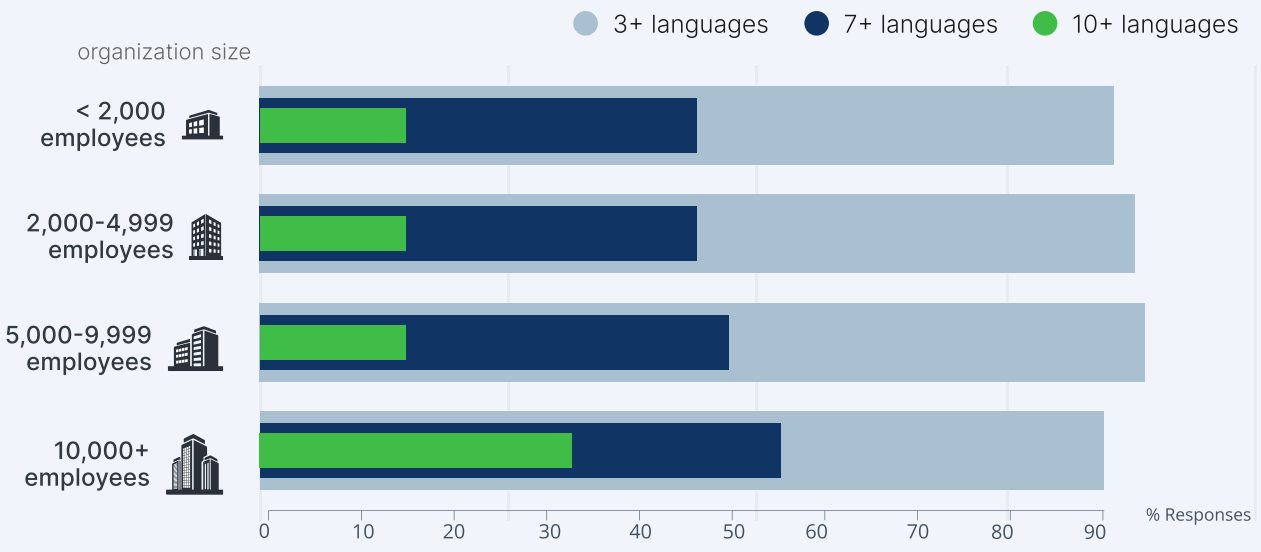


Figure 1.2 % of organizations using 3+, 7+ or 10+ programing languages by organization size

By **organization size**, the consolidation trend holds broadly, but with one notable exception. Organizations with 1,000–9,999 employees cluster tightly in the 42%–47% range for 7+ language usage, suggesting relatively consistent behavior across that band. The outlier is the largest tier: organizations with 10,000 or more employees report 7+ language usage at 50%, the highest of any size group in 2025, and a direct reversal of last year's pattern, when the largest organizations had the *lowest* rate.

This shift is particularly visible at the 10+ language threshold, where organizations in the 10,000+ tier report 30%, compared to just 14% across every other size band. The inflection point we observed in 2024, where scale drove standardization, appears to have shifted, or the largest organizations are simply adding new technology stacks faster than they can rationalize them.

Seniority introduces a striking perception gap. C-suite respondents report 7+ language usage within their organizations at 64%, while individual contributors report just 35%, a 29-point gap that is consistent with what we observed in 2024. Senior leaders may have broader visibility into the portfolio of languages used across teams, or they may be counting languages they've approved for use rather than those actively deployed. Either way, the divergence is substantial enough to suggest that organizations' self-reported language complexity looks meaningfully different depending on where you sit in the hierarchy.

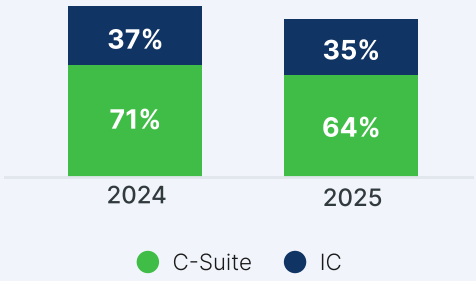


Figure 1.3 Percentage of respondents who report using 7+ languages within their organization

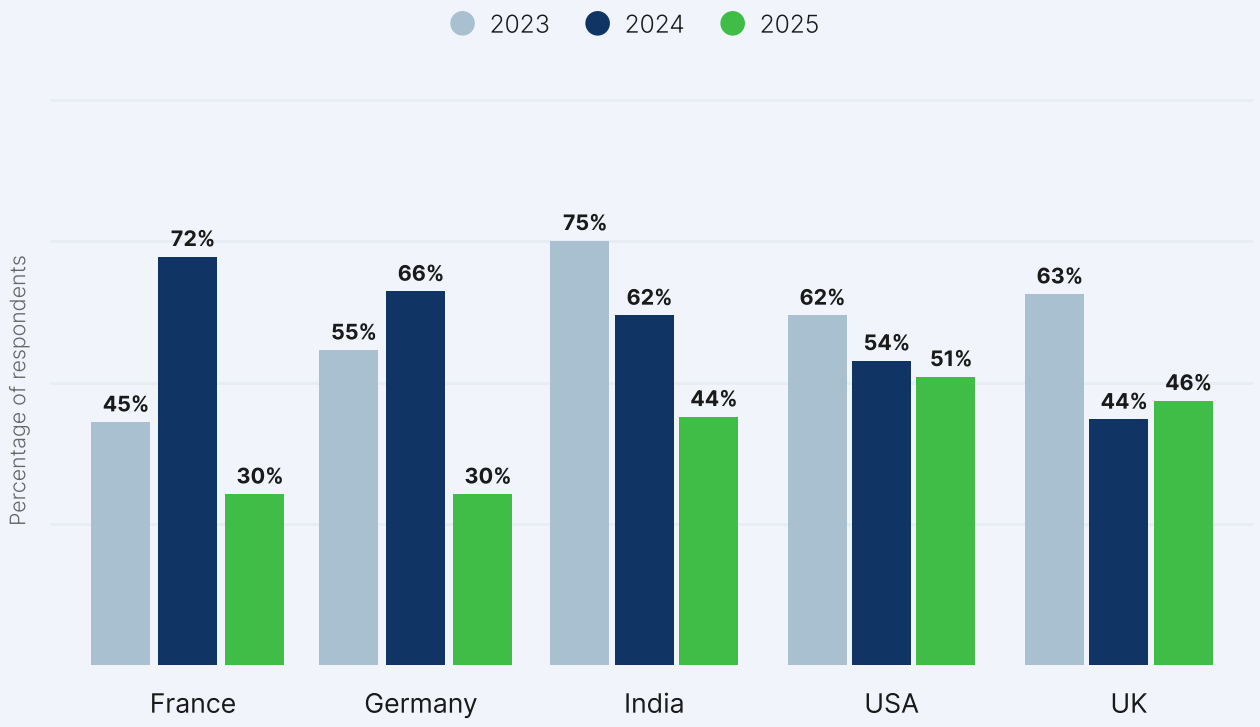


Figure 1.4 Percentage of 7+ languages usage by country

The **country-level data** tells perhaps the most striking story of this year's findings. The United States now reports the highest rate of 7+ language usage among surveyed countries at 51%, not because US organizations have grown more complex, but because every other country declined faster. The US has followed a gradual, consistent downward trend across all three survey years (62% → 54% → 51%), while other markets have seen far more volatile swings.

India's decline is the most dramatic in absolute terms: from 75% in 2023 to 62% in 2024 to 44% in 2025; a 31-point drop over three years. Germany and France tell a different story: both surged to the mid-60s in 2024 before falling sharply, and both now sit at 30%, the joint lowest of any surveyed country. France recorded the largest single-survey swing of any country or year in this dataset, dropping 42 percentage points from its 2024 peak of 72%.

The synchronized decline in these two markets (and the speed of it) may reflect the influence of European regulatory pressure, as organizations operating under frameworks like the EU Cyber Resilience Act rationalize their stacks in response to new software supply chain obligations.

New packages per year per package type

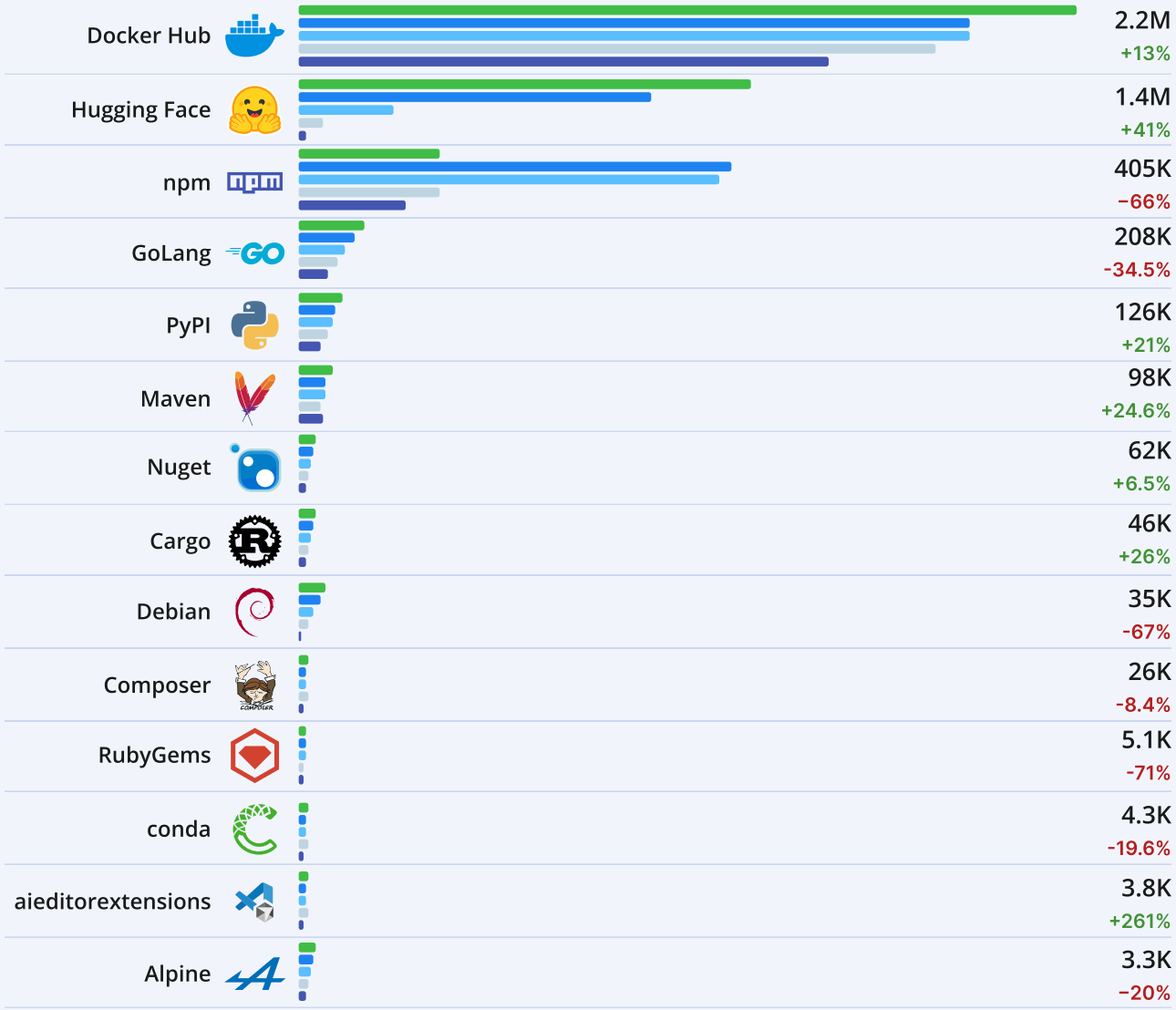


Figure 2. Number of new OSS packages created per year, by package type, 2021–2025 (JFrog Catalog database, 2025)

JFrog Catalog data examining public registries adds a new chapter to what has become one of the most consequential trends in the software supply chain: the rise of AI/ML as an ecosystem unto itself.

By raw volume, **Docker Hub** remains the largest source of new packages of any registry we track, adding 2.2 million new packages in 2025, up from 1.9 million in 2024 (+16%). The growth is consistent with Docker Hub's role as the de-facto container repository: steady, predictable, and largely reflective of the natural expansion of containerized workloads rather than any structural shift.

The more consequential story surrounds **Hugging Face**. Where the growth of Docker Hub has been incremental, Hugging Face has evolved from a fast-growing outlier into a fundamentally different category of ecosystem. It surpassed 1.4 million new models in 2025, a 41% year-over-year increase following roughly 4x growth the year prior. Hugging Face now rivals Docker Hub in volume while representing an entirely new class of artifact: AI models and datasets, with distinct provenance, licensing, and integrity challenges that traditional package governance was not built to address.

Importantly, AI adoption leaves a footprint across the entire ecosystem, not just in AI-specific registries. Organizations expanding their AI workloads show corresponding growth in Docker, PyPI, and npm — the infrastructure layer that AI pipelines run on. In this sense, growth in traditional ecosystems is itself a signal of rising AI activity, even when the artifacts involved are not AI-specific.

---

npm presents a more complex picture. After spiking to 1.19 million new packages in 2024, the registry saw +400,000 new packages in 2025, a 66% year-over-year decline. The cause is not fully apparent, though one possibility is that npm has become more effective at detecting and blocking spam and low-quality packages at the point of publication. The 2024 spike in npm packages coincided with a period of unusually high automated package activity, and a tightening of registry-level controls could explain the return to lower volumes. What is clear is that the drop in new packages does not reflect a decline in npm adoption. JFrog Platform traffic data shows npm overtaking Maven as the most-used ecosystem by request volume in 2025, its highest relative position since we began tracking.

PyPI continues its steady climb, adding 126K new packages in 2025, up from 104K the year prior. The growth is organic and consistent across all five years of data, reflecting the sustained expansion of Python as the default language for data science, automation, and AI tooling. There are no dramatic inflection points here, which is itself meaningful. Python's ecosystem is growing because developers keep choosing it, not because of any single catalytic event.

Maven showed its first meaningful acceleration in years, growing 24% in 2025 to reach 98K new packages, up from a relatively flat four-year range of 63K–79K. The timing aligns with the rapid expansion of Java-based AI and backend tooling, and suggests that the Java ecosystem, long considered mature and stable, is being pulled back into active growth by the same AI wave reshaping every other registry.

Cargo, the Rust package registry, has grown from 20K new packages in 2021 to 46K in 2025, a 129% increase over four years, and the most consistent compound growth rate of any ecosystem in this dataset. As noted in the 2025 report, this trajectory reflects sustained pressure from government and industry to adopt memory-safe languages, and Cargo's unbroken upward trend suggests that pressure is translating into real adoption rather than stated intent.

Top package technologies in use by organizations

** Requests = aggregate of upload and download counts*

Package type	Number of Repositories	Artifacts	Market Share (% of *requests)	Traffic YoY
npm	143,244	1.62B	33.17%	+35.5%
Maven	315,522	5.93B	30.13%	+20.4%
Docker	305,238	5.90B	14.15%	+31.2%
Generic	287,044	2.65B	7.18%	+126.8%
PyPI	79,232	182.9M	3.31%	+79.4%
YUM	35,298	55.4M	2.49%	+18.0%
Cargo	2,862	1.4M	0.28%	+381.2%
OCI	31,708	108.7M	0.20%	+79.2%
HelmOCI	8,350	4.2M	0.20%	+565.6%
Hugging Face	2,387	44K	0%	+1190.78%

Figure 3. Technologies used, plus action counts, number of repos, and total size of artifacts stored for each (JFrog SaaS platform data, 2024-2025)

Overall platform traffic on JFrog SaaS grew 33.5% year-over-year across all tracked package ecosystems. That growth rate, on top of what was already a record base, reflects both organic expansion of existing development activity and the emergence of new workload types, particularly AI/ML pipelines pulling model artifacts at scale and agentic workflows that further accelerate that velocity.

The headline finding is a first in this report's history: npm overtook Maven as the most-used package ecosystem, meaning JavaScript has surpassed Java as the dominant enterprise package ecosystem by request volume.

npm now accounts for 33.2% of total platform traffic versus Maven's 30.1%. npm's traffic grew 35.5% year-over-year, outpacing both Maven (+20.4%) and the overall market (+33.5%), gaining share while Maven lost it. While neither ecosystem is shrinking and the total is actually growing, npm is taking a larger slice of a bigger cake. This is more than a traffic shift.

Agentic workflows and AI-assisted development favor scripting languages: JavaScript and Python are faster to iterate in, easier for non-developers to work with, and the default output of many AI coding assistants. Enterprise Java isn't going anywhere, but the new layer of software being built on top of it runs on npm and PyPI.

The same dynamic explains **PyPI passing YUM**. PyPI grew at 79.4%, more than double the overall market rate, while YUM grew at only 18%, well below market. YUM is still growing in absolute terms, but Python is being adopted much faster.

Some ecosystems show dramatic percentage growth while remaining a small fraction of overall traffic. **Cargo/Rust** grew 381% in traffic and 127% in repositories, which is consistent with its five-year new package growth trend and ongoing government mandates for memory-safe development languages. Rust still accounts for only 0.3% of total platform traffic, but it has nearly quadrupled in a single year and shows no sign of plateauing. The interpretation: AI/ML workloads running on Python now represent a larger share of enterprise software activity than legacy Linux package management. That is a meaningful signal about where enterprise development is heading.

The 3,578% growth in **OCI** repositories and 79% growth in OCI traffic, together with 411% growth in HelmOCI repositories and 565% growth in HelmOCI traffic, reflect a structural shift in how Kubernetes application packaging works. The Kubernetes community made OCI the default packaging format for Helm charts at the platform level; organizations didn't choose OCI so much as the ecosystem moved there by design. The numbers reflect that transition reaching critical mass in 2025, and JFrog's support for both Helm Legacy and OCI protocols made the migration straightforward for customers.

It is also worth noting the scale of **Hugging Face's** growth on the JFrog Platform. Traffic grew 1,190% year-over-year, and the number of artifacts stored grew from 12,750 to 43,823 — a 244% increase. Together these figures signal that organizations are not just standing up model infrastructure, but actively using it. **AI models are entering the enterprise software supply chain at scale.**

AI repository growth within organizations

There was substantial growth across JFrog's dedicated repository types for AI and ML artifacts in 2025. As organizations move beyond experimenting with AI to actively integrating models into their development pipelines, the infrastructure required to manage those artifacts is growing with them, and so are the governance challenges that come with tracking provenance, licensing, and integrity at scale. Note that these figures reflect growth in explicitly AI-designated repository types only; AI's broader footprint, visible in Docker, PyPI, and Helm chart growth, extends further than these numbers alone capture.

The public ecosystem is growing faster than organizations can govern it. According to JFrog Catalog analysis, Hugging Face published 1.4 million new models in 2025, making it the second largest source of new packages we track, behind only Docker Hub. AI editor extensions on OpenVSX grew from 996 in 2023 to 1,051 in 2024, then jumped to 3,803 in 2025 — a 262% single-year increase that reflects how rapidly the developer tooling layer shifted toward AI-native workflows.

JFrog Platform data tells the same story from within enterprise environments. Hugging Face traffic on the JFrog Platform grew 1,190% year-over-year. To put that in context: across all four quarters of 2025, Hugging Face traffic grew at an average of 182% quarter-over-quarter, compared to 13% for npm, 11% for Docker, and 6% for Maven over the same period. The three largest traditional ecosystems grew steadily and predictably throughout 2025. Hugging Face grew in a different order of magnitude entirely.

Quarter	Hugging Face	npm	Docker	Maven
Q1	40%	20%	−3%	2%
Q2	543%	10%	12%	7%
Q3	4%	9%	18%	10%
Q4	143%	11%	17%	6%
2025 avg	**183%**	**13%**	**11%**	**6%**

Figure 4. QoQ growth by ecosystem (JFrog database, 2025)

It is also worth noting that in 2025 JFrog added support for two new AI-specific package types — **NIM (NVIDIA Inference Microservices)** and **AI editor extensions** — both appearing in the platform data for the first time this year.

AI artifacts are becoming a standard component of the enterprise software supply chain, and the governance frameworks to manage them are still catching up.

