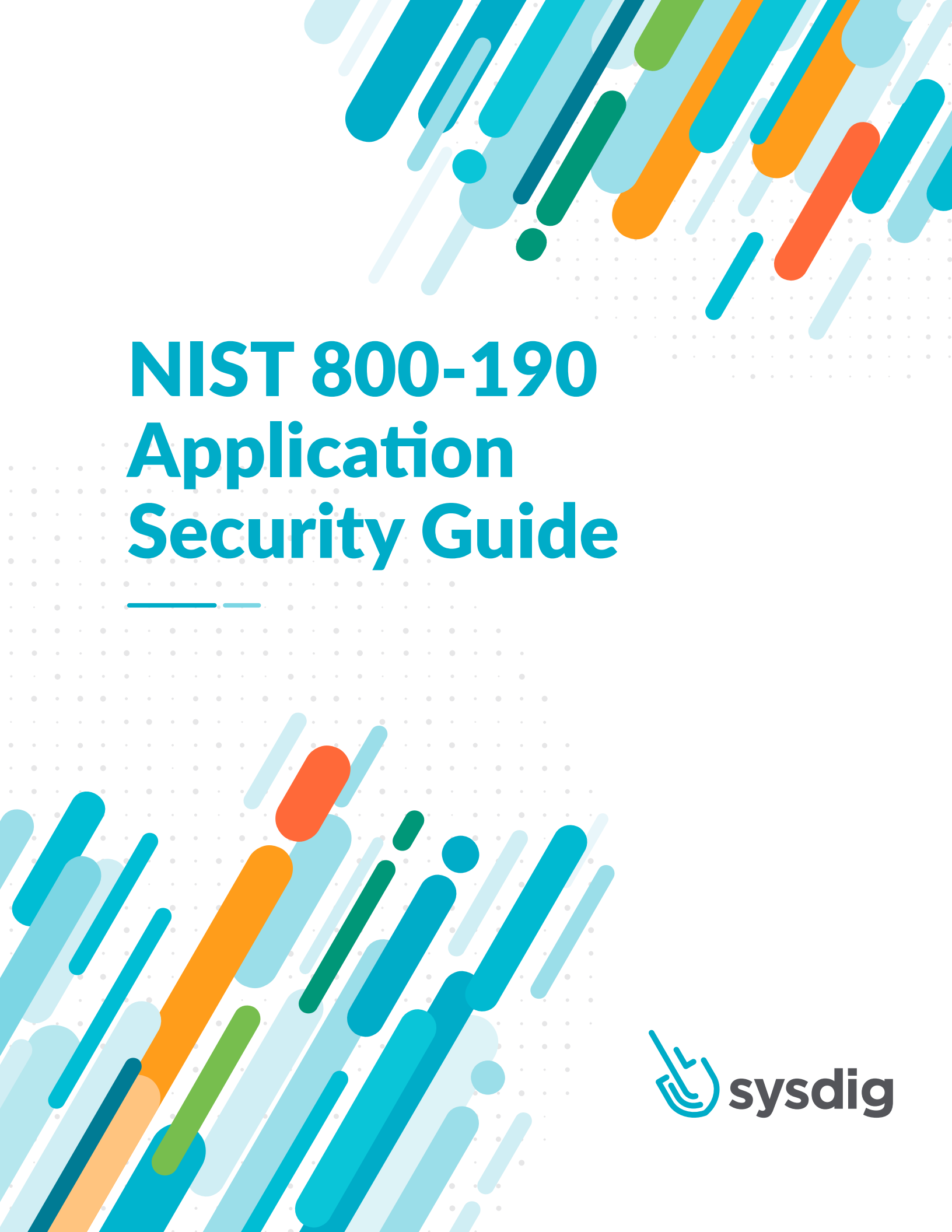
NIST 800-190 Application Security Guide



Contents

Intro to Sysdig Secure

Sysdig Secure brings together image scanning, run-time protection and forensics capabilities to identify vulnerabilities, block threats, enforce compliance and audit activity across your microservices.

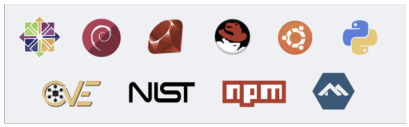
Accelerate the development of reliable, secure software

sysdig — A single platform to improve software development and reduce risk at every stage of the lifecycle

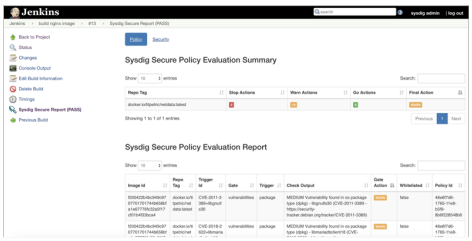
Build	Run	Respond
• Measure service perfomance	• Measure compliance	• Proactively alert on incidents
• Microservice debugging	• Monitor application performance	• Reduce MTTR with forensics
• Dectect vulnerable images	• Block zero-day threats	• Capture detailed audit records

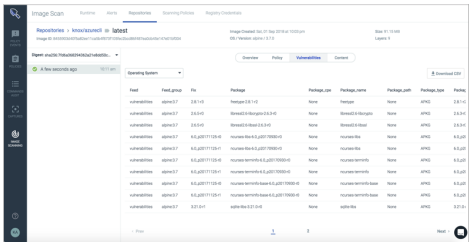
Build

- Fail fast by integrating scanning into the CI/CD process
- Scan images stored in registries for vulnerabilities & compliance
- Robust vulnerability databases
- Configure custom policies for security, reliability, and compliance
- Prevent vulnerable images from being deployed



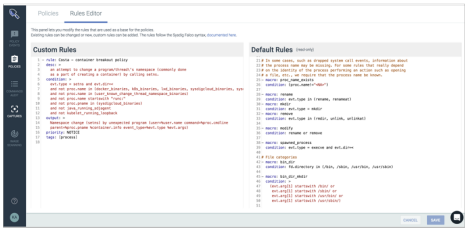
Jenkins JFrog Artifactory GitLab



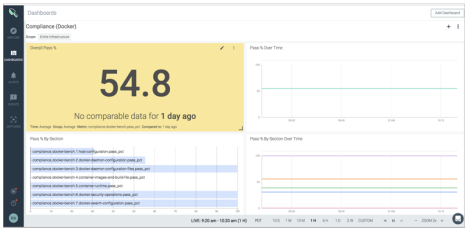


Run

- Default policies from CIS, Falco Community, & Sysdig threat research teams
- Protect hosts & containers
- Detect the launch of privileged containers, sensitive mounts, container breakouts
- Robust engine for process execution, network, file, and system call activity
- CIS Compliance Benchmarks

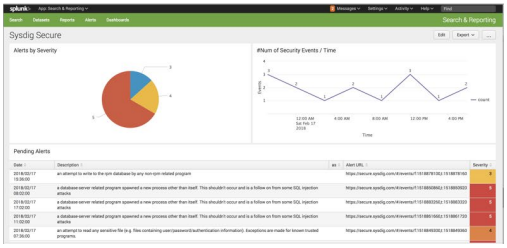


kubernetes
amazon web services
Azure Container Service
OPENSHIFT
GKE
Pivotal Container Service
MESOS



Respond

- Capture all activity before, during, and after any incident
- Automatic event enrichment with hundreds of labels & tags
- Audit user activity across containers, hosts, and services
- Robust notification channels & integrations



About NIST 800-190

The National Institute of Standards and Technology (NIST) is a physical sciences laboratory and a non-regulatory agency of the United States Department of Commerce.

In their Special Publications (SP), the organization shares technical reports, recommendations, practice guides, industry handbooks and other similar technical documents intended for external distribution.

The SP 800-190 publication (usually referred as NIST 800-190) that is the focus of this guide was published in September 2007. It describes the potential security concerns associated with the use of containers and provides recommendations for addressing these concerns.

It is a no-regulatory document, centered in container technologies, that comprises seven main sections:

- Section 1: Introduction
- Section 2: Introduces containers, including their technical capabilities, technology architectures and uses.
- Section 3: Explains the major risks for the core components of application container technologies.
- Section 4: Recommends countermeasures for the risks identified in Section 3.
- Section 5: Defines threat scenario examples for containers.
- Section 6: Presents actionable information for planning, implementing, operating and maintaining container technologies.
- Section 7: Conclusion

SP 800-190 also includes five appendices:

- Appendix A lists NIST resources for securing non-core components of container technologies.
- Appendix B lists the NIST Special Publication 800-53 security controls and NIST Cybersecurity Framework subcategories that are most pertinent to application container technologies, explaining the relevancy of each.
- Appendix C provides an acronym and abbreviation list for the document.
- Appendix D presents a glossary of selected terms from the document.
- Appendix E contains a list of references for the document.

Sections 3 and 4 have the same subsection structure. While the focus is on explaining the risks in Section 3, recommendations are presented for those risks in Section 4 .

