

[Open in app](#)**Jonathan**[Follow](#)

29 Followers

[About](#)

# Certified Kubernetes Security Specialist (CKS) Preparation Part 8 — Runtime Security & System Hardening



Jonathan Mar 24 · 8 min read

If you have not yet checked the previous parts of this series, please go ahead and check [Part1](#), [Part2](#), [Part3](#), [Part4](#), [Part5](#), [Part6](#) and [Part7](#).

In this article, I would focus on the preparation around **runtime security** and **system hardening** in CKS certification exam.

## strace

The full name of strace is system call trace, which means it is a process that would act like a sidecar on system call interface and note down every syscall. [This Medium article](#) gives a pretty straightforward explanation on how most people use strace in practice and [this one](#) would give a thorough walk-through if anyone is interested in the details. At the end of the day, there are so many parameters you could use with strace and all really depends on the situation. For demonstration, we would use the commonly used ones when debugging Kubernetes operations.

One of the examples is to see whether we could read secrets stored in etcd. In order to achieve that, we would need to know the process ID etcd is running on.

```
#SSH into K8s master nodes
- ps aux | grep etcd
```



```
root@kali:~# cat /etc/kubernetes/pki/etcd/ca.conf
root 1112 9.9 5.4 1166584 444264 7 Ssl 13:57 0:37 kube-apiserver --audit-policy-file=/etc/kubernetes/audit/policy.yaml --audit-log-path=/etc/kubernetes/audit/logs/audit-log --audit-log-maxage=5 --audit-log-maxbackups=1 --audit-log-maxsize=198 --advertise-address=192.168.1.4 --allow-privilege-escalation=true --authorization-mode=Node,RBAC --client-ca-file=/etc/kubernetes/pki/ca.crt --enable-admission-plugins=NodeRestriction --enable-bootstrap-token-auth --etcd-cafile=/etc/kubernetes/pki/etcd/ca.crt --etcd-certfile=/etc/kubernetes/pki/apiserver-etcd-client.crt --etcd-keyfile=/etc/kubernetes/pki/apiserver-etcd-client.key --etcd-servers=https://127.0.0.1:2379 --insecure-port=80 --kubernetes-client-certificate=/etc/kubernetes/pki/apiserver-kubelet-client.crt --kubernetes-client-key=/etc/kubernetes/pki/apiserver-kubelet-client.key --kubernetes-preferred-address-types=InternalIP,ExternalIP,Hostname --proxy-client-cert-file=/etc/kubernetes/pki/front-proxy-client.crt --proxy-client-key-file=/etc/kubernetes/pki/front-proxy-client.key --requestheader-allowed-names=front-proxy-client --requestheader-client-ca-file=/etc/kubernetes/pki/front-proxy-ca.crt --requestheader-extra-headers-prefix=X-Remote-Extra- --requestheader-group-headers=X-Remote-Group --requestheader-username-headers=X-Remote-User --secure-port=8443 --service-account-issuer=https://kubernetes.default.svc.cluster.local --service-account-key-file=/etc/kubernetes/pki/sa.pub --service-account-signing-key-file=/etc/kubernetes/pki/sa.key --service-cluster-ip-range=192.168.0.0/12 --tls-cert-file=/etc/kubernetes/pki/apiserver.crt --tls-private-key-file=/etc/kubernetes/pki/apiserver.key
jonn 12654 0.0 0.0 14360 1632 pts/0 S+ 14:03 0:00 grep etcd
```

```
sudo strace -p 4295
```

```

jonw@CHS-Master:~$ strace -p 4295
strace: Could not attach to process. If your uid matches the uid of the target process, check the setting of /proc/sys/kernel/yama/ptrace_scope, or try a
gain as the root user. For more details, see /etc/sysctl.d/10-ptrace.conf: Operation not permitted
strace: attach: ptrace(PTRACE_SEIZE, 4295): Operation not permitted
jonw@CHS-Master:~$ sudo strace -p 4295
[sudo] password for jonw:
strace: Process 4295 attached
restart_syscall(<... resuming interrupted futex ...>) = -1 ETIMEDOUT (Connection timed out)
futex(0x1ac6130, FUTEX_WAKE_PRIVATE, 1) = 1
futex(0x1ac6030, FUTEX_WAKE_PRIVATE, 1) = 1
futex(0xc00005e4c8, FUTEX_WAKE_PRIVATE, 1) = 1
futex(0x1aca000, FUTEX_WAIT_PRIVATE, 0, {tv_sec=0, tv_nsec=17949143}) = -1 ETIMEDOUT (Connection timed out)
futex(0x1ac6130, FUTEX_WAKE_PRIVATE, 1) = 1
futex(0x1ac6030, FUTEX_WAKE_PRIVATE, 1) = 1
futex(0xc00005e4c8, FUTEX_WAKE_PRIVATE, 1) = 1
futex(0xc0001e8848, FUTEX_WAKE_PRIVATE, 1) = 1
futex(0x1aca000, FUTEX_WAIT_PRIVATE, 0, {tv_sec=0, tv_nsec=1185592}) = -1 ETIMEDOUT (Connection timed out)
futex(0x1ac6130, FUTEX_WAKE_PRIVATE, 1) = 1
futex(0x1ac6030, FUTEX_WAKE_PRIVATE, 1) = 1
futex(0xc00005e4c8, FUTEX_WAKE_PRIVATE, 1) = 1
futex(0xc0001e8848, FUTEX_WAKE_PRIVATE, 1) = 1
futex(0x1aca000, FUTEX_WAIT_PRIVATE, 0, {tv_sec=0, tv_nsec=80489054}) = -1 ETIMEDOUT (Connection timed out)
futex(0x1ac6130, FUTEX_WAKE_PRIVATE, 1) = 1
futex(0xc0001e8848, FUTEX_WAKE_PRIVATE, 1) = 1
futex(0x1aca000, FUTEX_WAIT_PRIVATE, 0, {tv_sec=0, tv_nsec=39938618}) = -1 ETIMEDOUT (Connection timed out)

