ISOVALENT
now part of CISCO

From Observability to Enforcement:

Lessons Learned Implementing eBPF Runtime Security

KubeCon | CloudNativeCon
North America 2024

Intro

ISOVALENT
now part of CISCO

Goal: move from Observability to Enforcement

Observability: inspect certain interactions between workloads and the OS.

Examples:

- Alert whenever pods attempt to load a kernel module.
- Alert whenever processes spawned with `kubectl exec` accesses `/my/extremely/secret/file` in certain pods.

Enforcement: disallow certain interactions between workloads and the OS.

Examples:

- Disallow pods from loading a kernel module.
- Disallow processes spawned with `kubectl exec` from accessing `/my/extremely/secret/file` in certain pods.

Outline

- eBPF for observability and enforcement
- Building a Tetragon enforcement policy
- Bringing enforcement to the Kubernetes world
- Examples and resources

Observability

Alert if an app other than A1 opens file X

A2 opened X

App A1

App A2

...

Copying and processing all events in user-space wastes memory and CPU

observer

user

kernel

open("x") open("y") open("x") open("z")

All file accesses from all apps

The kernel (privileged) part is simple

ISOVALENT
now part of CISCO

Programmable and **performant** in-kernel "virtual machine" that **safely** executes native code on certain events/hooks (analogy: "JavaScript for the kernel").

In-kernel filtering with eBPF

Block any app other than A1 from opening file X

A2 opened X

App A1

App A2

Once the filtering logic is in-line with the operation, we can go beyond observability

observer

user

kernel

open("x") open("y") open("x") open("z")

ISOVALENT
now part of CISCO

Enforcement with eBPF

- Filtering logic remains the same
- Define enforcement actions
  - Override (block) the call before it happens
  - Send a signal (e.g., SIGKILL)

Enforcement is different from observability:

- Pushing logic into eBPF is a **correctness** not an efficiency concern
  - For observability you can still apply some filtering in user-space and potentially delay the decision on whether an event is filtered or not

Open Source

- Apache 2.0 (userspace) & GNU GPL (eBPF)
- Part of CNCF as a subproject of Cilium

eBPF-based

- Generic low level process events
- In-kernel filtering and enforcement

Kubernetes-native

- Configuration via custom resources
- Kubernetes metadata in events

tetragon

ISOVALENT
now part of CISCO


tetragon
Metrics
Events
SIEM
fluentd
Logs
Traces
JSON
Tetragon Agent
Pod
app.py
Func Calls
Pod
GO
app.go
Code Exec
Linux Kernel
Kernel Runtime
tetragon
Smart Collector
Stack Traces
Ring Buffer
Metrics
Hash Maps
Process Execution
Syscall Activity
System Calls
File Access
VFS
Seq Attack
TCP/IP
NS Escapes
Priv Escalations
Namespaces
Data Access
Storage
HTTP, DNS, TLS
eth0
Network

Deep Visibility, Many Use Cases

- Process Execution & Syscalls
- Network Connections
- Disk & File Access
- Linux namespace changes
- Linux capabilities & privilege changes

ISOVALENT
now part of CISCO

Reliability & Low Overhead

Reliability

- Safe to run thanks to eBPF verifier
- Complete visibility
- In-kernel enforcement

Low Overhead

- eBPF programs are generally fast
- No user-kernel context switches
- In-kernel filtering
- In-kernel aggregation

ISOVALENT
now part of CISCO

Portable & Transparent

Tetragon Agent

- Running on any Linux system
- Kubernetes: daemonset
- Bare metal: container or binary

User Applications

- Any language, any framework
- No changes to code or config
- No sidecars, no restarts

ISOVALENT
now part of CISCO

Integrations and data pipelines

- SIEM systems, e.g. **Splunk, Elastic**
- Observability tools for metrics and logs, e.g. **Prometheus, Grafana, OpenTelemetry**
- DYI, e.g. **S3, ClickHouse**
- Varying maturity stages, but possibilities are endless

ISOVALENT
now part of CISCO

Let’s build a Tetragon enforcement policy

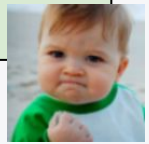
Block access to /my/extremely/secret/file

```yaml
apiVersion: cilium.io/v1alpha1
kind: TracingPolicy
metadata:
  name: block-secret-access
spec:
  kprobes:
  - call: sys_openat
    syscall: true
    selectors:
    - matchActions:
      - action: Sigkill
```

hook point - syscall

action - send signal

```
$ cat /my/extremely/secret/file
Killed!
```



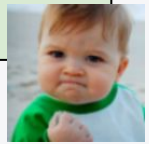
```
$ cat /etc/hostname
Killed!
```

Block access **only** to /my/extremely/secret/file

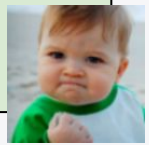
```yaml
apiVersion: cilium.io/v1alpha1
kind: TracingPolicy
metadata:
  name: block-secret-access
spec:
  kprobes:
  - call: sys_openat
    syscall: true
    args:
    - index: 1
      type: string
    selectors:
    - matchArgs:
      - index: 1
        operator: Equal
        values:
        - /my/extremely/secret/file
      matchActions:
      - action: Sigkill
```

filter by file path - arg passed to the syscall

```
$ cat /my/extremely/secret/file
Killed!
```



```
$ cat /etc/hostname
amazing-vm
```



