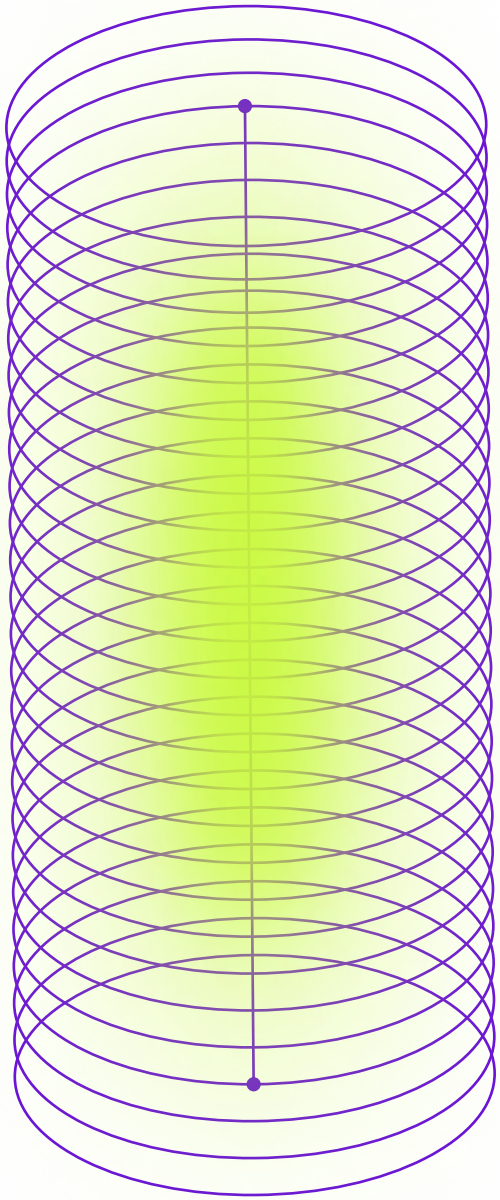
Guide

Mastering Kubernetes Security Posture Management: A Step-by-Step Guide

uptycs

Table of Contents

Kubernetes Security Posture Management: A Key New Paradigm



Kubernetes is central to modern cloud-native application development, providing the flexibility to scale and manage workloads efficiently. Yet, its complexity introduces significant security challenges, including misconfigurations, container vulnerabilities, runtime threats, and gaps in access controls. Without a proactive approach to Kubernetes security, organizations risk exposing sensitive data and compromising critical systems.

This guide offers a comprehensive approach to Kubernetes Security Posture Management (KSPM), breaking down actionable strategies for securing your clusters. From NSA hardening and container vulnerability management to runtime security and advanced access controls, you'll learn how to strengthen your defenses, achieve compliance, and operationalize security across your Kubernetes environments. Let's dive in.

01 Mastering Kubernetes Security #1: NSA Hardening for KSPM

Organizations across the globe are embracing Kubernetes security posture management and harnessing its power to deliver cloud-native applications on top cloud providers like Amazon Web Services (AWS), Azure, and Google Cloud Platform. However, Kubernetes' default security limitations mean safeguarding your cluster and data should be a top priority.

Kubernetes security posture management can be a challenge, particularly since clusters are a common source of misconfigurations in the cloud, can be shared between multiple teams, and the learning curve for Kubernetes is steep. But there is help available. Last August, the National Security Agency (NSA) and the Cybersecurity and Infrastructure Security Agency (CISA) released an updated version of the Kubernetes Hardening Guide.

The guide helps organizations solve those key misconfigurations and provides guidance to enterprises on solving key misconfigurations such as:

- **Access controls**. The most secure access follows the zero trust model for Kubernetes, i.e., only give what is absolutely necessary. However, in Kubernetes, many users can end up getting cluster role binding privileges, which gives them access to the entire cluster when they don't necessarily need or shouldn't have that - especially in shared multi-tenant cluster use cases.

- **Network Security.** Since all pods can talk to each other by default, this increases the lateral attack surface inside the cluster. The NSA guide gives great insight into how to solve this using Kubernetes network policies.

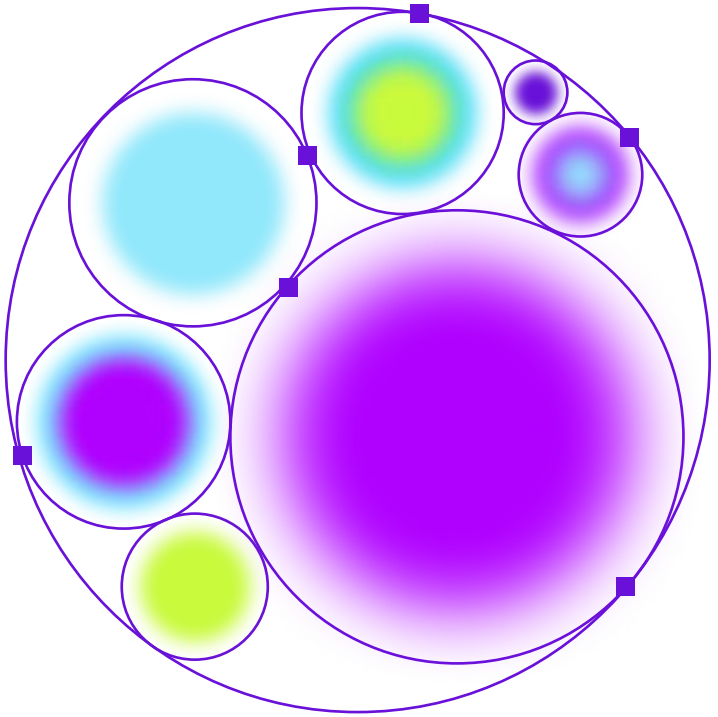
- **Container runtime issues.** Examples of this include overprivileged containers and containers with mutable file systems, which can lead to vulnerabilities and can lead to breakout.

While the insights the guide provides are a fantastic first step, there are some key challenges we saw in the market when it came to using the recommendations:

- **Operationalizing the Insights and Misconfiguration Failures:** While the guide was a great start, it was hard for customers to keep up with an entire single PDF guide and operationalize it as part of their day to day processes without codifying the rules. Non-compliant infrastructure should be visible in a single pane of glass, and any failures should be easily assignable to DevOps and platform teams responsible for the infrastructure.

- **Enabling continuous Kubernetes security posture management checks on existing and new infrastructure:** Every time a new cluster is deployed or a new namespace is provisioned, the same NSA checks should be done on new infrastructure without manual intervention. Moreover, without a system in place, it is hard for customers to do daily checks, leading them to do it only on a quarterly basis, which can lead to significant blind spots, especially since assets are ephemeral. For example, a container can be spun up and down after doing something non-compliant in minutes. Therefore, continuous checks and alerting are key.

- **Remediation info and real-time evidence:** Sometimes, while it can be great to have the information as a CISO, in order to make sure your dev and platform teams understand the risk of the issue, real-time evidence is needed. If remediation steps are provided, that leads to less friction in fixing the issue as the devs have a path forward on what to do.



While these three challenges can seem daunting, the Uptycs platform streamlines visibility into key compliance risks and provides real-time evidence and remediation steps that are continuous, actionable, and insightful. Let's see an example of this in action:

Step 1: Assessing the state of the system

When you land on the compliance dashboard, you get a single pane of glass view into the different standards being run across your cluster fleet. Any cluster you enroll, whether EKS, GKE, AKS, or even ones you manage yourself, can undergo the same compliance scanning process. Compliance scanning is done every 6 hours and is configurable. This gives the user a good understanding that their assets are being continuously monitored for compliance issues and an overall sense of the state of their system at any present moment.

In the example below, we see that roughly 40% of our assets are compliant when it comes to NSA Hardening, and 60% are non-compliant. We can see that 63 rules passed while 101 failed, and below, we get a breakdown across the different NSA Hardening categories of our pass/fail rate.

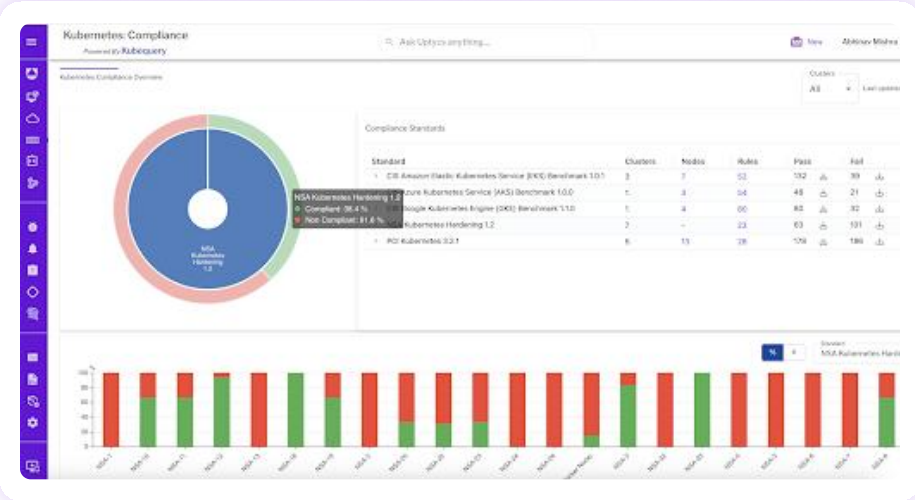


Figure 1 - Compliance overview

Next, let's look at what actual rules are failing so we can appropriately triage.

