Sandlock: The Lightest AI Sandbox

Cong Wang

Founder and CEO, Multikernel Technologies, Inc.

The Industry Got the Threat Model Wrong

Multikernel TECHNOLOGIES

- Containers and microVMs were built for **adversaries** trying to escape
- AI agents are **not adversaries** , they follow your prompts
- The real threats: **prompt injection** and **agent mistakes**
- An agent runs `rm -rf /` not because it is malicious, but because it misinterpreted the prompt

> *"You do not need a hypervisor to stop `rm -rf /`. You need a filesystem allowlist."*

Isolation Is Not Security

Multikernel TECHNOLOGIES

- A microVM agent can still read your **SSH keys** -- because you mounted them
- A container agent can reach **169.254.169.254** and steal IAM credentials
- All tools share the **same permissions** inside one big sandbox

> *"Isolation answers 'can it escape?' Security answers 'what can it access inside?'"*

- The defense is **policy, not isolation**: deny by default, allowlist by path and host
- Per-tool-call sandboxing, not one box for everything

Rootless Sandboxing

Multikernel TECHNOLOGIES

Inside the sandbox: agents read files, write files, call APIs, none require root

Outside the sandbox: privileged infrastructure expands the blast radius

- Docker daemon runs as root; Firecracker needs `/dev/kvm`
- Privileged components become the target when isolation fails

The right approach:

- Unprivileged kernel interfaces: Landlock, seccomp-bpf, seccomp user notif
- No daemon, no root, no cgroups

What is Sandlock?

Multikernel TECHNOLOGIES

Lightweight process sandbox for Linux -- policy, not isolation

- **5ms startup**: fork() + apply policy + exec()
- **97% native performance**: near-zero steady-state overhead
- **No root, no cgroup, no VMs**: runs as a normal unprivileged user
- **Resource controls**: CPU & memory limits, process caps, disk quota
- **Per-tool-call sandboxing**: each tool gets its own policy
- **Built for AI**: HTTP ACL, COW filesystem, deterministic execution

Architecture

- **sandlock-core**: Rust library with async supervisor (tokio)
- **sandlock-cli**: Full-featured command-line interface

- **sandlock-ffi**: C ABI for Python and other language bindings

Three Layers of Enforcement

Multikernel TECHNOLOGIES

```
Layer 1: Landlock (kernel)
Filesystem read/write/exec per path
TCP port bind/connect restrictions
IPC scoping and signal isolation

Layer 2: seccomp-bpf (kernel)
Pre-compiled BPF bytecode (~1KB)
Blocklist (default) or allowlist mode
Near-zero overhead in kernel space

Layer 3: seccomp user notification (async)
Intercepts syscalls without killing proc
Resource limits, network ACL, COW, /proc
Tokio-based event loop in parent process
```

How It Works: Confinement Flow

After `fork()`, before `exec()`:

1. `PR_SET_NO_NEW_PRIVS` -- prevent privilege escalation
2. **Landlock** rules applied (filesystem + network + IPC)
3. **seccomp-bpf** filter installed
4. Notification FD sent to parent → parent starts **async supervisor**
5. Optional: vDSO patched for time/random virtualization

Total setup: ~5ms: no namespace, no cgroup, no mount

The Async Supervisor

Multikernel TECHNOLOGIES

The parent runs a **tokio event loop** intercepting syscalls:

Category	Syscalls	Action
Process	fork, clone, execve	Enforce process limit, policy callback
Memory	mmap, brk, mremap	Track anonymous memory usage
Network	connect, bind, sendto	IP allowlist, on-behalf execution
Filesystem	openat, unlinkat, mkdirat	COW interception, /proc virtualization
Determinism	getrandom, clock_nanosleep	Seeded PRNG, frozen time

Network Isolation

Multikernel TECHNOLOGIES

Domain-level allowlist:

- Resolve domains at sandbox creation → build IP allowlist
- `connect()` intercepted, checked, executed on behalf of child

HTTP-level ACL (method + host + path):

- Built-in MITM proxy for HTTPS inspection
- Fine-grained rules: `POST api.openai.com/v1/chat/completions`

Port virtualization:

- Multiple sandboxes can `bind()` the same port (e.g., 6379)
- Supervisor remaps to ephemeral ports transparently

Copy-on-Write Filesystem

Multikernel TECHNOLOGIES

Seccomp-based COW (no root, no mount namespace):

- Intercepts `openat`, `unlinkat`, `mkdirat`, `renameat2`
- Writes go to an **upper** directory; reads resolve upper-then-lower
- On exit: **commit** changes to workdir or **discard** (dry-run)

```
# Dry-run: preview what a build would change
sandlock run --dry-run --workdir . -w . \
  -r /usr -r /lib -r /bin -r /etc -- make build
```

/proc Virtualization

Multikernel TECHNOLOGIES

The sandbox sees an **isolated view** of the system:

- `/proc/cpuinfo` -- shows only allocated CPUs
- `/proc/meminfo` -- reports sandbox memory limit as total
- `/proc/loadavg` -- computed from sandbox's own processes
- `/proc/uptime` -- relative to sandbox start time
- `/proc/[pid]/*` -- only sandbox's own PIDs visible

> *"The confined process sees a consistent, isolated view -- as if it has the machine to itself."*

