SEC-1c-3b Bericht

Bot+Worker DB-Cutover auf `tradingbot_gui_app` & anschließende Container-Bereinigung

Datum: 2026-05-14 (UTC) | Phase: SEC-1c-3b CLOSED | Stack: `steve-tradingbot` | master HEAD: `d351a3a`
Closure-Memo: `memory/sec_1c_3b_closure_20260514.md`

CUTOVER ERFOLGREICH **0 ROLLBACK** **RESTART-WINDOW < 5 MIN** **CLEANUP EMPFOHLEN**

> **Kurzfassung für eilige Leser.** Bot und Worker schreiben jetzt mit der eingeschränkten Datenbank-Rolle `tradingbot_gui_app` in `tradingbot_gui`. Die legacy SUPERUSER-Rolle ist aus dem Runtime-Pfad raus. Worker schreibt seit 07:37:31 UTC alle 30 Sekunden saubere Heartbeats; Bot scant TESTNET LIVE. Es gibt 2.3 GB verwaiste Container-Volumes aus dem Volume-Prefix-Drama während des Cutovers — die sind unbenutzt und können gefahrlos entfernt werden. Zusätzlich liegen 17.7 GB reclaimable Images und 44 GB Build-Cache rum. Insgesamt ~64 GB frei machbar.

1. Was ist passiert

1.1 Ausgangslage (vor 07:30 UTC)

Bot (`clawbot`) und Worker (`clawbot-worker`) verbanden sich beide mit der Postgres-Datenbank `tradingbot_gui` über die Rolle `tradingbot_gui`. Diese Rolle ist seit der Initial-Installation **PostgreSQL-SUPERUSER** — sie darf alles: `CREATE/DROP/ALTER TABLE`, `TRUNCATE`, `CREATE ROLE`, `ALTER SYSTEM`, `BypassRLS`, Replication. Der App-Layer braucht davon real nur Lesen/Schreiben einzelner Zeilen.

Das ist seit dem 2026-05-12 INCIDENT als *Worst Case-Vektor* klassifiziert: Eine Bot-Schwachstelle in `db_emitter.py` würde dem Angreifer komplette DB-Kontrolle geben.

1.2 Vorarbeit SEC-1c-3a (08:55 UTC vorhergehend)

Drei neue DB-Rollen wurden erzeugt:

Rolle	Privilegien	Zweck
`tradingbot_gui_app`	SELECT / INSERT / UPDATE / DELETE auf alle Tabellen + USAGE auf Sequences + TEMPORARY	Runtime-User für Bot/Worker/Filament-App
`tradingbot_gui_migrator`	ALL PRIVILEGES + USAGE/CREATE auf Schema `public`	Laravel-Migrationen / Schema-Änderungen
`tradingbot_gui`	SUPERUSER (unverändert)	DBA-Notfälle nur über SSH+psql — *nicht mehr* in App-Containern

`ALTER DEFAULT PRIVILEGES` wurde für sowohl den Migrator als auch den legacy SUPERUSER gesetzt — zukünftige Tabellen, egal wer sie anlegt, bekommen automatisch DML-Grants für den App-User. Damit ist die Rollen-Trennung wartungsfrei.

Laravel hatte den Cutover bereits in SEC-1c-3a hinter sich. Bot und Worker fehlten noch.

1.3 SEC-1c-3b Atomic Block (07:30–07:42 UTC, 11 Schritte)

Step 1 — Preflight Snapshot

Bot Host-PID 363, Worker Host-PID 1, `BINANCE_TESTNET=true`, `commands.id=13` cancelled, `managed_proposals = 0`, `managed_assets_history = 0`, Mainnet-5-Layer-Block aktiv. Master HEAD `d351a3a`.

Step 2 — Backup

Root-`.env` (`/projekte/Steve-TradingBot/.env`) gesichert nach `/root/sec-1c-3b-backup-20260514T071500Z/env.pre` mit mode 600.

