Runtime security for your Kubernetes clusters

Jack Munday / IT-CD-PI

November 14th, 2024

Kubernetes is *not* secure by default

Threats can occur at many different levels.

- **Container**

  Over-privileged, privilege escalation, image vulnerabilities…

- **Host**

  Compromised host, Resource abuse in multi-tenant envs…

- **Internal & Perimeter Networking**

  Man-in-Middle, DOS, Lateral movement between pods…

- **Cluster**

  Unauthorised access, credential theft …

Each should be tackled independently to avoid large shifts in environments and make the process less overwhelming.

Kubernetes is *not* secure by default

Threats can occur at many different levels.

- **Container**

  Over-privileged, privilege escalation, image vulnerabilities…

- **Host**

  Compromised host, Resource abuse in multi-tenant envs…

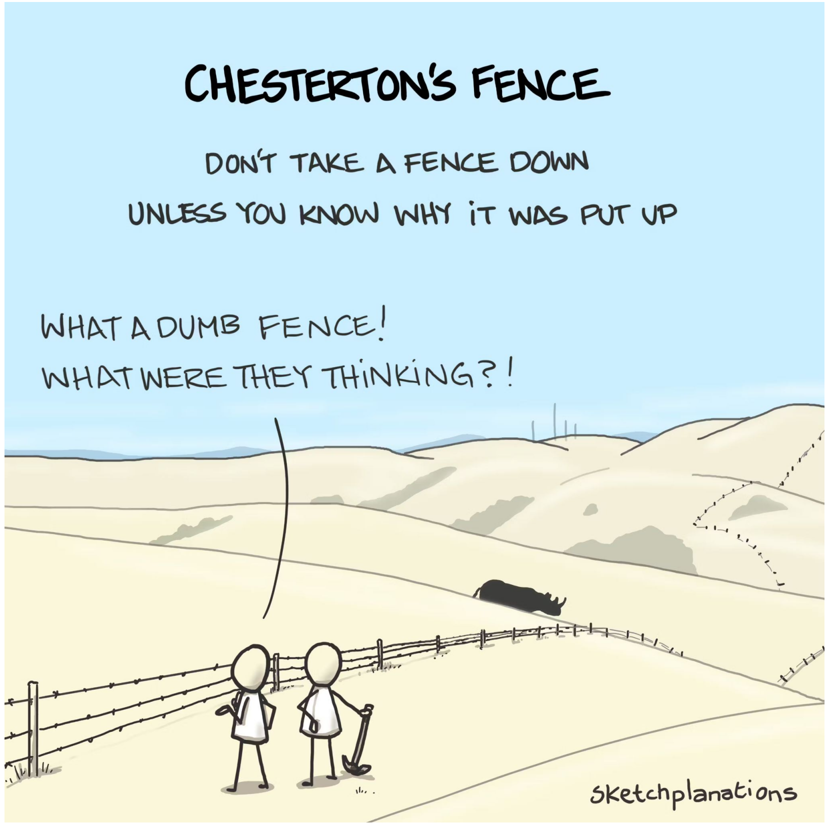
- **Internal & Perimeter Networking**

  Man-in-Middle, DOS, Lateral movement between pods…

- **Cluster**

  Unauthorised access, credential theft …

Each should be tackled independently to avoid large shifts in environments and make the process less overwhelming.


CHESTERTON'S FENCE
DON'T TAKE A FENCE DOWN
UNLESS YOU KNOW WHY IT WAS PUT UP
WHAT A DUMB FENCE!
WHAT WERE THEY THINKING?!
sketchplanations

