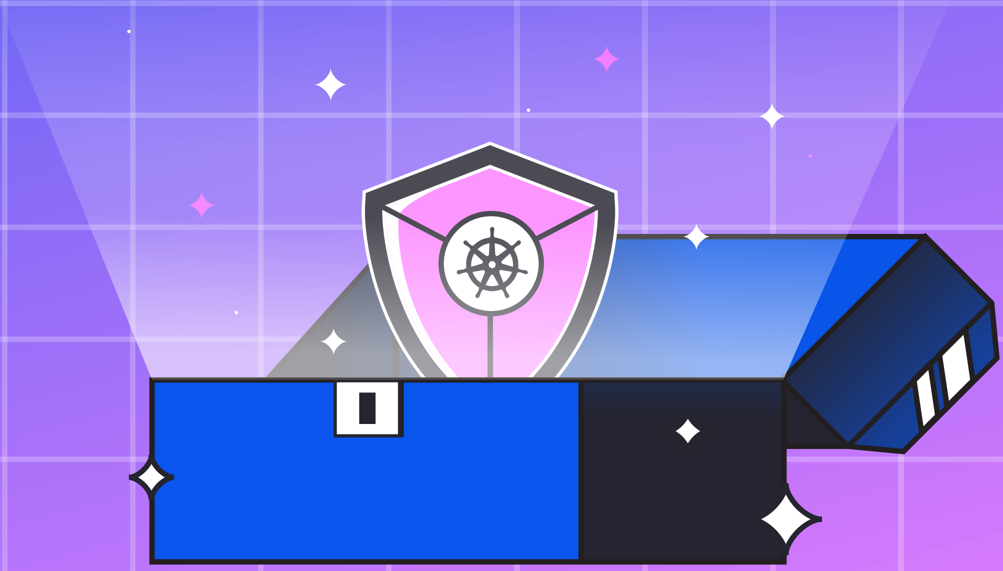
# The 2023 Kubernetes Security Report

The 2023 Kubernetes Security Report, based on our analysis of over 200,000 cloud accounts, uncovers risks in K8s environments and discusses security controls and mitigation steps.



*The data is updated as of August 2023*

# Table of Contents

**Executive Summary** 3

**Data Collection and Analysis** 4

Kubernetes Attack Chain in Detail 4

General Usage 5

Initial Access 7

*Control Plane* 7

*Data Plane* 9

Lateral Movement and Privilege Escalation 10

*RBAC Permissions* 10

*Container Escape* 10

*In-cluster separation* 12

Impact 13

*Denial of Service* 13

*Pivoting to the cloud* 14

Security Controls and Mitigations 15

**About wiz** 17

# Executive Summary

Kubernetes (K8s) has transformed the way applications are deployed and managed in the cloud-native landscape. However, as Kubernetes adoption continues to soar, so do the security risks. The purpose of this report is to prioritize the existing risks based on real-world data, and to highlight appropriate defense recommendations.

## 1 Insights

The Wiz Threat Research team has analyzed the Kubernetes security posture of over 200,000 cloud accounts. In the process of preparing this report, we observed the expected trends (dominance of managed K8s distributions, popularity of EKS etc.), but we also saw statistics that surprised us (such as underutilization of security controls).

By analyzing the obtained statistics in the context of a typical K8s attack chain, we gained the following insights:

- An attacker has the least chance of obtaining initial access through the control plane: the proportion of Kubernetes control plane misconfigurations or vulnerabilities is relatively low. Data plane vulnerabilities offer more opportunities for attackers.
- Once an attacker is past the initial access, the opportunities are ample for lateral movement and privilege escalation within a cluster.
- Impact is the last line of defense, and the security practices there are lacking — especially concerning the cloud impact due to the multitude of options to pivot into cloud. These options will only grow as Kubernetes becomes more tightly integrated into a bigger cloud environment.
- Perhaps the worst trend we've seen is the underutilization of existing security controls applicable across the attack stages. This suggests the need for security feature adoption to be prioritized by the community over the introduction of new security features.

## 2 Recommendations

As K8s cluster operators, we cannot control the rise of attacks, but we can prepare to address them "smartly." This is the premise of this report. To cite a basketball analogy: we propose using a zone defense. In zone defense, instead of each player guarding a corresponding player on the other team, every defensive player is given an area (a zone) to cover. Instead of reactively pairing security controls for every potential attack vector, we propose proactively covering the most vulnerable points and using the wider security options as a backup shield. Specifically, we recommend:

- Continuous scanning for external exposure, and externally-facing security posture reviews for initial access protection.
- Detection and remediation of critical vulnerabilities in the publicly-exposed pods and services — for initial access protection and attack surface reduction.
- Runtime protection — detection of malicious code execution to catch attacks that passed through the initial defense.

- Aggressive usage of in-cluster separation security controls: first and foremost Pod Security Standards; but also smart namespace-based isolation with RBAC, network policies, and user namespaces for prevention of lateral movement.
- Continuous review of IAM and RBAC hygiene of K8s workloads — to constrain the impact of potential compromise at the namespace / cluster level, and prevent pivoting to a broader cloud environment.

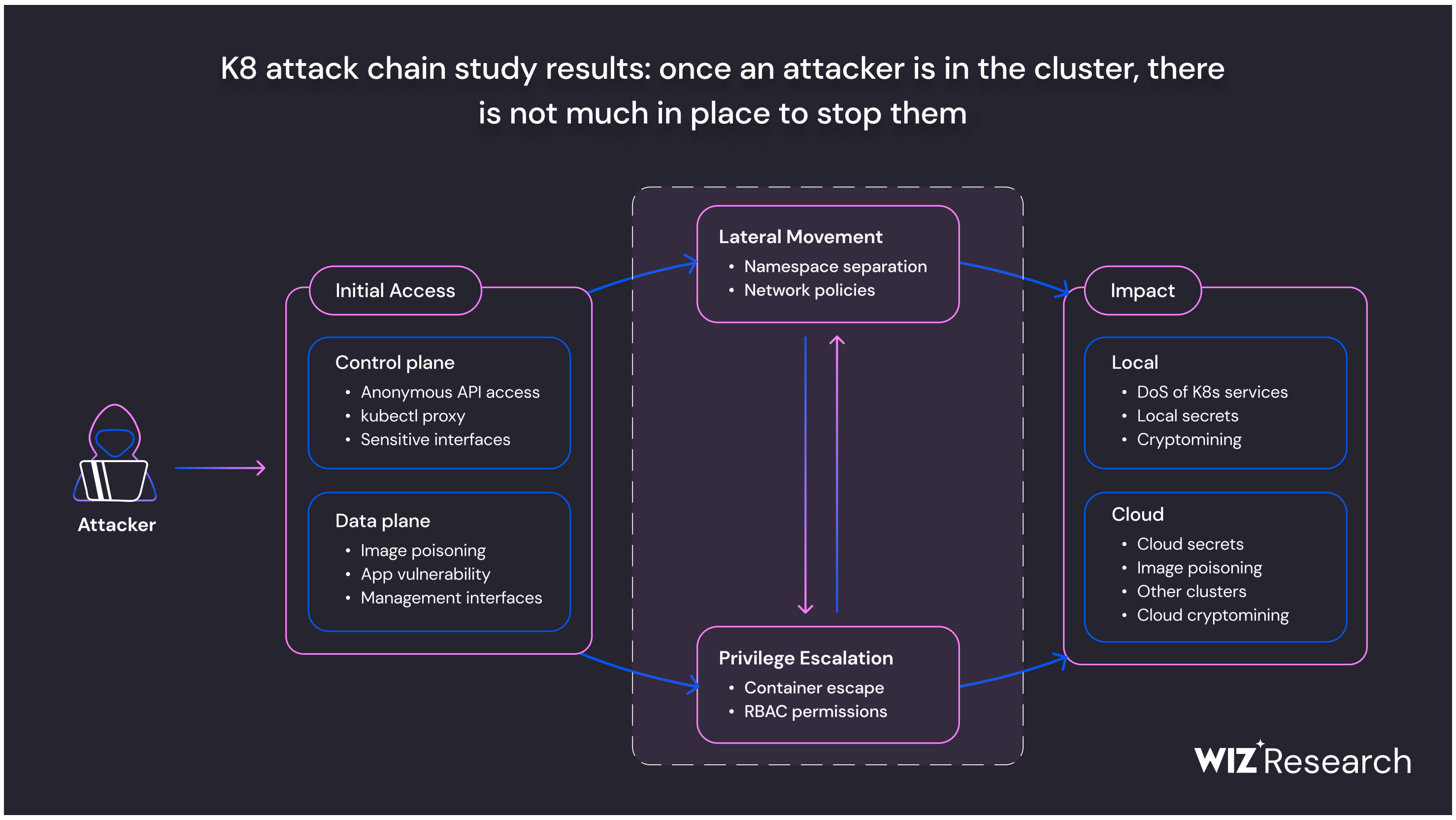
# Data Collection and Analysis

In this report — which is based on our scans of over 200,000 cloud accounts — we dive into the state of Kubernetes security, providing statistics that shed light on the prevalence of threats in Kubernetes environments and uncovering risks that often go unnoticed.

By projecting the observed statistics onto the typical Kubernetes attack chain, we can reason about the most vulnerable points on the ecosystem. That's why, rather than listing statistics in an ad-hoc fashion, we quantify the risks by attributing them to the stages of the full attack path — from initial access to impact. For every attack stage, we apply a series of statistics. To conclude, we also provide a series of statistics on security controls and mitigation methodss. With this report, we hope to enable the cloud-native community to examine issues such as the weakest links in environments and the easiest ways to defend against attacks.