```

```
- sudo su
- cd /proc/4295/fd
- ls -l | grep 7
```

```
root@CKS-Master:/proc/4295/fd# ls -l | grep 7
lrwx----- 1 root root 64 Mar 22 13:57 0 -> /dev/null
l-wx----- 1 root root 64 Mar 22 13:57 1 -> pipe:[48395]
l-wx----- 1 root root 64 Mar 22 13:57 10 -> /var/lib/etcd/member/wal/0.tmp
lrwx----- 1 root root 64 Mar 22 13:57 11 -> socket:[53287]
lrwx----- 1 root root 64 Mar 22 13:57 12 -> socket:[54338]
lrwx----- 1 root root 64 Mar 22 13:58 17 -> socket:[54346]
l-wx----- 1 root root 64 Mar 22 13:57 2 -> pipe:[48396]
lrwx----- 1 root root 64 Mar 22 13:58 27 -> socket:[54364]
lrwx----- 1 root root 64 Mar 22 13:57 3 -> socket:[49473]
```



```
lrwx—— 1 root root 64 Mar 22 13:57 36 → socket:[54379]
lrwx—— 1 root root 64 Mar 22 13:58 37 → socket:[54382]
lrwx—— 1 root root 64 Mar 22 13:58 38 → socket:[53376]
lrwx—— 1 root root 64 Mar 22 13:58 39 → socket:[53379]
lrwx—— 1 root root 64 Mar 22 13:57 4 → anon_inode:[eventpoll]
lrwx—— 1 root root 64 Mar 22 13:58 43 → socket:[53387]
lrwx—— 1 root root 64 Mar 22 13:58 47 → socket:[53399]
lrwx—— 1 root root 64 Mar 22 13:57 5 → socket:[49539]
lrwx—— 1 root root 64 Mar 22 13:58 57 → socket:[53425]
lrwx—— 1 root root 64 Mar 22 13:57 6 → socket:[49540]
lrwx—— 1 root root 64 Mar 22 13:58 63 → socket:[53437]
lrwx—— 1 root root 64 Mar 22 13:58 67 → socket:[53446]
lrwx—— 1 root root 64 Mar 22 13:57 7 → /var/lib/etcd/member/snap/db
lrwx—— 1 root root 64 Mar 22 13:58 70 → socket:[53455]
lrwx—— 1 root root 64 Mar 22 13:58 71 → socket:[53458]
lrwx—— 1 root root 64 Mar 22 13:58 72 → socket:[53471]
```

At this point, it seems like directory “7” contains the information that K8s needs. We could create a simple secret and try to locate the value within it.

```
- kubectl create secret generic credit-card --from-literal
ssecret=1111222233334444
```

```
#Make sure you are still in directory /proc/4295/fd.
```

#"-A10" and "-B10" mean show 10 lines before and after the searching string.

```
- cat 7 | strings | grep 1111222233334444 -A10 -B10
```

```
/cks-master_1e3fca22-95e3-4437-a4ed-6472494ab89e
%/registry/secrets/default/credit-card
Secret
credit-card
default"
*$7e8f411b-1ab4-4d17-b9db-24145606acba2
kubectl create
Update
FieldsV1:+
){"f:data":{".":{"f:cc":{}},"f:type":{}}
1111222233334444
Opaque
7/registry/services/endpoints/kube-system/kube-scheduler
Endpoints
kube-scheduler
kube-system"
*$7a2f0a7d-d471-421c-9149-41ce39683adb2
(control-plane.alpha.kubernetes.io/leader
{"holderIdentity":"cks-master_1586a4e0-70fa-4519-9548-fdb1
2020-11-01T16:41:56Z","leaderTransitions":1}z
```