Popular libraries

#	Docker	Maven	PyPI	npm
1	node	commons-io:commons-io	charset-normalizer	debug
2	python	org.slf4j:slf4j-api	certifi	@types/node
3	nginx	org.ow2.asm:asm	urllib3	semver
4	postgres	commons-codec:commons-codec	setuptools	caniuse-lite
5	alpine	org.apache.commons:commons-lang3	requests	get-intrinsic
6	maven	org.apache.commons:commons-text	packaging	electron-to-chromium
7	busybox	org.codehaus.plexus:plexus-utils	typing-extensions	@babel/parser
8	redis	com.fasterxml.jackson.core:jackson-databind	idna	@babel/types
9	openjdk	com.fasterxml.jackson.core:jackson-core	PyYAML	undici-types
10	docker	com.fasterxml.jackson.core:jackson-annotations	click	browserslist
11	ubuntu	com.google.guava:guava	Pygments	@babel/generator
12	golang	org.apache.maven:maven-model	numpy	@babel/traverse
13	mongo	org.codehaus.plexus:plexus-classworlds	cffi	@babel/helper-validator
14	mysql	org.codehaus.plexus:plexus-archiver	MarkupSafe	es-object-atoms
15	centos	org.apache.maven:maven-core	Jinja2	update-browserslist
16	elasticsearch	org.codehaus.plexus:plexus-component	cryptography	qs
17	eclipse-temurin	org.junit.platform:junit-platform-commons	attrs	@babel/compat-data
18	rabbitmq	net.bytebuddy:byte-buddy	pytz	brace-expansion
19	gradle	org.yaml:snakeyaml	pip	@babel/helpers
20	ruby	org.apache.maven:maven-plugin-api	tzdata	resolve

Figure 5. Top 20 downloaded packages for Docker, Maven, PyPI, npm into JFrog SaaS (JFrog database, 2025)

Public download counts tell you how often a package was fetched. They don't tell you how many organizations actually depend on it. A single CI pipeline can fetch the same package thousands of times a day, inflating registry metrics far beyond what reflects real adoption. Rather than counting downloads, our research analyzes what is **actively being requested into production environments** across thousands of JFrog SaaS customer accounts, giving a more accurate picture of what enterprises actually run.

Docker

For Docker, the official Node.js image overtook Alpine as the most-requested image in 2025, a shift from prior years when Alpine dominated as the minimal base image of choice. The top 20 continues to reflect the most widely used operating systems and runtime environments, consistent with their role as parent images in multi-stage builds.

Three notable changes from 2024: hello-world dropped out; whereas its presence last year signaled a healthy volume of developer onboarding and POC activity. Its absence may indicate a maturing customer base. Grafana and Amazon Corretto also exited, replaced by mysql, elasticsearch, rabbitmq, and gradle, a shift toward data infrastructure and backend service components. This suggests organizations are pulling in more complete application stacks rather than just language runtimes.

Maven

The Maven top 10 is largely stable year-over-year. The more significant story is in the bottom half. JUnit 4 (`junit:junit`) dropped out entirely, replaced by `org.junit.platform:junit-platform-commons`. This is a clear signal that the Java ecosystem's long-running migration from JUnit 4 to JUnit 5 is now reflected in `enterprise` production environments, not just new projects.

Apache HttpComponents (httpcore, httpclient) also dropped out, likely displaced by newer HTTP client implementations built into modern Java versions. `org.yaml:snakeyaml` entered the top 20 at #19. SnakeYAML has an extensive CVE history, most notably CVE-2022-1471 (CVSS 9.8 — constructor deserialization RCE), which was only addressed in version 2.0 released in February 2023. Its appearance in the top 20 suggests it remains widely deployed across enterprise Java environments.

PyPI

The PyPI top 20 is the most stable of all four ecosystems — 16 of 20 packages appeared in last year's list. The new entrants are telling: Jinja2 (#15) and Pygments (#11) are templating and syntax-highlighting libraries most commonly pulled in by documentation tooling and AI code generation frameworks. Their rise is consistent with the growth of Python-based AI tooling documented elsewhere in this report. wheel, importlib-metadata, and zipp dropped out; older packaging infrastructure increasingly handled natively by modern pip versions, which itself entered the top 20 at #19.

npm

The npm top 20 saw the most significant rotation of any ecosystem with nine new packages entered. The @babel ecosystem expanded from three entries in 2024 to seven in 2025, now occupying positions 7, 8, 11, 12, 13, 17, and 19. Babel has become the dominant JavaScript transpilation toolchain, allowing modern JS code to be converted to browser-compatible formats, and its pervasiveness as a transitive dependency means most JavaScript projects pull in multiple Babel packages whether or not they explicitly depend on Babel directly.

Two other new entrants are worth noting: `undici-types` (#9) signals the broad adoption of Node.js's built-in fetch implementation over third-party HTTP libraries. `get-intrinsic` and `es-object-atoms` reflect the growing presence of core-js low-level JavaScript primitives as transitive dependencies across the npm graph. Nine packages dropped out compared to 2024, including chalk, commander, glob, and minimatch, widely used CLI utilities being replaced by native Node.js equivalents, such as util.styleText.

Pace at which new OSS packages are being injected into an organization

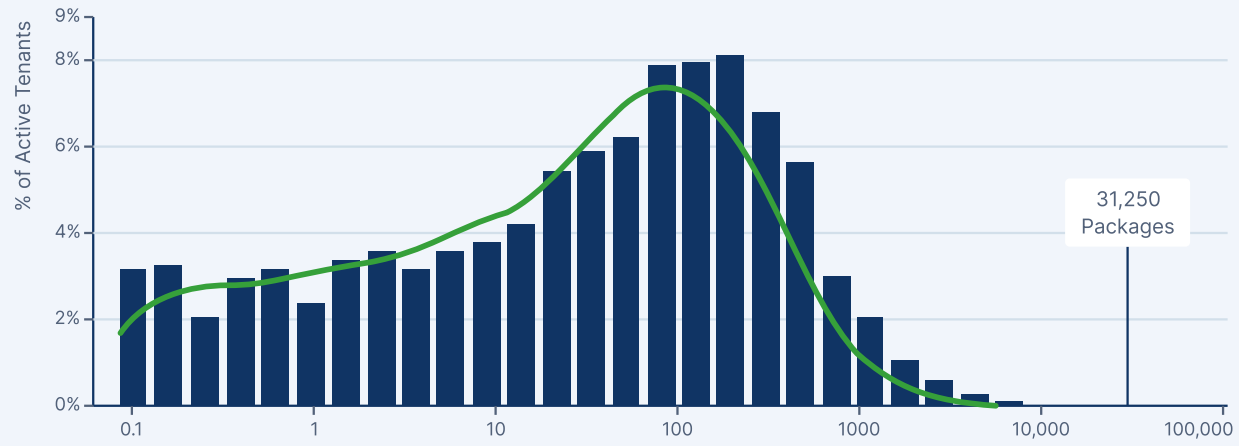


Figure 6. Distribution of new packages created monthly for active tenants in 2025 (JFrog database, 2025)

In 2025, organizations using JFrog SaaS brought a total of over **11.7 million new packages** into their software supply chains, up from seven million in 2024, a 67% increase in a single year. For the average organization, that is around 3,500 packages throughout the year. As in prior years, this number is pulled upward by a small number of very heavy users. The largest single organization brought in approximately 375,000 new packages over the course of the year.

The median organization (a more representative measure) brought in 509 packages annually in 2025, up from 231 in 2024. That is roughly **42 new packages per month**, or more than one new package per day.

That pace matters. Every new package is a potential entry point for a vulnerability, a license obligation, or a malicious payload. At a rate of more than one per day, the question is not whether organizations are exposed to supply chain risk; it is whether they have the **tooling and processes to evaluate each new dependency** before it reaches a developer's machine.

AI library adoption within organizations

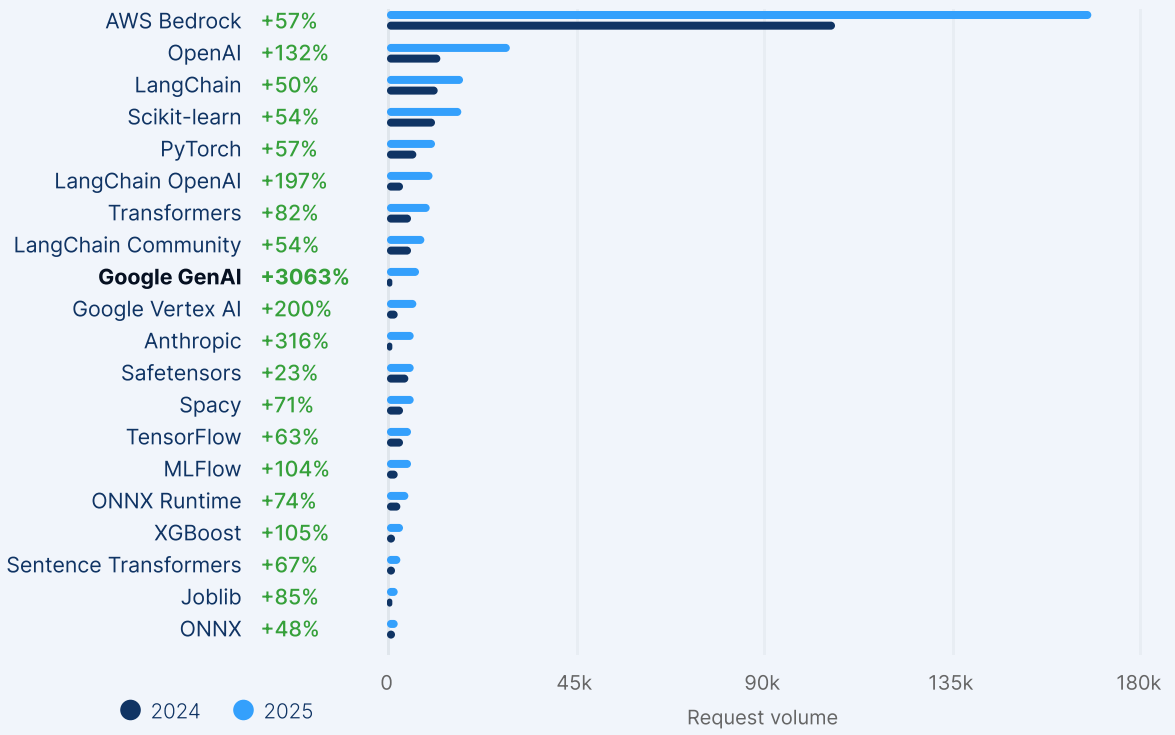


Figure 7. AI and ML library usage by request volume across JFrog customer environments (JFrog platform data, 2025)

Total AI and ML library usage across JFrog customer environments grew 74% year-over-year, from 194,734 to 337,573 in 2025.

Among JFrog customers, the share actively using AI and ML libraries grew from 34% in 2024 to 41% in 2025. Among those already using AI packages, the average number of libraries per customer grew from 6.3 to 9.3 (+47%). Organizations aren't just adopting AI; **they are deepening their dependency on it**, pulling in more frameworks, more providers, and more tooling as their AI workloads mature.

AWS Bedrock remains dominant, but its lead is narrowing

The most widely referenced AI library by a significant margin, Bedrock grew from 107,186 to 168,582 (+57%), accounting for nearly half (47%) of all tracked AI library usage. Its growth is steady, but its rate is among the slowest for established major libraries and well below the 74% market average. Every other significant API provider grew faster. Bedrock's dominance reflects its head start and deep enterprise integration; its below-market growth rate reflects the rapid rise of alternatives.

The API provider landscape is diversifying rapidly.

Google GenAI is the most dramatic mover in the dataset, growing +3,063% from a near-zero base (244 to 7,718), reflecting rapid enterprise uptake of Gemini following its 2024 launch. Google Vertex AI grew +200% (2,250 to 6,742). Anthropic grew +316% (1,580 to 6,565). Mistral AI grew +657% (208 to 1,574). Groq grew +426% (254 to 1,336). OpenAI grew +132% (12,651 to 29,395). The pattern is clear: organizations are no longer building on a single AI provider. Multi-model architecture is becoming the default, and the library footprint reflects it.

Agentic frameworks are the fastest-growing new category.

CrewAI grew +206% (785 to 2,404) and AutoGen grew +68% (348 to 584), both enabling multi-agent AI workflows. Their growth signals that organizations are moving beyond single-turn AI interactions toward more complex, orchestrated pipelines.

LangChain is consolidating as the orchestration layer.

LangChain Core grew +50% to 18,196. LangChain OpenAI grew +197% to 10,893. LangChain Community grew +54% to 8,742. Across its three main packages, LangChain is increasingly the connective tissue between model providers and application logic.

MLflow's growth reflects the expanding model management burden.

MLflow grew +104% (2,793 to 5,695). As a framework for model creation, evaluation, and storage, its adoption reflects the fact that organizations are building and managing more models and agents than ever before. The more AI workloads an organization runs, the more it needs structured tooling to track experiments, version models, and manage the resulting artifacts.

Ollama's growth confirms that self-hosting is operational, not aspirational.

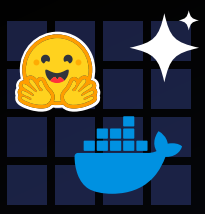
Ollama grew +327% (404 to 1,726), a trajectory that reflects a real and growing operational pattern rather than a stated preference.

Local inference is losing ground to hosted APIs.

Llama-cpp, one of the most common libraries for running models locally, declined 22% (356 to 276), the largest decline in the dataset. As hosted API quality and cost-efficiency improve, the case for raw local inference weakens. Organizations are shifting toward managed serving solutions like Ollama or cloud APIs.

SECTION 2

Key takeaways

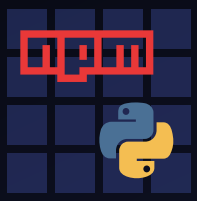


The AI ecosystem is now the second largest source of new packages — and the fastest growing.

Hugging Face published 1.4 million new packages in 2025, behind only Docker Hub's 2.2 million, and growing at 41% year-over-year following roughly 4x growth the year prior.

The supply chain has a new dominant input, one that operates differently from traditional package ecosystems: faster growing, less mature in tooling support, and pulling from a registry that most sourcing controls weren't built to reach.

41% of organizations are actively using AI libraries, up from 34% in 2024, and those who do average 9.3 libraries per organization. The other 59% are not yet using AI packages at all. That gap will likely close faster than it opened.

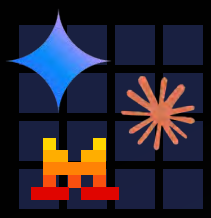


Enterprise are running on agents, not jars.

npm overtook Maven as the most-consumed package ecosystem for the first time in this report's history. PyPI passed YUM. These are not rounding errors. They reflect a structural shift in where enterprise software is being built and by whom.

Agentic workflows and AI-assisted development may favor scripting languages; JavaScript and Python are faster to iterate in, easier for non-developers to work with, and the default output of many AI coding assistants.

The security tooling, approval workflows, and governance policies most large organizations have in place were built around Java-centric supply chains. The ecosystems now carrying the most traffic are different ones. That gap is where risk lives.

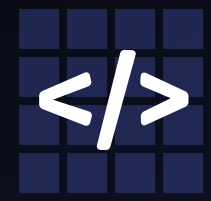


The multi-AI, multi-model architecture is reshaping your supply chain.

The multi-model architecture is the new normal and it's reshaping your supply chain. Two years ago, an organization's AI footprint meant a handful of Python ML libraries. Today, it means API clients for multiple providers, orchestration frameworks, inference engines, local serving tools, fine-tuned models pulled from public registries, and IDE extensions with direct access to codebases and credentials.

The evidence is in the library data: Google GenAI grew +3,063%, Anthropic +316%, Mistral AI +657%, Groq +426%, all growing at multiples of the market rate.

Agentic frameworks like CrewAI (+206%) and local serving tools like Ollama (+327%) are entering production pipelines alongside traditional ML libraries. The composition of the enterprise software supply chain has changed structurally, not just in volume, but in kind. And unlike traditional packages, many of these new components sit outside the governance frameworks organizations have spent years building.



Language consolidation is accelerating, but with hidden security implications.

Language diversity dropped sharply in 2025, falling below the 2023 baseline. Organizations appear to be rationalizing their technology stacks: fewer languages, more governable pipelines, smaller attack surfaces. This is broadly positive. But consolidation also concentrates risk: when the majority of enterprise development runs on two or three ecosystems, a single critical vulnerability in a foundational library reaches further than ever before.

The Accelerating Risk in Your Software Supply Chain

The AI-fueled software supply chain threat landscape is changing in two directions simultaneously. Existing risks, such as CVEs and malicious packages, have accelerated to levels with no historical precedent. New categories of threats materialized with almost no warning: CI/CD pipelines weaponized as worm distribution channels, agentic skills morphed from productivity assets into potential infection vectors with direct access to developers' machines. JFrog identified 969 malicious AI agent skills carrying high-impact payloads, putting a number on what was previously a theoretical risk.

What changed is not that AI creates risk. That was already established. What changed is where the infection point has moved. Attackers now have access to the same AI capabilities as defenders: the same models, the same automation, the same ability to operate at scale and speed. This is the adversarial symmetry of the AI era. A model does not know whose pipeline it is running in. It does not pick sides. What determines the outcome is not which side has better AI, it is which side has better governance and policy enforced around it.

Further, attackers are no longer waiting for developers to pull a malicious package. They are hacking into CI/CD pipelines before a package is published, and inside the IDE extension before a line of code is written. The campaigns documented in this section are not variations on last year's threats. They represent a structural shift in where the supply chain can be compromised, and a preview of what happens when that attack surface meets adversaries operating at AI speed.

Vulnerabilities found in a given technology or package type

In 2025, 48,244 CVEs were published to NVD, a 20.7% increase over 2024's 39,962. Part of this growth may be attributed to the proliferation of AI-generated code: vibe-coded applications tend to produce well-known vulnerability classes (e.g., SQL injection, XSS, missing authorization) at a volume that outpaces manual code review.

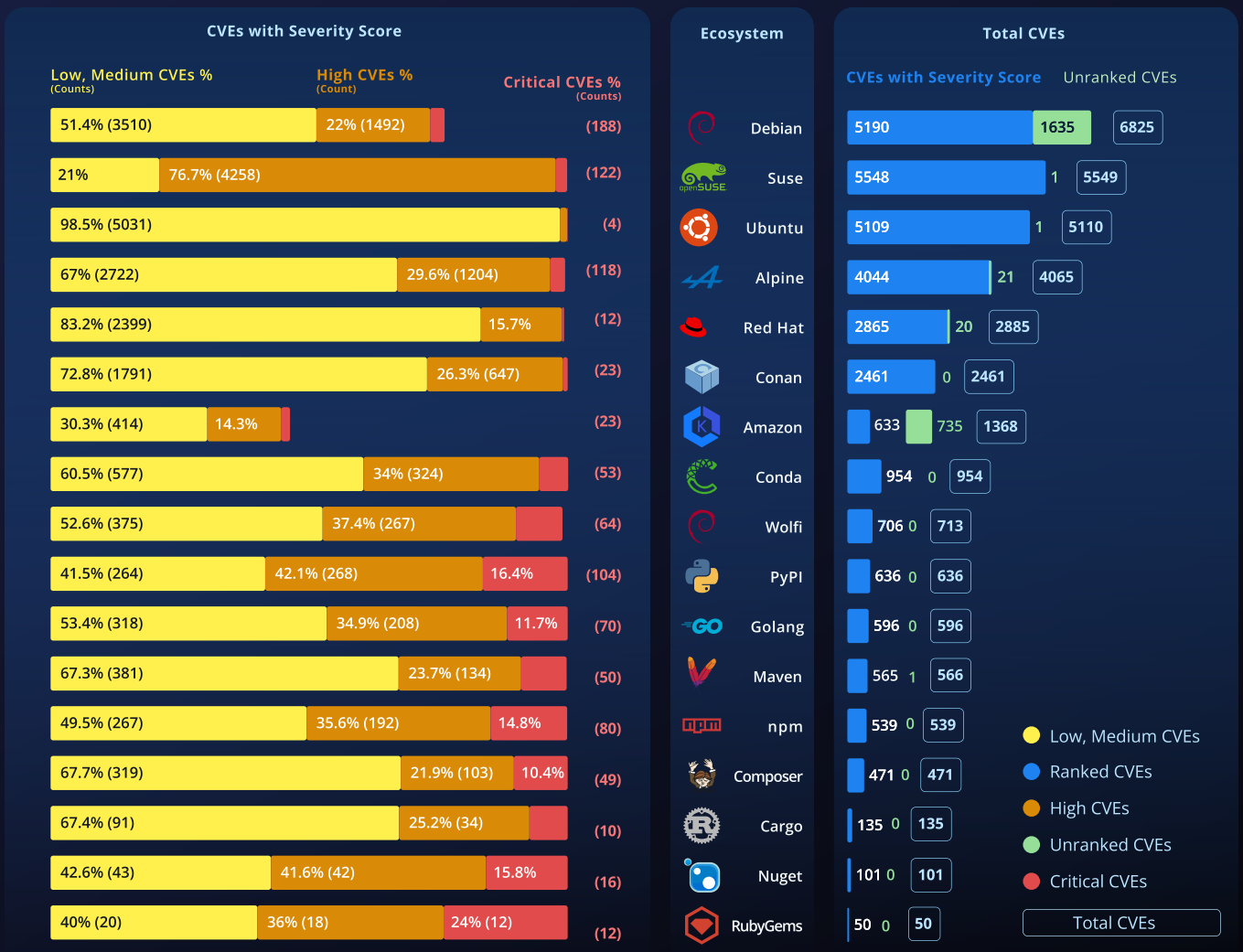


Figure 8. CVEs tracked across open-source ecosystems monitored by JFrog (JFrog database, 2025)

JFrog tracks CVEs across the open-source ecosystems it monitors, focusing on the developer-facing subset where organizations are most exposed. The distribution is heavily skewed: **Linux** distribution ecosystems account for the majority of raw volume, with **Debian** alone at 6,825 CVEs, **SUSE** at 5,549, and **Ubuntu** at 5,110, but that volume is concentrated in Medium-severity findings with low critical rates. Linux distributions' severity ratings take into account ecosystem-specific configuration, which narrows the actual severity of a given vulnerability in practice.