3.1. Image Risks	4.1. Image Countermeasures
3.1.1. Vulnerabilities	4.1.1. Vulnerabilities
3.1.2. Misconfiguration	4.1.2. Misconfiguration
3.1.3. Malware	4.1.3. Malware
3.1.4. Clear-text secrets	4.1.4. Clear-text secrets
3.1.5. Untrusted images	4.1.5. Untrusted images
3.2. Registry Risks	**4.2. Registry Countermeasures**
3.2.1. Insecure connections	4.2.1. Insecure connections
3.2.2. Stale images	4.2.2. Stale images
3.2.3. Insufficient authorization restrictions	4.2.3. Insufficient authorization restrictions
3.3. Orchestrator Risks	**4.3. Orchestrator Countermeasures**
3.3.1. Full administrative access	4.3.1. Full administrative access
3.3.2. Unauthorized access	4.3.2. Unauthorized access
3.3.3. Inter-container network traffic	4.3.3. Inter-container network traffic
3.3.4. Mixed workload sensitivity levels	4.3.4. Mixed workload sensitivity levels
3.3.5. Node trust	4.3.5. Node trust
3.4. Container Risks	**4.4. Container Countermeasures**
3.4.1. Vulnerabilities in the runtime	4.4.1. Vulnerabilities in the runtime
3.4.2. Unbounded network access from containers	4.4.2. Unbounded network access from containers
3.4.3. Insecure configurations	4.4.3. Insecure configurations
3.4.4. App vulnerabilities	4.4.4. App vulnerabilities
3.4.5. Rogue containers	4.4.5. Rogue containers
3.5. Host Risks	**4.5. Host Countermeasures**
3.5.1. Large attack surface	4.5.1. Large attack surface
3.5.2. Shared kernel	4.5.2. Shared kernel
3.5.3. Host component vulnerabilities	4.5.3. Host component vulnerabilities
3.5.4. Improper user access rights	4.5.4. Improper user access rights
3.5.5. File system tampering	4.5.5. File system tampering

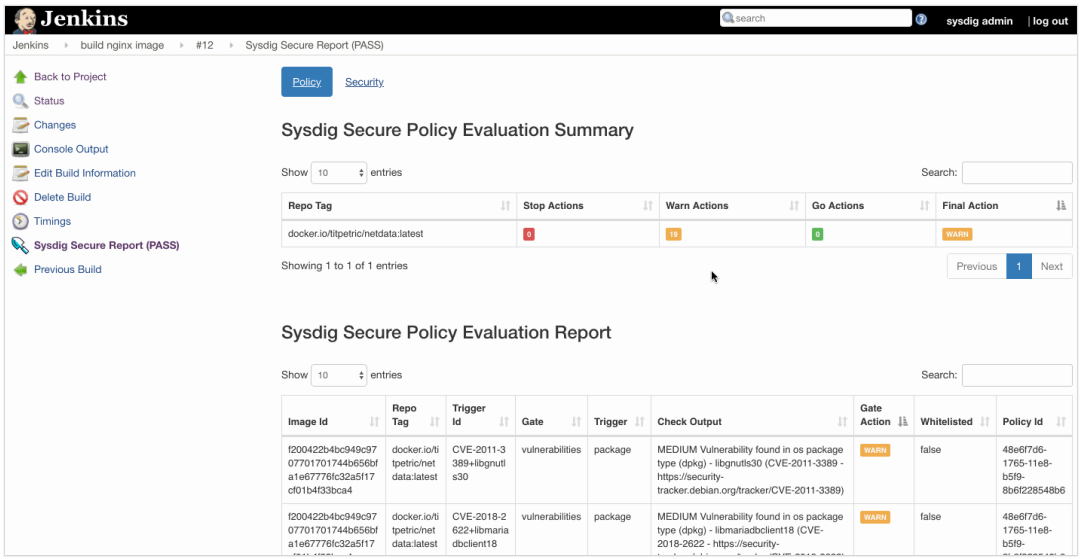
We will follow the structure of section 4 subsections, explaining how Sysdig Secure comprises features that have specific application to each of the risks described in the NIST publication SP 800-190.

Section 4.1 Image Countermeasures

4.1.1 Image vulnerabilities

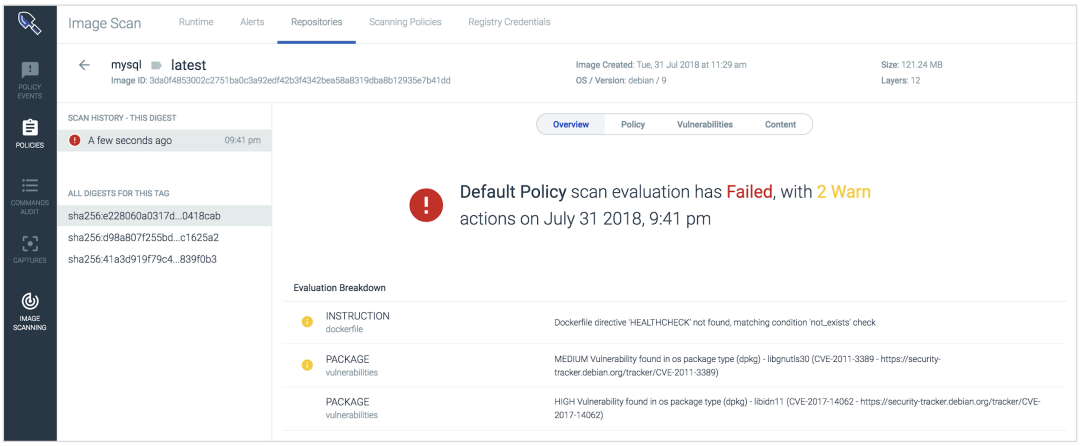
Sysdig Secure can integrate and monitor the risk and compliance of your images from build to deployment.

- **Build process integrations**. Sysdig Secure can integrate with many CI/CD tools, like Jenkins, Circle-CI, Gitlab CI/CD, Azure Pipelines, GitHub actions, Bamboo, AWS CodePipeline and CodeBuild, and Tekton. Also, other integrations are possible via API.



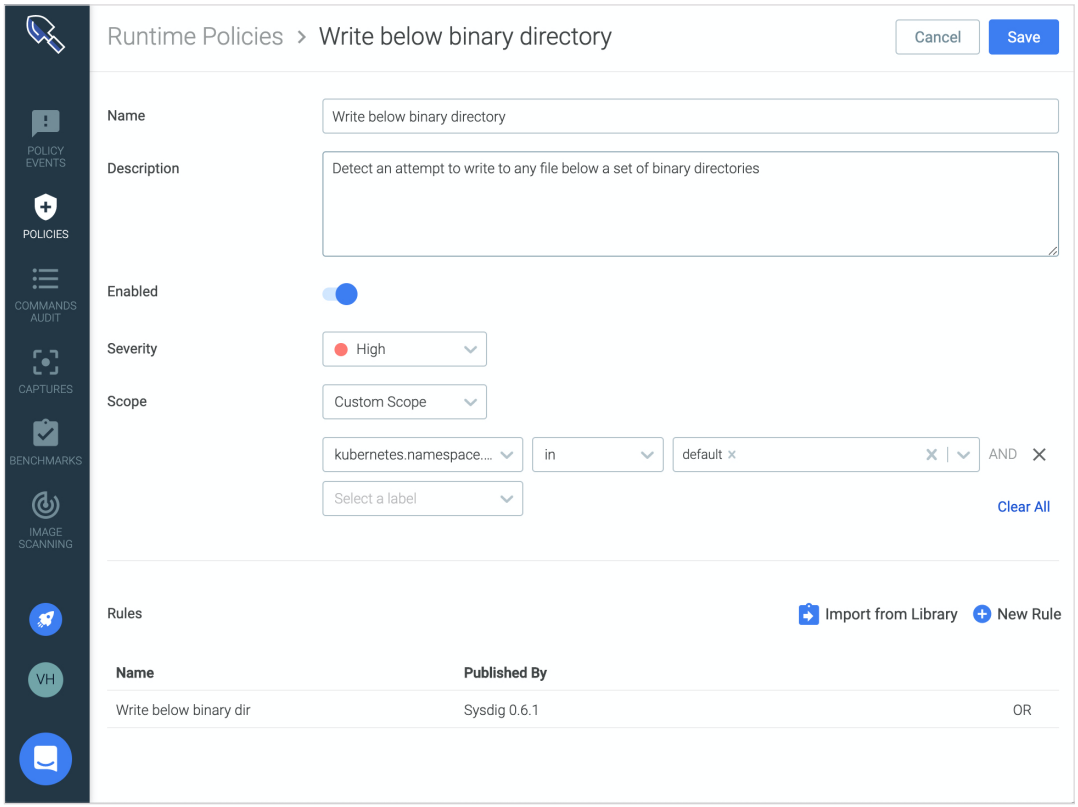
- **Registry Integrations.** Sysdig Secure can scan images stored in any Docker V2 compatible registry, such as CoreOS Quay, Amazon ECR, Docker Private Registries, Google Container Registry, JFrog Artifactory, Microsoft ACR, SuSE Portus and VMWare Harbor. Check out this post about Sysdig scanning images stored in Azure Container Registry.

- **Run-time Integrations.** Sysdig Secure validates that images that are running have been scanned as part of the CI/CD process, or within a registry. If images haven't been scanned, a team can alert and trigger actions. For images that were scanned earlier in the CI/CD process or in a registry, Sysdig Secure tracks the status of the images that are running and can alert if new vulnerabilities are discovered in contents of the running images.



The Sysdig scanning engine stores all of the final OS files, packages and language specific packages of a container. This means we'll get contents from all layers, and not flag false positives for packages that may have been introduced in one layer and then removed in another. These reports are available via the Sysdig Secure console, directly within Jenkins, and can be exported via the API.

Sysdig Secure can also implement metadata driven run-time policies for different environments (dev, prod, test) namespaces, clusters, etc. This makes it easy to enforce a different set of policies and actions that are relevant to each environment.



Here are a couple examples of image policy evaluation rules we've seen organizations put in place before images are deployed into production.

Security

- Does the image have critical vulnerabilities with a fix?
- Are there secrets or credentials exposed in the image?
- Does this image have exposed ports that I've blacklisted?

Compliance

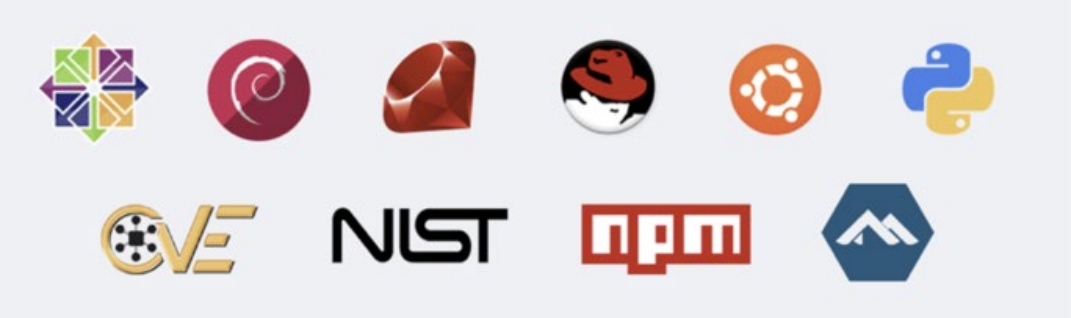
- What license types is the image using?
- Is this image built on a distribution our organization doesn't use?

