Why Unprivileged Containers Are Less Secure

Cong Wang

Founder and CEO, Multikernel Technologies, Inc.

The Most Misleading Names in Linux Security

Multikernel TECHNOLOGIES

"Rootless" and "unprivileged" containers tell you two true things:

- Rootless: the container runtime does not need root
- Unprivileged: root inside the container is not root on the host

What most people hear is a third thing:

- **A container with fewer privileges, therefore a safer one**

> *That third reading is backwards. A user namespace grants privileges.*

Two Questions About a Sandbox

Multikernel TECHNOLOGIES

1. Identity (how bad a non-kernel breakout is): who is the process on the host?

- Root container: UID 0 on the host
- User namespace: an unprivileged UID, often your own
- Only matters for user-space escapes: a kernel exploit runs as the kernel

2. Reach (how likely a kernel breakout is): how much kernel code can it execute with arguments it controls?

- Most real escapes are kernel bugs, and a kernel bug only triggers from code you can reach

> *Ranked by reach: Sandlock < unprivileged user < user namespace < root*

PART 1

A User Namespace Is a Bag of Capabilities

What `bwrap --unshare-user` actually hands you

What the Bag Holds

Multikernel TECHNOLOGIES

Capability	What it unlocks inside a user namespace
`CAP_SYS_ADMIN`	Mount tmpfs, proc, sysfs, overlayfs, FUSE; create mount, network, PID, IPC, UTS, cgroup, time namespaces
`CAP_NET_ADMIN`	Interfaces, routes, netfilter and nf_tables, tc qdiscs and filters, virtual devices
`CAP_NET_RAW`	Raw sockets, `AF_PACKET`
`CAP_SETUID`, `CAP_SETGID`	Write ID maps for nested namespaces, switch among mapped IDs
`CAP_SYS_CHROOT`	`chroot`
`CAP_CHOWN`, `CAP_DAC_OVERRIDE`, `CAP_FOWNER`	Bypass file permissions, on files owned by mapped IDs
`CAP_SYS_PTRACE`, `CAP_KILL`	Trace and signal processes in the namespace
`CAP_SYS_MODULE`, `CAP_SYS_RAWIO`, `CAP_BPF`, …	Nothing: still checked against the host

Every Capability, One Syscall

Multikernel TECHNOLOGIES

```
$ B="bwrap --ro-bind / / --dev /dev --proc /proc --unshare-user --unshare-net"
$ $B grep CapEff /proc/self/status
CapEff: 0000000000000000      # bwrap drops them by default
$ $B --uid 0 --cap-add ALL grep CapEff /proc/self/status
CapEff: 000001ffffffffff      # all 41 capabilities
$ $B python3 -c 'import os; os.unshare(os.CLONE_NEWUSER | os.CLONE_NEWNET); \
    print(open("/proc/self/status").read())' | grep CapEff
CapEff: 000001ffffffffff      # dropped, then taken back with one syscall
```

- Inside the sandbox, `unshare(CLONE_NEWUSER|CLONE_NEWNET|CLONE_NEWNS)` still works: the user namespace comes first, its fresh capabilities create the rest
- Any user can configure interfaces, routes, netfilter, tc, and mount tmpfs, proc, sysfs, overlayfs, FUSE
- Scoped to that namespace, but **the kernel decides what that covers**
- `--disable-userns` closes the nested path, but it is opt-in

`capable()` vs `ns_capable()`

Multikernel TECHNOLOGIES

```
/* fs/super.c */
bool mount_capable(struct fs_context *fc)
{
        if (!(fc->fs_type->fs_flags & FS_USERNS_MOUNT))
                return capable(CAP_SYS_ADMIN);
        else
                return ns_capable(fc->user_ns, CAP_SYS_ADMIN);
}
```

- `capable()` asks about the init user namespace: a normal user fails
- `ns_capable()` asks about **your** namespace: you own it, you pass
- Linux 7.0: **269** `ns_capable()`-family call sites, **129** in `net/`

PART 2

The CVE Record

Every bug behind a check a normal user fails, and a user namespace passes

Behind a Capability Check

Multikernel TECHNOLOGIES

CVE	Where	Capability	Outcome
CVE-2017-7308	AF_PACKET ring setup	`CAP_NET_RAW`	Heap overflow, local root
CVE-2018-18955	User namespace ID mapping	`CAP_SYS_ADMIN` + `CAP_SETUID`	Read host `/etc/shadow`
CVE-2020-14386	AF_PACKET receive path	`CAP_NET_RAW`	Heap overflow, local root
CVE-2021-22555	netfilter x_tables	`CAP_NET_ADMIN`	Local root, exploited in the wild
CVE-2022-0185	Filesystem context parsing	`CAP_SYS_ADMIN`	Kubernetes container escape
CVE-2022-0492	cgroup v1 `release_agent`	`CAP_SYS_ADMIN`	Code runs as host root
CVE-2023-0386	overlayfs copy-up	`CAP_SYS_ADMIN`	Host setuid binary, exploited in the wild
CVE-2023-32233	nf_tables	`CAP_NET_ADMIN`	Use-after-free, local root
CVE-2024-1086	nf_tables	`CAP_NET_ADMIN`	Local root, used by ransomware
CVE-2026-53075	PPP	`CAP_NET_ADMIN`	Administer a netns inherited from the host

Netfilter

Multikernel TECHNOLOGIES

CVE-2021-22555: heap OOB write in iptables compat code, there since 2.6.19

- Broke out of a pod in Google's kCTF Kubernetes cluster; CISA: exploited in the wild

CVE-2024-1086: nf_tables use-after-free. The public exploit's README requires:

- `CONFIG_USER_NS=y`, unprivileged user namespaces enabled, nf_tables enabled
- All three are **the default on Debian and Ubuntu**
- October 2025: CISA confirmed ransomware groups were using it

Filesystems and User Namespaces Themselves

Multikernel TECHNOLOGIES

CVE-2022-0185: integer underflow in `legacy_parse_param`, reached through `fsconfig`

- Needs `CAP_SYS_ADMIN`; the exploit got it with `unshare(CLONE_NEWNS|CLONE_NEWUSER)`

CVE-2023-0386: mount overlayfs in a user namespace, copy up a setuid file where the host honors it

- Added to CISA's exploited list in June 2025

CVE-2018-18955 (Jann Horn): nested ID maps with more than five ranges

- Create outer namespace, write a crafted map for the inner one, read `/etc/shadow`

Distros Already Know

Multikernel TECHNOLOGIES

- **Debian:** carried `kernel.unprivileged_userns_clone` for years to turn it off
- **Hardened deployments:** `user.max_user_namespaces = 0`
- **Ubuntu 24.04:** apps may create a user namespace, but get no capabilities in it unless an AppArmor profile allows it
- **Qualys, March 2025:** three bypasses of Ubuntu's restriction, yielding a user namespace with full capabilities, described as a way to reach kernel bugs the restriction was meant to hide

> *Even the mitigation is attack surface.*

PART 3

Going Below Unprivileged

Do not change who the process is, change what it can ask the kernel

Sandlock: Close the Way In

Multikernel TECHNOLOGIES

Sandlock: open source process sandbox for Linux 6.12+, built on Landlock, seccomp-bpf and seccomp user notification. No root, no namespaces, no cgroups.

Cannot create a user namespace

- seccomp denies `unshare`, `setns`, namespace flags on `clone`; no `clone3`
- Every bug in the table stays closed, even where user namespaces are on

Removes syscalls a normal user is allowed to make

- `bpf`, `perf_event_open`, `userfaultfd`, `keyctl`, `ptrace`, `io_uring_*`, raw sockets, …
- Google, 2023: **60%** of kCTF exploit submissions targeted `io_uring`

Answer Syscalls in User Space

Multikernel TECHNOLOGIES

seccomp user notification routes selected syscalls to a supervisor process:

```
sandboxed: socket(AF_NETLINK, ..., NETLINK_ROUTE)
   |  seccomp RET_USER_NOTIF
   v
supervisor: socketpair(AF_UNIX) -> inject one end (NOTIF_ADDFD)
            synthesize RTM_NEWLINK / RTM_NEWADDR for loopback only

kernel netlink / rtnetlink parser: never sees the request
```

- Other netlink protocols: rejected
- `/proc/cpuinfo`, `meminfo`, `mounts`, `/proc/net/*`: sealed memfd with synthesized content
- Sensitive `/proc` and `/sys` paths: denied

A Supervisor Bug Is Not a Kernel Bug

Multikernel TECHNOLOGIES

The supervisor parses untrusted input too. This **moves** attack surface.

Bug in	Attacker gets
Kernel netlink parser	the kernel
Sandlock supervisor	the user who started it: a normal user

Landlock confines what is left

- Paths: read, write, execute per directory tree
- Network: TCP `bind` and `connect` by port
- Scoping: abstract Unix sockets and signals stay inside the sandbox

Nested Sandlock

```
$ S=$(command -v sandlock)
$ sandlock run --no-supervisor -r /proc -r /usr -r /lib -r /lib64 -r /bin -r /etc -r $S -w /tmp -- \
    $S run -r /usr -w /tmp -- echo hello
hello
```

```
outer sandlock          Landlock + deny-only seccomp (incl. unshare, setns)
  inner supervisor      parses untrusted input: netlink, /proc, network policy
    AI agent            inner Landlock + seccomp, stacked on the outer
```

- **Outer, `--no-supervisor`:** kernel-only enforcement, no parser to exploit
- **Inner, supervised:** the full policy, virtualized netlink and `/proc`
- **Escape the inner, land in the outer:** still no user namespace, only the outer's paths
- Supervisor-free outer is required anyway: one seccomp listener per task

What User Namespaces Get Right

Multikernel TECHNOLOGIES

A tradeoff, not a verdict. User namespaces win on identity.

- **A different view:** private PID tree, mount table, network stack. Sandlock enforces policy on the existing view and virtualizes the leaky parts
- **Software that expects root:** a package manager in an image

Sandlock's UID/GID mapping is **opt-in**, and then it does use a user namespace:

- seccomp still applies inside: no mount, no other namespaces, no `CAP_NET_ADMIN`
- Documented as the exception it is

The Ranking

	Who you are after a breakout	**Kernel reach**
Root	root	everything
User namespace	an unprivileged UID	a normal user, **plus every** `ns_capable()` **path**
Unprivileged user	an unprivileged UID	the ordinary syscall interface
Sandlock	an unprivileged UID	**a subset** of the ordinary syscall interface

- A user namespace improves column 1 over root, and worsens column 2 over a normal user
- Sandlock keeps column 1 of a normal user and shrinks column 2 below it

> ***Sandlock < unprivileged user < user namespace < root***

Key Takeaways

- "Unprivileged" is about identity, not kernel reach
- User namespaces widen the way out, for **every** process on the host
- Measure a sandbox by how much kernel it can reach
- Nest Sandlock: an escape lands in another sandbox
- Turn unprivileged user namespaces off. Sandlock does not need them

Questions?

Multikernel TECHNOLOGIES

Contact: cwang@multikernel.io

Website: sandlock.io

Open Source: github.com/multikernel/sandlock