**PSP USECASE FROM YOUTUBE VIDEO**

**Agenda :**

1. PSP intro and basic concepts
2. Demo : enable psp + basic example
3. RBAC for PSPS
4. Full demo with RBAC
5. Simplify PSP adoption in production
6. Advanced details

**What is PSP ?**

* It’s a cluster level resource that controls security sensitive aspects of pod specification.
* PSP’s define a set of conditions & restrictions that will be enforced over your pods spec & run time capabilities.
* PSP’s need to explicitly be enabled in the k8s API server manifest

--enable-admission-plugins = PodSecurityPolicy

* Cloud providers usually have an easier way.
* PSP objects are cluster wide resources they don’t belong to any particular namespace.
* Kubernetes RBAC will link the PSPs with service accounts declared by pods.

**Quick examples :**

I don’t want developers to spawn privileged pods.

Web-frontend pods must have ReadOnlyRootFile system.

Pods should not be executed as a root user.

**Do I Really need PSP’s ?**

Yes, otherwise a Kubernetes user or service can spawn over privileged pods. This is what the malicious actor will leverage for container breakout ( breakout of that container & use other pods in the node) and spreading.

If there are no restrictions to the security context in the pod spec, the attacker can do anything the docker command could : running a privileged container, using node resources etc.

**DEMO : psp’s not enabled**

*deploy-dangerous-pod.yaml*

apiVersion : apps/v1

kind : Deployment

metadata :

labels:

app : dangerous-deploy

name : dangerous-deploy

spec:

replicas :1

selector :

match labels:

app : dangerous-deploy

template :

metadata:

labels:

app:dangerous-deploy

spec:

containers :

* image : alphine

name: alphine

stdin: true

tty: true

securitycontext:

privileged:true

hostPID : true

hostNetwork : true

apply this yaml file to create pod :

**kubectl apply –f deploy-dangerous-pod.yaml**

**kubectl get po -** (you can see your pod running)

**kubectl attach –ti deploy/dangerous-deploy** -(you can attach to the running container)

Now, you are inside your container type **ls**  you can see all the directories present.

**ps :** we can see all the processes running on the host in other pods (namespace, processId name space of the host)

**ifconfig :** The “ifconfig” command is used for displaying current network configuration information.

We are also sharing network namespace so we can run a privilege command called “**nsenter”**

**nsenter :** enter the mount namespace of the host. Right now I am the root user, I can run “**docker ps”** command and do anything to the containers running there.

**Hardening K8S security with PSPs**

Understanding and implementing PSPs in your cluster will help you follow the golden principle of least privilege.

PSPs will enable granular control of the clearance level for the different workloads in your environment :

* Restrictive PSP by default
* Additional permissions for privileged workloads only

( granular control means giving control to only those processes that needs it & to the privileged ones)

**This is how a psp file looks like**

apiVersion: policy/v1beta1

kind : PodSecurityPolicy

metadata :

name : example

spec:

allowedCapabilities :

* NET\_ADMIN
* IPC\_LOCK

allowedHostPaths :

* pathPrefix : /dev
* pathPrefix : /run
* pathPrefix : /

fsGroup :

rule : RunAsAny

hostNetwork : true

privileged : true

runAsUser:

rule: 'RunAsAny'

seLinux:

rule: 'RunAsAny'

supplementalGroups:

rule: 'RunAsAny'

volumes :

* hostPath
* secret

This PSP in particular is very permissive :

* Additional linux capabilities : NET\_ADMIN & IPC\_LOCK
* Mounting host paths is allowed
* Can attach to the host network
* Can run as privileged

**AllowedHostPaths :**

Your container is running on a system. Now, this system has no way of communicating with the container as according to the container concepts, its independent wrt resources. This makes a need to set paths to communicating with the system. The host path is (generally) the shares that you want to give the container access to.