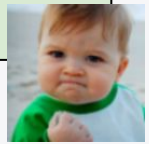
```
$ /my/cat /my/extremely/secret/file
Killed!
```

Block access to /my/extremely/secret/file except from /my/cat

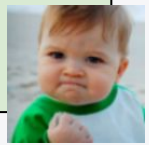
```yaml
apiVersion: cilium.io/v1alpha1
kind: TracingPolicy
metadata:
  name: block-secret-access
spec:
  kprobes:
  - call: sys_openat
    syscall: true
    args:
    - index: 1
      type: string
    selectors:
    - matchArgs:
      - index: 1
        operator: Equ
        values:
        - /my/extremely/ret/file
      matchBinaries:
      - operator: NotIn
        values:
        - /my/cat
      matchActions:
      - action: Sigkill
```

filter by binary that called the syscall

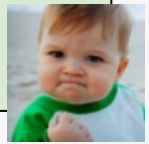
```
$ cat /my/extremely/secret/file
Killed!
```



```
$ cat /etc/hostname
amazing-vm
```



```
$ /my/cat /my/extremely/secret/file
I love pizza!
```



Problem 1: hook point

```yaml
apiVersion: cilium.io/v1alpha1
kind: TracingPolicy
metadata:
  name: block-secret-access
spec:
  kprobes:
  - call: sys_openat
    syscall: true
    args:
    - index: 1
      type: string
    selectors:
    - matchArgs:
      - index: 1
        operator: Equal
        values:
        - /my/extremely/secret/file
      matchBinaries:
       - operator: NotIn
         values:
         - /my/cat
      matchActions:
      - action: Sigkill
```

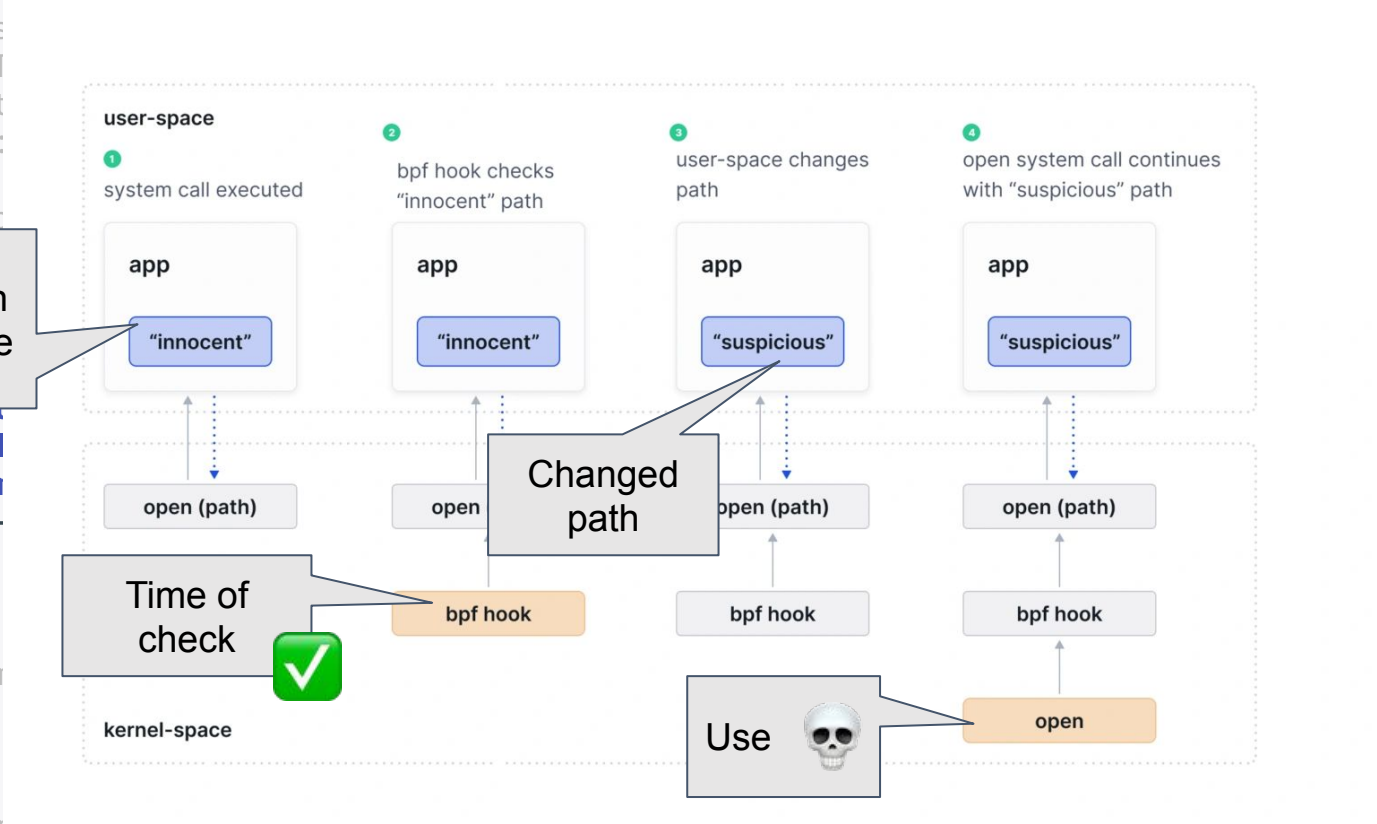
there are different system calls to open a file

```
$ ./evil-cat /my/extremely/secret/file
I love pizza!
```