Reliability

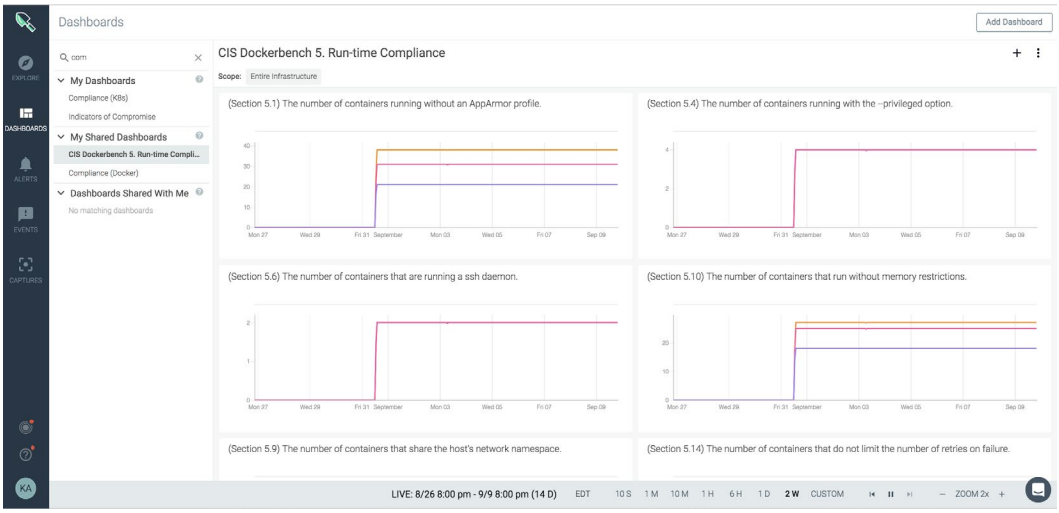
- Does my image have health checks?
- Are my developers building large images that can impact our infrastructure?
- Are my developers using an unofficial version of Ruby, Node, Java or Python packages?

Section 4.1.2 Image configuration defects

1. The policies mentioned in the section above map to many of the secure configuration settings and third-party best practices. Sysdig Secure will integrate with many different third-party vulnerability sources to make sure images are using vulnerability free and standard OS packages.
    - Once an image is running in an organization's environment, Sysdig Secure has a robust set of default rulesets from our open source detection engine, Falco. Falco rules will detect suspicious behaviors such as sensitive mounts, unexpected inbound/outbound activity, and the modification of system binaries.
2. Sysdig Secure creates a robust set of artifacts as part of the image analysis process (like a fingerprint database for container images). Because Sysdig Secure has stored all of the OS Packages, files and other contents of the image, we can update the status of the image in real time as updates come in from different feeds, all without having to rescan the image. This means that even a running container will be flagged, without rescanning, if a new CVE is discovered for it.



3. Sysdig Secure can integrate with Kubernetes Admission controllers to prevent vulnerable and non-compliant images from running on the clusters. Also, once containers are running, Sysdig Secure will actively monitor those containers with 200+ compliance checks to make sure the configuration of a container doesn't drift during its lifespan. Sysdig Secure has the option to fail an image build as well as stop or pause running containers if vulnerabilities or undesirable activity is detected.



4. Sysdig Secure Runtime security can enforce only trusted containers to be allowed, deployed and run (even if doing so manually outside of an orchestrator or automation tool). Sysdig Secure is also continually updating its CVE database, and comparing with all image fingerprints, running containers will also be identified if new CVEs that affect running workloads are released .

Section 4.1.3 Embedded malware

Sysdig Secure Runtime policies allow real-time enforcement to prevent containers from doing undesirable activities, such as running additional/unknown executables, opening unexpected ports, and performing questionable activity on the underlying host.

Section 4.1.4 Embedded clear text secrets

Sysdig Secure can detect if secrets, credentials or other pieces of sensitive data are included in an image, and can fail a build if that sensitive info is baked into the container image. This will help ensure that developers are following best practices and organizations use the native secrets management tooling effectively.

Here's an example JSON output of a rule that can be used to scan using a regex for potential secrets, credentials or other sensitive data.

```
"rules": [
  {
    "id": "f9e2e33c-46fa-4578-8f39-e8aa4ad559df",
    "trigger": "CONTENTMATCH",
    "params": [
      {
        "name": "SECRETCHECK_CONTENTREGEXP",
        "value": ".*\.pem"
      }
    ],
    "action": "STOP",
    "gate": "SECRETCHECK"
  }
```

Here's a Falco rule included in Sysdig Secure rules library that detects a grep execution trying to find private keys or passwords:

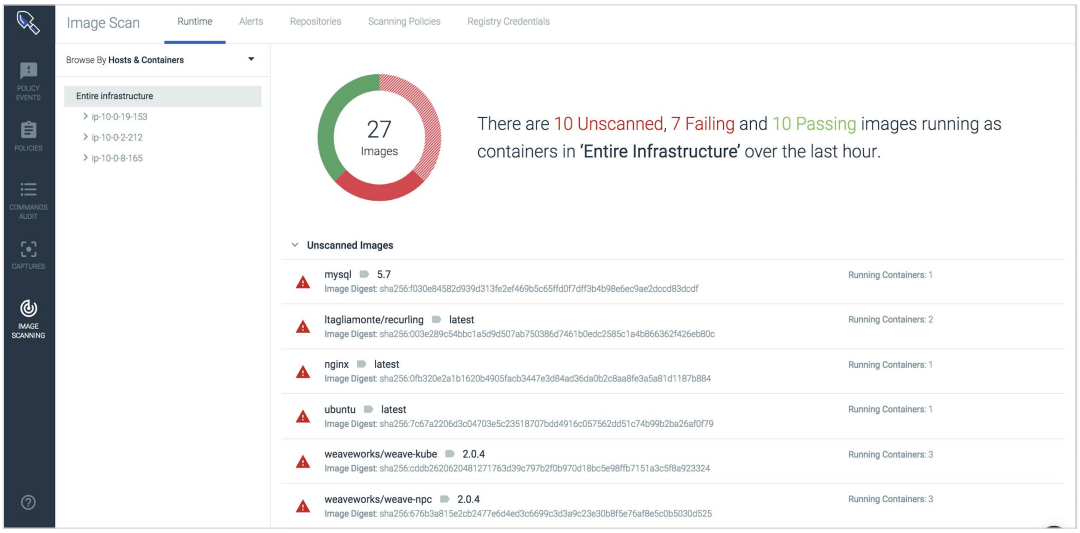
```
rule: Search Private Keys or Passwords
desc: >
  Detect grep private keys or passwords activity.
condition: >
  (spawned_process and
  ((grep_commands and private_key_or_password) or
   (proc.name = "find" and (proc.args contains "id_rsa" or
    proc.args contains "id_dsa"))))
output: >
  Grep private keys or passwords activities found
  (user=%user.name command=%proc.cmdline container_id=%container.id
  container_name=%container.name
  image=%container.image.repository:%container.image.tag)
priority: WARNING
tags: [process, mitre_credential_access]
```

Section 4.1.5 Use of untrusted images

Sysdig Secure can configure custom rules to kill containers that are from outside a known list of trusted registries. This helps protect against the common developer practice of pulling down images directly from Dockerhub. Here's an example Falco rule to detect this behavior:

```
- rule: image_from_external_registry
  desc: Container launched with an image from an external registry
  condition: >
    evt.type=execve and proc.vpid=1 and not container.id = host
    and not container.image startswith registry.gitlab.com
    and proc.cmdline startswith runc
  output: >
    container launched from external image
    (user=%user.name command=%proc.cmdline %container.info
    image=%container.image)
  priority: NOTICE
```

Sysdig Secure will track the status of all of the different repositories that we've analyzed within the registry, as well as the images that are running live within your environment. If the status of any of these images changes, Sysdig Secure can alert users that the risk of that image has increased.



Sysdig Secure can be integrated into image build pipelines, such as Jenkins. This pipeline can be extended to cryptographically sign a container image that has passed the Sysdig Secure Image Scanning process. Sysdig Secure Runtime Security can then be used to restrict the running of containers that don't come from trusted sources, such as those being cryptographically signed.

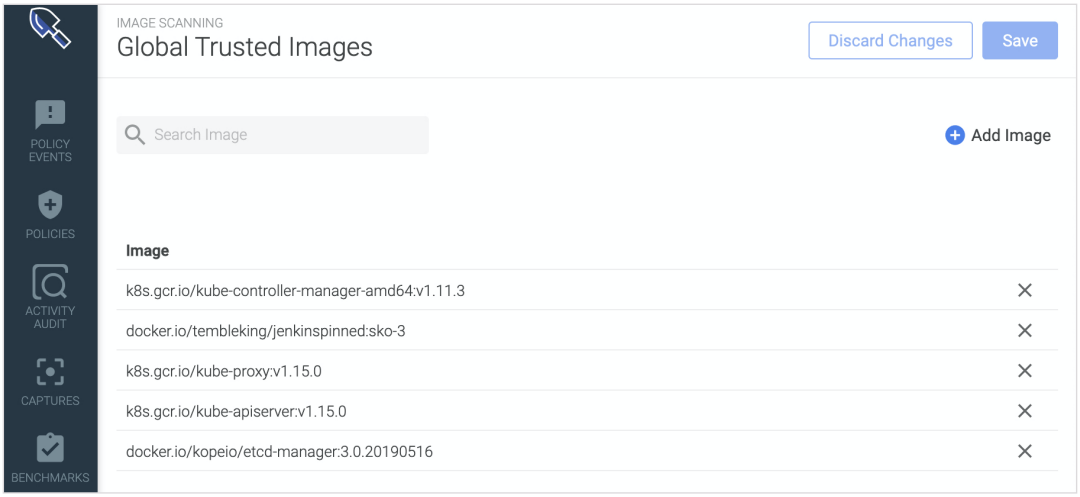
Section 4.2 Registry Countermeasures

Section 4.2.1 Insecure connections to registries

Sysdig Secure is designed to complement a secured registry. If this registry is run within a Sysdig Secure environment (with the agent monitoring and enforcing the host), then standard Sysdig Secure Runtime policies can be used to limit undesirable activity. However, Sysdig Secure should be used to extend a robust security policy of any application, including adequate firewalls, role based access controls, authentication enforcement and access auditing.