Step 3 — .env-Update ohne Plaintext-Echo

Per Python-Subprocess wurden drei Schlüssel in die Root-`.env` geschrieben — der Passwort-Wert kam aus `/root/.secrets/sec-1c-3a-app.SECRET-READ-ONCE` als Subprocess-Environment-Variable, niemals als Argument oder Echo:

```
GUI_DB_DSN=postgresql://tradingbot_gui_app:<pw>@gui-db:5432/tradingbot_gui
GUI_DB_USER=tradingbot_gui_app
GUI_DB_PASSWORD=<pw>
```

Wichtig: das ist die **Root**-`.env` auf Host-Ebene, nicht `gui/.env`. Docker Compose substituiert beim Container-Start `${GUI_DB_*:-default}` aus diesem File in die env-Section von `clawbot` und `clawbot-worker`.

Step 4 — Compose-Sanity

`docker compose -p steve-tradingbot config --no-interpolate` bestätigte: env-Section verweist auf neue Variablen ohne Klartext-Substitution. Kein Secret-Leak im Output.

Step 5 + 6 — Recreate Bot und Worker

Hier kam das Volume-Drama (siehe §1.4). Final-Befehl, der funktioniert hat:

```
docker compose -p steve-tradingbot up -d --no-deps clawbot clawbot-worker
```

Restart-Window: knapp 30 Sekunden pro Container. Worker-Loop sprang an um **07:30:55 UTC**, Bot main.py wurde vom Watchdog um **07:38:05 UTC** als in-container PID 1196 hochgezogen.

Step 7 — Positive Permission Tests

```
SELECT current_user;          -- tradingbot_gui_app
```

```
INSERT INTO decision_logs ... -- 1 row created
SELECT count(*) FROM commands; -- read OK
```

Step 8 — Negative Permission Tests (alle 6 müssen blocken)

Versuch als `tradingbot_gui_app`	Erwartet	Ergebnis
`CREATE TABLE rbac_test (...)`	permission denied	OK
`DROP TABLE decision_logs`	permission denied	OK
`ALTER TABLE decision_logs ADD COLUMN ...`	permission denied	OK
`TRUNCATE decision_logs`	permission denied	OK
`CREATE ROLE attacker`	permission denied	OK
`ALTER SYSTEM SET ...`	permission denied	OK

Step 9 — Boundary-Recheck

cmd 13 weiterhin `cancelled`, `managed_proposals/history = 0/0`, `BINANCE_TESTNET=true`, Mainnet-Block ungestört, master HEAD `d351a3a` (kein Commit — der Cutover war rein host-side).

Step 10 + 11 — Operator-Aufgabe + Closure

Closure-Pin geschrieben. Operator-Aufgabe (Sektion 7 unten) bleibt offen.

1.4 Das Volume-Prefix-Drama

Während Step 5/6 entstand ein *Volume-Prefix-Mismatch*, der erst sichtbar wurde, als der Worker mit `ModuleNotFoundError: No module named 'trading.command_worker'` crash-loopte. Ursache: Die `docker-compose.yml` hat in Zeile 1 die Direktive `name: clawbot-docker`.

> **Verhalten von Docker Compose**: ohne expliziten `-p <name>` nimmt Compose den `name:`-Wert aus dem File. Die *originalen* Container waren aber mit Projekt-Namen `steve-tradingbot` gestartet (vermutlich über Umgebungsvariable `COMPOSE_PROJECT_NAME` oder frühere CLI-Aufrufe). Damit existierten parallel **zwei Sets benannter Volumes**:
>
> - `steve-tradingbot_*` — die echten, gefüllten, produktiv genutzten Volumes (mit aktuellem Code vom 2026-05-12).
> - `clawbot-docker_*` — leere/stale Volumes, die Compose unbemerkt anlegt, wenn ohne `-p` aufgerufen.

Der erste Recreate-Versuch ohne `-p` mountete die leeren `clawbot-docker_*`-Volumes. `command_worker.py` lag aber im `steve-tradingbot_clawbot-workspace`-Volume. Resultat: Worker-Crash-Loop.

Fix: Container stoppen + entfernen, dann mit explizitem Projekt-Prefix neu starten:

```
docker stop clawbot clawbot-worker
docker rm clawbot clawbot-worker
docker compose -p steve-tradingbot up -d --no-deps clawbot clawbot-worker
```

Durable Rule (gepinnt): Recreates für diesen Stack *immer* mit `-p steve-tradingbot`, sonst Volume-Mismatch.

