cisco

Kubernetes Security

PSP Deprecation: An Opportunity for a New Security Model

Who Am I?

Ariel Shuper

Principal Product Manager @Cisco ET&I

Past:

VP Product Management @Porshift (acquired by Cisco) developing a cloud native security tools for Kubernetes and Istio, Head of Serverless Security@Aqua, developing serverless security tools for public clouds, Cloud Security PM @Check Point SW technologies

OSS contributor:

- *Kubei (runtime vulns. Scanning)*
- *Istio Security WG*
- *MITRE ATT&CK matrix for containers*

Kubernetes Security: Pod Security Basics

Pods are the basic unit of execution and control in Kubernetes

- A Pod is composed of one or more containers: an “app” container and optionally more helper containers

Each Pod has a ***securityContext*** that controls runtime security configurations for the pod

These settings are very powerful but can be used to enlarge the attack surface if not managed carefully

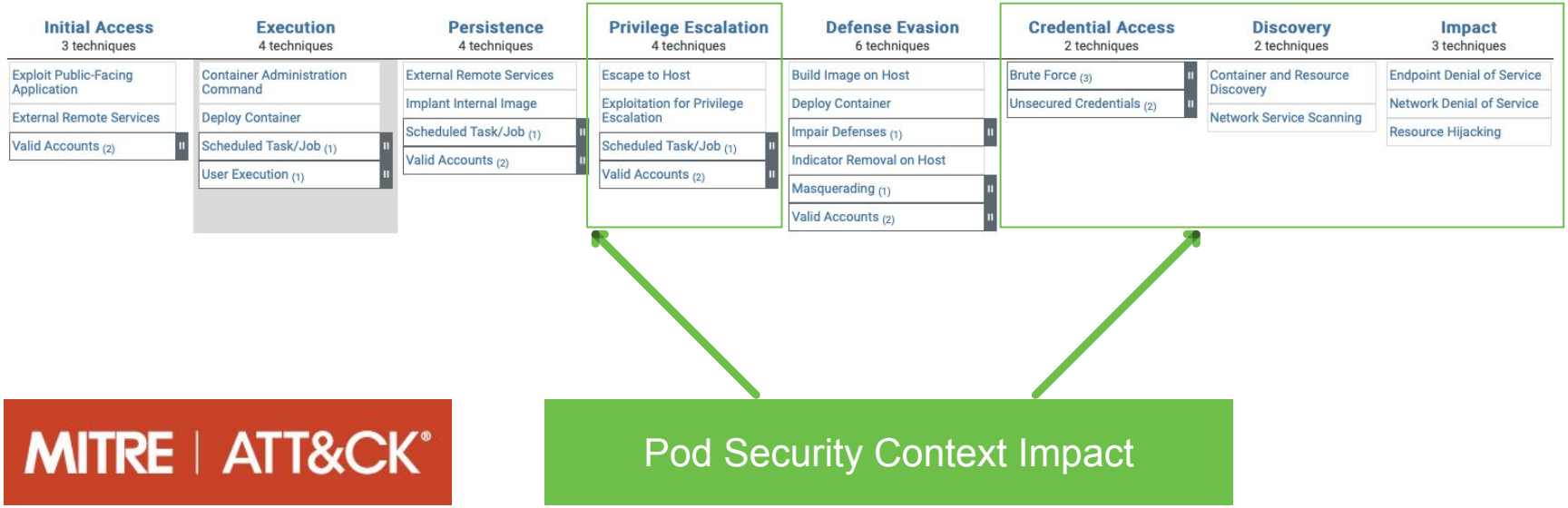
Pod Security Policies (PSPs) were designed to control the security-context settings that pod must pass for it to run -> Otherwise it’ll be rejected

