WIZ

State of SDLC Security 2026

Examining Security Across Source Code, Developer Tooling, Automation Systems, and AI-Assisted Development

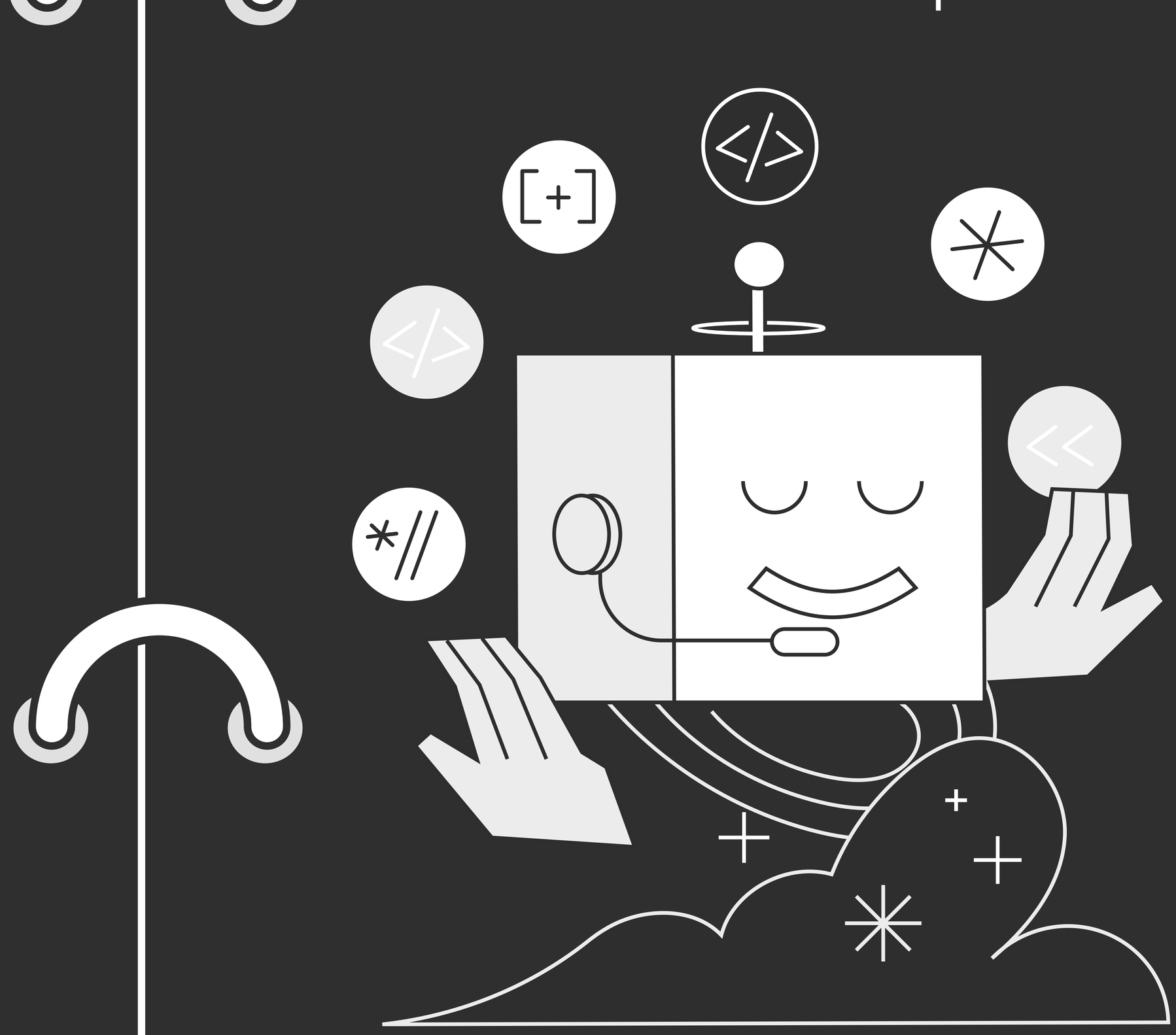


Table of Contents

Introduction

Application security no longer begins at runtime. It begins upstream, in the code developers write and in the infrastructure that builds, reviews, and delivers that code into production.

Modern software development has dramatically increased speed and scale. Open source ecosystems, AI-assisted coding, and automated pipelines have made software cheaper and faster to produce than ever before. At the same time, these forces have quietly expanded the attack surface across the software development lifecycle.

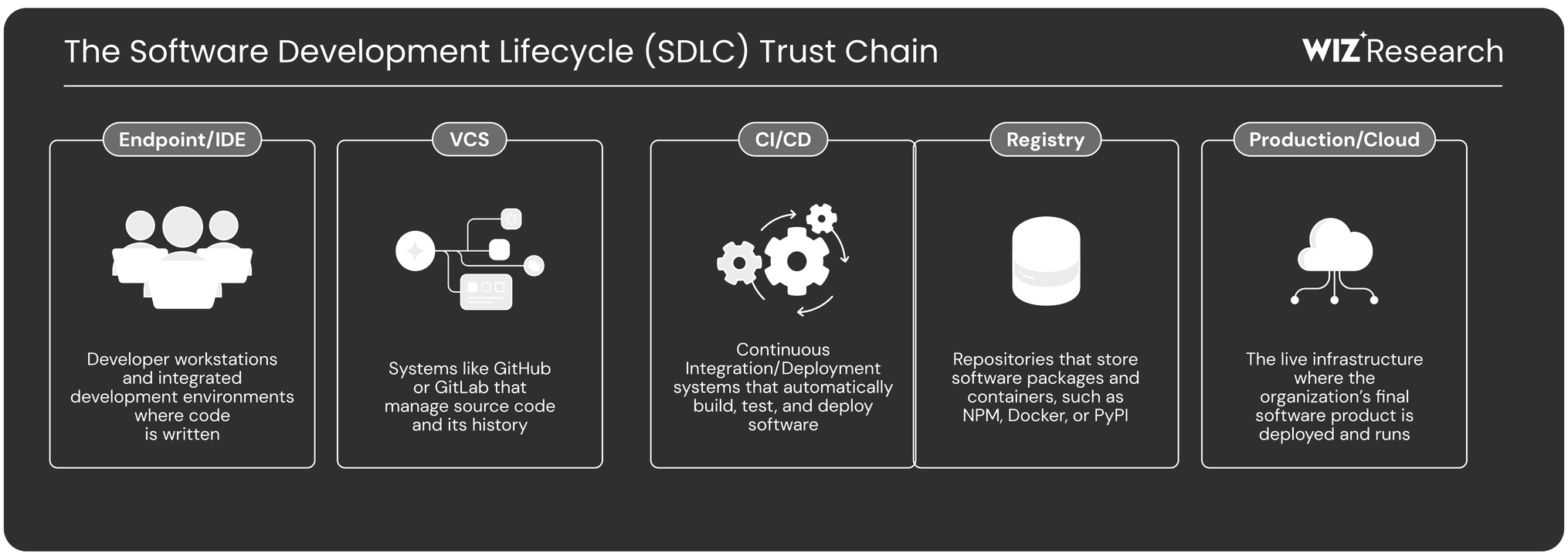
The result is a structural shift in where risk accumulates. Exposure doesn't necessarily emerge from rare vulnerabilities or novel exploitation techniques, but from where trust concentrates and automation connects development systems directly to production environments.