[Open in app](#)

Quoted from [Udemy — Kubernetes CKS 2021 Complete Course](#)

**\*\*As my testing environment suddenly breaks, using the CKS course screen captures to show the expected result when getting into etcd's data.\*\***

## Falco Installation and Use Scenarios

Falco is a CNCF project which is invented to trace all Kubernetes administrators' actions. In fact, it could be both a great auditing tool and also a rule engine to identify any violation.

Falco could be installed to K8s through standalone mode or daemonset mode (having a Pod running in every node). In this example, we would install this service through standalone mode. Head [here](#) for more details.

After installation, check the service status after starting.

```
root@CKS-Worker:/home/jonw# systemctl start falco
root@CKS-Worker:/home/jonw# systemctl status falco
● falco.service - LSB: Falco syscall activity monitoring agent
   Loaded: loaded (/etc/init.d/falco; generated)
   Active: active (running) since Sun 2021-02-07 16:51:57 UTC; 1s ago
     Docs: man:systemd-sysv-generator(8)
  Process: 38778 ExecStart=/etc/init.d/falco start (code=exited, status=0/SUCCESS)
    Tasks: 10 (limit: 4915)
   CGroup: /system.slice/falco.service
           └─38889 /usr/bin/falco --daemon --pidfile=/var/run/falco.pid

Feb 07 16:51:57 CKS-Worker falco[38798]: Loading rules from file /etc/falco/falco_rules.yaml:
Feb 07 16:51:57 CKS-Worker falco[38798]: Loading rules from file /etc/falco/falco_rules.local.yaml:
Feb 07 16:51:57 CKS-Worker falco[38798]: Sun Feb  7 16:51:57 2021: Loading rules from file /etc/falco/falco_rules.local.yaml:
Feb 07 16:51:57 CKS-Worker falco[38798]: Loading rules from file /etc/falco/k8s_audit_rules.yaml:
Feb 07 16:51:57 CKS-Worker falco[38798]: Sun Feb  7 16:51:57 2021: Loading rules from file /etc/falco/k8s_audit_rules.yaml:
Feb 07 16:51:57 CKS-Worker falco[38809]: Started LSB: Falco syscall activity monitoring agent.
Feb 07 16:51:57 CKS-Worker falco[38809]: Starting internal webserver, listening on port 8765
Feb 07 16:51:57 CKS-Worker falco[38809]: 16:51:57.777298000: Notice Privileged container started (user=root user_loginuid=0 command=container:
Feb 07 16:51:57 CKS-Worker falco[38809]: 16:51:57.777066000: Notice Privileged container started (user=<NA> user_loginuid=0 command=container:
Feb 07 16:51:57 CKS-Worker falco[38809]: 16:51:57.802682000: Notice Privileged container started (user=<NA> user_loginuid=0 command=container:
lines 1-19/19 (END)
```

Falco rules would be located in /etc/falco.

```
root@CKS-Worker:/etc/falco# ls
falco.yaml  falco_rules.local.yaml  falco_rules.yaml  k8s_audit_rules.yaml  rules.available  rules.d
```

Let's give it a test by performing administrative actions, such as writing additional information inside /etc/<whatever file>, inside a Pod (container) and see what logs

[Open in app](#)

