WIZ

State of Code Security Report 2025

Table of Contents

Executive Summary

For this report, Wiz researchers examined the ratio of public versus private repositories, occurrence of secrets in code, GitHub Apps security, workflow security settings, and more. In the following pages we will present our findings and draw some conclusions. Here is a preview of the most impactful takeaways stemming from this effort:

1. Even “best case scenario” numbers show a worrisome habit of keeping cloud secrets in code repositories. Alarmingly, 61% of organizations have secrets exposed in public repositories. A threat actor scanning public repos could stumble across these secrets – which might include SaaS API keys, access tokens, or cloud credentials – and exploit them to wreak havoc. For instance, with a leaked AWS access key an attacker could extract sensitive data from cloud storage, incurring significant financial loss and reputational damage.

2. Version Control Systems (VCS) and CI/CD security posture practices are lacking, with workflows and actions boasting high privileges. This is extremely concerning because it is an important part of the software development lifecycle and involves direct access to the production environment.

3. Not all VCS are created equal – at least not when it comes to security. GitHub is not only the leader by number of repositories, but also by percentage of public repositories (over 30%). That creates an appealing target for malicious actors, and the potential for catastrophe. Imagine a developer inadvertently commits an AWS access key to a public GitHub repo, which is then found by a threat actor who uses it to log in to the tenant's cloud environment and deploy instances for cryptomining.

Throughout the process of collection and analysis, it became clear that the broader connection between code repositories and cloud environments provides essential context for interpreting the statistics. Some examples:

- Weak security practices around self-hosted runners, which require particular attention. Runners serve as the middle ground between code and cloud, and can allow attackers to pivot in either direction.
- Code repositories with cloud and SaaS secrets leading to cloud environments.
- Vulnerabilities in code repositories manifesting themselves post-deployment, in cloud.

As such, the report illustrates the reality that **code and cloud are two deeply connected domains in today’s agile, cloud-native world.**

Now let’s explore the takeaways in greater detail. At the end, we will provide recommended best practices.

2024: Year in Review

The past 12 months have been noteworthy for both the number and impact of supply chain attacks. Arguably, the most prominent one was an attack on XZ Utils that exposed the security community to the lengths which malicious actors are willing to go to plant a well-hidden backdoor in a widely used software package. The Wiz Research team investigated this incident extensively and created an overview of the backdoor functionality:

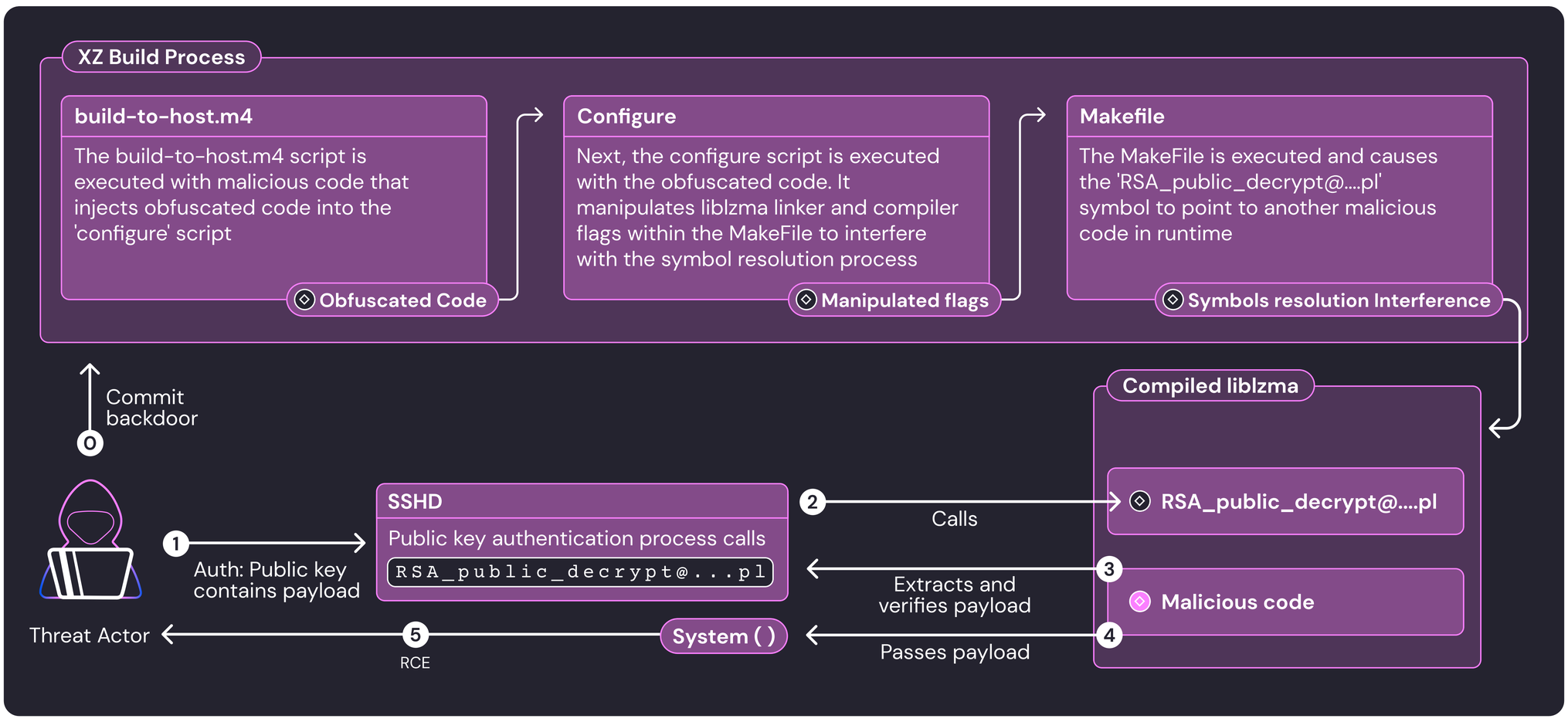


Figure 1: xz-utils attack flow

This incident made headlines worldwide because it illustrated the dangers of an inside actor, in this case a determined individual that was nearly able to infiltrate a widely used open-source project and insert the potential SSH backdoor into a multitude of systems.

A somewhat overlooked angle of this attack is the “disgruntled employee” scenario. Consider XZ Utils project being your GitHub organization and the wicked “JiaTan” (the username of the original bad actor, likely a threat group) being one of your employees. How would you protect against such a scenario?

We believe the answer lies in **multi-layered VCS defenses** – for example, branch protection and PR reviews in addition to the default authentication – and **behavioral monitoring** of users (for example, using GitHub audit logs).

Unlike the XZ Utils incident, exposed secrets are a well-known and well-researched attack vector. This does not prevent attackers from continuing to abuse leaked secrets for their leverage. Again, this year, there were multiple incidents of secrets exposure that eventually led to further compromises (e.g. EMERALDWHALE, RDS Database Exfiltration, ShinyHunters Ransomware). These are great examples of the visibility problem – security scanners must have **visibility into all repositories** and must know how to validate secrets and estimate the secret impact.

Other supply chain attacks this year targeted the NPM and PyPI ecosystems by exploiting multiple language packages. The usual attack vectors include dependency confusion, outright malicious packages available for download, and takeovers of legitimate repositories – either via dangling repositories or a leak of secrets – for packages actively in use. One recent example of malicious NPM packages included a package masquerading as a cookie parser that was actually trying to delete the local filesystem. A rarer attack, however, made waves this summer. A Chinese company named Funnull acquired the Polyfill domain and GitHub repo, and inserted malware into polyfill.js that redirected users to gambling websites. Further pivoting revealed that Funnull had exposed a CloudFlare API key that linked the company to several CDN providers which were also serving malicious scripts.

These incidents stress the dangers that arise from system dependencies. The solution, as we see it, is to challenge security tools to **integrate with the SDLC on multiple levels** – in IDE, in VCS, CI/CD and production environment.

Finally, the CI/CD security research community benefitted from some great findings presented at industry conferences, where presenters discussed novel attack vectors on VCS and CI/CD systems. Some of our favorites include, "Self-Hosted GitHub CI/CD Runners: Continuous Integration, Continuous Destruction" and "H-MY-DC: Abusing OIDC all the way to your cloud". This work involved extensive research on misconfigurations related to GitHub self-hosted runners that resulted in persistent access in CI/CD environments and theft of deployment secrets.

Being such an unexpected and underestimated attack vector, this area is still not widely abused by APTs and other threat actors, potentially because of the lack of good VCS compromise monitoring solutions. However, we believe this will soon change as threat actors catch up with recent research. We must therefore add secure VCS configuration, workflows security, and run-time **security of self-hosted runners** to the long feature list of modern code security tools.

Current Landscape

In this section, we share key statistics—from general usage patterns to specific security control metrics—to provide actionable insights that help counter the attacks discussed above.

Methodology

This report includes insights from Wiz Research based on data collected throughout 2024. Over this period of time, hundreds of thousands of code repositories were analyzed.

