
OS isolation: the enforcement ladder

Key takeaways

1. Isolation is a ladder: \$PATH and conda advise; namespaces and cgroups enforce.
2. The security property is the delegation list, not the word 'container'.
3. The supervisor owns the process tree, the cleanup, and the receipt.

Common misunderstandings of OS isolation

Which claim would you defend right now? Choose before the evidence.

“Running code inside Docker makes it safe by default.”

“A timeout is enough to contain resource exhaustion.”

“Killing the top-level process cleans up every descendant it started.”

Keep your choice. Look for the mechanism or counterexample that would make it fail.

Most submissions are merely buggy; design for the hostile one

Grading runs a stranger's program with your privileges

```
$ cd /Users/zappa/hw1-grading && ls
student.py  test_hw1.py  # their code, your tests
$ python -m pytest          # imports student.py – their code runs as zappa
```

Danger	Example	At stake
Theft	student.py reads ~/.ssh and posts it to a webhook	confidentiality
Tampering	student.py rewrites test_hw1.py so every test passes	integrity
Exhaustion	a fork loop in student.py starves every other grading job	availability
Escape	student.py starts a daemon that outlives the grading run	containment

- The same threat model covers your agent running any unfamiliar repository's build or tests.

Keep untrusted code inside a resource boundary

A container is a bundle of mechanisms, not a magic security property

Docker, host handed through

```
docker run --rm \
  -v /:/host \
  -v /var/run/docker.sock:/var/run/docker.sock \
  agent-build
```

Docker, explicit capabilities

```
docker run --rm --network none \
  --read-only --cap-drop ALL \
  --memory 1g --pids-limit 128 \
  -v ./submission:/work:ro \
  --tmpfs /scratch:rw,size=2g \
  agent-build
```

- Same tool, same kernel mechanisms.
- Mounts and network decide confidentiality and integrity; the limits decide availability.
- The delegation list is the security property, not the word ‘container.’

Grant capabilities explicitly; deny by default

Each granted capability becomes one launch flag

```
adversary = "buggy or hostile repository code"
allowed = {
  "read":    ["/submission"],          # -v ./submission:/work:ro
  "write":   ["fresh 2 GiB scratch"],  # --tmpfs /scratch:rw,size=2g
  "connect": [],                      # --network none
  "consume": {"memory": "1 GiB",       # --memory 1g
              "pids": 128,             # --pids-limit 128
              "wall_clock": "10 min"}, # supervisor deadline
}
```

- ‘Should the grader reach the network?’ is reviewable; a pile of flags is not.
- Read, write, and connect buy confidentiality and integrity; consume buys availability.
- Deny is the default: anything not granted does not cross the boundary.

Instapoll

Instapoll · Review · Oct 20 A

An agent needs to run an unfamiliar repository's test suite. Which resource would you be least willing to expose to that process?

- A. One read-only copy of the repository under test
- B. A new empty temporary directory for build output
- C. A loopback-only network interface with no listeners
- D. Your normal home directory containing credentials and SSH keys

Isolation is a ladder, not a switch

Each rung virtualizes a bigger interface and is harder to cross



\$PATH

One name lookup, redirected by a shell variable.

Bypassed by any absolute path.



conda

An install tree, redirected by environment variables — no root required.

Leaks by bug, by design, or by malice.



container

The kernel's name tables — PIDs, mounts, network, users — via namespaces; CPU and memory metered by cgroups.

Still shares the host kernel.



VM

The hardware itself: VT-x enters and exits the guest, EPT remaps its memory, VT-d contains its devices.

Strongest rung; boots a kernel to get there.

Isolation bounds the job in space; the supervisor bounds it in time

One owner for the job's lifetime and its evidence

```
job = sandbox.start(policy)          # every mechanism so far
try:
    result = job.wait(deadline=600)  # time is a resource too
finally:
    job.kill_cgroup()               # tree-wide: children cannot reparent away
    receipt = job.receipt()

receipt == {"exit": None, "signal": "SIGKILL",
            "oom_kills": 1, "cpu_s": 599, "peak_mem": "1.0 GiB"}
# not a failing test suite — the job hit its memory ceiling
```

- Kill the cgroup, not the shell PID: descendants reparent and outlive their parent.
- The receipt says which ceiling ended the job — test failure, timeout, or OOM — so the grade is explainable.

Key Takeaways

1. Isolation is a ladder: \$PATH and conda redirect names, namespaces and cgroups enforce, a VM removes the shared kernel.
2. The security property is the delegation list — what a job may name, mount, connect to, and consume.
3. The supervisor owns the process tree, the cleanup, and the receipt.