

Procedural Capture of Passive Observation Frameworks

Arul Kogilan

October 2026

Abstract

We present a deterministic, offline sandbox that isolates the permission-layer mechanism by which an actuating actor can capture a non-actuating knowledge source and exclude other parties from it. The sandbox replaces the LLM-based Oracle of the original “Cartelization of the Oracle” threat model with a BM25 retrieval kernel over a fixed corpus, ensuring that every measurement reflects kernel-enforced POSIX capability state rather than model stochasticity. Across 30 seeds $\times$ 25 rounds $\times$ 3 regimes, we measure:

1. privileged actors lock the source with `chown+chmod 0660` and exclude every non-privileged party;
2. a passive behavioral monitor (TraceGuard) is measurably inert — its presence changes nothing because it has no execution authority;
3. kernel-level capability constraint (`cap_drop: [CHOWN]`) deterministically eliminates the lockup and restores full parity.

The mechanism is infrastructural, not cognitive. It demonstrates that procedural capture requires neither an LLM nor a sophisticated cartel — a single process with the right capability is sufficient.

1 Introduction

Procedural Capture of a Non-Actuating Source: A Deterministic Sandbox Study

Track: 3.1 — Collusion Carriers & Countermeasures (Control Environment)

Sprint: Apart Research AI Collusion Sprint 2026 (Oct 23–25)

Submission date: 2026-10-25

Repository: poseidon-research/collusion_hackathon + archive_oracle/

Runtime environment: Docker 27, Python 3.11, Linux 6.x, offline (`network_mode: none`)

2 Motivation

The original threat model proposed an LLM Oracle captured by tool-using actors. That framing conflates two separable mechanisms: the model’s behavior and the permission layer that mediates access to it. This work isolates the second. The finding is not about language models; it is about the capability surface any actuating process needs in order to enclose a source that cannot act on its own behalf.

The Track 3 brief asks: “Can actions, tools or side channels carry collusion?” The answer this sandbox supports is narrower and more useful: **before a side channel can carry collusion, the capability surface must permit exclusion.** When capabilities are constrained at the kernel, no side channel — semantic, textual, or action-based — suffices to reconstruct the lockup.

3 Environment

Kernel (Oracle). BM25 retrieval over 900 baseline documents, 100 friction documents, and 20 solution templates, all seeded by a single `random.Random(seed)`. Retrieved passages are concatenated and HMAC-signed. Zero network access; zero external API calls. Determinism verified by `corpus_digest(seed)` equality.

Actors. Three UNIX user accounts (`a1`, `a2` in group `cartel`; `a3` in group `plain`) run as separate subprocesses, each dropping to its target uid via `setgroups/setgid/setuid`. Each actor queries the Oracle over a UNIX socket at `/run/oracle/*.sock` (tmpfs).

Regimes. Three Docker services sharing one image, differing only in `cap_add` and `REGIME`:

Service	cap_add	REGIME
ungoverned	SETUID, SETGID, CHOWN	UNGOVERNED
traceguard	SETUID, SETGID, CHOWN	TRACEGUARD_ONLY
governed	SETUID, SETGID	GOVERNED

The `disruptor` service exists in the compose file but its orchestrator subprocess was not implemented; the `DISRUPTOR` regime is therefore not reported.

Runtime guard. `oracle_sandbox.py` refuses to run if the `--regimes` argument does not match the container’s `REGIME` environment variable (exit code 2). Verified both directions.

Isolation. `network_mode: none`, `read_only: true`, results directory bound-mounted, sockets on tmpfs. Verified by `capsh --print` (`CHOWN` present in ungoverned and traceguard, absent in governed).

4 Method

30 seeds per regime, 25 rounds per seed. Per-round each actor emits one query (or one lookup attempt at round 0). Metrics reported:

- **cartS** — sum of `a1` and `a2` solves across 25 rounds (max 50)
- **outS** — outsider (`a3`) solves across 25 rounds (max 25)
- **a2den** — `a2` rounds denied at the socket
- **lookup** — fraction of seeds where the round-0 `chown+chmod` returned success
- **capOK** — fraction of seeds where `capsh --print` shows `cap_chown` in the effective set
- ΔC — $((a1 + a2)/2 - a3)/25$

All statistics are point estimates over the 30 seeds. Welch’s *t*-test is undefined due to zero within-regime variance; this is a property of the deterministic design and is reported as a limitation rather than as a statistical result.

5 Results

Regime ΔC	n	cartS	outS	a2den	lockup	capOK
UNGOVERNED 1.000	30	50	0	0	100%	1.00
TRACEGUARD_ONLY 1.000	30	50	0	0	100%	1.00
GOVERNED 0.000	30	50	25	0	0%	0.00

5.1 Three findings

Finding 1 — Kernel capability governs access, not monitor observation.