Research Framework

This report is based on analysis by Wiz Research across over 1,000 enterprise cloud and development environments with Wiz Code installed, observed between March and April 2026. It examines SDLC security as a system composed of two interconnected domains:

- **Code:** Source code, dependencies, and secrets introduced during development.
- **SDLC Infrastructure:** Developer endpoints, IDEs, version control systems, and CI/CD platforms that move code into production.

Findings related to SDLC infrastructure are aligned with the Software Infrastructure Threat Framework (SITF), which models attacker behavior across developer endpoints, version control systems, and CI/CD pipelines, including build runners. SITF provides the lens through which infrastructure-level trust relationships are evaluated throughout this report.



Executive Summary

The findings in this report highlight the most significant patterns shaping risk across modern software development environments, from source code and dependencies to developer tooling and automation infrastructure.

1 Risk scales through concentration, not novelty

Most exposure does not stem from rare exploits. It emerges where widely reused languages, packages, tools, and automation concentrate trust across thousands of environments. Across ecosystems, dependency adoption follows power-law distributions, meaning a small core set of packages appears across a disproportionate share of organizations.

2 Software is being produced faster than traditional review models can scale

AI-assisted development and open source reuse have made software cheaper and faster to produce, increasing how often insecure patterns appear and propagate. Python and JavaScript dominate environments, and AI-related API keys now represent a disproportionate and rapidly growing share of leaked secrets.

3 Developer tooling now sits at the center of the SDLC trust chain

Developer endpoints, IDEs, and extensions operate with direct access to code, credentials, and automation. With **at least 80% of organizations using AI IDE extensions** and at least **71% running at least one AI coding assistant** according to our research, trusted execution paths are expanding faster than many organizations can centrally govern.

4 Permissions define impact more than vulnerabilities

In version control and CI/CD systems, write access and automation determine blast radius. With approximately **45–50% of organizations using GitHub Actions**, and a small set of widely reused actions and third-party applications requesting powerful repository and workflow scopes, compromise often becomes a supply chain event by design rather than by exploit sophistication.

5 AI can amplify structural weaknesses across the SDLC

AI increases code volume, reuse, and automated change propagation, allowing existing weaknesses to spread faster and farther across development environments. The Moltbook exposure illustrates this dynamic, where AI-generated development patterns unintentionally exposed millions of API keys.

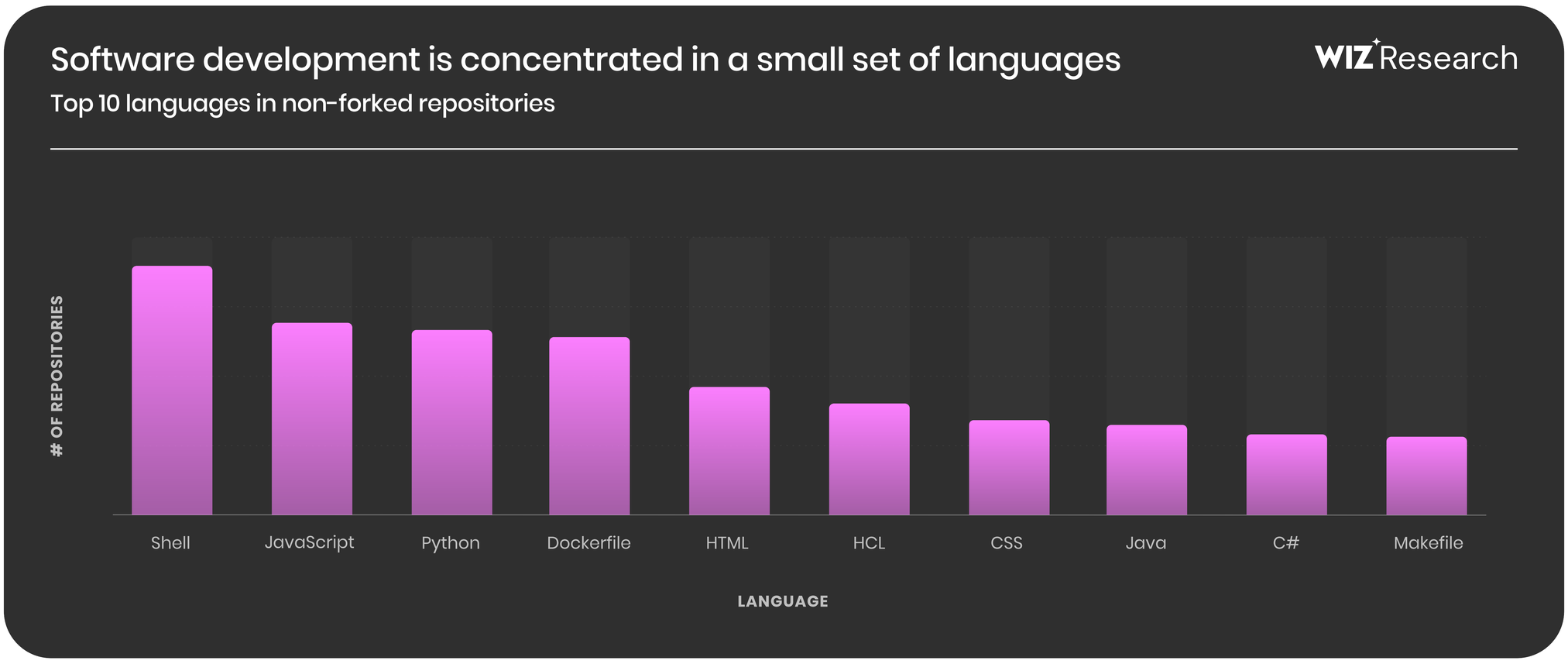
Source Code and Dependencies

1 Language concentration amplifies blast radius

Modern application risk is rarely defined by isolated coding mistakes. It is shaped by volume, reuse, and systemic concentration.

AI-assisted development and open source ecosystems have dramatically lowered the cost of writing and integrating software. That efficiency has a structural side effect: insecure patterns now appear more frequently and propagate more widely before teams can meaningfully review them.

