
KubeCon
CloudNativeCon
North America 2022
BUILDING FOR THE ROAD AHEAD
DETROIT 2022
TiKV

BUILDING FOR THE ROAD AHEAD

DETROIT 2022

Using eBPF Superpowers to generate Kubernetes Security Policies

Mauricio Vásquez-Bernal & Alban Crequy

Agenda

- K8s security policies
- Generating security policies
- Auditing security policies
- Limitations

K8s Security Policies

- K8s offers different kind of policies to secure the clusters
  - Seccomp
  - Pod Capabilities
  - Network Policies
- Configuring those policies requires knowledge about the application
  - Which system calls should it perform?
  - Which Linux capabilities does it need?
  - Which network endpoints does it communicate with?
- Most of the times the application developer is not the one configuring those

Generating Security Policies

- Observe the application and use this data to generate policies
  1. Observe the application
     - Technologies like eBPF allow to capture the activity of an application
       - Low overhead
       - Great flexibility
  2. Generate a policy
     - Depending on the policy kind this is easy or very difficult

BUILDING FOR THE ROAD AHEAD

DETROIT 2022

Seccomp

Secure Computing

Seccomp

- Linux kernel mechanism to limit the system calls a process can make
- A seccomp profile defines a list of syscalls and an action for them:
  - KILL: Kills the calling process
  - TRAP: Sends SIGSYS to the calling thread
  - ERRNO: Returns an error without executing the syscall
  - LOG: Logs and execute the syscall
- https://man7.org/linux/man-pages/man2/seccomp.2.html

Seccomp Profile


```
{
    "defaultAction": "SCMP_ACT_ERRNO",
    "syscalls": [
        {
            "names": [
                "futex",
                [...],
                "recvfrom",
            ],
            "action": "SCMP_ACT_ALLOW"
        }
    ]
}
```

Seccomp in Kubernetes

- Security profile defined in the pod security context (k8s >= 1.19)
  - Can be defined at pod or container level

```
spec:
  securityContext:
    seccompProfile:
      type: Localhost
      localhostProfile: myprofile.json
```

File under `/var/lib/kubelet/seccomp` on the node.

https://kubernetes.io/docs/tutorials/security/seccomp/

Security Profiles Operator (SPO)

- Tool to handle security profiles in Kubernetes
- Provides SeccompProfile Custom Resource
  - Saves profile on nodes to be referenced in pod spec

```
apiVersion: security-profiles-operator.x-
k8s.io/v1beta1
kind: SeccompProfile
metadata:
  namespace: ns
  name: profile1
spec:
  defaultAction: SCMP_ACT_LOG
```

```
apiVersion: v1
kind: Pod
metadata:
  name: test-pod
spec:
  securityContext:
    seccompProfile:
      type: Localhost
      localhostProfile: operator/ns/profile1.json
  containers:
    - name: test-container
      image: nginx
```

Defining Seccomp Profiles

- An allow-list is preferred
  - More secure
  - Must include all system calls needed by the application
- How to know which syscalls are needed my application?
  - Analyze the code
  - Use strace
  - Define a profile with SCMP_ACT_LOG and check the audit log
  - **Use a tool that records the syscalls performed in a container**

Security Profiles Operator eBPF Recorder

Demo

Inspektor Gadget seccomp advisor

Demo

Auditing Seccomp Policies

- How to know if a given security profile will break my application?
- Use "`defaultAction: SCMP_ACT_LOG`"
  - Check syslog
  - Problem: Difficult to understand
- Alternative: "seccomp audit" gadget

Inspektor Gadget seccomp audit

Demo

Linux Capabilities

Linux Capabilities

- Since Linux 2.2 privileges are divided into units
  - It's possible to give a process only some privileges
- Those privileges are called capabilities:
  - CAP_CHOWN: change file UIDs and GIDs
  - CAP_NET_BIND_SERVICE: bind ports < 1024
  - CAP_SYS_BOOT: call reboot()
  - Many more: https://man7.org/linux/man-pages/man7/capabilities.7.html

Linux Capabilities in Kubernetes

- The container runtime grants some capabilities by default
- Others can be added / dropped by using the container securityContext

```
securityContext:
  capabilities:
    drop:
      - all
    add: ["NET_ADMIN", "SYS_TIME"]
```

