KUBECON · MUMBAI 2026

Root Without Risk

A Decade-Long Quest for True Container Isolation

Kubernetes User Namespaces · KEP-127 → GA in v1.36

hostUsers:

false

Alpha · v1.25

Beta · v1.30

GA · v1.36

Whoami

Sumir Broota

Tech Architect (AWS/GCP) • Kubestronaut • BreachForce & MumbaiFOSS Moderator

Recently started my own tech consultancy so if you have an interesting tech challenge

Let’s Connect:

sumirbroota.com • connect@sumirbroota.com

THE PROBLEM

Root in the container is root on the host

Default Linux containers share the host's user namespace.

INSIDE CONTAINER

uid=0(root) gid=0(root)

CAP_SYS_ADMIN CAP_NET_ADMIN CAP_SYS_PTRACE

```
$ id
uid=0(root) gid=0(root) groups=0(root)
```

shared user namespace

ON THE HOST

uid=0(root) gid=0(root)

Container escape ⇒ full root on host

```
$ ps -p $PID -o uid,cmd
  0 nginx: master process
```

One kernel bug = full host compromise.

CVE-2019-5736 CVE-2024-21626 CVE-2021-25741 Azurescape

EXPLOIT SCENARIO

CAP_SYS_ADMIN: a loaded gun, defused

⚠ hostUsers: true (default pre-1.30)

```yaml
apiVersion: v1
kind: Pod
spec:
  hostUsers: true
  containers:
  - name: app
    securityContext:
      capabilities:
        add: ["CAP_SYS_ADMIN"]
```

```
# mount -o bind /host /mnt
# succeeds!
cat /mnt/etc/shadow
root:$6$abc123...
```

× Host filesystem compromised

hostUsers: false (KEP-127)

```yaml
apiVersion: v1
kind: Pod
spec:
  hostUsers: false
  containers:
  - name: app
    securityContext:
      capabilities:
        add: ["CAP_SYS_ADMIN"]
```

```
# mount -o bind /host /mnt
mount: permission denied
# id
uid=0(root) # only inside ns
```

✓ Capability scoped to pod userns only

THE JOURNEY

9 years from idea to GA

KEP-127 across 11 Kubernetes releases

9
Years from KEP to GA

3
Runtimes modified

7+
CVEs mitigated

WHY

9 years?

Four hard problems, solved one at a time.

01

Kernel support

- idmap mounts need Linux 5.12+ minium
- tmpfs idmap support got added in 6.3+
- Many distros & filesystems took time to implement the same

02

Alpha rewrites

- v1.25 chown approach for vol mount too slow
- v1.27 redesign to idmap mounts
- Stateful pod support added v1.28

03

CRI protocol overhaul

- New UserNamespace proto message
- IDMapping added to Mount
- All runtimes had to follow

04

Storage & UX

- chown duplicated images per pod
- Volume compatibility (NFS, devices)
- PSS profile integration needed

BREAKTHROUGHS

Two changes flipped the curve

v1.27 — idmap mounts

−97%

image storage overhead saved

- Replaces recursive chown of rootfs
- Linux 5.12+ (files) · 6.3+ (overlayfs)
- Single idmap per overlayfs · PR #12092

v1.30 / v1.33 — Beta graduations

hostUsers: false

now a one-line pod field

Version	Date	Status
v1.30	Mar 2024	Beta (off)
v1.33	2025	Beta (on by default)
v1.36	Apr 2026	GA

- PR #123593 · configurable UID/GID ranges
- v1.33 enabled by default feature on kubelet
- v1.34 adds Prometheus metrics

pkg/kubelet/userns/userns_manager.go

kubernetes/kubernetes

*Psuedo code demonstrating happy path

```go
func (m *UsernsManager) GetOrCreateUserNamespaceMappings(
 logger klog.Logger, pod *v1.Pod, runtimeHandler string,
) (*runtimeapi.UserNamespace, error) {
 if pod == nil || pod.Spec.HostUsers == nil {
 return &runtimeapi.UserNamespace{
 Mode: runtimeapi.NamespaceMode_NODE,
 }, nil
 }
 if *pod.Spec.HostUsers {
 return &runtimeapi.UserNamespace{
 Mode: runtimeapi.NamespaceMode_NODE,
 }, nil
 }
 m.lock.Lock(); defer m.lock.Unlock()
 userNs, err := m.createUserNs(logger, pod) // ← allocate range
 if err ≠ nil { return nil, err }
 return &runtimeapi.UserNamespace{
 Mode: runtimeapi.NamespaceMode_POD, // ← per-pod userns
 Uids: toCRI(userNs.UIDMappings),
 Gids: toCRI(userNs.GIDMappings),
 }, nil
}