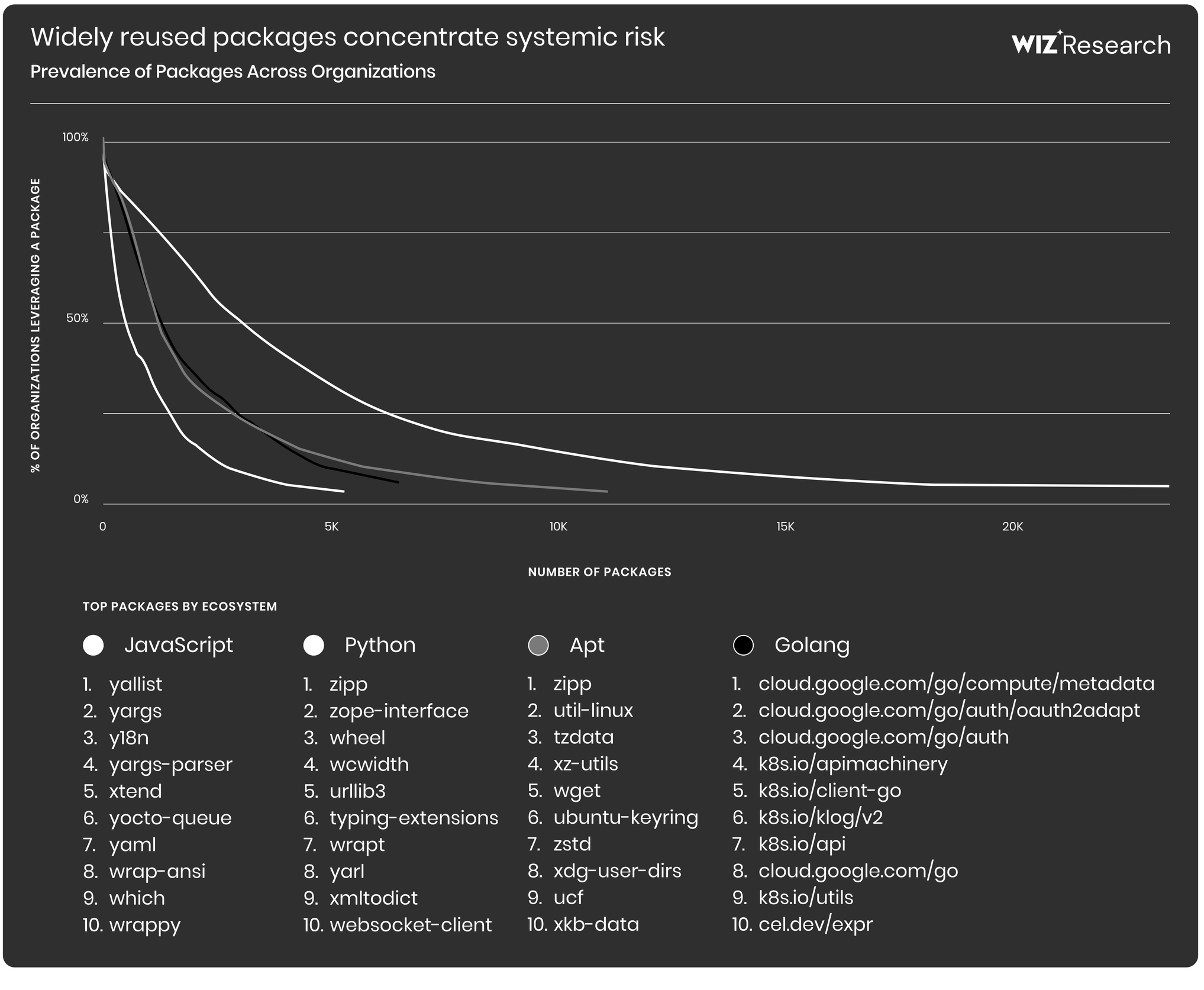
Language usage illustrates this concentration. Python and JavaScript together account for the majority of observed codebases, with Java continuing to grow year over year while compiled languages remain a minority across environments.



This matters because widely used languages share common ecosystems, libraries, and development patterns. **When weaknesses emerge in these highly concentrated environments, they do not remain isolated defects.** They propagate through shared code, dependencies, and practices, becoming cross-environment exposure events.

2 Dependency prevalence drives systemic exposure

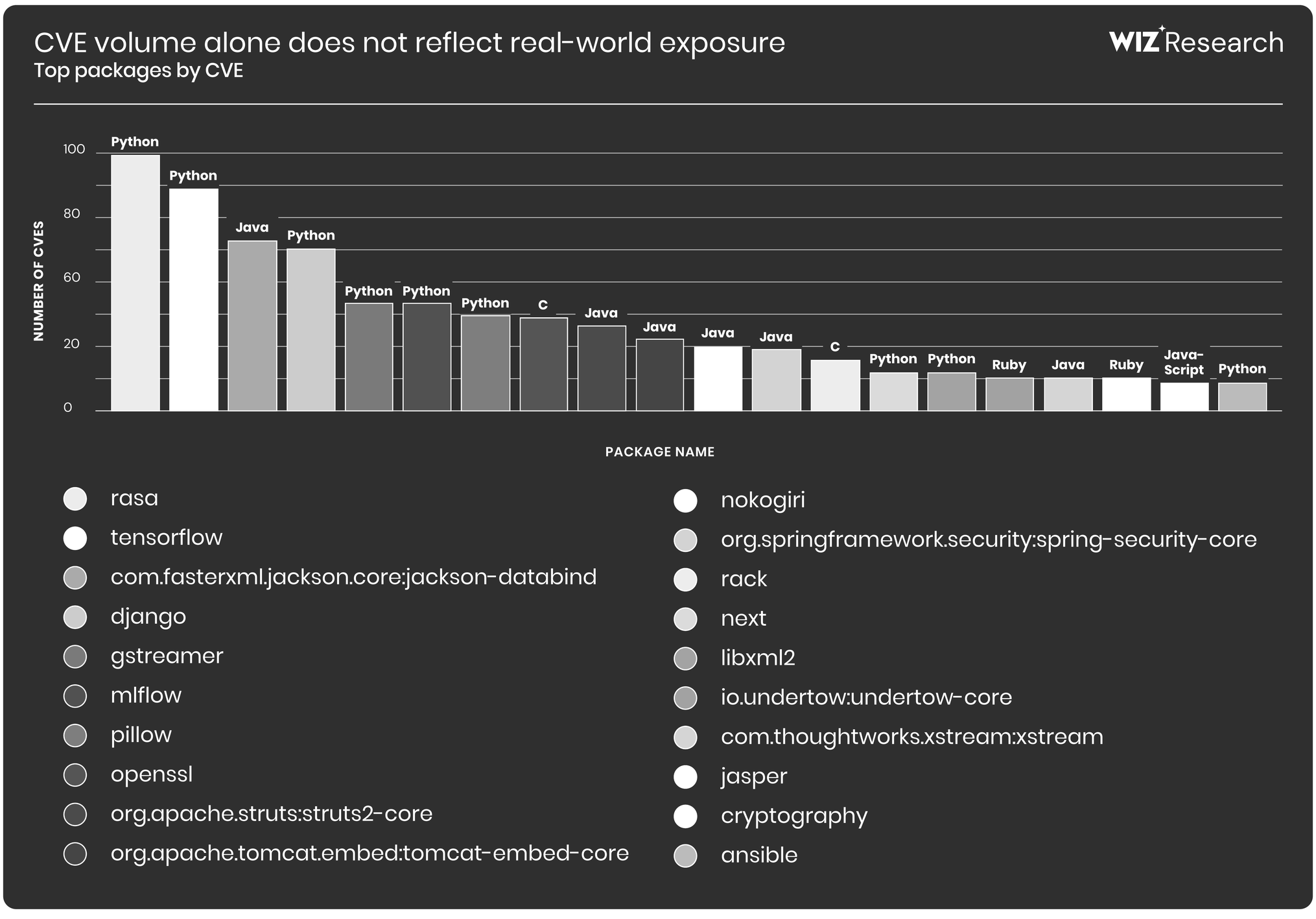
Across four major ecosystems, package adoption by organizations follows a clear power-law distribution.