## 1 Kubernetes Attack Chain in Detail

The typical attack chain on a Kubernetes cluster is as follows:

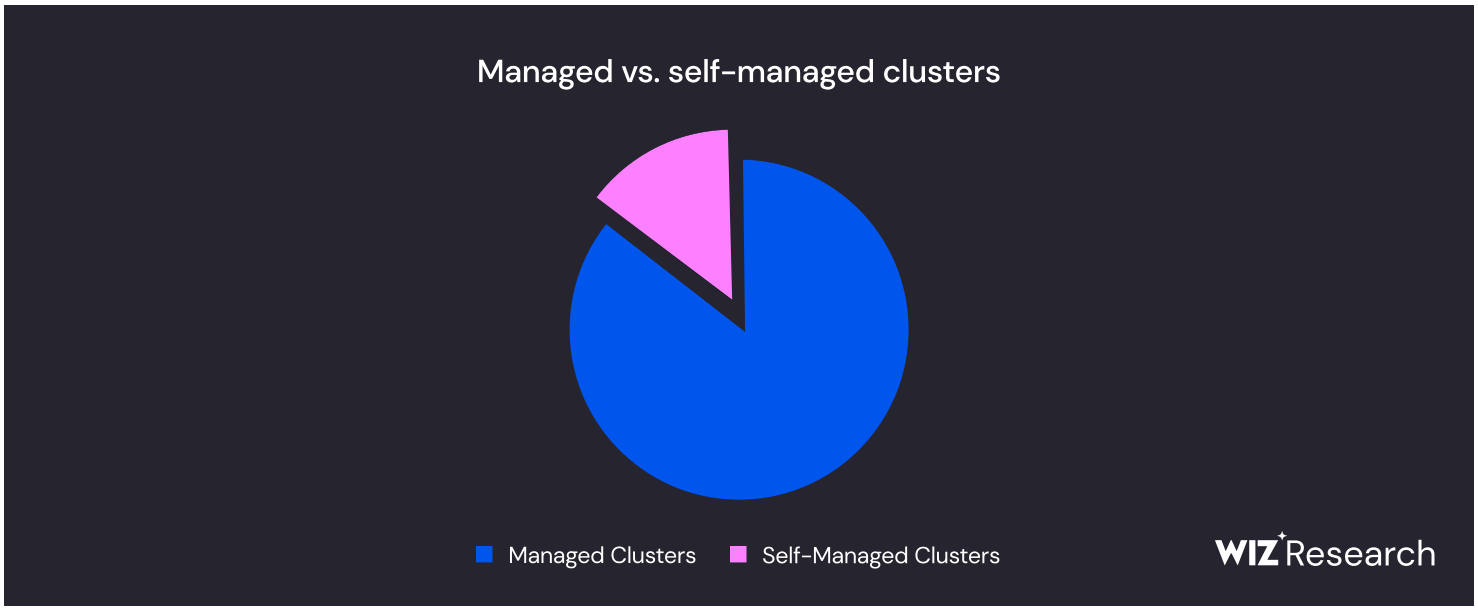


K8 attack chain study results: once an attacker is in the cluster, there is not much in place to stop them

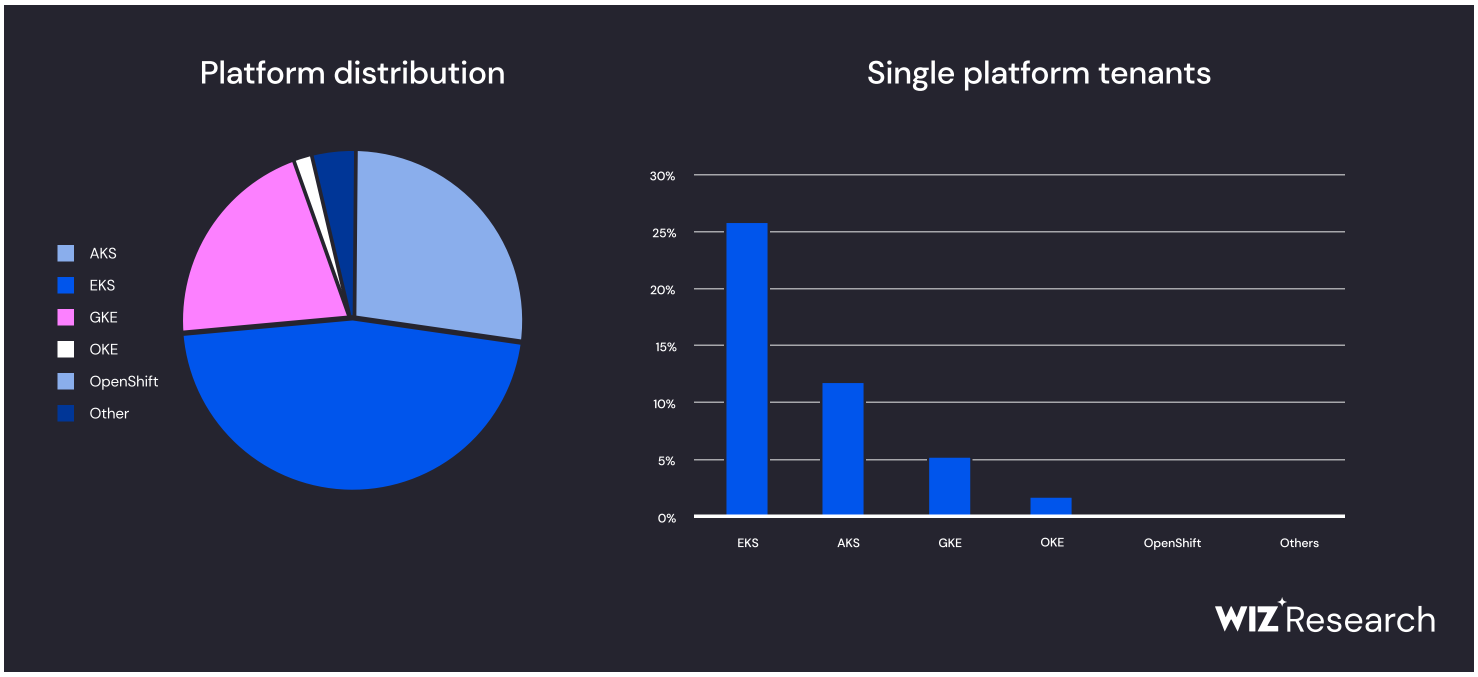
First, the attacker must gain initial access to the cluster — either through the control or data plane. Then, multiple iterations of lateral movement or privilege escalation might follow until the attacker reaches their goal, which is manifested in Impact. For every attack stage — Initial Access, Lateral Movement, Privilege Escalation, Impact – we devise and obtain relevant statistics and offer takeaways. In addition, we show the usage of security controls.

## 2 General Usage

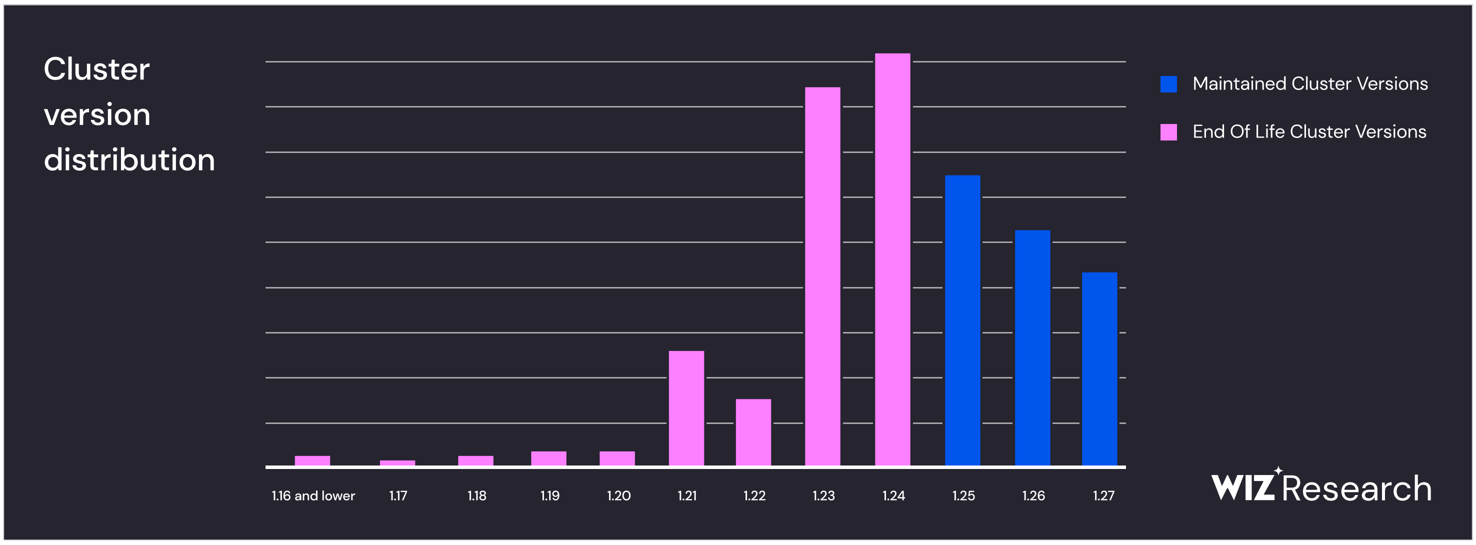
First and foremost, we provide a general Kubernetes usage statistic to define the context of the future numbers. The most basic statistic is the ratio of **managed vs. self-managed clusters** — it shows a definitive preference for managed clusters in cloud.