ISOVALENT
now part of cisco

Problem 2: filtering by file path

```
apiVers
kind: T
metadat
  name:
spec:
  kprob
```




```
  sel
  - m
```

```
  matchActions:
  - action: Sigkill
```

SOVALENT
now part of CISCO

Problem 2: filtering by file path

```yaml
apiVersion: cilium.io/v1alpha1
kind: TracingPolicy
metadata:
  name: block-secret-access
spec:
  kprobes:
  - call: sys_openat
    syscall: true
    args:
    - index: 1
      type: string
    selectors:
    - matchArgs:
      - index: 1
        operator: Equal
        values:
        - /my/extremely/secret/file
      matchBinaries:
       - operator: NotIn
         values:
         - /my/cat
      matchActions:
      - action: Sigkill
```

there are different system calls to open a file

vulnerable to TOCTOU

access path might be different

```
$ ./evil-cat /my/extremely/secret/file
I love pizza!
```

```
$ cd /my
$ cat extremely/secret/file
I love pizza!
```

ISOVALENT
now part of CISCO

Problem 3: action

```yaml
apiVersion: cilium.io/v1alpha1
kind: TracingPolicy
metadata:
  name: block-secret-access
spec:
  kprobes:
  - call: sys_openat
    syscall: true
    args:
    - index: 1
      type: string
    selectors:
    - matchArgs:
      - index: 1
        operator: Equal
        values:
        - /my/extremely/secret/file
      matchBinaries:
       - operator: NotIn
         values:
         - /my/cat
      matchActions:
        - action: Sigkill
```

there are different system calls to open a file

vulnerable to TOCTOU

access path might be different

killing the process might be not enough

```
$ ./evil-cat /my/extremely/secret/file
I love pizza!
```

```
$ cd /my
$ cat extremely/secret/file
I love pizza!
```

ISOVALENT
now part of cisco

security_ hooks for Linux Security Modules (LSM)

overwriting might be unsupported

```
long sys_openat(
    int dfd,
    const char __user *filename,
    int flags,
    umode_t mode
)
```

pointer to user memory!

```
int security_file_open(
    struct file *file
)
```

ISOVALENT
now part of CISCO

Secure enforcement policy

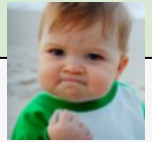
```yaml
apiVersion: cilium.io/v1alpha1
kind: TracingPolicy
metadata:
  name: block-secret-access
spec:
  kprobes:
  - call: security_file_open
    syscall: false
    args:
    - index: 0
      type: file
    selectors:
    - matchArgs:
      - index: 0
        operator: Equal
        values:
        - /my/extremely/secret/file
      matchBinaries:
      - operator: NotIn
        values:
        - /my/cat
      matchActions:
      - action: Override
        argError: -1
      - action: Sigkill
```

hook point - LSM

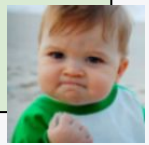
use file struct for filtering

override the return value in addition to sigkill

```
$ /evil/cat /my/extremely/secret/file
Killed!
```



```
$ cd /my
$ cat extremely/secret/file
Killed!
```



ISOVALENT
now part of CISCO

How do we apply these ideas in the Kubernetes world?

Filtering by Kubernetes identities: applying policies to workloads selectively

Namespaced policies

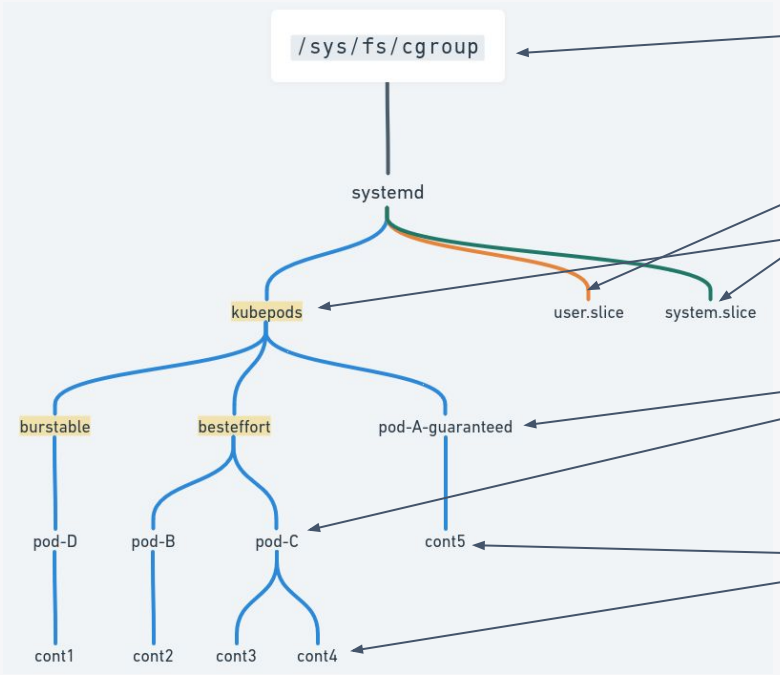
```yaml
apiVersion: cilium.io/v1alpha1
kind: TracingPolicyNamespaced
metadata:
  name: block-secret-access
  namespace: vendor-app
spec:
  ...
```