PSP Structure (source: AWS/EKS)

```yaml
apiVersion: policy/v1beta1
kind: PodSecurityPolicy
metadata:
  name: eks.privileged
  annotations:
    kubernetes.io/description: 'privileged allows full unrestricted access to
      pod features, as if the PodSecurityPolicy controller was not enabled.'
    seccomp.security.alpha.kubernetes.io/allowedProfileNames: '*'
  labels:
    kubernetes.io/cluster-service: "true"
    eks.amazonaws.com/component: pod-security-policy
spec:
  privileged: true
  allowPrivilegeEscalation: true
  allowedCapabilities:
  - '*'
  volumes:
  - '*'
  hostNetwork: true
  hostPorts:
  - min: 0
    max: 65535
  hostIPC: true
  hostPID: true
  runAsUser:
    rule: 'RunAsAny'
  seLinux:
    rule: 'RunAsAny'
  supplementalGroups:
    rule: 'RunAsAny'
  fsGroup:
    rule: 'RunAsAny'
  readOnlyRootFilesystem: false
```

```yaml
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRole
metadata:
  name: eks:podsecuritypolicy:privileged
  labels:
    kubernetes.io/cluster-service: "true"
    eks.amazonaws.com/component: pod-security-policy
rules:
- apiGroups:
  - policy
  resourceNames:
  - eks.privileged
  resources:
  - podsecuritypolicies
  verbs:
  - use
---
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRoleBinding
metadata:
  name: eks:podsecuritypolicy:authenticated
  annotations:
    kubernetes.io/description: 'Allow all authenticated users to create privileged pods.'
  labels:
    kubernetes.io/cluster-service: "true"
    eks.amazonaws.com/component: pod-security-policy
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: ClusterRole
  name: eks:podsecuritypolicy:privileged
subjects:
  - kind: Group
    apiGroup: rbac.authorization.k8s.io
    name: system:authenticated
```

The Security Context of PSP



Why PSP is Deprecated

PSP is difficult to use correctly in the cluster:

- Adding PSP to a running cluster without policies will result in denying pods entry
- The lack of auditing functionality makes it difficult to roll-out PSP in existing clusters
- When a PSP resource is created it does nothing - in order to use it the user or target pod's service account must be authorized to use the policy

PSP started deprecation Kubernetes v1.21 and will be removed in v1.25

The New Model: PSS & Admission Controller

fwd:cloudsec 2021

- A new model simplifies the usage of pod's security-context
- Pod Security Standards define the isolation/hardening levels for Pods (how to restrict the behavior of pods in a clear, consistent fashion)
- A built-in *Pod Security admission controller* enforce the security
- Kubernetes Policies which are decoupled from their enforcements

Pod Security Standards (Kubernetes v 1.22)

The Pod Security Standards define three different *policies* to broadly cover the security spectrum.

These policies are *cumulative* and range from highly-permissive to highly-restrictive

Profile	Description
Privileged	Unrestricted policy, providing the widest possible level of permissions. This policy allows for known privilege escalations.
Baseline	Minimally restrictive policy which prevents known privilege escalations. Allows the default (minimally specified) Pod configuration.
Restricted	Heavily restricted policy, following current Pod hardening best practices.

Pod Security Standards

Privileged

Not secure and alls all settings

Baseline

Min restriction: focused on privileges escalation:

- Host Namespaces
- Privileged Containers
- Capabilities
- HostPath Volumes
- Host Ports
- AppArmor, SELinux
- /proc mount type
- Sysctls

Restricted

Heavily restricted, pod hardening best practices:

- Volume types
- Running as Non-root
- Non-root Groups
- Seccomp

Kubernetes Admission Controller

An admission controller intercepts requests to the Kubernetes API server prior to persistence of the object

There are two special controllers:

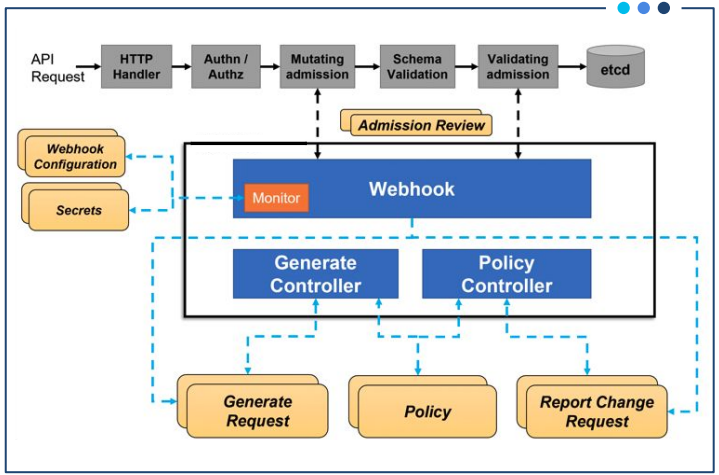
MutatingAdmissionWebhook (may modify an object)	ValidatingAdmissionWebhook (validate object state)