- kubectl exec test -it -- bash
- echo user >> /etc/passwd

Traced Logs (shown in /var/log/syslog. search with keyword “falco”)

```

jonw@CKS-Worker:~$ sudo tail -f /var/log/syslog | grep falco
Feb  7 16:54:01 CKS-Worker falco: 16:54:01.926149567: Error File below /etc opened for writing (user=omsagent user_loginuid=998 command=Status
Report.sh /opt/microsoft/omsconfig/Scripts/StatusReport.sh 312E76F1-5108-4C26-8765-CAA7F6DDB602 StartTime parent=sh pcmdline=sh -c /opt/micros
oft/omsconfig/Scripts/StatusReport.sh 312E76F1-5108-4C26-8765-CAA7F6DDB602 StartTime file=/etc/opt/omi/conf/omsconfig/last_statusreport progra
m=StatusReport.sh gparent=dsc_host gpparent=python2 gggparent=sh container_id=host image=<NA>)
Feb  7 16:54:03 CKS-Worker falco: 16:54:03.423903336: Error File below /etc opened for writing (user=omsagent user_loginuid=998 command=Status
Report.sh /opt/microsoft/omsconfig/Scripts/StatusReport.sh 312E76F1-5108-4C26-8765-CAA7F6DDB602 EndTime parent=sh pcmdline=sh -c /opt/microsof
t/omsconfig/Scripts/StatusReport.sh 312E76F1-5108-4C26-8765-CAA7F6DDB602 EndTime file=/etc/opt/omi/conf/omsconfig/last_statusreport program=St
atusReport.sh gparent=dsc_host gpparent=python2 gggparent=sh container_id=host image=<NA>)
Feb  7 16:55:48 CKS-Worker falco: 16:55:48.743996076: Notice A shell was spawned in a container with an attached terminal (user=root user_logi
nuid=-1 k8s_test_test_default_c9fc92d9-971d-4113-bf64-be6773d28003_1 (id=137e2d04e4f5) shell=bash parent=runc cndline=bash terminal=34816 cont
ainer_id=137e2d04e4f5 image=nginx)
Feb  7 16:56:01 CKS-Worker falco: 16:56:00.992188215: Error File below /etc opened for writing (user=root user_loginuid=-1 command=bash parent
=<NA> pcmdline=<NA> file=/etc/passwd program=bash gparent=<NA> gpparent=<NA> gggparent=<NA> container_id=137e2d04e4f5 image=nginx)

```

## Falco Rule Management

All default Falco rules are in /etc/falco/falco\_rules.yaml. If administrators would like to check how each rule works, we could use text editor (vim/nano) to view the content.

```

GNU nano 2.9.3 falco_rules.yaml

#
# Copyright (C) 2020 The Falco Authors.
#
#
# Licensed under the Apache License, Version 2.0 (the "License");
# you may not use this file except in compliance with the License.
# You may obtain a copy of the License at
#
#     http://www.apache.org/licenses/LICENSE-2.0
#
# Unless required by applicable law or agreed to in writing, software
# distributed under the License is distributed on an "AS IS" BASIS,
# WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
# See the License for the specific language governing permissions and
# limitations under the License.
#

# See xxx for details on falco engine and rules versioning. Currently,
# this specific rules file is compatible with engine version 0
# (e.g. falco releases ≤ 0.13.1), so we'll keep the
# required_engine_version lines commented out, so maintain
# compatibility with older falco releases. With the first incompatible
# change to this rules file, we'll uncomment this line and set it to

```

There are 2 essential ways of changing Falco rules, one by *changing the rule found in falco\_rules.yaml directory* and the other by *adding the new rule in falco\_rules.local.yaml for overwriting the existing ones.*



Open in app



```
GNU nano 2.9.3                                falco_rules.yaml                                Modified
# output: "Interactive root (%user.name %proc.name %evt.dir %evt.type %evt.args %fd.name)"
# priority: WARNING

- rule: System user interactive
  desc: an attempt to run interactive commands by a system (i.e. non-login) user
  condition: spawned_process and system_users and interactive and not user_known_system_user_login
  output: "System user ran an interactive command (user=%user.name user_loginuid=%user.loginuid command=%proc.cmdline container_id=%container.id)"
  priority: INFO
  tags: [users, mitre_remote_access_tools]

# In some cases, a shell is expected to be run in a container. For example, configuration
# management software may do this, which is expected.
- macro: user_expected_terminal_shell_in_container_conditions
  condition: (never_true)

- rule: Terminal shell in container
  desc: A shell was used as the entrypoint/exec point into a container with an attached terminal.
  condition: >
    spawned_process and container
    and shell_procs and proc.tty != 0
    and container_entrypoint
    and not user_expected_terminal_shell_in_container_conditions
  output: >
    A shell was spawned in a container with an attached terminal (user=%user.name user_loginuid=%user.loginuid %container.info
    shell=%proc.name parent=%proc.pname cmdline=%proc.cmdline terminal=%proc.tty container_id=%container.id image=%container.image.repository)
  priority: NOTICE
  tags: [container, shell, mitre_execution]
```

Copy the content and append to the bottom of falco\_rules.local.yaml

```
# You may obtain a copy of the License at
#
# http://www.apache.org/licenses/LICENSE-2.0
#
# Unless required by applicable law or agreed to in writing, software
# distributed under the License is distributed on an "AS IS" BASIS,
# WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
# See the License for the specific language governing permissions and
# limitations under the License.
#

#####
# Your custom rules!
#####

# Add new rules, like this one
# - rule: The program "sudo" is run in a container
#   desc: An event will trigger every time you run sudo in a container
#   condition: evt.type = execve and evt.dir=< and container.id != host and proc.name = sudo
#   output: "Sudo run in container (user=%user.name %container.info parent=%proc.pname cmdline=%proc.cmdline)"
#   priority: ERROR
#   tags: [users, container]

# Or override/append to any rule, macro, or list from the Default Rules

- rule: Terminal shell in container
  desc: A shell was used as the entrypoint/exec point into a container with an attached terminal.
  condition: >
    spawned_process and container
    and shell_procs and proc.tty != 0
    and container_entrypoint
    and not user_expected_terminal_shell_in_container_conditions
  output: >
    %evt.time,%user.name,%container.name,%container.id
  priority: WARNING
  tags: [container, shell, mitre_execution]
```