Label filters

```yaml
apiVersion: cilium.io/v1alpha1
kind: TracingPolicy
metadata:
  name: block-secret-access
spec:
  podSelector:
    matchLabels:
      app: nginx
  ...
```

Filtering needs to happen at eBPF

Bridging k8s and the kernel with cgroups



root

systemd workloads

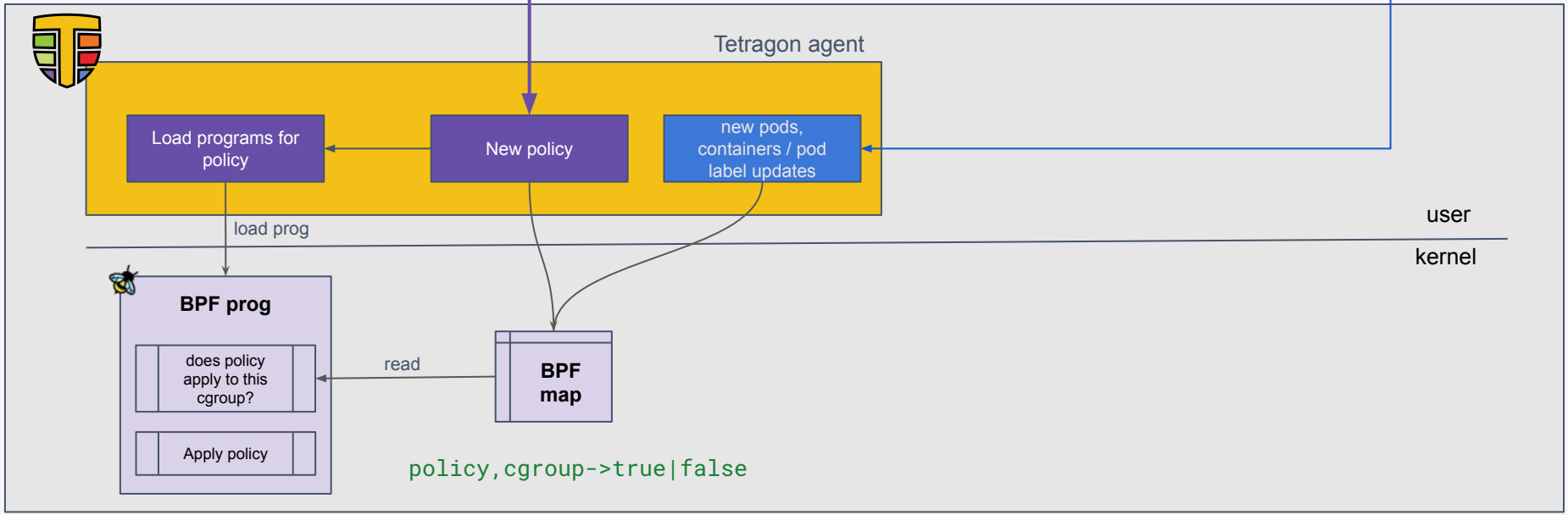
K8s workloads

One cgroup per pod

One cgroup per container

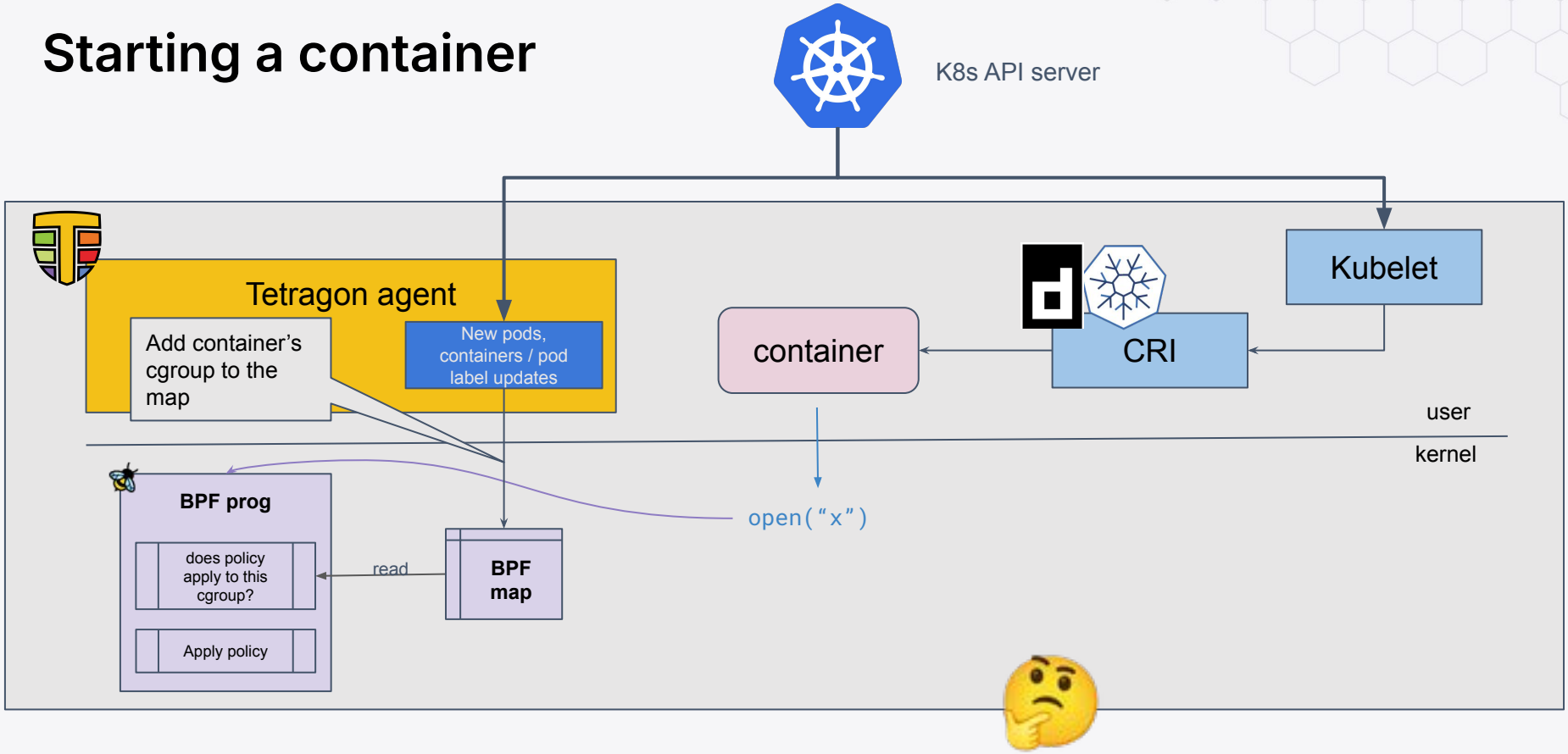
ISOVALENT
now part of CISCO

Tetragon workflow



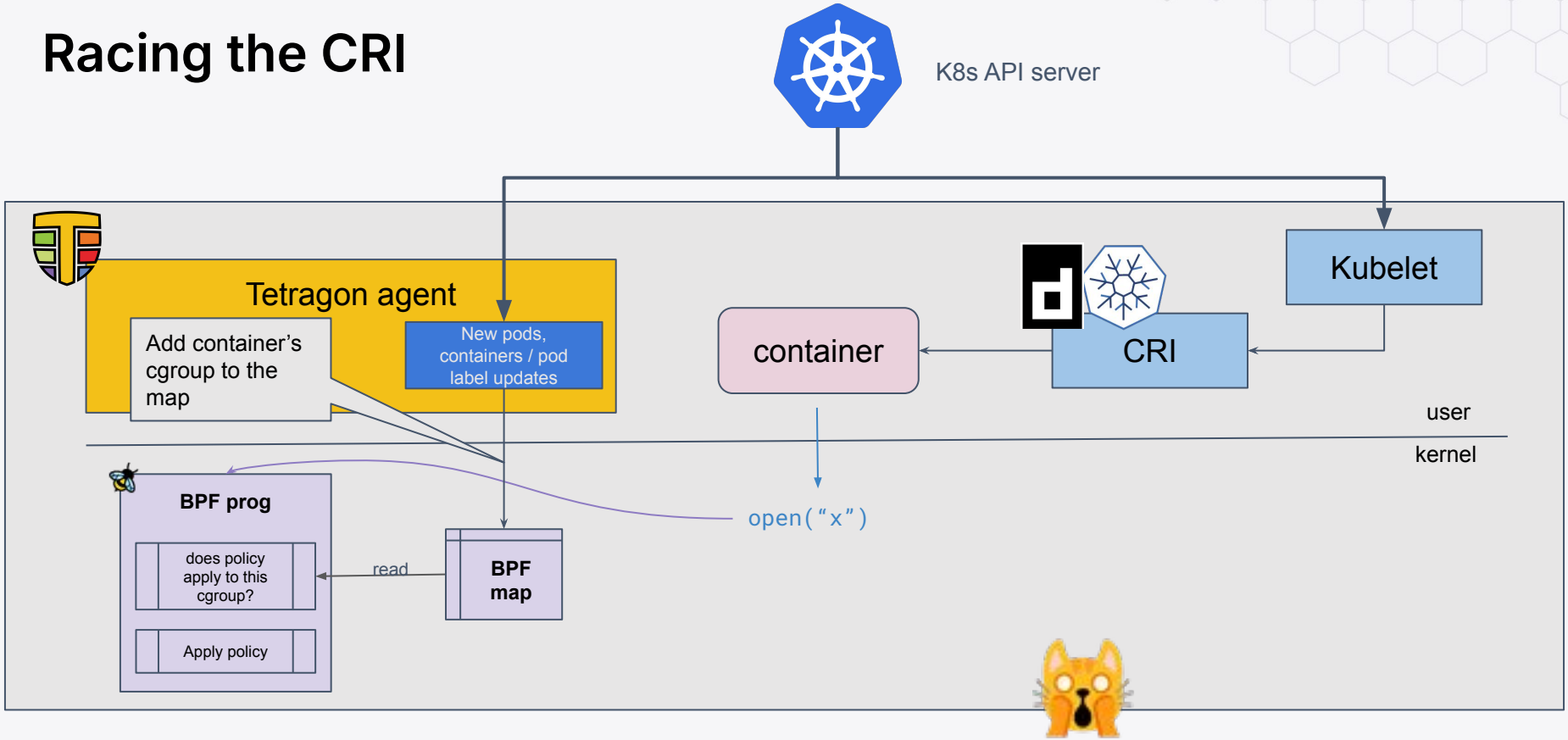
ISOVALENT
now part of CISCO

Starting a container