The more operationally relevant picture is in the developer-facing package ecosystems, where CVE counts are smaller but critical severity rates are dramatically higher. **RubyGems** had the highest critical rate at 24%, followed by **PyPI** and **NuGet** at 16% each. These are the ecosystems developers pull dependencies from daily, and a Critical CVE in npm or PyPI is far more likely to be reachable in production code than one buried in a Debian base image.

Several developer ecosystems showed notable year-over-year shifts. **npm** saw a sharp increase, rising from 410 CVEs in 2024 to 539 in 2025 (+31%) after a dip the prior year. **Maven** grew even more strongly, from 382 CVEs to 566 (+48%), reversing a steep 2024 decline that was partly attributable to NVD backlog effects. **Golang** continued its multi-year upward trend, reaching 596 CVEs (+23%). **PyPI** grew modestly (+2%), maintaining its position as the highest critical-density major ecosystem at 16% Critical. **Cargo/Rust** saw a significant jump from 61 CVEs in 2023 to 135 in 2025. While it is still the smallest developer ecosystem by volume, it has a trajectory worth watching as Rust adoption accelerates.

For security teams, the ecosystems worth watching most closely are not the ones with the highest absolute CVE counts today, but the ones growing fastest; npm, Maven, Golang, and Cargo are all trending in the same direction.

Most common types of vulnerabilities

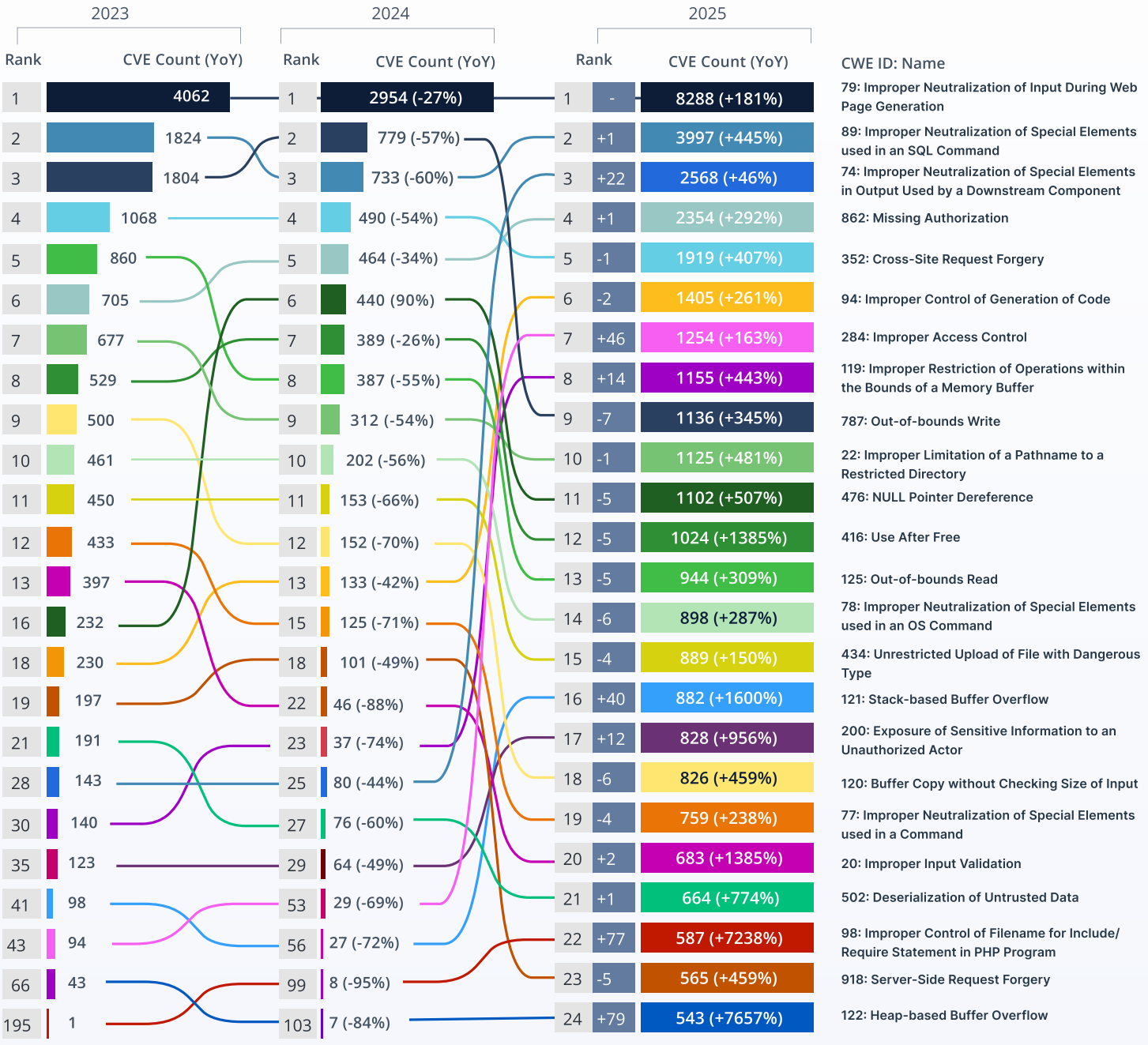


Figure 9. Most common vulnerability types disclosed in 2025, compared to 2023 and 2024

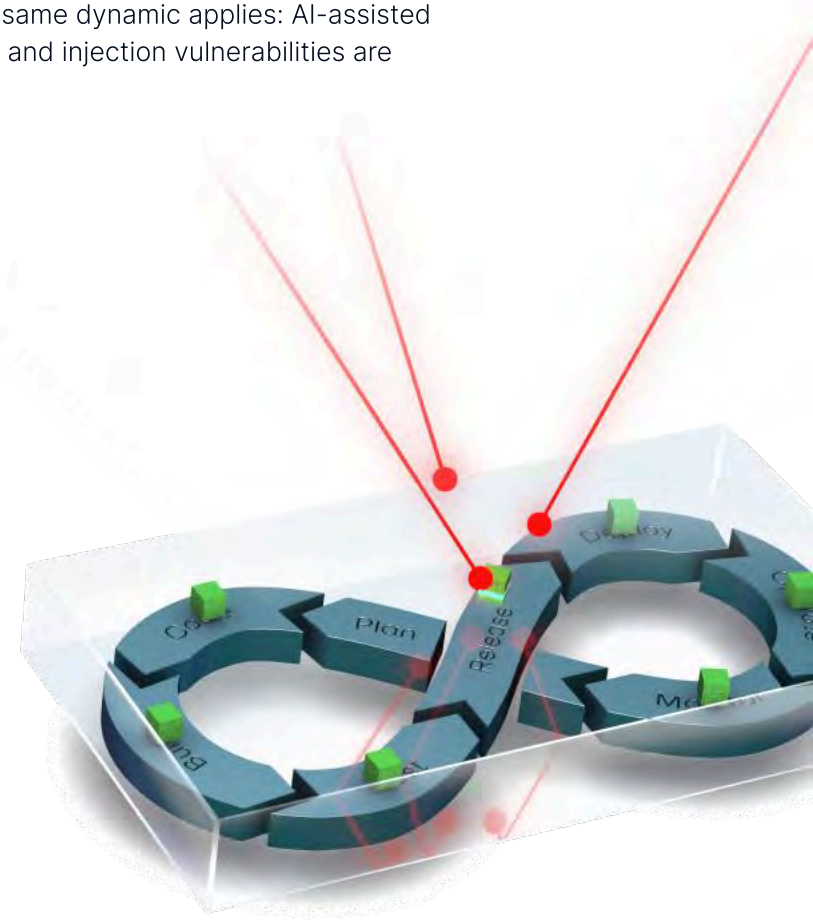
In 2025, 562 unique CWE IDs were assigned to CVEs, up from 243 in 2024, a 131% increase. The breadth of vulnerability types is expanding, but the most significant story is at the top of the distribution, where a single theme dominates: the rise of web application vulnerabilities, driven in large part by AI-generated code that doesn't apply secure development practices.

The top three vulnerability types in 2025 tell a consistent story:

1. **Cross-site Scripting (CWE-79)** retained the #1 position it held in 2024 but nearly tripled in volume, growing from 2,954 to 8,288 CVEs (+181%). Cross-site Scripting (XSS) is a classic web application vulnerability, and its surge is consistent with the explosion of vibe-coded web applications being published without adequate input validation.
2. **SQL Injection (CWE-89)** jumped from #3 in 2024 to #2, growing from 733 to 3,997 CVEs (+445%). The rise is directly attributable to AI-generated web applications that lack parameterized query practices, one of the most basic secure coding requirements for any database-connected application.
3. **Injection (CWE-74)** entered the top tier as the most dramatic new arrival, growing from 80 to 2,568 CVEs (+3,110%). Like XSS and SQL Injection, CWE-74 is predominantly a web application weakness, and the same dynamic applies: AI-assisted development is producing web applications at scale, and injection vulnerabilities are following.

The common thread across all three is significant. **XSS**, **SQL Injection**, and **Injection** are not new or exotic vulnerability classes; they have been understood and preventable for decades. Their simultaneous surge in 2025 is a direct signal of what happens when **application development accelerates faster than secure coding practices can be applied**.

Outside the top three, **Out-of-bounds Write (CWE-787)** illustrates a different dynamic. It held the #3 position in 2024 with 779 CVEs, but dropped to #10 in 2025 despite growing to 1,136 CVEs (+46%). The underlying risk in memory-unsafe code hasn't diminished, the web application vulnerability surge has simply grown around it.



Common vulnerability impacts for high-profile CVEs 2025

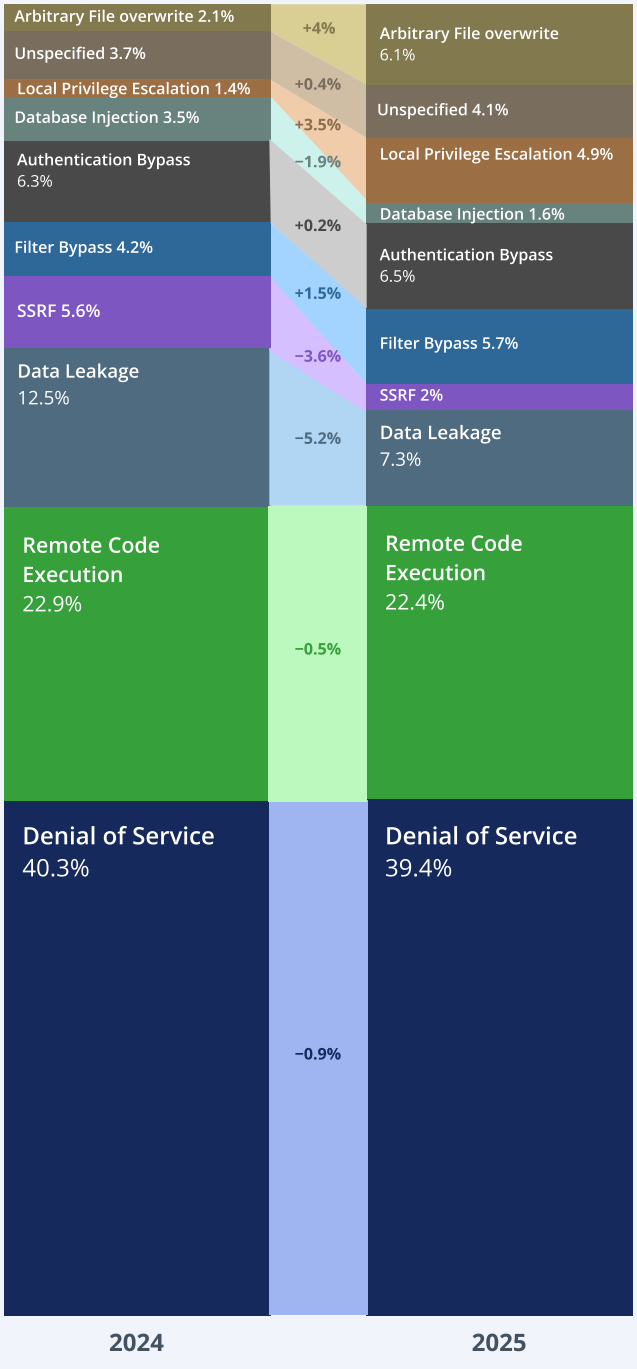


Figure 10. Common vulnerability impacts for high-profile CVEs 2024 and 2025

In 2025, the JFrog Security Research team analyzed hundreds of **high-profile CVEs (HPCVEs)** in open-source software packages. The team's CVE research goal is to provide in-depth analysis for CVEs in widely used open-source projects, researching the real-world severity of public vulnerabilities to help security teams understand when and how they affect their software projects.

The distribution of impact categories is remarkably stable year-over-year. **Denial of Service (DoS)** remains #1 at 39.1% (97 CVEs), **Remote Code Execution (RCE)** remains #2 at 22% (55 CVEs), and **Data Leakage** remains #3, though its share dropped from 12.5% to 7.3% as the overall dataset grew while its absolute count held steady.

The stability of this distribution across multiple years is itself meaningful. The types of vulnerabilities that reach high-profile status in enterprise open source are not changing; Denial of Service and Remote Code Execution have dominated this list consistently. What changes year to year is **which specific packages carry them** and **how exploitable they are** in practice.

The one notable exception in 2025 is **Arbitrary File Overwrite**, which jumped from three CVEs (2.1%) to 15 CVEs (6.0%), nearly tripling as a share of HPCVEs and representing the biggest year-over-year rise of any impact category.

Methodology note

The JFrog Security Research team considers several factors when prioritizing CVEs for research. The team focuses on technologies relevant to JFrog clients and prioritizes "High" and "Critical" severity issues (CVSS ≥7.5), supplemented by ML-based severity prediction when no CVSS score is available. CVEs actively exploited in the wild or with high media profiles are also prioritized regardless of public severity rating.

Severity of the vulnerabilities being introduced into your software supply chain

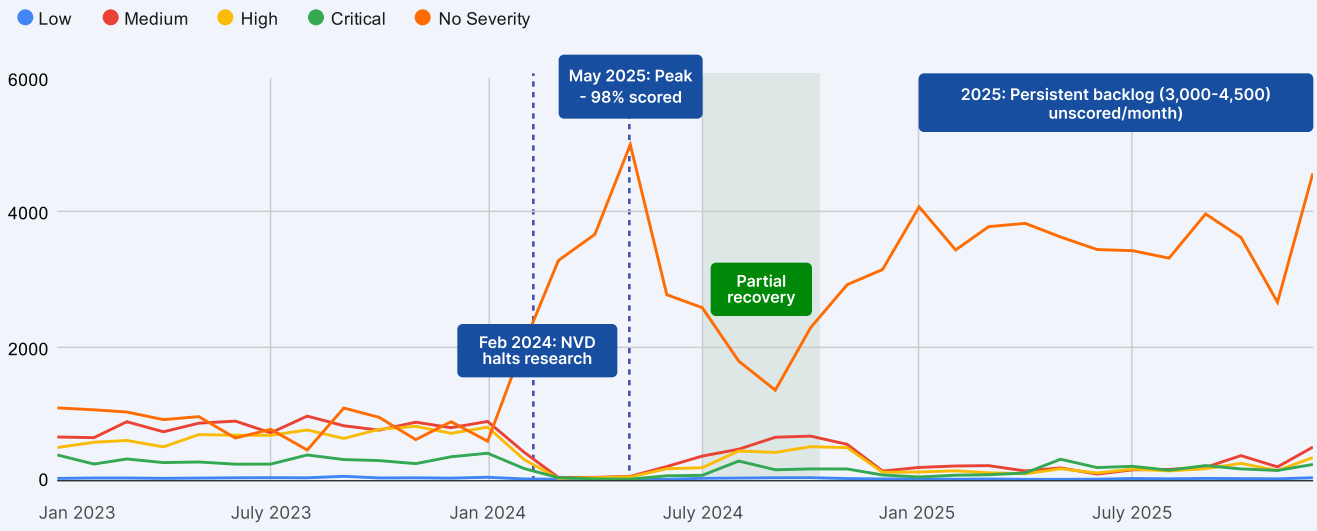


Figure 11. Monthly CVEs by severity, scored at time of publication month - NVD Primary (NVD + ADP) scores only (National Vulnerability Database)

In last year's report, we highlighted the significant disruption to NVD's CVE analysis pipeline that began in February 2024, when NVD paused its research efforts amid restructuring and budget cutbacks. At the time, this created a massive backlog of unscored CVEs: by May 2024, 98% of CVEs published that month had no severity score at all, and prompted NVD to contract CISA as the first Authorized Data Publisher (ADP) to help clear the queue.

A year later, the data shows that while CISA's involvement brought a partial recovery in mid-2024 (with scored CVEs rebounding through the summer), the improvement was short-lived. Throughout 2025, the backlog has remained stubbornly high, with 3,000 to 4,500 CVEs per month going unscored at publication time, far worse than the 2023 baseline of 600 to 1,100. One contributing factor is the surge in CVE submissions, potentially driven in part by AI-generated vulnerability reports flooding the pipeline, which continues to outpace NVD's analysis capacity.

Among the CVEs that do receive scores, the overall trend in severity distribution remains consistent with prior years: Medium and High severity CVEs continue to dominate in volume, Low severity CVEs remain scarce, and Critical severity CVEs fall between the two extremes. However, the large and persistent “No Severity” category means that the true severity landscape is only partially visible at any given time, making it more important than ever for organizations to **look beyond NVD scores when prioritizing vulnerabilities.**

NVD's CVSS scoring model evaluates the **theoretical severity** of a successful exploit, not the likelihood of exploitation in any given environment. The JFrog Security Research team re-rates each HPCVE it analyzes based on **real-world exploitability**, contextual factors, and actual attack complexity. In 2025, the gap between NVD scores and JFrog's assessment widened further.

Of the 248 HPCVEs analyzed, 113 had NVD-assigned severity scores, up from 57 in 2024. The remaining 54% had no NVD score at all, and JFrog rated these independently.

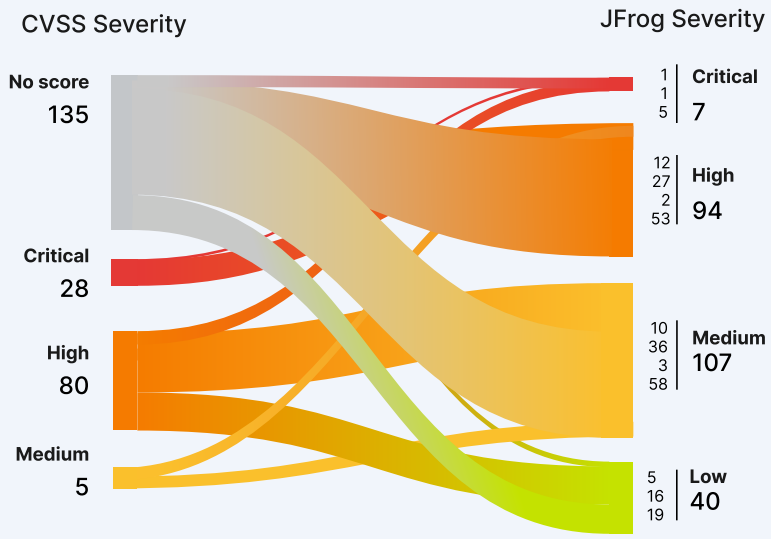


Figure 11.1 % NVD CVSS scores versus JFrog Security Research severity ratings (JFrog database, 2025)

The divergence between NVD and JFrog severity ratings continued to grow. Of the 28 CVEs rated Critical by NVD, JFrog confirmed only one, CVE-2025-1550, as Critical, meaning 96% were downgraded, up from 88.2% in 2024. Sixty-five percent of NVD High ratings were also downgraded, up from 56.8% in 2024.