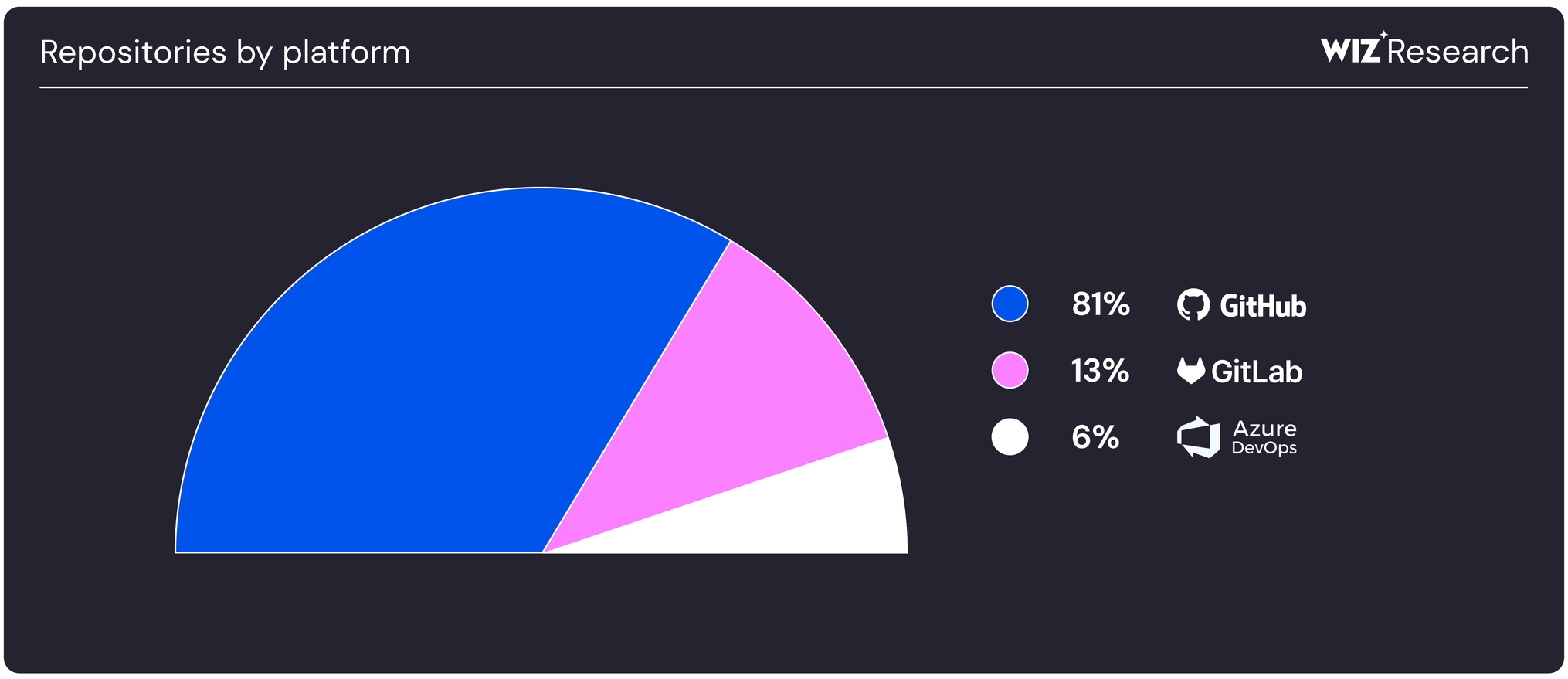
We gained these insights in large part from the release of Wiz Code, which granted us a unique opportunity to connect cloud deployments with their source code. This new offering extends the trademark precision of Wiz Cloud into version control systems (VCS), and Continuous Integration / Continuous Deployment (CI/CD) pipelines – allowing our customers to truly "shift left" at scale without losing cloud context in the process.

Adoption of Wiz Code enabled the Wiz Threat Research team to compile key statistics that shed light on how organizations store and use their code, along with the usage and security posture metrics of their VCS and CI/CD systems.

Overall Usage: Key Findings

GitHub leads the pack with 80% of total repos

This statistic is not inherently surprising, yet it is helpful to give context to subsequent findings. We see that **GitHub is a leader** in a total number of repositories, reflecting the widespread popularity of GitHub within the enterprise development community:



Note: Wiz Code also supports GitLab and AzureDevOps version control systems, which comprise 20% of total repositories.

VCS Trends: GitHub dominates, multi-platform strategies are on the rise

Much like how companies often rely on multiple cloud providers, so too do they sometimes use multiple VCSs for code management and CI/CD for various business reasons. We observed the following numbers on multi-VCS usage:

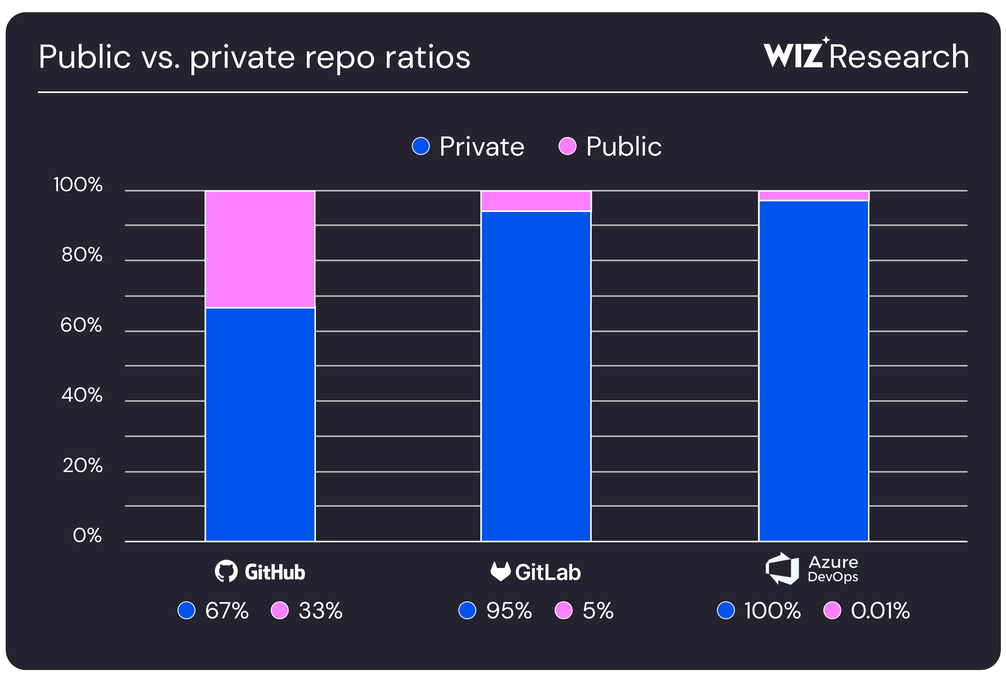
- Only about 5% of the organizations use more than one platform.
- Of those, only a few organizations in our sample set have all three platforms installed, the rest have two platforms.
- Of those with more than one platform, all but a few have GitHub as one of the platforms.

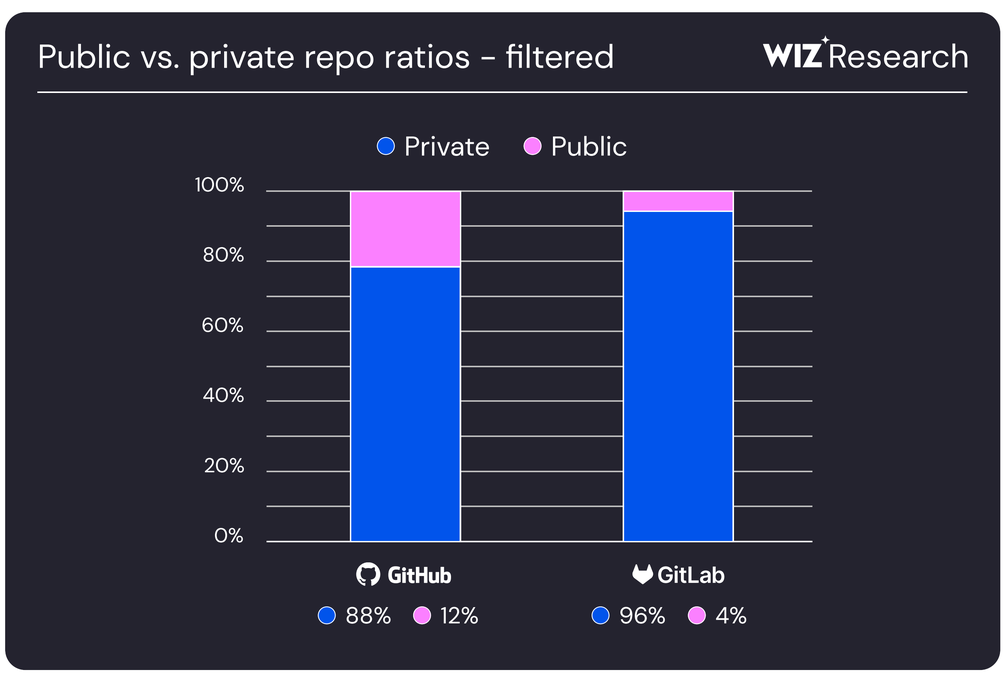
These numbers confirm the popularity of GitHub, along with its "stickiness" – companies can have multiple version control systems, but GitHub is always one of them. It is not too far-fetched to suggest that the popularity of open source is one of the major drivers behind GitHub adoption.