Deterministic Execution

Multikernel TECHNOLOGIES

Frozen time:

- vDSO patched: `clock_gettime` returns fixed timestamp
- Timer syscalls adjusted by supervisor

Seeded randomness:

- `getrandom()` replaced with **ChaCha8 PRNG**
- `/dev/urandom` replaced with seeded memfd

```
sandlock run --time-start "2000-01-01T00:00:00Z" \
             --random-seed 42 -- ./build.sh
```

Same seed = same output, every time

COW `fork()`

Multikernel TECHNOLOGIES

- Initialize expensive state once (e.g., load ML model)
- `fork()` clones the process with shared COW memory
- Each child processes a shard independently
- **~530µs per fork, ~1,900 forks/sec**
- No full process restart, no VM overhead
- Ideal for low-latency parallel workloads

Pipelines for XOA Mode

Multikernel TECHNOLOGIES

Compose sandboxed stages with tight policies

```
[Planner: no data] --pipe--> [Executor: data, no network]
```

- Each stage has its own sandbox policy
- Data flows through kernel pipes only -- prevents exfiltration
- Planner can be networked while executor remains isolated
- Enables XOA mode: separate reasoning, action, and observation stages

Dynamic Policy Callbacks

```
def on_event(event, ctx):
    if event.syscall == "execve" \
       and event.argv_contains("curl"):
        return True  # deny

    if event.category == "file" \
       and "/secrets" in (event.path or ""):
        return errno.EACCES

    return 0  # allow

result = Sandbox(policy, policy_fn=on_event).run(
    ["python3", "untrusted.py"]
)
```

- Per-syscall decisions with rich metadata
- Switch between setup and restricted phases at runtime

Per-Tool-Call Sandboxing

Multikernel TECHNOLOGIES

The problem: one sandbox for all tools = union of all permissions

Sandlock's approach: each tool invocation gets its own sandbox

```
Agent invokes "web_fetch" → fork + Sandbox(net only, no fs write)
Agent invokes "file_write" → fork + Sandbox(fs write, no net)
Agent invokes "run_tests" → fork + Sandbox(read-only src, no net)
```

- Fork + Landlock confinement in **< 1ms** per tool call
- A compromise of one tool cannot access another tool's permissions
- Least privilege at the **right granularity**: per-tool, not per-agent

Performance: Startup

Metric	Bare Metal	Sandlock	Docker
`/bin/echo` startup	2 ms	**7 ms**	307 ms
Overhead	--	**+5 ms**	+305 ms

- **44x faster** than Docker
- No image pull, no layer unpack, no namespace creation
- First run as fast as subsequent runs

Performance: Redis Benchmark

Multikernel TECHNOLOGIES

Metric	Bare Metal	Sandlock	Docker
Redis SET	82K rps	**80K rps**	52K rps
Redis GET	79K rps	**77K rps**	53K rps
p99 latency	0.5 ms	**0.6 ms**	1.5 ms

- **97% of native throughput**
- **2.5x lower tail latency** than Docker
- seccomp filters run in kernel -- near-zero per-syscall cost

CLI Usage

```
# Basic sandbox
sandlock run -r /usr -r /lib -w /tmp -- ls /tmp

# Resource limits (memory + processes) + timeout
sandlock run -m 512M -P 20 -t 30 -- ./compute.sh

# Network: allow only OpenAI API
sandlock run --net-allow-host api.openai.com \
  -r /usr -r /lib -r /etc -- python3 agent.py

# HTTP ACL: method + host + path
sandlock run \
  --http-allow "POST api.openai.com/v1/chat/completions" \
  --http-deny "* */admin/*" \
  -r /usr -r /lib -r /etc -- python3 agent.py
```

Python SDK

```python
from sandlock import Sandbox, Policy

policy = Policy(
    fs_readable=["/usr", "/lib", "/etc"],
    fs_writable=["/tmp"],
    max_memory="256M",
    max_processes=10,
    http_allow=["POST api.openai.com/v1/chat/completions"],
)
result = Sandbox(policy).run(["python3", "agent.py"], timeout=30)
```

```python
# Two-stage pipeline: planner (no data) -> executor (no network)
result = (
    Sandbox(planner_policy).cmd(["python3", "plan.py"])
    | Sandbox(executor_policy).cmd(["python3", "execute.py"])
).run()
```

Requirements & Availability

Multikernel TECHNOLOGIES

Kernel: Linux 6.12+ for full features (Landlock ABI v6)

Privileges: No root, no cgroups, no systemd, no kernel modules

Interfaces:

- **CLI**: `sandlock run ...`
- **Python**: `pip install sandlock`
- **Rust**: `sandlock-core` crate
- **C FFI**: `libsandlock_ffi.so`

Open source: github.com/multikernel/sandlock (Apache 2.0)

Key Takeaways

- The threat model is **policy, not isolation**
- Per-tool-call sandboxing: least privilege at the **right granularity**
- **44x faster startup** than Docker, **97% native throughput**
- **No root, no container, no VMs**
- HTTP ACL, COW filesystem, deterministic execution

> *"The lightest sandbox that doesn't compromise on security."*

Questions and Feedback

Multikernel TECHNOLOGIES

Contact: cwang@multikernel.io

Open Source: github.com/multikernel/sandlock