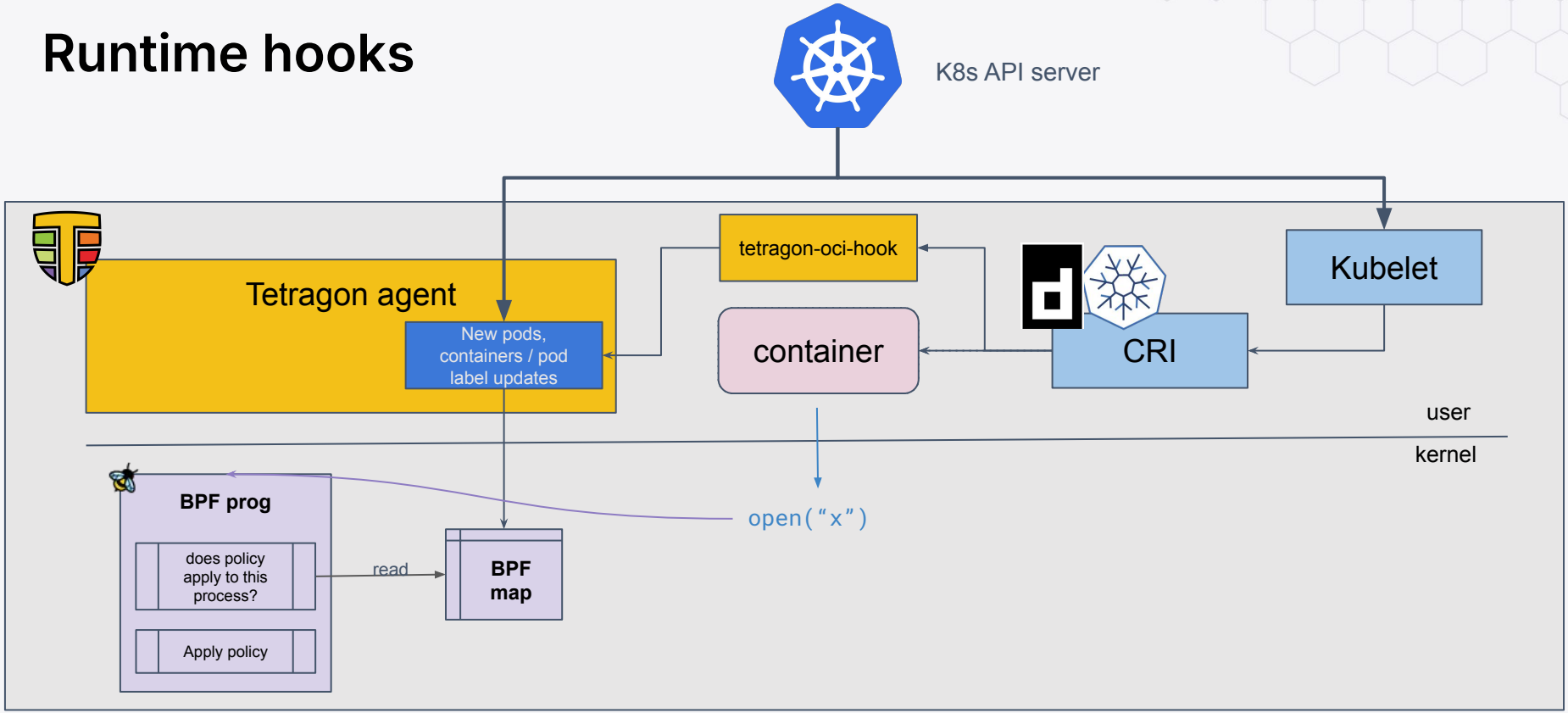
ISOVALENT
now part of CISCO

Racing the CRI



ISOVALENT
now part of CISCO

Runtime hooks



ISOVALENT
now part of cisco

Tetragon runtime hooks

- A different daemonset (`tetragon-rthooks`) implements the necessary functionality for hooking into the CRI

