
KubeCon
CloudNativeCon
North America 2023

Survive eBPF with bpfd

Andrew Stoycos & Shane Utt

Andrew Stoycos
Senior SWE @ Red Hat
Bpfd Maintainer
Network Policy API Maintainer

Shane Utt
Staff SWE @ Kong
SIG Network Chair
Gateway API Maintainer

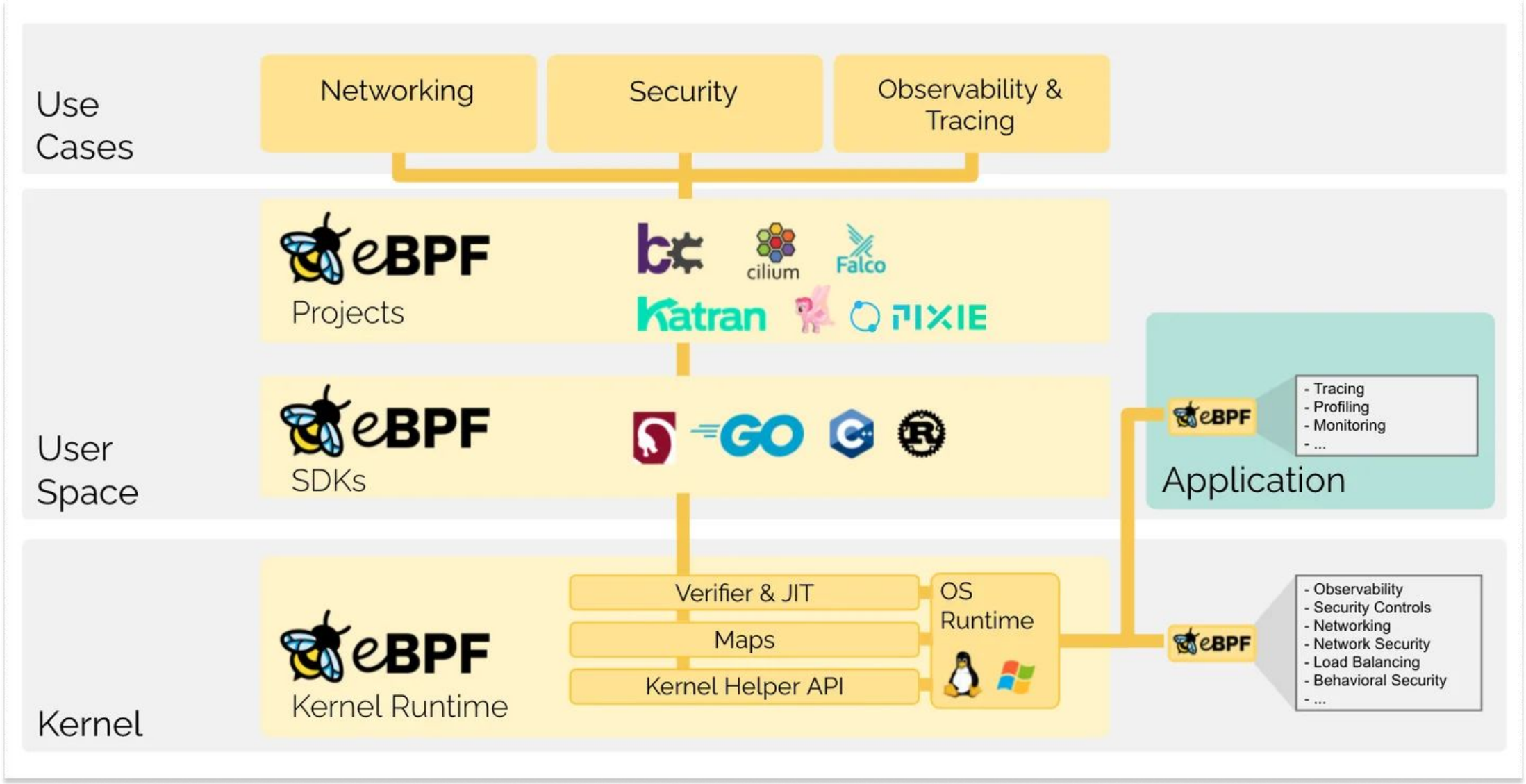
KubeCon | CloudNativeCon
North America 2023


KubeCon | CloudNativeCon

North America 2023

What is eBPF, and how does it relate to Kubernetes?

Quick introduction to eBPF



Why eBPF in Kubernetes?

- eBPF is a powerful general-purpose framework to extend the Linux Kernel
- Use cases from observability, networking, security and more
- Many existing examples of eBPF in kubernetes
  - Cilium and Calico CNIs
  - Pixie: Open source observability
  - KubeArmor: Container-aware Runtime Security Enforcement System
  - Blixt: Gateway Api L4 conformance implementation
  - NetObserv: Open Source Operator for network observability


KubeCon | CloudNativeCon

North America 2023

Issues with eBPF Programs in Kubernetes

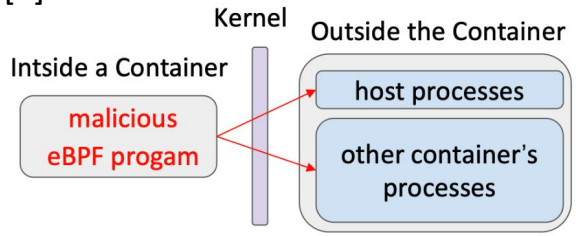
Security Issues with eBPF

- eBPF is *not* namespaced it can easily escape container isolation
- Relies on **highly privileged containers**
- Vulnerable to **supply chain attacks**

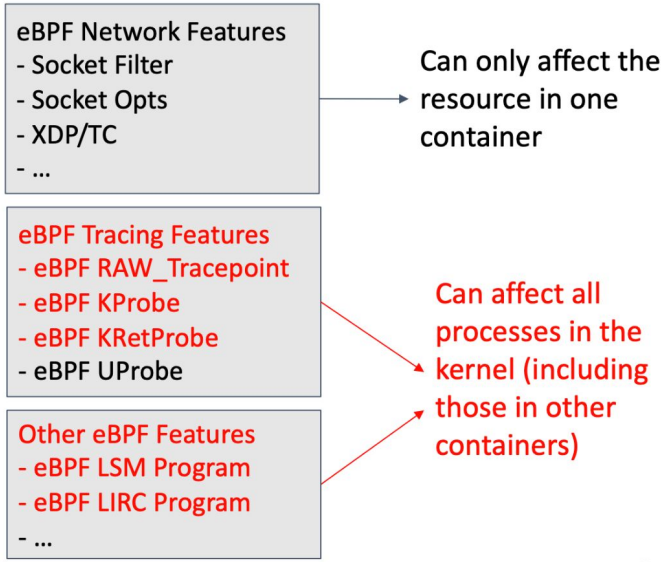
eBPF is not namespaced

- eBPF has to be loaded from **rootful containers** - rootless containers will not work since bpf is not namespaced.
- Even with the basic set of permissions, root privilege escalation is possible.
  - See: CVE-2022-23222 - now patched.
- Since eBPF in containers are able to modify the host kernel, they are not really containing anything.

[1]



Some eBPF features are not restricted by the container namespaces and can affect all processes in the kernel.



[1]source: Cross Container Attacks: The Bewildered eBPF on Clouds

Highly Privileged Containers

At minimum **CAP_BPF** is required, which should effectively be considered as root.

However, in practice:

- Many BPF programs and attach points require additional capabilities:
  - **CAP_SYS_PTRACE**, **CAP_NET_ADMIN** and even **CAP_SYS_ADMIN**
- In the “compile on-the-fly” use-case, these privileges are maintained for the lifetime of the program loading probes.
- In all cases, these privileges include things that aren’t strictly necessary for eBPF and are too coarsely grained to be useful.