Step 2: Triaging through the rules

Next, we can click the list of rules and get an overview across different categories of what is passed, what is failing, and how many assets for the given rule are passing/failing. The good thing is that just like the NSA Hardening Guide, Uptycs also logically organizes it into the relevant sections so you can focus on a specific security task that is most relevant to you instead of traversing through tables of 100s of rules.

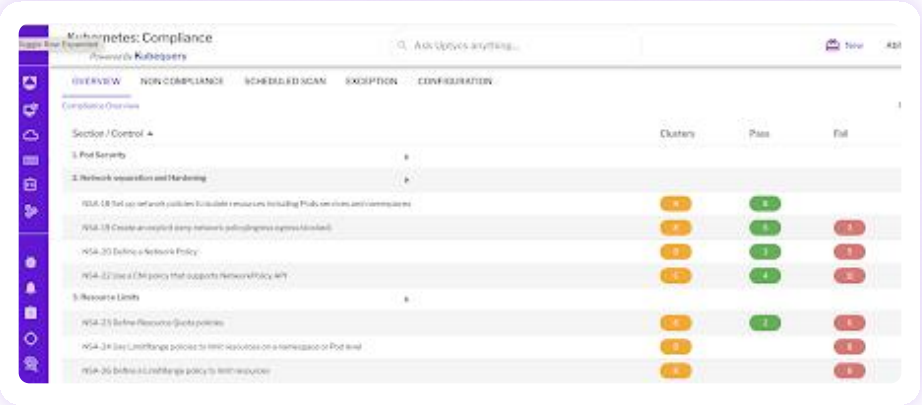


Figure 2 - Rules listing

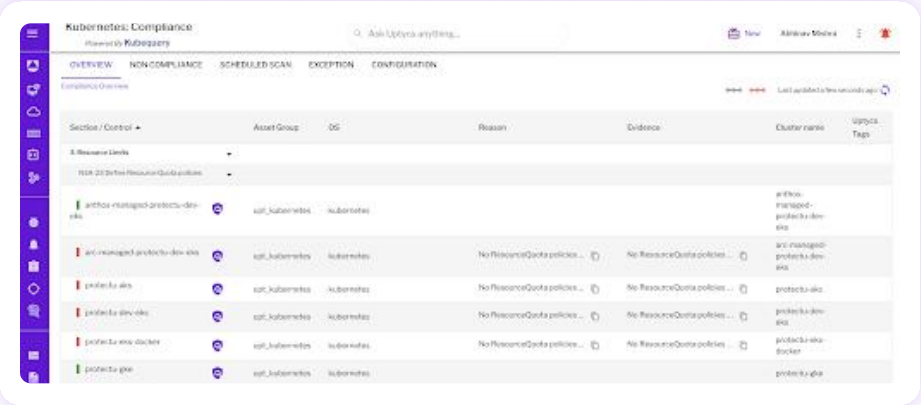


Figure 3 - Cluster listing

Clicking a specific rule, for example, NSA-23 - Defining Resource Quota Policies, gives us the clusters that are passing and failing this rule. In this case, most of my clusters do not have resource quota policies defined for their namespaces. This can be problematic, especially in shared cluster environments where one user assigned to a namespace can over-inflate their namespace and limit other applications and pods from getting deployed. This can be especially problematic if you have important software add-ons for security, observability, cost metrics, and more that constantly need to be running on the cluster.

Suppose you assign this to a DevOps or platform engineer to take a closer look at. Let's see how we can use the evidence-based view to get more details.

Step 3: Generating the evidence and remediation to take action

Many times, as a security ops engineer, you may get pushback asking why this check is so important. By clicking the purple magnifying glass, you get an evidence-based view that gives the real source of why this issue is so important, the evidence behind it, and the remediation steps. For example, for the no resource policies defined issue, you can see exact remediation steps of how one can define a ResourceQuota with CPU and memory limits for a given namespace.

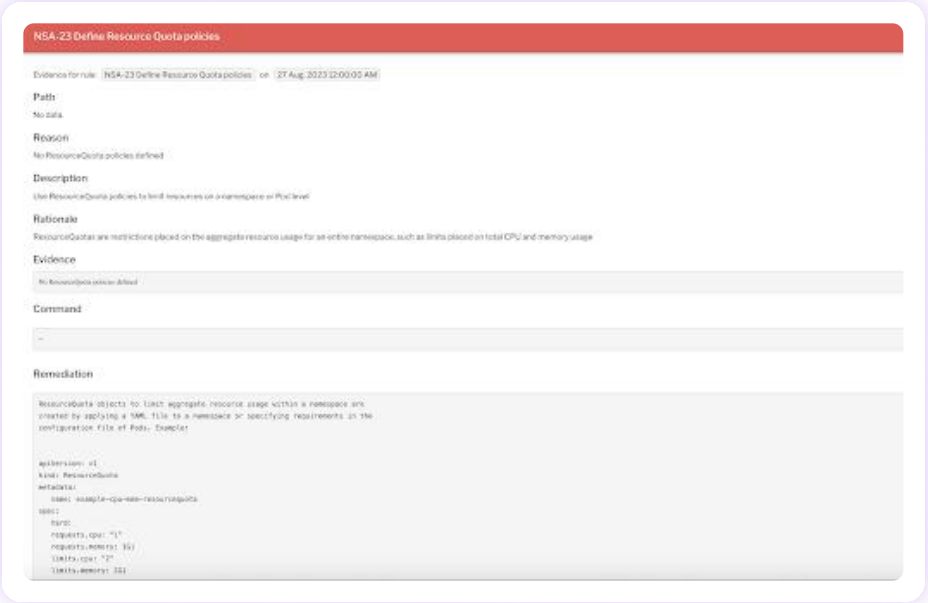


Figure 4 - Resource quota evidence

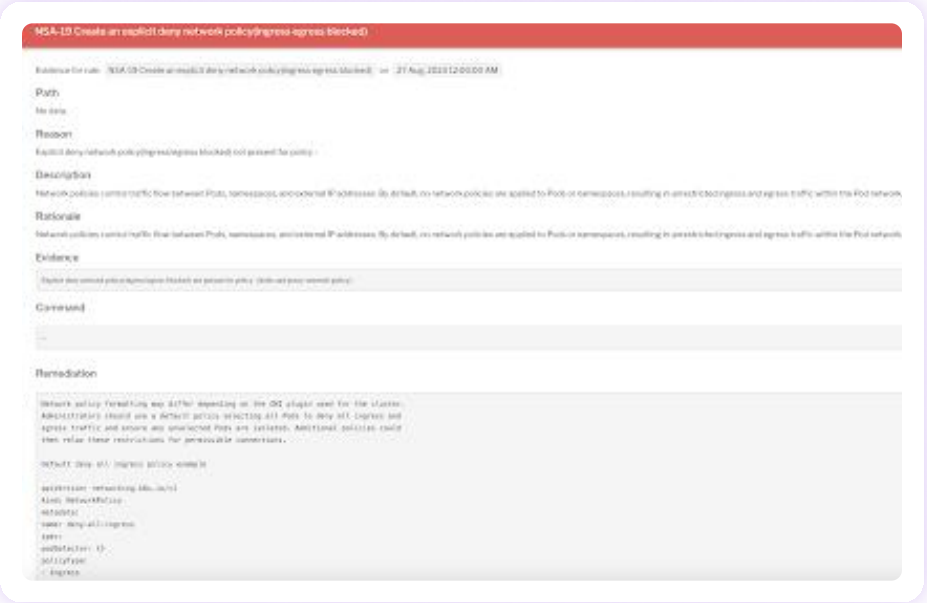


Figure 5 - Network policy evidence

Another example below shows the importance of implementing a default network policy to prevent lateral movement inside the cluster. This is especially important because, by default, every pod can talk to one another inside a cluster, increasing your lateral surface attack in case one pod with vulnerabilities is compromised.