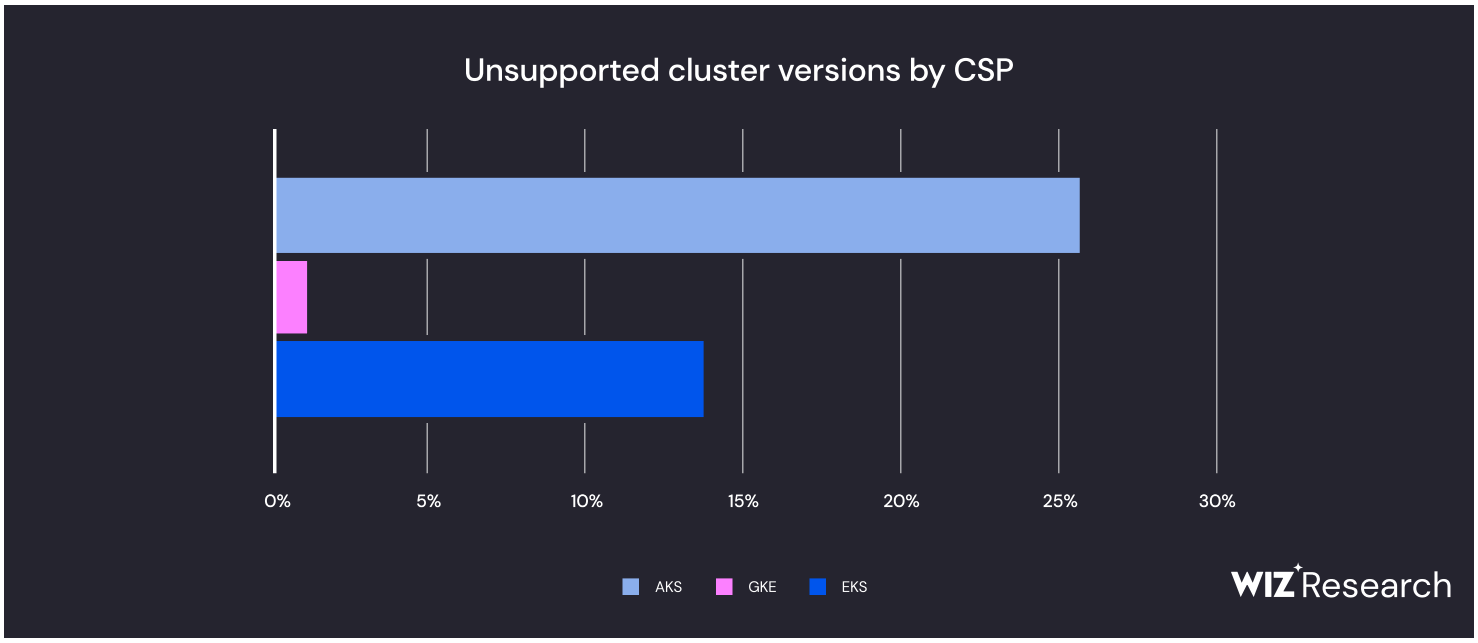
Digging deeper into the popularity of **cluster flavors**, EKS emerges as the most prevalent platform (which is not a surprise), followed by AKS and GKE. EKS is also a leader among those tenants who use a **single Kubernetes platform**.



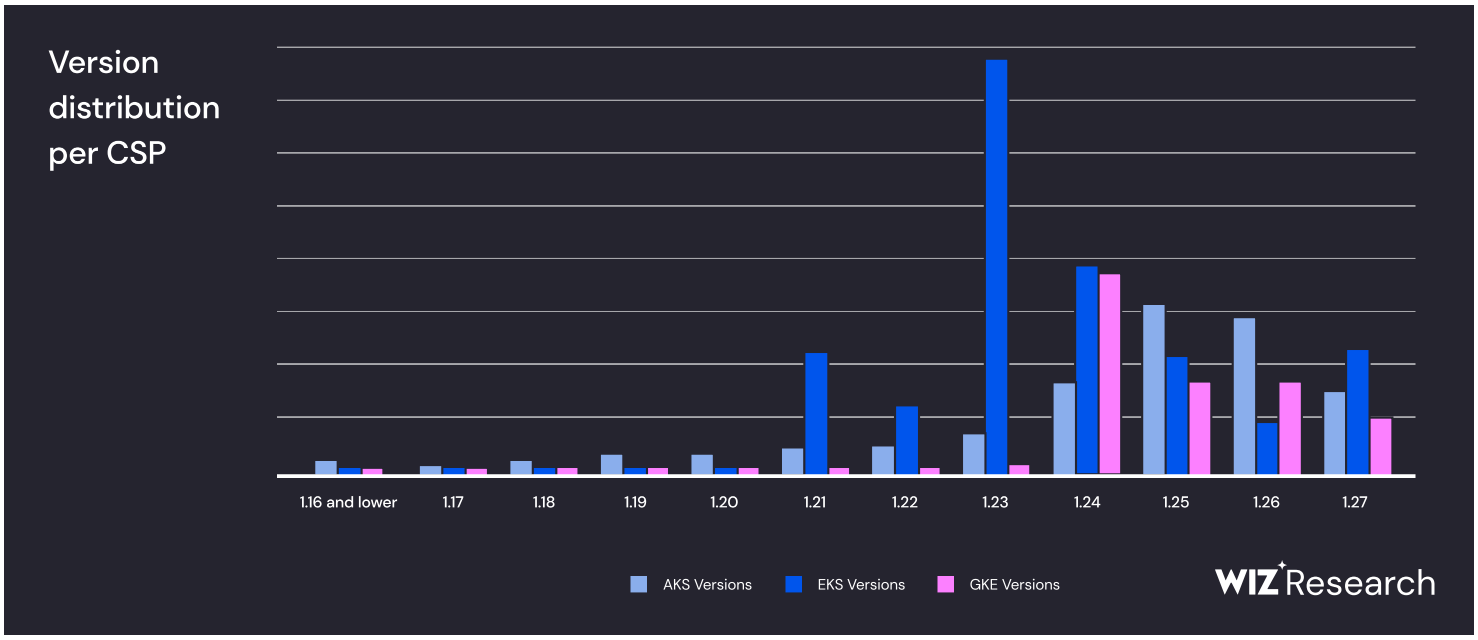
Cluster version distributions show that users generally follow upgrades and adopt the new versions quickly. For example, even though version 1.27 was released in April and adopted by CSPs in June, the portion of v1.27 clusters in use is comparable with clusters on v1.25. However, with the extremely fast release cycle of three months on average, Kubernetes maintains release branches for the most recent three minor releases, which amounts to releases for approximately 1 year. CSPs generally offer a similar, albeit delayed, support span. Hence, prompt version migration in K8s clusters is extremely important. As a reference, v1.24 is upstream EOL since July of this year. As such, over 58% of clusters have actually passed upstream EOL.



Inspecting the CSP End of Support dates, we find the best version update hygiene in GKE, with EKS and AKS having a similar number of unsupported versions:



When slicing the version distribution numbers by cluster flavor, we observe **different distribution profiles per CSP:**



- EKS is generally slower to upgrade. EKS also is the only platform where we observe the concept of the “sticky” versions, with K8s users staying on them longer than others. **The stickiest versions are v1.21 and v1.23**. Given the fact that the last supported EKS cluster version is v1.24, it is surprising to see so many unsupported EKS versions.
- GKE has on average the most updated version profile and does not have older versions running.
- AKS allows users to run older Kubernetes versions, with the **oldest observed version being 1.7** — which reached upstream EOL in April 2018!

## 3 Initial Access

*Recent attacks on Kubernetes infrastructures show attackers’ preference for using control plane misconfigurations for initial access. However, our numbers suggest that this is driven by the ease of exploitation, rather than by the prevalence of control plane misconfigurations. We project that as control plane security defaults improve, awareness increases, and control plane misconfigurations become rarer, attackers will turn to the data plane vectors, which will then provide more attack surface potential.*