```

UsernsManager

Per-node allocator (kubelet)

01

Pod opts in via hostUsers: false

Nil or true → returns NODE mode (shared host userns)

02

Manager allocates a 65536-ID range

createUserNs() picks next available bit in bitmap

03

Returns CRI UserNamespace{POD, uids, gids}

Passed to container runtime via CRI call

`AllocationBitmap` `/var/lib/kubelet/pods/$UID/userns`

CRI Protocol

api.proto: New Messages for Namespace-Aware Runtimes

pkg/apis/runtime/v1/api.proto

kubernetes/cri-api

*Psuedo code demonstrating happy path

BEFORE NamespaceOption — pre-KEP-127

```
message NamespaceOption {
   NamespaceMode network = 1;
   NamespaceMode pid = 2;
   NamespaceMode ipc = 3;
}
```

AFTER ...with User Namespace Support

```
message NamespaceOption {
   ...
   UserNamespace userns_options = 4; // ← new
}

message UserNamespace {
   NamespaceMode mode = 1; // NODE | POD
   repeated IDMapping uids = 2;
   repeated IDMapping gids = 3;
}

message IDMapping {
   uint32 host_id = 1;
   uint32 container_id = 2;
   uint32 length = 3;
}
```

Also: Mount(protobuf message) gains uid_mappings / gid_mappings for idmap volumes.

```
repeated IDMapping uid_mappings = 6;
```

CONTAINER RUNTIME

Containerd: from chown to idmap

KEY PULL REQUESTS

PR #7679
Initial userns CRI support
+909 / -20 · Dec 2022

PR #10307
Multi-entry UID/GID mappings
+557 / -72 · Sep 2024

PR #12092
Single idmap per overlayfs
perf optimization · 2024

CONFIGURATION & VERIFICATION

/etc/containerd/config.toml

```
[plugins."io.containerd.snapshotter.v1.overlayfs"]
slow_chown = true # fallback if kernel < 6.3
```

If snapshotter 'overlayfs' doesn't support idmap mounts on this host, configure slow_chown to fall back to recursive chown...

Verify idmap is active:

```
$ mount | grep overlay
overlay on / type overlay (rw,relatime,
lowerdir=/tmp/ovl-idmapped823885363/0,...)
```

MINIMUM REQUIREMENTS

containerd ≥ 2.0 | runc ≥ 1.2 / crun ≥ 1.9 | kernel ≥ 6.3 (overlayfs idmap)

libcontainer/specconv/spec_linux.go — opencontainers/runc

```
complexFlags = map[string]func(*configs.Mount) {
    "idmap": func(m *configs.Mount) {
        m.IDMapping = new(configs.MountIDMapping)
        m.IDMapping.Recursive = false
    },
    "ridmap": func(m *configs.Mount) {
        m.IDMapping = new(configs.MountIDMapping)
        m.IDMapping.Recursive = true // recursive for rbind
    },
}
```

tests/cmd/fs-idmap/fs-idmap.go

```
// Probes whether the filesystem supports MOUNT_ATTR_IDMAP.
// Used by runc to decide between idmap and slow_chown fallback.
```