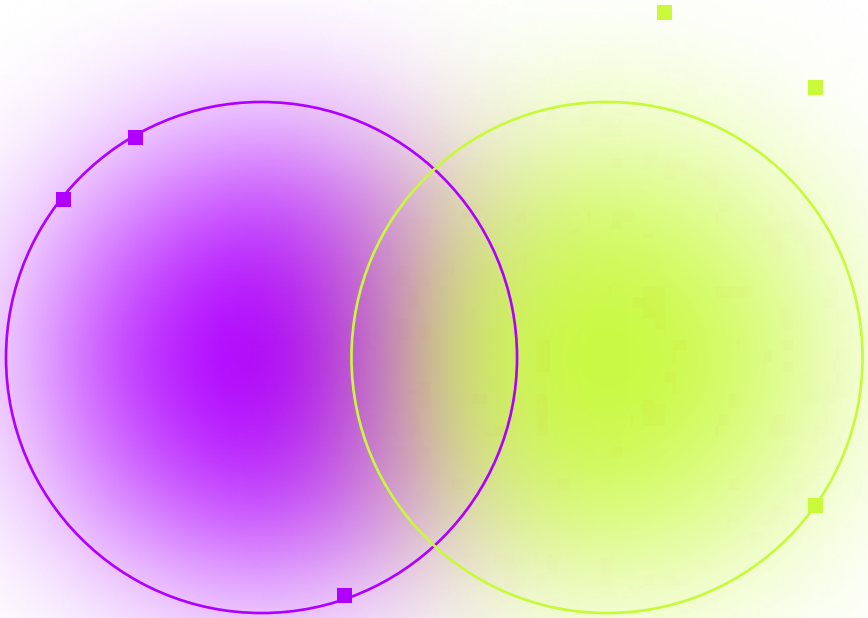
In Summary

Uptycs provides a 3-step process of visualizing, triaging, and generating evidence to remediate that allows CISOs and security admins to continuously monitor Kubernetes security posture management. This process applies to NSA Hardening and any standard such as CIS Kubernetes Benchmarks out of the box.

02 Mastering Kubernetes Security #2: Container Vulnerability Management

When you run cloud-native workloads, vulnerabilities are always found, whether a certain vulnerable package or a known exploit. You could have thousands or even hundreds of thousands of vulnerabilities at any time. Multiply that by the number of container assets and images, and you have a big challenge of figuring out where to start.

Let's dive into vulnerability management and how to implement shift left controls to prevent any vulnerabilities, malware, or secrets from entering your SDLC and supply chain.



Top Kubernetes and container vulnerability management challenges

Here are some of the challenges we see when it comes to vulnerability management for Kubernetes and containers today:

- **Vulnerability prioritization and key performance indicator (KPI) based measurement:** With all the noise and scale of ephemeral workloads such as containers and pods, SecOps admins many times have to go through a lot of noise to get at the heart of the issues, such as which vulnerabilities are most critical or which ones need the most fixing based on their specific environments. Moreover, a lack of KPIs means that vulnerability management solutions today are good with information but not good with providing actionable insights to operationalize vulnerability fixing at scale.

- **Insecure/fragmented security across the software development life cycle (SDLC):** Without a secure DevOps pipeline, customers must be more reactive in remediating vulnerabilities. No images with known vulnerabilities leaking credentials or containing malware

should ever be promoted into a position where they could easily or accidentally be deployed to production systems. In addition, SecOps admins also have to use disparate tooling to catch vulnerabilities in their CI versus their registry, which can lead to blind spots and more time spent evaluating results versus taking action!

- **Lack of flexible controls to fit your developer needs:** While critical vulnerabilities may be important to be aware of, there may not be a fix available. It's also beneficial to differentiate between dev-test and production environments. While always enforcing rules in production, consider only auditing in dev-test, ensuring CI image builds aren't halted, and keeping developers unhindered.

Let's see how Uptycs unified CNAPP and XDR security platform enables you to solve these challenges while providing a single solution for all your Kubernetes/containers vulnerability management from code to cloud.

Step 1: Vulnerability prioritization and KPI measurement

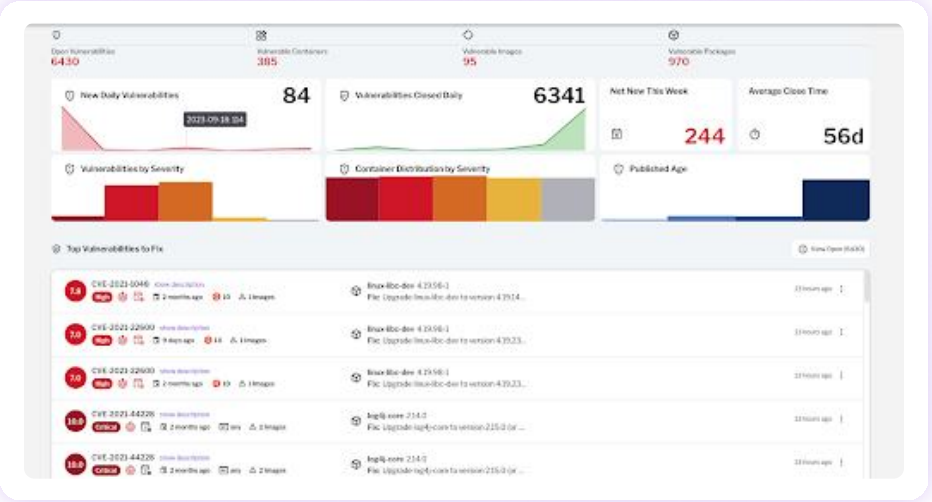


Figure 1 – Container vulnerabilities dashboard in the Uptycs platform

The Uptycs platform gives you a single place to start for anything and everything on your Kubernetes and containers vulnerability management! Specifically, the dashboard empowers SecOps admins to do the following:

- Prioritize what is important: Uptycs does intelligent prioritization for you by leveraging key, actionable metrics such as:
  - Whether a fix is available or not
  - Whether the vulnerability is exploitable or not

- CVSS Score
- Number of assets affected

By starting here you can instantly cut the noise down and start with what is most important!

- Operationalize daily/weekly triages: Through metrics such as what are your new daily vulnerabilities or net new this week - SecOps admins now have an easy way to see what is most relevant now and assign a JIRA ticket or add to an exception list with one single click.

- Continuously measure and improve: Using metrics such as average close time, CISOs and SecOps admins can continuously track and measure their overall velocity when fixing vulnerabilities and improving continuously. The Uptycs platform gives you a resolved vulnerability view to measure and audit your vulnerability fixes, including specific package updates.

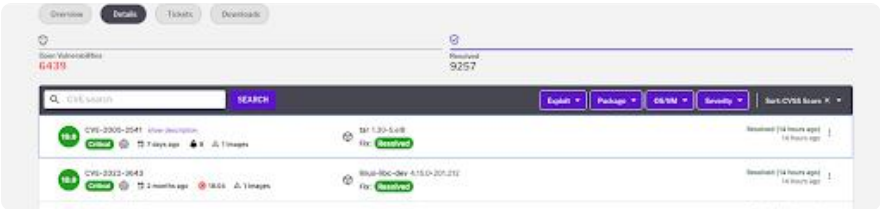


Figure 2 – Resolved vulnerabilities shown in the Uptycs platform

Step 2: Implementing shift left controls across your SDLC

Next, in order to be more proactive in fixing your vulnerabilities earlier in the SDLC, Uptycs offers CI and Registry scanning capabilities that are extremely easy to onboard, allowing you to secure your innovation pipeline from code to cloud.

Figure 3 - Secure your innovation pipeline from code to cloud

Capability	Supported Platforms
CI Scanning	Jenkins, GitHub Actions, GitLab, AWS Codebuild, Travis CI
Registry Scanning	JFrog Artifactory, Private Docker, Amazon ECR, Azure ACR, Google GCR

What sets Uptycs apart?

We can uniquely scan images across CI and Registry for vulnerabilities, malware, and secrets - ensuring your code is always at its safest.

Ready to elevate your Kubernetes and container security? Explore how Uptycs can empower your strategy at Uptycs Containers & Kubernetes.

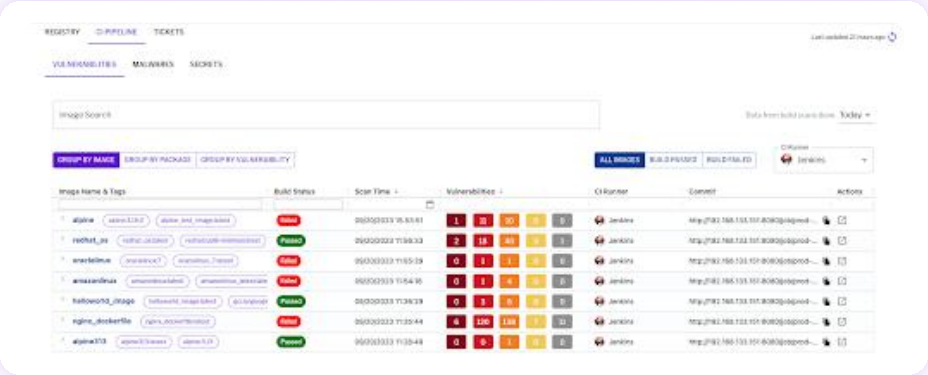


Figure 4 – CI Image scanning in the Uptycs platform

When you start with CI Scanning, the Uptycs platform shows you the security results and passed and failed image builds so that you can correlate your security findings to your SDLC.