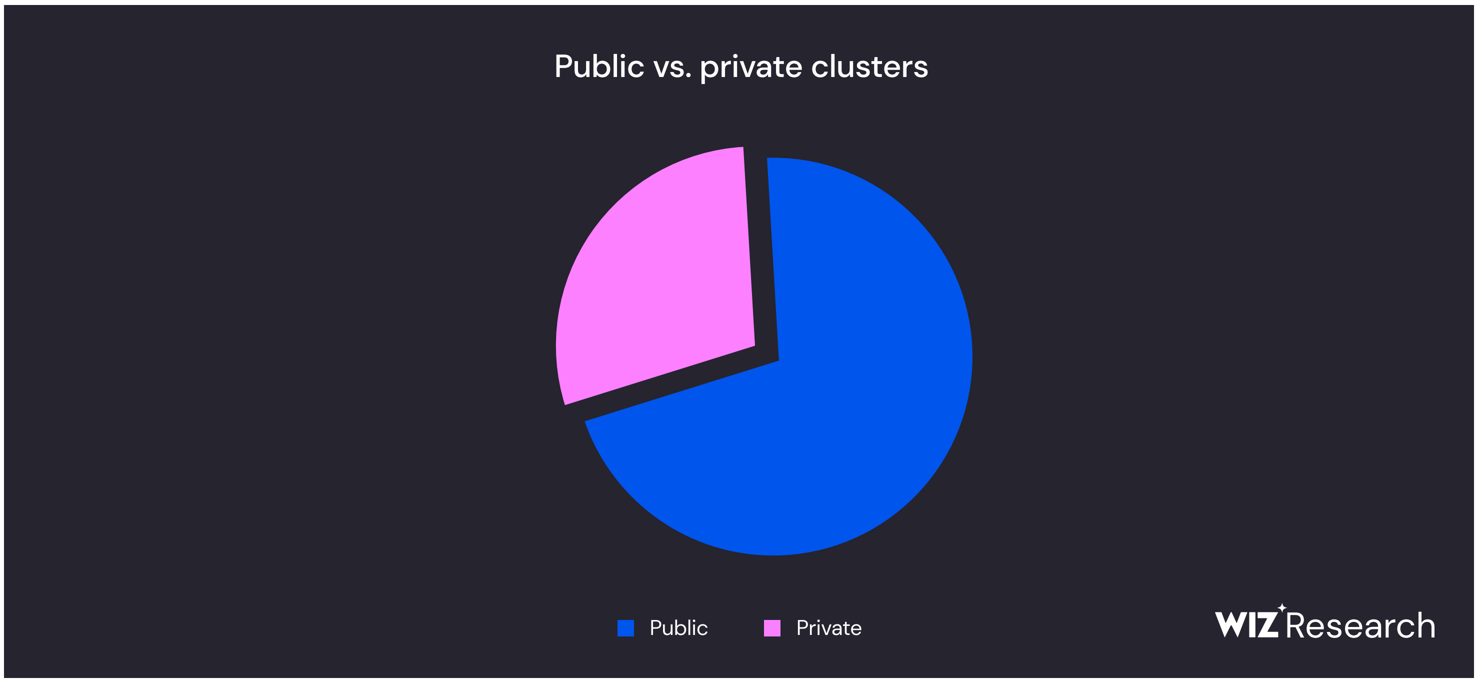
### Control Plane

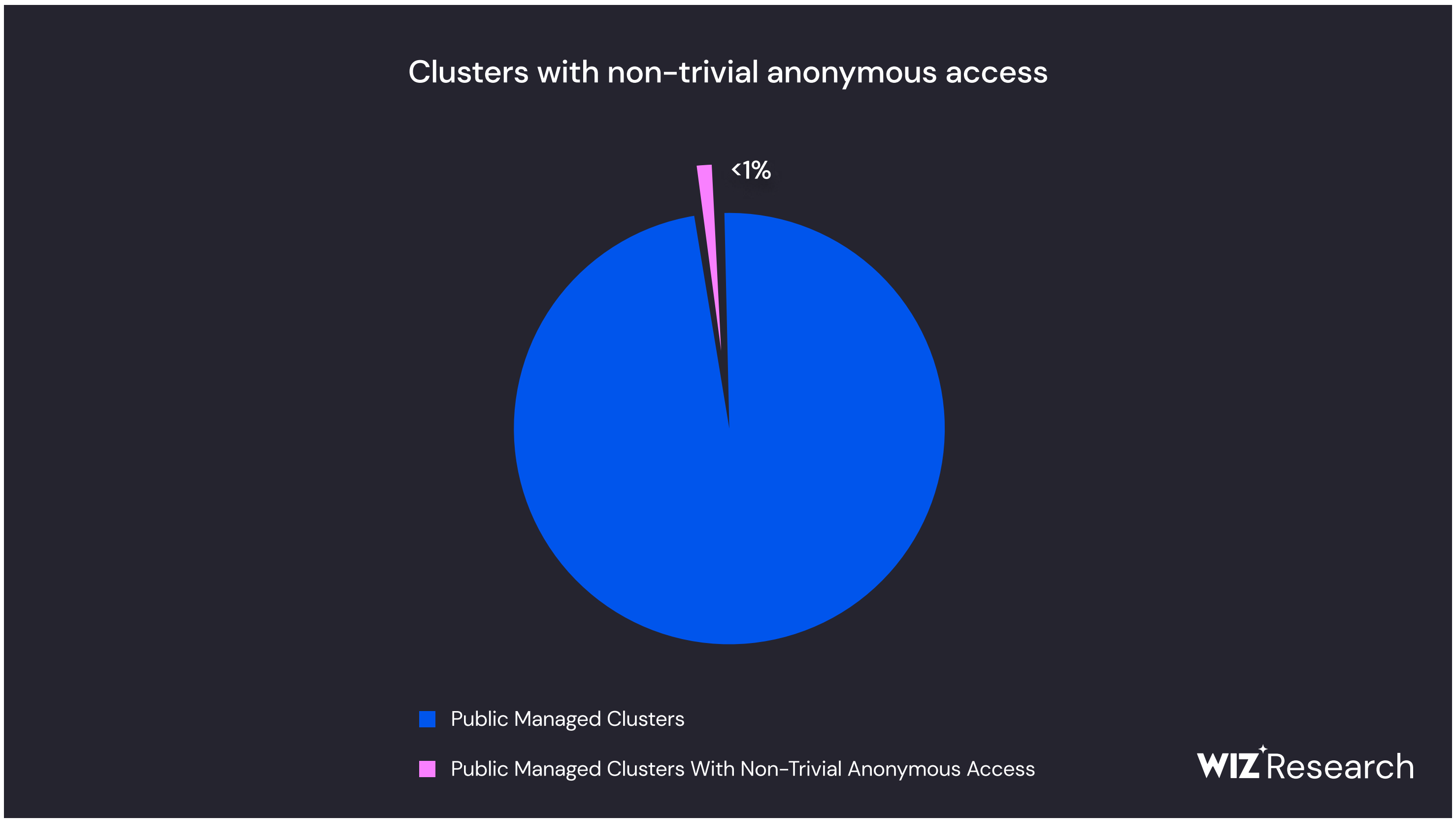
There are several attack vectors that attackers can use to land access to the cluster. They can be roughly categorized in two types — control plane exposure and data plane exposure. We explore both vectors.

Probably the easiest way to get into the cluster is to abuse a public API server with anonymous authentication enabled.

Public clusters are very common, and in fact, we observed that there are more public than private clusters (69%). However, to explore the **API server exposure**, we looked beyond the ratio of public vs. private clusters, for anonymous access misconfiguration.

The first condition for this dangerous API server exposure is enabled anonymous authentication. It is enabled by default on EKS and GKE and disabled on AKS. In addition, a `system:anonymous` user must be bound to an interesting role with a non-trivial access (by default, `system:anonymous` is bound to the `system:viewer` role, which only allows access to basic info about the cluster, such as showing the cluster version). When counting clusters with these conditions, less than 1% of the clusters answer the requirements:

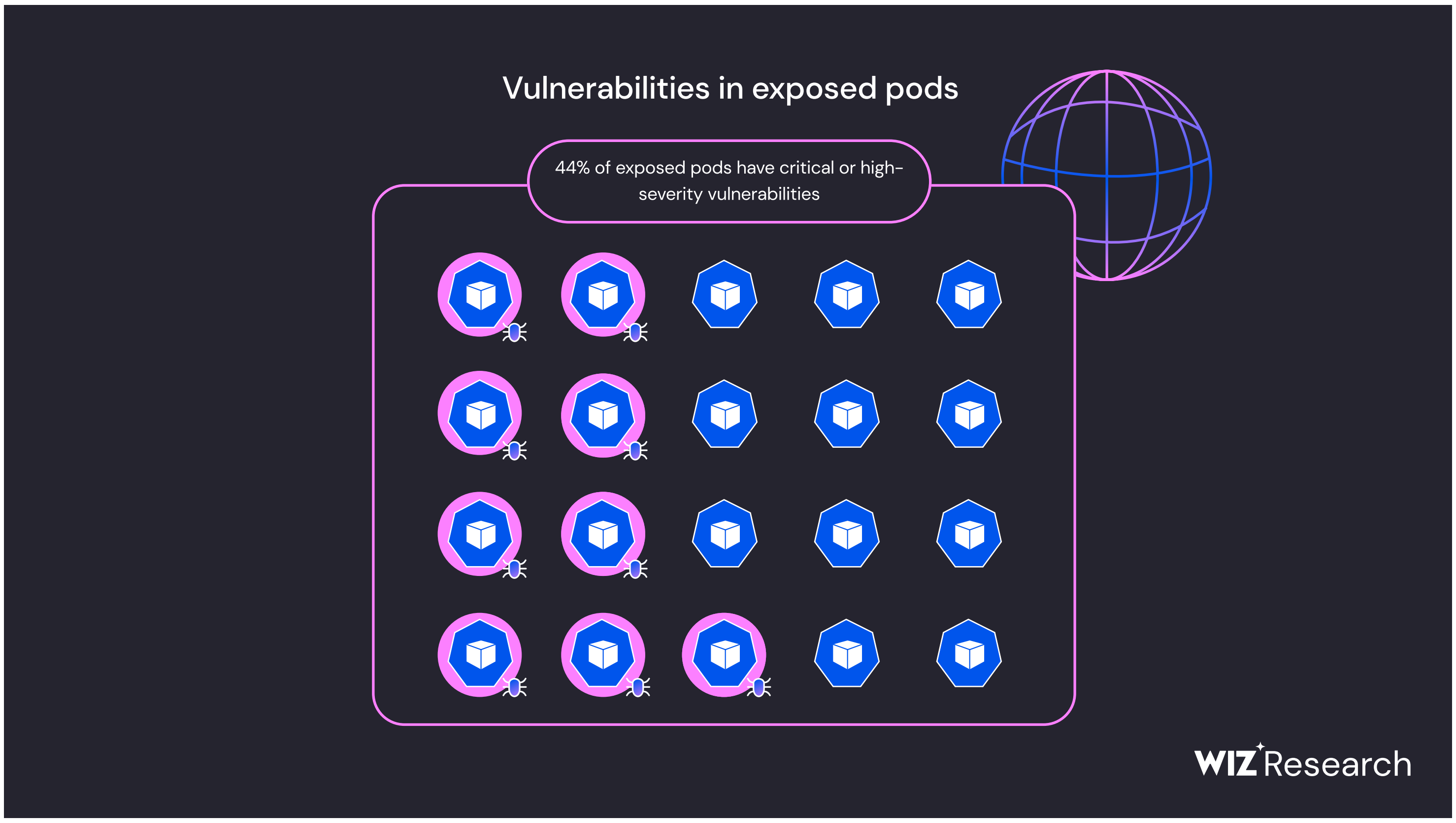

Public vs. private clusters


Clusters with non-trivial anonymous access

Compared to vulnerabilities in the data plane, vulnerabilities in the Kubernetes control plane are often overlooked. We were shocked to still find **clusters vulnerable to CVE-2021-25741** — the last high-severity vulnerability in the K8s control plane since 2021. This vulnerability allows malicious images to access the host file system and, in effect, escape into the host.