Getting Started

Building your docker images

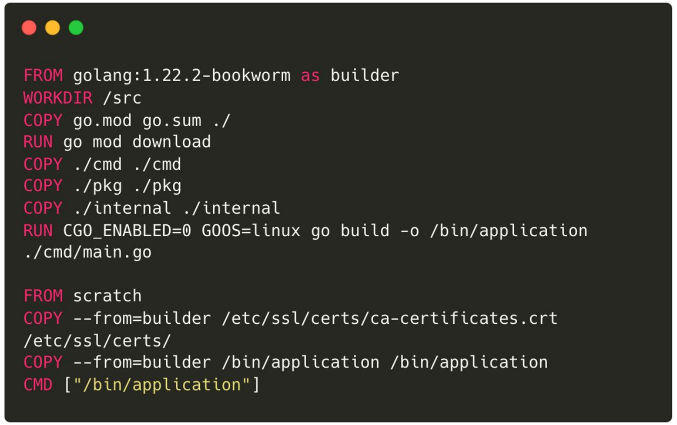
Avoid packaging unrequired tooling into containers using multi-stage builds.

- **Scratch for compiled languages.**
- **Distroless for interpreted languages.**

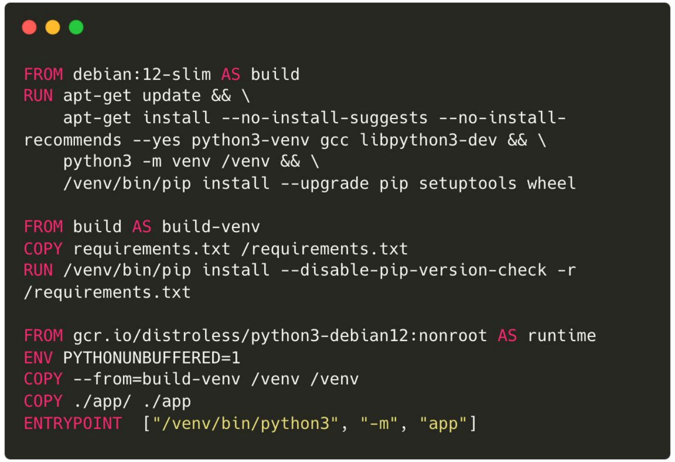
Removing a shell limits an attackers ability to use your container for anything other than its intended purpose.

Using multistage builds in this way will typically ensure your images are layered appropriately and can also reduce the number of vulnerabilities.

https://kubernetes.docs.cern.ch/docs/containers/base-images/



```
FROM golang:1.22.2-bookworm as builder
WORKDIR /src
COPY go.mod go.sum ./
RUN go mod download
COPY ./cmd ./cmd
COPY ./pkg ./pkg
COPY ./internal ./internal
RUN CGO_ENABLED=0 GOOS=linux go build -o /bin/application
./cmd/main.go

FROM scratch
COPY --from=builder /etc/ssl/certs/ca-certificates.crt
/etc/ssl/certs/
COPY --from=builder /bin/application /bin/application
CMD ["/bin/application"]
```



```
FROM debian:12-slim AS build
RUN apt-get update && \
    apt-get install --no-install-suggests --no-install-
recommends --yes python3-venv gcc libpython3-dev && \
    python3 -m venv /venv && \
    /venv/bin/pip install --upgrade pip setuptools wheel

FROM build AS build-venv
COPY requirements.txt /requirements.txt
RUN /venv/bin/pip install --disable-pip-version-check -r
/requirements.txt

FROM gcr.io/distroless/python3-debian12:nonroot AS runtime
ENV PYTHONUNBUFFERED=1
COPY --from=build-venv /venv /venv
COPY ./app/ ./app
ENTRYPOINT  ["/venv/bin/python3", "-m", "app"]
```