Section 4.2.2 Stale image in registry

Sysdig Secure can be used to only allow authorized container images, including restrictions on version or tags such as 'latest.' Combined with Sysdig Secure Image Scanning, this helps support an image management workflow to ensure that only authorized images and versions are used, and kept up-to-date accordingly.



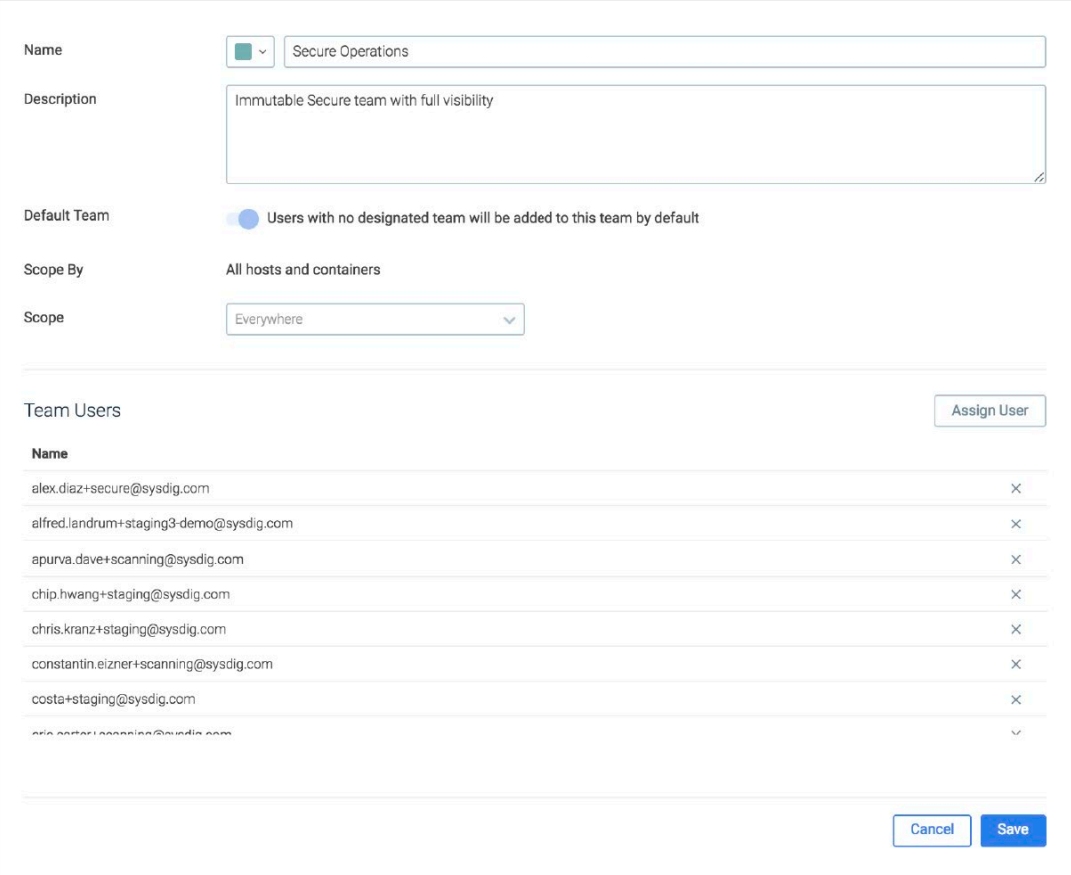
Section 4.2.3 Insufficient authentication and authorization restrictions

Sysdig Secure is designed to complement a secured registry. If this registry is run within a Sysdig Secure environment (with the agent monitoring and enforcing the host), then standard Sysdig Secure Runtime policies can be used to limit undesirable activity. However, Sysdig Secure can be used to extend existing security policies (role based access controls, authentication enforcement and access auditing). Sysdig Secure itself can be secured with the use role based access control (teams) as covered in 4.3.1.

Section 4.3 Orchestrator Countermeasures

Section 4.3.1 Unbounded administrative access

Sysdig Secure provides rich service-based access control to the data that is coming from your containerized applications. By using Sysdig Secure “Teams”, cluster operators can scope data with labels from hosts, containers, orchestrators and cloud providers to only show the content that is relevant to those application teams.



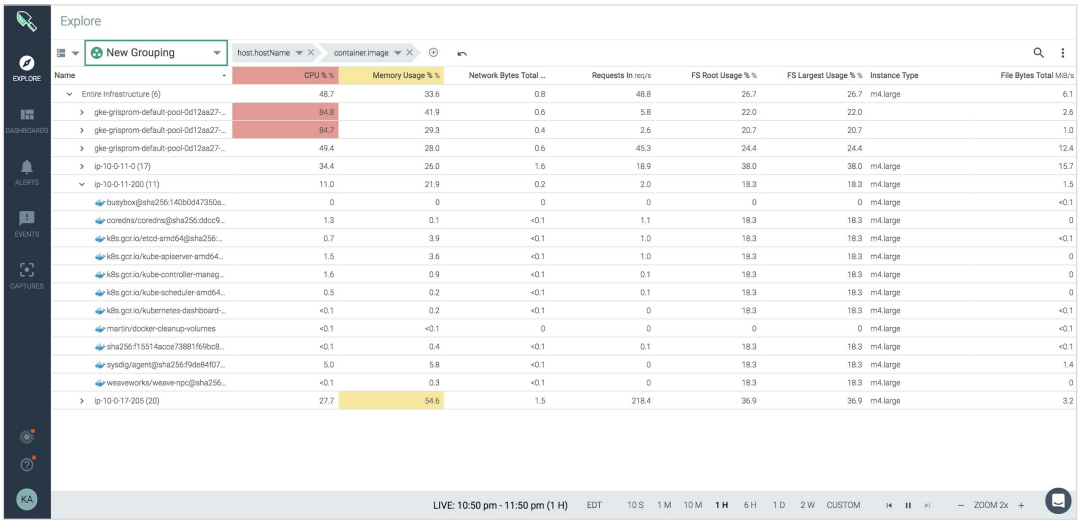
We can detect full Kubernetes administrative access with a Falco rule available in Sysdig Secure's rules library.

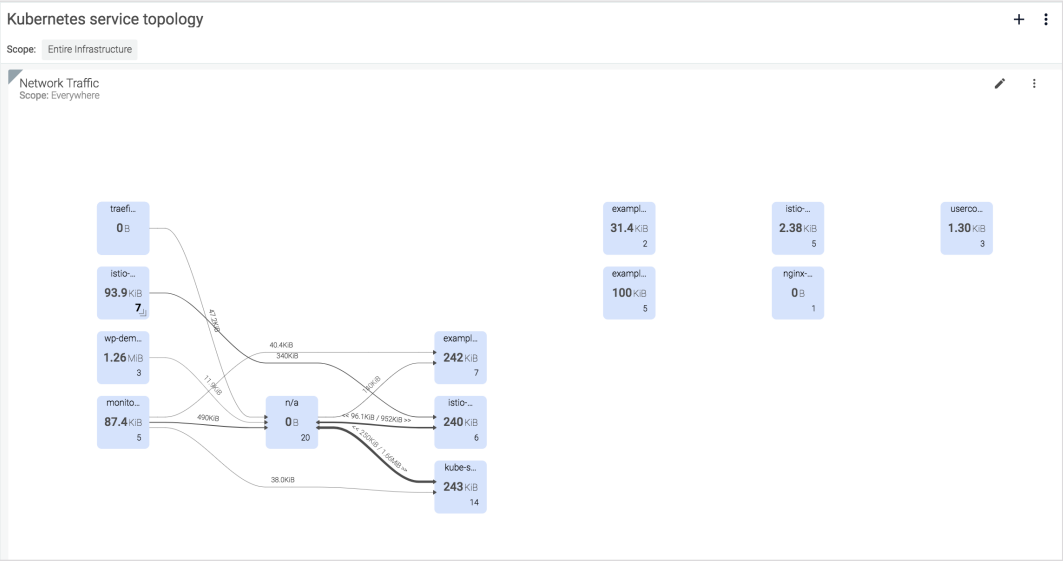
```
- rule: Full K8s Administrative Access
  desc: >
    Detect any k8s operation by an administrator with full access.
  condition: >
    kevt
    and non_system_user
    and ka.user.name in (admin_k8s_users)
    and not allowed_full_admin_users
  output: >
    K8s Operation performed by full admin user
    (user=%ka.user.name target=%ka.target.name/%ka.target.resource
    verb=%ka.verb uri=%ka.uri resp=%ka.response.code)
  priority: WARNING
  source: k8s_audit
  tags: [k8s]
```

Section 4.3.2 Unauthorized access

In addition to Sysdig Secure Teams covered in 4.3.1, single sign-on and centralized authentication can be used with Sysdig, such as LDAP, Active Directory, Google Auth, SAML and OpenID. We also recommend either running the Sysdig backend components on infrastructure that is encrypted (such as AWS EBS encrypted volumes), or leveraging Sysdig's own fully managed and secured SaaS platform.

At Sysdig, we provide robust dashboards for inventory management of containers, hosts, pods, deployments or any other construct that an organization needs to monitor.





You can find a Falco rule in the Sysdig Secure rules library to detect that an anonymous request to the Kubernetes API of a cluster has been allowed.