This pattern is not new, but it is accelerating. An ever-increasing volume of CVEs rated Critical or High by NVD are, on closer analysis, significantly less exploitable in practice. Security teams relying solely on NVD scores to prioritize remediation are working from an **increasingly inflated picture of risk**, spending time on vulnerabilities that are unlikely to be reachable in their environments while potentially deprioritizing those that are.

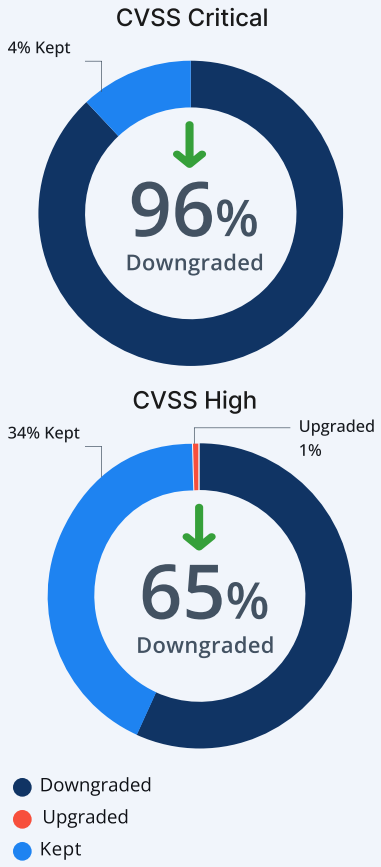


Figure 11.2

Your remediation queue may be longer than it needs to be. JFrog Security Research found that 96% of NVD-rated Critical CVEs and 65% of High CVEs **carry less real-world risk than their scores suggest.**

Applicability ratings of high-profile CVEs

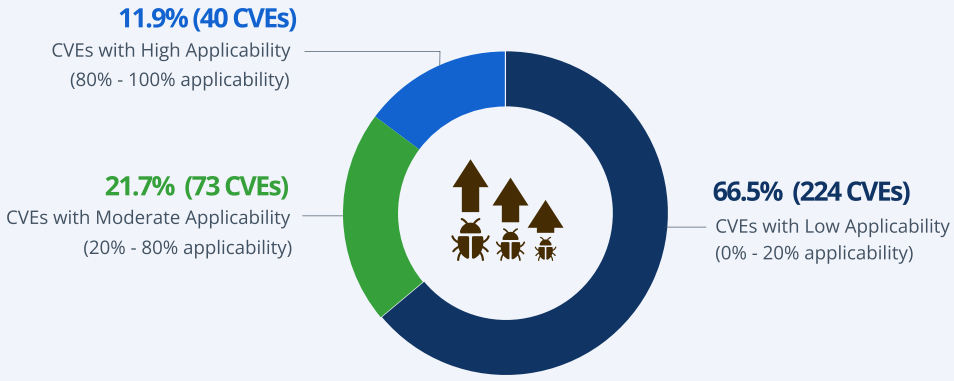


Figure 11.3. Applicability Ratings of 2025 CVEs as Seen in Xray Customers' Scan Results
(Proprietary JFrog Security Research using CVE and JFrog databases)

Simply assigning severity scores to CVEs is insufficient for assessing the impact of a vulnerability on a specific software product. JFrog Security Research goes beyond assigning JFrog Severity scores; the team also evaluates **the conditions that affect the exploitability** of these vulnerabilities in practice. To this end, JFrog creates applicability scanners that determine whether the criteria for exploitability are met in a particular software product it scans.

The JFrog Security Research team created applicability scanners for 337 CVEs published in 2025 (CVE-2025-*), focusing on High and Critical CVEs from the most popular components and technologies among JFrog customers. Only **40 CVEs (11.9%) were found to be highly exploitable**, with an applicability rate greater than 80% in artifacts scanned by JFrog Xray. By contrast, **224 CVEs (66%) were found with a low exploitability rate** and an applicability rate of 0%–20%.

CVE-2025-66478 (React2Shell) in Next.js was one of the most notable examples of high applicability; 96% of the software artifacts that Xray flagged with this CVE were found applicable, meaning the exploitability conditions are met in nearly all of them. This vulnerability can be triggered through a specifically crafted malicious HTTP request, which, when deserialized by React, enables remote code execution on the server. Its high applicability rate reflects a straightforward reason: the default configurations of React and Next.js are vulnerable. Creating a Next.js app with the standard `create-next-app` command results in a vulnerable application.

At the other end of the spectrum, **CVE-2025-32988** was among the least applicable CVEs, with fewer than 0.1% of scanned artifacts deemed applicable. This double-free vulnerability in GnuTLS requires an attacker to supply user input to specific GnuTLS APIs and pass a user-controlled Object Identifier (OID) argument. This is a highly improbable scenario in practice, as OIDs are typically fixed and cannot be controlled externally.

Applicability types of high-profile CVEs

Knowing that a vulnerability exists in a component is only the starting point. The harder question, and the more operationally relevant one, is whether it can actually be exploited in a given environment. JFrog Security Research approaches this by going beyond traditional code reachability analysis, which only asks whether vulnerable code can be called. Real-world exploitability also depends on application configuration and the conditions present in the underlying operating system. Missing either factor leads to misclassification in both directions: flagging risk that isn't there, or missing risk that is.

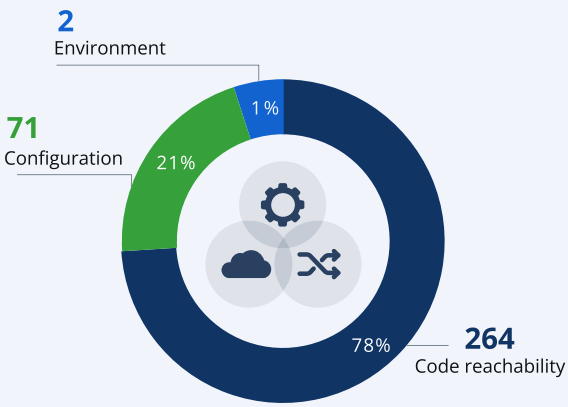
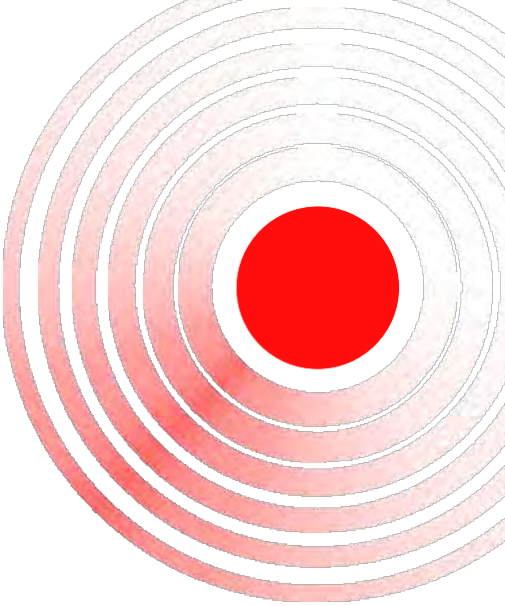


Figure 11.4. Applicability types of CVEs in 2025 (Proprietary JFrog Security Research using CVE and JFrog databases)

JFrog classifies applicability into three types. The majority, 78% of 2025 HPCVEs, are **code-reachability-based**, meaning exploitability can be determined by analyzing the code itself through call reachability or static analysis. A further 21% are **configuration-based**, where exploitability depends on specific application or library configuration settings rather than code paths alone. The remaining 1% are **environment-based,** where exploitability depends on conditions in the underlying operating system or runtime.

CVE-2025-32988, the GnuTLS double-free vulnerability covered earlier in this section, is a good illustration of code-reachability-based applicability; fewer than 0.1% of scanned artifacts were deemed applicable, because exploitation requires passing a user-controlled OID argument to specific APIs, a condition that almost never exists in practice. Configuration-based applicability is well illustrated by CVE-2025-32462 that was published last year for the sudo package, where exploitability depends entirely on a non-default entry in the sudoers configuration file, invisible to any code reachability analysis.

The developer tooling attack surface



The software supply chain has always been about what developers pull in. In 2025, attackers moved upstream, into the tools developers use to write the code. IDE extensions, AI coding assistants, and Model Context Protocol (MCP) servers are now standard developer infrastructure. They have access to codebases, credentials, and CI/CD pipelines. And they are an attack surface that most security frameworks were not built to address.

IDE extensions: a new ecosystem, a new threat vector

JFrog tracked AI coding assistant IDE extensions on OpenVSX for the first time in 2025. The OpenVSX registry, used by AI-native IDEs including Cursor, VSCodium, and others, grew from approximately 1,000 extensions in 2023 to 3,803 in 2025, a 262% increase. A growing ecosystem is an increasingly attractive target.
In 2025, 56 malicious extensions were detected on OpenVSX, the first time this attack surface has been tracked in this report. The highest-profile was GlassWorm, the first self-propagating worm targeting VS Code extensions, discovered by Koi Security in October 2025. GlassWorm compromised seven extensions by hiding malicious code inside invisible Unicode characters, harvested developer credentials to spread autonomously, and deployed a remote access trojan on infected machines, reaching approximately 35,800 installations.

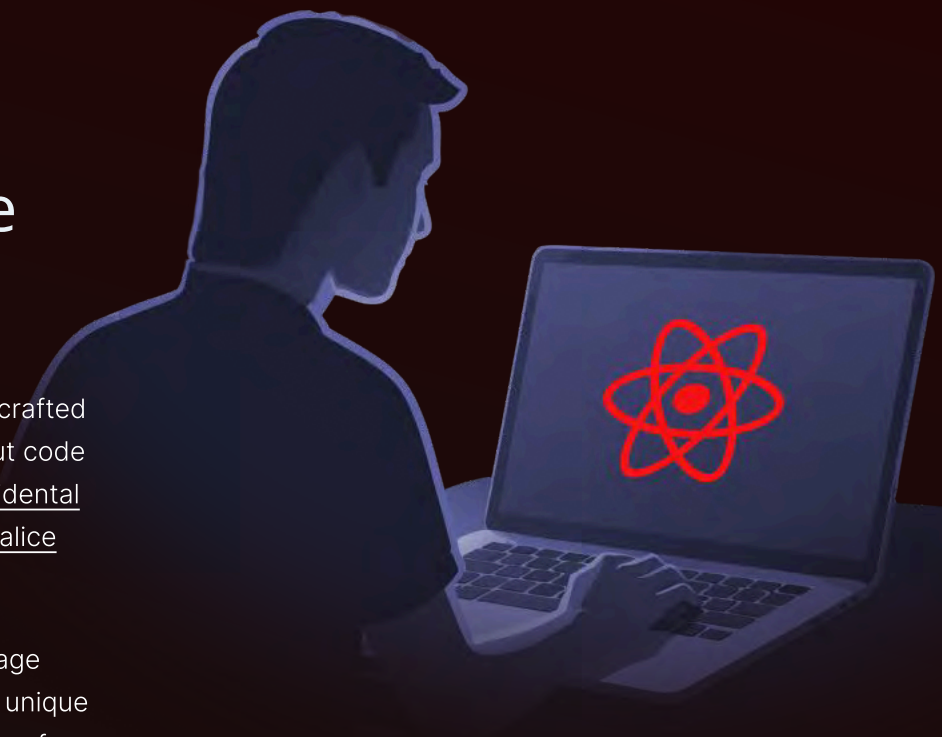
MCP servers: the agentic layer is now a vulnerability surface

MCP servers give AI assistants access to tools, APIs, databases, and local systems, making them a high-value attack surface. In 2025, JFrog identified 20+ critical remote code execution (RCE) vulnerabilities across various MCP servers, including CVE-2025-6514, a CVSS 9.6 RCE in mcp-remote, published in July 2025.
In early 2026, JFrog extended its scanning to AI agent skill registries and identified 969 malicious AI agent skills carrying critical-impact payloads, confirming that the agentic attack surface expanded beyond traditional package registries and IDE extensions into the skills and tools that power autonomous AI workflows.

CVE-2025-11953: critical RCE puts millions of React Native developers at risk

The most significant developer-targeted finding was CVE-2025-11953, a remote code execution vulnerability with a CVSS score of 9.8 affecting React Native developers. Disclosed by the JFrog Security Research team and published in November 2025, the vulnerability put millions of React Native developers at risk. The attack was trivially simple to execute and developer machines were vulnerable by default, even to attackers outside the local network.

Some malicious packages are worse than others



This section is about packages deliberately crafted to harm: not CVEs, not misconfigurations, but code with a payload. The distinction matters. Accidental vulnerabilities can be patched; intentional malice requires a different kind of defense.

npm leads all ecosystems in malicious package volume, and by a significant margin. 171,592 unique malicious packages were detected in 2025, up from 31,132 in 2024 (+451%). **RubyGems** surged from 190 to 785 (+313%), and **Hugging Face** entered the tracked set for the first time with 495 malicious models. By volume, npm dwarfs every other ecosystem combined.

Volume is the wrong measure of danger.
The most damaging campaigns were also the smallest by package count.

Campaign comparison

Campaign	Packages	Real-world impact
Qix	25	2.5M+ compromised downloads
s1ngularity	8	83,000 secrets leaked
Shai-Hulud (1+2)	~1,000	Credential harvester, DNS hijacking, destructive wiper
Big Red	~80,000	No direct user payload — potential dry run

Eight packages leaked 83,000 secrets. Twenty-five packages generated 2.5 million compromised downloads. The Big Red campaign published approximately 80,000 packages and caused neither. **The takeaway: package count is the least useful measure of supply chain risk.**

On a positive note, NuGet showed the strongest ecosystem improvement of the year, down 97% from 671 malicious packages in 2024 to just 19 in 2025, possibly due in part to Microsoft's ongoing efforts to improve malicious package detection and prevention within the NuGet ecosystem.

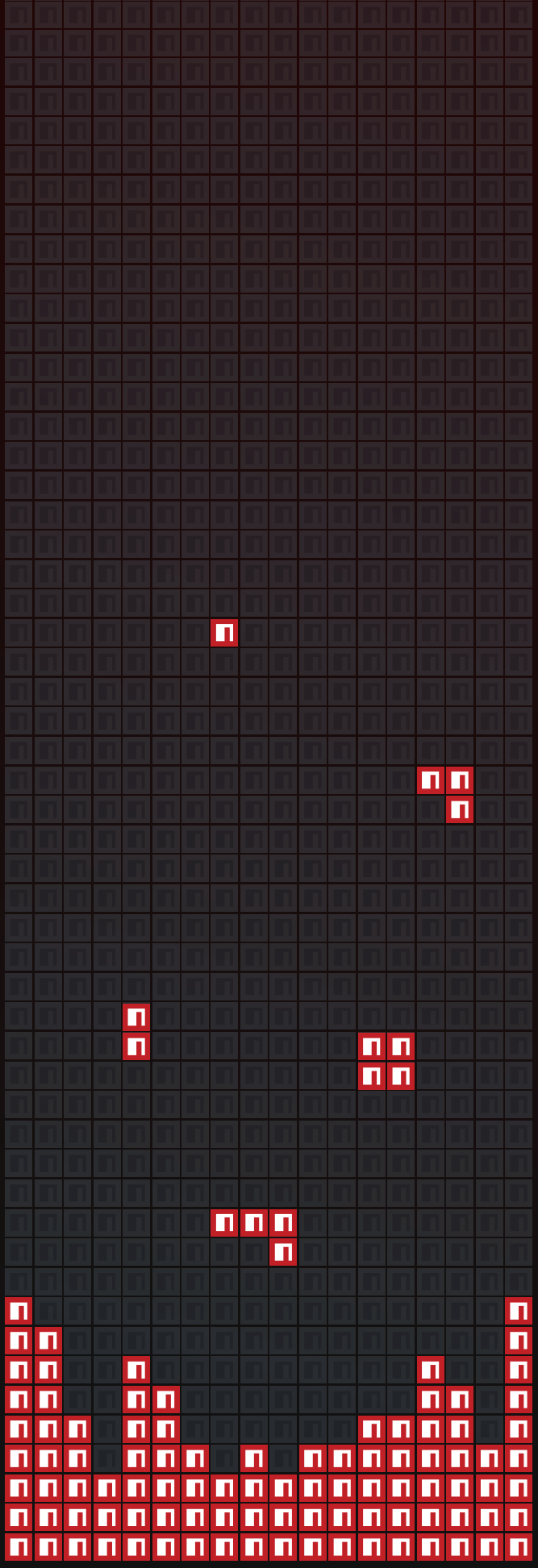
Misconfigurations, mistakes, and the impact of human error

The **s1ngularity** attack exploited a dangerous combination of GitHub Actions misconfigurations. The first was the use of the pull_request_target trigger, which allowed a workflow from a malicious fork to run with elevated repository permissions and access to secrets. The second was a command injection introduced by echoing an unsanitized PR title — specifically, ${{ github.event.pull_request.title }} — directly into a shell script.

Because the GitHub Token was set to read/write rather than the recommended read-only, the attacker used the injected command to programmatically trigger a separate workflow_dispatch event. That secondary workflow had access to the repository's npm publishing token, which the attacker exfiltrated to publish malicious packages under the project's scope.

The s1ngularity attack didn't require a zero-day or sophisticated malware. Just a misconfigured workflow and a token with more permissions than it needed. These are not exotic conditions. They are common ones. CI/CD pipelines are now part of the attack surface, and misconfigurations in them carry the same consequences as vulnerabilities in the code they build.

Eight packages and one misconfigured workflow were enough to leak 83,000 secrets.



State of leaked secrets in binary artifacts

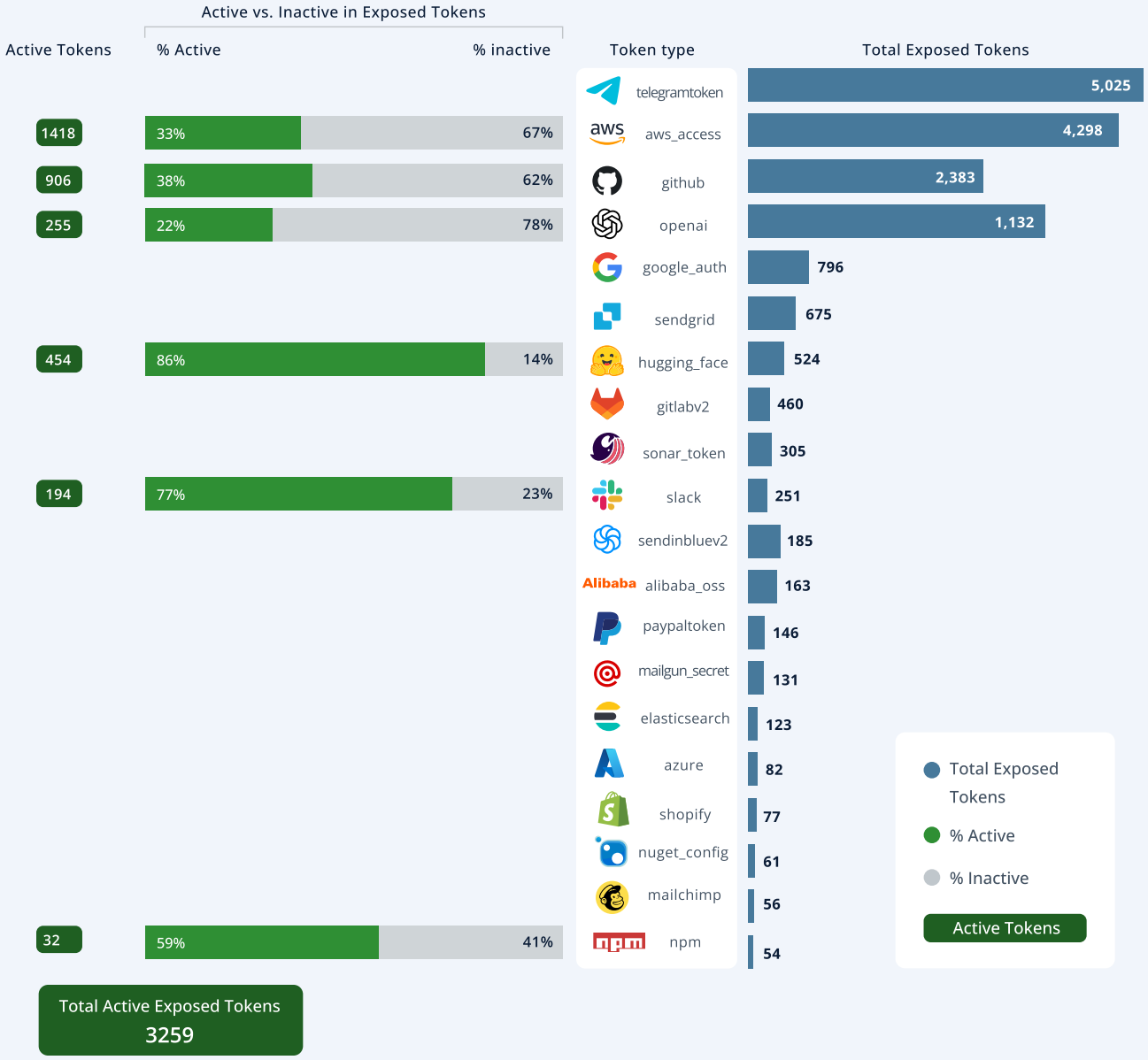


Figure 12. Top 20 most exposed token types in 2025 (Proprietary JFrog Security Research Data)

The JFrog Security Research team scanned public binary repositories for exposed secrets: tokens, API keys, and credentials left in artifacts, container images, and build outputs. In 2025, 17,637 exposed tokens were found across tracked ecosystems. Of those, 3,260 were verified as still active at the time of data collection, an overall active rate of 18.5%.