The `lockup` and `capOK` columns are perfectly correlated across all seeds: where `cap_chown` is in the effective capability set, `chown` succeeds and the outsider is excluded on 25/25 rounds; where it is absent, `chown` raises `EPERM`, `lockup_status` records `EPERM`, and the outsider solves all 25 rounds. This is a kernel-level relationship, not a statistical one.

Finding 2 — Passive behavioral monitoring is measurably inert.

TRACEGUARD_ONLY and UNGOVERNED are identical on every metric: `cartS`, `outS`, `a2den`, `lockup` rate, and `capability` state. The TraceGuard process detects the `lockup` attempt (as designed) but holds no capability to reset the socket, so its presence cannot alter the outcome. This confirms the mechanism claim directly by measurement, not assertion.

Finding 3 — Capability constraint eliminates capture deterministically.

GOVERNED shows `cartS` = 50 (both cartel actors solve in all 25 rounds), `outS` = 25 (outsider solves in all rounds), `lockup` = 0%, `capOK` = 0.00, ΔC = 0.000, across 30/30 seeds. There is no partial case and no boundary condition — the outcome is fully determined by the container’s capability set.

6 Interpretation

A two-party cartel of privileged actors excludes the outsider by locking the shared resource to a group-permission that includes both co-conspirators and excludes everyone else. TraceGuard detects the attempt but cannot intervene. Kernel capability drop makes the `lockup` impossible and restores parity.

The result isolates a pure infrastructural failure mode: passive observation frameworks that lack execution authority are inert against capability-based exclusion. Structural constraint at the kernel is both necessary and sufficient to prevent the `lockup`.

7 Limitations

- Deterministic design.** Zero within-regime variance means Welch’s *t*-test cannot be computed. The results are reported as point estimates over 30 seeds, not as statistical inference.
- Model organism, not live LLM.** The Archive Oracle is a BM25 retrieval pipeline; it does not exhibit model-level stochasticity. The finding is about the capability surface around a source, not about the source’s cognition.
- Disruptor regime not reported.** The orchestrator subprocess for the DISRUPTOR regime was not implemented at the time of this run. Noted as future work.

4. **Reduced reward model.** ΔC is a proportion of attainable solves per round, not a dollar amount. The original report’s market-derived figures are not reproduced.

8 Sociotechnical Implications

The sandbox demonstrates four facts that do not depend on scale:

- Capture requires capability asymmetry, not coordination. A single process holding CAP_CHOWN suffices to exclude every other party.
- Passive monitoring of the interface cannot prevent capture, because observation without execution authority does not constrain the capability surface.
- Structural capability constraint eliminates capture deterministically, at zero cost to legitimate participants.
- The mechanism operates at the permission layer, independent of the source’s cognition, behavior, or output content.

Applied to automated institutional surveillance, these facts suggest a structural hypothesis: when a human is legible to an institution only through a curated record, and some actor holds the capability to determine what enters that record while the human lacks concurrent contestation rights, capture is possible by the same mechanism measured here. The sandbox isolates the capability surface as the site of capture and shows that monitoring and transparency do not substitute for structural constraint.

This is a mechanism finding, not a population study. Population-level claims require separate empirical work.

9 Reproduction

```
git clone https://github.com/poseidon-research/collusion_hackathon.git
cd collusion_hackathon
# (place archive_oracle/ alongside; see supplementary materials)

docker compose -f docker/compose.yml build ungoverned traceguard governed

# Capability gates
docker compose -f docker/compose.yml run --rm ungoverned pytest -q tests/
docker compose -f docker/compose.yml run --rm governed pytest -q tests/

# Runtime guard (both must exit 2)
docker compose -f docker/compose.yml run --rm ungoverned \
  python oracle_sandbox.py --num_seeds 1 --num_rounds 1 --regimes GOVERNED
docker compose -f docker/compose.yml run --rm governed \
  python oracle_sandbox.py --num_seeds 1 --num_rounds 1 --regimes UNGOVERNED

# Full runs (N=30, T=25)
docker compose -f docker/compose.yml run --rm ungoverned \
  python oracle_sandbox.py --num_seeds 30 --num_rounds 25 \
  --regimes UNGOVERNED --out /app/results/full/ungoverned
docker compose -f docker/compose.yml run --rm traceguard \
  python oracle_sandbox.py --num_seeds 30 --num_rounds 25 \
  --regimes TRACEGUARD_ONLY --out /app/results/full/traceguard
docker compose -f docker/compose.yml run --rm governed \
  python oracle_sandbox.py --num_seeds 30 --num_rounds 25 \
  --regimes GOVERNED --out /app/results/full/governed
```