Perform the same action like executing into a terminal within Pod (container) and check the worker node's /var/log/syslog to see what is showing.

```
jon@CKS-Worker:/etc/falco$ sudo tail -f /var/log/syslog | grep falco
Feb  7 17:08:53 CKS-Worker falco: Starting internal webserver, listening on port 8765
Feb  7 17:08:53 CKS-Worker falco: 17:08:53.711459800: Notice Privileged container started (user=<NA> user_loginuid=0 command=container:be8a626bd8c3 k8s_weave_weave-net-ltsbj_kube-system_dec8a525-7d52-4c56-915f-c8d19114e112_u (id=be8a626bd8c3) image=weaveworks/weave-kube:2.8.1)
```

[Open in app](#)


```

m=StatusReport.sh gpparent=dsc_host ggpparent=python2 gggparent=sh container_id=host image=<NA>)
Feb  7 17:09:03 CKS-Worker falco: 17:09:03.683436384: Error File below /etc opened for writing (user=omsagent user_loginuid=998 command=Status
Report.sh /opt/microsoft/omsconfig/Scripts/StatusReport.sh 5171FE91-45DC-489F-B456-9FF0138C391C EndTime parent=sh pcmdline=sh -c /opt/microsof
t/omsconfig/Scripts/StatusReport.sh 5171FE91-45DC-489F-B456-9FF0138C391C EndTime file=/etc/opt/omi/conf/omsconfig/last_statusreport program=St
atusReport.sh gpparent=dsc_host ggpparent=python2 gggparent=sh container_id=host image=<NA>)
Feb  7 17:09:13 CKS-Worker falco: 17:09:13.175486641: Warning Shell history had been deleted or renamed (user=root user_loginuid=-1 type=opena
t command=bash fd.name=/root/.bash_history name=/root/.bash_history path=<NA> oldpath=<NA> k8s_test_test_default_c9fc92d9-971d-4113-bf64-be677
3d28003_1 (id=137e2d04e4f5))
Feb  7 17:09:17 CKS-Worker falco: 17:09:17.149266758: Warning 17:09:17.149266758,root,k8s_test_test_default_c9fc92d9-971d-4113-bf64-be6773d280
03_1,137e2d04e4f5

```

## Immutability

Immutable basically is a fancy way of saying unchangeable. With this concept in mind, it is much easier to understand the goal of setting up this attribute in K8s Pods. At the moment, startupProbe would be used to achieve the purpose.

startupProbe is designed for detecting legacy application (old applications) to have sufficient time for starting up before it is being monitored by K8s. This probe is invented since readinessProbe and livenessProbe serve other situations.

```

apiVersion: v1
kind: Pod
metadata:
  creationTimestamp: null
  labels:
    run: immutable
    name: immutable
spec:
  containers:
  - image: httpd
    name: immutable
    resources: {}
  startupProbe:
    exec:
      command:
      - sh
      - -c
      - |
        rm /bin/touch
        rm /bin/bash
    initialDelaySeconds: 5
    periodSeconds: 5
  dnsPolicy: ClusterFirst
  restartPolicy: Always
status: {}

```

[Open in app](#)

```
jonw@CKS-Master:~$ kubectl exec immutable -it -- bash
OCI runtime exec failed: exec failed: container_linux.go:349: starting container process caused "exec: \"bash\": executable file not found in $PATH": unknown
command terminated with exit code 126
```

Another immutable Pod example with read-only root file system. This time, empty directory needs to be created for writing down the essential logs.

```
apiVersion: v1
kind: Pod
metadata:
  creationTimestamp: null
  labels:
    run: immutable2
  name: immutable2
spec:
  containers:
  - image: httpd
    name: immutable2
    resources: {}
    securityContext:
      readOnlyRootFilesystem: true
    volumeMounts:
    - mountPath: /usr/local/apache2/logs
      name: cache-volume
  volumes:
  - name: cache-volume
    emptyDir: {}
  dnsPolicy: ClusterFirst
  restartPolicy: Always
status: {}
```

## kube-apiserver Auditing

Auditing on kube-apiserver is enabled to understand exactly how every request starts and ends in the K8s cluster. A simple kube-apiserver auditing policy template can be searched [here](#).

```
# Log all requests at the Metadata Level.
apiVersion: audit.k8s.io/v1
kind: Policy
```



[Open in app](#)

After creating a directory under `/etc/kubernetes` called “audit”, we put the created `policy.yaml` under the directory `/etc/kubernetes/audit`. Then, we head over to `kube-apiserver` configuration file to ensure all configuration files are being loaded. [Here](#) is the detailed information. However, the following screenshot could be an easier way to enable the same functionality.