A long tail of packages exists, but a relatively small core set appears across a disproportionately large percentage of organizations. In practice, a large share of environments rely on just a few thousand widely used packages before adoption drops off sharply. Unlike typical assumptions, risk is not concentrated in a single dominant package.

Instead, a broader set of widely adopted components collectively creates systemic exposure across environments. When a widely adopted package contains a flaw, misconfiguration, or compromised release, exposure scales immediately. **Prevalence becomes more important than raw vulnerability count.**

The same pattern appears when looking at packages with the highest number of CVEs.



High CVE count alone does not determine systemic risk. **Real-world impact is shaped by the combination of vulnerability severity and how broadly a package is reused across environments.** Widely adopted components can create disproportionately large exposure when weaknesses emerge in software trusted across many organizations.

3 Leaked secrets provide direct infrastructure access

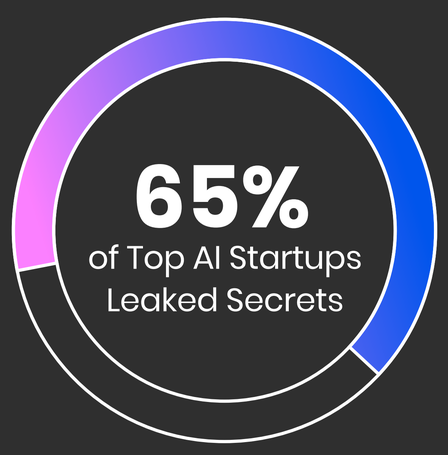
Secrets further amplify this concentration dynamic. Unlike prior analyses, this dataset focuses on **validated secrets**, meaning credentials that were confirmed to be active and usable at the time of discovery. Validated secrets found in public repositories frequently provide infrastructure-level access rather than application-only access. These include:

- Cloud provider credentials
- CI/CD tokens
- Third-party API keys
- AI service credentials

Credentials for AI platforms represent a disproportionate share of leaked secrets relative to the maturity of the ecosystem. According to our research, the most commonly exposed credentials belong to platforms such as Hugging Face, Weights & Biases, and Azure OpenAI, though this list is expected to evolve rapidly as the AI ecosystem shifts.

Analysis of public repositories reveals widespread leakage of sensitive credentials such as API keys. **Credentials tied to popular AI platforms appear disproportionately often, reflecting how frequently these services are used in modern development workflows rather than weaknesses in the platforms themselves.** Rapid development practices and automated workflows increase the likelihood that these secrets are exposed in code before they are detected or revoked.

The Scale of AI Secret Exposure



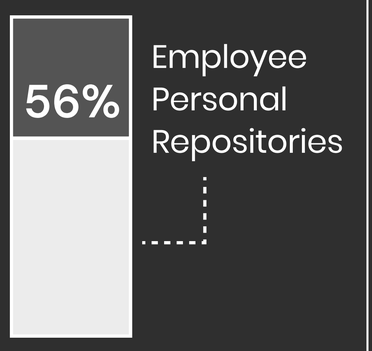
A scan of the Forbes AI 50 companies revealed verified secret leaks on GitHub.

Over $400B

Company Value Exposed

This is the combined valuation of the AI companies found with verified leaks.

Leaks Extend Beyond Company-Owned Repositories



56% of secrets with company impact were found in employees' personal repositories.

The Anatomy of a Leak: What and Where

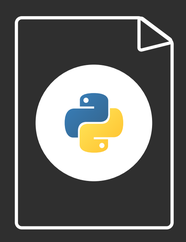
AI-Related Secrets are the Most Common Leaks

4 of the top 5 most frequently found validated secrets were for AI platforms.

Top 3 Leaked Secret Types

- Hugging Face
- Weights & Biases
- Azure OpenAI

Python Notebooks (.ipynb) are a Secret "Goldmine"



Due to their mix of code and output, .ipynb files are the #1 source of leaked secrets.

Top 3 Leaking File Types

- .ipynb
- python
- .env

WIZ Research

The most leak-prone file types include notebooks (.ipynb), Python files, and .env configuration files. These reflect mainstream development workflows, particularly in AI-heavy environments where models, datasets, and API integrations are frequently developed in interactive environments.

AI-assisted development further accelerates this pattern. In September 2025, Wiz Research found that roughly one in five organizations using AI-powered vibe-coding platforms had applications affected by systemic security issues, illustrating how **generation defaults can translate into repeatable weaknesses** rather than just isolated mistakes.

Wiz Research observed a similar pattern in the Base44 platform, where shared generation logic introduced systemic design flaws that enabled unauthorized access to private applications across multiple environments.

4 Implications

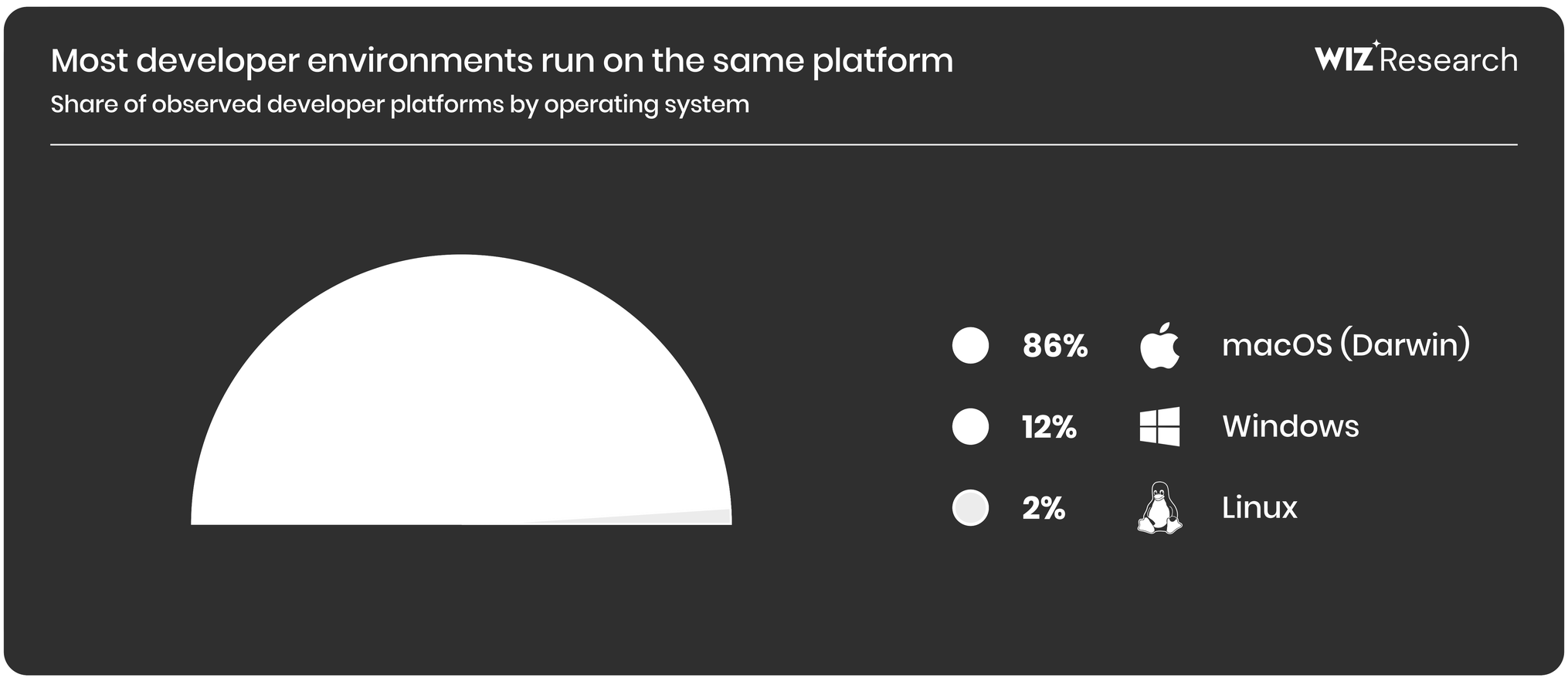
Without visibility into where reuse concentrates and which secrets grant infrastructure-level access, **security teams risk chasing individual findings while systemic exposure continues to accumulate upstream.**