**PSP control aspects**

There are many control aspects or policies that can be defined using PSP :

* Prevent privileged pods from starting and control privilege escalation :
* Privileged : false
* allowPrivilegedEscalation : false
* Restrict access to the host namespaces ,network , file system the pod can access :
* hostNetwork: false
* hostIPC: false
* hostPID: false
* Restrict the users/groups a pod can run as
* runAsUser :

rule: 'MustRunAsNonRoot'

* supplementalGroups:

rule: 'MustRunAs'

ranges:

# Forbid adding the root group.

- min: 1

max: 65535

* fsGroup:

rule: 'MustRunAs'

ranges:

# Forbid adding the root group.

- min: 1

max: 65535

* Limiting the volumes a pod can access
* Volumes ( default types )
* Restrict other parameters like read-only root file systems.
* readOnlyRootFilesystem: false

**PSPs and admission controllers**

PSPs are implemented as an admission controller in the Kubernetes API.

This has several implications :

* Pods do not meet the requirements will be rejected before creation.
* Once you enable PSP every pod need to be explicitly allowed.
* Different service accounts need different PSPs like either permissive or restrictive. How we enable it is by using specific RBAC permissions .

**DEMO : enable PSP + basic example**

In aws eks we don’t have to enable psp it is enabled by default from Kubernetes version 1.13

For suppose by default a restrictive PSP is enabled then ,

***non-privileged-pod.yaml***

apiVersion : apps/v1

kind : Deployment

metadata :

labels:

app : non-privileged-deploy

name : non-privileged-deploy

spec:

replicas :1

selector :

match labels:

app : non-privileged-deploy

template :

metadata:

labels:

app: non-privileged-deploy

spec:

containers :

* image : alphine

name: alphine

stdin: true

tty: true

securitycontext:

runAsUser : 1000

runAsGroup : 1000

if you create the deployment the existing PSP will allow the pod to run.

***privileged-pod.yaml***

apiVersion : apps/v1

kind : Deployment

metadata :

labels:

app : privileged-pod-deploy

name : privileged-pod -deploy

spec:

replicas :1

selector :

match labels:

app : privileged-pod -deploy

template :

metadata:

labels:

app: privileged-pod -deploy

spec:

containers :

* image : alphine

name: alphine

stdin: true

tty: true

securitycontext:

privileged:true

hostPID : true

hostNetwork : true

if we try to deploy this in the output you can see the replica set is not available and the pod is not created. If you run the **kubectl describe** command then you can see an **Error** which mentions that this **deployment is not able to be validated by any pod security policy. Because in the pod security policy that is applied is not allowed to have privileged container or use hostpid, hostnetwork.**

**RBAC for PSPs :**

Here we can see how we associate a pod to a specific PSP :

One of the major roadblocks for ops team while implementing PSP is RBAC

* PSPs are cluster wide resources that are not by default associated to any workload.
* After creating a PSP the next thing you need to create is a Role ( or cluster Role ) which grants “use” access to this PSP.

**Role :**

**apiVersion**: rbac.authorization.k8s.io/v1

**kind**: Role

**metadata**:

**name**: exampleRole

**rules**:

- **apiGroups**: ['policy']

**resources**: ['podsecuritypolicies']

**verbs**: ['use']

**resourceNames**:

- examplePSPpolicy

After creating a role you need to assign it to “who will be allowed to use this PSP” by creating role binding.

**RoleBinding :**

**apiVersion**: rbac.authorization.k8s.io/v1

**kind**: RoleBinding

**metadata**:

**name**: example-rolebinding

**roleRef**:

**kind**: Role

**name**: exampleRole

*apiGroup: rbac.authorization.k8s.io*

**subjects**:

*# Authorize specific service accounts:*

- **kind**: ServiceAccount

**name**: web-frontend-serviceaccount

*namespace: web-frontend*

for default namespace PSP is enabled but if we try to use a different namespace and try to deploy the same previous ***non-privileged-pod.yaml*** then you can see there will be no replica-set the pod will not be created.

If you describe your deployment then there will be an **error : unable to validate against any pod security policy**

Creating cluster role and cluster role binding :

***Clusterrole-use-restricted.yaml***

**apiVersion**: rbac.authorization.k8s.io/v1

**kind**: ClusterRole

**metadata**:

**name**: use-restricted-psp

**rules**:

- **apiGroups**: ['policy']

**resources**: ['podsecuritypolicies']

**verbs**: ['use']

**resourceNames**:

- restricted-psp

---

**apiVersion**: rbac.authorization.k8s.io/v1

**kind**: ClusterRoleBinding

**metadata**:

**name**: restricted-role-bind

**roleRef**:

**kind**: ClusterRole

**name**: use-restricted-psp

*apiGroup: rbac.authorization.k8s.io*

**subjects**:

*# Authorize all service accounts:*

- **kind**: Group

**name**: system:serviceaccounts

*apiGroup: rbac.authorization.k8s.io*

In the above rbac we are allowing any service account to use this PSP by default. This psp will allow any pod with no additional privileges to be deployed on to the cluster.