The ratio of public repos in GitHub is 3 times higher than in other VCS platforms

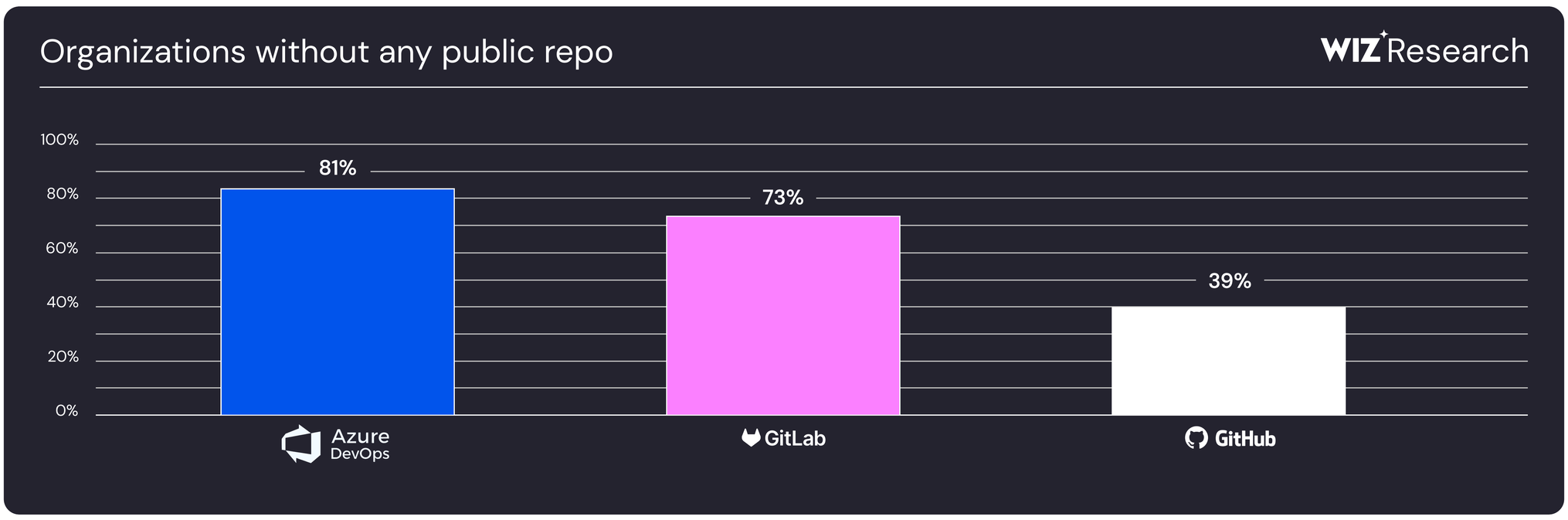
The percentage of public repositories is about 35% in GitHub and less than 10% in other platforms. This does not necessarily mean that GitHub users are less secure, but it does confirm our earlier point: GitHub is the default platform for open-source code. Note that not all public repositories are automatically reachable from the wider internet, as the notion of public vs. private repository is defined by the platform and is fetched from the platform API. Still, it's a useful metric that shows the purpose of a code repository and is a factor that affects security posture.

The subsequent graph shows the statistics across active, non-fork, and unarchived repositories (i.e. filtered) with similar outcomes.





Another useful metric may be the ratio of organizations without any public repositories:



This mirrors the overall ratio of public vs. private repositories.

Wiz Code automatically inventories your code repositories and developer identities from connected VCS platforms, giving you full visibility into what's being built, by whom, and where it's deployed.

For each repository, Wiz shows visibility status, scan types (scheduled or PR-triggered), issues detected, and connections to cloud resources via the Wiz Security Graph. Developer identities are also analyzed for inactivity, excessive permissions, and more.

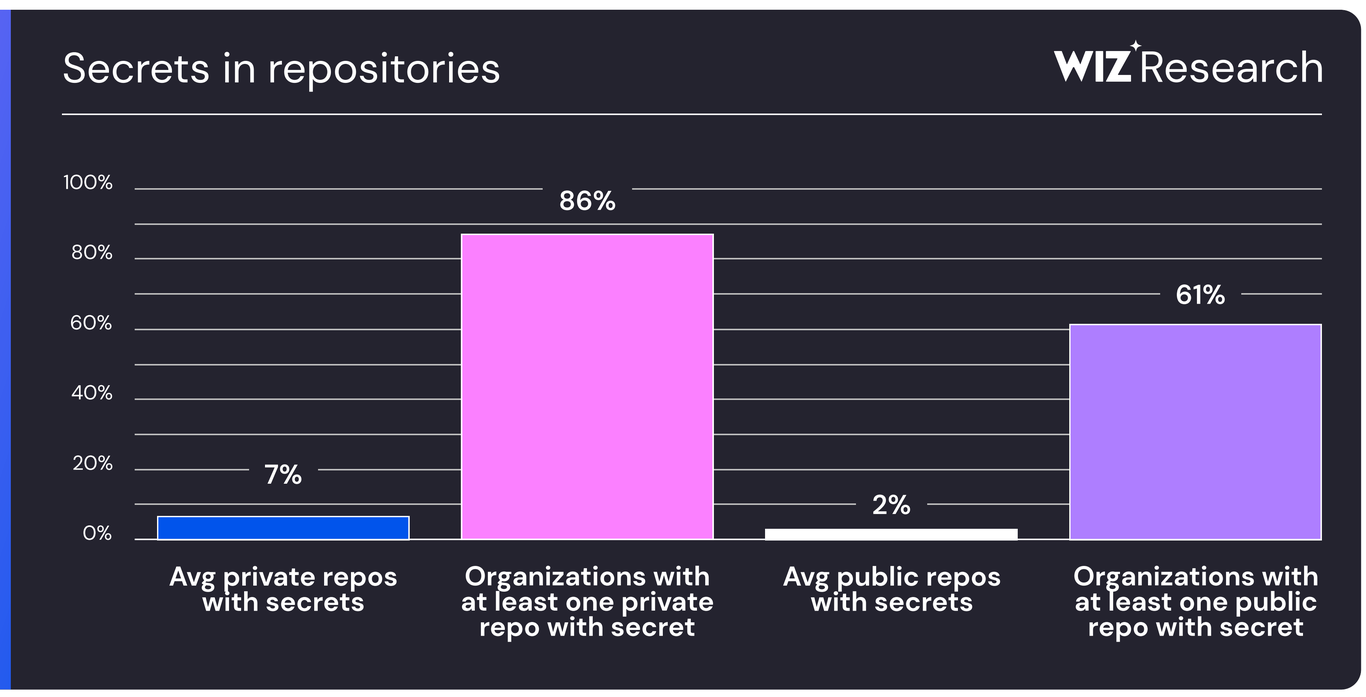
This centralized view simplifies governance and security across the development lifecycle.

Repositories: Key findings

Secrets in public repositories are only the tip of the iceberg

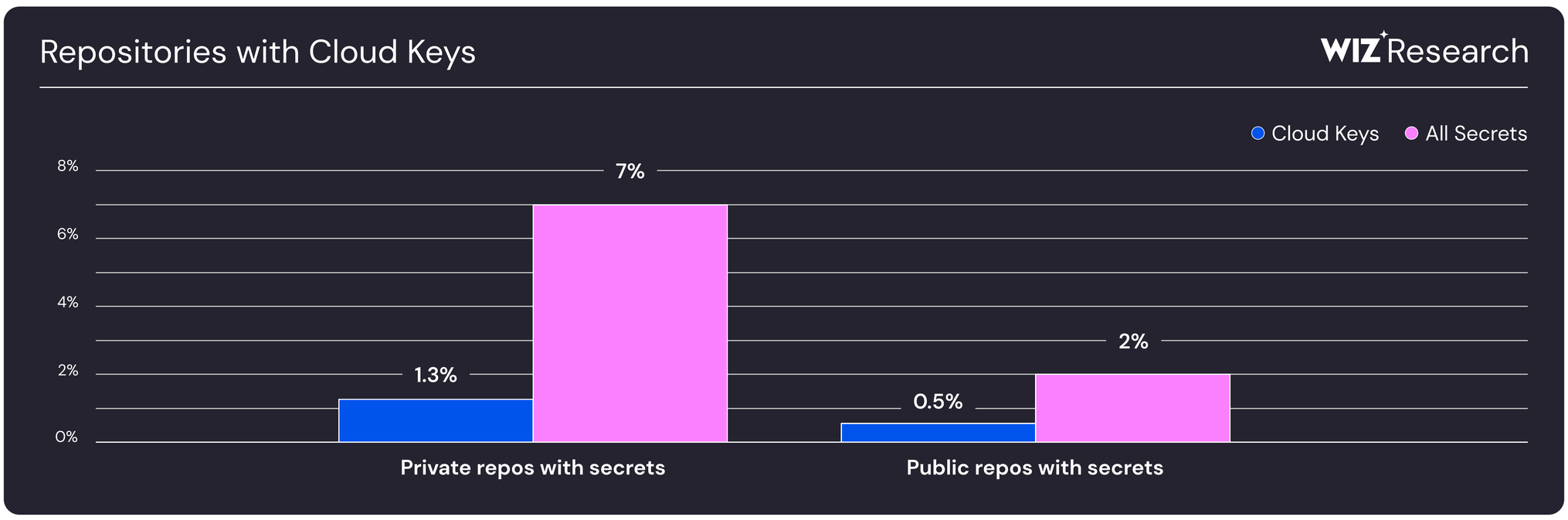
We still find secrets in public and private repositories. While it is heartening to see the number of secrets drop between private and public repos (from 7% to 2%), even private repos are not a good place to store secrets. Secrets must be encrypted and stored inside a secrets manager, a point which we also make in the Wiz Security Best Practices Guide.