Developer Endpoints and IDEs

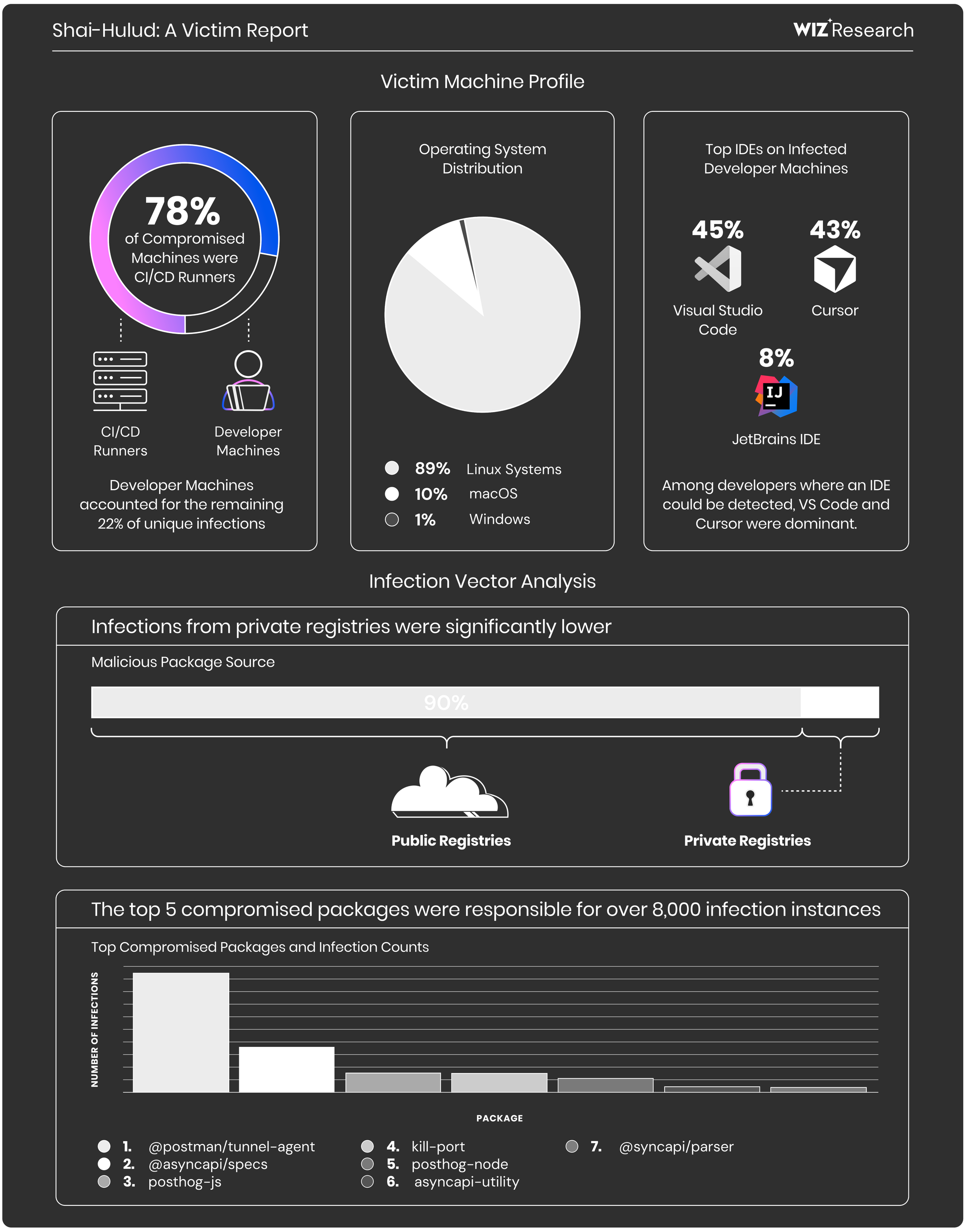
1 Developer environments are standardized but highly privileged

Developer endpoints have evolved into some of the most powerful systems in the software development lifecycle.

Most environments converge around a small number of operating systems. For example, according to our research, **macOS (Darwin) accounts for roughly 86%** of observed developer platforms, with Windows and Linux making up a much smaller share.



This pattern is also visible in [Shai-Hulud victimology,](#) where compromised environments consistently reflected common developer workstation configurations.



Standardization is double-edged. It simplifies defensive baselining, but it also makes attacker targeting efficient and repeatable. **When developer environments share similar operating systems, development tools, and configuration patterns, attackers can reuse techniques across organizations with minimal adaptation.**

