

# Lesson 08 — Sandboxing and Least Privilege for AI Agents

An autonomous agent is a non-deterministic process with a toolbelt. That is not a thing you hand the keys to. This lesson is the five-layer sandbox you put around it before you let it run.

Audience: Devs + Security + Operators

AI OpSec

Sandboxing

**Notes:** The goal is not to trust the agent. The goal is to make it safe to run an untrusted agent.

# What you will learn

- **Containerization** — isolate the agent process, filesystem, and user namespace from the host
  - **Network isolation** — internal-only bridge, no outbound, explicit allowlists when you need egress
  - **Filesystem scoping** — read-only mounts, named volumes for writable state, nothing else
  - **Capability drops** — drop ALL, add back only what the agent demonstrably needs
  - Apply all four to a working reference: `ctf-labs-env/`
-

# The agent threat model

An autonomous agent has three components. Any of the three can be turned against you.

- **(a) The LLM** — a non-deterministic, prompt-injectable component that decides what to do next
- **(b) The toolbelt** — code that runs on your machine when the LLM calls it. Shell, filesystem, network, browser
- **(c) The state** — the agent's memory, its scratch files, its tokens, its working directory

Prompt-inject the model, the toolbelt executes the attacker's intent on your state. That is the whole game.

---

# "I trust the model" is not a control

**A non-deterministic component cannot be a security control.**

Models are prompt-injectable, output-unstable, and their behavior changes between versions. Anything that depends on the model "deciding not to do the bad thing" is not a control — it is a hope.

- Defense in depth: assume the model *will* be compromised and design the layers below it to contain the blast
  - The layers below the model are deterministic: kernels, namespaces, cgroups, iptables. Those are controls.
-

# Sandbox layer 1: containerization

- A container is a process with its own PID namespace, mount namespace, network namespace, and (optionally) user namespace
- The kernel enforces the boundaries — not the agent, not the runtime, not a linter. The kernel.
- Run the agent as a non-root user inside the container, and enable user namespace remapping on the host if your runtime supports it
- `no-new-privileges:true` — a process cannot gain privileges via setuid binaries inside the sandbox

Containerization alone is not enough. It is the substrate on which the other four layers sit.

---

# Sandbox layer 2: network isolation

- **Internal-only bridge:** Docker network with `internal: true`. The container has no route to the host and no route to the internet.
- **Default deny:** nothing egresses unless you explicitly added a path for it
- **Explicit allowlists:** if the agent genuinely needs outbound, put an allowlist proxy in front of it — never give the agent raw internet

```
networks:  
  isolated:  
    driver: bridge  
    internal: true
```

One line turns off the entire egress attack surface.

---

# Sandbox layer 3: filesystem scoping

- **Read-only by default:** mount host paths with `:ro`. The agent can read, cannot write.
  - **Writable state in named volumes:** one volume, one path, everything else read-only
  - **No bind-mounting** `/`, `/home`, or `~/.ssh` — ever. Not even "just for this one test."
  - **Principle:** an attacker who compromises the agent should find nothing on disk worth stealing and nowhere to persist
-

# Sandbox layer 4: capability drops

- Linux capabilities are the fine-grained pieces of "root." Drop them all, then add back only what you need.
- `cap_drop: - ALL` — zero capabilities by default
- `cap_add:` — a short, justified list. Each added cap is a documented exception.
- If you cannot justify a capability in one sentence, it does not go in the list

```
cap_drop:
- ALL
cap_add:
- CHOWN
- SETUID
- SETGID
- DAC_OVERRIDE
```

# Sandbox layer 5: loopback-only port publishing

- Docker's default `- "6080:6080"` publishes on `0.0.0.0` — every host on your LAN can reach it
- Bind to loopback instead: `- "127.0.0.1:6080:6080"`
- Inbound DNAT still works for the local user; no one on the LAN, the coffee shop, or the corporate VPN can hit it
- This is a one-character change that fails closed for most remote attackers

```
ports:  
- "127.0.0.1:6080:6080"
```

---

# Reference architecture: `ctf-labs-env/`

The lab environment for this course already applies all five layers. The relevant bits of the compose file:

```
services:
  lab:
    container_name: ctf-lab
    ports:
      - "127.0.0.1:6080:6080"
    networks:
      - isolated
    security_opt:
      - no-new-privileges:true
    cap_drop:
      - ALL
    cap_add:
      - CHOWN
      - SETUID
      - SETGID
```

Five layers, one file. Use this as a template for agents you build.

---

# What the agent can still do inside the sandbox

Least privilege does not mean zero privilege. An agent in this sandbox can still:

- Read its workspace (the mounted repo or task directory)
- Execute tools and spawn subprocesses within the container
- Write to its named scratch volume at a single defined path
- Accept inbound connections on published loopback ports from the host operator

What it **cannot** do:

- Reach the internet (no egress route)
  - Escape the filesystem (everything else is read-only or absent)
  - Elevate privileges (`no-new-privileges`, caps dropped)
  - Be reached from the LAN (loopback-only bind)
-

# Rules to remember

- "I trust the model" is not a security control.
- Default deny, explicit allow: nothing reaches out unless you added it.
- Containerize the agent process. Filesystem, network, caps — all scoped.
- Use an internal-only Docker bridge for agents; inbound DNAT still works on a loopback-bound port.
- Drop all capabilities, then add back only what the agent demonstrably needs.
- Writable state goes into a named volume at a single path. Everything else is read-only.