1.5 Bot main.py Restart-Choreografie

Während der Container-Recreates lief der Host-Watchdog (`/projekte/Steve-TradingBot/bot_watchdog.sh`, Cron `*/5`) ganz normal weiter und versuchte mehrfach main.py zu spawnen. Da die `bot.pid`-Datei nach jedem Recreate aus dem alten Volume kam (PID 363 = tot), löste das mehrere Respawn-Zyklen aus. Das ist normales Watchdog-Verhalten, kein Fehler.

Final-Stand nach 07:38:05 UTC: main.py als in-container PID **1196**. Log zeigt sauberen Boot:

```
2026-05-14 07:38:05 | main | INFO | 🚀 Trading Bot startet...
2026-05-14 07:38:05 | main | INFO |    Modus: TESTNET LIVE
2026-05-14 07:38:08 | scanner.data_fetcher | INFO | 💰 Testnet-Balance: USDT free=9206.44 locked=0.00
2026-05-14 07:38:11 | main | INFO | 🔗 TESTNET-Connection: USDT-Saldo 9206.44  |  Live-Routing ENABLED
2026-05-14 07:39:48 | main | INFO | ✅ Scan-Zyklus abgeschlossen. 0 Signale generiert.
```

2. Beweise für „neue DSN funktioniert wirklich“

WORKER HEARTBEATS (ALLE 30S)
6+ rows in 3 Minuten

LOOP ITERATION
390 → 540 steigend

BOT SCAN-ZYKLUS
abgeschlossen, 0 Signale

DB-AUTH MANUAL TEST
tradingbot_gui_app ✓

Worker schreibt sauber in `bot_statuses` mit dem neuen DB-User. Auszug:

```
id   | recorded_at           | environment | bot_state    | metadata_json
-----+-----------------------+-------------+--------------+---------------------------------------
9822 | 2026-05-14 07:40:04+00 | testnet     | worker_alive | loop_iteration: 540, processed_count: 0
```

```
9821 | 2026-05-14 07:39:34+00 | testnet     | worker_alive | loop_iteration: 510
9820 | 2026-05-14 07:39:03+00 | testnet     | worker_alive | loop_iteration: 480
9819 | 2026-05-14 07:38:33+00 | testnet     | worker_alive | loop_iteration: 450
9818 | 2026-05-14 07:38:02+00 | testnet     | worker_alive | loop_iteration: 420
9817 | 2026-05-14 07:37:31+00 | testnet     | worker_alive | loop_iteration: 390
```

Worker-Process inside container schreibt seit 07:37:31 UTC ohne Auth-Fehler — wäre die DSN falsch, käme `psycopg2.errors.InvalidPassword` nach dem ersten Insert-Versuch und der Loop würde stehen. Stattdessen: monotone Loop-Iteration-Counter, exakt 30s Cadence, zero errors in Worker-Log.

3. Aktueller Container-Zustand

Container	Status	Image	Anmerkung
`clawbot`	**Up healthy**	steve-tradingbot-clawbot	Bot main.py PID 1196 in-container
`clawbot-worker`	**Up (unhealthy)**	steve-tradingbot-clawbot-runtime:latest	**False-Positive** — Healthcheck-Override fehlt (siehe §4.1)
`steve-tradingbot-gui`	**Up**	steve-tradingbot-gui	Filament, noch SUPERUSER → SEC-1c-3c
`steve-tradingbot-gui-db`	**Up healthy**	postgres:16-alpine	9 Tage uptime
`clawbot-searxng`	**Up 8d**	searxng/searxng:latest	Suchmaschinen-Proxy für Bot
`portainer`	**Up 33h**	portainer/portainer-ce:latest	Docker-UI auf 9443 (localhost-bound)
`openclaw-telegram`	**Up 10d**	openclaw-telegram:latest	Separates Projekt
`militaria-bot`	**Up 9d**	militaria-bot-bot	Anderes Projekt
`kw-website-caddy`	**Up 9d healthy**	caddy:2-alpine	Anderes Projekt
`dev-wordpress-1`	**Exited 33h**	wordpress:php7.4-apache	Cleanup-Kandidat (dev-Stack)
`dev-wpcli-1`	**Exited 33h**	wordpress:cli-php7.4	Cleanup-Kandidat
`dev-db-1`	**Exited 33h**	mysql:5.7	Cleanup-Kandidat

4. Was jetzt sinnvoll ist (priorisiert)

4.1 Worker-Healthcheck korrigieren **Quick-Win, ~5 min**

> **Problem:** `docker ps` zeigt `clawbot-worker` als `(unhealthy)`. Root-Cause: das `steve-tradingbot-clawbot-runtime:latest`-Image wurde per `docker commit` vom Bot-Container abgeleitet. Es hat darum den *Bot*-Healthcheck geerbt, der per `curl` auf `127.0.0.1:18791` (OpenClaw-Gateway) zugreift. Der Worker-Container fährt aber kein OpenClaw — Healthcheck schlägt darum permanent fehl.