Against the best practices, private repositories are used as a place to store secrets



Cloud keys represent a big part of exposed secrets

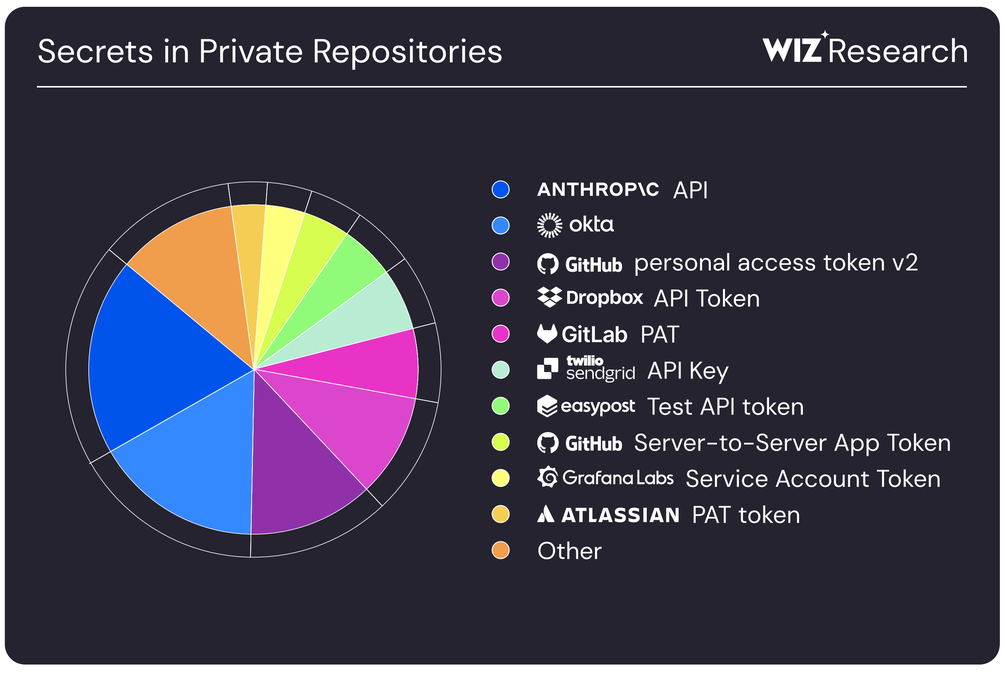
Not all secrets are created equal – some carry greater potential impact than others, if used to malicious advantage. We are particularly interested in the secrets that attackers can use to perform a lateral movement to the cloud, and so we compiled similar statistics on cloud keys:

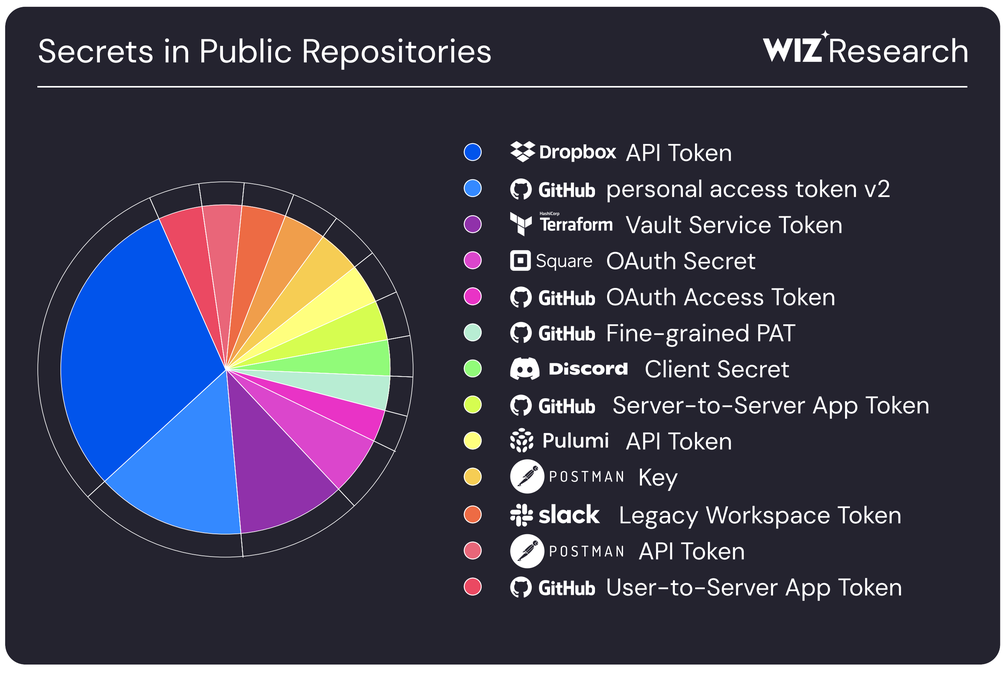


The numbers confirm our suspicions that cloud keys, albeit in a smaller amount, can still be found in private and public repositories. The number is likely even higher for the general population, since our data is drawn from organizations protected by Wiz (Wiz has multiple policies and controls prompting users to fix these issues).

Further analysis into the secret types yields three noteworthy observations. The first relates to the differences between secret types in public vs private repositories. Second: the abundance of VCS tokens. Third: the absolute dominance of SaaS API and cloud keys like Dropbox, SendGrid, and Databricks.

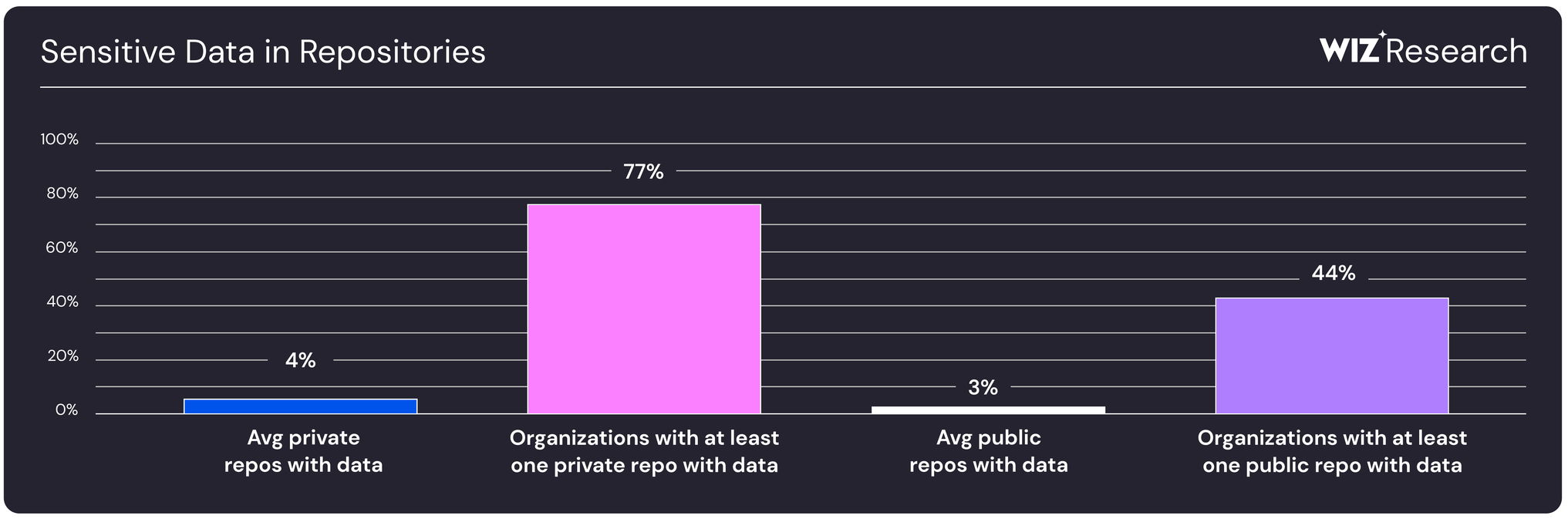
Point number one indicates a false sense of security when storing keys in private repos. Number 2 underscores the potential of lateral movement given the leakage of the initial VCS token. It is especially disturbing to see GitHub Personal Access Tokens (PATs) heading the rankings in public repositories. Finally, the dominance of SaaS tokens and cloud keys illustrates the tight integration between the VCS and the cloud environments.