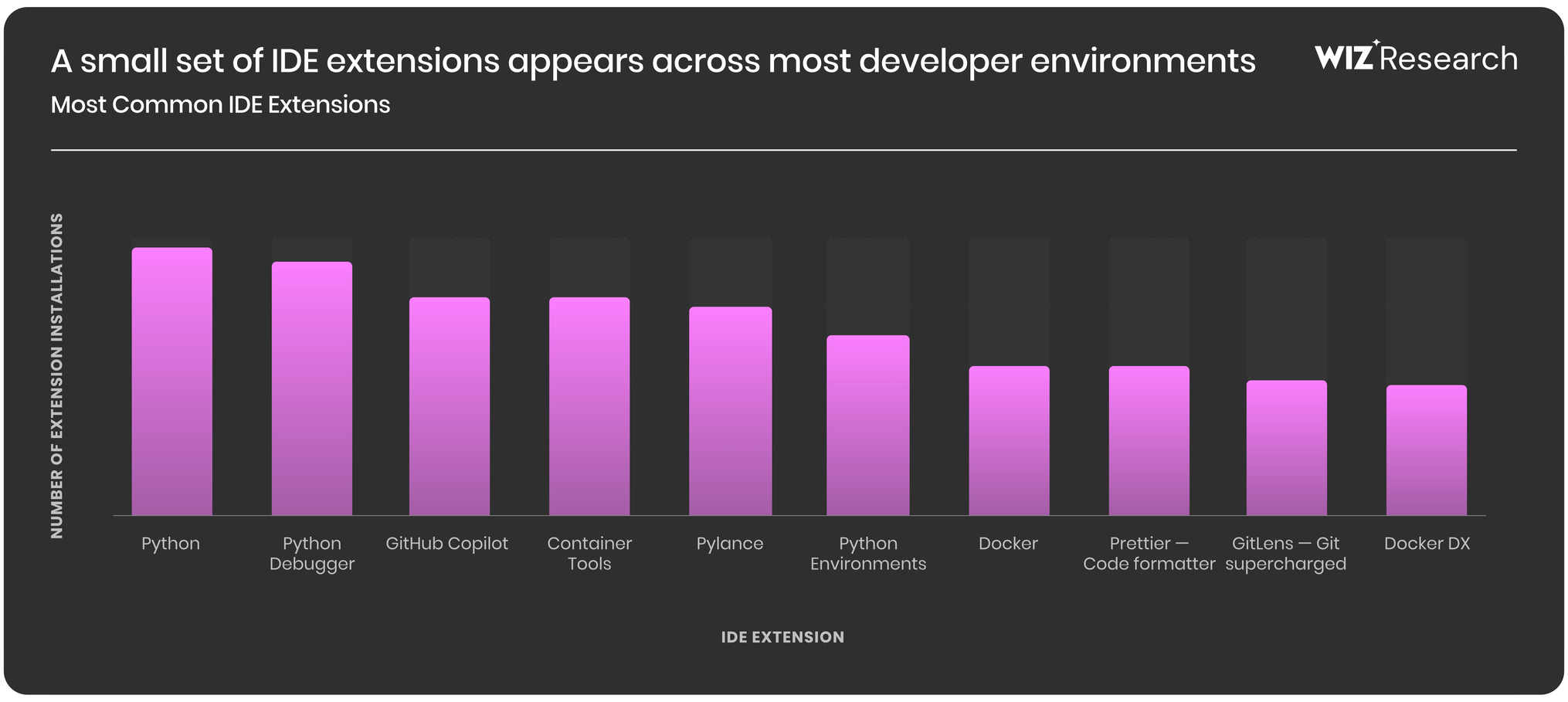
At the same time, developer endpoints operate with broad privileges by design. They provide direct access to source code, package managers, credentials, version control systems, and deployment automation. In practice, the developer machine often sits at the center of the SDLC trust chain.

2 The extension layer introduces fragmented trust

While the platform layer is relatively standardized, the extension and tooling layer is highly fragmented. Dozens to hundreds of IDE extensions are commonly observed across developer environments. Many operate with extensive privileges, including:

- Filesystem access
- Network access
- Full development context awareness

At the same time, a relatively small set of extensions appears frequently across environments, particularly language tooling, container tooling, and AI-assisted development extensions such as coding assistants. This combination of widespread reuse and long-tail diversity reinforces the fragmented and decentralized nature of the developer tooling layer.



These extensions function as trusted execution paths inside developer machines. They are tightly embedded in everyday workflows yet rarely monitored with the same rigor applied to production systems.

This creates a structural asymmetry: the underlying platform is predictable, but the trust layer is decentralized and largely driven by developer choice. Security teams often have limited visibility into which tools operate inside development environments and what privileges they hold.

3 AI coding assistants are rapidly expanding trusted execution paths

AI-assisted development is accelerating this dynamic.

80% | at least 80% of organizations leverage AI IDE extensions

71% | of organizations have at least one AI coding assistant present

According to Wiz's State of AI in the Cloud research, at least 80% of organizations have developers using AI IDE extensions, and more than 70% have at least one AI coding assistant present in their environment. It is common for these tools to be adopted bottom-up by developers rather than deployed through centralized governance.

Adoption also does not converge on a single platform. Multiple AI coding assistants frequently coexist within the same environment, including both commercial and open-source tools. As a result, several AI systems may influence code generation, refactoring, and development workflows simultaneously.

Coding assistants operate with broad local privileges and deep context into the development environment. They can read files, suggest code changes, and interact with repositories and build tooling. As these systems become embedded in everyday workflows, they increase the number of trusted execution paths inside already privileged environments.

Once a developer endpoint is compromised, attackers inherit everything the developer can reach, including:

- Source code
- Package managers
- Credentials
- Version control access
- CI/CD systems
- Downstream cloud resources

In practice, the developer machine functions as a command center for modern software delivery.

4 Implications

Developer endpoints represent one of the most privileged and least scrutinized surfaces in the SDLC. Their underlying platforms are standardized and predictable, while their extension layers introduce a wide and often poorly understood set of trusted execution paths.

As AI tooling becomes embedded in everyday development workflows, the number of systems influencing code creation and automation continues to grow. Without stronger visibility and governance over developer environments, compromise of a single workstation can quickly translate into access across repositories, automation pipelines, and cloud infrastructure.