Since pods that load eBPF are mostly long-lived and often don’t drop privileges it leaves a wide attack surface.

An Example: Scary Things we can do with eBPF

1. Snoop on all SSL traffic: github.com/alessandrod/snuffy
2. Redirect incoming TCP 22 (SSH) traffic the host into my container as part of a phishing attack
3. Deploy a keylogger: github.com/willfindlay/bpf-keylogger
4. Naively attach probes to application functions or tracepoints that are called at a high frequency, grinding the system to a halt.
5. Exploit the K8s Trampoline Pod Threat (DEMO!!)

Functional Problems with eBPF

- Silos: interoperability problems, can cause **cluster instability**
- Visibility into the ebpf subsystem is often limited at the cluster scope
- No fine grained versioning between userspace + kernelspace
- eBPF loading + management stack is duplicated by every application


KubeCon | CloudNativeCon

North America 2023

Enter bpfd

Introducing bpfd

- Open source project started in the Red Hat Emerging Tech Networking Group
- An eBPF program manager
  - **Manages lifecycle** (Loading/Unloading/Pinning) of eBPF Programs, removing CAP_BPF from applications
  - Program Cooperation
  - Central daemon for **managing loading policy** for security and visibility
- bpfd is developed in Rust, built on top of the **Rust eBPF library Aya**.
- Includes a **K8s operator and APIs** (developed in Go/controller-runtime)

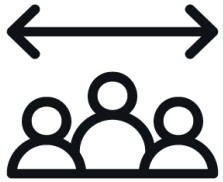
bpfd.dev

aya-rs.dev

bpfd: core-features

Security

- CAP_BPF isolation
- Kubernetes RBAC
- Cosign Integration



Productivity

- Application deduplication
- eBPF distribution and version control using our OCI bytecode image spec

Observability

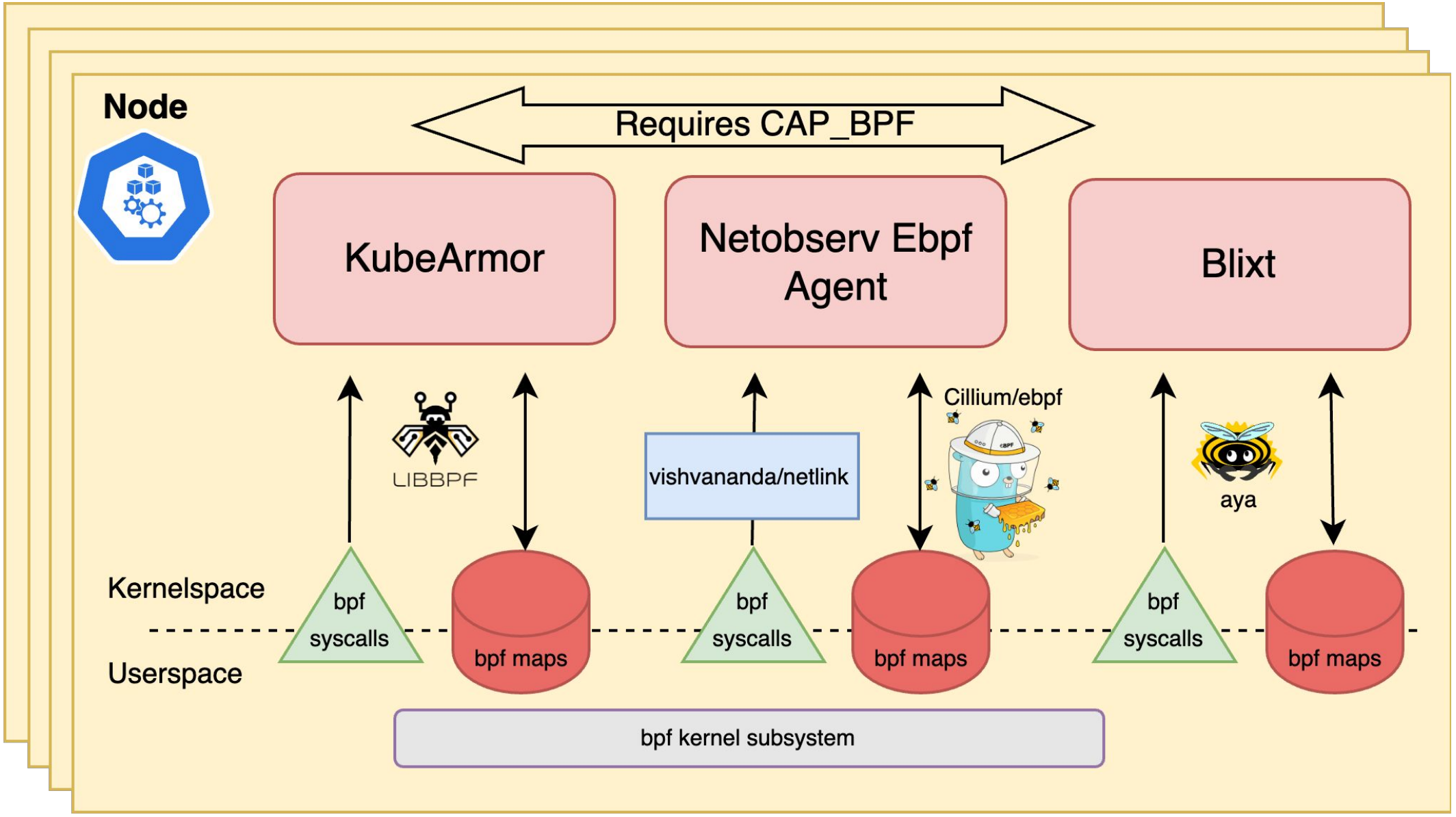
- eBPF subsystem monitoring
- In Kubernetes node based eBPF subsystem state reflected via K8s api

Program Support

- Support for:
  - XDP
  - TC
  - Tracepoint
  - Uprobe
  - Kprobe
- leverages the libxdp protocol for XDP/TC

