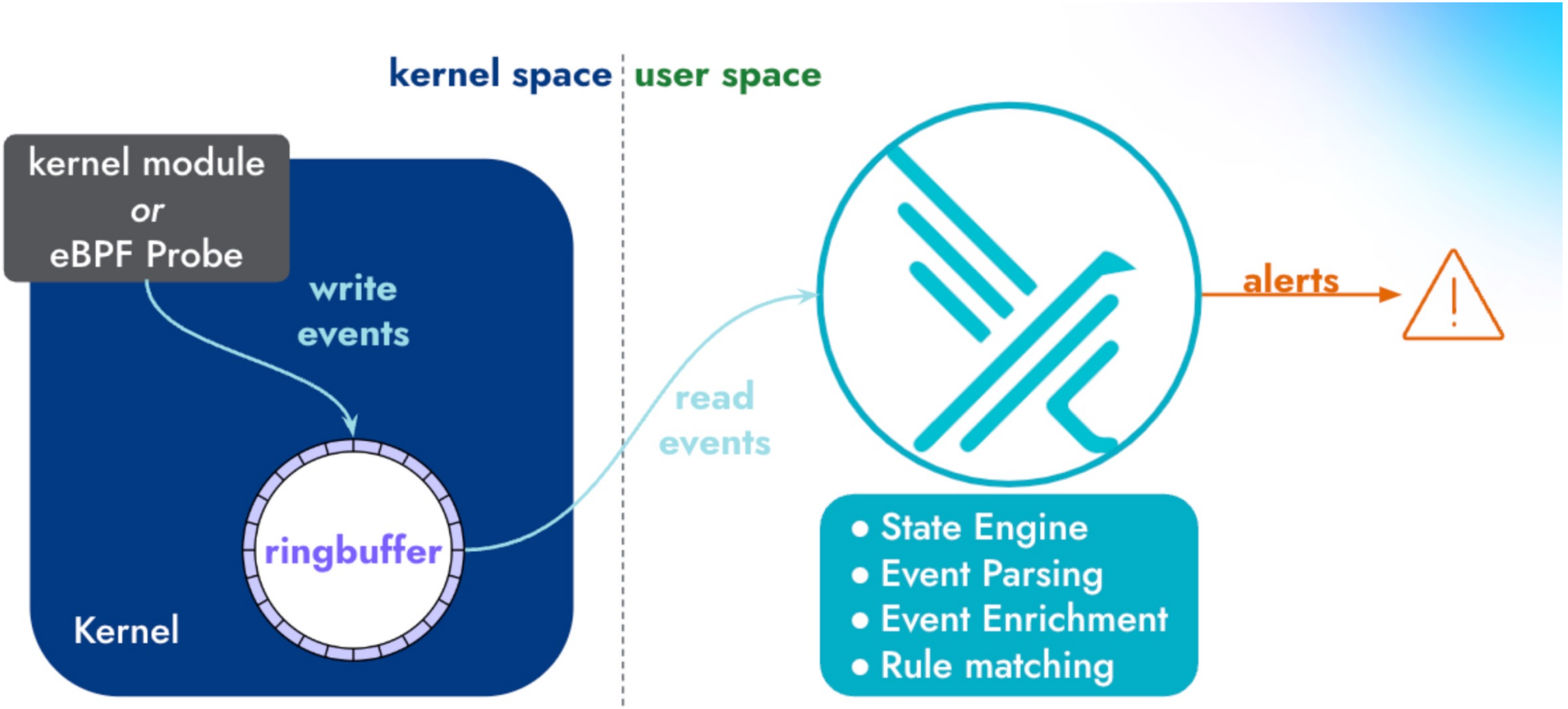
Falco

Detection

Two Different Types of Detection

We're going to find Falco really useful for two types of detection:

1) Container/Host-focused detection via eBPF / system call monitoring
2) Running rules against Kubernetes audit logs


kernel space
user space
kernel module
or
eBPF Probe
write
events
ringbuffer
Kernel
read
events
alerts
• State Engine
• Event Parsing
• Event Enrichment
• Rule matching

```
{
  "hostname": "falco-x8zt7",
  "output": "08:46:12.217241028: Warning Mount was executed inside a privileged container (user=root user_loginuid=-1 command=mount /dev/sda
nt pid=31536 k8s.ns=default k8s.pod=privpod6 container=4e45e91f6801 image=docker.io/bustakube/alpine-small-attack:latest)",
  "priority": "Warning",
  "rule": "Mount Launched in Privileged Container",
  "source": "syscall",
  "tags": [
    "T1611",
    "cis",
    "container",
    "filesystem",
    "mitre_lateral_movement"
  ],
  "time": "2023-08-08T08:46:12.217241028Z",
  "output_fields": {
    "container.id": "4e45e91f6801",
    "container.image.repository": "docker.io/bustakube/alpine-small-attack",
    "container.image.tag": "latest",
    "evt.time": 1691484372217241000,
    "k8s.ns.name": "default",
    "k8s.pod.name": "privpod6",
    "proc.cmdline": "mount /dev/sda /mnt",
    "proc.pid": 31536,
    "user.loginuid": -1,
    "user.name": "root"
  }
}
```