Wirkung: Keine. Worker läuft stabil. Aber Monitoring sieht „unhealthy“ und kann maskieren, wenn der Worker später *wirklich* dead ist.

Fix-Optionen:

1. **Healthcheck im Compose-File überschreiben** (empfohlen) — im Service `clawbot-worker` in `docker-compose.yml`:

   ```
   healthcheck:
     test: ["CMD-SHELL", "pgrep -f trading.command_worker || exit 1"]
     interval: 30s
     timeout: 5s
     retries: 3
   ```

   Greift beim nächsten Recreate. Sauberster Fix.
2. **Healthcheck komplett deaktivieren** (Quick-and-Dirty):

   ```
   healthcheck:
     disable: true
   ```
3. **Eigenes Worker-Image bauen** statt `docker commit`-Snapshot — größerer Eingriff, gehört in MH-7-Hardening-Phase.

4.2 Verwaiste Compose-Volumes löschen **Safe, ~2.3 GB**

Direkter Output des Drama-Cleanups. **Keine** aktive Container-Referenz auf diese Volumes (verifiziert via `docker ps -a` + `docker inspect`):

Volume	Größe	Inhalt
`clawbot-docker_clawbot-workspace`	**1.2 GB**	Bot-Code-Snapshot vom 2026-05-03 (stale)
`clawbot-docker_clawbot-openclaw`	**873 MB**	OpenClaw-Snapshot stale
`clawbot-docker_gui-vendor`	139 MB	Composer-Vendor stale
`clawbot-docker_gui-postgres-data`	47 MB	Leere Postgres-Initdb
`clawbot-docker_clawbot-data`	4 KB	leer
`clawbot-docker_clawbot-memu`	4 KB	leer
`clawbot-docker_clawbot-workflows`	4 KB	leer

Cleanup-Befehl:

```
docker volume rm \
  clawbot-docker_clawbot-data \
  clawbot-docker_clawbot-memu \
  clawbot-docker_clawbot-openclaw \
  clawbot-docker_clawbot-workflows \
  clawbot-docker_clawbot-workspace \
  clawbot-docker_gui-postgres-data \
  clawbot-docker_gui-vendor
```

Sicherheits-Check vor dem Löschen (auch wenn das hier schon verifiziert wurde — Defensive Practice):

```
for v in clawbot-docker_*; do
  USED=$(docker ps -a --filter volume=$v --format '{{.Names}}')
  [ -z "$USED" ] && echo "$v: SAFE TO DELETE" || echo "$v: IN USE BY $USED"
done
```

Erwartung: *jedes* Volume = SAFE TO DELETE. Wenn nicht: **STOP**.

4.3 Zwei freistehende „clawbot-…"-Volumes ohne Prefix Safe, ~16 KB

Vermutlich Überbleibsel von einem *noch früheren* Docker-Stack vor der Projekt-Name-Konvention. Quasi leer:

Volume	Größe
`clawbot-openclaw`	12 KB
`clawbot-workspace`	4 KB

Cleanup-Befehl:

```
docker volume rm clawbot-openclaw clawbot-workspace
```

4.4 Dangling Images aufräumen Safe, ~17.7 GB

Aktuell 51 Images, davon 13 aktiv. Verbleibende 38 sind `<none>`-tagged (dangling) oder Image-Generationen aus früheren Builds. Konkret:

- **clawbot-docker-clawbot-worker** (1.48 GB, 0 containers) — Leiche des Worker-Builds aus dem falschen Projekt-Prefix.
- **clawbot-docker-gui** (1.06 GB, 0 containers) — analog.
- **clawbot-docker-clawbot** (1.26 GB, 0 containers) — analog.
- Ca. 30 weitere `<none>:<none>`-Images aus Layer-Promotion bei früheren Bot-Builds.