Security Contexts can then be used to restrict adding tooling to containers

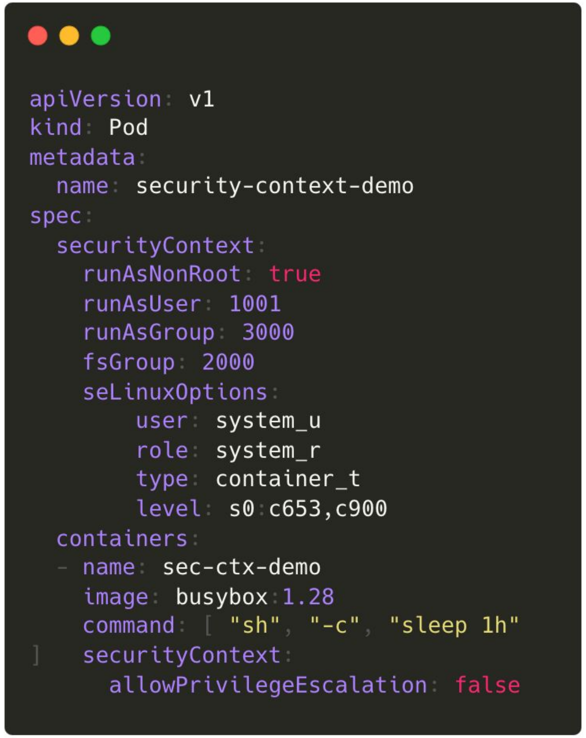
Ensure that (if compromised) additional tooling can not be added into a container.

Permissions can be restricted using Security Contexts.

- **SeLinux profile specified & SeLinux enabled on the host.**
- **Running as non-root**
- **Read-only File Systems**
- **Running with a UID > 1000 (i.e. default no privileges).**

Contexts can be applied at both the pod and container level.

Can be harder to implement; greater security posture



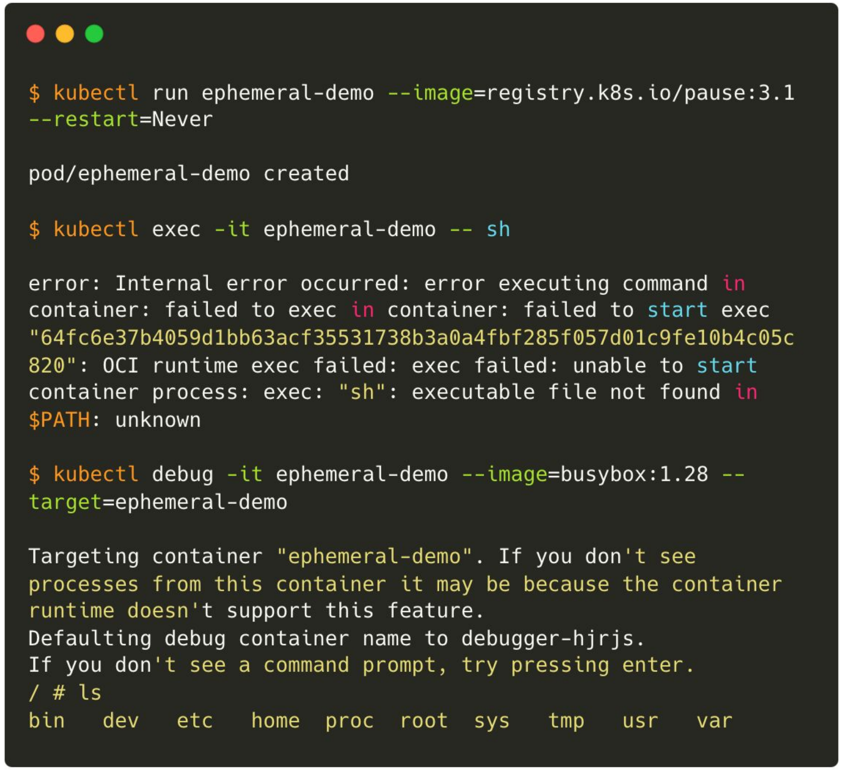
```
apiVersion: v1
kind: Pod
metadata:
  name: security-context-demo
spec:
  securityContext:
    runAsNonRoot: true
    runAsUser: 1001
    runAsGroup: 3000
    fsGroup: 2000
    seLinuxOptions:
        user: system_u
        role: system_r
        type: container_t
        level: s0:c653,c900
  containers:
  - name: sec-ctx-demo
    image: busybox:1.28
    command: [ "sh", "-c", "sleep 1h"
]   securityContext:
      allowPrivilegeEscalation: false
```

So now how do I actually debug anything?