## Data Plane

For data plane exposure risks, we looked at pods behind Kubernetes load balancers or ingress that are verifiably reachable from the internet. This provides a more relevant statistic rather than looking at the images across all pods. This analysis showed that **out of total exposed pods, 52% contain container images with known vulnerabilities**. Alarmingly, this number does not drop significantly when restricted to higher-severity vulnerabilities. About 44% of the images have High or Critical severity vulnerabilities. These numbers are generally consistent with other studies that show vulnerability prevalence. Having said that, the vulnerabilities in the exposed containers should be higher priorities than the vulnerabilities in the rest of the data plane.


Vulnerabilities in exposed pods
44% of exposed pods have critical or high-severity vulnerabilities
WIZ Research

We were also interested in knowing how widespread the anti-pattern is when it comes to managing the pods and containers in a "non-Kubernetesy" way. Unfortunately this still happens, although these incidents are very rare: only about 0.27% of clusters include containers with **exposed SSH access**. Management interfaces leading straight into containers can give easy access to attackers bypassing the traditional Kubernetes-level security controls.

## 3 Lateral Movement and Privilege Escalation

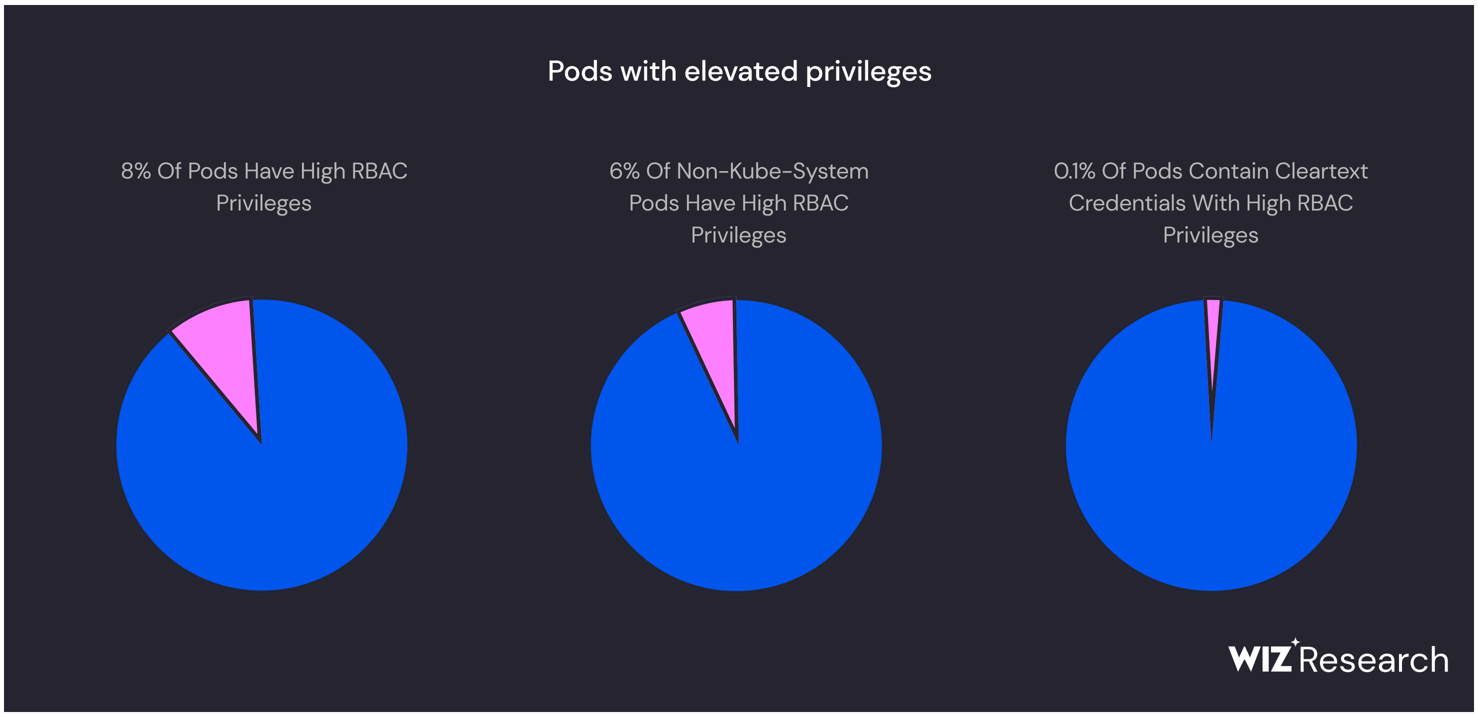
*The numbers show ample opportunities for attackers to move laterally within the cluster — either through the usage of the compromised workload permissions or through a worker node as a shared medium. Security separations in the cluster are underutilized.*

### RBAC Permissions

The proper configuration of Role-Based Access Control (RBAC) plays a main role in safeguarding the integrity and confidentiality of clusters. RBAC allows administrators to fine-tune access controls, granting permissions based on roles and responsibilities. However, when falling into the wrong hands, those permissions can be abused to laterally move within a cluster.

Our analysis shows that 8% of pods possess **elevated RBAC permissions**, potentially extending beyond operational needs. Within non-kube-system pods, 5.9% have elevated RBAC permissions.

Additionally, we analyzed the presence of **cleartext cloud credentials within container images** allowing elevated RBAC permissions. Our findings reveal that only 0.1% of pods employ images with these credentials in cleartext.



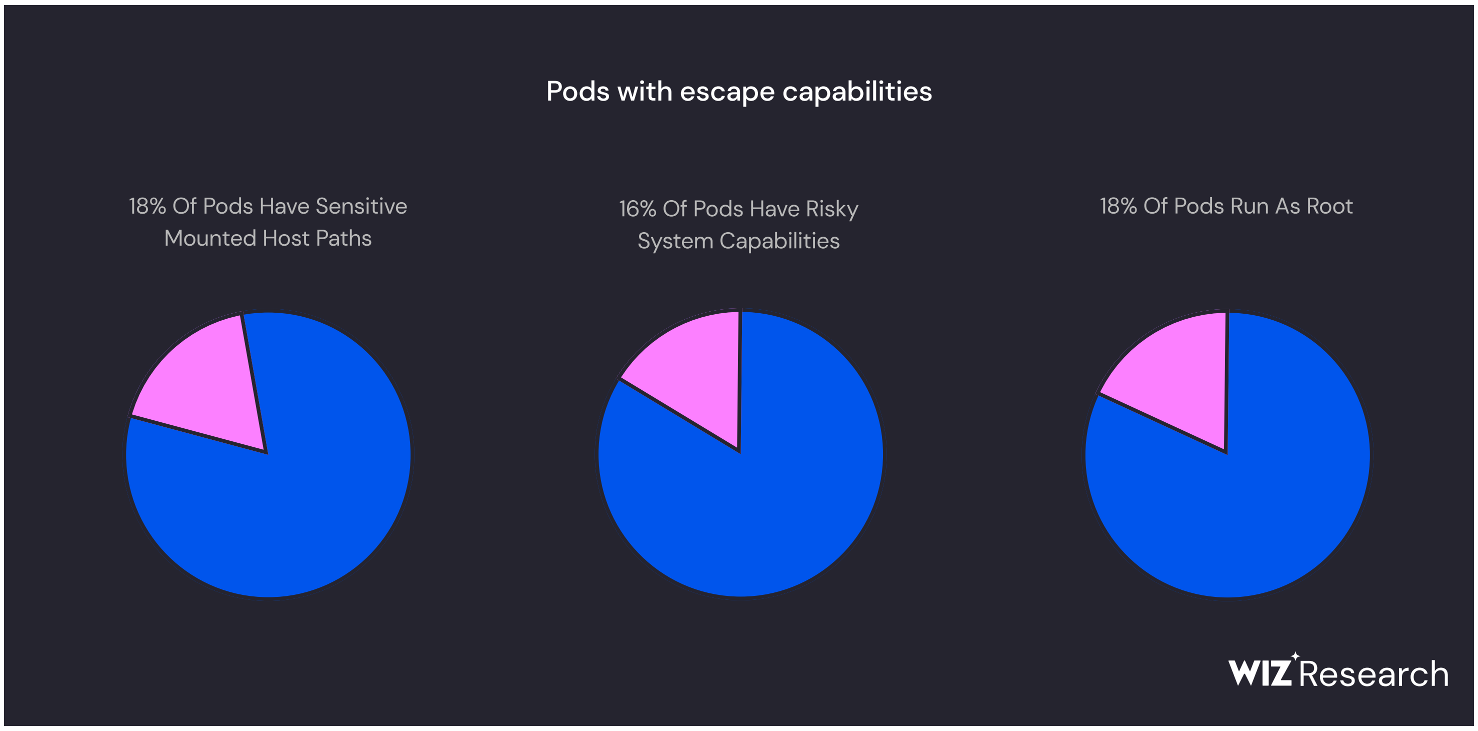
### Container Escape

Having fewer boundaries between container and host increases the potential for unauthorized access, privilege escalation, and lateral movement. This section looks at the different techniques used for container escape, including an analysis and statistics that reveal the widespread presence of such misconfigurations.