Cleanup-Befehl (zuerst die explizit benannten, dann dangling):

```
docker image rm clawbot-docker-clawbot-worker:latest \
                clawbot-docker-gui:latest \
                clawbot-docker-clawbot:latest

docker image prune -f
# erweitert: docker image prune -a -f  <-- nur wenn ALLE non-running Images weg sollen
```

> **Achtung bei `-a`:** löscht auch *getaggte* Images, die gerade keinen Container haben. Wenn du z.B. eine ältere Bot-Version griffbereit hältst (Rollback-Image), dann ohne `-a` bleiben.

4.5 Build-Cache Safe, ~44 GB

BuildKit-Cache von früheren Bot/GUI-Builds. Vollständig reclaimable. Wird beim nächsten `docker build` neu aufgebaut (kostet 5-10 Minuten beim ersten Build, danach wieder gecacht).

```
docker builder prune -f
# oder gezielter:  docker builder prune -af --filter "until=72h"
```

4.6 Exited Dev-Container Vorsicht — anderes Projekt

`dev-wpcli-1`, `dev-wordpress-1`, `dev-db-1` sind **nicht Teil von Steve-TradingBot** — sie gehören vermutlich zu einem WordPress-Dev-Stack. Beim `dev-db-1` hängen die `dev_db_data`- und `dev_wp_data`-Volumes mit Inhalt.

Empfehlung: *nicht* automatisch löschen. Erst klären: ist der WP-Dev-Stack noch in Benutzung? Falls ja, einfach so lassen (kostet ~0 GB). Falls nein:

```
docker rm dev-wpcli-1 dev-wordpress-1 dev-db-1
docker volume rm dev_db_data dev_wp_data   # NUR wenn Stack-Ende bestätigt
```

4.7 Anonyme Hex-Volumes Vorsicht — manuell prüfen

75 anonyme Volumes mit Hex-IDs. Davon ist ein Teil *aktiv* in Verwendung (z.B. von `militaria-mongo`, `openclaw-telegram`, `portainer`, Bot/Worker-Image-Builds). `docker volume prune` löscht *nur* die ungenutzten — aber bei Fremd-Stacks kann das gefährlich werden, wenn ein Container gerade stopped ist.

Empfehlung:

1. *Erst* Punkte 4.2–4.5 ausführen und Stack 24h beobachten.
2. *Dann* manuell pro Volume prüfen: `docker volume inspect <id> --format '{{.Mountpoint}}'` → reinkucken was drin liegt → bewusst entscheiden.

3. **Nie** `docker volume prune` blind ausführen, solange andere Projekte (militaria-bot, openclaw-telegram, KW-website) mit auf der Maschine laufen.

4.8 docker-compose.yml — Volume-Prefix-Fix dauerhaft Code-Hygiene

> **Root-cause des Volume-Dramas**: die Zeile `name: clawbot-docker` in `docker-compose.yml`. Sie wurde mal eingeführt, als das Repo unter dem Namen `clawbot-docker` gepullt war.

Drei Lösungswege:

1. **Beste Lösung** — `name: clawbot-docker` rauseditieren und überall `-p steve-tradingbot` nutzen. Compose nimmt dann das Directory-name als Default — also `steve-tradingbot`. Einmal-Migration: `docker compose -p steve-tradingbot up -d --force-recreate` nach dem Edit.
2. **Mittelweg** — Zeile zu `name: steve-tradingbot` ändern. Volumes bleiben unter `steve-tradingbot_*`, `-p` wird optional. Risiko: niemand weiß mehr, warum die Zeile da steht.
3. **Status quo** — Zeile lassen, durable Rule „immer mit `-p steve-tradingbot`" durchsetzen (gepinnt im Closure-Memo). Nachteil: jeder neue Operator stolpert wieder rein.

Empfehlung Variante 1, idealerweise im selben Wartungsfenster wie der Worker-Healthcheck-Fix (§4.1). Kombi-PR möglich.

