

# PLAN\_RISK\_GUARD\_TELEGRAM\_RATE\_LIMIT\_P3

**Status:** planned (backlog item, awaits operator GO) **Priority:** P3 **Erstellt:** 2026-05-17 **Quelle:** deferred from PLAN\_T1\_ROADMAP\_2\_WEEKS\_v3.4 Phase D (T1-RISK-GUARD-1) **Roadmap-ID:** RISK-GUARD-TELEGRAM-RATE-LIMIT

## 1. Motivation / Why

Plan v3.4 §8 letzter Absatz:

Telegram-Spam-Reduktion: - per-Block-Telegram raus - statt: rate-limited summary alle 60min (max 1 Nachricht) - echte Runtime-Anomalien (z.B. Exchange-Error) bleiben Realtime

In Phase D wurde der Risk-Guard live, aber das Notif-Refactoring blieb deferred, weil:

- Phase D blockt aktuell nur DCA-Rescues. Diese sind keine zwingenden Telegram-Events vor dem Refactor.
- Die Block-Reasons werden bereits ins `bot_stdout.log` (logger.info mit 📢 prefix) und in `pos['_last_risk_guard_block']` geschrieben.
- Kein realer Telegram-Spam ist daher heute aktiv; es entsteht erst wenn ein Operator-getriggert `send_*_alert` -Pfad an Risk-Guard- Events angedockt wird.

Zweck dieser Phase: Saubere, rate-limited Operator-Sichtbarkeit, **ohne** neuen Spam-Vektor zu eröffnen.

## 2. Aktueller Zustand (2026-05-17, post Phase D)

- `trading/risk/risk_guard.py` liefert `GuardDecision(allowed, reason, meta)`.
- `trading/execution/paper_trade.py` DCA-Rescue: bei Block → `logger.info(" 📢 T1-RISK-GUARD-1: DCA-Rescue blocked for {symbol} ...")`
- `pos['_last_risk_guard_block'] = decision.to_dict()`.
- Telegram-Aufrufe gibt es heute nur über `reports/reporter.py` (`send_dca_rescue_alert`, `send_*` etc.) auf erfolgreiche / fehlerhafte Events — Risk-Guard-Blocks werden dort NICHT durchgereicht.
- Telegram-HTML-Escape ist seit `NOTIFIER-DCA-RESCUE-HTML-ESCAPE` (commit 59489a5) korrekt.

## 3. Zielzustand

```
Risk-Guard fires block:
├─ always: logger.info 📢 block (bestehend)
├─ always: pos['_last_risk_guard_block'] (bestehend)
└─ neu: Notif-Aggregator nimmt Event mit Reason+Counter auf
      │
      │ ── Cooldown-Fenster offen?
      │      └─ NEIN → flush vorheriges Summary nach Telegram
      │              Fenster neu starten (z.B. 60min)
      │
      └─ Realtime-Eskalation für ausnahmen:
            exchange_error / traceback / mainnet_touch
            → send sofort (kein Throttle)
```

Effekte:

- max 1 Summary-Telegram pro 60 Minuten ueber alle Risk-Guard-Blocks
- Realtime nur fuer echte Runtime-Anomalien
- keine Verhaltens-Änderung im DCA-/SL-Pfad
- Block-Reasons + Counts pro Reason im Summary

## 4. Scope

Layer	Änderung
neu	<code>trading/notifications/risk_event_aggregator.py</code> — Pure-data Aggregator + Throttle-Logik
edit	<code>trading/reports/reporter.py</code> — <code>send_risk_guard_summary(payload)</code> hinzufügen
edit	<code>trading/execution/paper_trade.py</code> DCA-Rescue Block-Site — Aggregator <code>.record(...)</code> aufrufen
edit	<code>trading/main.py</code> — pro Scan-Cycle <code>aggregator.maybe_flush(reporter)</code> rufen
tests	<code>tests/test_risk_event_aggregator.py</code> — Throttle, Aggregation, Realtime-Bypass