You also get the same view with registry scanning, allowing admins to filter by a given CVE vulnerability and check on what parts of their SDLC, CI Registry, and Runtime that vulnerability is present.

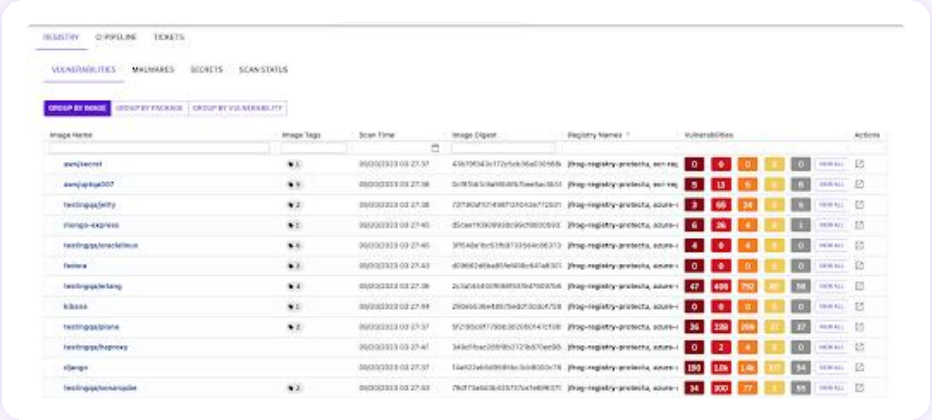


Figure 5 – Registry scanning in the Uptycs platform

As you start to prioritize fixes of images across your SDLC or look at your most vulnerable images, you get a single pane of glass view of your image security across the SDLC to answer key questions such as:

- Did the developer image go through the security controls across the SDLC? If not, what was missed?
- Did the image get deployed from a trusted registry? Images should only be pulled from trusted registries that are ideally being scanned for vulnerabilities.
- Did the developer bypass/not use the enterprise registry and instead deploy from registries such as DockerHub?

These insights can be key to ensuring your development teams and business units follow the right process while ensuring you have end-to-end security controls across your developer pipeline.

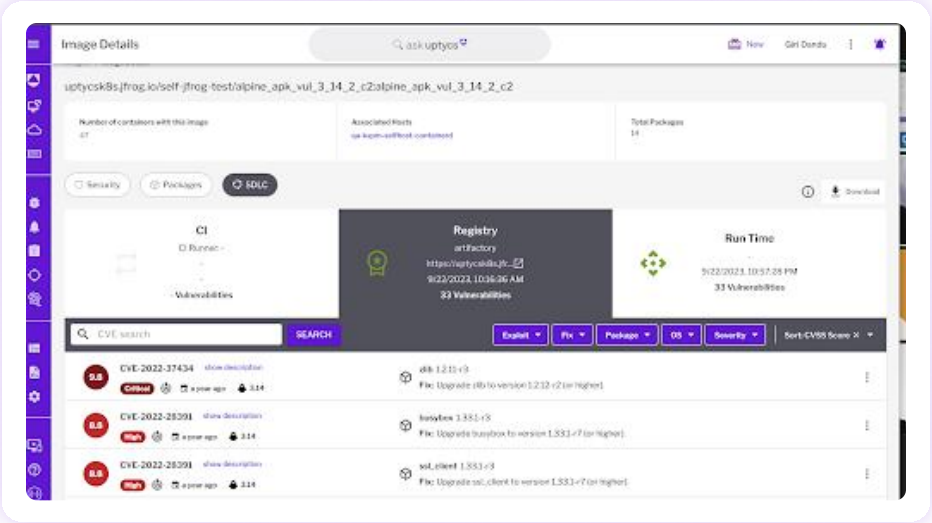


Figure 6 – Image traceability through the SDLC

Step 3: Enabling developer-friendly guardrails

While you may want to enable guardrails in runtime, a delicate balance must be achieved in the SDLC to ensure you have security controls in place while not hindering developer velocity.

In order to facilitate this, Uptycs offers key capabilities as part of the CI Scanning capabilities, such as the following:

- **Grace Periods:** Users may configure a vulnerability "Grace Period," which allows a vulnerability, once first detected, to remain within an image for a set amount of time. If the vulnerability remains within an image once the Grace Period has elapsed, then builds will fail until the vulnerability is no longer detected. This grants developers time to remediate the newly detected vulnerabilities without immediately interrupting their task when the vulnerability was first discovered.

- **Policy Configurations:** What if you wanted to ignore vulnerabilities with no fixes as part of your CI Scanning? Or only fail based on a set of critical CVEs or even ignore certain vulnerabilities? As part of your Uptycs CI Scanning configuration, you can enable certain policy configurations and flags to tailor your vulnerability management to your business needs.

  #Configure vulnerability scanning.

  Vulnerabilities:
  fatal_cvss_score: 7 # Configure the maximum allowable CVSS score.

  ignore_no_fix: true # Ignore any detected vulnerabilities for which there are no known fixes.

- **Image Admission Controller:** With the image admission controller (Fig. 6), you can implement secure guardrails by blocking images coming from registries scanning in Uptycs and with critical or high vulnerabilities from being used in runtime. This gives SecOps admins confidence that critical or high vulnerabilities will never make it into their container and Kubernetes runtime environments while allowing developers the necessary time to fix the image vulnerabilities earlier in the SDLC.

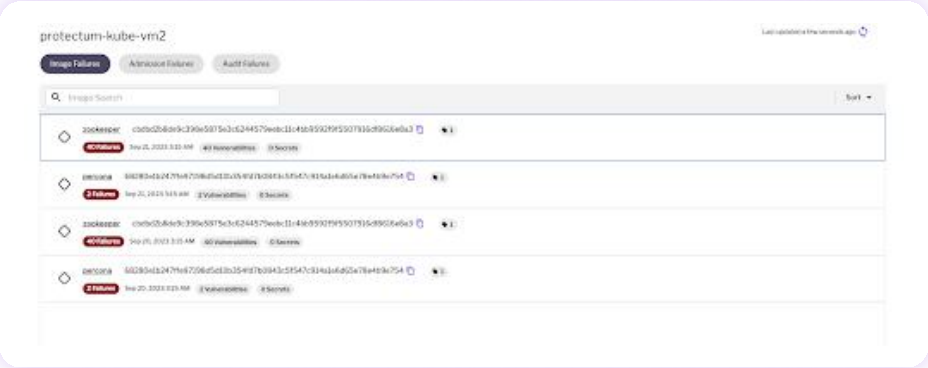
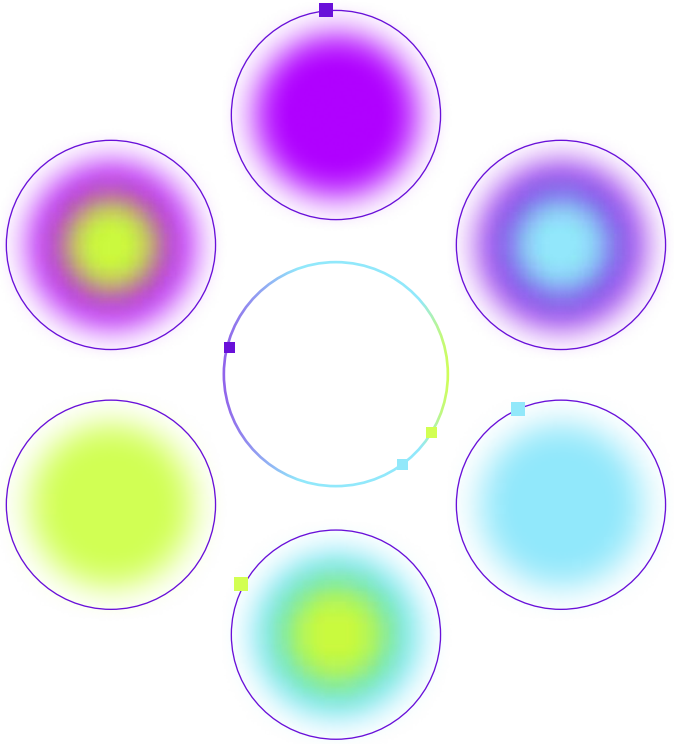


Figure 7 – Image admission controller in Uptycs platform

In Summary

Navigating vulnerabilities in Kubernetes and container environments is complex. As these vulnerabilities mount, challenges arise in prioritization, securing the SDLC, and ensuring developer flexibility. Uptycs unified CNAPP and XDR is designed to meet these challenges head-on. By offering intelligent vulnerability prioritization, seamless integration with the SDLC through CI and Registry scanning, and developer-centric guardrails, Uptycs ensures businesses can effectively manage risks while fostering innovation.