**\*\*Be aware that there is NO “readOnly: true” in `k8s-audit` volumeMount.\*\***

```
apiVersion: v1
kind: Pod
metadata:
  annotations:
    kubeadm.kubernetes.io/kube-apiserver.advertise-address.endpoint: 192.168.1.4:6443
  creationTimestamp: null
  labels:
    component: kube-apiserver
    tier: control-plane
  name: kube-apiserver
  namespace: kube-system
spec:
  containers:
  - command:
    - kube-apiserver
    - --audit-log-path=/etc/kubernetes/audit/logs/audit.log
    - --audit-policy-file=/etc/kubernetes/audit/policy.yaml
    - --encryption-provider-config=/etc/kubernetes/etcd/ec.yaml
    - --advertise-address=192.168.1.4
    - --allow-privileged=true
    - --authorization-mode=Node,RBAC
    - --client-ca-file=/etc/kubernetes/pki/ca.crt
    - --enable-admission-plugins=NodeRestriction
    - --enable-bootstrap-token-auth=true
    - --etcd-cafile=/etc/kubernetes/pki/etcd/ca.crt
    - --etcd-certfile=/etc/kubernetes/pki/apiserver-etcd-client.crt
    - --etcd-keyfile=/etc/kubernetes/pki/apiserver-etcd-client.key
    - --etcd-servers=https://127.0.0.1:2379
```

```
name: k8s-etcd
readOnly: true
- mountPath: /etc/kubernetes/audit
  name: k8s-audit
hostNetwork: true
priorityClassName: system-node-critical
volumes:
- hostPath:
    path: /etc/ssl/certs
    type: DirectoryOrCreate
  name: ca-certs
- hostPath:
    path: /etc/ca-certificates
    type: DirectoryOrCreate
```

Open in app



```

    type: DirectoryOrCreate
    name: k8s-certs
  - hostPath:
    path: /usr/local/share/ca-certificates
    type: DirectoryOrCreate
    name: usr-local-share-ca-certificates
  - hostPath:
    path: /usr/share/ca-certificates
    type: DirectoryOrCreate
    name: usr-share-ca-certificates
  - hostPath:
    path: /etc/kubernetes/etcd
    type: DirectoryOrCreate
    name: k8s-etcd
  - hostPath:
    path: /etc/kubernetes/audit
    type: DirectoryOrCreate
    name: k8s-audit
status: {}