4.9 SECRET-READ-ONCE-Files shreddern Operator-Pflicht

Die Passwörter aus `/root/.secrets/sec-1c-3a-*.SECRET-READ-ONCE` sind jetzt in der Root-`.env` hinterlegt und werden über Container-ENV gelesen. Die Files brauchen wir nicht mehr.

```
cat  /root/.secrets/sec-1c-3a-app.SECRET-READ-ONCE       # → Password-Manager
cat  /root/.secrets/sec-1c-3a-migrator.SECRET-READ-ONCE  # → Password-Manager (für SEC-1c-3d)
shred -u /root/.secrets/sec-1c-3a-app.SECRET-READ-ONCE
shred -u /root/.secrets/sec-1c-3a-migrator.SECRET-READ-ONCE
```

5. Zusammenfassende Empfehlung „in welcher Reihenfolge?"

#	Aktion	Risiko	Gewinn	Dauer
1	§4.9 — SECRET-READ-ONCE shreddern	—	Hygiene	1 min
2	§4.2 — clawbot-docker_* Volumes löschen	0 (verifiziert orphan)	2.3 GB	1 min
3	§4.3 — clawbot-openclaw / clawbot-workspace löschen	0	16 KB	30 s
4	§4.4 — clawbot-docker-* Images + dangling prune	niedrig	~17 GB	2 min
5	§4.5 — Build-Cache prunen	niedrig	~44 GB	1 min
6	§4.1 + §4.8 — Worker-Healthcheck + compose-name-Fix (kombiniert, ein Recreate)	kontrolliert (Restart-Window 30s)	sauberes Monitoring + keine Volume-Stolperer mehr	10 min
7	§4.6 — Dev-Container (nach Klärung)	n/a — Operator-Decision	klein	—
8	§4.7 — Anonyme Volumes (manuell, später)	hoch wenn blind	schwer schätzbar	30 min

Schritte 1–5 zusammen kosten ~5 Minuten und bringen ~63 GB Speicherplatz zurück, ohne den Bot/Worker anzufassen. Schritt 6 wäre der nächste logische SEC-Block — kann mit SEC-1c-3c (GUI-Cutover) zusammengelegt werden, weil beide ein Restart-Fenster brauchen.

6. Was ich *nicht* empfehle

- **Kein** `docker system prune -a --volumes`. Das ist Brute-Force, würde Volumes anderer aktiver Stacks (militaria-bot, openclaw-telegram) potentiell mit-killen und ist nicht reversibel.
- **Keine Container-Restarts während Trading-Aktivität**. Aktuell ist die Lage entspannt (TESTNET, 0 Signale, 0 open positions). Für die Worker-Healthcheck-Korrektur in §4.1 trotzdem ein bewusstes Wartungsfenster wählen.
- **Keine Änderung am** `tradingbot_gui`**-SUPERUSER jetzt**. Die Rolle bleibt für DBA-Notfälle. Die Lock-Down-Phase (SEC-1c-3d, mit `NOSUPERUSER`) kommt erst, wenn auch SEC-1c-3c (GUI-Cutover) stabil läuft.

7. Offene Operator-Aufgaben

1. SECRET-READ-ONCE-Files lesen → Password-Manager → `shred -u` (§4.9)
2. Cleanup-Schritte §4.2–4.5 (optional aber empfohlen)
3. Operator-Entscheidung: **SEC-1c-3c** (Filament-Container von `tradingbot_gui` auf `tradingbot_gui_app` umstellen — analog zu was Bot/Worker gerade durch hatten) **oder** MH-5a Read-only Filament-Resource parallel weiterführen
4. Mittelfristig: Worker-Healthcheck + compose-name-Edit als Kombi-PR (§4.1 + §4.8)

8. Boundaries-Konformität dieser Phase

Boundary	Status
0× Bot-Code-Touch	eingehalten
0× Worker-Code-Touch	eingehalten

0× `docker cp`	**eingehalten**
0× DB-Migration	**eingehalten**
0× Mainnet-Aktivität	**eingehalten** (BINANCE_TESTNET=true)
0× Push, 0× Commit	**eingehalten** (master HEAD `d351a3a`)
0× Secret-Wert im Output/Transcript/Memory	**eingehalten**
0× GUI-Container-Touch	**eingehalten** (separates SEC-1c-3c)
cmd 13 stabil cancelled	**eingehalten**
managed_proposals/history 0/0	**eingehalten**

Erstellt: 2026-05-14 (UTC) · Stack: `steve-tradingbot` · Phase-ID: SEC-1c-3b · Closure-Memo: `memory/sec_1c_3b_closure_20260514.md` · Vorgänger: SEC-1c-3a (Laravel-Cutover, geschlossen 08:55 UTC vorher)