***restricted-psp.yaml***

apiVersion: policy/v1beta1

kind: PodSecurityPolicy

metadata:

name: restricted-psp

annotations:

serviceaccount.cluster.cattle.io/pod-security:restricted

serviceaccount.cluster.cattle.io/pod-security-version: “2000”

creationTimestamp: “202-03-10t17:22:23Z”

labels:

cattle.io/creator : norman

name:restricted-psp

resourceVersion : “11930”

selflink : /apis/policy/v1beta1/PodSecurityPolicies/restricted-psp

spec:

privileged : false

# Required to prevent escalations to root.

allowPrivilegeEscalation: false

requiredDropCapabilities:

- ALL

# Allow core volume types.

volumes:

- 'configMap'

- 'emptyDir'

- 'projected'

- 'secret'

- 'downwardAPI'

- 'persistentVolumeClaim'

hostNetwork: false

hostIPC: false

hostPID: false

seLinux:

rule: 'RunAsAny'

runAsUser:

# Require the container to run without root privileges.

rule: 'MustRunAsNonRoot'

supplementalGroups:

rule: 'MustRunAs'

ranges:

# Forbid adding the root group.

- min: 1

max: 65535

fsGroup:

rule: 'MustRunAs'

ranges:

# Forbid adding the root group.

- min: 1

max: 65535

readOnlyRootFilesystem: false

apply the cluster role using **kubectl apply –f clusterrole-use-restricted.yaml**

redeploy your pod now you can your un-privileged-pod will be created without any error.

Now, if you want to use a different policy and create a privileged pod then create a cluster role which uses the privileged PSP and then assign it to the service account and the namespace in which you want to create a privileged pod.

***Privileged-psp.yaml***

apiVersion: policy/v1beta1

kind: PodSecurityPolicy

metadata:

name: privileged

annotations:

serviceaccount.cluster.cattle.io/pod-security:restricted

serviceaccount.cluster.cattle.io/pod-security-version: “2000”

creationTimestamp: “202-03-10t17:22:23Z”

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'

**Cluster Role and cluster role binding :**

**apiVersion**: rbac.authorization.k8s.io/v1

**kind**: ClusterRole

**metadata**:

**name**: use-privileged-psp

**rules**:

- **apiGroups**: ['policy']

**resources**: ['podsecuritypolicies']

**verbs**: ['use']

**resourceNames**:

- privileged-psp

---

**apiVersion**: rbac.authorization.k8s.io/v1

**kind**: ClusterRoleBinding

**metadata**:

**name**: privileged-cr

**namespace** : psp-test

**roleRef**:

**kind**: ClusterRole

**name**: use-privileged-psp

*apiGroup: rbac.authorization.k8s.io*

**subjects**:

- **kind**: ServiceAccount

**name**: privileged-sa

apply this and then see by creating the pod this time the pod will be created.

**Simplify PSP adoption in production :**

PSPs are enabled cluster-wide as an admission controller :

* Any configuration issue will prevent you from running new pods
* Any SA that is not explicitly allowed will be forbidden
* You need to design and deploy the psp before you even enable the feature.

How to generate the perfect(least privilege) PSP :

* Too restrictive – pods won’t even run
* Too permissive – missing the point of using PSPs

**Sysdig and Rancher can help in creating PSPs :**

* Rancher manages PSP and also automates creation of related RBAC artifacts.
* Sysdig Secure can automatically generate the least privilege PSP from the workload definition provided.
* Validate policies prior to enforcement(dry run)

Combining actual work load behavior with the PSP definition to confirm if the PSP, effectively applied ,would break the current requirements.

**Sysdig :**

In the sysdig api we need to upload the deployment yaml then it will automatically create the psp for it we can then click on run to check if our psp is compliant.

Sysdig can also validate our image for any vulnerabilities.