KPLABS Course

Certified Kubernetes Security Specialist

Monitoring, Logging, and Runtime Security

ISSUED BY
Zeal

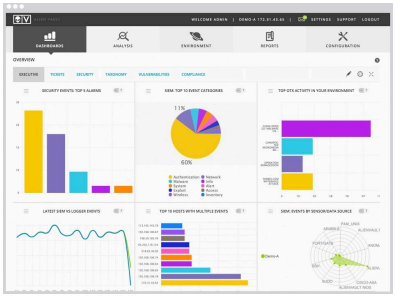
REPRESENTATIVE
instructors@kplabs.in

Module 1: Overview of Falco

1.1 Detection and Prevention

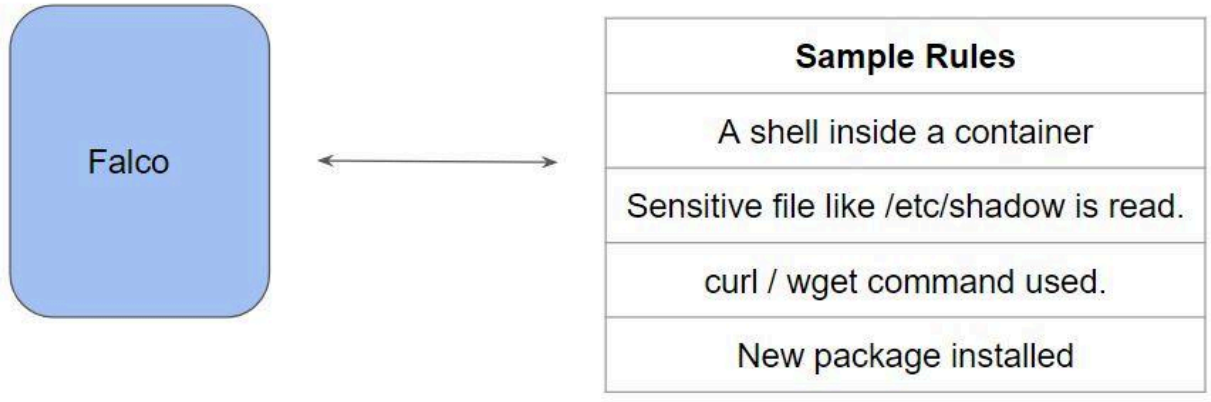
In Kubernetes, various features like Network Policies, RBAC, and others are primarily used for prevention purposes.

On the other hand, it is also important to add a certain set of detection measures that allows us to see what exactly is happening within the environment.



1.2 Basics of Falco

Falco is an open-source-based security tool that allows users to define a set of rules that will trigger an alert whenever the conditions are met.



1.3 Basic Rule Format

Falco rules are written in YAML and have a variety of required and optional keys.

rule	Name of the rule.
desc	Description of what the rule is filtering for.
condition	The logic statement that triggers a notification.
output	The message that will be shown in the notification.
priority	The “logging level” of the notification.

1.4 Sample Rule

Alerts whenever there is a shell spawned inside a container.

```
- rule: Detect bash in a container
  desc: You shouldn't have a shell run in a container
  condition: container.id != host and proc.name = bash
  output: Bash ran inside a container (user=%user.name command=%proc.cmdline %container.info)
  priority: INFO
```

Module 2: Overview of Sysdig

2.1 Basics of Sysdig:

In a normal scenario of troubleshooting and performance monitoring, we make use of the following tools

Sysdig offers the functionality of these tools along with a lot more.

strace	Discovering system calls
tcpdump	Network traffic monitoring
lsof	Files are opened by which process.
netstat	Network Connection monitoring
htop	Process Monitoring
iftop	Network Bandwidth monitoring