```

Checking the logs with

```
sudo tail -f /etc/kubernetes/audit/logs/audit.log
```

```

jonw@CKS-Master:/etc/kubernetes$ tail -f audit/logs/audit.log
{"kind":"Event","apiVersion":"audit.k8s.io/v1","level":"Metadata","auditID":"a91357fc-f306-4d97-ba4c-8c0a62b4740a","stage":"ResponseComplete",
"requestURI":"/readyz","verb":"get","user":{"username":"system:anonymous","groups":["system:unauthenticated"]},"sourceIPs":["192.168.1.4"],"use
rAgent":"kube-probe/1.20","responseStatus":{"metadata":{},"code":200},"requestReceivedTimestamp":"2021-02-06T19:07:40.079594Z","stageTimeStam
p":"2021-02-06T19:07:40.086859Z","annotations":{"authorization.k8s.io/decision":"allow","authorization.k8s.io/reason":"RBAC: allowed by Cluste
rRoleBinding \"/system:public-info-viewer\" of ClusterRole \"/system:public-info-viewer\" to Group \"/system:unauthenticated\""}}
{"kind":"Event","apiVersion":"audit.k8s.io/v1","level":"Metadata","auditID":"a253607d-0277-4af0-baea-a28e00a66a90","stage":"RequestReceived",
"requestURI":"/readyz","verb":"get","user":{"username":"system:anonymous","groups":["system:unauthenticated"]},"sourceIPs":["192.168.1.4"],"use
rAgent":"kube-probe/1.20","requestReceivedTimestamp":"2021-02-06T19:07:41.081387Z","stageTimestamp":"2021-02-06T19:07:41.081387Z"}
{"kind":"Event","apiVersion":"audit.k8s.io/v1","level":"Metadata","auditID":"a253607d-0277-4af0-baea-a28e00a66a90","stage":"ResponseComplete",
"requestURI":"/readyz","verb":"get","user":{"username":"system:anonymous","groups":["system:unauthenticated"]},"sourceIPs":["192.168.1.4"],"use
rAgent":"kube-probe/1.20","responseStatus":{"metadata":{},"code":200},"requestReceivedTimestamp":"2021-02-06T19:07:41.081387Z","stageTimeStam
p":"2021-02-06T19:07:41.089619Z","annotations":{"authorization.k8s.io/decision":"allow","authorization.k8s.io/reason":"RBAC: allowed by Cluste
rRoleBinding \"/system:public-info-viewer\" of ClusterRole \"/system:public-info-viewer\" to Group \"/system:unauthenticated\""}}
{"kind":"Event","apiVersion":"audit.k8s.io/v1","level":"Metadata","auditID":"b07adba8-75a8-41db-bc6e-bac9e62fddc5","stage":"RequestReceived",
"requestURI":"/apis/coordination.k8s.io/v1/namespaces/kube-system/leases/kube-scheduler?timeout=10s","verb":"get","user":{"username":"system:ku
be-scheduler","groups":["system:authenticated"]},"sourceIPs":["192.168.1.4"],"userAgent":"kube-scheduler/v1.20.2 (linux/amd64) kubernetes/faec
b19/leader-election","objectRef":{"resource":"leases","namespace":"kube-system","name":"kube-scheduler","apiGroup":"coordination.k8s.io","apiV
ersion":"v1"},"requestReceivedTimestamp":"2021-02-06T19:07:41.519781Z","stageTimestamp":"2021-02-06T19:07:41.519781Z"}
{"kind":"Event","apiVersion":"audit.k8s.io/v1","level":"Metadata","auditID":"cad042ba-a02e-48f8-86e8-ad4ab375dd0","stage":"RequestReceived",
"requestURI":"/apis/coordination.k8s.io/v1/namespaces/kube-system/leases/kube-controller-manager?timeout=10s","verb":"get","user":{"username":"
system:kube-controller-manager","groups":["system:authenticated"]},"sourceIPs":["192.168.1.4"],"userAgent":"kube-controller-manager/v1.20.2 (l
inux/amd64) kubernetes/faecb19/leader-election","objectRef":{"resource":"leases","namespace":"kube-system","name":"kube-controller-manager","a
piGroup":"coordination.k8s.io","apiVersion":"v1"},"requestReceivedTimestamp":"2021-02-06T19:07:41.519781Z","stageTimestamp":"2021-02-06T19:07:
41.519781Z"}

```

Every information starts with “{” and ends with “}”, in other words, JSON format. We could get a clear visual by putting the content in any online JSON formatter. For example, I would use [this site](https://www.jsonlint.com/) to get formatted JSON data.

```

{"kind":"Event",
  "apiVersion":"audit.k8s.io/v1",
  "level":"Metadata",
  "auditID":"a91357fc-f306-4d97-ba4c-8c0a62b4740a",
  "stage":"ResponseComplete",
  "requestURI":"/readyz",
  "verb":"get",
  "user":{"username":"system:anonymous",
    "groups":["system:unauthenticated"]},
  "sourceIPs":["192.168.1.4"],
  "userAgent":"kube-probe/1.20",
  "responseStatus":{"metadata":{},"code":200},
  "requestReceivedTimestamp":"2021-02-06T19:07:40.079594Z",
  "stageTimestamp":"2021-02-06T19:07:40.086859Z",
  "annotations":{"authorization.k8s.io/decision":"allow",
    "authorization.k8s.io/reason":"RBAC: allowed by ClusterRoleBinding \"/system:public-info-viewer\" of ClusterRole \"/system:public-info-viewer\" to Group \"/system:unauthenticated\""}
}

```





[Open in app](#)

```
# Don't log watch requests by the system:kube-proxy on endpoints or services
- level: None
  users: ["system:kube-proxy"]
  verbs: ["watch"]
  resources:
    - group: "" # core API group
      resources: ["endpoints", "services"]

# Don't Log authenticated requests to certain non-resource URL paths.
- level: None
```

## AppArmor

There is a lot of articles explaining what AppArmor does, but I would put down how I learn its core concepts and how I think it is applied in day-to-day scenarios. Essentially, AppArmor generates profiles and these profiles controls what processes could be executed and what could not. For example, if administrators write a script about “write” and “delete” a file in a certain directory and use AppArmor to generate a profile around it. By the second time the script is executed, the associated AppArmor rule would be initiated and for most cases, the script would be terminated with some failure until the associated profile is altered otherwise. [This](#) is a pretty good article that explains the core concepts of what AppArmor does.