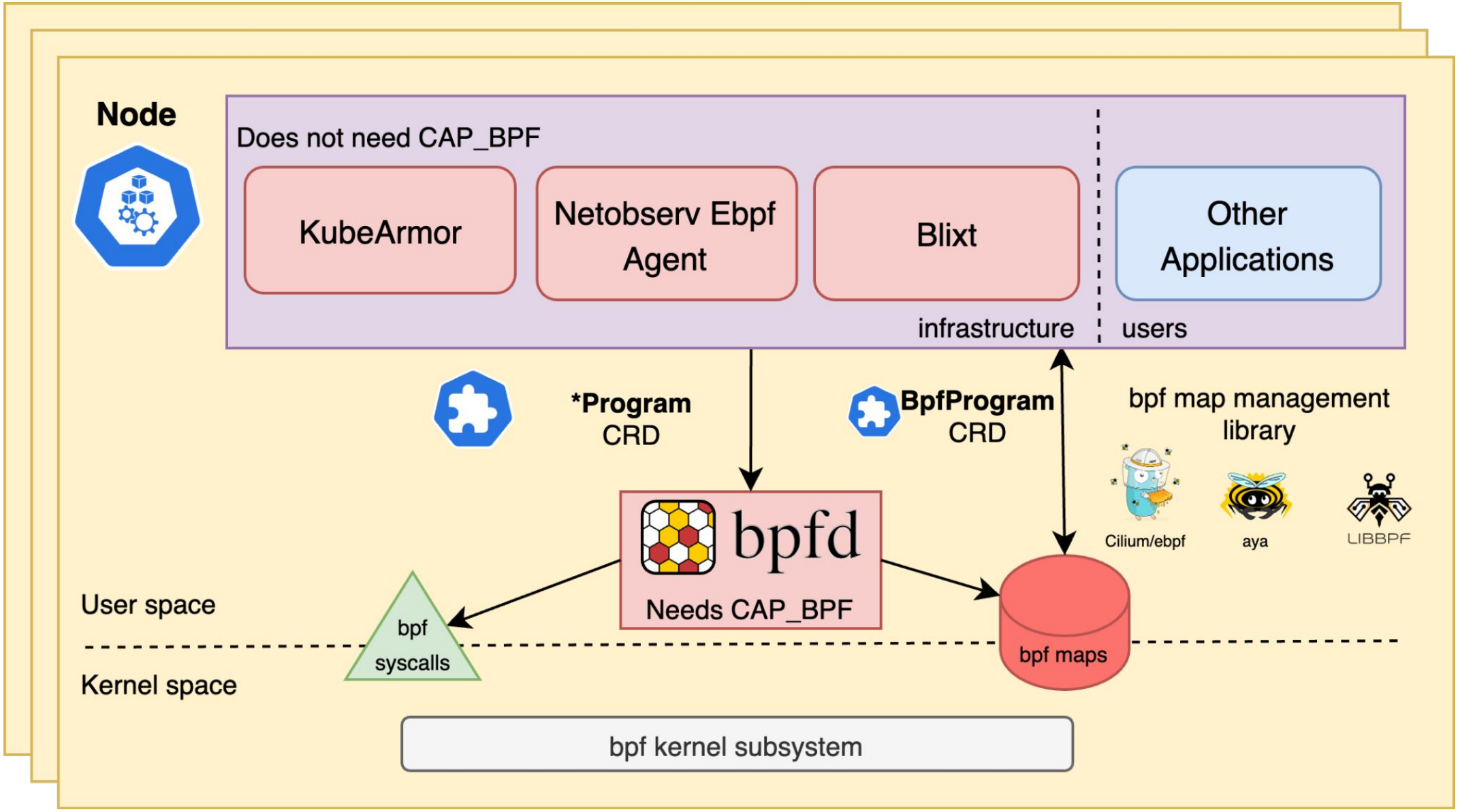
bpfd.dev

bpf on Kubernetes before bpfd



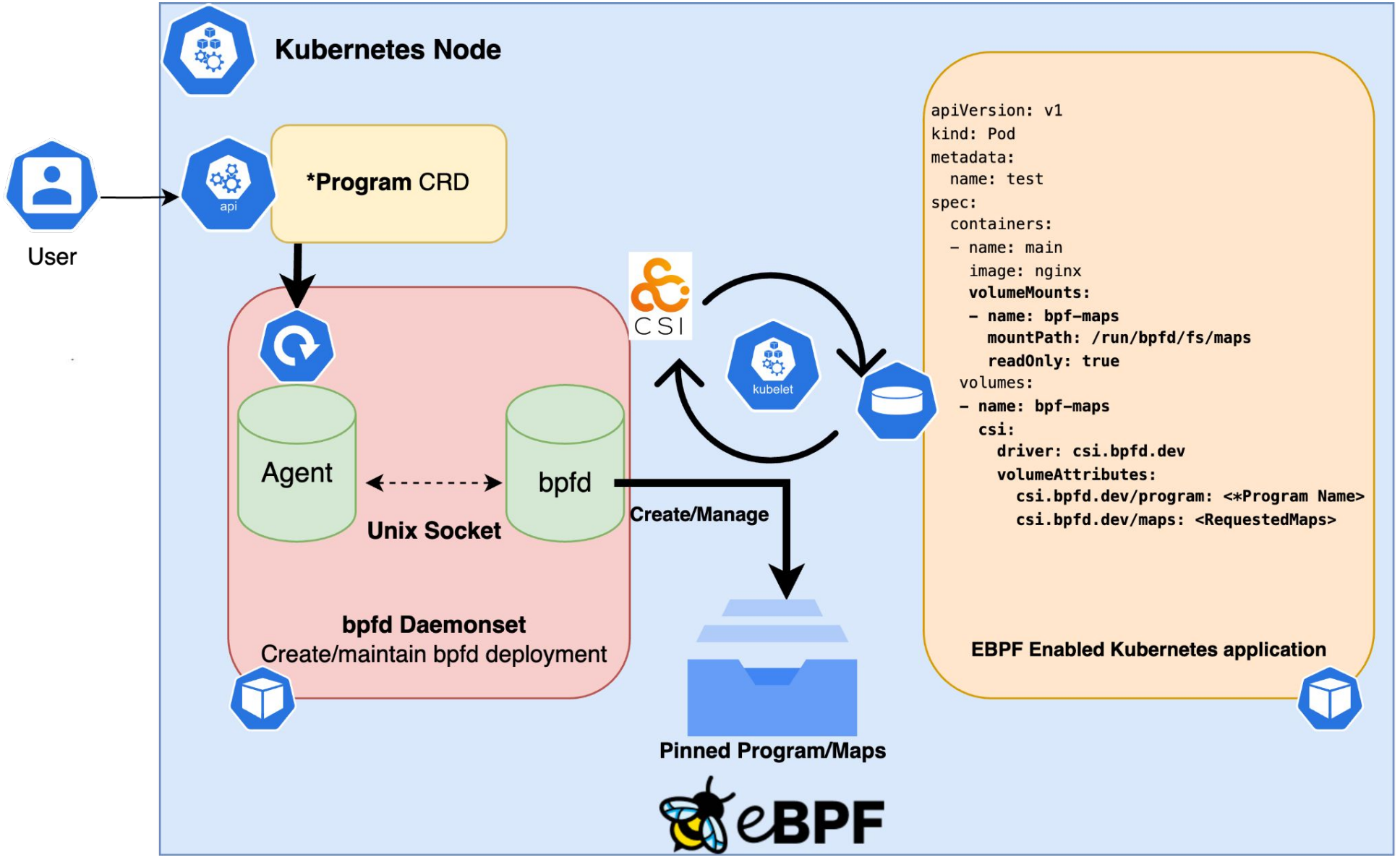
bpfd.dev

bpf on Kubernetes after bpfd



bpfd.dev

bpfd + Kubernetes Architecture



bpfd.dev

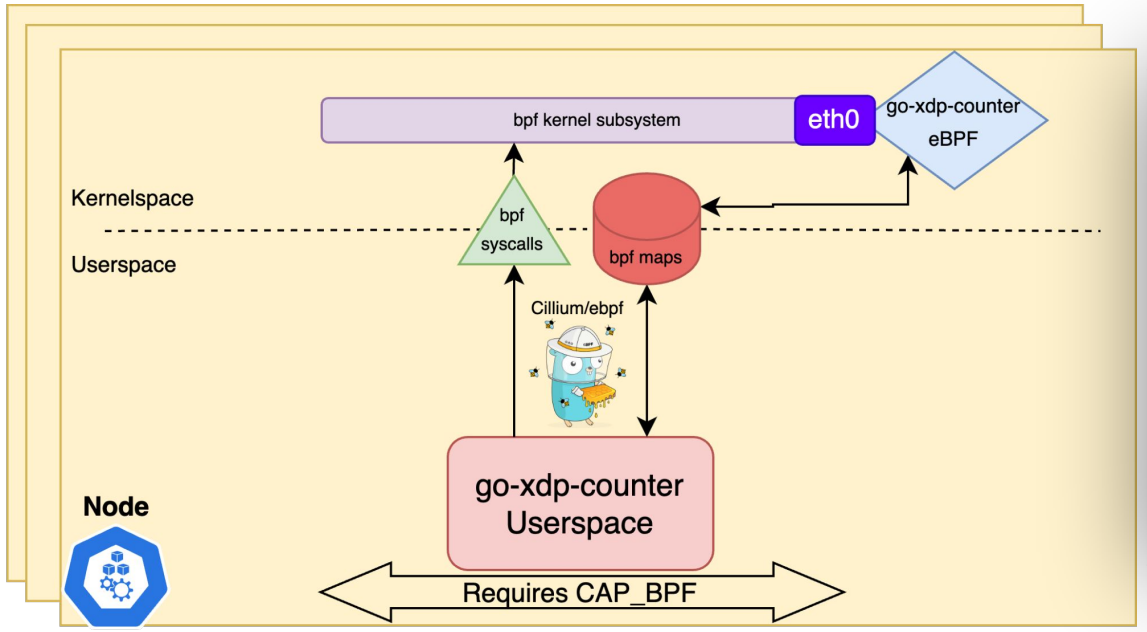

KubeCon | CloudNativeCon

North America 2023

DEMO

Peaceful eBPF: A Xdp Counter

- XDP Program attached to the main network interface on a node that simply counts the number of bytes going by.
- Userspace + eBPF programs are compiled together into one binary, requiring a privileged daemonset for its deployment

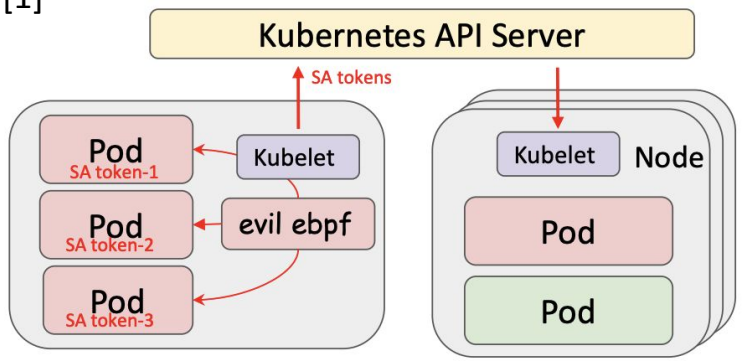



```
[astoycos@nfvsdn-02-oot examples]$ kubectl logs go-xdp-counter-ds-29klv -n go-xdp-counter
2023/10/31 03:04:06 Attached XDP program to iface "eth0" (index 2119)
2023/10/31 03:04:06 Press Ctrl-C to exit and remove the program
2023/10/31 03:04:09 25 packets received
2023/10/31 03:04:09 3809 bytes received

2023/10/31 03:04:12 49 packets received
2023/10/31 03:04:12 7552 bytes received