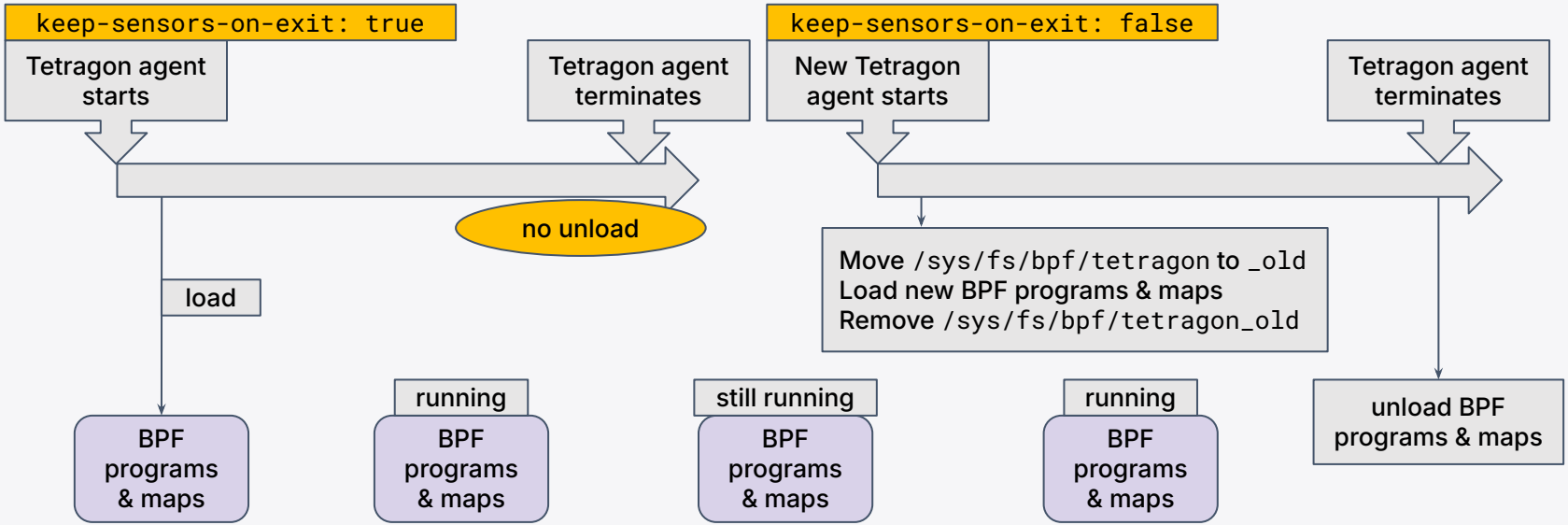
- Interfaces:
  - oci-hooks (cri-o)
  - NRI (Node Resource Interface) (containerd)
    - Enabled in containerd 2.0 by default

What if the agent goes down?

ISOVALENT
now part of CISCO

What if the agent goes down?

Enable persistent enforcement: **`keep-sensors-on-exit`**



tetragon.io/docs/concepts/enforcement/persistent-enforcement

ISOVALENT
now part of CISCO

How to get started with Tetragon enforcement policies?

Use label filters in policy

```yaml
apiVersion: cilium.io/v1alpha1
kind: TracingPolicy
metadata:
  name: block-secret-access-nginx
spec:
  podSelector:
    matchLabels:
      app: nginx
  kprobes:
  - call: security_file_open
    ...
    selectors:
    - matchPIDs:
      - operator: NotIn
        followForks: true
        isNamespacePID: true
        values:
        - 1
      matchArgs: ...
      matchActions: ...
```

k8s label filter - it's propagated to the kernel

allow access only from the pod - not e.g. kubectl exec

ISOVALENT
now part of CISCO

Another example: block loading kernel modules

overwrite (return error)

```
int security_kernel_module_request(
    char *kmod_name
)
int security_kernel_read_file(
    struct file *file,
    enum kernel_read_file_id id,
    bool contents
)
int security_kernel_load_data(
    enum kernel_load_data_id id,
    bool contents
)
```

filter READING_MODULE

ISOVALENT
now part of CISCO

Policy library

tetragon.io/docs/policy-library

examples/tracingpolicy in repo

tetragon / examples / tracingpolicy /

anfedotoff and kkourt Update LSM tracingPolicy examples

Name
..
cves
host-changes
process-credentials
process-exec
bpf.yaml
datagram-with-selectors.yaml
datagram-with-sock-tracking.yaml
datagram.yaml
datagram_518.yaml
dev_hard_start_xmit.yaml
dev_queue_xmit.yaml
dns-only-specified-servers.yaml

Documentation / Use Cases

Use Cases

This section presents various use cases on process, files, network and security monitoring enforcement.

By default, Tetragon monitors process lifecycle, learn more about that in the dedicated use cases.

