Security

Formal Verification (Security Models)

This page tracks OpenClaw's **formal security models** (TLA+/TLC today; more as needed).

> Note: some older links may refer to the previous project name.

Goal (north star): provide a machine-checked argument that OpenClaw enforces its intended security policy (authorization, session isolation, tool gating, and misconfiguration safety), under explicit assumptions.

What this is (today): an executable, attacker-driven **security regression suite**:

- Each claim has a runnable model-check over a finite state space.
- Many claims have a paired **negative model** that produces a counterexample trace for a realistic bug class.

What this is not (yet): a proof that "OpenClaw is secure in all respects" or that the full TypeScript implementation is correct.

Where the models live

Models are maintained in a separate repo: **vignesh07/openclaw-formal-models**.

Important caveats

- These are **models**, not the full TypeScript implementation. Drift between model and code is possible.
- Results are bounded by the state space explored by TLC; “green” does not imply security beyond the modeled assumptions and bounds.
- Some claims rely on explicit environmental assumptions (e.g., correct deployment, correct configuration inputs).

Reproducing results

Today, results are reproduced by cloning the models repo locally and running TLC (see below). A future iteration could offer:

- CI-run models with public artifacts (counterexample traces, run logs)
- a hosted “run this model” workflow for small, bounded checks

Getting started:

```
git clone https://github.com/vignesh07/openclaw-formal-models
cd openclaw-formal-models

# Java 11+ required (TLC runs on the JVM).
# The repo vendors a pinned `tla2tools.jar` (TLA+ tools) and provides `bin/tlc` +

make <target>
```

Gateway exposure and open gateway misconfiguration

Claim: binding beyond loopback without auth can make remote compromise possible / increases exposure; token/password blocks unauth attackers (per the model assumptions).

Green runs:

```
make gateway-exposure-v2
make gateway-exposure-v2-protected
```

Red (expected):

```
make gateway-exposure-v2-negative
```

See also: `docs/gateway-exposure-matrix.md` in the models repo.

Nodes.run pipeline (highest-risk capability)

Claim: `nodes.run` requires (a) node command allowlist plus declared commands and (b) live approval when configured; approvals are tokenized to prevent replay (in the model).

Green runs:

```
make nodes-pipeline
make approvals-token
```

Red (expected):

```
make nodes-pipeline-negative
make approvals-token-negative
```

Pairing store (DM gating)

Claim: pairing requests respect TTL and pending-request caps.

Green runs:

```
make pairing
make pairing-cap
```

Red (expected):

```
make pairing-negative
```

```
make pairing-cap-negative
```

Ingress gating (mentions + control-command bypass)

Claim: in group contexts requiring mention, an unauthorized “control command” cannot bypass mention gating.

Green:

```
make ingress-gating
```

Red (expected):

```
make ingress-gating-negative
```

Routing/session-key isolation

Claim: DMs from distinct peers do not collapse into the same session unless explicitly linked/configured.

Green:

```
make routing-isolation
```

Red (expected):

```
make routing-isolation-negative
```

v1++: additional bounded models (concurrency, retries, trace correctness)

These are follow-on models that tighten fidelity around real-world failure modes (non-atomic updates, retries, and message fan-out).

Pairing store concurrency / idempotency

Claim: a pairing store should enforce `MaxPending` and idempotency even under interleavings (i.e., “check-then-write” must be atomic / locked;

refresh shouldn't create duplicates).

What it means:

- Under concurrent requests, you can't exceed `MaxPending` for a channel.
- Repeated requests/refreshes for the same `(channel, sender)` should not create duplicate live pending rows.
- Green runs:
  - `make pairing-race` (atomic/locked cap check)
  - `make pairing-idempotency`
  - `make pairing-refresh`
  - `make pairing-refresh-race`
- Red (expected):
  - `make pairing-race-negative` (non-atomic begin/commit cap race)
  - `make pairing-idempotency-negative`
  - `make pairing-refresh-negative`
  - `make pairing-refresh-race-negative`

Ingress trace correlation / idempotency

Claim: ingestion should preserve trace correlation across fan-out and be idempotent under provider retries.

What it means:

- When one external event becomes multiple internal messages, every part keeps the same trace/event identity.
- Retries do not result in double-processing.
- If provider event IDs are missing, dedupe falls back to a safe key (e.g., trace ID) to avoid dropping distinct events.

Green:

```
make ingress-trace
make ingress-trace2
make ingress-idempotency
make ingress-dedupe-fallback
```

Red (expected):

```
make ingress-trace-negative
make ingress-trace2-negative
make ingress-idempotency-negative
make ingress-dedupe-fallback-negative
```

Routing dmScope precedence + identityLinks

Claim: routing must keep DM sessions isolated by default, and only collapse sessions when explicitly configured (channel precedence + identity links).

What it means:

- Channel-specific dmScope overrides must win over global defaults.
- identityLinks should collapse only within explicit linked groups, not across unrelated peers.

Green:

```
make routing-precedence
make routing-identitylinks
```

Red (expected):

```
make routing-precedence-negative
make routing-identitylinks-negative
```

‹ Tailscale Web ›

›