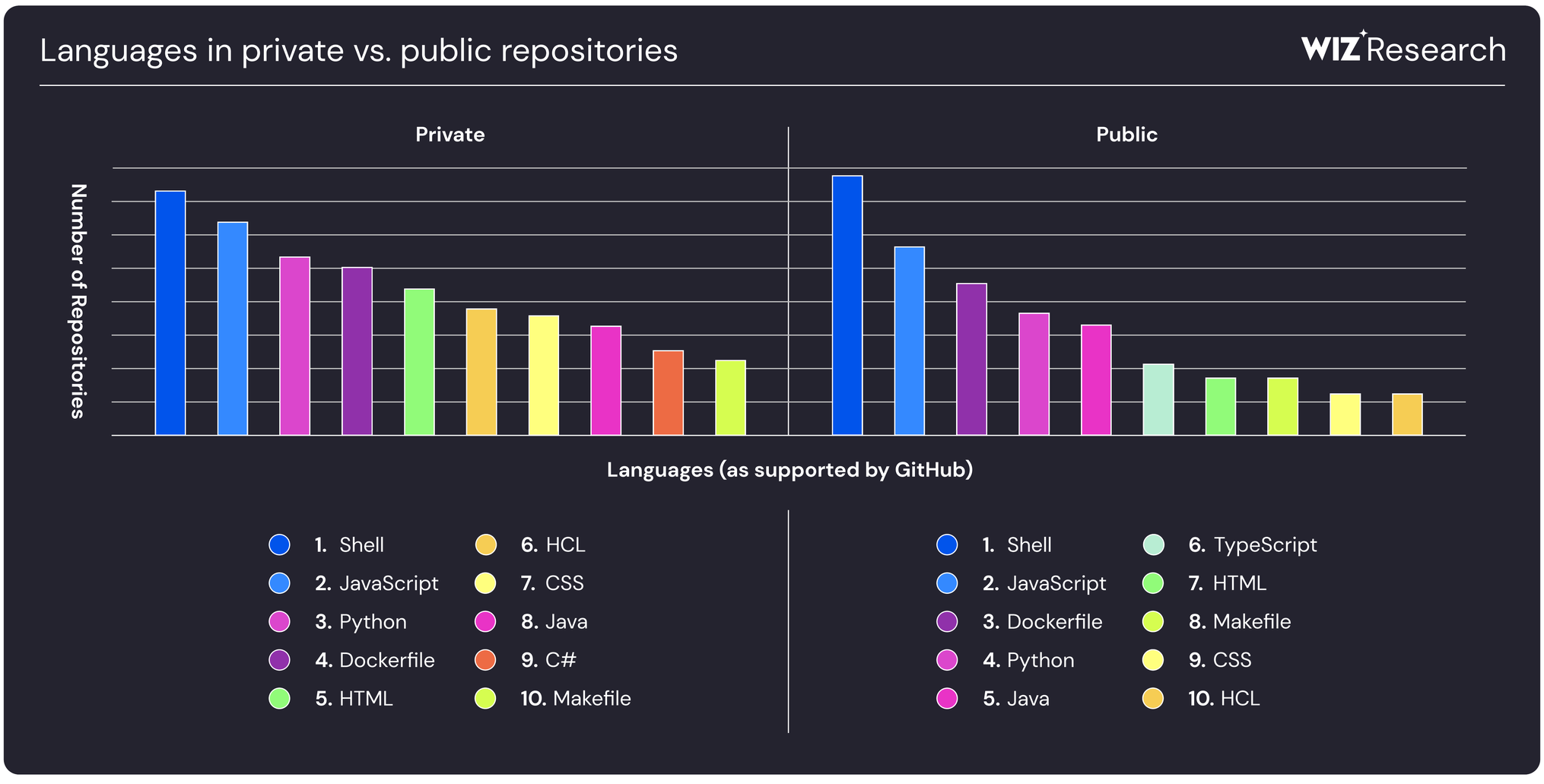
Thanks to the context gleaned from Wiz Cloud, Wiz Code connects secrets found in code repositories with the production cloud environment for quick exposure prioritization and remediation. For example, SaaS keys found in VCS repositories can be effectively validated against the actual cloud service.

There is a similar situation with sensitive data, in which we witness a more moderate decrease within public repos:



Scripting languages are more popular than programming languages

Languages are an important code repository characteristic that affects the security of the resulting product. Not only can it predict the amount and type of the application vulnerabilities, but also prepare the product owner to make more informed decisions regarding the security tools used during the SDLC. We compiled the language popularity numbers based on the language determination by GitHub API. The following are the histograms of the most common languages (each appearing in over 1000 repositories). The lists below show the top 10 most popular languages across private (on the left) and public (on the right) repositories.



Numbers point to the relative popularity of scripts and markup languages, whereas Java, the most popular compiled language, comes in at only seventh. This is true for both public and private repositories. The absence of C and C++ languages in repositories is also somewhat surprising. This may explain or corroborate the decrease in memory-related CWEs in recent years (see OWASP Top 25). In general, the prevalence of scripting languages creates demand for appropriate security tools. For example, if we look at the language categorization of Semgrep (a known light-weight SAST tool) rules in the default ruleset, we observe 530 Python rules vs only 56 C rules. Security and DevOps teams managing SDLC should customize their toolchain to support the languages in the repositories.

Worth noting is that Dockerfile technology appears in over 8% of total repositories, pointing to the prevalence of the containers as a deployment paradigm. Fixing the vulnerabilities and other issues only in containers will not address the root cause issue. Thus, given the containers popularity, the ability to trace the security issues from the deployed container back to the code line is crucial in modern security tools.

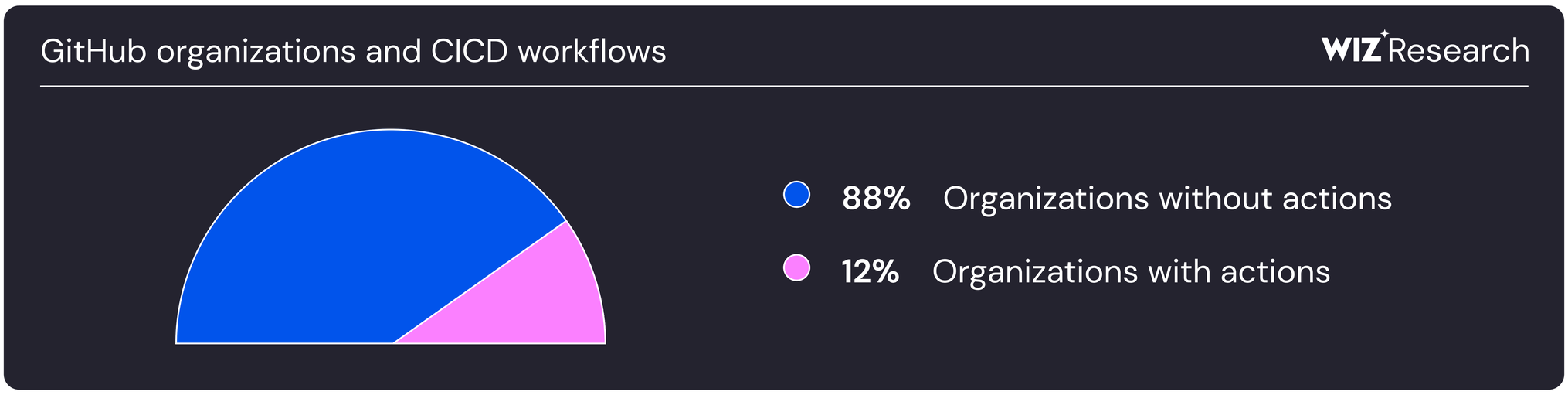
CI/CD security: the story of insecure defaults

For this section on CI/CD security, we have focused exclusively on GitHub for the following reasons:

1. The way the CI/CD infra is implemented is different across the platform and would require different data collection approaches.
2. GitHub is the most popular platform with the highest amount of studied attacks and therefore will be a more interesting subject of exploration than other platforms.