The syscall behind it

`mount_setattr(MOUNT_ATTR_IDMAP)`

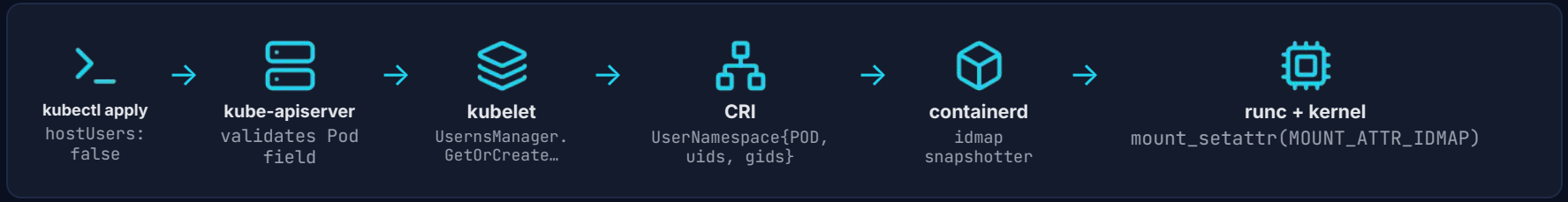
1. Open a userns fd with desired UID/GID map
2. Call mount_setattr with MOUNT_ATTR_IDMAP referencing that fd
3. Kernel rewrites ownership on-the-fly during VFS lookups

> “No data is rewritten on disk — the mapping is virtual.”
>
> Linux 5.12 mount API

OCI Spec v1.2.0 | options: ["idmap"] | options: ["ridmap"]

END-TO-END FLOW

From PodSpec to UID 65536: The Full Path



UID Mapping — How container root becomes unprivileged on the host

Inside the kernel

User namespaces, uid_map, and how syscalls see UIDs

```
# inside the pod
$ cat /proc/self/uid_map
0 65536 65536
# columns: container_id host_id length
```

1. setuid/getuid/stat translate through this table
2. Capabilities are scoped to the namespace owner
3. Discrete Action Control (DAC) checks happen against MAPPED UIDs on host
4. Unmapped UIDs appear as `nobody (65534)` inside

Container		Host
0 (root)	→	65536
1000 (alice)	→	66536
65535 (max)	→	131071

Kernel does the translation in `from_kuid_munged()`— userspace never sees host UIDs.

`kernel/user_namespace.c`

`CAP_SETUID owned by parent ns`

CVE-2024-21626

Leaky Vessels

runc leaked an open fd into the container's working directory.

Patched runc 1.1.12 | CVSS 8.6 | Disclosed Jan 31, 2024

The exploit, in 4 steps

```
{
"process": {
"cwd": "/proc/self/fd/7", // ← leaked fd
"args": ["/bin/sh"]
}
}
```

01 runc opens internal fd to /sys/fs/cgroup

02 fd is inherited by container process

03 Attacker sets cwd to `/proc/self/fd/N`

04 Container process now runs *outside* the chroot

Without userns

Escaped process = **uid 0** on host → overwrite /usr/bin/runc, read /etc/shadow.

With userns

Escaped process = **uid 65536** → DAC denies everything sensitive.

Also mitigated by user namespaces:

CVE-2025-31133, CVE-2025-52565, CVE-2025-52881

DEMO 1 · VULNERABLE

Escape without user namespaces

Terminal 1 — apply the pod

```yaml
apiVersion: v1
kind: Pod
metadata:
name: vulnerable-pod
spec:
hostUsers: true # default — vulnerable
containers:
- name: app
image: ubuntu:22.04
command: ["sleep", "3600"]
```

```
$ kubectl apply -f vulnerable-pod.yaml
pod/vulnerable-pod created
```

Terminal 2 — observe on host

```
$ PID=$(crictl inspect $(crictl ps -q --name app) \
  | jq .info.pid)
$ ps -p $PID -o uid,gid,cmd
UID GID CMD
0 0 sleep 3600 ← root on the host!
```

```
$ nsenter -t $PID -m -p touch /host-was-here
$ ls /host-was-here # exists on the host!
/host-was-here
```

VERDICT: container UID 0 = host UID 0. One escape and it's over.

DEMO 2 · DEFENDED

Same attack, with hostUsers: false

Terminal 1 — apply the pod

```
apiVersion: v1
kind: Pod
metadata:
name: demo-defended
spec:
hostUsers: false # ← the fix
containers:
- name: app
image: ubuntu:22.04
command: ["sleep", "3600"]
```

```
$ kubectl apply -f defended-pod.yaml
pod/demo-defended created
```

