Operator-Drill Testnet 502 — Scope-Bestätigung

Steve Trading Bot

Datum: 2026-05-08 (UTC)

Master-Tip: `043e3b4`

Bot-PID: 3736 (B-OUTAGE-RESILIENCE-1 aktiv seit 2026-05-08 18:47:13 UTC)

Modus: vor Ausführung — Scope-Lock und Vorbedingungen.

---

1. Methodik (Tier 1 — Mocked-Exchange Drill, NICHT live bot)

Kerngedanke: Eine dedizierte Drill-Harness verifiziert die Resilience-Mechanik, **ohne** dass der laufende Bot oder echte Exchange-Endpunkte berührt werden.

Komponente	Vorgehen
Drill-Skript	`trading/scripts/operator_drill_outage_502.py` (NEU, scoped commit)
Test-Harness	identisch zur Unit-Test-Pattern aus `test_b_outage_resilience_1_*`
Exchange	`unittest.mock.MagicMock` mit injizierten 502/Timeout/RateLimit-side-effects
LiveTrader	via `LiveTrader.__new__(LiveTrader)` (skip `__init__`) — kein echter ccxt-Connect
Ausführung	`docker run --rm python:3.11-slim …` — separater Prozess
Bot bleibt aktiv	PID 3736 läuft normal weiter; der Drill berührt seinen Adressraum nicht

2. Was gemockt/gepatcht wird

```
# Drill-Harness (skizziert):
ex = MagicMock()
ex.markets = {'SOL/USDT': {'active': True, 'spot': True}, ...}

# Phase 1: 502-Storm
ex.fetch_balance.side_effect = ccxt.ExchangeNotAvailable("502 Bad Gateway")
ex.create_market_buy_order.side_effect = ccxt.ExchangeNotAvailable("502")
ex.load_markets.side_effect = ccxt.ExchangeNotAvailable("502")

# Phase 2: Recovery
ex.fetch_balance.side_effect = None
ex.fetch_balance.return_value = {'USDT': {'free': 10000}}

lt = LiveTrader.__new__(LiveTrader)
lt._exchange = ex
lt._circuit_breaker = CircuitBreaker(...)
```

Keine Funktion am laufenden Bot wird gepatcht. Der Drill ist eine reine in-process-Simulation in einem throwaway-Container.

3. Wie verhindert wird, dass echte Orders ausgelöst werden

Schutzschicht	Garantie
1	Drill läuft im **eigenen** `python:3.11-slim` **Container** (Ephemeral), nicht im clawbot-Container
2	Keine `BINANCE_API_KEY`-Übergabe als ENV in den Drill-Container — Auth schlägt fehl auch wenn ccxt-Call erfolgte
3	`LiveTrader._exchange` ist **immer** `MagicMock`, niemals echte ccxt-Instanz
4	Source-grep im Drill-Skript via AST: `ccxt.binance(...)`, `_get_exchange(...)` etc. müssen 0× vorkommen
5	Bot's `live_portfolio.json` wird zum Audit nur via SHA-Hash beobachtet, nicht gelesen/mutiert durch den Drill

4. Drill-Punkte (8 Phasen)

#	Punkt	Mock-Setup	Erwartetes Verhalten	Validierung
1	Exception-Klassifikation	5xx/Timeout/RateLimit/Auth Exceptions injizieren	`classify(exc) -> (category, reason_token)` exakte Tokens	`assertEqual` auf reason + category
2	Retry/Backoff	`fetch_balance` failt 5x mit 502 dann success	5 Versuche mit exp.Backoff (mit `sleep_fn` no-op)	`call_count == 5`
3	Circuit-Breaker State	Threshold-1 transient failures, dann +1 = trip	`breaker.state` Closed -> Open	snapshot() + assertEqual(state, "open")
4	Buy-Pause	breaker open + `execute_buy()` aufgerufen	returns None mit Log `outage_circuit_breaker_active`; create_market_buy_order NICHT aufgerufen	mock.call_count == 0
5	Sell-Failure	`create_market_sell_order` failt 502 in execute_sell	EINE Order-Versuch (single-attempt), reason classified, return None	call_count == 1
6	Markets-Cache Resilience	`load_markets` failt + recovery	empty cache -> lazy reload -> 5 retries + position-aware `_is_tradable=True` für gehaltene Positions	snapshot + assertions
7	Recovery + Half-Open	clock advance über cooldown, dann success-probes	open -> half_open -> closed nach probes	breaker.state == "closed"
8	Logging-Telemetry	log-Capture während Drill	exakte Substrings: `[outage-retry]`, `paused: outage_circuit_breaker_active`	assertIn

5. Erwartete Log-Pattern

```
[outage-retry] fetch_balance.buy_pre attempt 1/5 failed category=transient reason=exchange_502 backo
[outage-retry] fetch_balance.buy_pre attempt 2/5 failed category=transient reason=exchange_502 backo
...
[outage-retry] fetch_balance.buy_pre retries exhausted (5/5) category=transient reason=exchange_502
TESTNET BUY <SYM> blocked: balance_fetch_failed underlying_reason=exchange_502 category=transient exe
TESTNET BUY <SYM> paused: outage_circuit_breaker_active breaker={state: open, ...}
[outage-retry] create_market_sell_order non-retryable category=permanent reason=insufficient_funds a
TESTNET-Markets refreshed: 2152 Pairs (recovery phase)
```