The first misconfiguration we examined is **mounts of sensitive external host paths** within pods. Our analysis reveals that among the examined pods, 18% engaged in this practice.

Another significant facet in the domain of container escape is the manipulation of various **system capabilities** assigned to pods. Among these, **privileged pods** stand out — pods granted a predefined collection of system capabilities. Typically, not all these capabilities are required for the pod's intended functionality; only specific ones are truly necessary. 10% of pods run as privileged. Further analysis shows that 6% of pods had specific risky system capabilities assigned. Overall, 16% of pods are assigned capabilities (in one way or another) that would allow container escape.

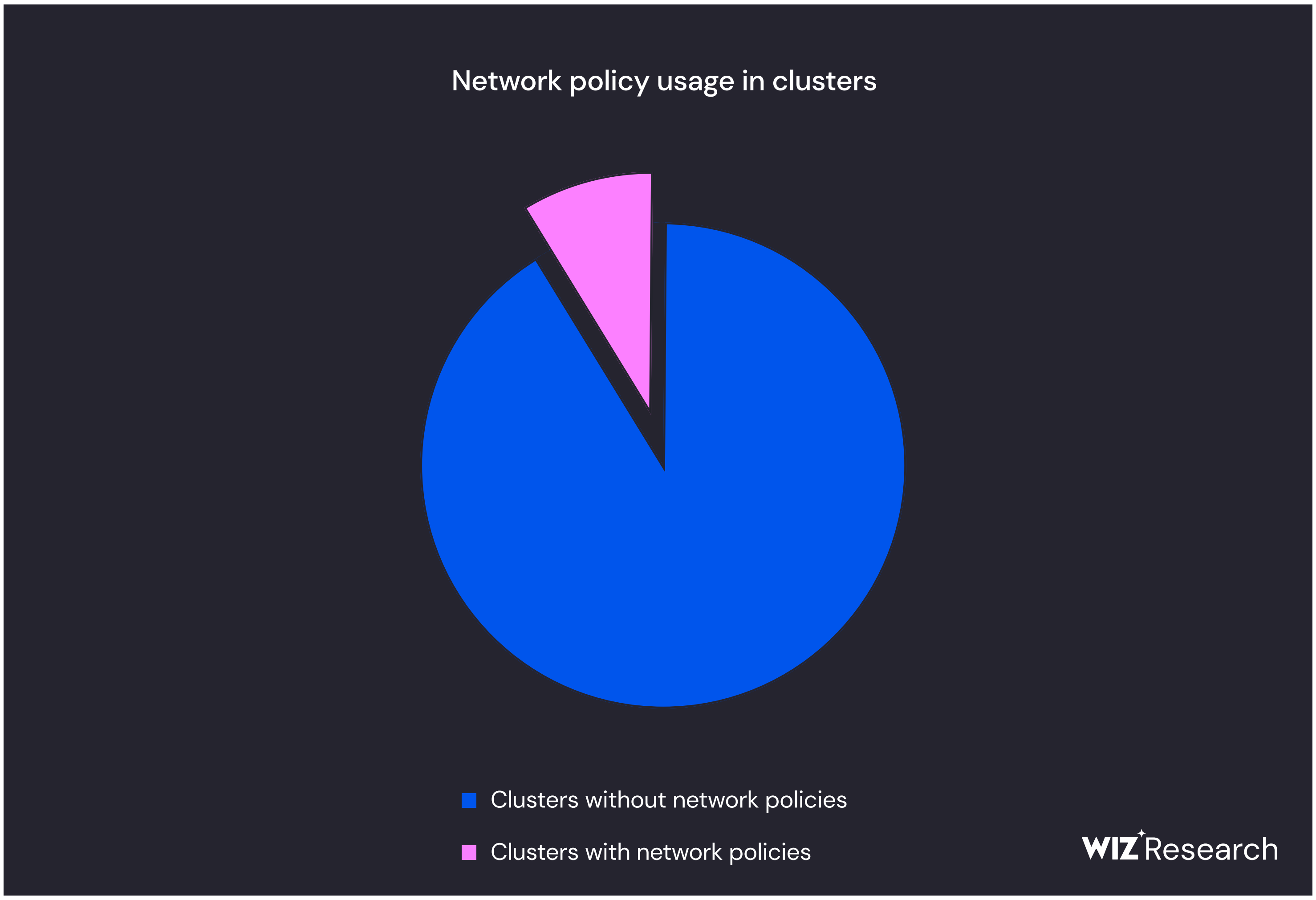
Lastly, container escape presents a concerning possibility for pods that **operate with root privileges**. When a pod runs as the root user (assuming no user namespace separation), it gains unfettered access to the host system, bypassing the typical security boundaries. In our investigation, we found that a notable subset of pods, approximately 18%, are configured to run with root privileges.



Overall, 22% of pods exhibit susceptibility to one or more of the aforementioned misconfigurations. Consequently, attackers can exploit any of these vulnerabilities to move laterally within clusters, potentially escalating their privileges to attain higher levels of access and control.

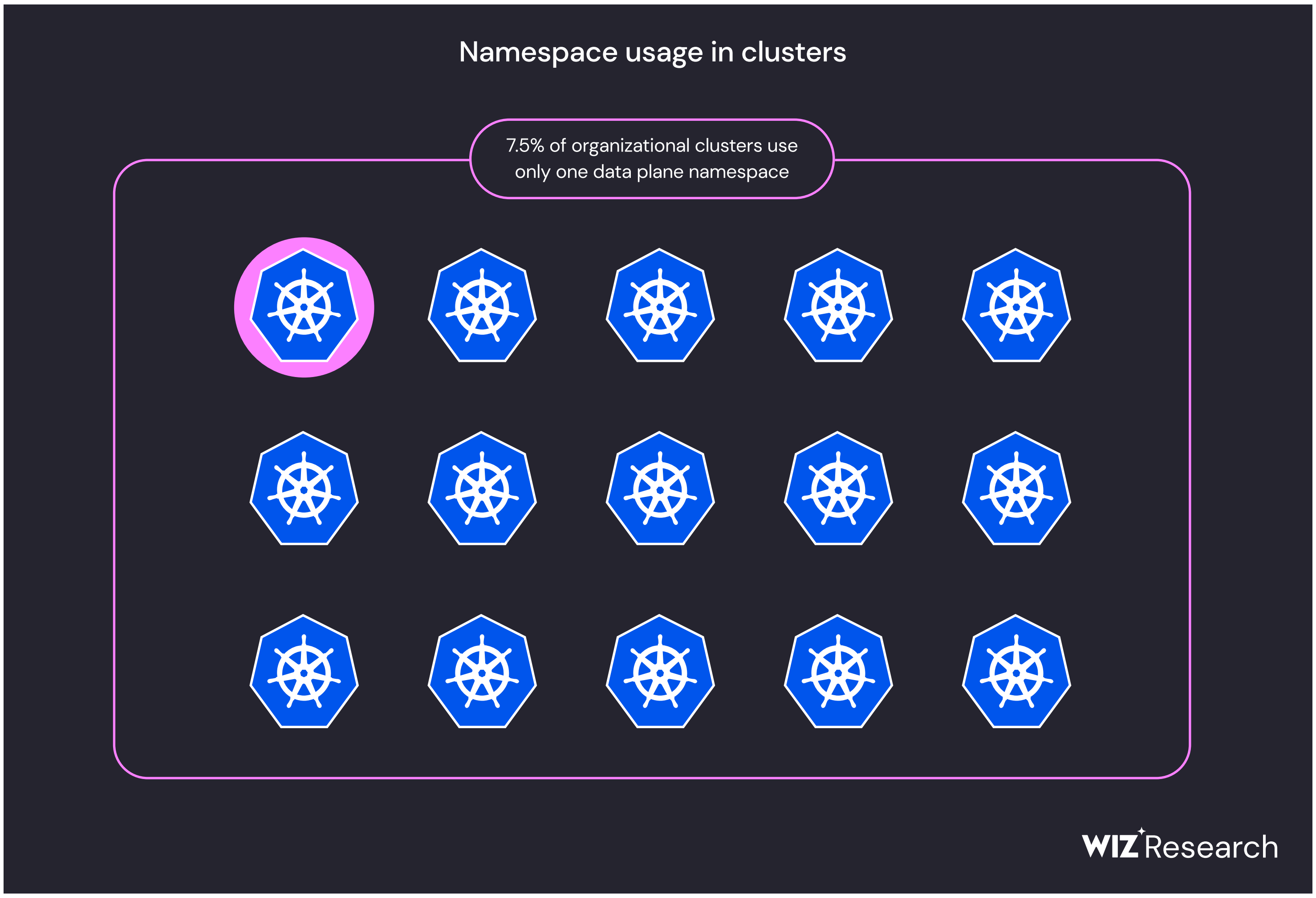
## In-Cluster Separation

By default, there is no network-level separation within the cluster. Pod A in namespace A can communicate with and access the services of Pod B in namespace B. This, of course, enables lateral movement within the cluster. Kubernetes Network Policies define the networking restrictions within the cluster on a policy level. We found that, among the observed clusters, **only 9% have namespaces with network policies**. In our opinion, the multi-tenancy in Kubernetes should be given more security attention through better namespace-isolation controls. Potentially through extension of the existing security frameworks (i.e. PEACH).



Kubernetes clusters are usually organized into namespaces. This structural design not only establishes a clear logical separation, but also contributes to security delineation within the cluster by segregating resources and principals into distinct namespaces. A barrier is established against lateral movement for potential attackers.

Our numbers show a **bad practice of grouping the workloads into the same namespace: about 7.5% of clusters have only one data plane namespace** (default).