Admission controllers may be "validating", "mutating", or both

The admission control process proceeds in two phases:

First phase: mutating admission controllers are run	Second phase, validating admission controllers are run

If any of the controllers reject the request, the entire request is rejected immediately



Source: Kyverno

New Security Model: The Challenges

The new model is a great initiative accelerating the adoption of security hardening best practices

There’re few aspects that worth consideration:

Namespaces level implementation:		Only ValidatingWebhook option	
Dictate same security standard for entire namespace	No granularity in applying different standard to different services (different risk/impact)	Deployment failures upon violation	Add additional configuration effort on the user

Proposed Security Model

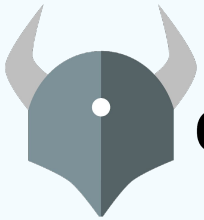
Hybrid Admission Controller

- Using both *MutatingAdmissionWebhook* and *ValidatingAdmissionWebhook*
- The “mutating” AC will “fix” the security context according to the policy (saves time, knowledge gap)
- The “validating” AC will enforce that there’re no deviations before deployments (all pods meet the desired state)

Granular Implementation of security-standards

- Using pod’s ID (e.g. SPIFFE) or Kubernetes labels to achieve more granular enforcement options
- Allowing to apply different policies on different services in the same namespace (e.g. default)

Security Policies



Open Policy Agent

VS

Kyverno

Kubernetes Policies: Why and How

Kubernetes uses the concept of “policy” to manage its objects

Policy engines allow to control Kubernetes resources in four categories:

Limit Ranges	Resources Quatas	Pod Security Context	Process ID (PID) limitations

CNCF have 2 opensource policy engines to manage policies in a Kubernetes:

OPA	Kyverno

The goal is to leverage these policy engines by developers to simplifies policy creation

Open Policy Agent

fwd:cloudsec 2021

- OPA is a lightweight general-purpose policy engine
- Policies are written using purpose-built, declarative language called Rego
- Each Rego file defines a policy module using a collection of rules that describe the expected state of the service
- OPA uses Gatekeeper (CNCF project for Kubernetes-specific implementation)
- CNCF stats:

Age 4 years	Status: Graduated	Github stars: 5.5K (OPA) 2k (Gatekeeper)

```
package kubernetes.validating.deny_privileged_mode

deny[msg] {
    some c
    input_container[c]
    c.securityContext.privileged
    msg := sprintf("Container '%v' should not run in privileged mode.", [c.name])
}

input_container[container] {
    container := input.request.object.spec.containers[_]
}

input_container[container] {
    container := input.request.object.spec.initContainers[_]
}
```

Rego based policy for PSS privileged mode)

Kyverno

fwd:cloudsec 2021

An open-source policy engine designed for Kubernetes.

It can validate, mutate, and generate configurations using admission controls and background scans

Kyverno policies are Kubernetes resources and do not require learning a new language

CNCF stats:

Age 2 years	Status: Sandbox	Github stars: 1.3K

```
kubectl create -f- << EOF
apiVersion: kyverno.io/v1
kind: ClusterPolicy
metadata:
  name: require-labels
spec:
  validationFailureAction: enforce
  rules:
  - name: check-for-labels
    match:
      resources:
        kinds:
        - Pod
    validate:
      message: "label 'app.kubernetes.io/name' is required"
      pattern:
        metadata:
          labels:
            app.kubernetes.io/name: "?*"
EOF
```

Open Policy Agent

V S

Kyverno

Advantages:

- Capable of expressing very complex policy
- More establishment in the community
- Supports multiple replicas for availability and scale

- Simplicity of policy expression allowing Kubernetes-style composition
- Solid mutation abilities
- Add generation and syncing options

Disadvantages:

- Dedicated programming language (rego)
- Mutation ability is emerging, validation use cases are more matured

- Complex policies are not possible
- Young and early stage (still unproven)

https://neonmirrors.net/post/2021-02/kubernetes-policy-comparison-opa-gatekeeper-vs-kyverno/

Summary

Hardening Pods' runtime configuration has a new and friendlier model

Pod-Security-Standards with an Admission Controller replace the PSP and RBAC

Suggesting few enhancements to the current security model

Declarative policy-engines and purpose build admission-controllers can be used to implement the suggested model

Feedback is welcomed https://bit.ly/fwdcs21-shuper

@arielshuper


CISCO