In practice, we could leverage AppArmor profile in K8s Pods, or to be more precise, in containers, to prevent any intentional parties to perform actions other than allowed ones.

We first would need to an AppArmor profile (let's call the customized profile “test”) and load it into the nodes that would be hosting the pods, it should be the worker nodes unless the cluster was configured differently.

Load the new AppArmor profile

```
sudo apparmor_parser /etc/apparmor.d/test
```

[Open in app](#)

```
sudo aa-status
```

```
apparmor module is loaded.
17 profiles are loaded.
17 profiles are in enforce mode.
  /sbin/dhclient
  /usr/bin/lxc-start
  /usr/bin/man
  /usr/lib/NetworkManager/nm-dhcp-client.action
  /usr/lib/NetworkManager/nm-dhcp-helper
  /usr/lib/connman/scripts/dhclient-script
  /usr/lib/snapd/snap-confine
  /usr/lib/snapd/snap-confine//mount-namespace-capture-helper
  /usr/sbin/tcpdump
docker-default
lxc-container-default
lxc-container-default-cgns
lxc-container-default-with-mounting
lxc-container-default-with-nesting
man_filter
man_groff
test
0 profiles are in complain mode.
23 processes have profiles defined.
23 processes are in enforce mode.
docker-default (3896)
```

Test creating a K8s pod with loading this AppArmor profile and see whether the container really is managed by the defined actions. For detailed information, check [here](#).

```
apiVersion: v1
kind: Pod
metadata:
  creationTimestamp: null
  annotations:
    container.apparmor.security.beta.kubernetes.io: localhost/test
  labels:
    run: apparmor
  name: apparmor
spec:
  containers:
  - image: nginx
    name: apparmor
    resources: {}
  dnsPolicy: ClusterFirst
  restartPolicy: Always
```

[Open in app](#)

Test the allowed actions by executing into the terminal of the container.

```
kubectl exec apparmor -it - bash
```

## Seccomp

Seccomp is the abbreviation of secure computing mode. To my understanding, it basically restricts the instance, virtual machine or container, to perform any action other than the preset ones. That includes `exit()`, `sigreturn()`, `read()` and `write()`. This is from [Linux manual page](#).

In practice, if we would like to have K8s Pods (containers) to run under this condition, we would need to make sure this configuration (let's call the configuration "default.json") is loaded somewhere in the core services. In this case, that would be in kubelet. The default path to read seccomp profile for kubelet is `/var/lib/kubelet`.

```
cat /var/lib/kubelet/seccomp/default.json
```

```
root@CKS-Worker:/home/jonw# cat /var/lib/kubelet/seccomp/default.json
{
  "defaultAction": "SCMP_ACT_ERRNO",
  "archMap": [
    {
      "architecture": "SCMP_ARCH_X86_64",
      "subArchitectures": [
        "SCMP_ARCH_X86",
        "SCMP_ARCH_X32"
      ]
    },
    {
      "architecture": "SCMP_ARCH_AARCH64",
      "subArchitectures": [
        "SCMP_ARCH_ARM"
      ]
    }
  ],
}
```



[Open in app](#)

```
    "SCMP_ARCH_MIPS64N32",
  ],
},
{
  "architecture": "SCMP_ARCH_MIPS64N32",
  "subArchitectures": [
    "SCMP_ARCH_MIPS",
```

Here is the way to setup seccomp in a K8s Pod(container).

```
apiVersion: v1
kind: Pod
metadata:
  creationTimestamp: null
  labels:
    run: seccomp
  name: seccomp
spec:
  securityContext:
    seccompProfile:
      type: Localhost
      localhostProfile: default.json
  containers:
  - image: nginx
    name: seccomp
    resources: {}
  dnsPolicy: ClusterFirst
  restartPolicy: Always
status: {}
```

Test the allowed actions by executing into the terminal of the container.

```
kubectl exec seccomp -it - bash
```

## Linux Basic Administration

It feels like it is the best to prepare this section of the CKS exam through some other existing online learning material. For example, Digital Ocean provides a thorough

[Open in app](#)

This last part of the CKS preparation series is extremely broad and it is closely related with many common Linux administration concepts. If you have not much experience around Linux, please make sure to do extra research on the Net and do plenty of practice before going for the real exam as the testing environment would only have more restrictions. This article covers the last piece of what CKS would be testing and the next article would just be giving some exam hacks and advice! Happy Learning!

[Kubernetes Security](#)[Preparation](#)[Certification](#)[Falco](#)[Syscalls](#)[About](#) [Help](#) [Legal](#)

Get the Medium app