2023/10/31 03:04:15 49 packets received
2023/10/31 03:04:15 7552 bytes received
```

An EVIL Xdp Counter Application 😈

- For this demo we've implemented a service-account token stealer
- Using eBPF tracepoint programs its relatively simple to:
  a. Break out of the Pod's container boundaries
  b. Listen for all /var/run/secrets/kubernetes.io/serviceaccount/token reads
  c. Steal the token from memory and write within our evil pod

[1]



On a vulnerable VM (node), all Pods' service accounts (SA) can be abused by eBPF attackers.

```
rules:
- apiGroups: ["stable.example.com"]
  resources: ["crontabs"]
  verbs: ["get", "list", "watch", "create", "update", "patch", "delete"]
```

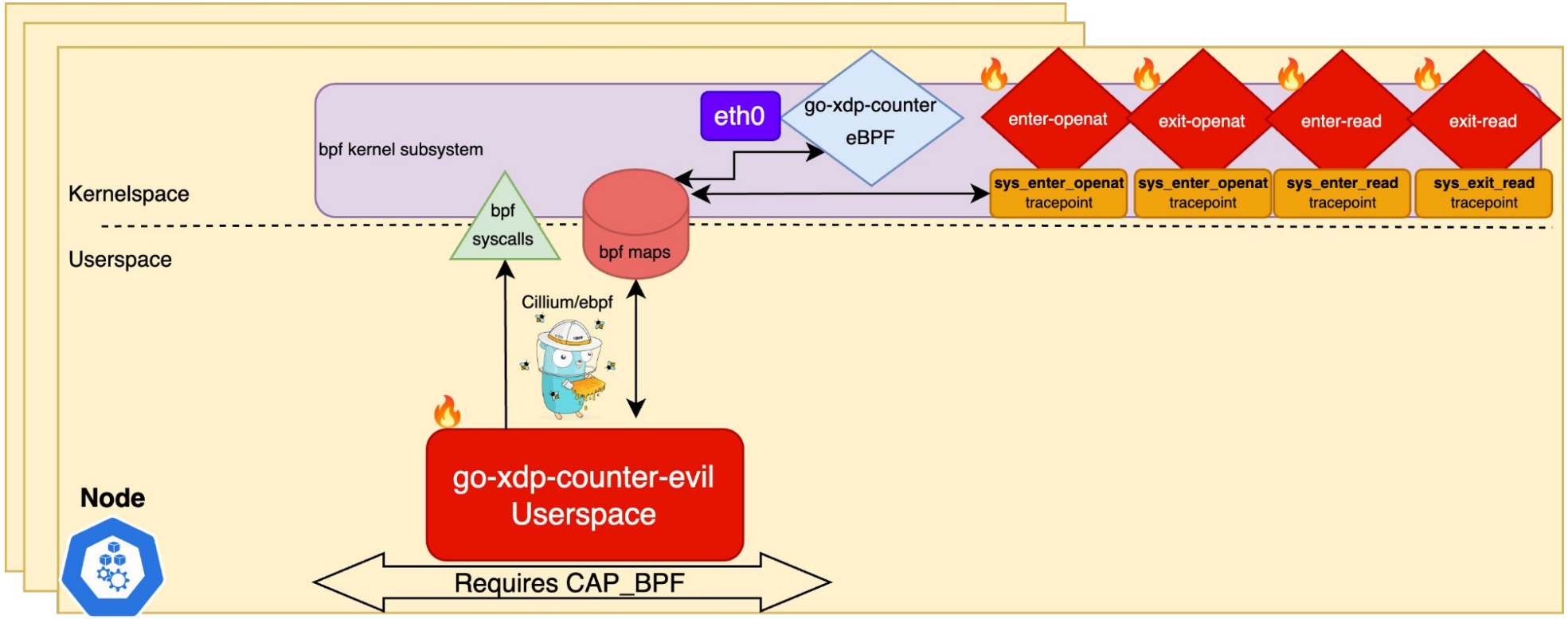
Some Pods have powerful permissions to affect Pods on other nodes.

```
# steal other Pods' service account tokens
$ export TOKEN=$(evil-ebpf-read
/var/run/secrets/kubernetes.io/serviceaccount/token)

# maniplute other nodes
$ curl -k --header "Authorization: Bearer $TOKEN"
https://172.16.22.202:10250/...
```