Namespace usage in clusters
7.5% of organizational clusters use only one data plane namespace
WIZ Research

## 4 Impact

*Best practices for cloud permission usage, although introduced some time ago by CSPs, are not followed by most cluster operators. We still too often see over-provisioning of cloud roles and usage of the worker node identity.*

### Denial of Service

Compromised pods lacking **resource quotas** can potentially serve as vectors for resource exhaustion attacks. Malicious actors can exploit the absence of resource constraints to consume excessive CPU, memory, or other resources, eventually leading to Denial of Service (DoS) or Distributed Denial of Service (DDoS) attacks, causing significant disruptions and rendering legitimate applications or services inaccessible due to resource depletion.

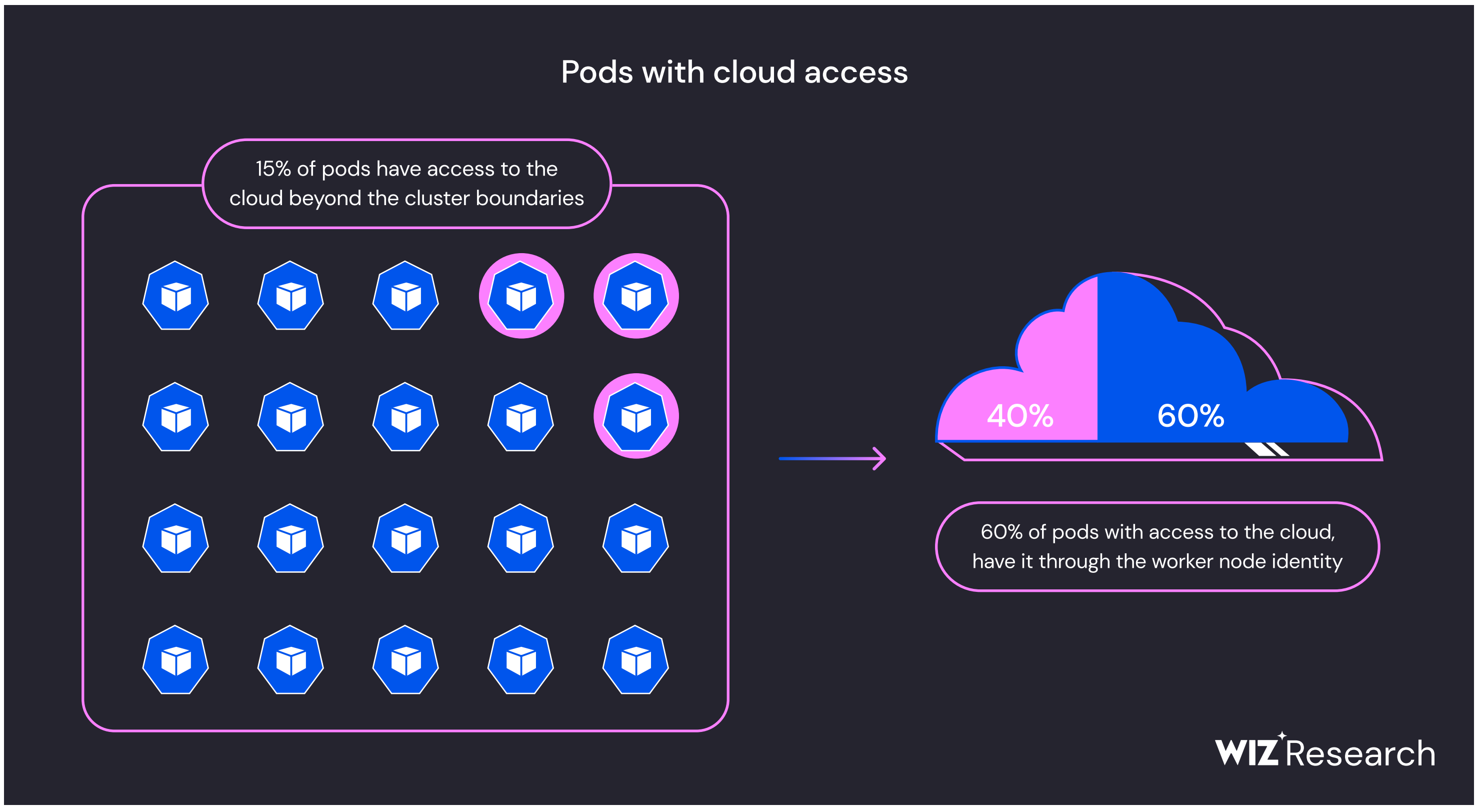
Our research shows that 19% of pods are currently running without resource quotas in Kubernetes clusters. This implies that a notable portion of the cluster's workloads are not subject to resource limitations, potentially creating an environment where certain pods can consume resources disproportionately. Proactively enforcing resource quotas for all pods becomes crucial to mitigate the threat of resource-based attacks and to ensure fair, efficient, and secure resource utilization across the environment.

## Pivoting to Cloud

A critical consideration arises when workload permissions must extend beyond the confines of the Kubernetes cluster itself. In such a scenario, the cloud environment becomes susceptible to potential attacks if adversaries manage to compromise the pod.

Pods are capable of acquiring cloud permissions in two ways: either by inheriting the worker node identity (the older and not recommended way), or by using more secure CSP-provisioned mechanisms (such as IRSA in AWS). The latter method enables you to securely assign AWS IAM roles to Kubernetes Service Accounts (SAs) assigned to pods. This delineates two distinct pathways by which a pod gains the authority to interact with various resources and services.

Among all the pods scrutinized, a significant 15% possessed **permissions that reach outside the cluster**. This revelation underscores the unsettling prospect of unauthorized infiltration into the broader cloud environment.



Further examination of the data revealed a significant trend: 60% of pods with permissions outside the clusters obtained them through the **worker node identity**. This contradicts best practices, as it leads to a lack of granularity in permissions. All pods on a node sharing the same permissions can result in over-privilege, increasing the risk of breaches by expanding the attack surface.

## 5 Security Controls and Mitigations

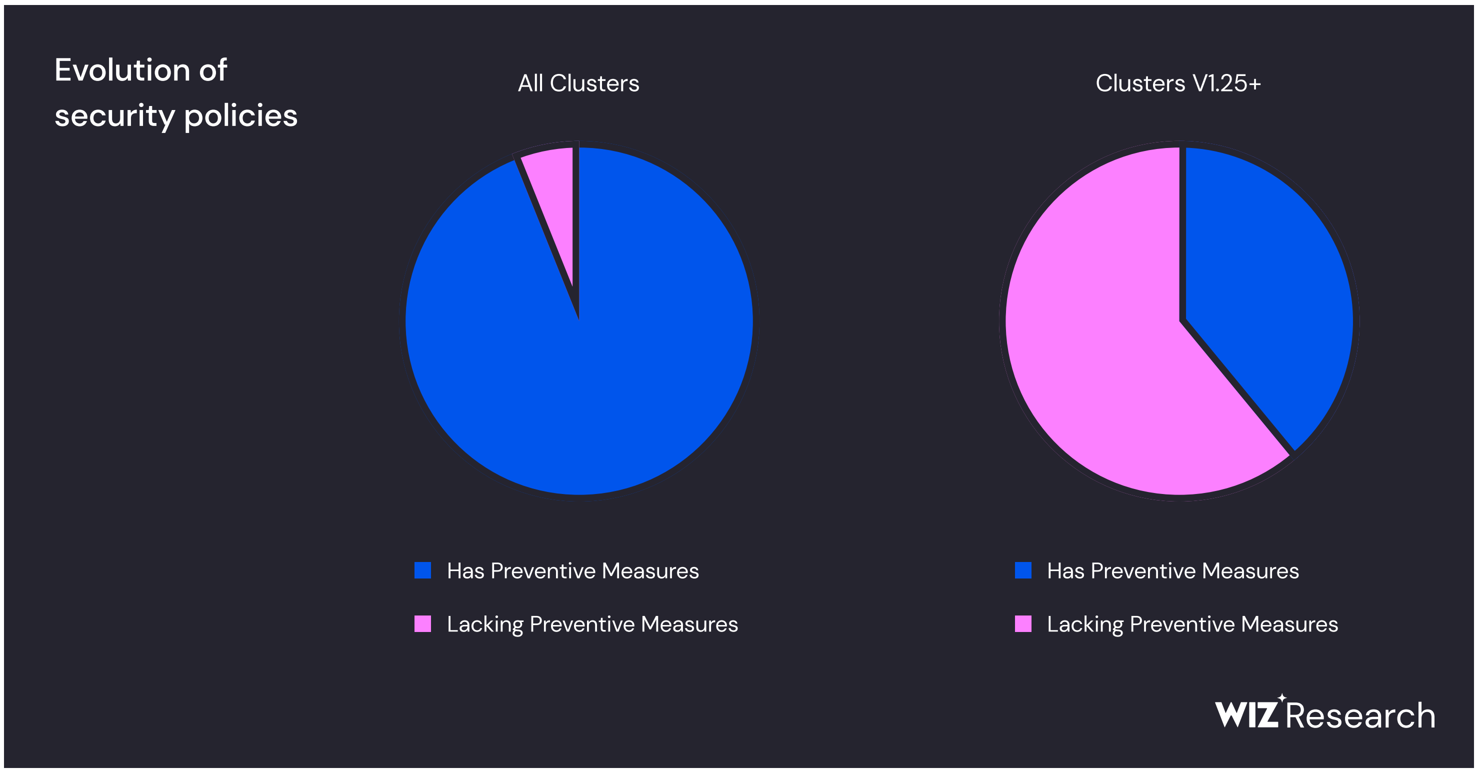
*The numbers show an increasingly concerning trend of weak security control adoption. This might be expected with such a complex feature as User Namespaces, but PSS has been around since v1.21. We observed two alarming trends: (1) clusters migrating from PSP without adopting PSA or 3rd-party admission controllers, and (2) underutilization of the PSS Restricted mode.*

For safeguarding Kubernetes clusters, a range of tools is available. Among these, the first and most straightforward was called PodSecurityPolicy (PSP), a Kubernetes feature engineered to enforce security policies on newly created Pods within the cluster. PSP enabled the definition and oversight of the security context for new pods. With the arrival of Kubernetes version 1.21, PodSecurityStandards (PSS) was created, offering a less-flexible but simple-to-use system of isolation ranks for pods on a namespace level. To enforce PSS, Pod Security admission controller (PSA) was introduced in parallel. Additionally, external admission controllers like Kyverno and Gatekeeper offer further safeguards for Kubernetes clusters. These intervene and process requests directed at the Kubernetes API server before they are persisted into the cluster's control plane, encompassing operations such as the creation, modification, or deletion of resources. As of Kubernetes version 1.25, PSP is no longer available. We talked extensively about migration strategies from PSP to PSS in this blog post.