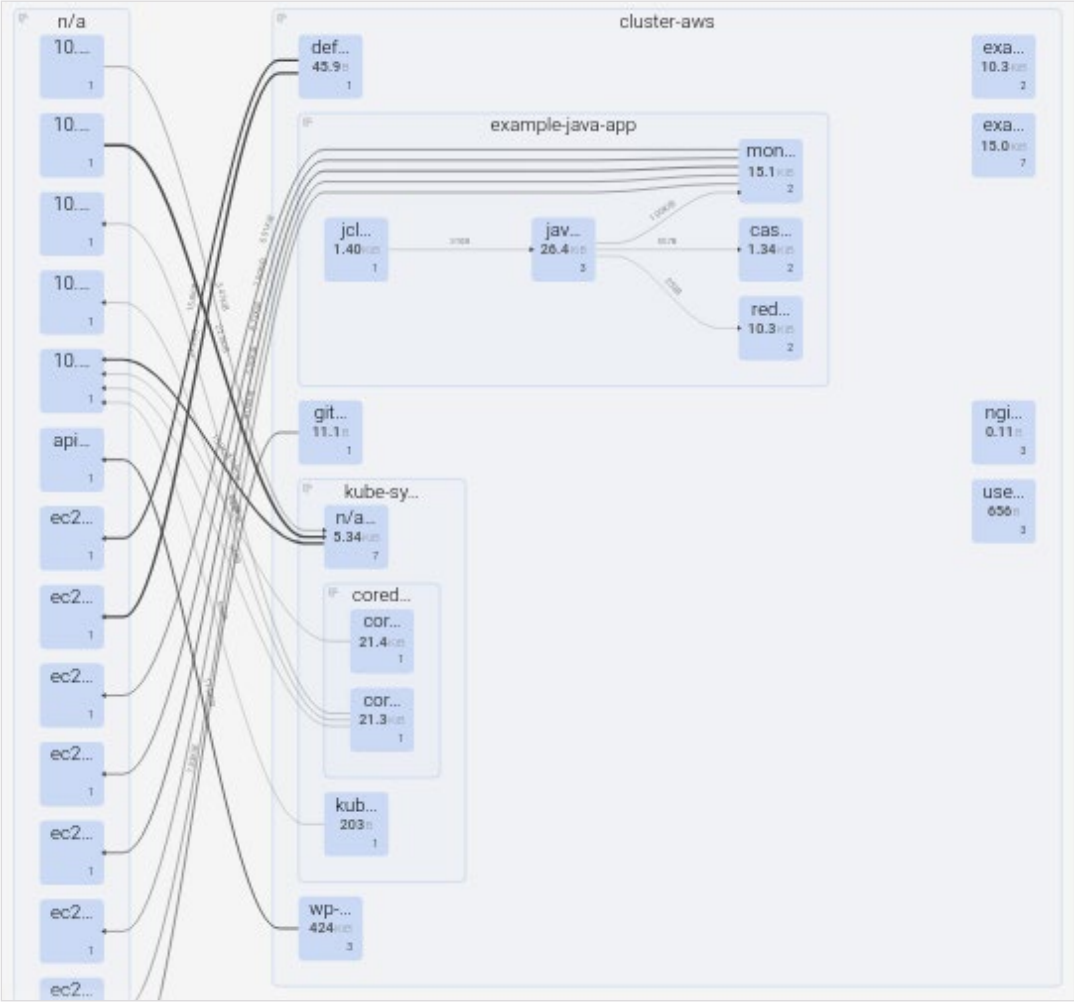
```
- rule: Anonymous Request Allowed
  desc: >
    Detect any request made by the anonymous user that was allowed
  condition: >
    kevt and ka.user.name=system:anonymous and
    ka.auth.decision!=reject and not health_endpoint
  output: >
    Request by anonymous user allowed (user=%ka.user.name
    verb=%ka.verb uri=%ka.uri reason=%ka.auth.reason))
  priority: WARNING
  source: k8s_audit
  tags: [k8s]
```

Section 4.3.3 Poorly separated inter-container network traffic

While Sysdig Secure is not responsible for configuring network segmentation (VLANs, VPCs, subnets, routing, firewall rules, etc.), it can help you visualize the connectivity between both the Orchestrators, as well as the containers running within the Orchestrator. Below is a network topology map that highlights the traffic flowing between these components.

The following screenshot illustrates the traffic communication within the application ‘example-java-app’ and also the traffic coming into the ‘kube-system’ components, including coredns. This is a dynamic communication map and shows actual communication between every component. This also includes external communication points, shown here as an

'n/a' group, as the external component does not include the relevant Kubernetes metadata information.



This information can be further complemented with Sysdig's capability to show connection table information (as shown below), and also with Sysdig's forensic capability to deeply interrogate and analyze questionable connections and activities.

Connections Table

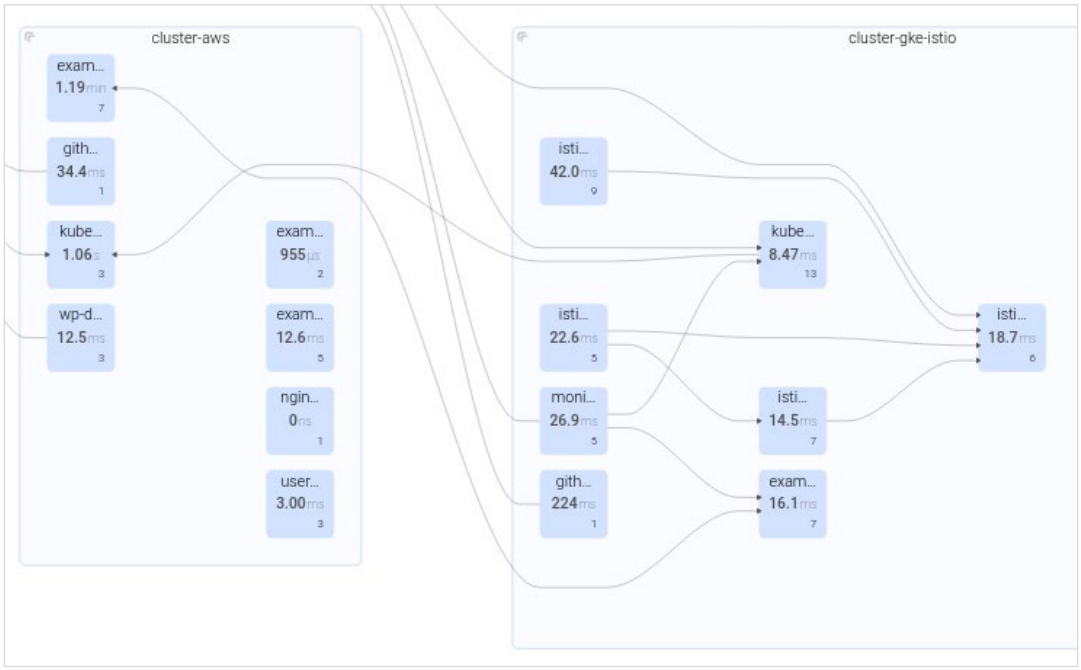
Local connection endpoint	Local connection ser...	Remote connection ...	Remote connection ...
ip-10-0-17-205 (k8s_mysql_mysql-58d5bd7559-...	mysqld:3306	10.36.0.9	10.44.0.14
ip-10-0-11-0 (k8s_wordpress_wordpress-6984d4...	apache2	10.44.0.14	10.44.0.14:3306
ip-10-0-17-205 (k8s_client_client-7ccb64ffc4-dm...	curl	10.104.181.65	10.104.181.65:80
ip-10-0-17-205 (k8s_javaapp_javaapp-67f8fcc7f6...	tomcat:33310	10.44.0.9	10.44.0.9
ip-10-0-17-205 (k8s_sysdig-agent_sysdig-agent-...	sdjagent	10.44.0.9	10.44.0.9:33310
ip-10-0-11-0 (k8s_javaapp_javaapp-67f8fcc7f6-m...	tomcat:33293	10.36.0.5	10.36.0.5
ip-10-0-11-0 (k8s_sysdig-agent_sysdig-agent-4zh...	sdjagent	10.36.0.5	10.36.0.5:33293
ip-10-0-11-200 (k8s_sysdig-agent_sysdig-agent-s...	cointerface	127.0.0.1	127.0.0.1:8125
ip-10-0-17-205 (k8s_sysdig-agent_sysdig-agent-...	cointerface	127.0.0.1	127.0.0.1:8125
ip-10-0-11-0 (k8s_sysdig-agent_sysdig-agent-4zh...	cointerface	127.0.0.1	127.0.0.1:8125
ip-10-0-11-200	kube-apiserver	ip-10-0-11-200 (k8s_etc...	etcd:2379
ip-10-0-17-205	python2.7	ip-10-0-17-205 (k8s_mo...	mongod:27017
ip-10-0-17-205 (k8s_vote_vote-674c99f6b6-27m...	gunicorn	127.0.0.1	127.0.0.1:8125
ip-10-0-11-0 (k8s_wordpress_wordpress-6984d4...	apache2:80	10.44.0.15	10.36.0.9
ip-10-0-17-205 (k8s_redis_redis-57944c5896-c6h...	redis-server:6379	10.44.0.2	10.44.0.1
ip-10-0-17-205 (k8s_vote_vote-674c99f6b6-27m...	gunicorn:80	10.36.0.3	10.44.0.2

For containers that should never be exposed to external traffic because they only service other containers in the cluster, a Falco rule can detect whenever an inbound or outbound external connection has been established.

```
- rule: Network connection outside local subnet
  desc: >
    Scoped images should only receive and send traffic
    to local subnet
  condition: >
    enabled_rule_network_only_subnet and
    inbound_outbound and
    container and
    not network_local_subnet and
    not k8s.pod.labels in
      (labels_whitelist_network_outside_subnet) and
    scope_network_only_subnet and
    k8s.ns.name in (namespace_scope_network_only_subnet)
  output: >
    Detected network connection outside local subnet
    (command=%proc.cmdline connection=%fd.name user=%user.name
    container_id=%container.id image=%container.image.repository
    namespace=%k8s.ns.name fd.rip.name=%fd.rip.name
    fd.lip.name=%fd.lip.name fd.cip.name=%fd.cip.name
    fd.sip.name=%fd.sip.name)
  priority: WARNING
  tags: [network, NIST, NIST_3.3, PCI, PCI_DSS_6.4.2]
```