One in every 10 organizations has GitHub Actions enabled

Although CI/CD infrastructure like GitHub Actions offers vast benefits, it also represents an additional attack surface (i.e. 3rd-party GitHub actions, reusable workflows, CI/CD misconfigurations, etc.). Surprisingly, only 12% of GitHub organizations enable actions at the organization level:

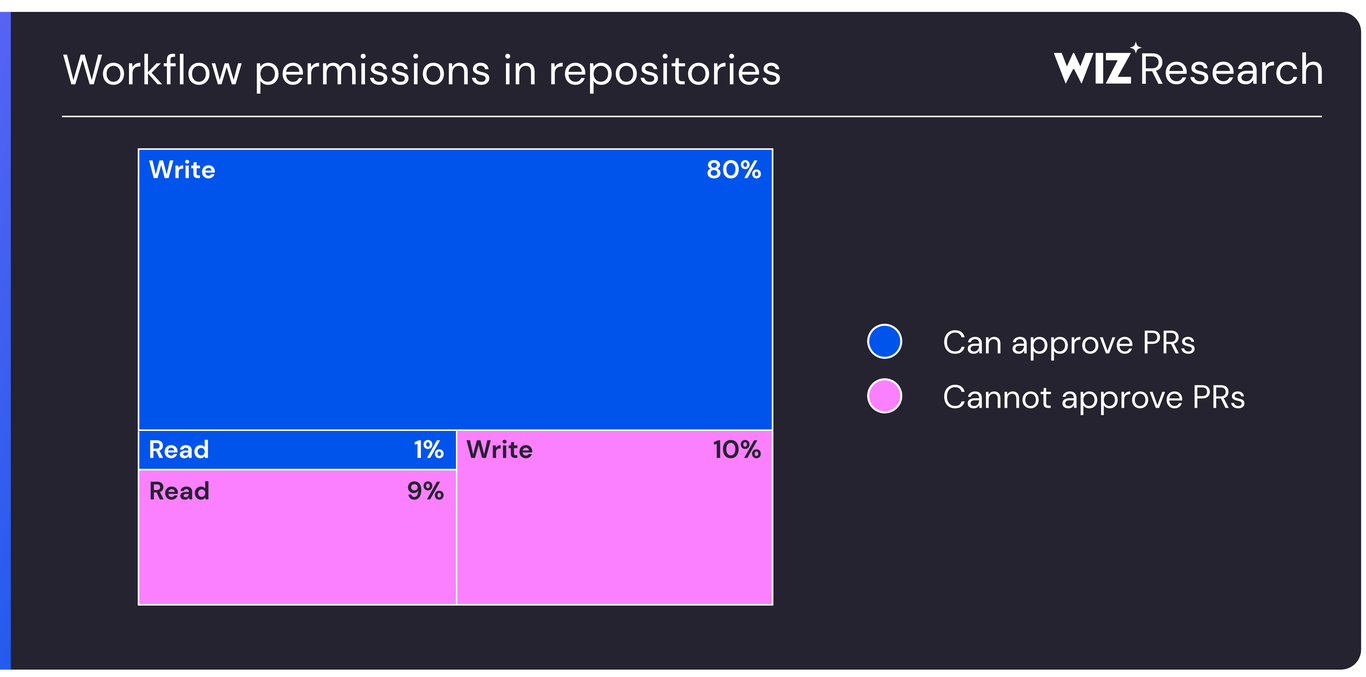


This not only minimizes the attack surface, but also suggests the common usage of VCS as a static data storage or to the contrary, as a space for developer ideation and collaboration without the pressure of building/deploying the final product.

Around 80% of workflow permissions in repositories are insecure

When a workflow job is executing, it is assigned a token with certain permissions. There are two default levels of permissions for the token as documented here. Organization admins can set these permissions at an enterprise, organization, or repository level. It is a good security practice to set these permissions to READ to prevent malicious or accidental writing to a repository. Of course, sometimes this ability is necessary (think linting workflows modifying source code), therefore, even though READ permissions have been the default setting since 2022, WRITE permissions remain an option. Unfortunately, the vast majority of repository workflows in GitHub have the worst default combination of permissions – allowed approval of pull requests (PRs) AND the ability to write content into the repo:

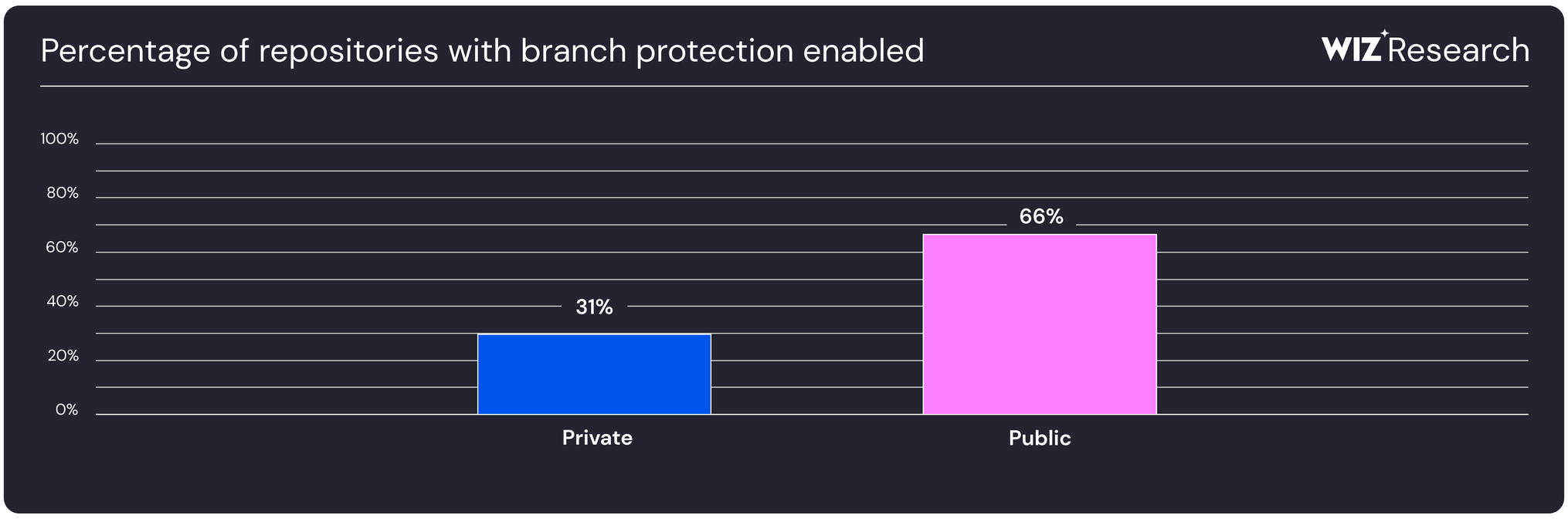
The majority of repository workflows have the worst default combination of permissions: allow approval of PRs and able to write content



There are multiple ways to compromise an existing workflow, the primary methods being command injection into a vulnerable workflow and malicious workflow dependency. Excessive permissions in pipelines offer opportunities for attackers to perform code pushes from the compromised workflows, and thus put additional pressure on detection of suspicious activity inside the CI/CD pipelines.

Branch protection is weak, false sense of security exists in public repos

Branch protection rules define a set of restrictions on collaborators' actions on a branch. For example, deciding which collaborators can force push or delete the branch and determine the push restrictions. Branch protection mitigates multiple attacks, the primary being account compromise and impersonation of legitimate users.



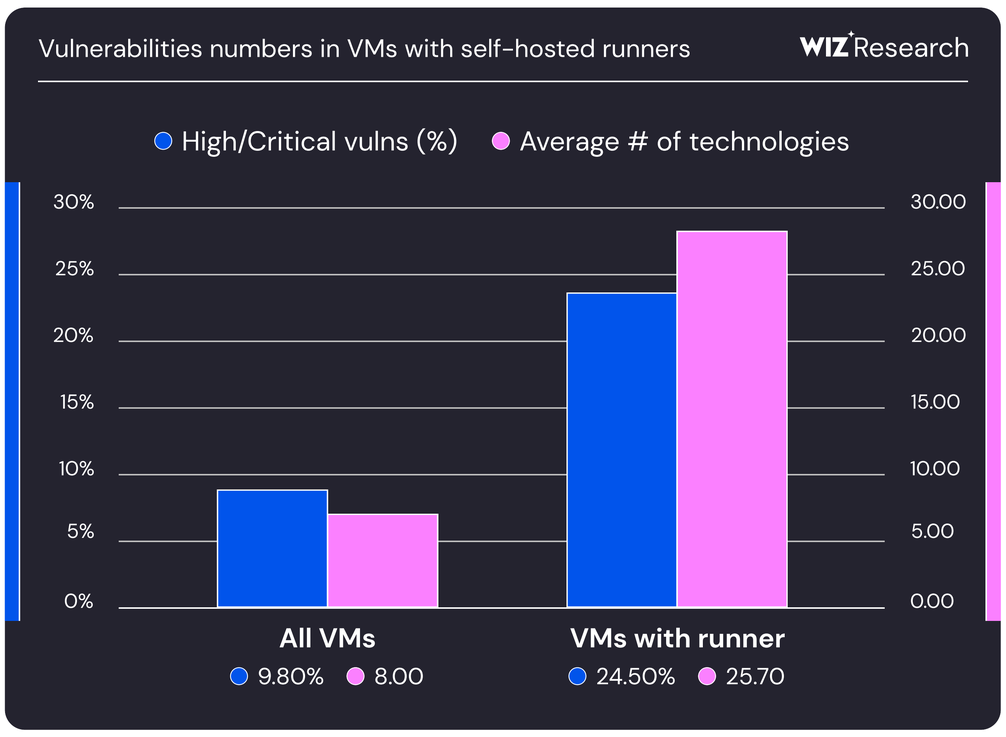
Numbers show the insufficient levels of rule protections in GitHub default branches. There is a significant difference in numbers between the private and public repositories, which again points to a false sense of security. Even for private repositories, credential stealing, and account takeovers are on the rise and this additional protection is needed to stop attackers from committing changes into code.