Removing tooling from images makes it harder for attackers to exploit, but also harder for you to debug.

Kubernetes offers *ephemeral containers* to solve this.

Allows you to attach another image to an existing one sharing its network stack, process and filesystem.



```
$ kubectl run ephemeral-demo --image=registry.k8s.io/pause:3.1
--restart=Never

pod/ephemeral-demo created

$ kubectl exec -it ephemeral-demo -- sh

error: Internal error occurred: error executing command in
container: failed to exec in container: failed to start exec
"64fc6e37b4059d1bb63acf35531738b3a0a4fbf285f057d01c9fe10b4c05c
820": OCI runtime exec failed: exec failed: unable to start
container process: exec: "sh": executable file not found in
$PATH: unknown

$ kubectl debug -it ephemeral-demo --image=busybox:1.28 --
target=ephemeral-demo

Targeting container "ephemeral-demo". If you don't see
processes from this container it may be because the container
runtime doesn't support this feature.
Defaulting debug container name to debugger-hjrjs.
If you don't see a command prompt, try pressing enter.
/ # ls
bin   dev   etc   home  proc  root  sys   tmp   usr   var
```

Demo: Securely connecting to DBOD from a python image

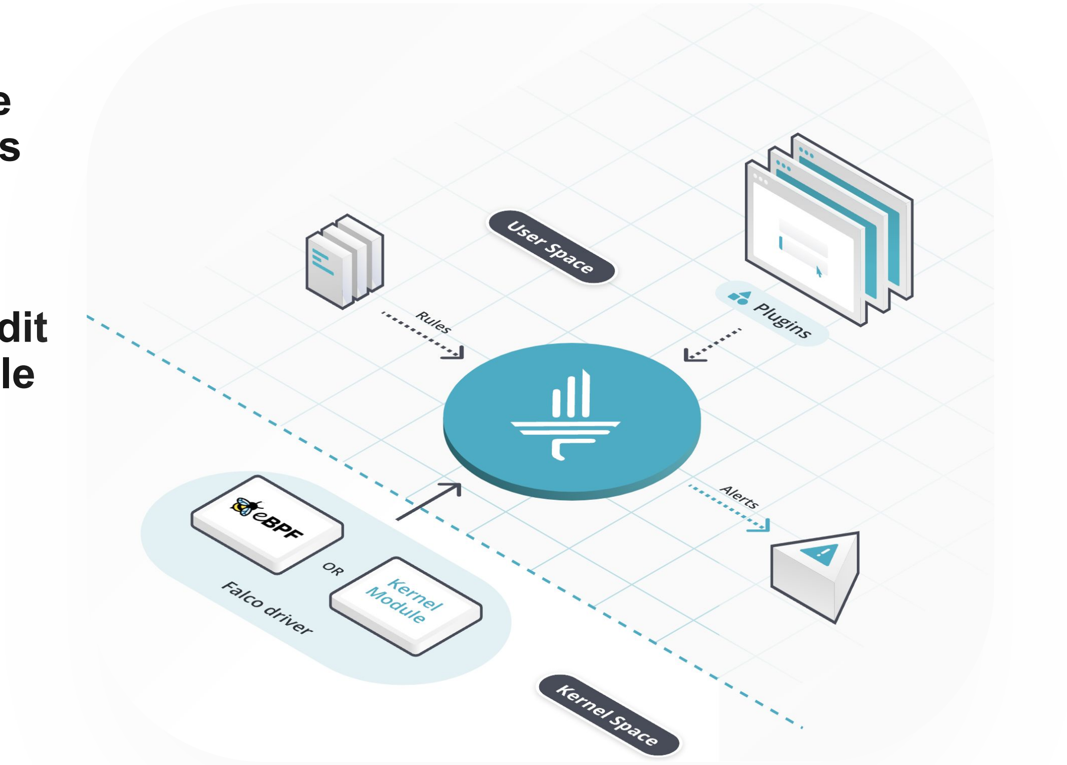
Understanding when you are compromised

Monitor for abnormal behaviour with `falco`

Cloud native tool that provides runtime security in VMs, containers, kubernetes and cloud environments.