[1]source: Cross Container Attacks: The Bewildered eBPF on Clouds

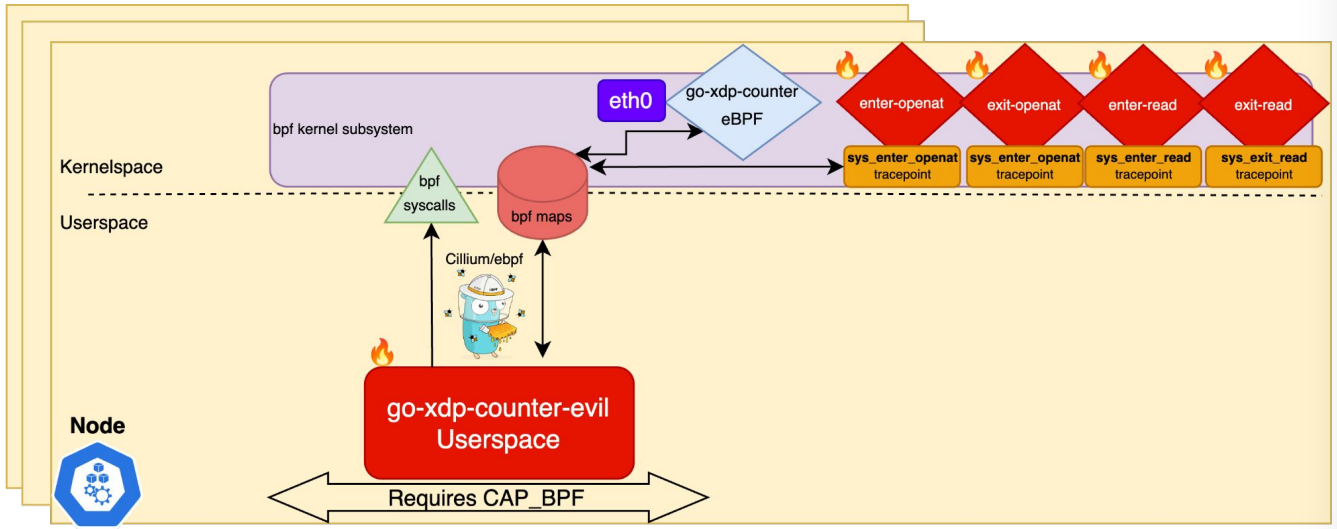
Peaceful eBPF -> Evil eBPF

- If the binary containing the userspace + eBPF programs is compromised (supply chain attack, privilege abuse, user spoofing, etc.) exploitation is relatively simple.
- Application container is privileged, and can easily load + attach multiple other 😈 bpf programs.



Evil eBPF Exploit

- Uh oh the evil program is now dumping service account tokens… Let's use them




```
2023/10/30 13:49:24
pid: 1284470

comm: coredns

token: ey{.....}oA

parsed token info: {
   "aud": [
      "https://kubernetes.default.svc.cluster.local"
   ],
   "exp": 1730208446,
   "iat": 1698672446,
   "iss": "https://kubernetes.default.svc.cluster.local",
   "kubernetes.io": {
      "namespace": "kube-system",
      "pod": {
         "name": "coredns-5d78c9869d-8fd5b",
         "uid": "ee7ee783-8d98-4ec9-8a2a-839863da835b"
      },
      "serviceaccount": {
         "name": "coredns",
         "uid": "2f730bae-fbe5-4d78-8090-721b080da198"
      },
      "warnafter": 1698676053
   },
   "nbf": 1698672446,
   "sub": "system:serviceaccount:kube-system:coredns"
}
```

Invisible Evil eBPF

- Due to the lack of standard observability into the eBPF subsystem in Kubernetes cluster, this evil eBPF could hide pretty easily FOREVER since everything still appears to be running as expected.

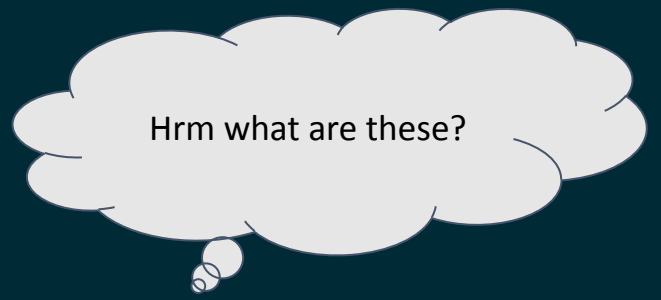

```
[astoycos@nfvsdn-02-oot bpfd]$ kubectl get pods -A
NAMESPACE            NAME                                                          READY   STATUS    RESTARTS   AGE
go-xdp-counter       go-xdp-counter-ds-rq2gq                                       1/1     Running   0          98m
kube-system          coredns-5d78c9869d-j9cdf                                      1/1     Running   0          99m
kube-system          coredns-5d78c9869d-jc4d4                                      1/1     Running   0          95m
kube-system          etcd-kubecon-na-2023-bpfd-demo-control-plane                  1/1     Running   0          99m
kube-system          kindnet-nzm5t                                                 1/1     Running   0          99m
kube-system          kube-apiserver-kubecon-na-2023-bpfd-demo-control-plane        1/1     Running   0          99m
kube-system          kube-controller-manager-kubecon-na-2023-bpfd-demo-control-plane 1/1   Running   0          99m
kube-system          kube-proxy-5m4cg                                              1/1     Running   0          99m
kube-system          kube-scheduler-kubecon-na-2023-bpfd-demo-control-plane        1/1     Running   0          99m
local-path-storage   local-path-provisioner-6bc4bddd6b-sqsgd                       1/1     Running   0          99m
```