The headline number understates the risk. The active rate varies dramatically by token type, and the tokens most likely to still be active are also the ones with the highest potential impact.

AWS credentials represent the highest-volume active threat. Of 4,298 AWS access tokens found, 1,418 were confirmed still active (a 33% active rate). A live AWS credential gives an attacker access to cloud infrastructure, storage, compute, and data.

GitHub tokens are both common and consequential: of 2,383 found, 906 were confirmed active (38%). A live GitHub token can provide access to source code repositories, CI/CD pipelines, and secrets stored in Actions, making it one of the most significant credential types in a software supply chain context.

Hugging Face tokens had the highest active rate of any token type found at meaningful volume: 524 tokens found, 454 confirmed active (87%). Nearly nine in ten exposed Hugging Face tokens were still live at the time of discovery, consistent with last year's finding of approximately 85% active. As organizations increasingly store proprietary models and datasets on Hugging Face, an exposed token provides access to those assets and, potentially, the ability to modify or poison them.

Slack tokens showed a similarly high active rate: of 251 found, 194 were active (77%), providing access to internal communications, shared files, and integrated workflows.

OpenAI API keys represent a growing category of risk. Of 1,132 found, 255 were confirmed active (23%). An exposed OpenAI key enables unauthorized API usage at the victim's expense, but more significantly, it may provide access to fine-tuned models, stored conversations, and organizational AI workflows.

Telegram was the single largest token type by volume, with 5,025 found, but 0% were verified active. These are likely expired or revoked bot tokens: high volume, negligible actual risk.

The pattern across all token types points to **the same root cause**: secrets are being committed to binary artifacts and build outputs, then published to repositories where they persist long after the credential was created. The organizations most exposed are those without automated scanning to detect and revoke exposed credentials before they are discovered by someone else.

SECTION 2

Key takeaways

Volume is not always the best measure of risk.

In 2025, the numbers were record-breaking across almost every category. But size and danger were not correlated. Qix used 25 packages to generate 2.5 million compromised downloads. S1ngularity used 8 packages to leak 83,000 secrets. Big Red published 80,000 packages and achieved neither. The lesson is consistent: the most dangerous supply chain events of the year were not the largest ones — and organizations that triage by volume alone will consistently prioritize the wrong threats.

The infection point has moved up the stack.

Attackers are no longer waiting for developers to pull a malicious package from a registry. They are compromising CI/CD pipelines, model registries, IDE extensions, and even Agent Skills, confirming the attack surface continues to move upstream into the tools that power autonomous workflows.

AI-generated code is already visible in the vulnerability data.

The top three CWE categories in 2025 are XSS, SQL Injection, and Injection — classic web application weaknesses that have been understood and preventable for decades. Their simultaneous resurgence is a direct consequence of AI-assisted development at scale, producing code without applying basic secure coding practices. The CVE growth rate is not just a function of more software being written. It is a function of how that software is being written.

NVD severity scores are becoming less useful as a triage tool.

96% of CVEs rated Critical by NVD were downgraded by JFrog in 2025, up from 88% in 2024. Only one CVE was confirmed Critical by both. Security teams that use NVD scores as their primary prioritization mechanism are spending remediation resources on vulnerabilities that pose little practical threat, while the gap between theoretical and real-world severity continues to widen.

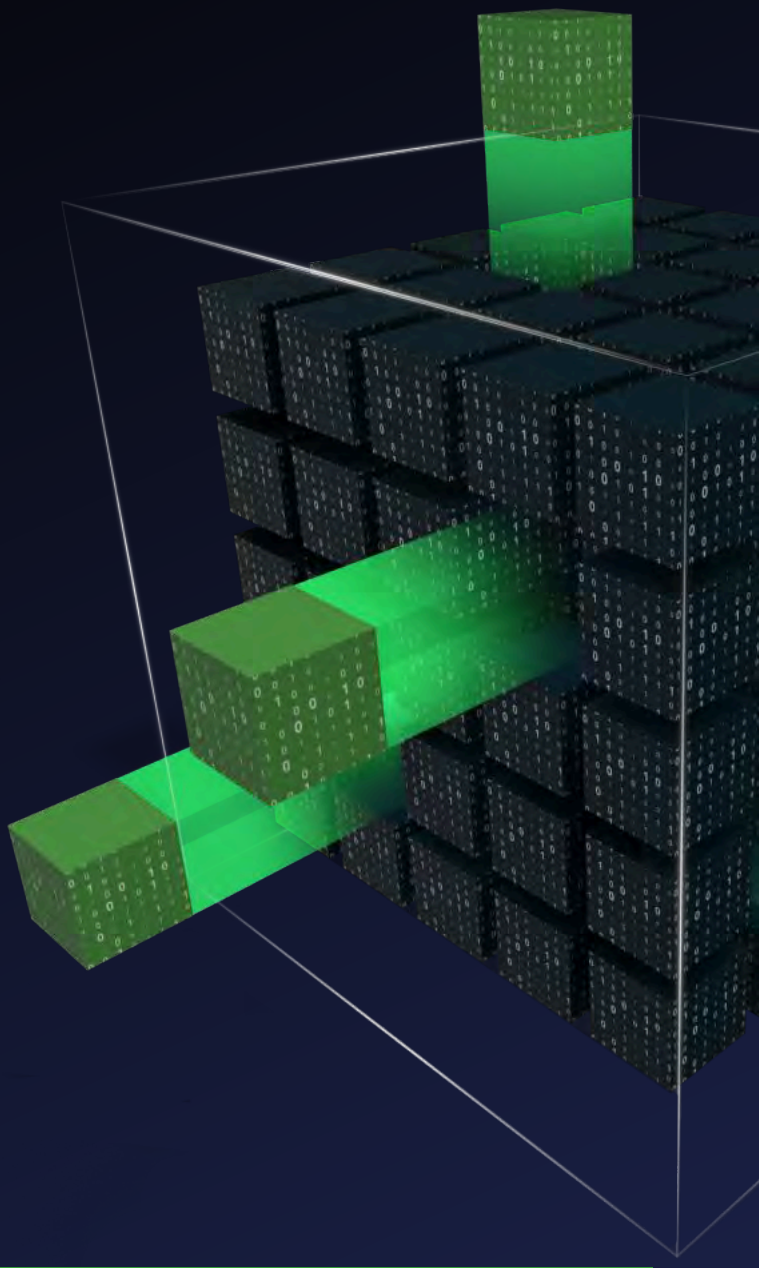
SECTION 3

How Organizations **Apply Security Efforts Today**

The picture that emerges this year is one of an industry caught between the infrastructure it built and the reality it now faces. Across more than 1500 security, DevOps, AppSec, and engineering professionals in North America, Europe, and South Asia, the pattern is consistent: the tools, frameworks, and governance processes organizations have invested in were not designed for this new layer or complexity.

AI-generated code, agentic developer tooling, and model artifacts are landing in pipelines that were built around traditional open-source dependencies.

Defenses are out of date.



1. Sourcing restrictions

Sourcing restrictions were built for package ecosystems. The supply chain has evolved to include model registries and IDE extensions; however, most sourcing controls haven't caught up. The same developer who can be blocked from downloading a suspicious npm package can freely install a malicious IDE extension, connect to an ungoverned MCP server, or pull a weaponized model from Hugging Face. The perimeter expanded. The controls haven't.

Sourcing and governance of open-source components

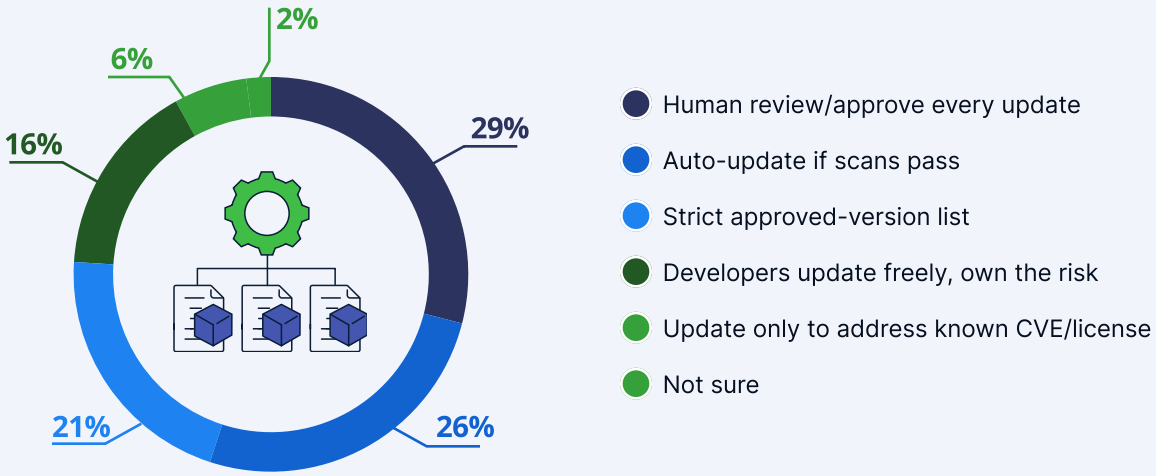
Q What approaches does your organization take towards curating or controlling open-source packages and other software components from public registries?

Six percent of respondents say their organization has no controls at all. Given that this report documents 171,592 malicious npm packages in 2025 alone, that is an indefensible position. A further 34% rely on a recommended policy and developer self-governance without enforcement. A policy without enforcement is an assumption. In a year where trusted packages like Chalk and Debug were hijacked and downloaded in over two million compromised versions, assumptions are not controls.

Best practice is **proxying public registries through a central artifact management solution**, a single control point before anything reaches a developer's machine. Fifty-nine percent have this via a network proxy or firewall, 51% maintain a curated approved list, and 47% use automated policy-based scanning. These numbers look healthy. The concern is that the supply chain now extends well beyond traditional package registries.

Managing version updates of packages from public registries

Q How does your organization manage the update to a new version of a package from public registries?



No single approach dominates. Twenty-nine percent require a human to review and approve each version update. Twenty-six percent auto-update as long as automated scans pass. Twenty-one percent maintain a strict approved-version list. The fragmentation itself is worth noting: **when different teams use different approaches within the same organization, the weakest link determines the actual control level.**

It is also worth noting that two of the three most common approaches, **manual review** and **static approved-version lists**, were not designed for the pace at which AI-assisted development now moves.

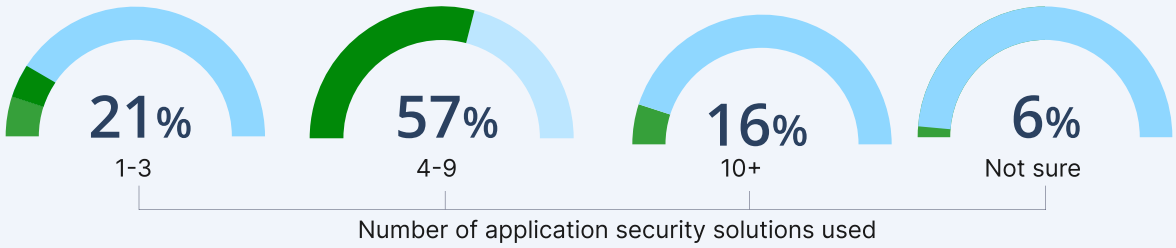
In an environment where dependencies update frequently and agentic workflows can introduce new packages without direct developer action, controls that depend on human review cycles or manually maintained lists will fall behind by design.

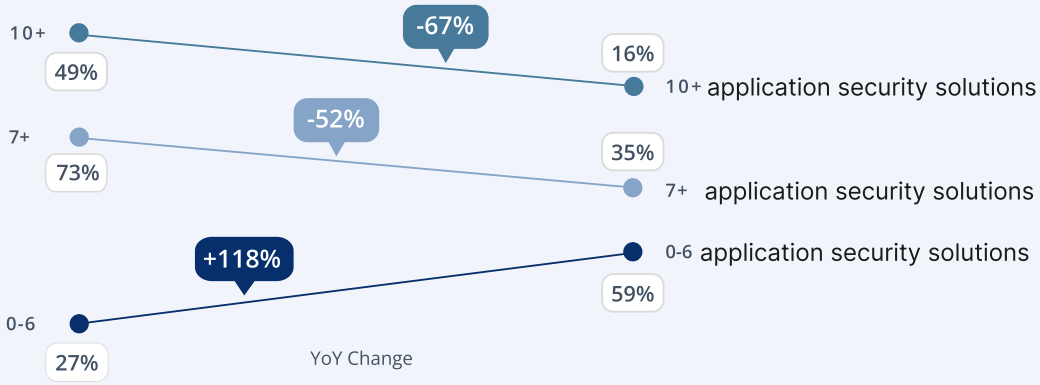
2. Scanning, scanning, scanning

The number of security tools organizations report using dropped sharply this year. Whether that reflects genuine consolidation or a shift toward platform-native security, the coverage gaps it reveals are the same.

Number of application security solutions organizations are using

Q How many application security solutions does your organization use?





Last year, 73% of respondents reported using seven or more application security solutions. This year it is 35% — a drop too large to dismiss as noise.

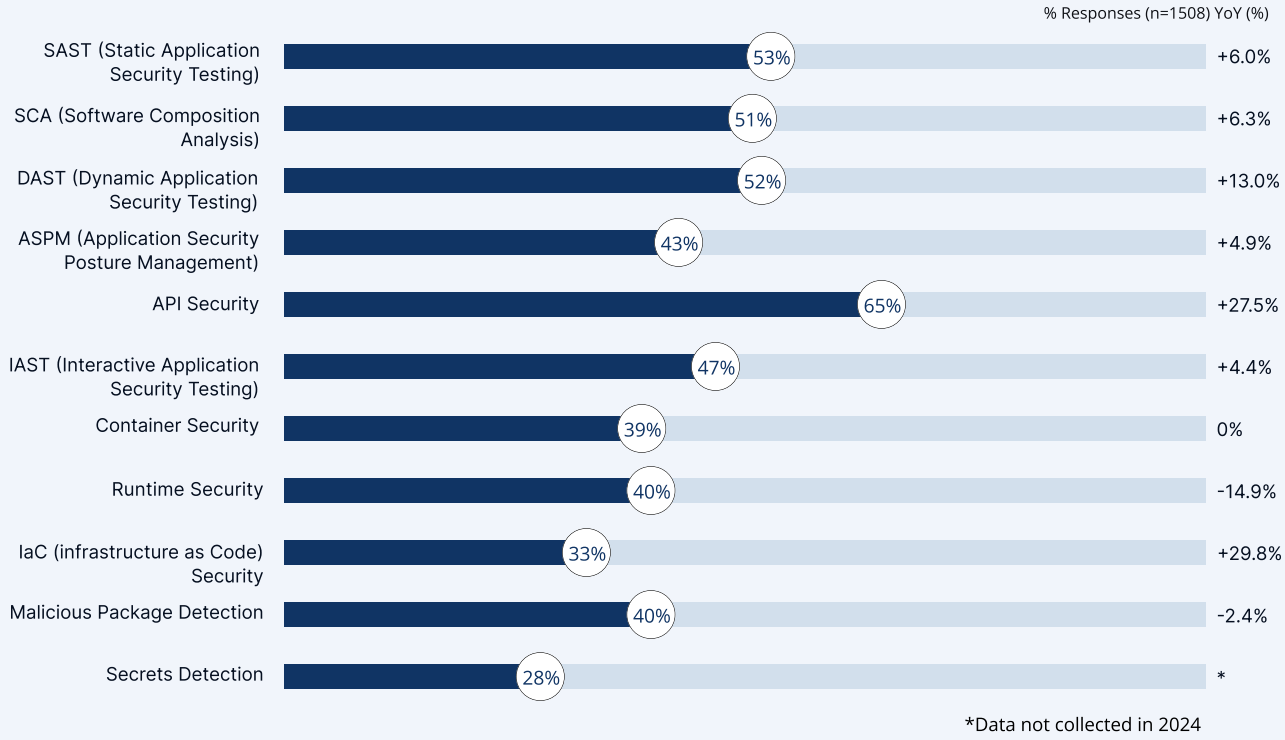
Two explanations are plausible. The first is genuine **consolidation**: organizations rationalizing their security stack, getting more coverage from fewer tools. The second is that **platform-native security** is replacing point solutions.

As we examine in detail later in this section, thirty-eight percent of respondents say they now rely primarily on the security capabilities of their existing DevOps platform. If platform solutions absorb functions that previously required separate tools, respondents may simply be counting fewer end-point solutions while maintaining similar coverage.

What is less clear is whether that coverage is actually equivalent. The tool type data covered later in this section raises questions.

Types of application security solutions organizations are using

Q What types of application security solutions are you using?

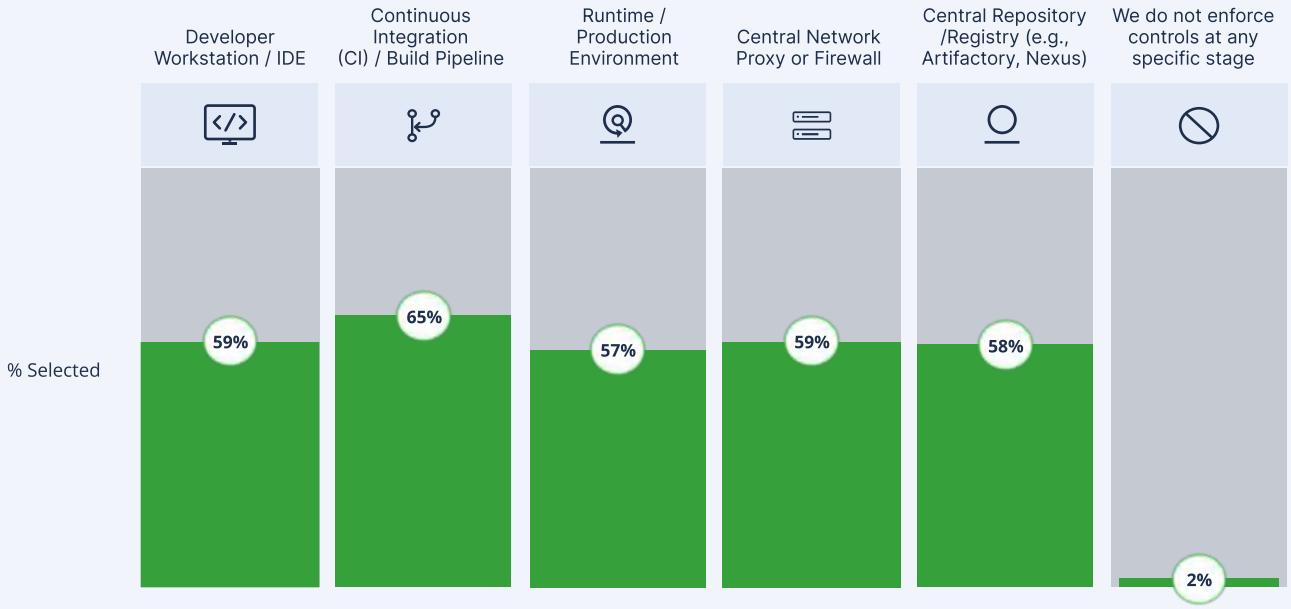


API Security leads at 65%, followed by SAST (53%), DAST (52%), and SCA (51%). These four are the closest the industry has to table stakes, and none of them break 65%.