Version Control Systems

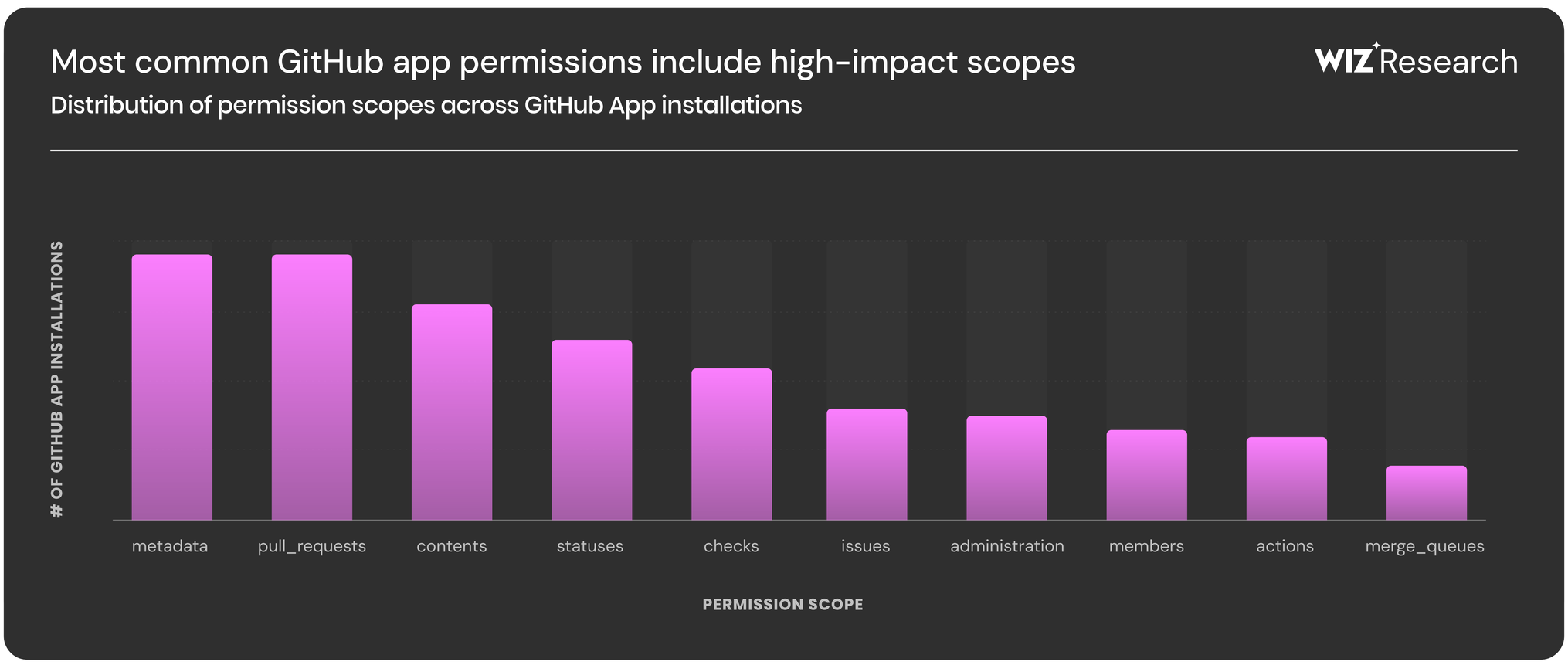
1 Version control is a trust engine, not a code warehouse

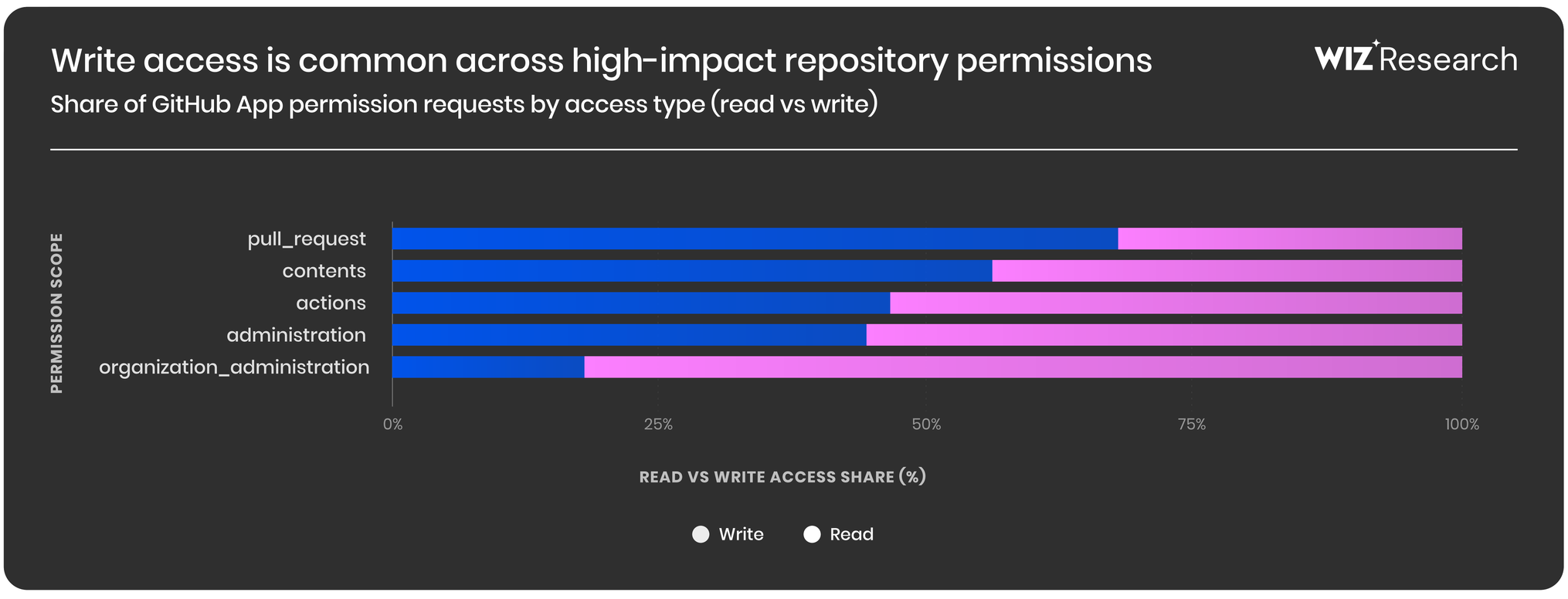
Version control systems encode the trust relationships that determine how software changes propagate. Repository exposure matters, but the more consequential risk is what happens once a repo is reachable through legitimate permissions and integrations.

2 Public exposure is not the main story, but it sets the baseline

Overall public repository exposure remains relatively low at approximately 2% according to our research, though exposure varies by platform and is meaningfully higher in GitHub-heavy environments. However, when viewed at the organizational level the picture looks different: **roughly 54% of version control organizations have at least one publicly exposed repository,** meaning that some degree of external code visibility remains common across environments.

The bigger risk emerges after access is obtained through OAuth apps, GitHub Apps, tokens, or compromised developer identities, because permissions convert access into impact.





3 Write permissions turn compromise into supply chain impact by default

The defining factor in version control is not a CVE. It is whether an identity, integration, or app can write. Write access enables direct code modification, approval path manipulation, and workflow changes that can spread downstream without requiring further exploitation.

4 Third-party integrations concentrate privilege the same way dependencies concentrate reuse

GitHub Apps frequently request powerful scopes, commonly including repository contents, pull requests, and workflow configurations. Many widely used apps request write permissions by default, and high-impact scopes tend to concentrate among a smaller set of commonly installed tools, which mirrors the dependency concentration pattern seen in code.

5 AI accelerates change propagation inside the same permission model

AI-assisted development does not change the underlying trust mechanics of version control, but it increases velocity within them. As established in Wiz's State of AI in the Cloud report, **at least 71% of organizations have at least one AI coding assistant present**, and these tools increasingly contribute to code generation and pull-request activity. When AI-generated pull requests and automated changes inherit existing permissions, the combination of automation plus write access can compress review cycles and allow risky changes to propagate faster than teams can consistently validate.

6 Implications

Version control is where trust decisions become software distribution. **Broad write permissions, persistent integrations, and rarely revisited app scopes can make supply chain impact the default outcome of compromise rather than an edge case.** Defenders reduce risk fastest by treating write access as a high-risk capability, continuously revalidating which apps and identities can modify code or workflows, and prioritizing governance where privilege and reuse are most concentrated.