03 Mastering Kubernetes Security #3: Runtime Admission Controls

Kubernetes is quickly becoming one of the most popular platforms for developing and deploying applications. However, simplifying application deployment can open up numerous security risks unless you take protective measures to secure your Kubernetes environment.

Let's focus on Runtime Admission Controls (RACs). RACs add a layer of protection that ensures compliance and verification after the admission controller has approved an object. By implementing appropriate rules and processes with runtime admission controls, your organization can ensure protection from any malicious or improper code execution.

Benefits of Admission Control

SecOps Teams want to ensure that insecure resources, be it a pod, container, or even an ingress controller, are deployed with secure defaults. Insecure and non-compliant resources or resources with insecure defaults should be blocked from deployment. At the same time, developers want to move fast and not be overly restricted when it comes to deploying simple applications, especially in dev-test environments.

The concept of admission control was created to enforce runtime policies. Admission controllers in a cluster provide the ability to set policies that can enable secure guardrails for deployments of Kubernetes objects, be it a cluster role binding, workload, namespace, and more. At the time of resource deployment, the Kubernetes API intercepts the request and validates it against a set of policies deployed on the cluster. If the policy is not passed, the object deployment will fail. These policies can be tailored to meet business needs and set secure defaults. For example, you can write policies around:

- **Privileged pods:** Kubernetes provides Pod Security Admission policies to be able to prevent insecure workloads such as privileged pods from being deployed. Privileged pods, if exploited, can lead to container breakout, which can lead to the exploitation of an entire host.

- **Namespace resource limits:** Each developer or business unit typically gets its own namespace in shared cluster setups. However, when these namespaces lack predefined resource limits, it often leads to competition

for resources, causing issues for customers. This can lead to a security issue as an attacker can deploy and inflate a namespace to a point with a large application that can kick out other mission-critical applications on a cluster. Admission policies can check for default resource limits defined on a namespace manifest and fail namespace deployments that don't have limits and/or quotas defined.

- **Insecure ingress controller:** Network security is a key challenge in Kubernetes from a dev/test point of view. To test if things actually work, developers deploy an nginx or kong ingress controller that allows any traffic specified using a wildcard hostname. This is especially seen given the number of IaC examples that exist today. However, this can lead to public internet exposure in production environments, a crucial security risk as any malicious unauthorized user can access the cluster. Admission policies can be written to check for specific hostname strings such as */wildcard and block appropriately.

Several open-source solutions are available to implement admission control, including OPA Gatekeeper (which employs Rego for policy definition) and Kyverno. Additionally, Kubernetes offers built-in features like Pod Security Admission and Pod Security Standards to address critical concerns, such as preventing privilege escalations. However, some key challenges occur today when using admission controls. Let's take a look at those challenges.

Key challenges with admission control

- **Breaking production:** While there are many ways to do admission control today, DevOps teams in many situations are hamstrung in enforcing admission control in production environments because it can break production applications if an incorrect policy prevents an object from being blocked. In those situations, developers may not be well equipped to understand the consequences of violating those policies, especially if those policies are primarily configured only by SecOps teams. Solutions, such as OPA Gatekeeper, can do dry runs or audits rather than enforcement to understand what policies would break without preventing actual deployment.

- **Lack of integration with DevSecOps tooling:** While customers may have tooling for CI or registry scanning for vulnerabilities, malware, and secrets, most admission controllers today don't look at the data that comes from the scanning results to make an informed decision on whether to allow or prevent a container image from being used for container deployment. The siloed decision-making process results in additional time spent correlating data and cautiously managing basic runtime deployments.

- **Increased cognitive load for developers:** Policy controls, much like vulnerability management, are essential. However, developers need to easily identify the most relevant policy failures. Simple developer-friendly tooling that allows engineers to triage what audit failures are most relevant to their specific namespaces and pods in a single UI is key to allowing developers to remediate issues faster.

- **Inconsistent policy enforcement across cluster fleet:** Management of policies at cluster fleet-level scale can be an extreme challenge, especially when there is a lack of visibility in which policies are applied where and duplication. Customers need single-pane-of-glass visibility to see which policies are enforced and to categorize the most important issues across different types of Kubernetes resources so the appropriate engineers get visibility into what is most relevant for their work.

- **Aligning/deciding criteria for admission:** Deciding the right balance of controls between SecOps and engineering can be a big challenge and requires consistent communication as new requirements come in. At the same time, how do you decide what to enforce, especially to strive for consistency? The indicators of compromise or default secure configurations must be broad enough to support all your different environments.

Enforcing admission controls using Uptycs

Uptycs solves the key challenges outlined above by offering native admission control for the following:

Image deployments: Uptycs can integrate directly with registry scanning to block images with critical or high vulnerabilities found in a registry scan from being used for container deployments on a cluster. This allows SecOps teams to be confident to shift left and catch security issues earlier in the software development lifecycle (SDLC) while enabling those same guardrails using valuable information from the registry scan result for runtime deployments.

In the example below, we see image failures for the Zookeeper image because 42 critical and high vulnerabilities were found.

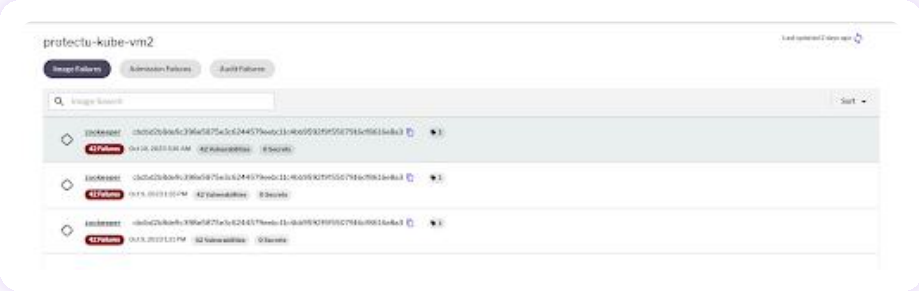


Figure 1 - Image admission failures

Runtime deployments: Uptycs integrates with OPA Gatekeeper and acts as an actual admission controller for OPA Gatekeeper so that you don't have to deploy it separately. Customers can create their own OPA Gatekeeper policies based on business needs or use some of the turnkey policies that Uptycs provides. With these policies stored in Git, customers can use GitOps to push the policies to their cluster fleet and gain single-pane-of-glass visibility at both the policy and individual asset levels.

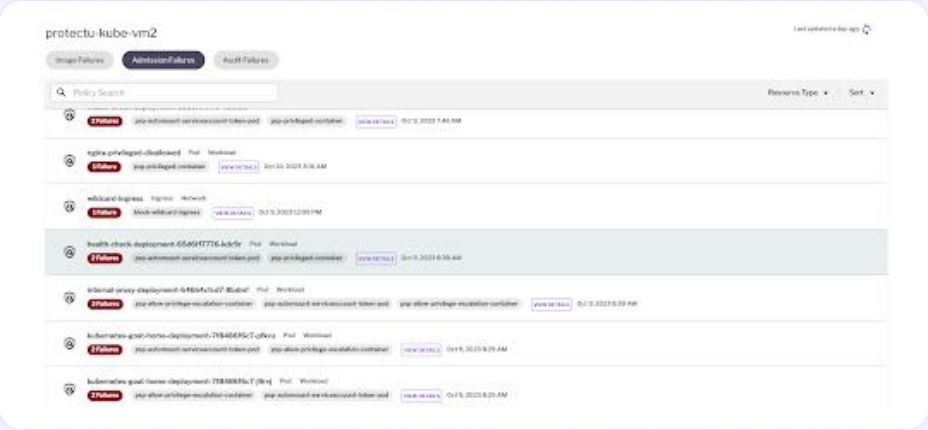


Figure 2 - Runtime Policy admission failures

As seen above, at the policy level, SecOps teams get a single-pane-of-glass view of their admission failures (i.e., which resources failed deployment and against which policies), as well as audits (i.e., which policies are being violated the most for existing resources that are already deployed).

In the example below, we see a wildcard ingress controller that has failed deployment because it is using a wildcard host, which can be used to intercept traffic from other applications and allow any malicious attacker on the Internet to enter the cluster.

Figure 3 - Example admission failure for ingress controller

Similarly, from a DevSecOps point of view, developers can look at audit failures for the specific namespaces they have access to rather than having to cut through all the noise across all the policies. This allows them to easily prioritize and reduce the cognitive load in understanding which policies they need to fix for their specific applications.

The example below shows that the nginxns namespace has one audit failure. The DevOps or developer engineer responsible for that namespace can click the audit failure to see what failed.