Two numbers stand out as significant gaps. **Secrets detection** sits at 28%, the lowest adoption rate of any named category. **Malicious package detection** sits at 40%, essentially unchanged from last year's 41%, even as malicious package volume in 2025 reached an all-time high. The threat grew dramatically and yet the coverage did not move. The most rapidly growing threat categories are also the least covered, and the gap is not closing.

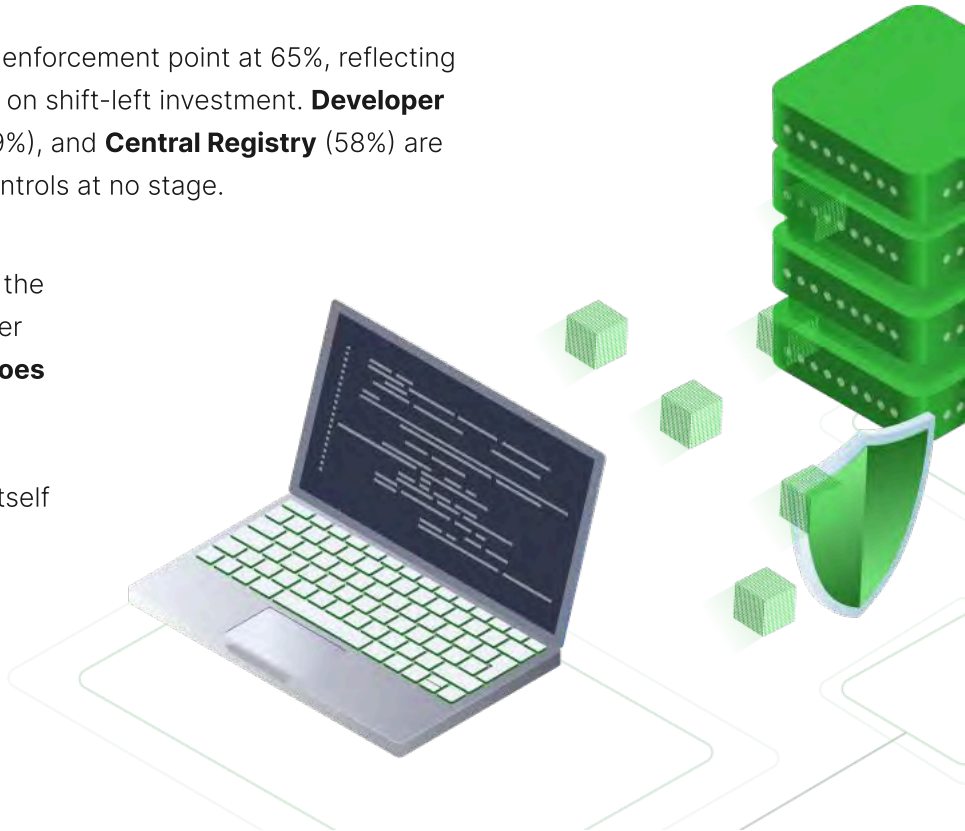
Stages at which organizations enforce security controls

Q At what stage(s) does your organization enforce security controls (e.g., blocking, scanning, approval) on open-source packages and other external components?



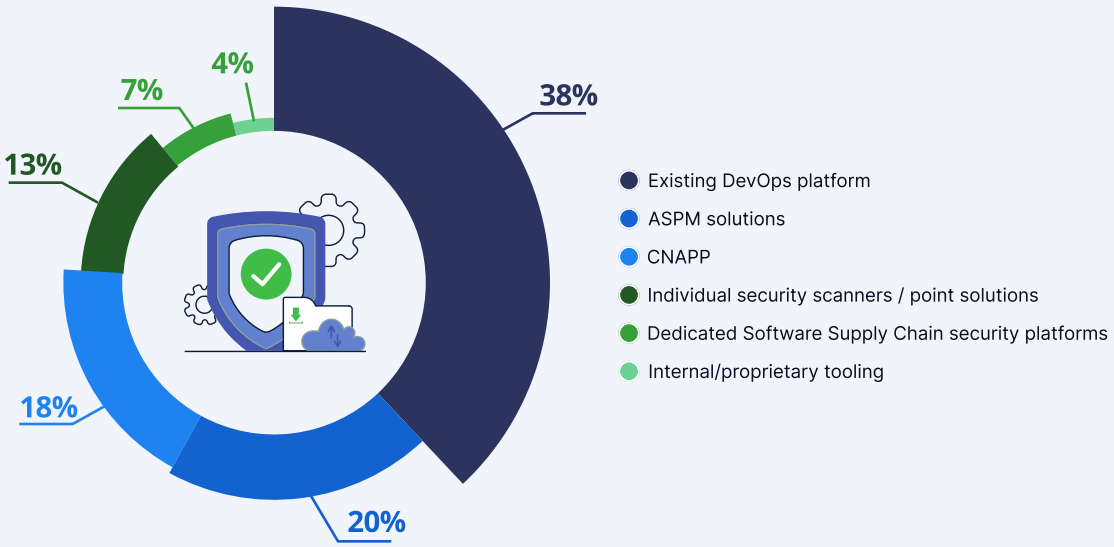
The **CI/Build pipeline** remains the primary enforcement point at 65%, reflecting the industry's long-standing concentration on shift-left investment. **Developer Workstation/IDE** (59%), **Central Proxy** (59%), and **Central Registry** (58%) are nearly tied for second. Only 2% enforce controls at no stage.

The challenge is that enforcing controls at the CI/Build stage catches risk after a developer has already referenced a dependency. **It does not prevent exposure** on the developer's machine. As the September npm hijacks demonstrated, the local environment can itself be the point of compromise.



Primary security tooling preference

Q Which class of application security solutions does your organization rely on most?



Thirty-eight percent rely primarily on their **existing DevOps platform** as their primary security capability. ASPM solutions rank second at 20%, **CNAPP** third at 18%. **Dedicated software supply chain security platforms** were cited by only 7% as their primary reliance.

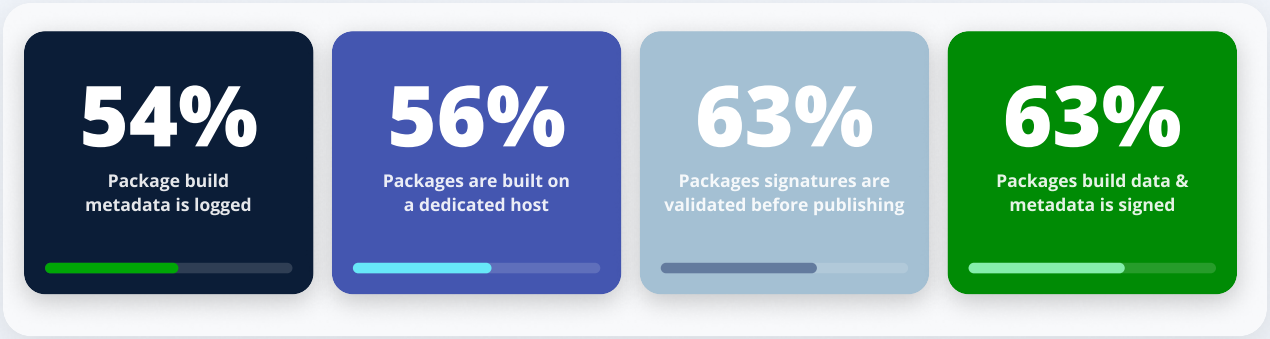
The platform-first preference is consistent with the tool consolidation picture. But not all DevOps platforms approach security the same way. The relevant question is where those security capabilities actually come from: are they purpose-built with genuine security pedigree, or are they thin wrappers around open-source tooling with limited ongoing investment?

A platform with a dedicated security research organization, continuously updated threat intelligence, and native enforcement across the software supply chain represents a meaningfully different security posture than one where security was added as a feature checkbox. Given the gaps in malicious package detection and secrets scanning noted above, **the origin and depth of a platform's security capabilities deserves scrutiny:** not just whether security is present, but how seriously it was built.

3. Establishing visibility and control across application pipelines

Security frameworks implemented in the organization

Q Which of the following security framework levels are implemented in your organization?



2.37 / 4
practices on average

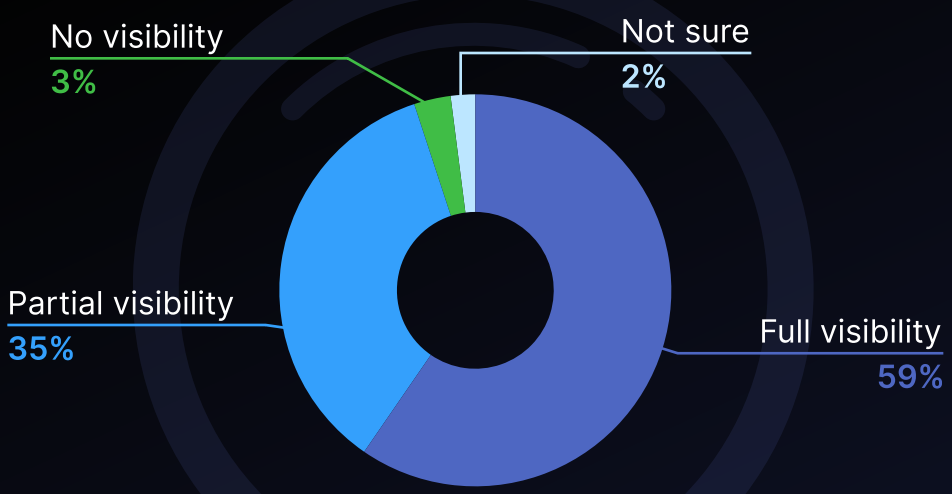
A chain of trust with gaps
The average organization implements just over half of the four practices — strong adoption of each in isolation, but consistent end-to-end coverage is rare.

Organizations that can see everything entering their pipeline, track who built what, and prove it on demand are in a fundamentally different security posture than those that can't. Supply-chain integrity frameworks like SLSA provide a structured way to harden how software is built and signed, and adoption of individual practices is reasonably strong: 63% sign their build data and metadata, 63% validate package signatures before publishing, 56% build packages on a dedicated host, and 54% log package build metadata. These are encouraging numbers in isolation.

The concern is the gap between organizations doing some of these practices and those doing all of them consistently. Each practice on its own is meaningful. Together, they form **a chain of trust, but a chain of trust with gaps is only as strong as its weakest link.**

Tracing the provenance of software running in production

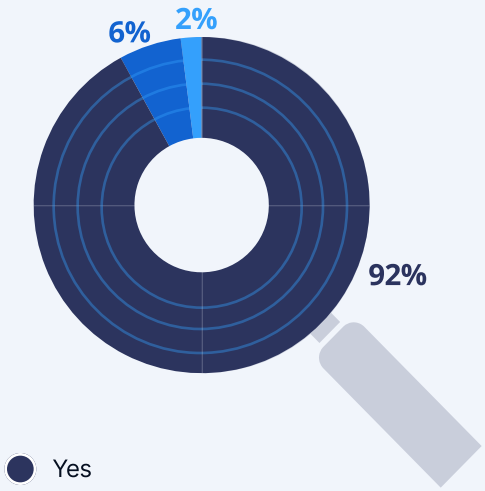
Q Do you have visibility into the provenance of software running in production?



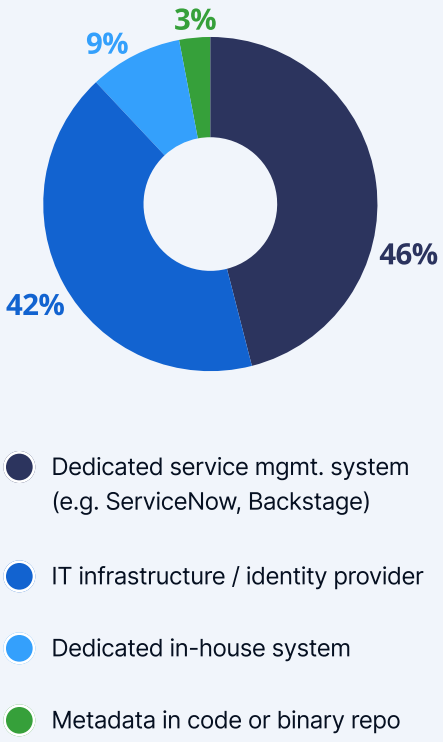
Fifty-nine percent of respondents report full visibility into the provenance of software running in production: who committed the code, what tests it went through, and where dependencies came from. Thirty-five percent have **partial visibility,** and 3% have **no visibility** at all. The partial visibility number is the one that matters most for compliance purposes. **Regulatory frameworks such as the EU Cyber Resilience Act, US Executive Order 14028, and SOC 2 increasingly require full provenance traceability**, not approximate provenance. An organization that can answer most of these questions but not all of them is in a similar position to one that can't answer them at all when an auditor asks.

Tracking the owner of each application

Q For each application you build in your organization, do you keep track of the owner of the application?



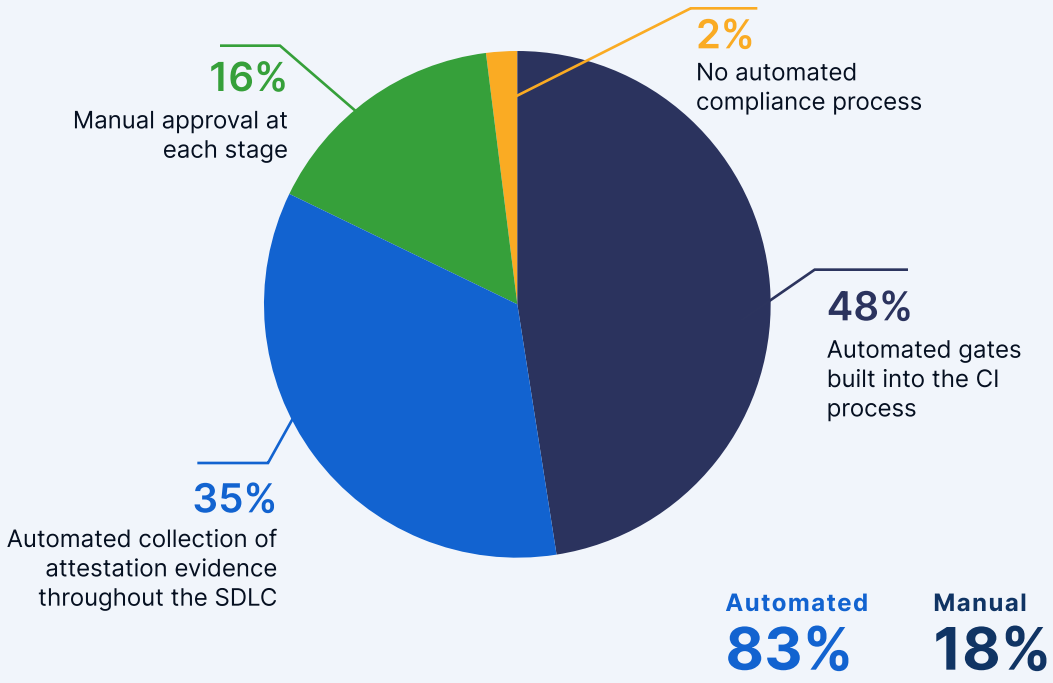
Q For each application you build in your organization, how do you keep track of the owner of the application?



Ninety-two percent of respondents say **yes**, they track the owner of each application they build, which is a strong baseline. Of those, 46% use a dedicated service management system like ServiceNow or Backstage, 42% use their IT infrastructure or identity provider, and 9% use a dedicated in-house system. Only 3% add metadata directly in the code or binary repository. The variety of methods is worth noting. Ownership tracking spread across four different approaches within an industry means there is **no dominant standard**, which makes cross-team and cross-organization accountability harder to establish.

Ensuring software quality and compliance standards

Q How do you ensure standards for software testing/quality are adhered to during the software creation and release process for compliance and governance purposes?

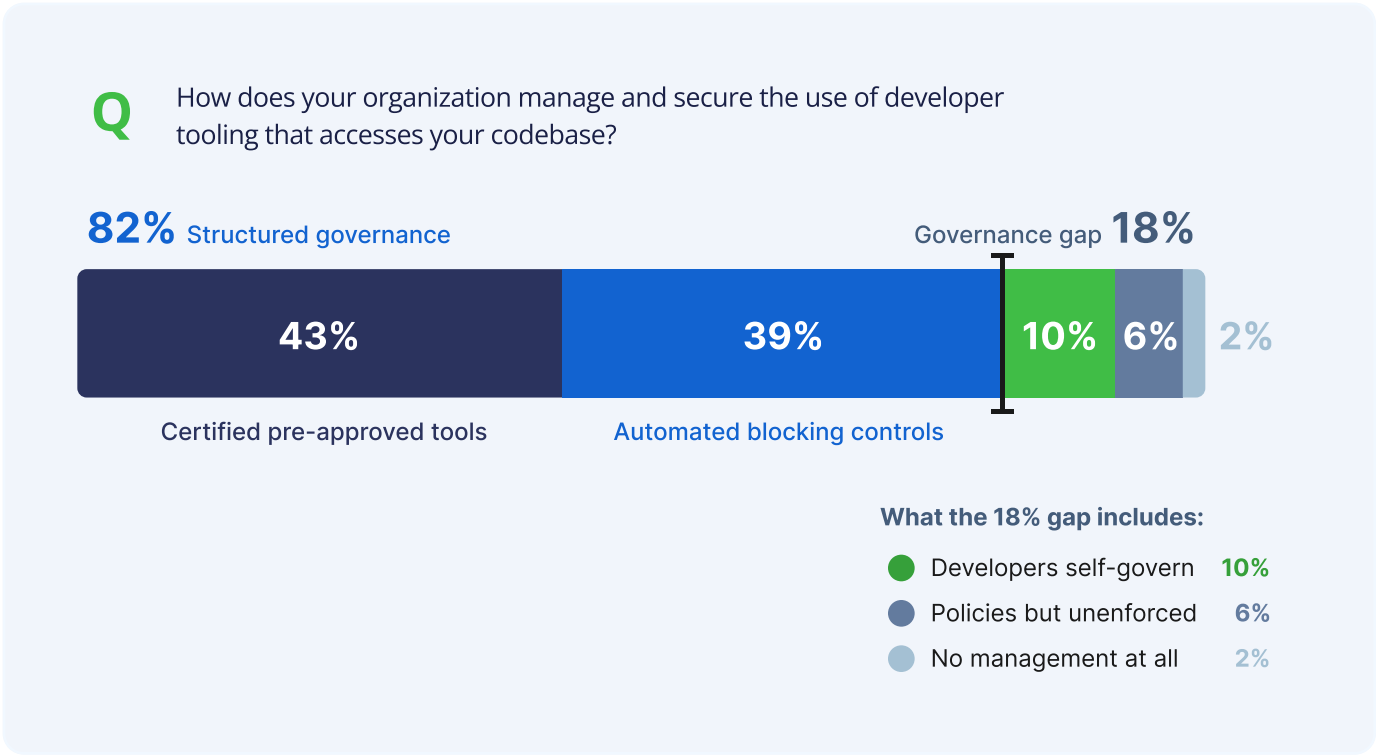


Forty-eight percent rely on **automated gates built into the CI process**, the most common approach. Thirty-five percent automatically collect attestation evidence throughout the software delivery lifecycle. Sixteen percent still rely on manual approval to advance software from one stage to the next. Any manual intervention in the promotion process is a potential point of inconsistency, an opportunity for accidental error, and a vector for intentional manipulation. At scale, manual gates slow delivery and introduce human error, and they are also the first thing that gets bypassed under deadline pressure.

4. A new governance gap: developer tooling

With 59% claiming full provenance visibility but 48% needing a week or more for audit prep, the resulting gap is worth examining. Full visibility should mean fast audit prep. The fact that it doesn't suggests that for many organizations, "full visibility" means the data exists somewhere, not that it is structured, accessible, and audit-ready on demand.