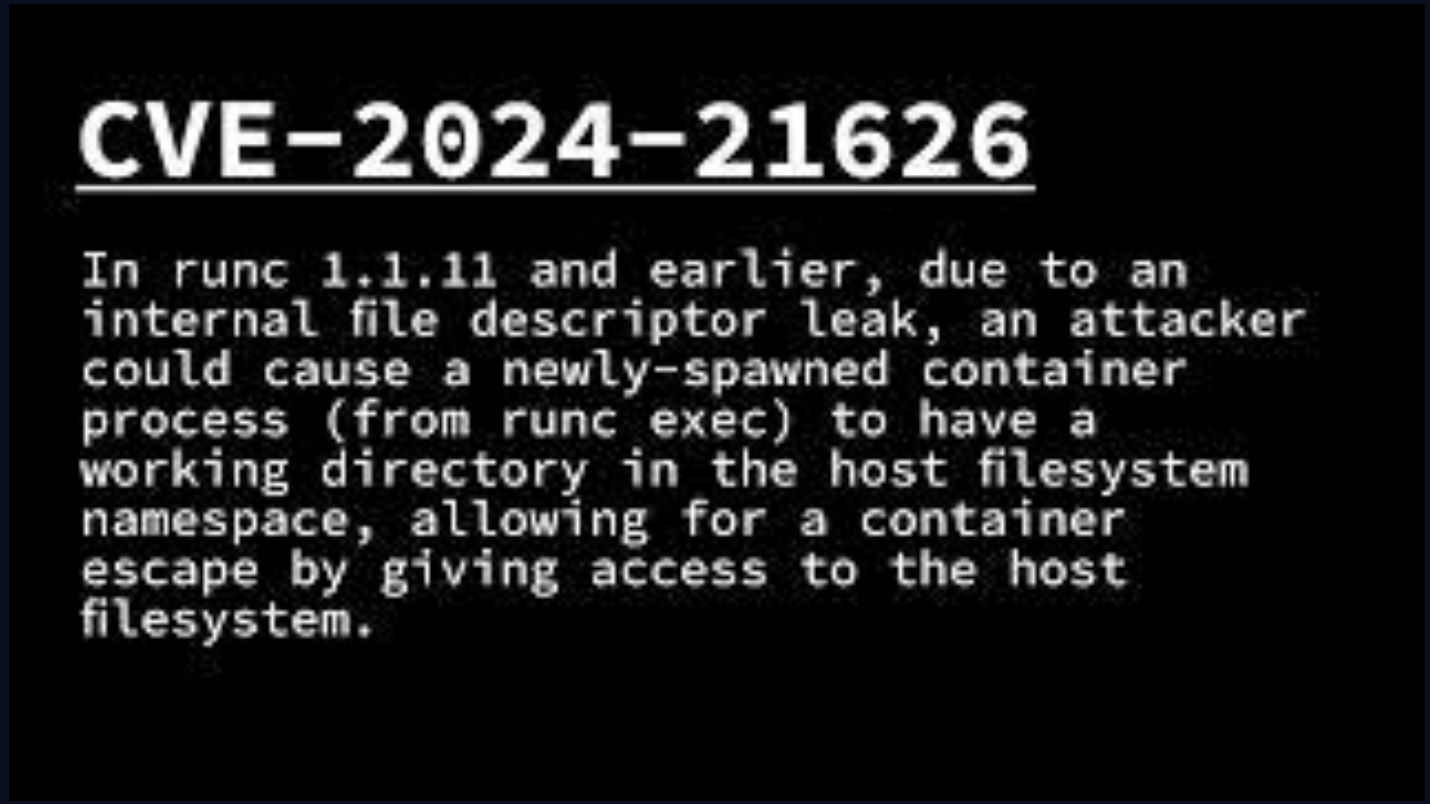
Terminal 2 — observe on host

```
$ ps -p $PID -o uid,gid,cmd
UID GID CMD
65536 65536 sleep 3600 ← unprivileged on host
```

```
$ kubectl exec demo -- cat /proc/self/uid_map
0 65536 65536
$ kubectl exec demo -- id
uid=0(root) gid=0(root) ← root *inside* only
```

VERDICT: container root maps to host UID 65536 — escape gains nothing useful.