Generating Capabilities Security Context

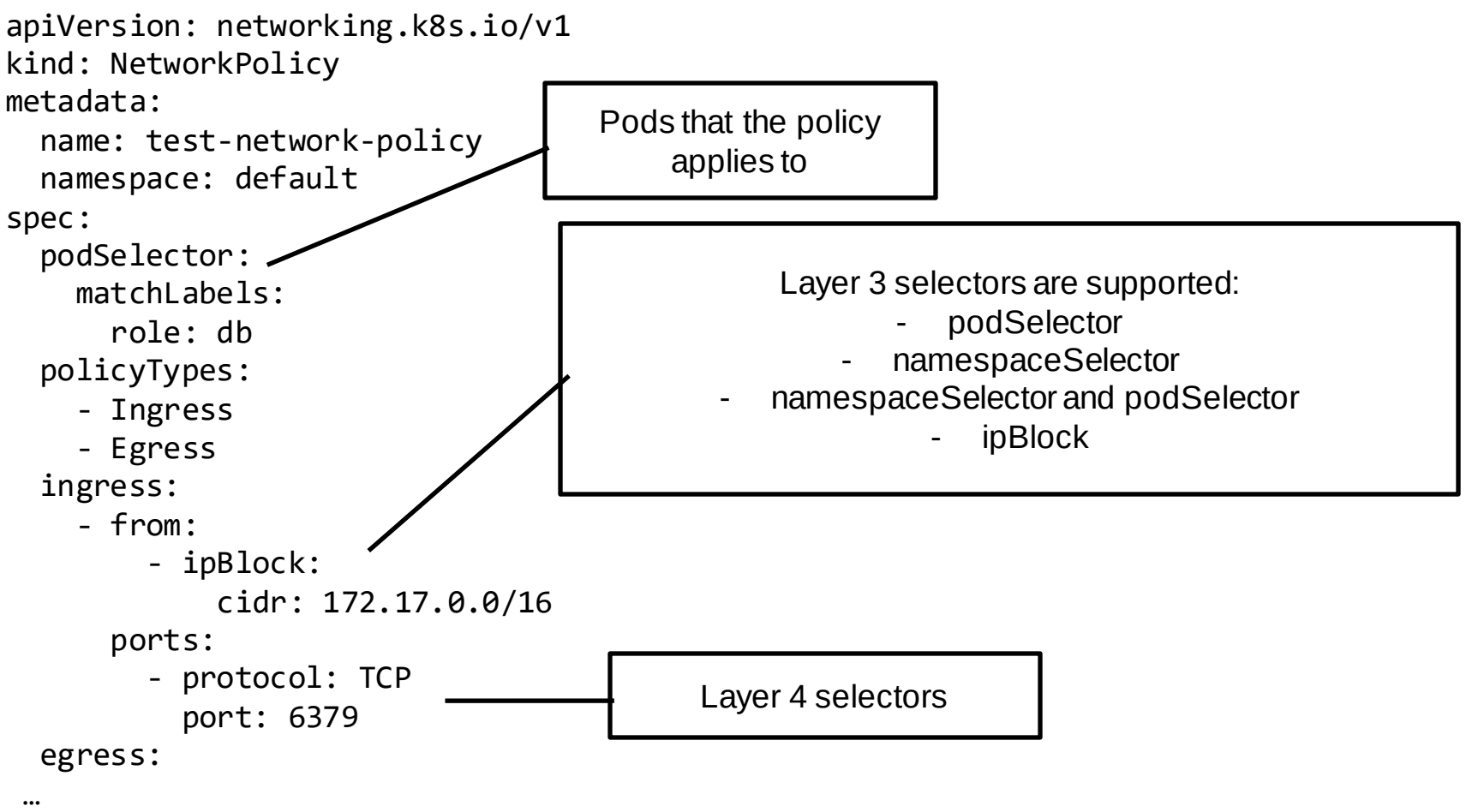
Demo

Network Policies

Network Policies

- Kubernetes mechanism to control how a pod can communicate with network “entities”
- Operate at IP or port level (Layer 3 or 4)
- Can be used for inbound (**Ingress**) or outbound (**Egress**) traffic
- https://kubernetes.io/docs/concepts/services-networking/network-policies/

The NetworkPolicy Resource



Defining Network Policies

- Network policies are usually created when defining the architecture of a solution
- In some cases, those are defined after the solution is deployed
- For those, using a policy advisor can be an option

Generating Network Policies

Demo

Auditing Network Policies

- Some tools allows to check when a network policy blocks traffic
- Cilium supports `auditMode: true`
  - https://docs.cilium.io/en/v1.8/gettingstarted/policy-creation/
- Other tools can determine the iptables rule that blocked a packet
  - https://github.com/box/kube-iptables-tailer

Limitations

- The application must generate all possible events while observing it
  - If an event is not generated, then it'll be blocked by the policy
- This technique assumes the application is trusted when observing
  - It's not compromised, and no malicious activity is generated while observing it

Similar Talk

- You and Your Security Profiles; Generating Security Policies with the Help of eBPF - John Fastabend & Natalia Reka Ivanko, Isovalent

BUILDING FOR THE ROAD AHEAD

DETROIT 2022

Thanks

KubeCon | CloudNativeCon

BUILDING FOR THE ROAD AHEAD

DETROIT 2022

Please scan the QR Code above to leave feedback on this session