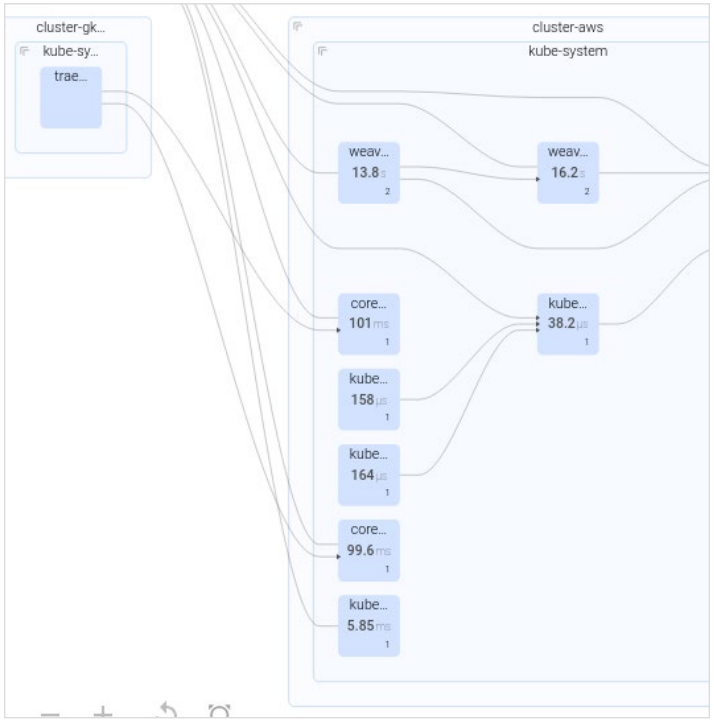
Section 4.3.4 Mixing of workload sensitivity levels

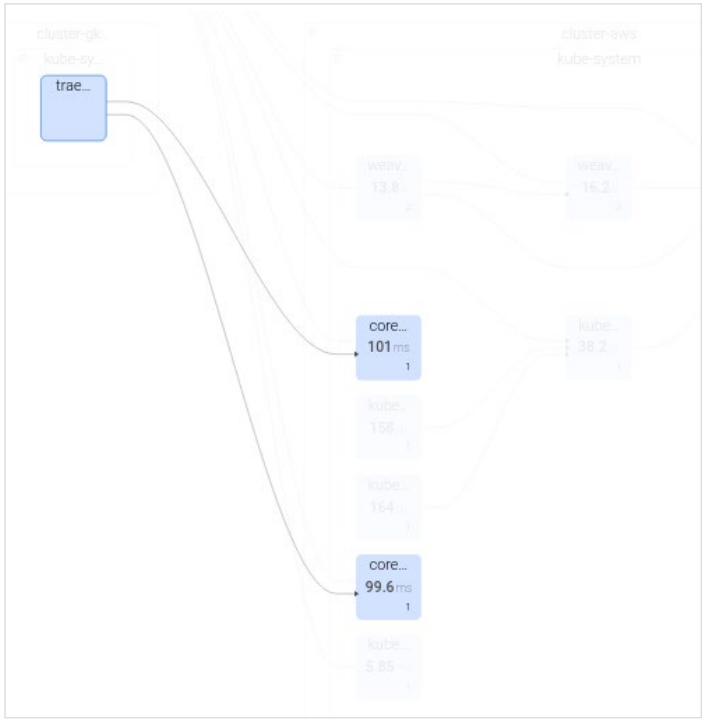
Coupled with our topology maps and connectivity information, we also leverage Orchestrator metadata to help group workloads together. This can make it easy to distinguish different security zones and applications, as well as identify any communication between them that maybe shouldn't exist. In the screenshot below, we see two independent Kubernetes clusters that happen to be hosted in different cloud providers, but could simply be hosted on physically isolated infrastructure. The topology map highlights where we have inter-cluster communication between these potentially security isolated environments.



Section 4.3.5 Orchestrator node trust

In addition to the capabilities provided by Sysdig above, the metadata that Sysdig ingests from the Orchestrators makes it easy to focus topology maps and dashboards with very specific requirements. In the following screenshot, the scope has been narrowed to the kube-system namespace and isolated by cluster. We have clicked the 'traefik' pod in the first cluster in order to highlight the traffic flow, and can see that this is talking to the 'coredns' pods in the second cluster. This helps identify and prove whether cluster resources within the orchestrator are fully isolated, or as highlighted here, share dependencies and support cluster inter-communication.





Falco rules can detect when a node that's not in a whitelist tries to join the cluster, or when it successfully joins it.

```
- rule: Untrusted Node Successfully Joined the Cluster
  desc: >
    Detect a node successfully joined the cluster outside
    of the list of allowed nodes.
  condition: >
    kevt and node
    and kcreate
    and response_successful
    and not allow_all_k8s_nodes
    and not ka.target.name in (allowed_k8s_nodes)
  output: >
    Node not in allowed list successfully joined the cluster
    (user=%ka.user.name node=%ka.target.name)
  priority: ERROR
  source: k8s_audit
  tags: [k8s]

- rule: Untrusted Node Unsuccessfully Tried to Join the Cluster
  desc: >
    Detect an unsuccessful attempt to join the cluster for a node
    not in the list of allowed nodes.
  condition: >
    kevt and node
    and kcreate
    and not response_successful
    and not allow_all_k8s_nodes
    and not ka.target.name in (allowed_k8s_nodes)
  output: >
    Node not in allowed list tried unsuccessfully to join the cluster
    (user=%ka.user.name node=%ka.target.name
    reason=%ka.response.reason)
  priority: WARNING
  source: k8s_audit
  tags: [k8s]
```

Section 4.4 Container Countermeasures

Section 4.4.1 Vulnerabilities within the runtime software

Sysdig Secure has agents on every node in the cluster to monitor all of the containers that are deployed in the environment. Sysdig Secure monitors the running images and evaluates those against the image policies defined by the platform administrators. This can be used to both prevent known vulnerabilities from being deployed, and from zero day attacks from being executed (or a more common scenario, simply preventing uninformed users from doing things they aren't aware they shouldn't do).

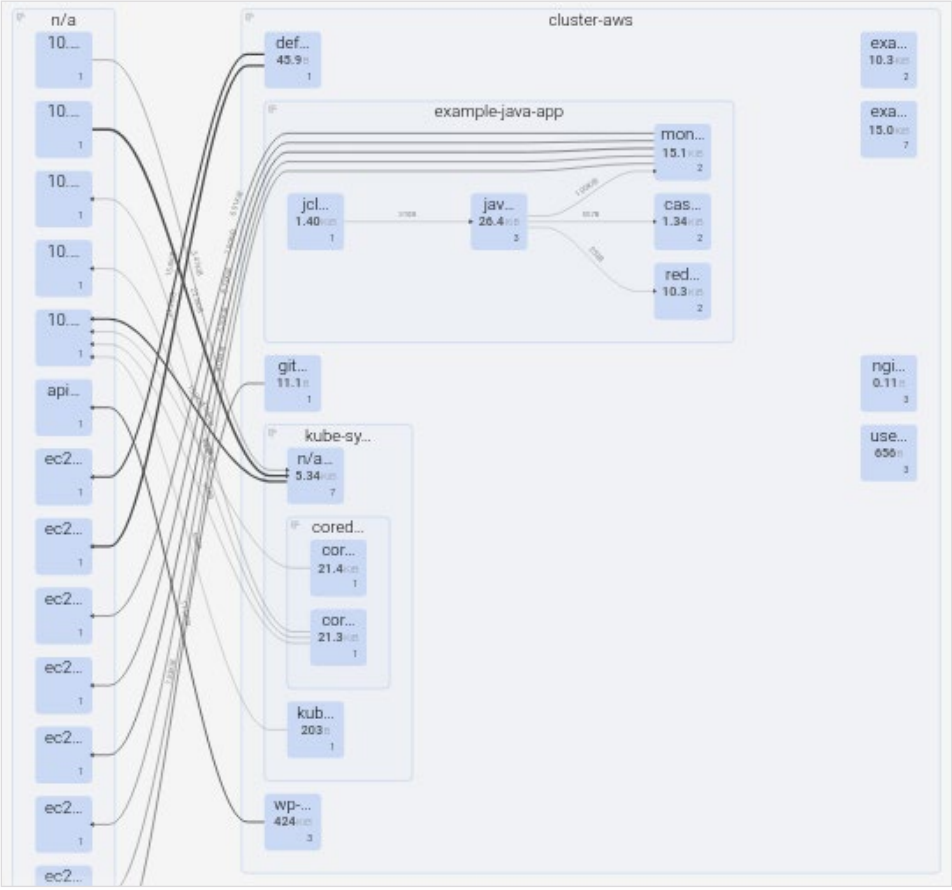
Several Falco rules will help you detect abnormal network connections, like the following one to detect attempts to use Kubernetes NodePorts from a container.

```
- rule: Unexpected K8s NodePort Connection
  desc: Detect attempts to use K8s NodePorts from a container
  condition: >
    (inbound_outbound) and fd.sport >= 30000 and
    fd.sport <= 32767 and container and
    not nodeport_containers
  output: >
    Unexpected K8s NodePort Connection (command=%proc.cmdline
    connection=%fd.name container_id=%container.id
    image=%container.image.repository)
  priority: NOTICE
  tags: [network, k8s, container, mitre_port_knocking]
```

Section 4.4.2 Unbounded network access from containers

Sysdig provides automatic discovery of containers and Kubernetes nodes and services with a real-time topology map showing all containers, hosts and processes. You can also see connections across namespaces, clusters, and hosts.

The below screenshot illustrates the traffic communication within the application 'example-java-app' and also the traffic coming into the 'kube-system' components, including coredns. This is a dynamic communication map and details actual communication between every component. This also includes external communication points, shown here as 'n/a'. The external component doesn't include the relevant Kubernetes metadata information which is why it's grouped as 'n/a'.