## 5. Aggregator-Skizze

```
class RiskEventAggregator:
    def __init__(self, *, window_seconds: int = 3600, realtime_reasons: tuple = (...)):
        self._events: list[dict] = []
        self._window_seconds = window_seconds
        self._last_flush_ts: float | None = None
        self._realtime = set(realtime_reasons)

    def record(self, *, reason: str, symbol: str, meta: dict) -> dict:
        """Returns {'realtime': bool, 'event': {...}}."""
        ...

    def should_flush(self, now_ts: float) -> bool:
        ...

    def build_summary(self) -> dict | None:
        """Returns aggregated payload or None when nothing to flush."""
        ...

    def maybe_flush(self, reporter, now_ts: float | None = None) -> bool:
        ...
```

## 6. Telegram-Payload

```
♥ Risk-Guard Summary (last 60min)
├─ DCA blocked: 3 events
│   ├── dca_blocked_bear_regime: 2x (KITE, LAYER)
│   └── dca_max_loss_exceeded: 1x (ENA)
└─ i No realtime-anomalies in window.
```

(HTML-Escape via existing `html.escape` pattern.)

## 7. Tests (Plan)

- `record()` + `maybe_flush()` ohne Events → no-op.
- `record(reason='dca_blocked_bear_regime')` x3 in 5min, dann `maybe_flush` nach 65min → genau 1 Summary mit `count=3`.
- Realtime reason (`exchange_error`) → `record` returnt `realtime=True`, Summary-Counter unbenommen.
- Fenster reset nach Flush; nachfolgende Events landen im neuen Fenster.
- `html.escape` auf Symbol-Strings.

## 8. Boundaries (gelten für die Phase)

- 0x Strategieparameter-Tuning
- 0x Risk-Guard-Logikänderung (D-Verhalten bleibt identisch)
- 0x neue Telegram-Bot-Permissions
- 0x DB-Migration
- 0x Mainnet
- 0x CommandBus-v6
- 0x Worker-Recreate
- 0x Push
- 0x Secrets / env dump

## 9. Cutover-Plan

Standard-SOT-1d (analog Phase D):

1. Pre-cutover Snapshot (HEAD, container-PIDs, env-flags).
2. Watchdog freeze: `CUTOVER_FREEZE_RISK_GUARD_TELEGRAM`.
3. `docker compose build clawbot`.
4. Container-Test im neuen Image: alle Risk-Guard- und neue Aggregator-Tests grün (rev-comprehensive run gegen alle Phase-D-Tests).
5. `docker compose up -d --force-recreate --no-deps clawbot`.
6. 3-Way MD5 für betroffene Files.
7. Bot spawn + Healthcheck + 0 Tracebacks.
8. Live-Smoke (Container): `from notifications.risk_event_aggregator import RiskEventAggregator agg =`

```
RiskEventAggregator(window_seconds=2) agg.record(reason='dca_blocked_bear_regime', symbol='TEST', meta={})
time.sleep(3) agg.maybe_flush(_dummy_reporter) # erwartet: exactly 1 summary built
```

9. Watchdog re-enable.

10. Roadmap-Update RISK-GUARD-TELEGRAM-RATE-LIMIT → done.

## 10. Optional Follow-up

---

Nicht in dieser Phase:

- Risk-Guard-Block-Events in `decision_logs.metadata_json` einbetten (DB-Persistenz für GUI-Anzeige). Eigenes Backlog-Item, falls operator später eine `/admin/risk-guard-events` -Seite haben will.

## 11. Erwartete Lieferung

---

- Commit-Vorschlag: `risk-guard-telegram-rate-limit: aggregate risk-guard blocks into hourly summary`
- Tests: erwartet ~6-10 neue Tests, gesamt-Suite muss bei 100% bleiben.
- Closure: dieselbe Form wie Phase D.

## 12. STOP

---

Kein Code vor erneutem Operator- `G0 RISK-GUARD-TELEGRAM-RATE-LIMIT` .

Bis dahin: \* Bot loggt Risk-Guard-Blocks via `bot_stdout.log` (logger.info). \* `pos['_last_risk_guard_block']` als observability-Pfad bleibt. \* Operator kann Block-Verteilung jederzeit aus den Logs ziehen.