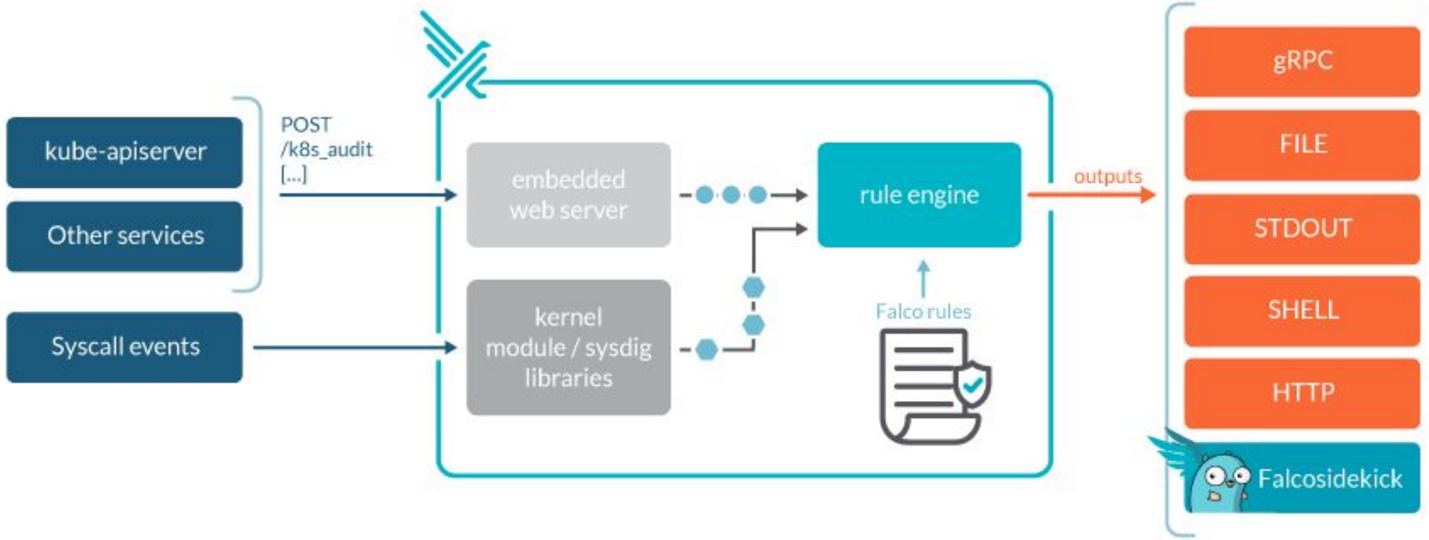
Monitors kernel events, kubernetes audit logs and a number of other configurable sources.

Alerts can then be forward to external providers for investigation / response (`alertmanager`, `pagerduty` etc…)