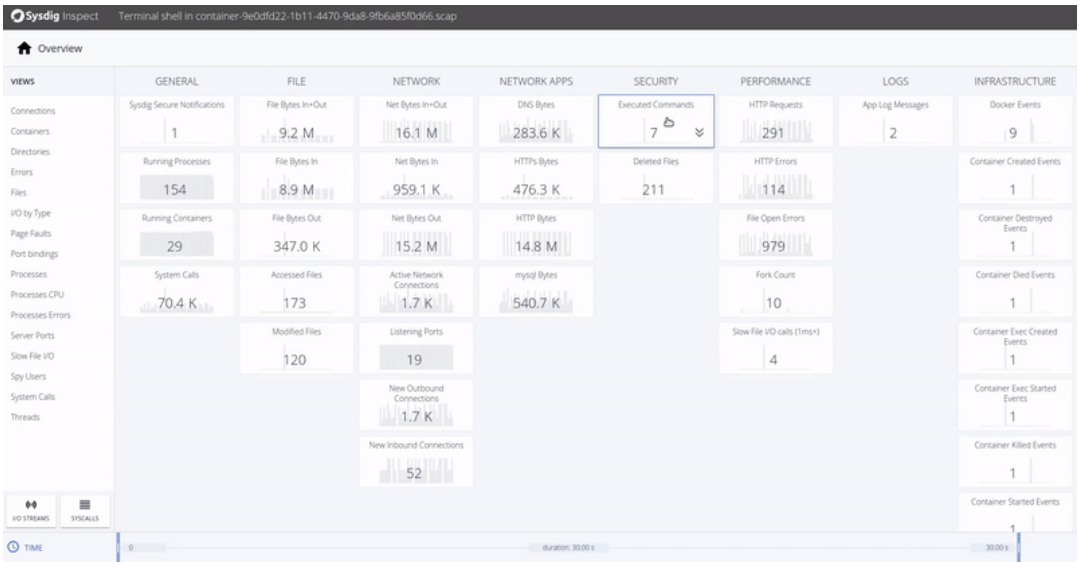


This information can be further complemented with Sysdig capability to show connection table information (as shown below), and also with Sysdig forensic capabilities to deeply interrogate and analyze questionable connections and activities.

Connections Table

Local connection endpoint	Local connection ser...	Remote connection ...	Remote connection ...
ip-10-0-17-205 (k8s_mysql_mysql-58d5bd7559-...	mysqld:3306	10.36.0.9	10.44.0.14
ip-10-0-11-0 (k8s_wordpress_wordpress-6984d4...	apache2	10.44.0.14	10.44.0.14:3306
ip-10-0-17-205 (k8s_client_client-7ccb64ffc4-dm...	curl	10.104.181.65	10.104.181.65:80
ip-10-0-17-205 (k8s_javaapp_javaapp-67f8fcc7f6...	tomcat:33310	10.44.0.9	10.44.0.9
ip-10-0-17-205 (k8s_sysdig-agent_sysdig-agent-...	sdjagent	10.44.0.9	10.44.0.9:33310
ip-10-0-11-0 (k8s_javaapp_javaapp-67f8fcc7f6-m...	tomcat:33293	10.36.0.5	10.36.0.5
ip-10-0-11-0 (k8s_sysdig-agent_sysdig-agent-4zh...	sdjagent	10.36.0.5	10.36.0.5:33293
ip-10-0-11-200 (k8s_sysdig-agent_sysdig-agent-s...	cointerface	127.0.0.1	127.0.0.1:8125
ip-10-0-17-205 (k8s_sysdig-agent_sysdig-agent-...	cointerface	127.0.0.1	127.0.0.1:8125
ip-10-0-11-0 (k8s_sysdig-agent_sysdig-agent-4zh...	cointerface	127.0.0.1	127.0.0.1:8125
ip-10-0-11-200	kube-apiserver	ip-10-0-11-200 (k8s_etc...	etcd:2379
ip-10-0-17-205	python2.7	ip-10-0-17-205 (k8s_mo...	mongod:27017
ip-10-0-17-205 (k8s_vote_vote-674c99f6b6-27m...	gunicorn	127.0.0.1	127.0.0.1:8125
ip-10-0-11-0 (k8s_wordpress_wordpress-6984d4...	apache2:80	10.44.0.15	10.36.0.9
ip-10-0-17-205 (k8s_redis_redis-57944c5896-c6h...	redis-server:6379	10.44.0.2	10.44.0.1
ip-10-0-17-205 (k8s_vote_vote-674c99f6b6-27m...	gunicorn:80	10.36.0.3	10.44.0.2

As the definitive line of defense, Sysdig Secure can detect malicious activity at runtime, such as a terminal shell launching within a container, and execute actions like stopping or pausing the container. In addition to this, a system capture can be taken which can then be used to forensically analyze the event. This forensic analysis is vitally important in understanding how an undesirable event happened, potentially leading to the detection of zero day vulnerabilities or unidentified code exploits.



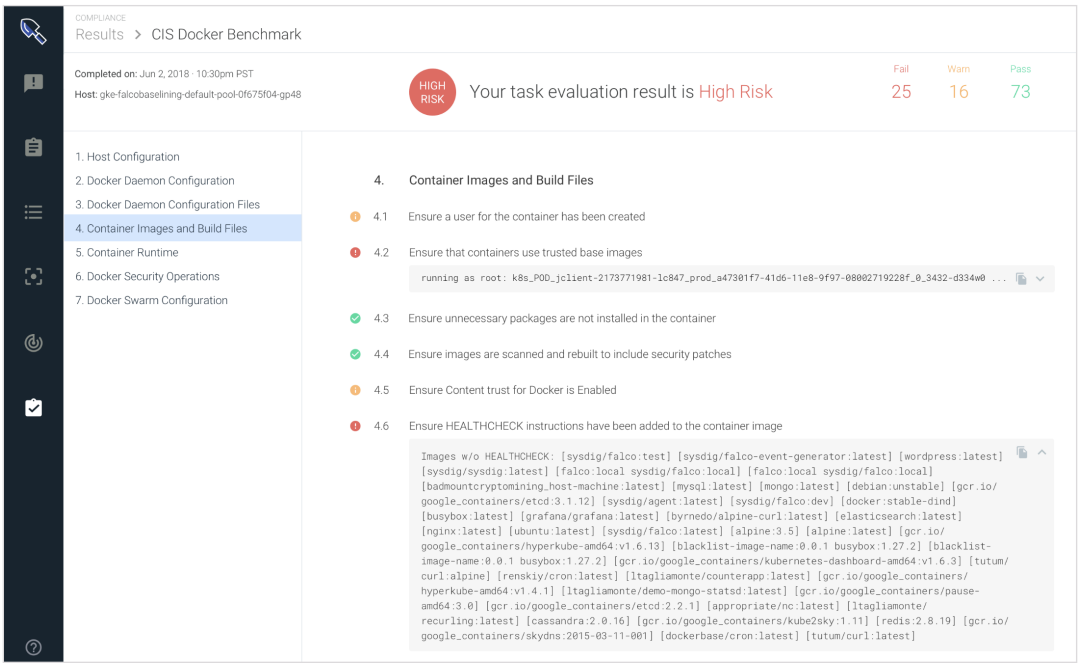
The Falco rule available in Sysdig Secure rules library to detect a terminal shell spawn in a container is the following:

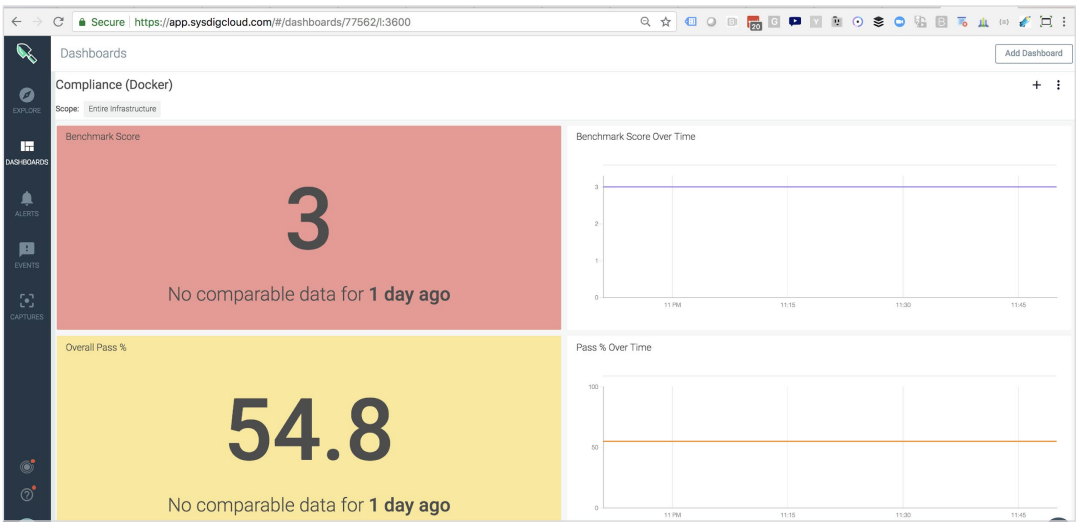
```
- rule: Terminal shell in container
  desc: >
    A shell was used as the entrypoint/exec point into a container
    with an attached terminal.
  condition: >
    spawned_process and container
    and shell_procs and proc.tty != 0
    and container_entrypoint
  output: >
    A shell was spawned in a container with an attached terminal
    (user=%user.name %container.info shell=%proc.name
    parent=%proc.pname cmdline=%proc.cmdline terminal=%proc.tty
    container_id=%container.id image=%container.image.repository)
  priority: NOTICE
  tags: [container, shell, mitre_execution]
```

Section 4.4.3 Insecure container runtime configurations

Sysdig Secure has automated the ability to continuously assess the compliance of containerized infrastructure with the CIS Docker Benchmark and the CIS Kubernetes Benchmark.

These results are exposed in two different formats. The first is metrics that can be used to monitor and alert on how your compliance posture is changing over time, so you know if your security posture is getting better or worse. The second is a report for auditors that is automatically generated from reports that facilitate the audit process.