2.2 Interactive Options

Sysdig Utility comes with a command-line option (sysdig) as well as interface UI (csysdig)

```
Viewing: Processes For: whole machine
Source: Live System Filter: evt.type!=switch
   PID     CPU USER          TH     VIRT      RES     FILE      NET Command
123827    6.50 root           6     577M      159M       0    0.00 csysdig
102296    6.00 root           9       1G      334M       0  15.22K kube-apiserver --advertise-address=172.31.59.221
   471    3.00 root          18       2G      107M       0   3.39K /usr/bin/kubelet --bootstrap-kubeconfig=/etc/kube
102255    2.50 root           7     793M      100M       0  17.89K kube-controller-manager --allocate-node-cidrs=tru
   663    1.50 root          27       1G      114M       0    0.00 /usr/bin/dockerd -H fd:// --containerd=/run/conta
  1966    1.50 root          14       2G       73M     70K   2.98K etcd --advertise-client-urls=https://172.31.59.22
  4714    0.50 root           9     730M       38M       0  893.50 /coredns -conf /etc/coredns/Corefile
   491    0.50 root          33       1G       49M       0    0.00 /usr/bin/containerd
  3550    0.00 root           9       1G       36M       0    0.00 /opt/bin/flanneld --ip-masq --kube-subnet-mgr
123819    0.00 ubuntu         1       6M        4M       0    0.00 /usr/lib/openssh/sftp-server
  1689    0.00 root          10     106M        6M       0    0.00 containerd-shim -namespace moby -workdir /var/lib
   488    0.00 root           1      17M        8M       0    0.00 /lib/systemd/systemd-logind
  4368    0.00 root          10     106M        5M       0    0.00 containerd-shim -namespace moby -workdir /var/lib
  4353    0.00 root          10     106M        5M       0    0.00 containerd-shim -namespace moby -workdir /var/lib
102256    0.00 root           9     729M       46M       0   1.51K kube-scheduler --authentication-kubeconfig=/etc/k
   534    0.00 www-data       1      57M        5M       0    0.00 nginx: worker process
  3265    0.00 root          10     106M        6M       0    0.00 containerd-shim -namespace moby -workdir /var/lib
   512    0.00 root           1       6M        2M       0    0.00 /sbin/agetty -o -p -- \u --noclear tty1 linux
  1641    0.00 root          11     853M       28M       0    0.00 /snap/amazon-ssm-agent/2996/ssm-agent-worker
  3123    0.00 root           1     964K        4K       0    0.00 /pause
   458    0.00 root           1       8M        3M       0    0.00 /usr/sbin/cron -f
```

2.3 Running sysdig

In its simplest form, when you run sysdig, you will see all the system calls that are happening within the system.

```
52 07:50:42.829676193 1 <NA> (0) > switch next=1157782 pgft_maj=0 pgft_min=0 vm_size=0 vm_rss=0 vm_swap=0
54 07:50:42.829686696 0 sysdig (1171310) > switch next=1155766 pgft_maj=0 pgft_min=2183 vm_size=447872 vm_rss=25612
 vm_swap=0
55 07:50:42.829687819 1 <NA> (1157782) > switch next=1170533(sshd) pgft_maj=0 pgft_min=0 vm_size=0 vm_rss=0 vm_swap
=0
56 07:50:42.829690356 0 <NA> (1155766) > switch next=1171310(sysdig) pgft_maj=0 pgft_min=0 vm_size=0 vm_rss=0 vm_sw
ap=0
57 07:50:42.829696854 1 sshd (1170533) < select res=1
58 07:50:42.829697168 0 sysdig (1171310) > switch next=1155766 pgft_maj=0 pgft_min=2183 vm_size=447872 vm_rss=25612
 vm_swap=0
59 07:50:42.829699893 0 <NA> (1155766) > switch next=1171310(sysdig) pgft_maj=0 pgft_min=0 vm_size=0 vm_rss=0 vm_sw
ap=0
60 07:50:42.829702032 1 sshd (1170533) > rt_sigprocmask
61 07:50:42.829703224 1 sshd (1170533) < rt_sigprocmask
62 07:50:42.829704021 1 sshd (1170533) > rt_sigprocmask
63 07:50:42.829704472 1 sshd (1170533) < rt_sigprocmask
64 07:50:42.829707634 1 sshd (1170533) > clock_gettime
65 07:50:42.829708525 1 sshd (1170533) < clock_gettime
68 07:50:42.829710931 1 sshd (1170533) > read fd=15(<f>/dev/ptmx) size=16384
69 07:50:42.829714826 1 sshd (1170533) < read res=1201 data=1 07:50:42.736703000 0 container:0d65633084bf (-1) > co
ntainer json={"container"
```

2.4 Filters

Since you will get a huge amount of data when monitoring system calls, you can use sysdig with filters to make the output more fine-grained.

```
root@ip-172-31-59-221:~# sysdig proc.name=nano
310161 07:55:39.213391058 0 nano (1173625) < execve res=0 exe=nano args=test.txt. tid=1173625(nano) pid=1173625(nan
o) ptid=1173599(bash) cwd= fdlimit=1024 pgft_maj=1 pgft_min=24 vm_size=652 vm_rss=4 vm_swap=0 comm=nano cgroups=cpu
set=/.cpu=/user.slice.cpuacct=/user.slice.io=/user.slice.memory=/user.slic... env=SHELL=/bin/bash.SUDO_GID=1000.SUD
O_COMMAND=/usr/bin/su.SUDO_USER=ubuntu.PWD=/... tty=34822 pgid=1173625(nano) loginuid=1000
310162 07:55:39.213423440 0 nano (1173625) > brk addr=0
310163 07:55:39.213424690 0 nano (1173625) < brk res=563F7341E000 vm_size=652 vm_rss=4 vm_swap=0
310164 07:55:39.213434121 0 nano (1173625) > arch_prctl
310165 07:55:39.213434779 0 nano (1173625) < arch_prctl
310166 07:55:39.213459275 0 nano (1173625) > access mode=4(R_OK)
310167 07:55:39.213464446 0 nano (1173625) < access res=-2(ENOENT) name=/etc/ld.so.preload
310168 07:55:39.213469242 0 nano (1173625) > openat
310169 07:55:39.213476122 0 nano (1173625) > fstat fd=3
310170 07:55:39.213477566 0 nano (1173625) < fstat res=0
```