Mitigation of CVE-2024-21626 on Kubernetes by enabling User Namespace support



https://www.youtube.com/watch?v=07y5bl5UDdA

CONFIGURATION

Tuning User Namespace

Custom UID/GID Ranges Setup

Create `kubelet` user (fixed name)

Configure `/etc/subuid` and `subgid`

```
# user:start_uid:count
Kubelet:100000:65536
# ideally set count to 65536 * max pods per node
```

Kubelet Configuration

Set max IDs via `userNamespacesPerPodLength`

MUST be a multiple of `65536`

Default: `65536` IDs per pod

Kubelet Configuration Example

```
apiVersion: kubelet.config.k8s.io/v1beta1
kind: KubeletConfiguration
userNamespacesPerPodLength: 65536

# Valid: 65536, 131072, 196608...
# Restart kubelet after changing
```

- Unique non-overlapping ranges per pod from nodes subuid pool
- Max pods = subuid count ÷ length
- Stable mapping across pod restarts

Requires Kubelet Restart
Existing pods retain mappings until restart. Plan node drains accordingly.

kubernetes.io/docs/concepts/workloads/pods/user-namespaces/#id-count-for-each-of-pods

Enabling User Namespaces

userns-pod.yaml

```yaml
apiVersion: v1
kind: Pod
metadata:
  name: userns-pod
spec:
  hostUsers: false ← the one line that matters
  containers:
  - name: app
    image: nginx:latest
    securityContext:
      runAsUser: 0 # root *inside* the pod
      runAsGroup: 0
      capabilities:
        add: ["CAP_SYS_ADMIN"]
    volumeMounts:
    - name: data
      mountPath: /data
  volumes:
  - name: data
    emptyDir: {}
```

Verify it worked

```
$ kubectl exec userns-pod -- \
cat /proc/self/uid_map 0 65536 65536
```

What changes

- runAsUser 0 → host uid **65536**
- CAP_SYS_ADMIN scope → pod userns only
- Volumes → idmap-mounted automatically

Caveats

- NFS volumes not supported
- volumeDevices incompatible
- Some PSS profiles may need updates

DEPLOYMENT

Shipping it to production

What you need on every node — and how to roll it out.

Requirements matrix

Component	Minimum	Recommended	Notes
Kubernetes	v1.30	v1.36 (GA)	hostUsers field stable
containerd	1.7	2.0+	idmap mounts in 2.0
OCI runtime	runc 1.1	runc 1.2 / crun 1.9	mount_setattr support
Linux kernel	5.12	6.3+	overlayfs idmap
Filesystem	ext4/xfs	overlayfs & tmpfs idmap supported	NFS unsupported

Rollout playbook

01 Audit pods for hostNetwork / hostPID / hostIPC, NFS & volumeDevices usage - exclude those (not supported)

02 Configure `/etc/subuid` · `/etc/subgid` on nodes (or use defaults)

03 Roll out to a dev nodepool with `hostUsers: false`

04 Watch metrics:
`started_user_namespaced_pods_total`,
`started_user_namespaced_pods_errors_total`

05 Update PSS Restricted profile cluster-wide

REALITY CHECK

Defense-in-Depth, Not a Silver Bullet

✓ DOES

What UserNS Actually Does

- Maps container root to unprivileged host UID — no real root on the host
- Mitigates container-to-host escape vulnerabilities (e.g., CVE-2024-21626)
- Multi-tenant UID isolation — containers can't interfere with each other via shared file ownership
- Easy Migration & added security especially for build pipelines

✗ DOES NOT

Limitations

- Does **NOT** prevent in-container lateral movement
- Does **NOT** block ServiceAccount token theft
- Does **NOT** prevent API server reconnaissance
- Does **NOT** work with `hostNetwork: true`
- Breaks NFS mounts (no idmap support)

Defence in Depth Solutions

0. Easy to implement User Namespaces

1. Non-root images (UID 65534), distroless, drop capabilities
2. `seccomp`, readOnlyRootFilesystem
3. Pod Security Standards (Baseline / Restricted)
4. MicroVMs (Kata Containers) for truly untrusted workloads

Real security is built along side the **Dockerfile**, not just the **PodSpec**

blog.zwindler.fr
reddit.com/r/kubernetes

Q & A

Ask me anything about user namespaces, KEP-127, or container security.

SB

Sumir Broota
Tech Architect

connect@sumirbroota.com
sumirbroota.com

List of Resources

Thank you

SB

Sumir Broota
Tech Architect

connect@sumirbroota.com
sumirbroota.com

Thanks especially to those contributors who helped make this feature a reality

@rata
@giuseppe
@saschagrunert… and many more

Thanks!
KubeCon Mumbai 2026