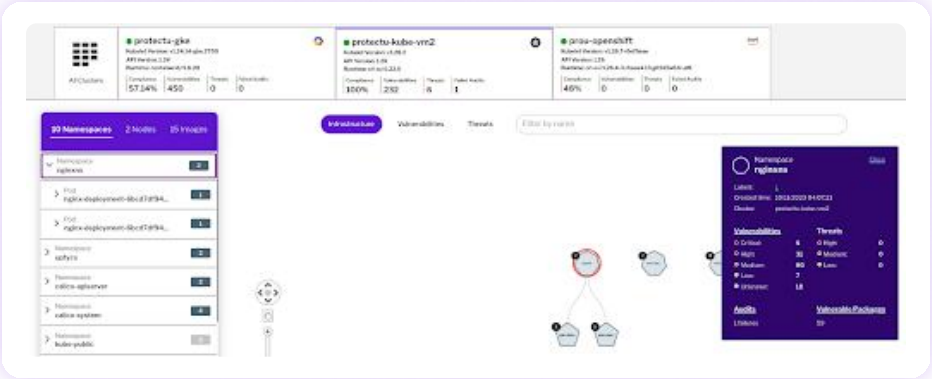


Figure 4 - Audit visibility at the namespace level

These controls can easily be enabled as part of the Uptycs protect capabilities during onboarding or via a simple helm update!

In Closing

As Kubernetes ecosystems continue to grow and evolve, ensuring secure defaults and robust hardening measures become paramount. Admission controls play a pivotal role in this security landscape, balancing developers' agility and the imperative for stringent security protocols.

While challenges exist, solutions like Uptycs offer a holistic approach to seamlessly integrate security throughout the development lifecycle. By harnessing the power of admission controls and integrating them with other security facets, organizations can fortify their Kubernetes deployments against potential misconfigurations and threats. As we journey ahead, embracing these tools and strategies will be key to building a resilient and efficient Kubernetes environment.

Stay tuned for our next blog in this series, where we'll discuss key threat indicators for Kubernetes in Uptycs based on Kubernetes GOAT.

04 Mastering Kubernetes Security #4: Authorization, Access & Secrets

The ease of managing and deploying K8S applications comes with security challenges, particularly in access control. Role-Based Access Control (RBAC) alone is not enough to secure access to your K8s, especially when managing sensitive data or "secrets."

The heart of the matter is, RBAC operates within a predefined scope of rules which, while useful, might not suffice in the face of evolving security threats and complex, dynamic Kubernetes environments. Moreover, the management of sensitive information, known as secrets, intertwines with RBAC, adding another layer of complexity to the security narrative.

Let's explore why RBAC might not be enough to solve your Kubernetes access control challenges, and what you should be doing instead.

RBAC in Kubernetes

Role-Based Access Control (RBAC) is a standard mechanism for managing who has access to the system's resources and what actions they can perform.

In Kubernetes, RBAC allows administrators to define roles with specific permissions like viewing, creating, or deleting resources, and assign these roles to users or groups.

Typical configurations involve setting up roles and role bindings that tie users/groups to roles. However, common misconfigurations can occur, such as overly permissive roles, incorrect role bindings, or stale roles that remain in the system even after they are no longer needed. These misconfigurations can open up avenues for unauthorized access or actions within the Kubernetes environment, posing significant security risks.

For instance, consider a scenario where an admin mistakenly binds a "delete" permission to a general user role, instead of a more restrictive admin role. This misconfiguration in role binding could potentially allow any general user to delete crucial resources within the Kubernetes environment, causing significant disruption or data loss.

Secrets management in Kubernetes

Secrets in Kubernetes serve as a means to store and manage sensitive information like passwords, tokens, and keys. Managing these secrets securely is paramount to ensuring the integrity and confidentiality of the system.

RBAC plays a role in secrets management by controlling who can access or manage secrets. However, common misconfigurations like overly permissive access to secrets, or improperly secured secrets (e.g., storing secrets in plain text or version control systems), can lead to exposure of sensitive data.

Furthermore, the intertwining of RBAC with secrets management can sometimes obscure the security landscape, making it challenging to ensure both robust access control and secure secrets management. As we delve deeper, we'll unravel the intricacies of this interaction and how a more holistic approach can provide a robust solution to these challenges.

Managing authorization, access, and secrets in Kubernetes

Kubernetes provides a robust platform for orchestrating containerized applications, however, managing authorization, access, and secrets within K8s poses its own set of challenges. Here are three best practices to ensure a secure K8s environment.

1. **Leverage Role-Based Access Control (RBAC)**
   Role-Based Access Control (RBAC) is a key pillar to Kubernetes security. RBAC allows you to assign roles and permissions to users, ensuring that only those with the correct access can view or modify data. By using RBAC, you can ensure that only authorized users are able to access sensitive data and resources, helping to protect your system from malicious actors.

2. **Monitor Permissions and Access**
   This will allow you to identify any potential risks in your system and take steps to secure them before they are exploited by malicious actors. Additionally, monitoring permissions on an ongoing basis will help you quickly identify any changes in user access that could indicate a potential breach or other security issue.

This is especially important when onboarding new applications and monitoring default service accounts that may have access to all permissions by default, even if assigned a specific role binding.

3. **Leverage Advanced Security Solutions**
   Advanced security solutions like Uptycs can also be used to map real-time threats to risky RBAC configurations, giving you end-to-end visibility of your Kubernetes environment and helping you remove any blind spots in your security posture. With Uptycs, you can gain insights into which users have access to what resources, as well as detect suspicious activity across all of your Kubernetes clusters in real time.

Advanced security solutions like Uptycs can also be used to map real-time threats to risky RBAC configurations, giving you end-to-end visibility of your Kubernetes environment and helping you remove any blind spots in your security posture. With Uptycs, you can gain insights into which users have access to what resources, as well as detect suspicious activity across all of your Kubernetes clusters in real time. In addition you can map real-time data plane threats with RBAC control plane misconfigurations.

Let's walk through an example:

First, in the Uptycs UI, under 'Detections', an alert highlights insecure Kubernetes kubectl access from a compromised pod. This can be risky because kubectl should typically be performed outside the cluster using a secure kubeconfig. If the kubeconfig inside the cluster is accessed it could lead to more unauthorized access to nodes, secrets, namespaces, and more.

The threat is mapped to the MITRE ATT&CK Framework with details like the default service account used, pod's namespace, and source IP. Navigating to the detection graph, we see the service account attempted a deployment.

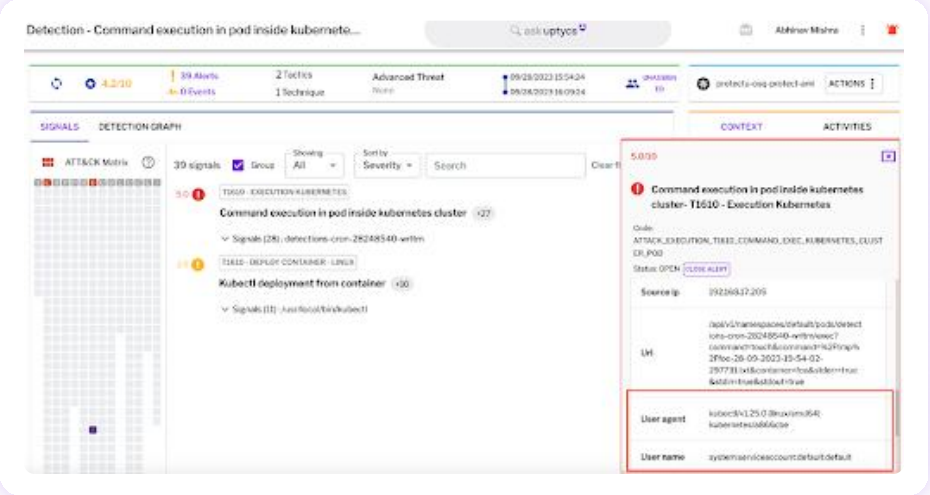


Figure 1 - Detection of insecure Kubernetes kubectl access in Uptycs

We then move to the 'RBAC Access Monitor' for this cluster, filtering by the compromised service account. It reveals excessive privileges across the cluster, including access to vulnerable pods, indicating a severe security risk. Uptycs showcases its strength in correlating real-time threats with RBAC misconfigurations, aiding in pinpointing and addressing security issues in Kubernetes.