Peaceful eBPF -> Evil eBPF

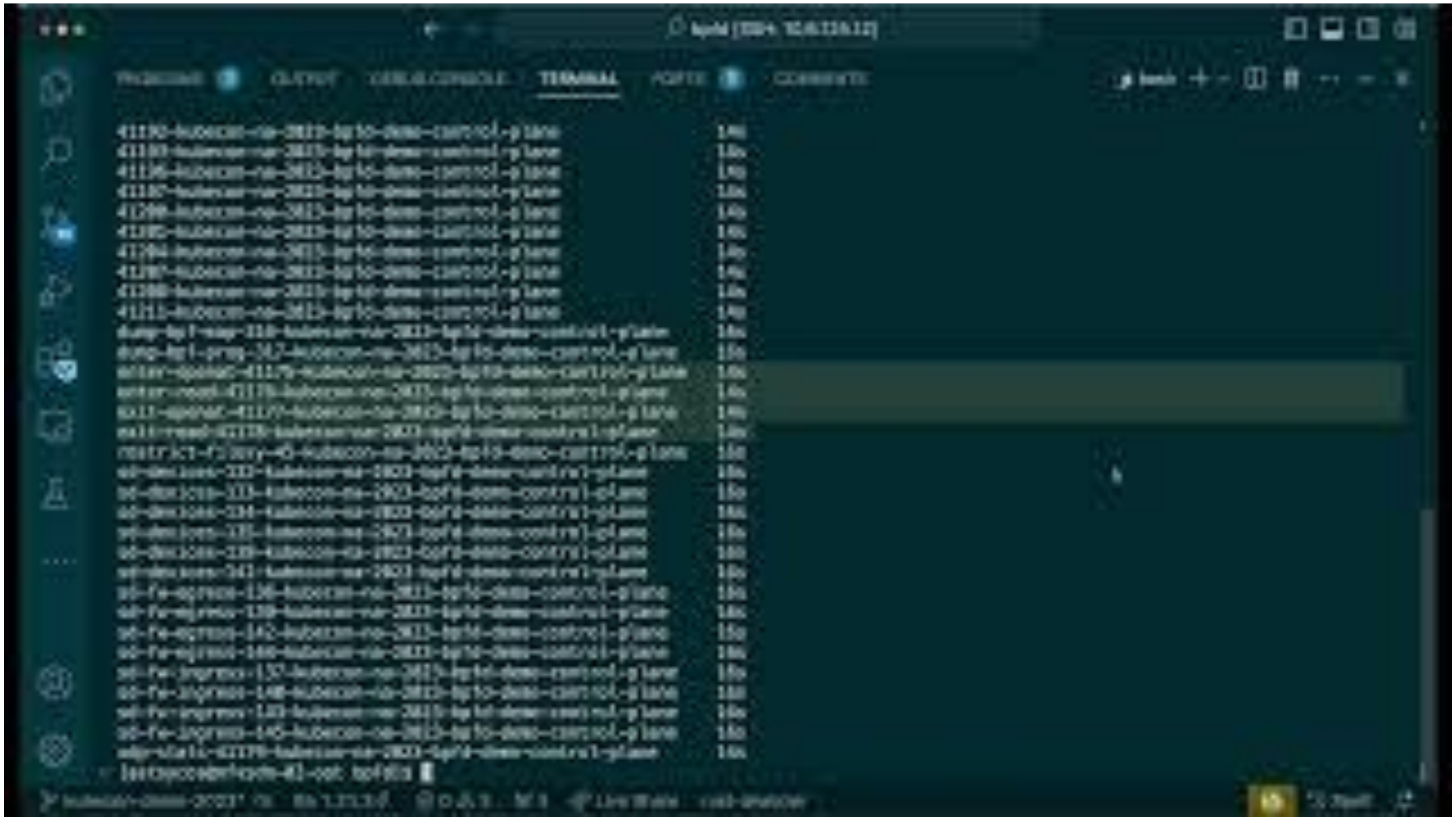
Enter bpfd: Discovery

- Step 1: Start by installing bpfd on the compromised cluster.
- Step 2: Easily inspect all of the bpf programs running on a given node

```
[astoycos@nfvsdn-02-oot bpfd]$ kubectl get bpfprogram -l kubernetes.io/hostname=kubecon-na-2023-bpfd-demo-control-plane
NAME                                                              AGE
...
dump-bpf-map-316-kubecon-na-2023-bpfd-demo-control-plane          61m
dump-bpf-prog-317-kubecon-na-2023-bpfd-demo-control-plane         61m
enter-openat-41175-kubecon-na-2023-bpfd-demo-control-plane        61m
enter-read-41176-kubecon-na-2023-bpfd-demo-control-plane          61m
exit-openat-41177-kubecon-na-2023-bpfd-demo-control-plane         61m
exit-read-41178-kubecon-na-2023-bpfd-demo-control-plane           61m
restrict-filesy-45-kubecon-na-2023-bpfd-demo-control-plane        61m
sd-devices-132-kubecon-na-2023-bpfd-demo-control-plane            61m
sd-devices-133-kubecon-na-2023-bpfd-demo-control-plane            61m
sd-devices-134-kubecon-na-2023-bpfd-demo-control-plane            61m
sd-devices-135-kubecon-na-2023-bpfd-demo-control-plane            61m
sd-devices-138-kubecon-na-2023-bpfd-demo-control-plane            61m
sd-devices-141-kubecon-na-2023-bpfd-demo-control-plane            61m
sd-fw-egress-136-kubecon-na-2023-bpfd-demo-control-plane          61m
sd-fw-egress-139-kubecon-na-2023-bpfd-demo-control-plane          61m
sd-fw-egress-142-kubecon-na-2023-bpfd-demo-control-plane          61m
sd-fw-egress-144-kubecon-na-2023-bpfd-demo-control-plane          61m
sd-fw-ingress-137-kubecon-na-2023-bpfd-demo-control-plane         61m
sd-fw-ingress-140-kubecon-na-2023-bpfd-demo-control-plane         61m
sd-fw-ingress-143-kubecon-na-2023-bpfd-demo-control-plane         61m
sd-fw-ingress-145-kubecon-na-2023-bpfd-demo-control-plane         61m
xdp-stats-41179-kubecon-na-2023-bpfd-demo-control-plane           61m
```



Enter bpfd: Discovery



Enter bpfd: Discovery


```yaml
apiVersion: bpfd.dev/v1alpha1
kind: BpfProgram
metadata:
  annotations:
    BTF-ID: "16887"
    GPL-Compatible: "true"
    JITed: "true"
    Kernel-Allocated-Memory-Bytes: "4096"
    Kernel-ID: "41175"
    Loaded-At: 2023-10-31T18:55:16+0000
    Map-IDs: '[22699 22700]'
    Name: enter_openat
    Size-JITed-Bytes: "198"
    Size-Translated-Bytes: "352"
    Tag: f034f9ddad3e2dd9
    Type: tracepoint
    Verified-Instruction-Count: "861"
  creationTimestamp: "2023-10-31T19:36:04Z"
  generation: 1
  labels:
    bpfd.dev/discoveredProgram: ""
    kubernetes.io/hostname: kubecon-na-2023-bpfd-demo-control-plane
  name: enter-openat-41175-kubecon-na-2023-bpfd-demo-control-plane
  resourceVersion: "3839"
  uid: b266facc-a0ef-42a1-9b2c-c84ede061ef9
spec:
  type: tracepoint
```

Let's take a closer look..
What Pod loaded this?

```yaml
apiVersion: v1
kind: Pod
metadata:
  creationTimestamp: "2023-10-31T18:54:54Z"
  generateName: go-xdp-counter-ds-
  labels:
    controller-revision-hash: c67d7f6b5
    name: go-xdp-counter
    pod-template-generation: "1"
  name: go-xdp-counter-ds-rq2gq
  namespace: go-xdp-counter
...
status:
  containerStatuses:
  - ...
    name: go-xdp-counter
    ready: true
    restartCount: 0
    started: true
    state:
      running:
        startedAt: "2023-10-31T18:55:16Z"
  hostIP: 172.19.0.2
  phase: Running
  podIP: 172.19.0.2
  podIPs:
  - ip: 172.19.0.2
  qosClass: BestEffort
  startTime: "2023-10-31T18:54:55Z"
```

bpfd: Mitigation

- Start by Deleting the evil application
- Re-deploy with bpfd

```yaml
apiVersion: bpfd.dev/v1alpha1
kind: XdpProgram
metadata:
  labels:
    app.kubernetes.io/name: xdpprogram
  name: go-xdp-counter-example
spec:
  bpffunctionname: xdp_stats
  # Select all nodes
  nodeselector: {}
  interfaceselector:
    primarynodeinterface: true
  priority: 55
  bytecode:
    image:
      url: quay.io/bpfd-bytecode/go-xdp-counter-evil:latest
```

Bytecode is still evil?

```yaml
apiVersion: apps/v1
kind: DaemonSet
metadata:
  name: go-xdp-counter-ds
  namespace: go-xdp-counter
…
spec:
…
    spec:
      nodeSelector: {}
      hostNetwork: true
      serviceAccountName: bpfd-app-go-xdp-counter
      securityContext:
        privileged: false
…
      containers:
      - name: go-xdp-counter
        image: quay.io/bpfd-userspace/go-xdp-counter:latest
…
        volumeMounts:
        - name: bpf-maps
          mountPath: /run/bpfd/fs/maps
          readOnly: true
      volumes:
      - name: bpf-maps
        csi:
          driver: csi.bpfd.dev
          volumeAttributes:
            csi.bpfd.dev/program: go-xdp-counter-example
            csi.bpfd.dev/maps: xdp_stats_map
```

bpfd: Mitigation with K8s RBAC

- Even if the bytecode image is compromised the evil program will not be loaded with bpfd!
- A **TracepointProgram** object would need to be created which is disallowed by the cluster admin via RBAC mechanisms.

```yaml
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRole
metadata:
  creationTimestamp: null
  name: go-xdp-counter-namespace-owner-role
rules:
- apiGroups:
  - bpfd.dev
  resources:
  - xdpprograms
  verbs:
  - create
  - get
  - update
```

```yaml
apiVersion: bpfd.dev/v1alpha1
kind: XdpProgram
metadata:
  labels:
    app.kubernetes.io/name: xdpprogram
  name: go-xdp-counter-example
spec:
  bpffunctionname: xdp_stats
  # Select all nodes
  nodeselector: {}
  interfaceselector:
    primarynodeinterface: true
  priority: 55
  bytecode:
    image:
      url: quay.io/bpfd-bytecode/go-xdp-counter-evil:latest
```