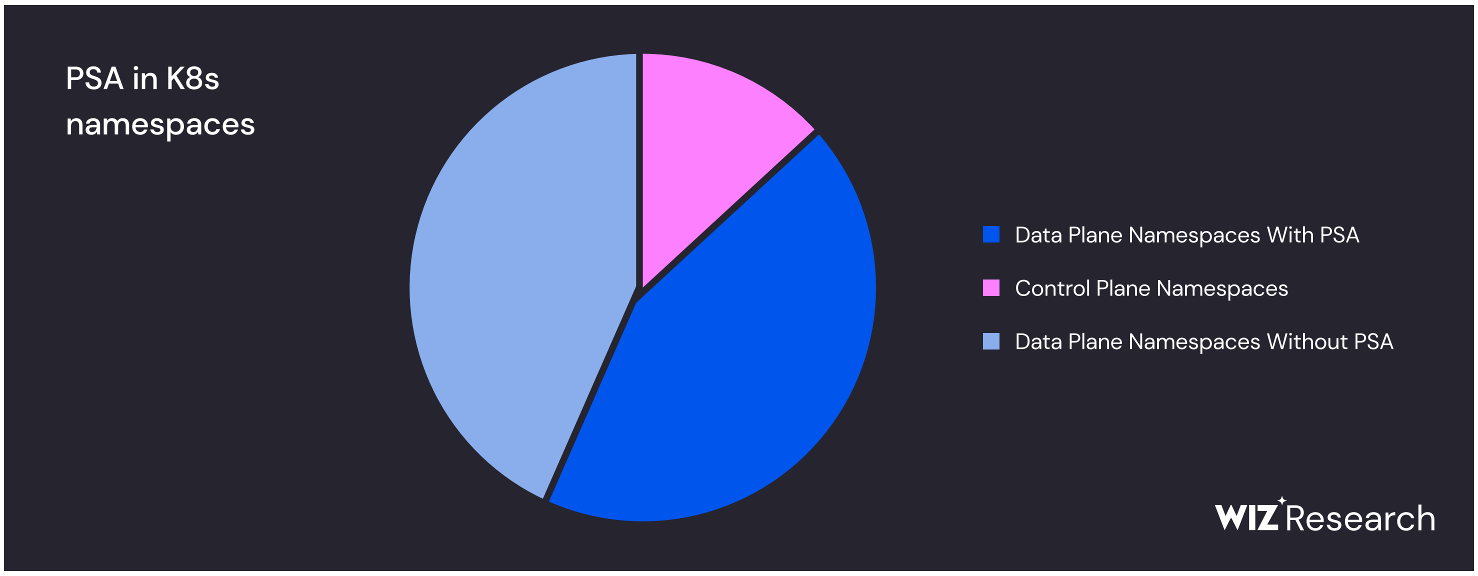
Based on our findings, we observe an encouraging trend in Kubernetes cluster security, revealing that merely 6% of clusters do not utilize PSP, external admission controller, or have at least one namespace without PSS enforced. This outcome underscores a commendably high level of protection across most clusters.

However, upon closer examination of clusters running version 1.25 and above — wherein PodSecurityPolicy (PSP) is no longer available — a different picture emerges. Within this subgroup, only 39% of clusters utilize a third-party admission controller or built-in Pod Security admission controllers in all data plane namespaces. This statistic underlines the ongoing challenge in achieving widespread PSS adoption, as a significant portion of clusters remain potentially vulnerable following the deprecation of PSP.

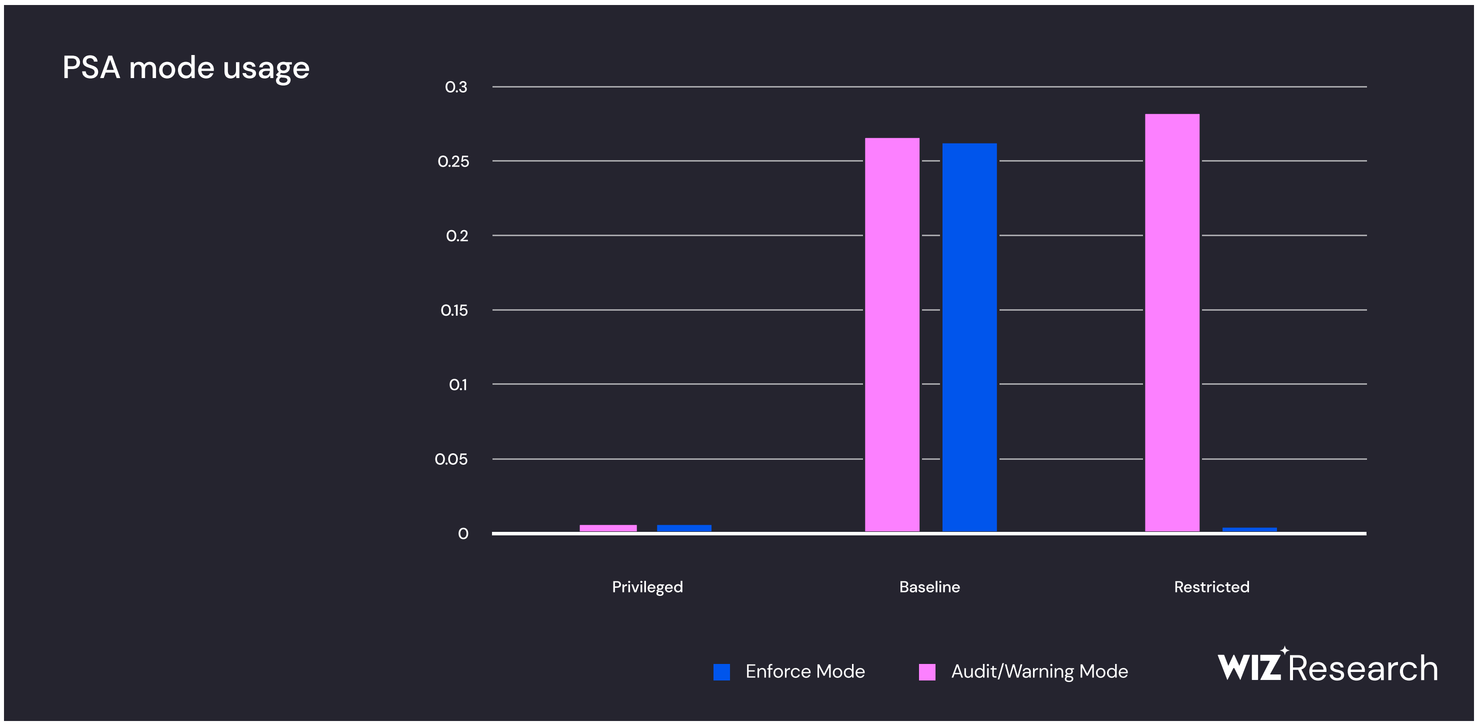
Notably, the fact that EKS does not prevent the version migration from v1.24 to v1.25 even if the customer “loses” PSP in the process can partially explain this trend.


Evolution of security policies

To gain more insight into the PSA usage, we zoomed in to a namespace level. We counted the non-control-plane namespaces regardless of the cluster version, and observed the usage of the labels that control the PSA. We learned that only about half of the data plane namespaces have PSA labels of any PSS level. This is expected because we included namespaces belonging to the older cluster versions as well.


PSA in K8s namespaces

However, when zooming in to namespaces with PSP/PSA and slicing the numbers by PSS level, we can see: (1) the obvious preference for Baseline over other levels, and (2) a disturbing drop in enforce mode applications in the Restricted level. A mere 0.13% of namespaces enforce Restricted level, which represents a x100 drop compared to Audit / Warning mode of the same level. This implies that users generally find Restricted mode to be impractical for use with production workloads.



Another recent security feature introduced in v1.25 is User Namespaces. This feature is touted as an additional isolation layer that improves host security and prevents many known container escape scenarios. We performed a deep dive into this feature in one of our blog posts and highlighted various limitations. It appears that these limitations, coupled with the lack of implementation on a container runtime side, contribute to a shocking **ZERO** pod instances using this feature in the field.

## About Wiz

Wiz secures everything organizations build and run in the cloud. Founded in 2020, Wiz is the fastest-growing software company in the world, scaling from $1M to $100M ARR in 18 months. Wiz enables hundreds of organizations worldwide, including 35 percent of the Fortune 100, to rapidly identify and remove critical risks in cloud environments. Its customers include Salesforce, Slack, Mars, BMW, Avery Dennison, Priceline, Cushman & Wakefield, DocuSign, Plaid, and Agoda, among others. Wiz is backed by Sequoia, Index Ventures, Insight Partners, Salesforce, Blackstone, Advent, Greenoaks, Lightspeed and Aglaé.

*Visit https://www.wiz.io/ for more information.*