ref: https://falco.org/img/falco-schema.svg

Falco Architecture



ref: https://sysdig.com/blog/intro-runtime-security-falco

`falco` is distributed at CERN at part of templates for >= 1.31.x

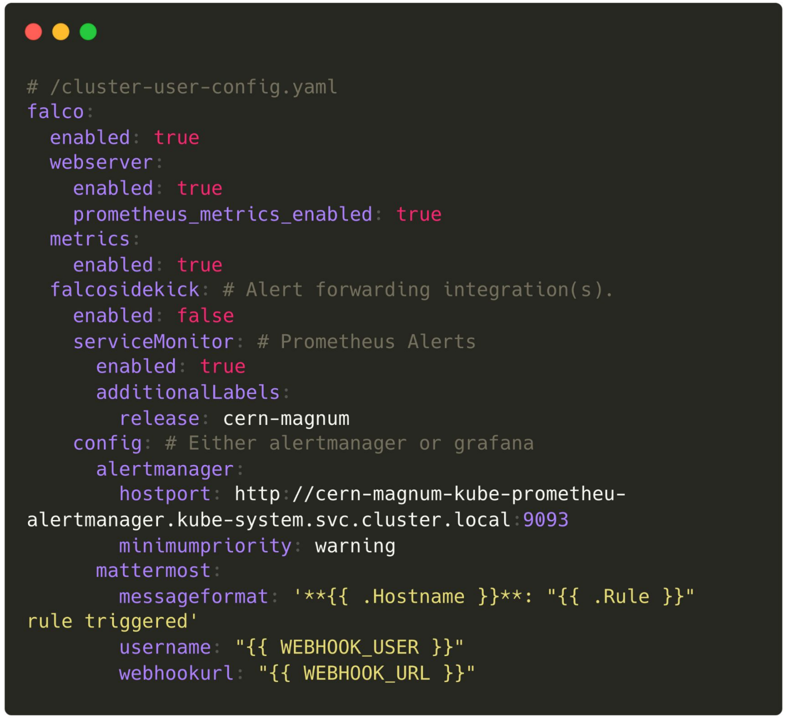
Falco comes pre-installed in CERN clusters, with event forwarding to `STDOUT` only.

i.e. `kubectl -n kube-system logs daemonset cern-magnum-falco`

Configuring an alerting source must be done manually on cluster creation:

- `alertmanager` *(recommended)*
- `prometheus`
- `mattermost`

Falco's default alerts are enabled in our distribution.



```
# /cluster-user-config.yaml
falco:
  enabled: true
  webserver:
    enabled: true
    prometheus_metrics_enabled: true
  metrics:
    enabled: true
  falcosidekick: # Alert forwarding integration(s).
    enabled: false
    serviceMonitor: # Prometheus Alerts
      enabled: true
      additionalLabels:
        release: cern-magnum
    config: # Either alertmanager or grafana
      alertmanager:
        hostport: http://cern-magnum-kube-prometheu-
alertmanager.kube-system.svc.cluster.local:9093
        minimumpriority: warning
      mattermost:
        messageformat: '**{{ .Hostname }}**: "{{ .Rule }}"
rule triggered'
        username: "{{ WEBHOOK_USER }}"
        webhookurl: "{{ WEBHOOK_URL }}"
```

```
$ openstack coe cluster create ... --merge-labels --labels cern_chart_user_values="$(cat
/cluster-user-config.yaml | base64 -w0)" my-falco-cluster
```

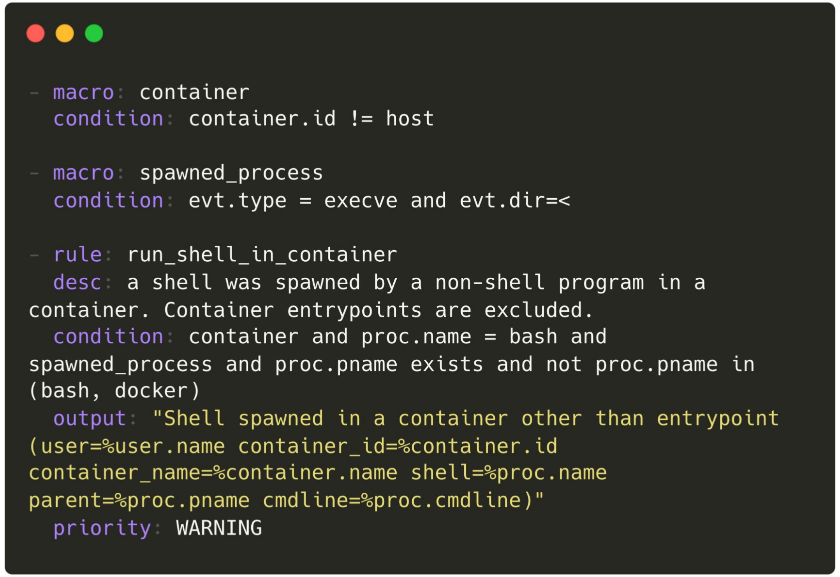
`falco` is alert focusing, not preventative

`falco` does not block or prevent any actions.