You can detect that a privileged container has been launched at any time using this Falco rule available at the Sysdig Secure rules library:

```
- rule: Launch Privileged Container
  desc: >
    Detect the initial process started in a privileged container.
    Exceptions are made for known trusted images.
  condition: >
    container_started and container
    and container.privileged=true
    and not falco_privileged_containers
    and not user_privileged_containers
  output: >
    Privileged container started (user=%user.name
    command=%proc.cmdline %container.info
    image=%container.image.repository:%container.image.tag)
  priority: INFO
  tags: [container, cis, mitre_privilege_escalation, mitre_lateral_
movement]
```

Section 4.4.4 App vulnerabilities

Sysdig Secure Runtime Security can be leveraged to detect application anomalies, alert and take action. Standard Sysdig Secure Runtime Security policies include the following whitelist / blacklist controls:

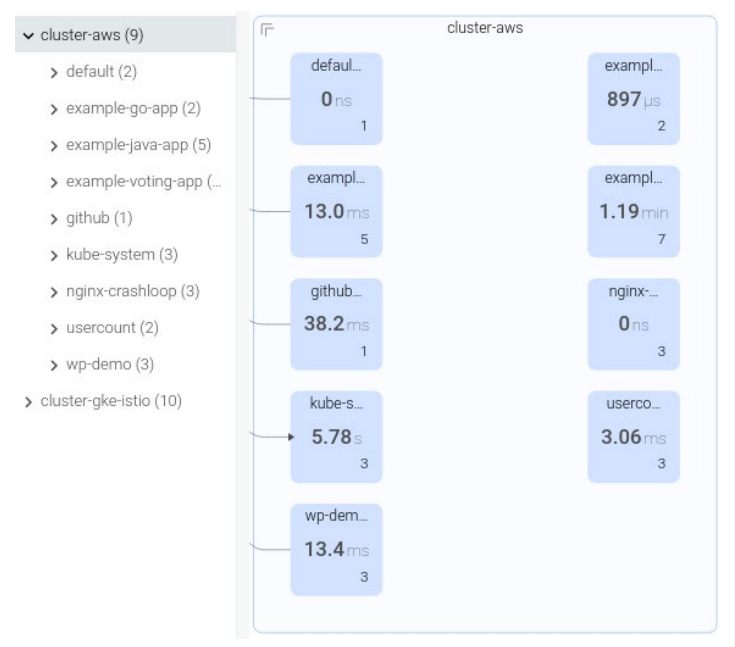
- Processes (i.e. mysqld, ssh, nginx, etc.)
- Container Images (i.e. cassandra, mongo:latest, nginx@sha256:a119c62...162d8b5, etc)
- Network Connections (i.e. 80, 443, TCP/UDP, etc.)
- Filesystem Operations (i.e. read or read/write operations to /etc, /var, /dev, /proc, etc.)
- System Calls (i.e. open, execve, chmod, chroot, etc.)

In addition to these controls, more advanced policies can include selective logic by using Falco rules from the rules library, or by creating your own. For example, this is a Falco rule available at the rules library that detects a modification of a binary executable under any of the registered binary directories.

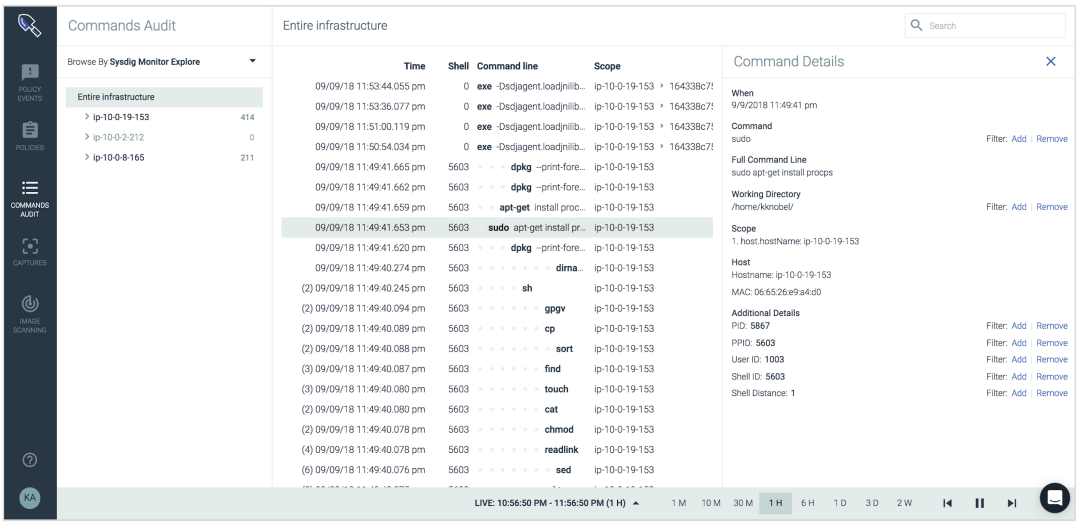
```
- rule: Modify binary dirs
  desc: >
    An attempt to modify any file below a set of binary directories.
  condition: >
    (bin_dir_rename) and
    modify and not package_mgmt_procs and
    not exe_running_docker_save
  output: >
    File below known binary directory renamed/removed
    (user=%user.name command=%proc.cmdline
    pcmdline=%proc.pcmdline operation=%evt.type file=%fd.name
    %evt.args container_id=%container.id
    image=%container.image.repository)
  priority: ERROR
  tags: [filesystem, mitre_persistence]
```

Section 4.4.5 Rogue container

At Sysdig, we automatically leverage an orchestrator's metadata, as well as any additional metadata (such as labels, tags, custom Sysdig Agent tags, etc.), and allow users to organize / create topologies views using this information. In addition, our full role-based access control ('Teams') allows views to be isolated and designed around a least-privileged model to ensure that users only see information that is relevant to them. This metadata is also used to apply Sysdig Secure policies and alerts. For example, any workload tagged as 'Test' will automatically inherit a 'Test' security policy and the relevant alerts. This allows us to drop straight into any environment and not change any practices, but still provide the benefit of enforcement and alerting.



We also monitor user activity and orchestrator events. This means that a full audit is available and stored completely off box, so there is no ability to tamper with this. The audit includes commands as well as orchestrator events (such as Kubernetes scaling, Docker kill, etc.) and can be extended with other events (such as Jenkins jobs, Splunk events, etc.)

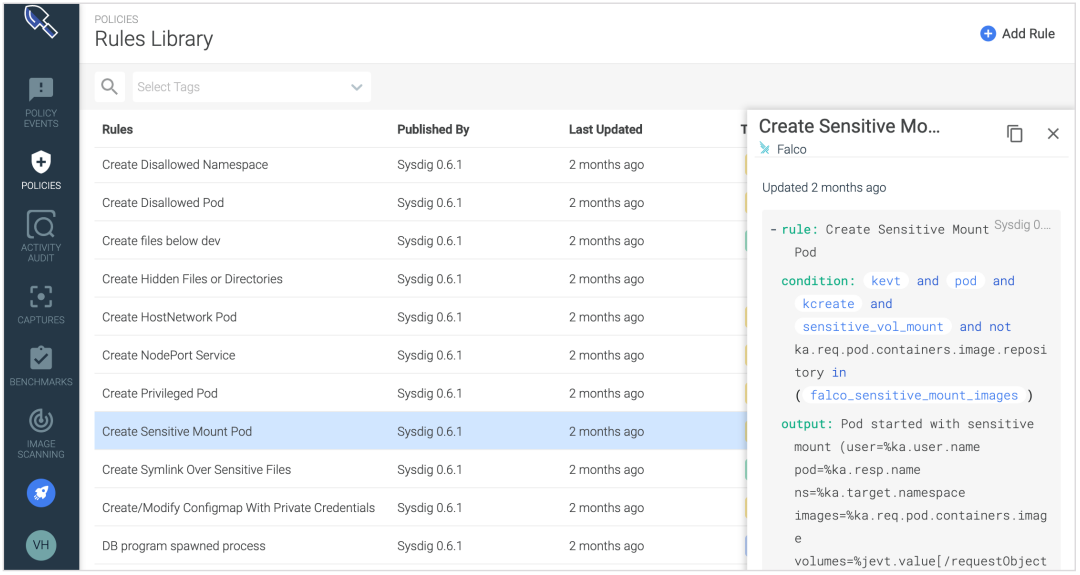



```
- rule: Create Disallowed Pod
  desc: >
    Detect an attempt to start a pod with a container image outside
    of a list of allowed images.
  condition: kevt and pod and kcreate and not allowed_k8s_containers
  output: >
    Pod started with container not in allowed list
    (user=%ka.user.name pod=%ka.resp.name ns=%ka.target.namespace
    images=%ka.req.pod.containers.image)
  priority: WARNING
  source: k8s_audit
  tags: [k8s]
```

Section 4.5 Host OS Countermeasure

Sysdig Secure Runtime Security can detect activities on the host OS, as well as within containers. The benefit of this is to provide a similar level of protection on the host OS as with the containers. However, Sysdig Secure provides security, compliance and monitoring for containers and Kubernetes platforms. Sysdig recommends following the NIST Guide for General Service Security. Sysdig should be a complimentary component to cover container and Kubernetes security, that is part of an overall security strategy.

Sysdig Secure has default policies that look for malicious behavior (i.e. sensitive mounts, writes below /etc, attempts to modify a binary directory) and many other default rules that look for unexpected file activity. If a policy is violated, actions can be taken to kill or pause a container.




```
- rule: Launch Sensitive Mount Container
  desc: >
    Detect the initial process started by a container that has a
    mount from a sensitive host directory (i.e. /proc).
    Exceptions are made for known trusted images.
  condition: >
    container_started and container
    and sensitive_mount
    and not falco_sensitive_mount_containers
    and not user_sensitive_mount_containers
  output: >
    Container with sensitive mount started (user=%user.name
    command=%proc.cmdline %container.info
    image=%container.image.repository:%container.image.tag
    mounts=%container.mounts)
  priority: INFO
  tags: [container, cis, mitre_lateral_movement]
```

To learn more about how Sysdig Secure validates compliance visit
https://sysdig.com/products/kubernetes-security/container-compliance/

You can also sign-up for a Sysdig Secure free 30-day trial at
https://sysdig.com/company/free-trial/

www.sysdig.com