CI/CD and Automation

1 CI/CD platforms combine execution, automation, and credentials by design

Continuous integration and delivery platforms are among the most powerful systems in the software development lifecycle. They are designed to execute code automatically, orchestrate build processes, and move artifacts into production environments.

Because these systems operate with trusted identities and automation privileges, they often inherit access to source repositories, cloud credentials, and deployment infrastructure. In practice, CI/CD platforms function as execution environments embedded directly inside the SDLC trust chain.

2 CI/CD is widely deployed across development environments

Adoption of CI/CD platforms is now standard across modern development pipelines.

45–50% | of organizations have GitHub Actions enabled

According to our research, approximately 45–50% of organizations have GitHub Actions enabled, immediately expanding the SDLC attack surface through automated workflows that execute code on every commit, pull request, or build event.

Automation at this scale means that seemingly small configuration decisions can have wide operational impact.

3 Workflow misconfigurations create reliable attack paths

Many CI/CD workflows contain configuration patterns that increase risk. In practice, these issues are widespread.

Common issues include:

- Missing permission restrictions
- Failure to pin third-party actions to specific versions
- Workflows with write access to repositories or pull requests

Individually, these issues may appear minor. When combined, they can create reliable attack paths that allow malicious code to execute during build processes or modify repository content through automated workflows.

4 Third-party actions concentrate risk through reuse

A relatively small number of automation components dominate real CI/CD workflow execution. SBOM analysis of GitHub Actions workflows shows that many organizations rely on the same small set of reusable actions embedded directly inside build pipelines.

While `actions/checkout` appears in nearly all observed environments, several additional actions show widespread adoption. For example, `actions/setup-node` appears in roughly **78% of organizations,** while actions such as `actions/setup-python` and `actions/cache` appear in **more than 60%.** These patterns reinforce how a relatively small set of reusable components sits inside CI/CD pipelines across large numbers of environments.

This concentration mirrors the dependency patterns observed earlier in the report. When a widely trusted action is compromised or abused, the impact can scale quickly because the same component is embedded in CI/CD workflows across large numbers of organizations. Reuse increases efficiency for developers, but it also concentrates risk within the CI/CD ecosystem.

5 CI/CD runners frequently inherit production-level access

The execution environment inside CI/CD pipelines often holds significant privileges.

Shai-Hulud victimology shows **GitHub Actions as the dominant CI/CD platform, observed in 59% of environments.** Within those environments, **self-hosted runners account for roughly one third of deployments.**

Compromised runners frequently contain:

- Cloud credentials
- Access tokens
- Build tooling
- Outbound network access

The distinction between runner types is significant. Managed runners are ephemeral and operate within provider-controlled infrastructure, limiting persistence opportunities. Self-hosted runners operate within the organization's own environment, allowing attackers to pivot laterally into internal systems if they gain execution.

6 Automation collapses the boundary between development and production

Once execution is obtained within CI/CD pipelines, the separation between development infrastructure and production environments effectively disappears.

Build systems routinely deploy infrastructure, publish artifacts, update container registries, and interact with cloud APIs. When those workflows execute **under compromised conditions, automation can propagate malicious changes or grant attackers direct access to downstream environments.**

7 Implications

CI/CD platforms represent one of the most direct paths from development-stage access to production compromise. They combine remote execution, automation, and credential access at organizational scale.

When these systems are treated as build conveniences rather than privileged execution environments, small configuration weaknesses can quickly translate into high-impact breaches. Effective defense requires treating CI/CD infrastructure as critical security infrastructure: continuously auditing workflow permissions, restricting runner privileges, and scrutinizing the third-party components that execute inside automated pipelines.

Conclusion

Across both code and SDLC infrastructure, the same structural pattern appears repeatedly.

Risk does not scale primarily through novel vulnerabilities or advanced exploitation techniques. Risk scales through concentration, privilege inheritance, and automation. **Weaknesses accumulate where reuse is highest, permissions are broadest, and development systems connect directly to production environments.**

In source code and dependencies, systemic exposure emerges from highly concentrated ecosystems. Widely used languages, frameworks, and packages create efficiency for developers but also concentrate risk. When weaknesses appear in components used across thousands of environments, even small issues can quickly become cross-environment exposure events.

Developer endpoints extend this dynamic. The underlying platform layer is highly standardized, while the trust layer introduced by extensions and development tooling remains fragmented and difficult to govern. As AI-assisted development becomes embedded in everyday workflows, these environments increasingly influence how code is generated, reviewed, and deployed, expanding the number of trusted execution paths inside already privileged systems.

Version control systems translate those trust relationships into software distribution. The primary risk is not repository exposure but permission design. **Broad write access, persistent integrations, and automation workflows mean that compromise often becomes supply chain impact by default rather than by exploit sophistication.**

CI/CD platforms then connect development systems directly to production infrastructure. These platforms combine automated execution, credentials, and deployment authority at scale. Once compromised, they collapse the boundary between development and runtime environments, allowing attackers to move quickly from code-level access to cloud impact.

Across every layer, AI acts as an accelerant. It increases code volume, speeds up reuse, and embeds automation more deeply into development workflows. While AI introduces some emerging risk categories, its most immediate effect is amplification: **existing weaknesses replicate faster, propagate further, and become easier for attackers to exploit.**

The implications for defenders are clear. Modern application security cannot rely solely on finding individual vulnerabilities or increasing alert volume. **Effective defense requires understanding how trust flows across the software development lifecycle and prioritizing the weaknesses** that create real attack paths across that chain.

Organizations that treat code, development tooling, version control, and CI/CD as separate security domains will continue to chase symptoms. Those that analyze how these systems interact, and where privilege and automation concentrate risk, will be better positioned to reduce systemic exposure without slowing innovation.

The future of application security is not about generating more findings. It is rooted in a clearer understanding of which weaknesses actually matter in the systems that build, trust, and ship software.

Methodology

This report is based on analysis by Wiz Research across a large set of real-world enterprise cloud and development environments observed between March and April 2026. Findings reflect observed configuration states, usage patterns, and security exposures in production environments.

Framework and Scope

The analysis is guided by the Software Infrastructure Threat Framework (SITF), which models attacker behavior across developer endpoints, version control systems, CI/CD platforms, and build runners. SITF informed the structure and scope of this report.

Accordingly, the study examines SDLC security across:

- Source code and dependencies
- Developer endpoints and IDEs
- Version control systems
- CI/CD pipelines and runners

Data Sources and Interpretation

The report combines anonymized telemetry from Wiz-protected environments, findings from Wiz Research investigations, victimology from campaigns such as Shai-Hulud and s1ngularity, public repository analysis for validated secrets and dependency prevalence, and aggregated insights from Wiz's State of AI in the Cloud report.

Percentages reflect the proportion of observed environments in which a given technology or configuration appears and should be interpreted as lower-bound estimates. Because usage patterns frequently follow power-law distributions, the analysis emphasizes concentration, reuse, and trust relationships over raw vulnerability counts.

Findings reflect environments monitored by Wiz and may skew toward large, cloud-native enterprises, though consistent patterns across organizations and incidents suggest the results are broadly representative of modern SDLC risk.