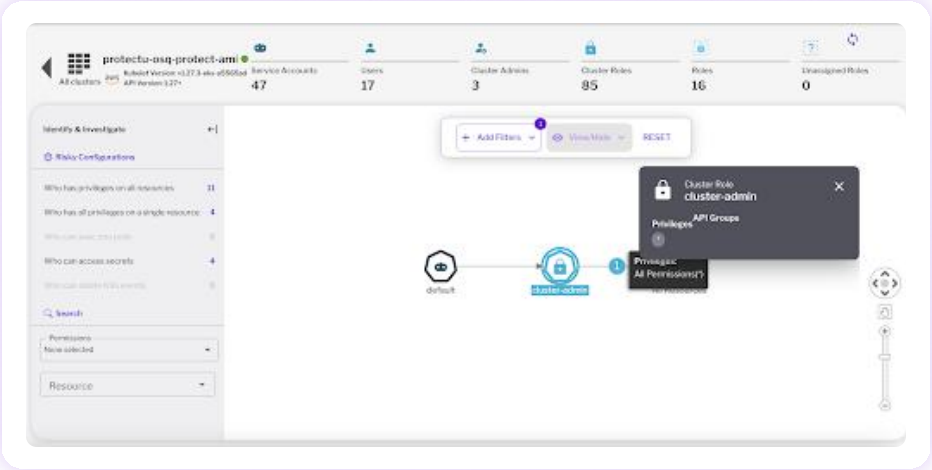
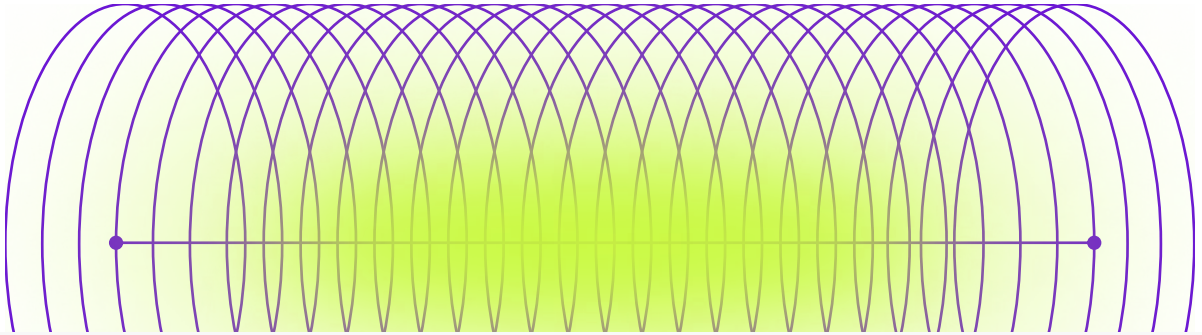


Figure 2 - Excessive privileges revealed in Uptycs' RBAC Access Monitor

In Closing

Navigating authorization, access, and secrets management in Kubernetes environments can often feel like a daunting task. RBAC provides a foundational layer of control, however, the dynamic and complex nature of K8s environments calls for a more comprehensive and proactive approach to security.

Tools like Uptycs further simplify this journey by offering a unified platform equipped with the necessary capabilities to address the challenges head-on, ensuring a secure, compliant, and efficient K8s environment. As the landscape of cybersecurity continues to evolve, embracing a holistic approach to K8s security that extends beyond RBAC, and aligning with robust security solutions like Uptycs, will prove instrumental in staying ahead of threats and ensuring a resilient K8s ecosystem.



05 Mastering Kubernetes Security #5: Incident Response with Detections

With the increasing attack surface and volume of noise generated from alerts, CISOs and SecOps are fatigued with having to figure out what is a false positive versus what is an actual threat that needs attention. Let's discuss a framework for catching different threats to secure and harden your Kubernetes clusters.

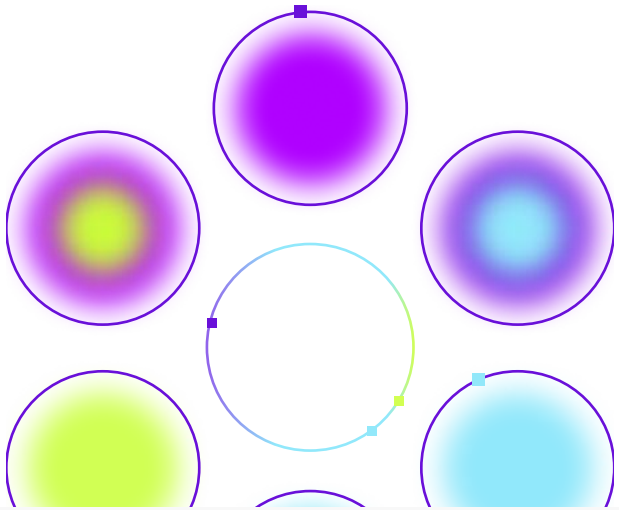
3 Steps to uncovering & addressing Kubernetes threats

Step 1: Think like a threat actor and list the most common types of attacks

While there are many different misconfigurations in Kubernetes, there are a handful of different types of attacks that are typical and can cover a lot of ground. Many frameworks such as OWASP Top 10 exist out there today that talk about the different security principles. However, one that we highly recommend looking at and have seen many of our own customers ask for is Kubernetes GOAT. What's exciting about Kubernetes GOAT is that it covers many end-to-end examples for example:

1. Privilege escalation
2. Container breakout into the host exposing cluster details
3. Port scanning
4. RBAC least privilege misconfigs (similar to our last blog post)
5. SSRF attacks

Kubernetes GOAT also provides details on how to actually replicate the attack. These insights are key to getting into the mind of an attacker and seeing what kind of commands they would perform, the starting points for most key attacks, and more.



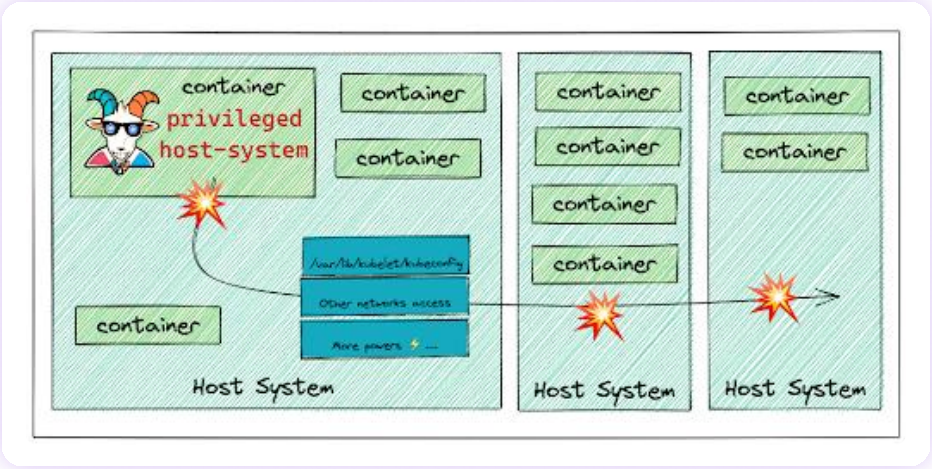


Figure 1 - Kubernetes container escape example

The Container Escape Example is a great one to look at and replicate to see how a malicious attacker can access the host/VM from a pod/container and then use that to access other cluster-level info, nodes, and even secrets. Other types of attacks can include:

- SSRF attacks
- Exposed services on node ports
- Reverse shell executions to exploit a system's vulnerabilities and attack the host
- Denial of service based on exploiting services that don't have memory or CPU limits (this can be extremely problematic in multi-tenant scenarios where stressing one namespace or pod can take down the entire cluster)

As seen in Figure 2 below, Uptycs can aggregate these kinds of attacks based on the latest threat intel so you can stay up to date and have the best possible breadth, including from frameworks such as Kubernetes GOAT and more!

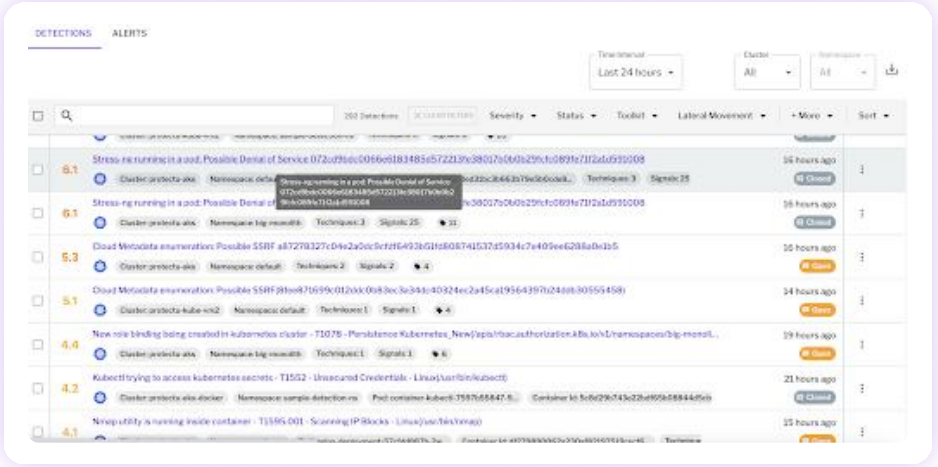


Figure 2 - Different types of Kubernetes detections

Step 2: Address all the blind spots through breadth of telemetry collection

In order to build robustness around your detections, you need to collect telemetry from some key sources to eliminate any blind spots. Let's take a look and briefly describe how Uptycs in particular tackles these:

Audit Logs: Audit logs contain key information from the Kubernetes API about any new resources created, actions performed, etc. Audit logging should always be turned on in your cluster from a compliance point of view as well as to triage certain issues. However, what audit logs can provide is a way for you to understand what policies, roles, and infrastructure entities are being created or modified. For example, it can tackle issues around:

- **RBAC - cluster role bindings:** Typically you want to limit the cluster roles and permissions inside your cluster. If cluster role bindings are unnecessarily being created it should be monitored. In addition, you want to perform a continuous audit of these types of roles and limit the permissions to only the specific actions needed.