Managing and securing tooling that accesses your codebase



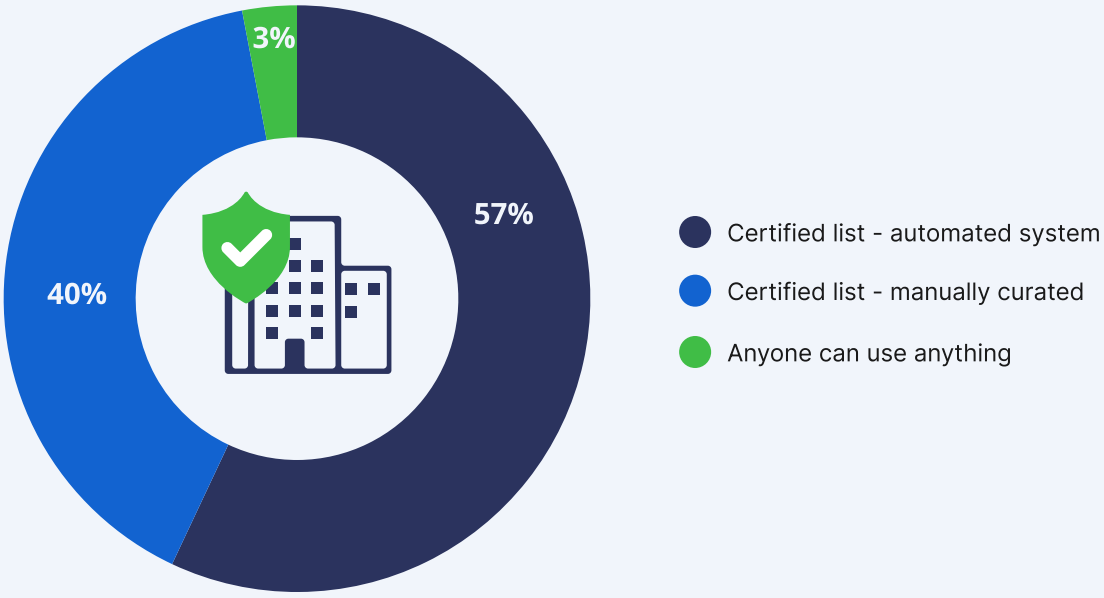
This subsection has no equivalent in last year's report. In 2024, IDE extensions, AI coding assistants, and MCP servers were not yet significant enough to survey about. In 2025, they are the **attack surface** where JFrog documented 56 malicious extensions, a CVSS 9.6 RCE vulnerability, and MCP servers with hundreds of high-risk findings. The governance data now exists to ask: how are organizations managing these tools?

Forty-three percent of organizations maintain a certified list of pre-approved tools and enforce policies on an as-needed basis. Thirty-nine percent use automated controls to monitor for and block unapproved tools. Ten percent rely on developers to self-govern their use of these tools entirely. Six percent have some policies but don't actively enforce them, and 2% have no management or restrictions at all. **That means 18% of respondents have either no governance or governance in name only for the tools sitting inside their developers' IDEs with direct access to their codebases, credentials, and CI/CD pipelines.** Given what this report documents about malicious IDE extensions and MCP server vulnerabilities, this is the clearest governance gap in this year's survey.

The 10% relying on developer self-governance deserves particular attention. Self-governance assumes developers are consistently making good security decisions about tooling when under deadline pressure, adopting new AI-assisted workflows, and while the attack surface is actively being exploited. That is a significant assumption, which could reasonably bring the “no governance” number closer to one-quarter of organizations.

Governing Model Context Protocol (MCP) usage

Q How does your organization govern Model Context Protocol (MCP) usage?

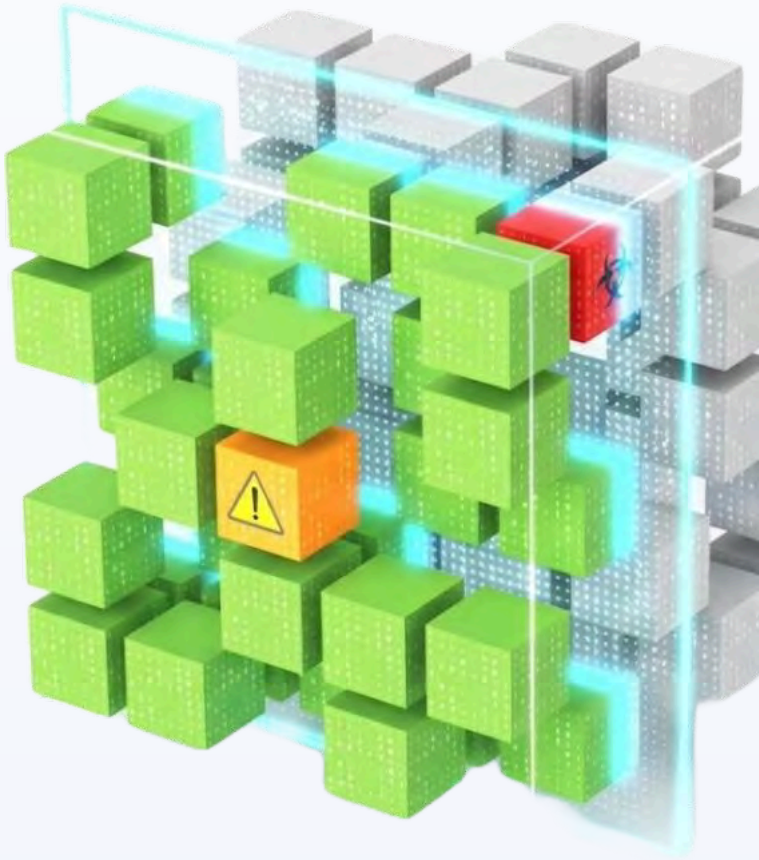


Compared to the developer tooling governance picture, MCP governance looks considerably more mature at first glance. Ninety-seven percent of respondents say they operate some form of certified list, either **manually curated** (40%) or through an **automated system** (57%). Only 3% say anyone can use whatever they want.

JFrog scanned thousands of MCP servers and utilities and found hundreds of high-risk findings, including CVE-2025-6514, an RCE vulnerability with a CVSS score of 9.6 affecting the mcp-remote utility versions 0.0.5 through 0.1.15, published in July 2025. If 97% of organizations claim certified MCP governance, the question worth asking is what that certification process actually checks. A list that tracks approved servers by name, without continuously scanning those servers for newly disclosed malicious activity and vulnerabilities, will always lag behind the threat. **Governance without scanning is a list, not a control.**

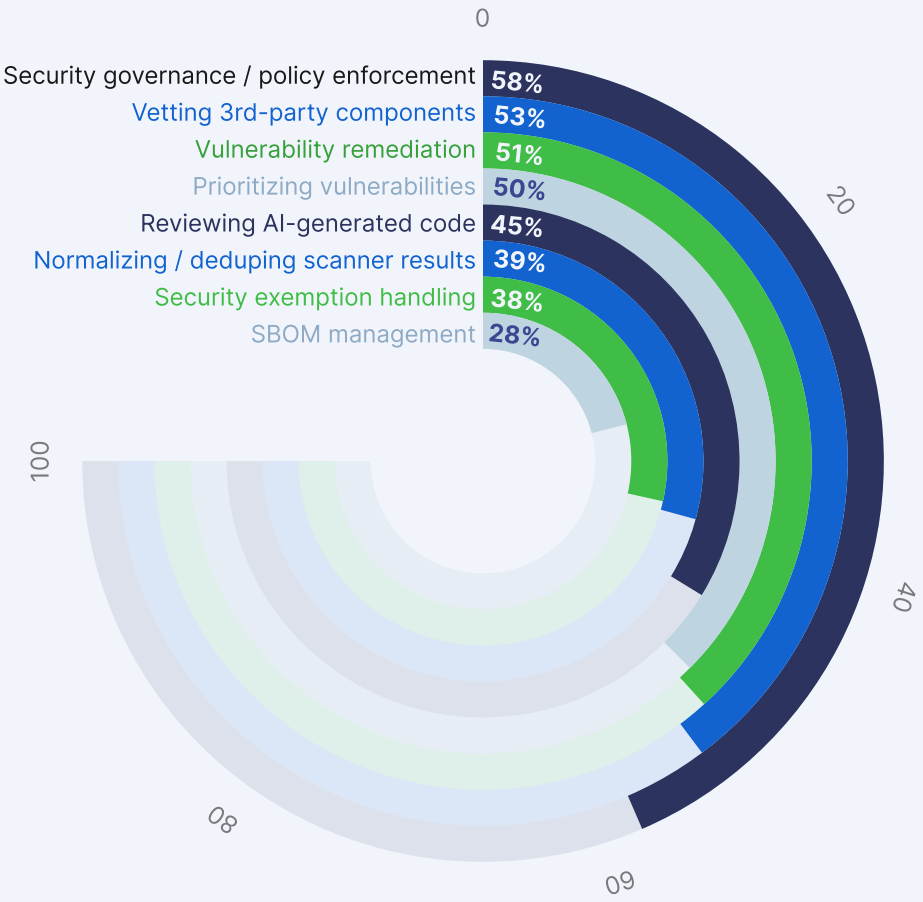
5. How much is this costing?

The security work documented in this section has always been labor-intensive, but the pace of development it needs to keep up with has accelerated sharply. AI-assisted development compresses timelines, agentic workflows introduce dependencies without direct developer action, and the volume of artifacts moving through pipelines has grown faster than any manual process can match. Security and compliance functions that depend on human review cycles, manual approvals, or hand-assembled evidence are not just inefficient, they are structurally mismatched to the environment they are meant to protect.



Top DevSecOps supply chain time burdens

Q Which software supply chain security responsibilities consume the most of your DevSecOps stakeholders' time?



Security governance and policy enforcement across the SDLC was most frequently cited at 58%. This is not a surprise, given the compliance burden documented elsewhere in this report. Vetting third-party components comes in second at 53%, and vulnerability remediation third at 51%.

New this year: reviewing and hardening AI-generated code comes in at 45%, nearly as common a time burden as vulnerability remediation. This category did not exist in last year's survey, and there are no mature, purpose-built automation tools for reviewing AI-generated code at scale. Organizations are largely doing this manually, which means it is expensive, inconsistent, and unlikely to keep pace with the volume of AI-assisted development documented earlier in this report.

Developer wait times for new OSS package approval

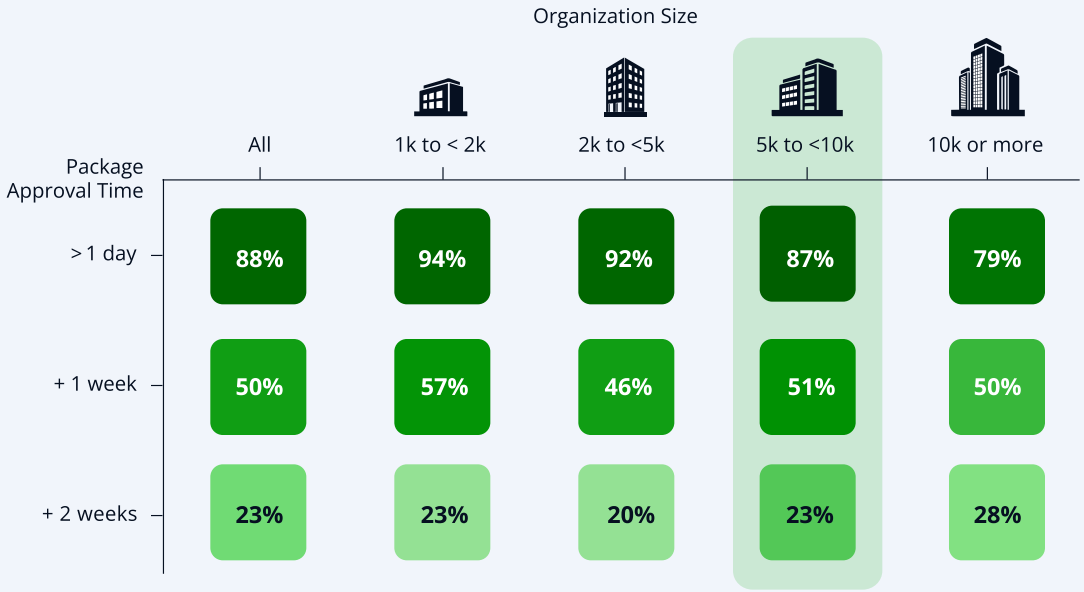
Q About how long do developers in your organization typically wait for new OSS package approval?

A week or longer

~~68%~~ (2024) → **50%** (2025) (−18%)

Two weeks or longer

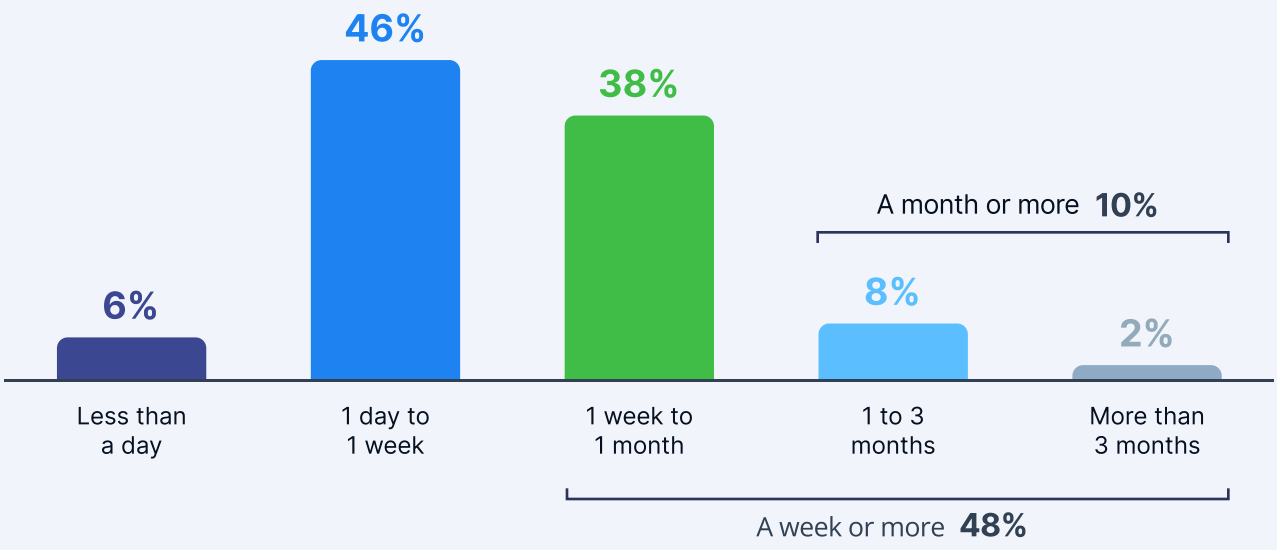
~~41%~~ (2024) → **23%** (2025) (−18%)



Here the news is better than last year. Fifty percent of respondents wait **a week or longer for a new package to be approved**, down from 68% in 2024. Those waiting two weeks or more dropped from 41% to 23%. This is a meaningful shift. It suggests that automation at the sourcing layer (e.g., automated scanning, curated approved lists, policy-based approval) is beginning to reduce the manual bottleneck. **The friction hasn't disappeared, but it is shrinking**. The goal should be to keep that trend moving: every week shaved off approval time is a week less incentive for developers to route around the process. Although as we've shown, a minimum wait of several days can actually be a good thing in order to avoid package hijack attacks.

Time required to generate proof for a compliance audit

Q On average, how long does it take your development teams to generate proof for compliance audit (e.g., SOC 2, HIPAA) per application?



59% claim full provenance visibility

but visibility ≠ audit-readiness

48% take a week or more for audit prep

Forty-eight percent of respondents take **a week or more to generate compliance audit proof per application**, 10% take a month or more, and only 6% do it in less than a day. This figure sits in tension with the 59% who reported full provenance visibility earlier in this section. The gap suggests that, for many organizations, **the data exists but isn't structured or accessible in a way that makes it audit-ready on demand**, and someone is still assembling evidence manually. That is the last mile of governance automation that most organizations haven't yet closed.

SECTION 3

Key takeaways

The perimeter expanded. The controls didn't.

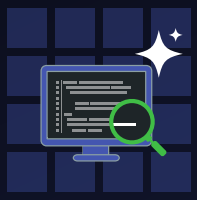
Sourcing controls were built for package registries, and for most organizations, that is still where they stop. Model registries, MCP servers, and IDE extensions operate largely outside the same governance frameworks, despite being the surfaces where this report documented active attacks in 2025. Extending sourcing controls to these new surfaces is no longer optional.

Fewer tools, but the gaps stayed the same.

The share of organizations using seven or more security tools dropped from 73% to 35%, potentially reflecting real consolidation around platform-native security. But the coverage gaps that matter most didn't close: malicious package detection is flat at 40%, and secrets detection debuted in the survey at just 28%. Consolidation is only progress if what remains covers what's actually at risk.

Governance that isn't enforced isn't governance.

Ninety-seven percent of respondents claim certified MCP governance, yet only 57% actually curate incoming MCP servers. Eighteen percent of organizations have no governance or unenforced policies for the developer tooling accessing their codebases. Fifty-nine percent report full provenance visibility, yet nearly half take a week or more to generate audit proof. Across sourcing, tooling, and compliance, the gap between reported governance and operational reality is the defining finding of this section.



Reviewing AI-generated code is now a top security burden without mature tooling to support it.

Forty-five percent of DevSecOps stakeholders cite reviewing and hardening AI-generated code as a major time consumer. While this category didn't appear in last year's survey, it's now nearly as common a burden as vulnerability remediation. Unlike vulnerability remediation, however, purpose-built automated tooling for AI-generated code review was still nascent at the time of this survey. As AI-assisted development accelerates and dedicated tooling matures, the race between the burden and the tools available to address it will define one of the most consequential DevSecOps challenges of 2026.

SECTION 4

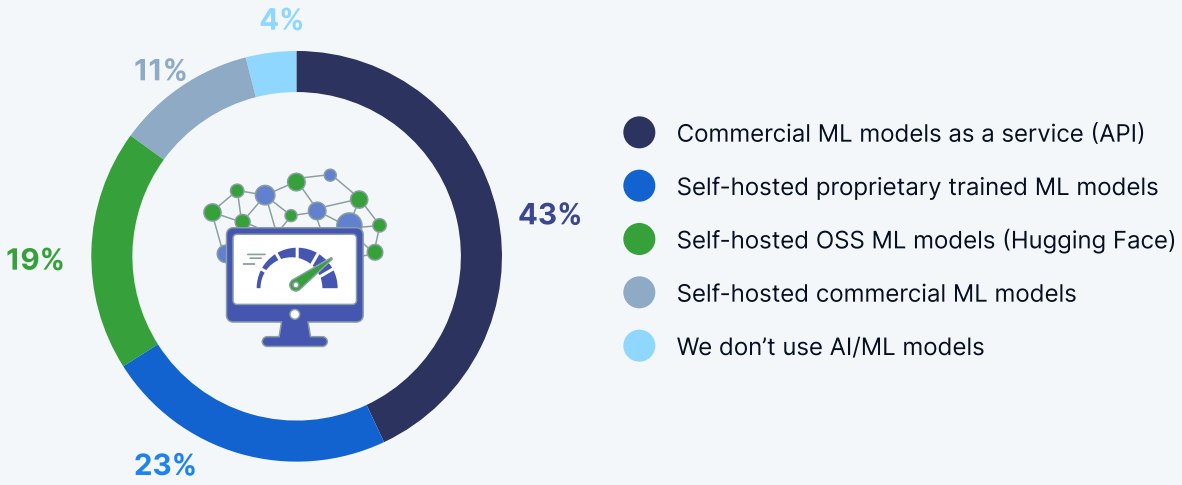
The AI Model and Agent Factors

The previous three sections document a supply chain under strain from known threats that grew bigger and faster. This final section documents something new: an emerging category of risk where the governance frameworks are being written in real time and the threat surface is expanding faster than organizations can map it.

AI is no longer a prospective consideration in the software supply chain, it is the supply chain. Models are dependencies, AI-generated code is production code, and agentic tools with access to codebases and credentials are now standard developer infrastructure. The question is no longer whether to govern AI in the supply chain, but whether governance is keeping up.

How organizations are consuming AI/ML models

Q What is your primary method for consuming AI/ML models as part of the applications you are developing?