Falco Rule Structure

```
- rule: shell_in_container
  desc: notice shell activity within a container
  condition: > evt.type = execve and evt.dir = <
             and container.id != host and
             (proc.name = bash or proc.name = ksh)
  output: > shell in a container …
  priority: WARNING
```

```
- rule: Run shell untrusted
  desc: >
    An attempt to spawn a shell below a non-shell application. The non-shell applications that are monitored are defined i
    with 'protected_shell_spawning_binaries' being the list you can easily customize. For Java parent processes, please nc
    Therefore, rely more on 'proc.exe' to define Java applications. This rule can be noisier, as you can see in the exhaus
    very behavior-driven and broad, it is universally relevant to catch general Remote Code Execution (RCE). Allocate time
    reduce noise. Tuning suggestions include looking at the duration of the parent process ('proc.ppid.duration') to defir
    for newer fields such as 'proc.vpgid.name' and 'proc.vpgid.exe' instead of the direct parent process being a non-shell
  condition: >
    spawned_process
    and shell_procs
    and proc.pname exists
    and protected_shell_spawner
    and not proc.pname in (shell_binaries, gitlab_binaries, cron_binaries, user_known_shell_spawn_binaries,
                           needrestart_binaries,
                           mesos_shell_binaries,
                           erl_child_setup, exechealthz,
                           PM2, PassengerWatchd, c_rehash, svlogd, logrotate, hhvm, serf,
                           lb-controller, nvidia-installe, runsv, statsite, erlexec, calico-node,
                           "puma reactor")
```

Lists and Macros

```
- list: shell_binaries
  items: [bash, csh, ksh, sh, tcsh, zsh, dash]
- list: userexec_binaries
  items: [sudo, su]
- list: known_binaries
  items: [shell_binaries, userexec_binaries]
- macro: safe_procs
  condition: proc.name in (known_binaries)
```

Deactivating Rules

```
- macro: user_known_read_sensitive_files_activities
  condition: (always_true)
```

Types of Events Caught via Audit Logging

- Attach/exec into pod

- Listing/reading resources: kubectl get pods, etc.
- Create/edit/delete deployment, namespace, pod, secret, …
- Create RBAC items: service account, role/cluster role, rolebindings, cluster role bindings

Falco Sidekick

Falco Sidekick is also open source.

It can send alerts to a number of communications and logging sources.

Configure the API Server to Write Audit Logs

```
k8s_audit_log_file: /var/log/kubernetes/audit/audit.log
```

```
k8s_audit_policy_file: /path/to/audit-policy.yaml
```

Installing Falco with Audit Log Rules via Helm

First, create your values.yaml file to configure the helm chart. An example is on the next three slides.

Then, install Falco via Helm.

```
helm repo add falcosecurity https://falcosecurity.github.io/charts
helm repo update
helm install falco falcosecurity/falco -n falco --create-namespace \
   -f values.yaml
```

values.yaml: include k8s_audit_rules.yaml and plugin

```
falco:
  rulesFile:
    - /etc/falco/falco_rules.yaml
    - /etc/falco/falco_rules.local.yaml
    - /etc/falco/k8s_audit_rules.yaml
    - /etc/falco/rules.d
  load_plugins: [k8saudit, json]
  jsonOutput: true
  priority: warning
```

values.yaml: enable Falco Sidekick

```
 # Publish to falco sidekick
  httpOutput:
    enabled: true
    url: "http://falcosidekick:2801/"

falcosidekick:
  replicaCount: 2
```

values.yaml – Use falcoctl for rule updates

```
falcoctl:
  artifact:
    install:
      enabled: true
    follow:
      enabled: true
  config:
    artifact:
      install:
        resolveDeps: false
        refs: [falco-rules:0, k8saudit-rules:0.5]
      follow:
        refs: [falco-rules:0, k8saudit-rules:0.5]
```

values.yaml – Use falcoctl for rule updates

```
falcoctl:
  artifact:
    install:
      enabled: true
    follow:
      enabled: true
  config:
    artifact:
      install:
        resolveDeps: false
        refs: [falco-rules:0, k8saudit-rules:0.5]
      follow:
        refs: [falco-rules:0, k8saudit-rules:0.5]
```

Alerts

Via Falco Sidekick, Falco can write alerts to flat files, send them to syslog, or send them to a ton of different integrations (Slack, Grafana,…)

You can even configure Falco for non-security purposes, just to have an easier way to parse out every change on the cluster.

https://github.com/falcosecurity/falcosidekick

There's even a web app dashboard: Falco Sidekick UI!

Falco Sidekick UI