Self-hosted runners in cloud present a serious risk to 35% of enterprises

There are two types of runners in GitHub that can perform CI/CD jobs: managed and self-hosted. Managed runners are provisioned by GitHub infrastructure and have a high degree of tenant isolation (in fact, a separate VM is created in Azure for every job and then destroyed). Self-hosted runners are managed completely by the tenant and thus are considered less secure (This article is a great primer on self-hosted runners as an attack vector).

We find that over **5%** of GitHub repositories have self-hosted runners configured to run workflows. On the tenant level, however, about **35%** of enterprises use at least one self-hosted runner. We have further confirmed that most self-hosted runners are non-ephemeral and as such can be shared not only between the jobs in the same repository flow, but between different repositories and even across different GitHub organizations. Therefore, the impact of one compromised runner typically exceeds one repository. The numbers show the prevalence of self-hosted runners and underscore the need to build better security practices and robust defenses for this use case.

Self-hosted runners also represent an additional attack vector into your VCS and CI/CD systems. Ideally, VMs and containers hosting a runner should be hardened and isolated and in general have better security practices than other workloads in cloud. What we see, however, is the opposite – VMs with runners installed have on average more High and Critical vulnerabilities than all other VMs, which is contrary to the initial intuition:



We dug deeper for the explanation and were yet again surprised to learn that on average, VMs with self-hosted runners had more software packages installed. On average, **3 times more technologies** were recognized on VMs with GitHub or GitLab runners compared to VMs without. This, of course, explains the differences in vulnerabilities, but not entirely explains why users see CICD infrastructure as a target for excess software. Given the sensitivity of CICD workloads, we urge DevOps teams to harden CICD workloads and avoid software bloating at all costs to minimize the attack surface.

Wiz strengthens build and pipeline security by combining Wiz Code's configuration scanning for VCS and CI/CD infrastructure with runtime protection for CI/CD runners via the lightweight eBPF-based Wiz Sensor. Designed for Kubernetes and Linux workloads, the Wiz Sensor provides real-time detection of threats targeting CI/CD runners.

GitHub Actions – no limits

Usage of GitHub Actions is set at the organizational level. There are two crucial settings that control the degree of action freedom:

1. Repositories: whether all/selected/none repositories are permitted to run actions.
2. Actions: whether all/selected/local actions are permitted.

The numbers show that once GitHub Actions are enabled at the org level, chances are they are enabled at the repository level, too ("All repositories" is the most common setting). Moreover, most repositories do not limit the actions that can be used by the repository workflow use. The most common configuration is: enable all (external and internal) actions in all the repositories in the organization:

GitHub actions - degrees of freedom

WIZ Research

		Actions are permitted on:	
		All repositories	Selected repositories
Which actions are permitted?	all actions / workflows permitted	80%	2%
	selected actions / workflows permitted	17%	1%
	local only actions / workflows permitted	0.2%	0%

This again shows the dangers of insecure defaults: once actions are enabled for the organizations, users rarely change the scope of the actions which, which leaves them at the widest possible execution scope.

Dangerous permission scopes are prevalent in GitHub Apps

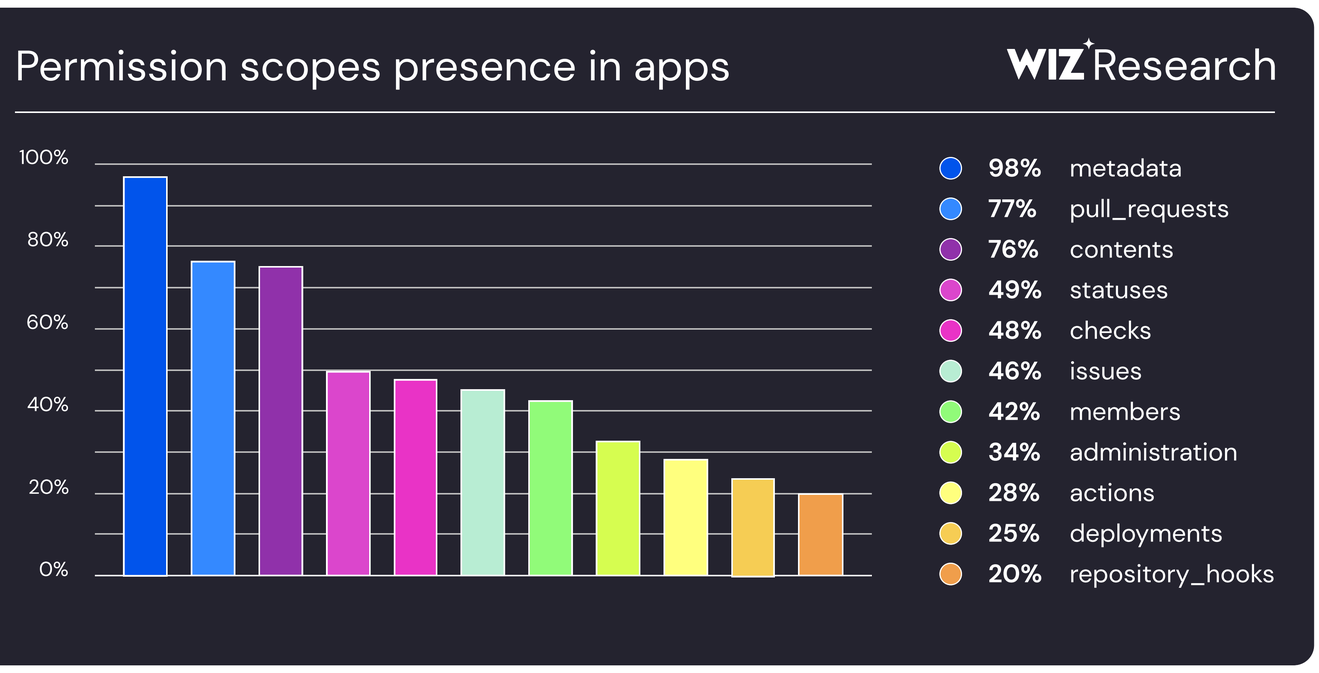
GitHub Apps are designed to augment and extend the existing functionality and workflows on GitHub with commercial, open source, and in-house tools. In fact, this is one of the ways Wiz Code integrates with your VCS environment. After the initial rollout, GitHub Apps quickly gained popularity and became one of the main ways to consume 3rd-party functionality directly in VCS.

When you install the app, you grant the app permissions to read or modify your repository and organization data. Permissions are fine-grained and are grouped into scopes that deal with certain groups of resources (*secrets, pull_requests, issues, etc.*), with each scope having READ or WRITE access to its resources. As such, we are interested to see the presence of scopes and the access of each scope.

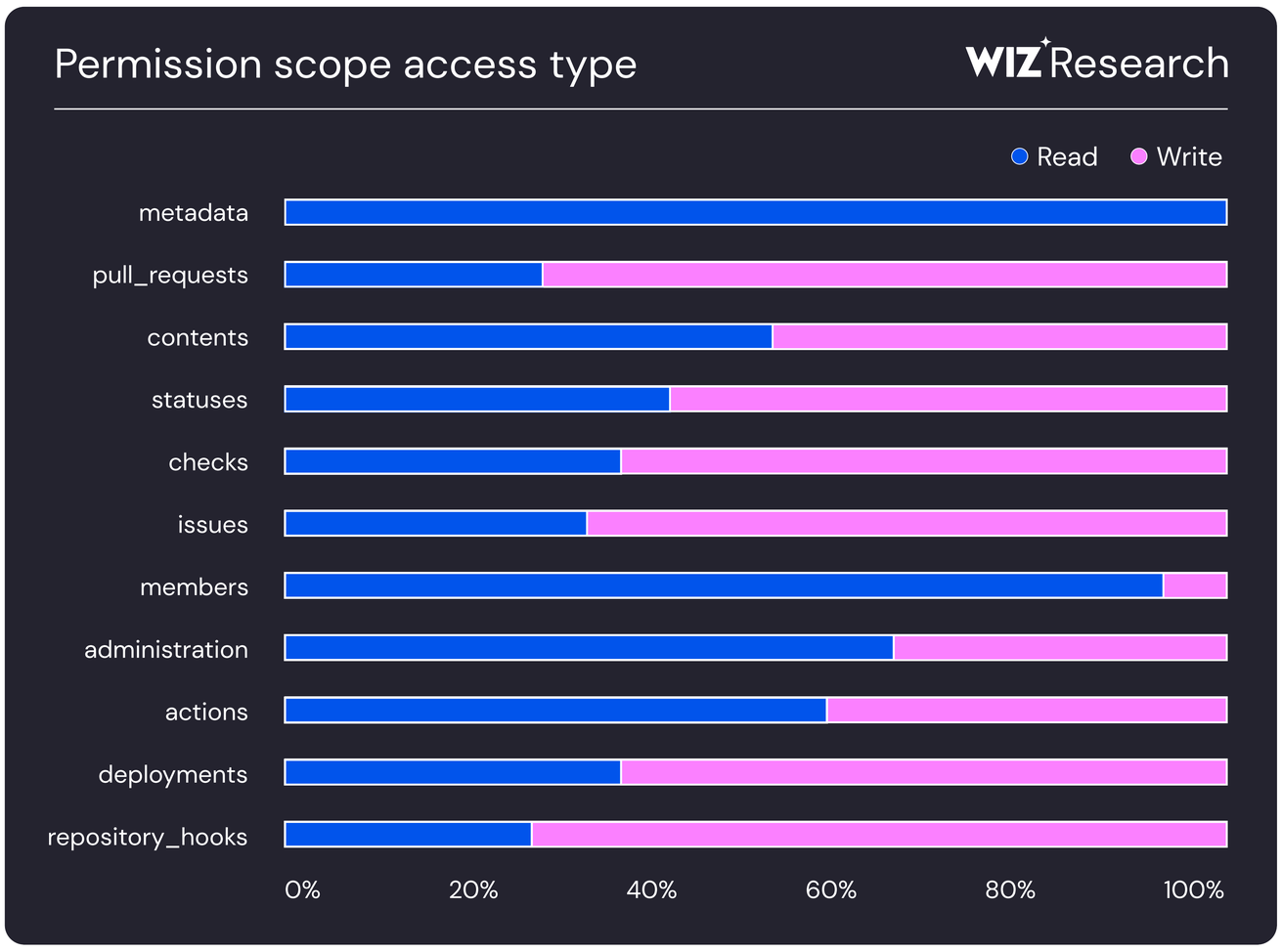
We observe the most popular scopes among the apps are *metadata, pull requests and contents* with all the rest of the scopes trailing further behind:

DANGEROUS SCOPES:

_pull_requests_ and _contents_ are among three of the most popular scopes in GitHub Apps



This is unfortunate, because *pull_requests* and *contents* are powerful scopes that allow read and modification of the repository code. Naturally, the impact of any subsequent 3rd-party app compromise (via supply chain, credential leak, or other means) is proportional to the app permissions. Finally, when slicing the most popular scopes by access types, we observe that the “safest” scope are *metadata* and emails with 100% of apps using it for READ only. On the contrary, *workflows, pull_requests and repository_hooks* are the scopes with most WRITE access type (100%, 80%, 77% respectively):



Wiz Code provides an extensive set of configuration and compliance checks, implementing frameworks such as OpenSSF SCM Best Practices, CIS GitHub Benchmark, and others. This set of controls helps to ensure secure configuration of your VCS and CI/CD environments.

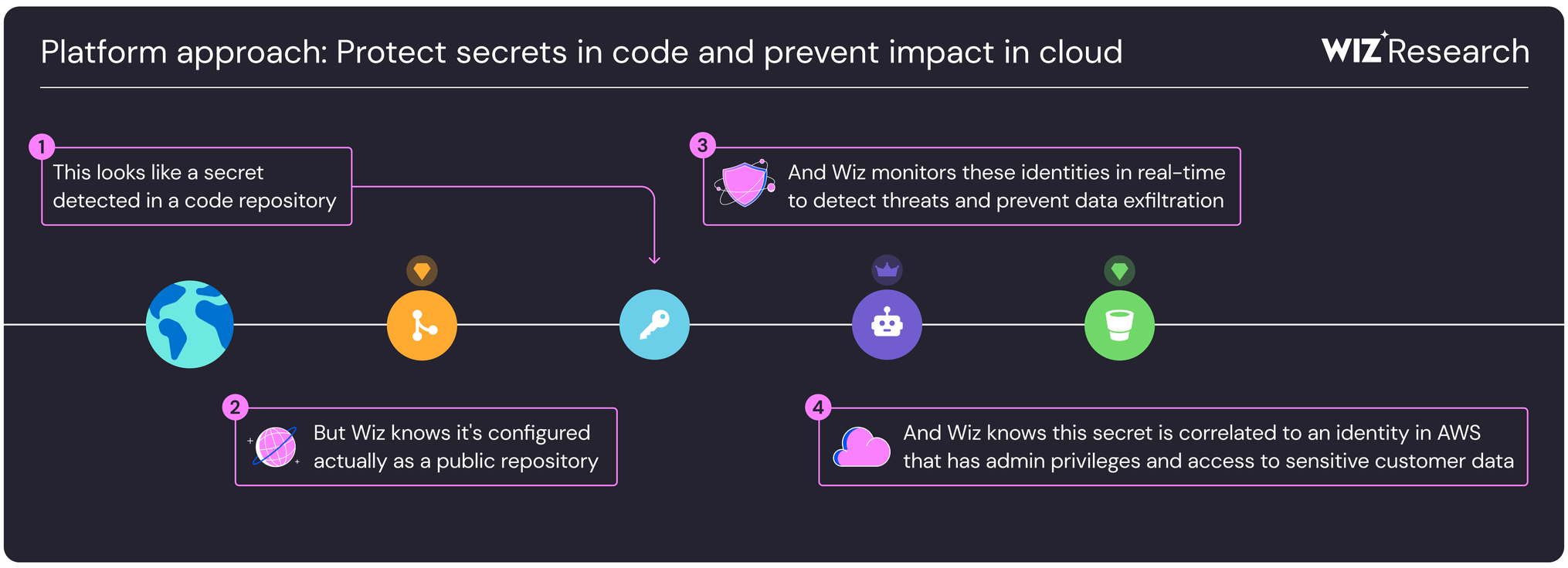
How Wiz Can Help

Wiz connects code and cloud to protect secrets and prevent incidents.

In the cloud, security practitioners must manage risks across the full environment. Secrets such as API keys, credentials, and tokens often bridge these two worlds, making them high-value targets for attackers. Wiz connects code and cloud through a single platform that enables organizations to identify, correlate, and mitigate risks with a comprehensive approach.

The platform approach ensures secrets in code are not just detected but contextualized and secured. Our Wiz Code offering identifies exposed secrets in repositories, while Wiz Cloud correlates them to sensitive configurations, such as a public repository with a secret linked to an AWS identity. We then determine whether that identity has admin privileges and access to sensitive customer data — access which could be gained directly or through toxic combination of risk. In the latter scenario, our graph-based approach will reveal the complex relationships between resources to spotlight validated attack paths. With Wiz Defend, organizations monitor these identities in real-time to detect potential misuse, such as suspicious access or exfiltration attempts, and prevent breaches before they happen. The combined power of Wiz Code, Cloud, and Defend gives customers a unified way to assess risks in context and rapidly respond to threats with confidence.

A critical metric for understanding the scope of this challenge is the proportion of secrets that are cloud keys, as discussed earlier in the "Key Findings" section. We discovered that cloud keys represent a large portion of exposed secrets. With cloud adoption skyrocketing, a significant percentage of exposed secrets are tied directly to cloud environments, underscoring the importance of this correlation. Wiz not only highlights these secrets but also provides visibility into their sensitivity and usage patterns, offering stronger protection for cloud-native applications.



Conclusion

The mission of the Wiz Research team is to view the cloud from the vantage point of an attacker and leverage our observations to help the security community better combat critical risk. Incorporating code security into these efforts makes sense, given how extensively today's organizations have embraced cloud-native approaches and subsequently fused the worlds of code and cloud. We should not look at VCS and CI/CD security in isolation, for several reasons.

First, these systems are tightly integrated with production. The **impact** of an attack on code systems can be amplified if the attacker manages to pivot to the production environment. Knowing and monitoring all potential attack paths is imperative.

Second, some threats require **production context** as they are ultimately manifested either at deployment or run time. Think, for example, about malware or a vulnerability presented in the container image – only when the container is instantiated do we know whether the affected code is actually run and with what privileges, what data access, etc. In other words, "context is king" for effective issue prioritization. Modern applications exist as complex, evolving blueprints that span code repositories, deployment pipelines, and cloud infrastructures. Vulnerabilities in code become inseparable from those in cloud. Yet security is often implemented in silos or vertically, with various teams focusing on distinct stages of application development, deployment, and runtime monitoring.

Attackers operate differently: they seek to exploit toxic combinations of risk in an effort to move laterally across interconnected systems, without concern for any domain or tool boundaries. To stay ahead, security must ensure that issues are addressed from the codebase through to the cloud environment.

In February 2025 the CloudVulnDB project expanded its scope to include GitHub and GitLab, in addition to traditional cloud service providers such as AWS, GCP, and Azure.

Since its inception the database has existed as an open project to list all known cloud vulnerabilities and CSP security issues. The addition of these two new platforms is noteworthy because it underscores some of the key themes of this report – namely, the interconnectedness of modern-day tooling and the need for security practitioners to be able to search across the *full* environment to uncover risk, without blind spots.

Visit CloudVulnDB