apiVersion: bpfd.dev/v1alpha1
kind: **TracepointProgram**
name: enter-openat

apiVersion: bpfd.dev/v1alpha1
kind: **TracepointProgram**
name: enter-openat

apiVersion: bpfd.dev/v1alpha1
kind: **TracepointProgram**
name: enter-openat

apiVersion: bpfd.dev/v1alpha1
kind: **TracepointProgram**
name: enter-openat

Create

Compromised service account

bpfd-app-go-xdp-counter

Create

K8s RBAC Controls

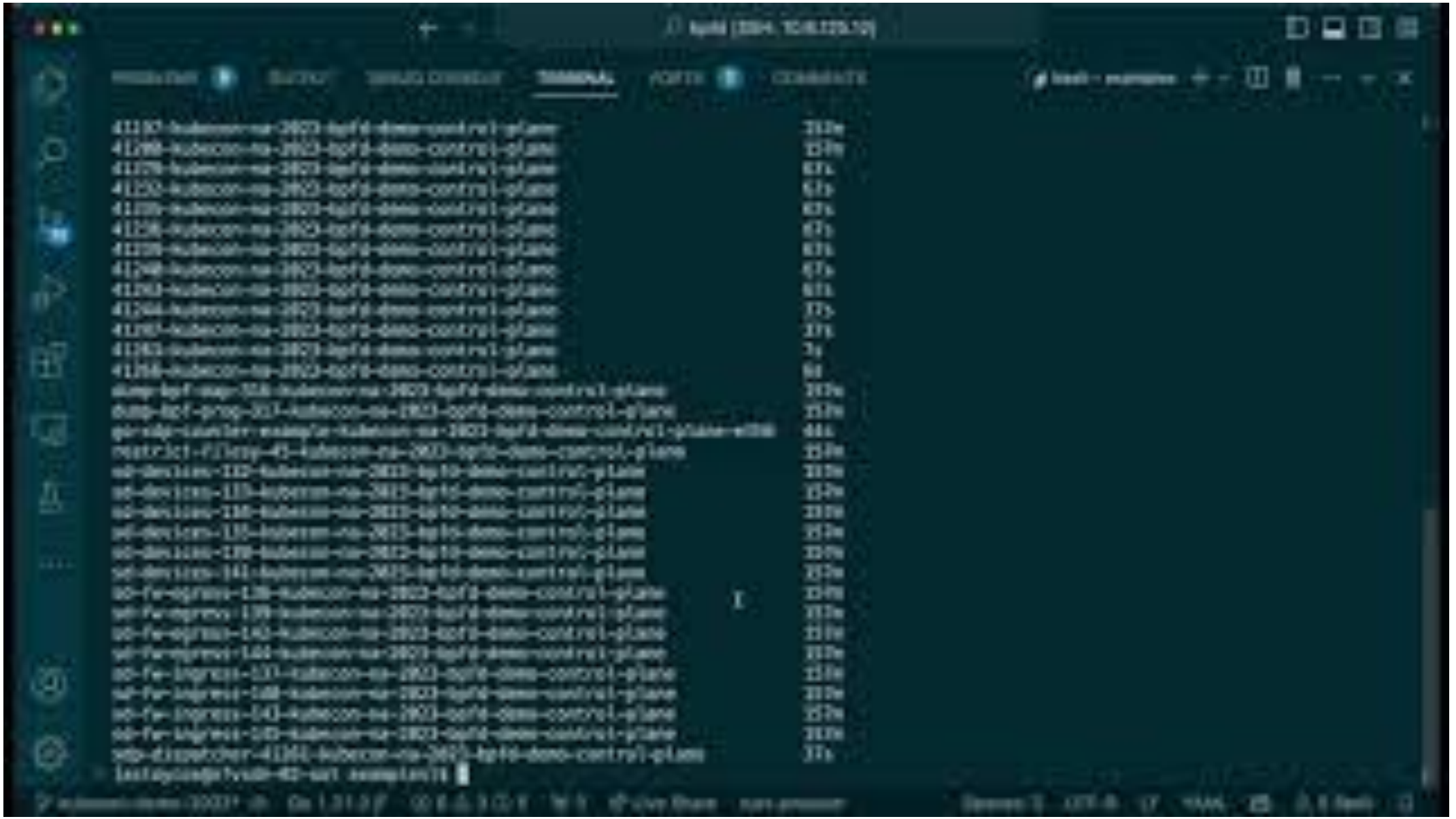
bpfd: Mitigation with bytecode signing

- Now we're loading with bpfd so it's much harder to load unintended programs since all bpf behavior is declarative.
- However what if the bytecode image was compromised?
  a. **quay.io/bpfd-bytecode/go-xdp-counter-evil:latest (unsigned)**
  b. **quay.io/bpfd-bytecode/go-xdp-counter:latest (signed)**

Bytecode is still evil, but Cosign integration warns the user.

```
[astoycos@nfvsdn-02-oot examples]$ kubectl logs bpfd-daemon-bg99x -n bpfd
Defaulted container "bpfd" out of: bpfd, bpfd-agent, node-driver-registrar
[INFO  bpfd] Log using env_logger
[INFO  bpfd] Has CAP_BPF: true
[INFO  bpfd] Has CAP_SYS_ADMIN: true
[INFO  bpfd::certs] CA Certificate file /etc/bpfd/certs/ca/ca.pem does not exist. Creating CA Certificate.
[INFO  bpfd::certs] bpfd Certificate Key /etc/bpfd/certs/bpfd/bpfd.key does not exist. Creating bpfd Certificate.
[INFO  bpfd::certs] bpfd-client Certificate Key /etc/bpfd/certs/bpfd-client/bpfd-client.key does not exist. Creating bpfd-client Certificate.
[INFO  bpfd::oci_utils::cosign] Starting Cosign Verifier, downloading data from Sigstore TUF repository
[INFO  bpfd::serve] Listening on /bpfd-sock/bpfd.sock
[INFO  bpfd::storage] CSI Plugin Listening on /run/bpfd/csi/csi.sock
[WARN  bpfd::oci_utils::cosign] The bytecode image: quay.io/bpfd-bytecode/go-xdp-counter-evil:latest is unsigned
[INFO  bpfd::command] Loading program bytecode from container image: quay.io/bpfd-bytecode/go-xdp-counter-evil:latest
[INFO  bpfd::oci_utils::cosign] The bytecode image: quay.io/bpfd/xdp-dispatcher:v2 is signed
[INFO  bpfd::bpf] Added xdp program with name: xdp_stats and id: 40668
```

bpfd: Mitigation




KubeCon | CloudNativeCon

North America 2023

Getting Involved

Getting Started

- You can **deploy to a standard Linux system** and use **bpfctl**
- You can **deploy the bpfd operator on a kind cluster** (or similar)
- We provide several examples, or you can bring your existing program into the fold
  - **TC**, **XDP**, **Tracepoint**, e.t.c.
- Our website has getting started information, and several guides

https://bpfd.dev

Work in Kubernetes SIG Network with Gateway API

- Blixt: A Layer 4 load-balancer deployable with bpfd
  - Part of the Gateway API project
  - TC for ingress/egress networking
  - Used for CI and testing scenarios
  - Maintained by us!
- Potentially a good jumping in point if you're interested in eBPF networking

https://github.com/kubernetes-sigs/blixt

Community

https://bpfd.dev/community

- Weekly community meetings on Thursdays
- **#bpfd** channel on Kubernetes Slack (**#ebpf** for general ebpf discussions as well)
- We're also very active in the Aya and Rust communities on Discord

https://aya-rs.dev/community

Roadmap

https://bpfd.dev/community

- See the Bpfd Github Project
- Plans to apply CNCF Sandbox project
- A future daemon-less design?