Provides insights on the low level behaviours in your environments, to inform organisational policy definitions.

Crafting rules can be complex, requiring a good understanding of low level behaviours.

Fortunately falco comes with a wide range of sensible default rules (for alerting).



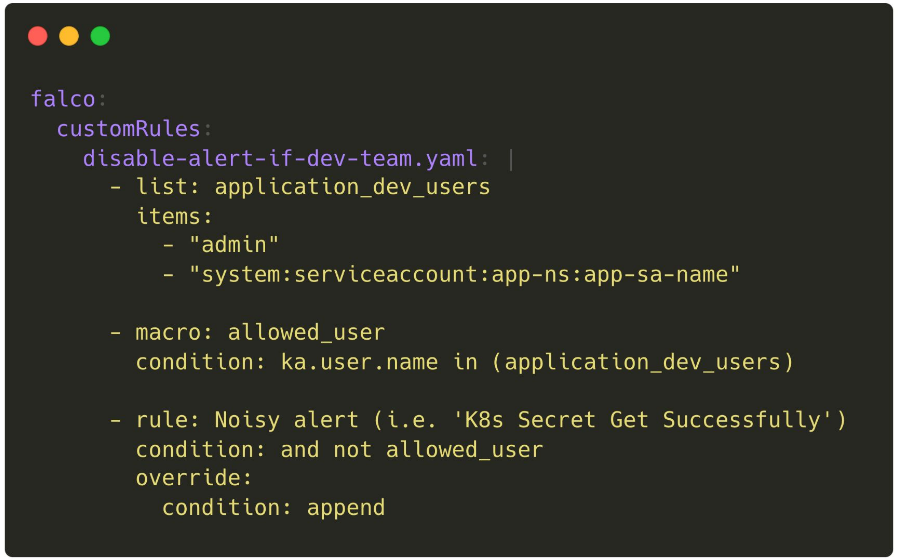
```
- macro: container
  condition: container.id != host

- macro: spawned_process
  condition: evt.type = execve and evt.dir=<

- rule: run_shell_in_container
  desc: a shell was spawned by a non-shell program in a
container. Container entrypoints are excluded.
  condition: container and proc.name = bash and
spawned_process and proc.pname exists and not proc.pname in
(bash, docker)
  output: "Shell spawned in a container other than entrypoint
(user=%user.name container_id=%container.id
container_name=%container.name shell=%proc.name
parent=%proc.pname cmdline=%proc.cmdline)"
  priority: WARNING
```

Writing your own alerts

Clusters by default have only the `falco_rules.yaml` rules enabled from the falcosecurity/rules repository.

You can choose to write your own alerts or override behaviours of existing alerts via a yaml snippet.

Rules take a layering approach to allow one to easily override behaviour without needing to rewrite a whole set of rules.



```yaml
falco:
  customRules:
    disable-alert-if-dev-team.yaml: |
      - list: application_dev_users
        items:
          - "admin"
          - "system:serviceaccount:app-ns:app-sa-name"

      - macro: allowed_user
        condition: ka.user.name in (application_dev_users)

      - rule: Noisy alert (i.e. 'K8s Secret Get Successfully')
        condition: and not allowed_user
        override:
          condition: append
```

Demo: Setting up Alerting with Falco

`https://gitlab.cern.ch/jmunday/webinars`

Next Steps: Preventative Measures

- ✱ Working to integrate the `falco` setup inside of `cern-magnum` with central monit.
- ✱ Expanding on default alerting rules to cover a wider range of scenarios.
- ✱ Preventative measures can be achieved using Admission Controllers
  - `Open Policy Agent Gatekeeper, Kyverno, etc …`

For more information, please visit:

https://kubernetes.docs.cern.ch/docs/security/falco
https://falco.org/docs/