2.5 Sysdig Chisels

Sysdig's chisels are little scripts that analyze the sysdig event stream to perform useful actions

```
root@ip-172-31-59-221:~# sysdig -cl

Category: Application
---------------------
httplog         HTTP requests log
httptop         Top HTTP requests
memcachelog     memcached requests log

Category: CPU Usage
-------------------
spectrogram     Visualize OS latency in real time.
subsecoffset    Visualize subsecond offset execution time.
topcontainers_cpu
                Top containers by CPU usage
topprocs_cpu    Top processes by CPU usage

Category: Errors
----------------
topcontainers_error
                Top containers by number of errors
topfiles_errors Top files by number of errors
topprocs_errors top processes by number of errors
```

Module 3: Writing Falco Rules

3.1 Basic Rule Format

Falco rules are written in YAML and have a variety of required and optional keys.

rule	Name of the rule.
desc	Description of what the rule is filtering for.
condition	The logic statement that triggers a notification.
output	The message that will be shown in the notification.
priority	The “logging level” of the notification.

Here is a sample Falco rule:

```
- rule: Detect bash in a container
  desc: You shouldn’t have a shell run in a container
  condition: container.id != host and proc.name = bash
  output: Bash ran inside a container (user=%user.name command=%proc.cmdline %container.info)
  priority: INFO
```

3.2 Rule Elements

A Falco rules file is a YAML file containing three types of elements:

Elements	Description
Rules	Conditions under which an alert should be generated. A rule is accompanied by a descriptive output string that is sent with the alert.
Macros	Macros provide a way to name common patterns and factor out redundancies in rules.
Lists	Collections of items that can be included in rules, macros, or other lists

Module 4: Falco Rule Writing - Exam Perspective

XYZ use-case

Format as follows

time	uid	process-name

Capture log for 25 seconds and store it in /tmp/log.txt

Module 5: Audit Logging

5.1 Revising Auditing

Auditing provides a security-relevant, chronological set of records documenting the sequence of actions in a cluster.

- what happened?
- when did it happen?
- who initiated it?
- on what did it happen?
- from where was it initiated?
- to where was it going?


```
{
  "kind": "Event",
  "apiVersion": "audit.k8s.io/v1",
  "level": "Metadata",
  "auditID": "388ca4e1-c368-45b4-aca9-652e32baf8e1",
  "stage": "RequestReceived",
  "requestURI": "/api/v1/namespaces/default/secrets?limit=500",
  "verb": "list",
  "user": {
    "username": "bob",
    "groups": [
      "system:masters",
      "system:authenticated"
    ]
  },
  "sourceIPs": [
    "127.0.0.1"
  ],
  "userAgent": "kubectl/v1.19.0 (linux/amd64) kubernetes/e199641",
  "objectRef": {
    "resource": "secrets",
    "namespace": "default",
    "apiVersion": "v1"
  },
  "requestReceivedTimestamp": "2020-12-03T16:55:10.357789Z",
  "stageTimestamp": "2020-12-03T16:55:10.357789Z"
}
```


5.2 Audit Policy Levels

Audit policy defines rules about what events should be recorded and what data they should include.

Audit Levels	Description
None	don't log events that match this rule.
Metadata	Log request metadata (requesting user, timestamp, resource, verb, etc.) but not request or response body.
Request	Log event metadata and request body but not response body.
RequestResponse	Log event metadata, request and response bodies.

5.3 Stages

The kube-apiserver processes request in stages and each stage generates an audit event.

Stage	Description
RequestReceived	The stage for events generated as soon as the audit handler receives the request, and before it is delegated down the handler chain.
ResponseStarted	Once the response headers are sent, but before the response body is sent. This stage is only generated for long-running requests (e.g. watch).
ResponseComplete	The response body has been completed and no more bytes will be sent.
Panic	Events generated when a panic occurred.