6. Erwartete CircuitBreaker-States

Phase	Vorgang	Erwarteter State
Init	nach `CircuitBreaker(...)`	`closed`
Trip	nach `failure_threshold` consecutive transient failures	`open`
Cooldown active	innerhalb `cooldown_seconds` (Default 1800s — im Drill 5s via `time_fn`)	`open`
Cooldown elapsed	nach Cooldown	`half_open`
Probe success	nach `half_open_required_successes` (Default 2)	`closed`
Probe failure	bei Failure in HalfOpen	zurück auf `open` mit Reset-Cooldown

7. Stop-Regeln

Bedingung	Action
Bot-PID 3736 ändert sich während des Drills	sofort STOP, Bericht
`.env` mtime ändert sich	sofort STOP
live_portfolio.json hash ändert sich anders als durch normalen Bot-Trade	sofort STOP
Drill-Test schlägt unerwartet fehl	STOP, Diagnose
Echter ccxt-Call würde versucht (AST-Quelltext-Grep zeigt verbotene Imports/Calls)	Drill würde gar nicht ausgeführt
Drill-Container hängt > 5min	STOP via Timeout

8. Wie Recovery verifiziert wird

```
# Nach trip + cooldown:
clock.advance(cooldown_seconds + 1)
assert breaker.state == "half_open"
ex.fetch_balance.side_effect = None
ex.fetch_balance.return_value = {"USDT": {"free": 10000}}
out1 = lt._call_read(ex.fetch_balance, op_label="probe1")
out2 = lt._call_read(ex.fetch_balance, op_label="probe2")
assert out1.ok and out2.ok
assert breaker.state == "closed"

# allow_call("buy") wieder True
allowed, reason = breaker.allow_call("buy")
assert allowed is True and reason is None
```

9. Post-Drill-Checks (Außerhalb des Drill-Containers)

Check	Quelle	Erwartet
Bot-PID = 3736 unverändert	`docker exec clawbot ls /proc/3736`	exists
Bot-PID-cmdline unverändert	proc cmdline	`python3 main.py --paper`
`.env` mtime unverändert	`stat -c %Y`	`1777991334`
live_portfolio.json — keine durch Drill ausgelösten Mutations	sha256 vor/nach Drill	Drift erlaubt nur wenn Bot währenddessen normal getradet hat
`bot_statuses` — keine Drill-bedingten Einträge	DB metadata_json filter	0
`commands` table	`SELECT count(*)`	1 (G6.5-Demo, unverändert)
Real testnet orders — keine durch Drill	Bot-Log-Search	nur reguläre Bot-Trades

10. Drill-Skript-Struktur (geplant, ~250 Zeilen)

```
trading/scripts/operator_drill_outage_502.py

Sections:
  1. Imports (NO ccxt.binance constructors, NO _get_exchange — AST-validated)
  2. Drill-Harness (make_bare_livetrader_for_drill mit MagicMock exchange)
  3. Phase 1 — 502-Storm: fetch_balance + create_market_*_order failt 502
  4. Phase 2 — Timeout: RequestTimeout
  5. Phase 3 — RateLimit: RateLimitExceeded
  6. Phase 4 — Trip CircuitBreaker via N consecutive transient failures
  7. Phase 5 — Buy-Pause: BUY blocked by open breaker
  8. Phase 6 — Sells-Allowed: SELL goes through even with breaker open
  9. Phase 7 — Markets-Cache: load_markets fails + lazy reload + position-aware fallback
  10. Phase 8 — Recovery: cooldown elapses, half_open, success-probes, breaker closes
  11. Final assertion summary + JSON-Report
```

Output: ein JSON-Report `/tmp/operator_drill_outage_502_<UTC>.json` mit Pass/Fail pro Phase + Log-Capture.

11. Tests/Checks nach dem Drill

1. Drill-Skript schlägt 0/N Phasen fehl
2. Bot-Health-Sweep (gleiche Checks wie nach STATE-RECON-1-Apply)
3. Existing Bot-Test-Suite läuft nochmal (282/282) als Regression-Sweep
4. Memory-Save: `operator_drill_outage_502_status.md` mit Drill-Resultat

12. Boundary-Bestätigungen für den Drill

Punkt	Status
Keine echten Mainnet-Calls	OK (separate Container, MagicMock-only)
Keine echten Testnet-Orders	OK (Drill-Skript hat 0 ccxt-Imports — AST-validated)
Keine `live_portfolio.json`-Mutation	OK (Drill liest nicht mal die Datei)
Keine `.env`-Mutation	OK (Drill-Container hat ENV nur für PYTHONPATH)
Kein G10 / Worker / Telegram-Hook	OK (Drill scope schließt explizit aus)
Kein Push	OK (Repo hat kein Remote)
Bot bleibt aktiv	OK (Drill ist out-of-process)

13. Follow-up (notiert, nicht jetzt bearbeitet)

> **B-FEE-FIX-4 — Empty-fee fallback in `_parse_filled`**: Aktuelle Beobachtung aus POST-RECON-AUDIT: ALGO/APE/POL/ZRO haben alle `fee_buy=0.0`, vermutlich weil ccxt-Binance-Testnet keine fee-Felder in market-order-Responses liefert und `_extract_fee_in_quote` dann 0 zurückgibt. Der `cost × TRADE_FEE`-Fallback existiert aktuell nur im balance-diff-verify-Branch von `execute_buy`, nicht im normalen `_parse_filled`-Pfad. Sinnvoller Mini-Phase-Fix nach dem Drill. Nicht blockierend.

Stand: 2026-05-08 19:27 UTC. Vor Drill-Ausführung. Erfordert explizites GO.