- **Network policies:** Typically to enable a zero-trust model, you want to limit the number of possible connections through network policies. If your network policies, especially cluster-level ones that enable tenant isolation are being modified, it's important for you to audit those actions before something inadvertently takes place.

Uptycs maps collects audit log data and maps them into key sets of security events and detections for you. In fact, we have over 60 different types of detections just around audit logs that we collect as part of our Kubernetes security events package!

Runtime telemetry: In addition to audit log data on the control plane, you want to understand in real time what is happening in your data plane. This allows you to map what is happening in real time to key misconfigurations in your control plane. Some types of information you may want to collect are:

- **Process events:** If a malicious attacker enters your container or pod, you want to identify the process running to stop it, kill it, and more.

- **Files:** These fall into two key buckets.

  a. One is files that are insecure whether they be a misconfigured kubeconfig file that has overprivileged access or cryptominer malware. Understanding what files are present in your system in real-time can be key to your ability to threat hunt faster.

  b. Second is files that are not insecure but are important to your cluster operations. This could include files like secrets/sensitive data or certificates. You should always make sure these files are at the minimum encrypted and ideally not accessible from within the cluster and pod itself in case those are compromised. Typically secrets should be stored in a secrets manager such as Hashicorp Vault or AWS Secrets Manager.

- **Socket events:** If insecure network connections are made to a malicious IP, then it is important to detect those and understand where they came from. Socket events can help you identify that.

Uptycs collect this data via our runtime osquery-based sensor which is able to go in-depth and collect these different types of event data.

Once you identify the telemetry sources, correlate them using SQL or a rule engine into different types of events so that you can quickly identify and act on any data found in real time. Uptycs with its real-time security data lake empowers users by:

- **Automating checks of the most known detections:** based on event rules that map back to the real-time security data lake and data collected from a variety of sources, Uptycs eases the burden of having to build event rules by detecting and alerting users on the most common types of detections.

- **Providing a scalable security detection engine:** Uptycs can scale to millions of events and prioritize the threats that are most relevant and risky via ML and anomaly-based detections as well as correlating telemetry at scale from the different sources mentioned.

- **Empowering customizability through detections as code:** Allowing SecOps to build their own detections based on Sigma rules as well as using simple SQL queries through the single console investigate engine and security data lake.

Step 3: Think like a threat hunter

Once you have a system for identifying threats, it's important to understand where your Kubernetes threat came from and perform the right techniques. Let's take a look at an example:

Example: Identifying a malicious port scan process through YARA rule signature

One common attack is using a port scan via nmap as described here. Nmap enumerates for exposed ports and vulnerabilities associated with those open ports. In order to think like a threat actor and stop this attack, we need to think about the following:

How is the attack being masked?

- If it's an internal user, we need to know their intentions behind the execution. Are they doing it for genuine reasons? And are they doing it from a namespace that

has access to sensitive information or their own app namespace for debugging purposes?

- If it's an external user where is the malicious IP coming from and how. We need to look for compromises in the system. Do I have an exposed node port? Or a network policy/ingress controller that allows any time of traffic?

How is the attack being masked?

In Figures 3 and 4 below we see that the attacker is renaming nmap to something else to hide from defense systems. Uptycs gives SecOps teams superpowers by allowing them to analyze the process using a YARA rule signature. In this case, we see that Uptycs shows that the port scan, nmap is being masqueraded behind /qwer in the process.

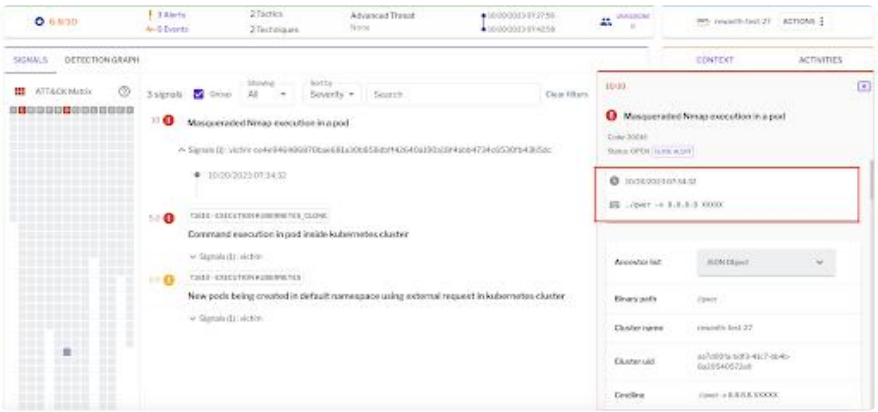


Figure 3 - Original nmap hiding behind /qwer

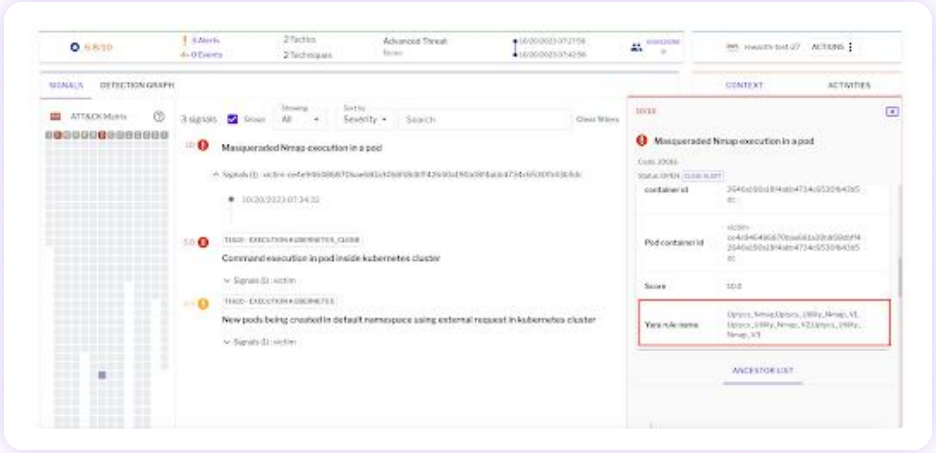


Figure 4 - YARA rule scan being performed to discover masqueraded nmap execution

From there, Uptycs of course offers container process remediation to kill malicious processes running on a container such as the masqueraded nmap. These kinds of threat-hunting capabilities and mindset are needed, however, to avoid these kinds of attacks.

06 Elevating Your Kubernetes Security Posture

From NSA hardening guidelines and container vulnerability management to runtime admission controls and effective secrets management, Kubernetes Security Posture Management (KSPM) empowers organizations to tackle the unique security challenges of cloud-native ecosystems.

Each chapter of this guide has highlighted critical security concerns and actionable solutions for bolstering your Kubernetes defenses:

Hardening with NSA guidance: Establish foundational security practices to reduce misconfigurations and strengthen access controls.

Container vulnerability management: Prioritize and address risks across the software development lifecycle (SDLC) to maintain a robust security posture.

Runtime admission controls: Implement guardrails to ensure compliance and mitigate risks in real time.

Access and secrets management: Go beyond RBAC to secure sensitive data and eliminate common configuration gaps.

Incident detection and response: Adopt a proactive threat-hunting mindset to detect, prioritize, and neutralize vulnerabilities.

As Kubernetes adoption grows, so does the complexity of securing dynamic, distributed workloads. Tools like Uptycs help simplify this journey by providing a unified platform to visualize, prioritize, and act on threats with unparalleled efficiency. With real-time monitoring, actionable insights, and automated remediation, Uptycs empowers security and DevOps teams to confidently safeguard their Kubernetes environments.

By following the strategies outlined in this guide and leveraging advanced solutions, your organization can not only protect its assets but also build a secure, compliant, and resilient Kubernetes ecosystem. Start transforming your Kubernetes security today.

uptycs

Uptycs is dedicated to leading security innovations in hybrid cloud environments, ensuring robust protection and enabling our customers to innovate safely and efficiently. Included in the 2024 CNAPP Market Guide, Uptycs provides comprehensive security solutions that bridge the gap from code to cloud. Our platform excels in Cloud Workload Protection (CWPP), Vulnerability Management, Cloud Security Posture Management (CSPM), Detection & Response, Software Pipeline Security, XDR, and Risk & Compliance. Trusted by leading enterprises like PayPal and Comcast, Uptycs transforms potential vulnerabilities into fortified security, ensuring your digital environments are safeguarded from development through runtime.

Secure Everything from Dev to Runtime

Learn more at Uptycs.com