For more advanced use cases, Tetragon can observe tracepoints and arbitrary kernel calls via kprobes. For th must be extended and configured with custom resources objects named TracingPolicy. It can then generates `process_tracepoint` and `process_kprobes` events.

Process lifecycle
Tetragon observes by default the process lifecycle via exec and exit

Filename access
Monitor filename access using kprobe hooks

Network observability
Monitor TCP connect using kprobe hooks

Linux process credentials
Monitor Linux process credentials

Host System Changes
Monitor Host System changes

Security Profiles
Observe and record security events

eBPF Subsystem Interactions

Description

Audit BPF program loads and BPFFS interactions

Use Case

Understanding BPF programs loaded in a cluster and interactions between applications and programs can identify bugs and malicious or unexpected BPF activity.

Policy

bpf.yaml

Example jq Filter

```
jq 'select(.process_kprobe != null) | select(.process_kprobe.function_name | test("bpf_check")) | "\(.time
```

Example Output

```
"2023-11-01T02:56:54.926403604Z /usr/bin/bpftool prog list programType:BPF_PROG_TYPE_SOCKET_FILTER program
```

ISOVALENT
now part of cisco

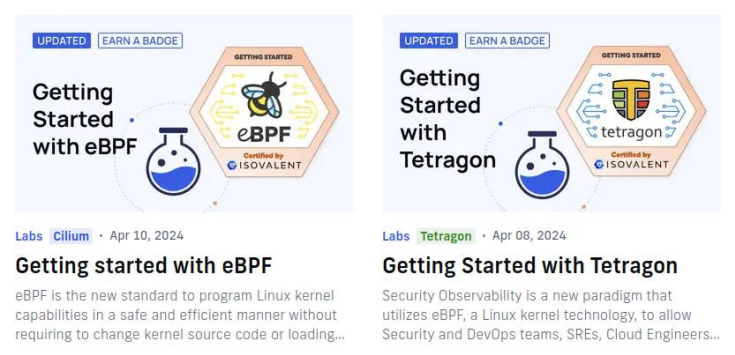
Learn more

Visit Isovalent booth for labs and book signings!

Tetragon

docs: tetragon.io 📝

repo: github.com/cilium/tetragon



eBPF

ebpf.io 📝

labs: isovalent.com/resource-library/labs

books: isovalent.com/resource-library/books

Contribfest 👩‍💻 👩‍💻

Tomorrow 4:30 - 6:00 PM

Room 355A

Come to contribute,

discuss, get help or learn!

🚨 Contribfest: Kickstart Your eBPF Journey with Tetragon

Thursday November 14, 2024 4:30pm - 6:00pm MST

Salt Palace | Level 3 | 355 A

Tetragon and eBPF have a lot of buzz and this is your chance to get involved diving into the bytecode or docs! Tetragon's docs are still young and your new contributor's perspective will be a superpower for spotting issues or unclear wording in the various quickstarts, guides, and concepts pages. The project's CLI, tetra, is another great opportunity for those interested in code contributions around ease of use, testing, and consistency in flags and output. Tetragon's documentation tech stack uses Markdown, built with Hugo, and a customized Docsy theme. The CLI is written in Go with the Cobra library and uses gRPC to communicate with the agent. While this session should help you get more familiar with Tetragon and can lead to more contributions in the future, those technologies are also used in Kubernetes and many other CNCF projects.

Stay in touch

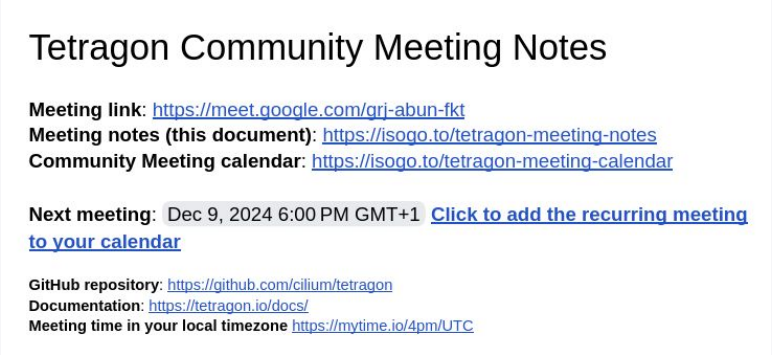
community meeting

Monthly on the second Monday
6:00 PM Europe time

Slack

#tetragon channel in Cilium & eBPF Slack
cilium.slack.com

More info: isogo.to/tetragon-meeting-notes
or tetragon.io, Github, Slack



Tetragon Community Meeting Notes

Meeting link: https://meet.google.com/grj-abun-fkt
Meeting notes (this document): https://isogo.to/tetragon-meeting-notes
Community Meeting calendar: https://isogo.to/tetragon-meeting-calendar

Next meeting: Dec 9, 2024 6:00 PM GMT+1 **Click to add the recurring meeting to your calendar**

GitHub repository: https://github.com/cilium/tetragon
Documentation: https://tetragon.io/docs/
Meeting time in your local timezone https://mytime.io/4pm/UTC