https://aya-rs.dev/community

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

Building new projects with bpfd

Not sure if we're going to keep this or scrap this…

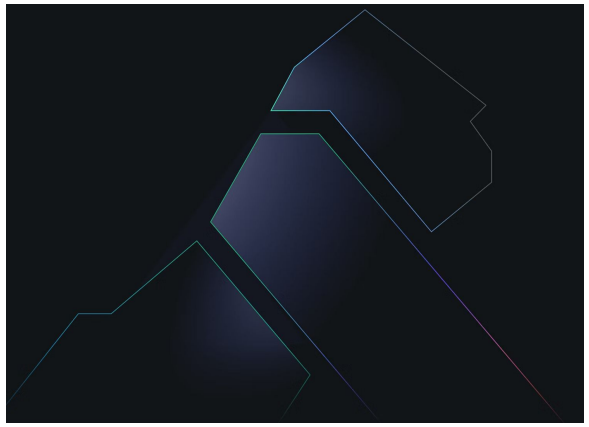
https://bpfd.dev/getting-started/example-bpf/

Planning Slide

1. Quick into to eBPF
2. the problems with ebpf in Kubernetes
3. move to security focused discussion, explain how bad eBPF security posture currently is due to wild west loader, privileged containers, e.t.c.
3. Intro to bpfd and the problems it's trying to solve
4. Demo
   - Sample "Normal" application
   - Application gets hijacked
     - root cluster exploit (more likely)
     - Supply chain exploit of OCI container image (less likely)
   - User can see what's happening with Bpfd
   - Add some Cosign container image signing

Current Adoption

RedHat



Kong

Kubernetes SIGs

https://github.com/kubernetes-sigs/blixt

bpfd Features: loading + cooperation

Linux Node

```
Bpfd State
Name:          xdp_stats
Image URL:     quay.io/bpfd-bytecode/go-xdp-counter:latest
Pull Policy:   Always
Global:        None
Metadata:      None
Map Pin Path:  /run/bpfd/fs/maps/39664
Map Owner ID:  None
Maps Used By:  39664
Priority:      55
Iface:         eno2
Position:      0
Proceed On:    pass, dispatcher_return
```

```
Bpfd State
Name:          pass
Image URL:     quay.io/bpfd-bytecode/xdp_pass:latest
Pull Policy:   Always
Global:        None
Metadata:      None
Map Pin Path:  /run/bpfd/fs/maps/39662
Map Owner ID:  None
Maps Used By:  39662
Priority:      50
Iface:         eno2
Position:      0
Proceed On:    pass, dispatcher_return
```

Why  ?

https://rust-lang.org

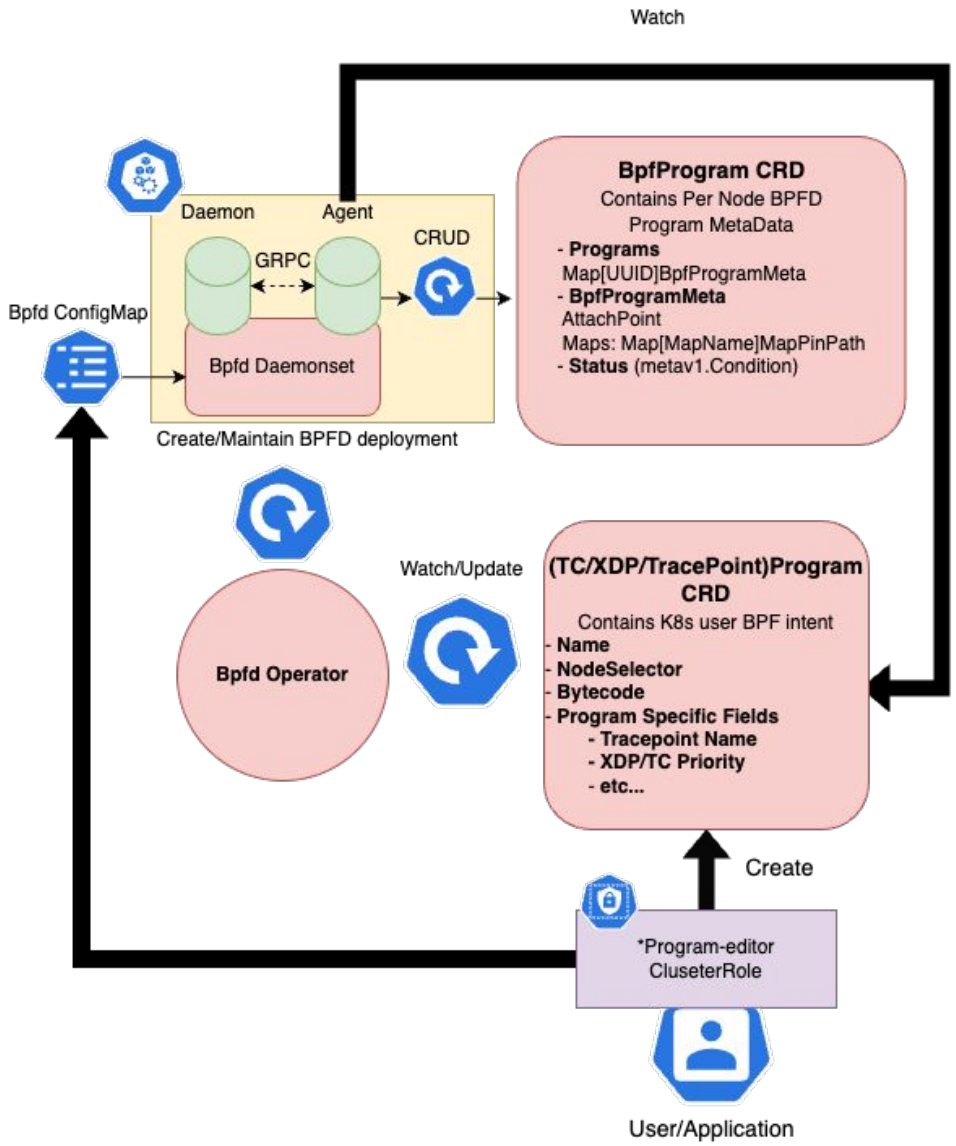
Aya

https://aya-rs.dev

- Support for the **BPF Type Format (BTF)**, which is transparently enabled when supported by the target kernel. This allows eBPF programs compiled against one kernel version to run on different kernel versions without the need to recompile.
- Support for f**unction call relocation and global data maps**, which allows eBPF programs to make function calls and use global variables and initializers.
- **Async support** with both tokio and async-std.
- Easy to deploy and fast to build: aya doesn't require a kernel build or compiled headers, and not even a C toolchain; **a release build completes in a matter of seconds.**

BPFD Kubernetes Operator

- Bpfd Operator
  - Written in go using Operator SDK framework
  - Local kind setup can be tested with `make run-on-kind`
- K8s APIs
  - *Program CRDs
  - BpfProgram CRD
  - Bpfd Configmap
- eBPF Bytecode Image Specifications
  - Allows fine-grained separate versioning control for userspace and kernel space
  - Opens the door for signing of these container images in order to verify bytecode ownership



BPFD Security