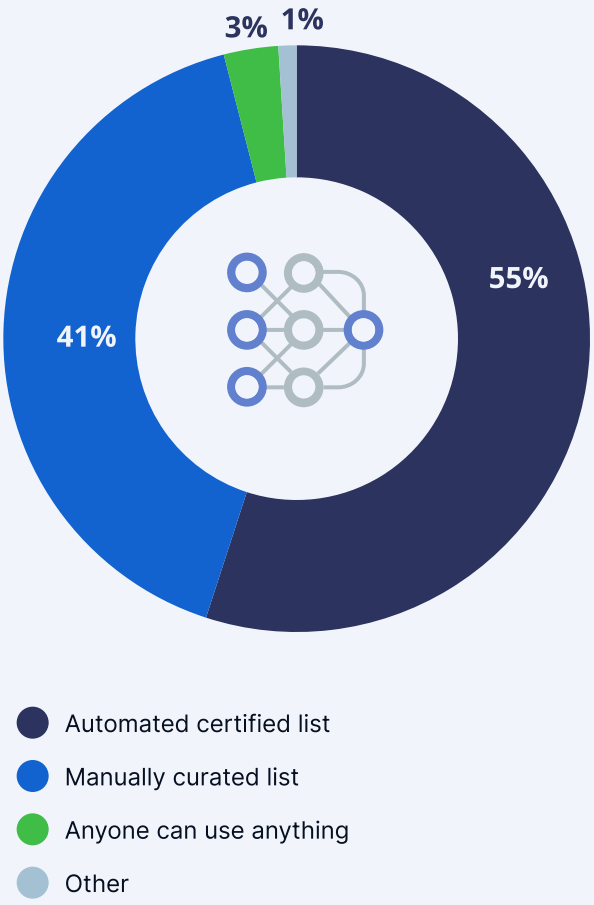
43% of respondents primarily consume AI/ML models as a **commercial API service:** OpenAI, Claude, Gemini, and equivalents. API-based consumption is generally the most governable pattern: the model never enters the organization's infrastructure, access can be credentialed, and calls can be logged and audited. That said, it is not risk-free — prompt injection and sensitive data leakage through API calls are documented attack vectors, covered later in this section.

The more complex governance challenge sits with the remaining 53% who are **self-hosting models** in some form: 23% self-host proprietary trained models, 19% self-host OSS models downloaded from registries like Hugging Face, and 11% self-host commercial models. Unlike API-based consumption, self-hosted models introduce a class of risk where the model artifact itself enters the organization's infrastructure. Like any software dependency, it can carry malicious payloads, vulnerable components, and license obligations — and requires the same governance disciplines as any other package in the supply chain.

This report documents **495 malicious models on Hugging Face** in 2025. The 19% of organizations primarily self-hosting OSS models are pulling from those same registries. Only 4% of respondents say they don't use AI/ML models at all, a figure that will likely approach zero within the next reporting cycle.

Governing model artifacts

Q How does your organization govern AI/ML model artifact usage?

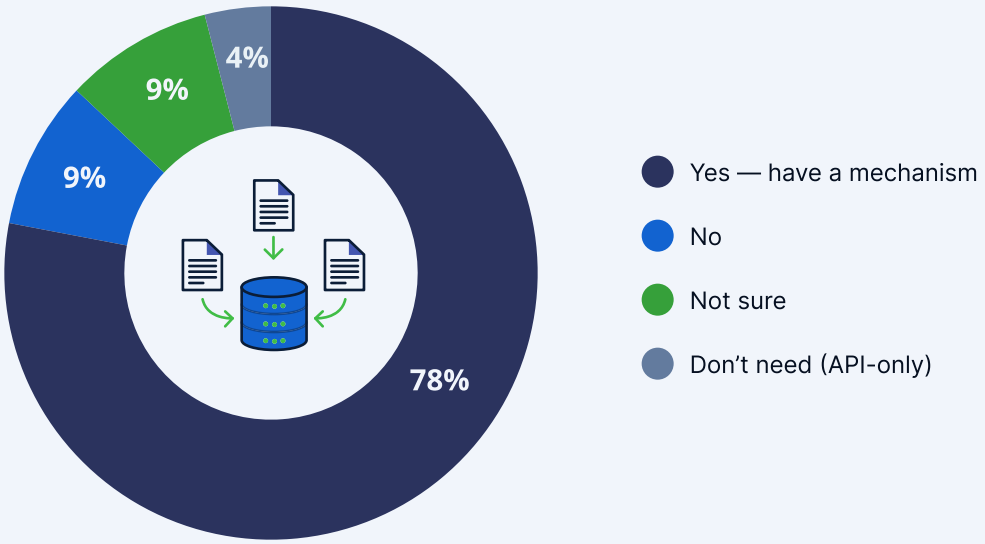


Fifty-five percent use an **automated certified list of approved models and versions**, 41% use a manually curated certified list, and 3% allow anyone to use anything they want. Taken together, 96% operate some form of certified governance, consistent with last year's 94% and suggesting that model artifact governance is maturing.

The concern, as flagged in the previous section, is what "certified" actually means in practice. **A certified list that isn't scanned for malicious payloads or vulnerabilities is a list of names, not a security control.** If those 495 malicious models on Hugging Face appeared on a certified list before their payloads were discovered, the governance framework provided no protection.

Governing AI inputs and outputs

Q Does your organization have a mechanism in place to check the inputs and outputs of your AI/ML services?



78% of respondents say yes, they have a mechanism in place to check for prompt injection, sensitive data leaks, and similar AI-specific risks. Twenty-two percent do not, are unsure, or believe they don't need it because they only use API-based models.

The 22% without input/output checking represents a meaningful gap. Prompt injection is now a documented attack vector, not a theoretical one, and sensitive data leakage through AI services has produced real incidents. The organizations relying solely on the API provider's controls have **outsourced this responsibility rather than eliminated the risk.**

Trust in AI-suggested security fixes

Q Which of the following statements most closely describes your trust regarding a security fix suggested by AI tools?

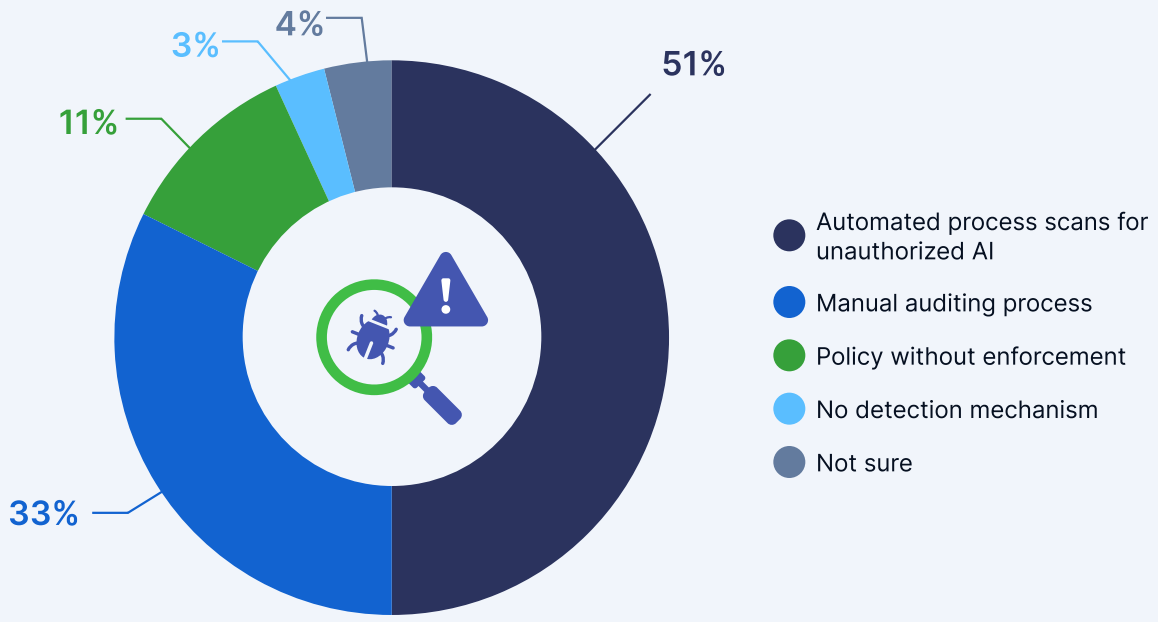
This question cuts to the heart of one of the most consequential governance questions of 2025: as AI tools suggest code fixes, security patches, and remediation steps, how critically are developers evaluating them?

23% would use the AI's suggestion as a **definitive solution**, requiring only a quick review before implementation. 63% would use it as a **strong starting point,** carefully reviewing the logic before implementing. 12% would only use it for **general direction**, writing the fix themselves from scratch. Two percent would **not use** an AI-suggested security fix at all.

The 23% treating AI suggestions as near-definitive is the number that warrants attention. AI coding tools are known to produce **plausible-looking but incorrect security fixes** which address the symptom without resolving the underlying vulnerability, or that introduce new weaknesses. A quick review is not always sufficient to catch these failures. This connects directly to the finding in the previous section that reviewing and hardening AI-generated code is now a top DevSecOps burden, and it suggests that burden is justified.

Detecting shadow AI

Q Do you have a way to detect shadow AI (unauthorized usage of AI-powered tools or services) in your development environment or applications?



Fifty-one percent have an **automated process** that scans for and flags unauthorized AI usage. Thirty-three percent use a **manual auditing process**. Together, 83% of respondents have some form of shadow AI detection.

The remaining 18% have either: a policy without detection (11%), no detection at all (3%), or aren't sure (4%). A policy against unauthorized AI that isn't actively enforced is the same dynamic as the sourcing restriction policies documented earlier in this report — an assumption, not a control.

Further, the adoption of tools and processes for shadow AI detection varied widely by region in this report, representing a large automation divide for multinational companies.

Shadow AI is the supply-chain risk organizations can't see. A developer using an unauthorized AI coding assistant may be routing code through external infrastructure, storing sensitive context in a third-party service, or generating code with dependencies that bypass every approved sourcing control. The 18% with no active detection have no visibility into any of this.

SECTION 4

Key takeaways

API-first / Headless AI consumption masks self-hosting risk.

Forty-three percent of organizations primarily consume AI via commercial APIs, which are the easiest model to govern. But 53% are self-hosting in some form, pulling models from the same registries where 495 malicious models were detected in 2025. Self-hosted models are supply chain dependencies and need to be governed as such.

Certified ≠ Secure: the illusion of AI governance.

Ninety-seven percent claim certified model governance and 78% check AI inputs and outputs. These numbers look strong. But a certified list that isn't scanned for payloads, and an input/output checker that doesn't cover the full attack surface, provide the appearance of governance more than the reality. What matters is what the certification process actually catches, and the malicious model data from this report suggests the current bar needs raising.

Twenty-three percent put too much faith in AI security fixes.

Nearly one in four respondents would treat an AI-suggested security fix as near-definitive with only a quick review. AI tools produce convincing but incorrect security fixes with enough frequency that this posture carries real risk. The 63% of respondents who see AI suggestions as strong starting points requiring careful review have the right instinct.

Shadow AI is the ungoverned edge.

Eighteen percent of organizations have no active shadow AI detection. Every developer using an unauthorized AI tool is a potential data leakage point, a sourcing control bypass, and a governance blind spot. Detection is not optional; it is the only way to know the actual scope of AI usage in an organization. How organizations detect shadow AI varies sharply by region — creating an uneven control surface for any company operating across borders.

Methodology

This report incorporates a combination of insights derived from JFrog usage data, CVE analysis results from the JFrog Security Research team, and commissioned third-party survey data. Here is a more detailed look into each source:

JFrog Platform usage data

Technology usage trends highlighted in this report come from full-year analysis of anonymized JFrog Platform data, drawing on multiple years of historical data representing thousands of customers, hundreds of thousands of repositories, and Petabytes of artifacts.

Package popularity is represented by year-over-year growth across three dimensions: repository count, artifact count, and request volume (uploads and downloads). Request count provides a good representation of how often different package types are being actively used in software development.

It is possible that a handful of enterprises could skew these rankings. However, because we also look at artifact actions, we can safely conclude which package type is actively being used as part of the development process.

Analysis by the JFrog Security Research Team

As a designated CNA, the JFrog Security Research Team regularly monitors and investigates new vulnerabilities to understand their true severity and publishes this information for the benefit of the community and all JFrog customers.

This report includes data pulled from public sources via the JFrog Catalog service, CVE information pulled from the National Vulnerability Database, and proprietary analysis performed by the JFrog Security Research Team on those data sources.

Commissioned survey results

JFrog commissioned Atomik Research to conduct an international online survey of 1,508 IT professionals working in select industries[1] throughout the United States (n=508), the United Kingdom (n=125) , India (n=167), Germany (n=120), France (n=125), Australia (n=165), Singapore (n=174) and Spain (n=124).

The sample consists of full-time employees who hold specific job functions[2] within their organization's information technology, information systems or technology departments. Moreover, all respondents indicate their employing organization consists of 1,000 or more total employees and confirm the presence of a software development team with at least 50 team members within their organization. All participants had the option to access English, French, German, Hindi, and Spanish translations of the online questionnaire.

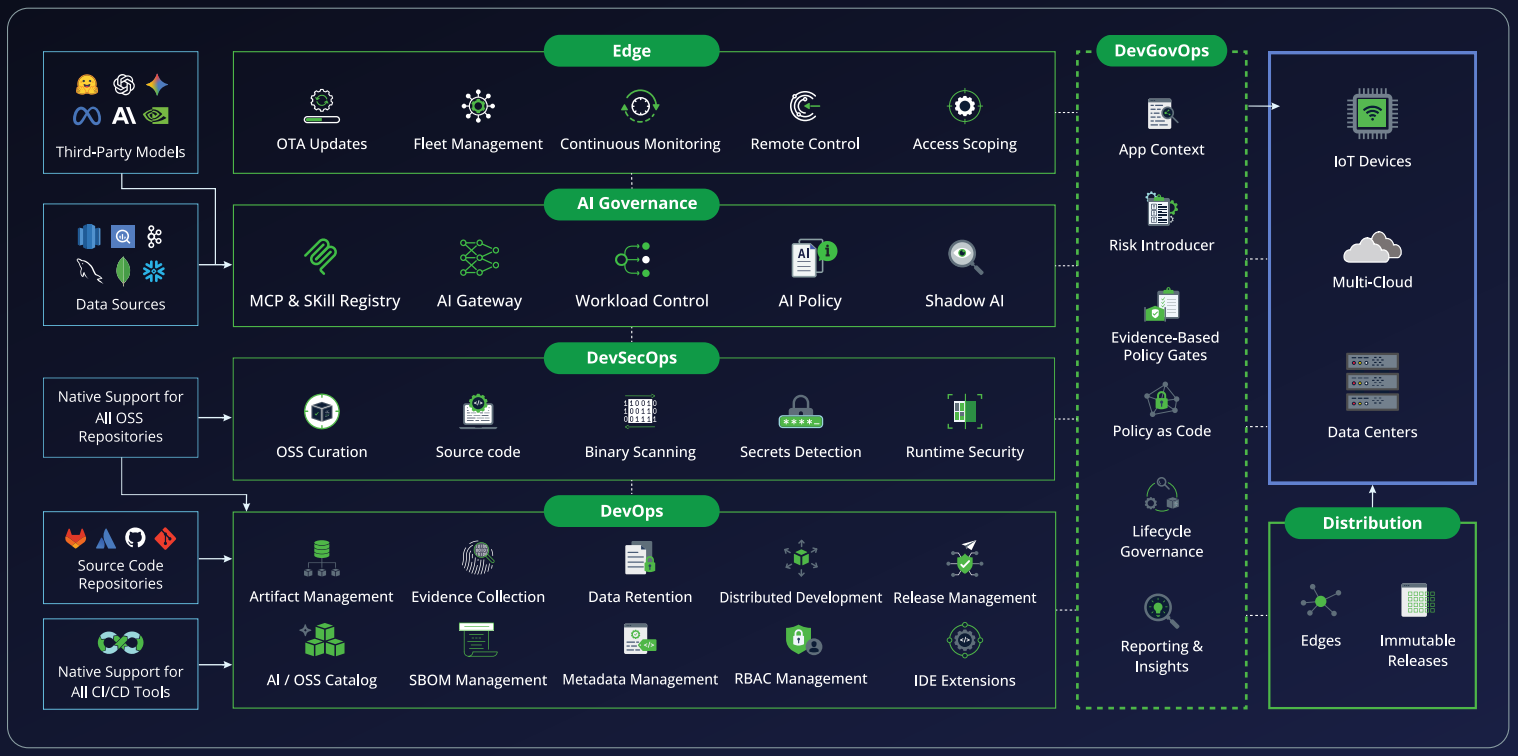
The margin of error for the overall sample is +/- 3 percentage points with a confidence level of 95 percent. Fieldwork took place between January 28 and February 13, 2026. Atomik Research is a creative market research agency.

[1] To qualify for participation, all respondents must indicate they are employed at organization that serve the following industries: (a.) aerospace (b.) architecture and engineering (c.) automotive (d.) banking, financial services, insurance & fintech (e.) energy, oil, gas (f.) government or public sector (g.) healthcare and life sciences (h.) hospitality (i.) manufacturing (j.) retail (k.) technology (l.) transportation and logistics (m.) utilities, telecom & power

[2] To qualify for participation, all respondents must indicate having job functions of, or similar to, the following: (a.) AI specialist or AI engineer (b.) application security engineer (d.) cybersecurity engineer (e.) data scientist (f.) developer (g.) DevOps architect (h.)DevOps engineer (i.) engineering manager (j.) machine learning specialist or ML engineer (k.) platform engineer (l.) security architect (m.) security researcher (n.) site reliability engineer (o.) software architect (p.) software developer (q.) software engineer (r.) solutions architect in addition to indicating employment in their organization's information technology, information systems, technology departments or IT product development department .

About JFrog

JFrog Ltd. (Nasdaq: FROG), the creators of the unified DevOps, DevSecOps, DevGovOps and MLOps platform, is on a mission to create a world of software delivered without friction from development to production. Driven by a “Liquid Software” vision, the JFrog Platform is a software supply chain system of record that is designed to power organizations as they build, manage, and distribute secure software with speed and scale. Holistic security features help identify, protect, and remediate against threats and vulnerabilities. The universal, hybrid, multi-cloud JFrog Platform is available as both SaaS services across major cloud service providers and self-hosted. Millions of users and approximately 6,600 organizations worldwide, including a majority of the Fortune 100, depend on JFrog solutions to securely embrace digital transformation in the AI era. Learn more at www.jfrog.com or follow us on X @JFrog.



JFrog ARTIFACTORY

The gold standard for managing the lifecycle of software artifacts, containers, and ML models, with native support for over 60 different package and artifact technologies.

JFrog RUNTIME

Get real-time visibility into runtime vulnerabilities at the package level, prioritize potential threats and quickly identify its source and developer for fast remediation.

JFrog CURATION

The guardian at the gate of your software supply chain, blocking malicious or risky packages, models, and IDE extensions before they ever reach a developer's machine.

JFrog XRAY

The security layer that spans the entire lifecycle, continuously scanning binaries, containers, and AI artifacts for vulnerabilities and license compliance issues from build through release.

JFrog ADVANCED SECURITY

Next level application security with software supply chain security exposure scanning, code scanning, and contextualized vulnerability analysis to cut through alert noise so remediation efforts have real impact.

JFrog APPTRUST

The governance layer tracking ownership, lifecycle stages, and compliance evidence so that risk is managed at the application level.

JFrog CONNECT

Bring enterprise DevOps and security practices to IoT development to manage IoT fleets and software updates at scale.

JFrog ML

Go from idea to production with the all-in-one solution to build, deploy, manage, and monitor all your AI workflows, from GenAI and LLMs to classic ML.

JFrog DISTRIBUTION

Extend your circle of trust to the last mile of software delivery and take software to the ideal location for optimal consumption.

JFrog AI CATALOG

The source of truth for your AI ecosystem, with registries for models, MCP servers, agent skills, and other AI artifacts, enabling teams to rapidly adopt AI without compromising on governance, security, or compliance.

Cautionary note regarding forward looking statements

This report contains “forward-looking” statements, as that term is defined under the U.S. federal securities laws, including, but not limited to, statements regarding the JFrog usage data and risks related to the impact of AI on the growth and security of the software supply chain.

These forward-looking statements are based on our current assumptions, expectations and beliefs and are subject to substantial risks, uncertainties, assumptions and changes in circumstances that may cause JFrog’s actual results, performance or achievements to differ materially from those expressed or implied in any forward-looking statement.

There are a significant number of factors that could cause actual results, performance or achievements, to differ materially from statements made in this report, including but not limited to risks detailed in our filings with the Securities and Exchange Commission, including in our annual report on Form 10-K for the year ended December 31, 2025, our quarterly reports on Form 10-Q, and other filings and reports that we may file from time to time with the Securities and Exchange Commission. Forward-looking statements represent our beliefs and assumptions only as of the date of this press release. We disclaim any obligation to update forward-looking statements except as required by law.