SODU SECURE

WEEKLY SECURITY REVIEW

FMS64

2026-05-30 · Commit 9ef5df114b

Risiko-Lage: HOCH

HIGH 5 MEDIUM 2 LOW 4

11 offene Findings · 0 überfällig · 4 Quick Wins

Management Summary — FMS64

> 📣 **Diese Woche:** Gute Basis: **keine kritischen** Findings — 5 hohe(s) ist/sind noch offen. **11 neue** Finding(s) diese Woche.

Kennzahl	Wert
Offene Findings gesamt	**11**
Kritisch / Hoch	**0 / 5**
🆕 Neu diese Woche	11
✅ Diese Woche behoben	0
⚠️ Überfällig (SLA überschritten)	**0**
⚡ Quick Wins (geringer Aufwand)	4

Verlauf — offene Findings über die Wochen

Die Verlaufsgrafik erscheint ab dem zweiten wöchentlichen Lauf.

Schweregrad-Verteilung

Severity	Anzahl
HIGH	5
MEDIUM	2
LOW	4

🔴 Top-Risiken (kritisch / hoch)

- HIGH SQL injection via unvalidated object keys in car-earnings update (`src/backend/Models/carsEarnings.model.js`)
- HIGH Any authenticated user can delete any user account (cascading data destruction) (`src/backend/Controllers/user.controller.js`)
- HIGH Broken object-level authorization: cross-tenant modification and deletion of resources by ID (`src/backend/Controllers/driver.controller.js`)
- HIGH Broken object-level authorization: cross-tenant disclosure of financial and personal data (`src/backend/Controllers/archives.controllers.js`)
- HIGH Real Uber auth/session tokens and customer/driver PII committed to the git repository (gitignore ineffective) (`src/backend/carsAllPostRequests.json`)

⚡ Quick Wins — hohe Wirkung, geringer Aufwand

Diese Findings sind laut Einschätzung schnell behebbar — ideale erste Schritte:

- HIGH SQL injection via unvalidated object keys in car-earnings update (`src/backend/Models/carsEarnings.model.js`)
- HIGH Any authenticated user can delete any user account (cascading data destruction) (`src/backend/Controllers/user.controller.js`)
- MEDIUM Cross-tenant payout-rule bleed and tampering via unscoped driver_shares JOIN in earnings calculation (`src/backend/Models/earning.model.js`)
- MEDIUM Mass assignment of ownership (user_id) on driver-share update allows reassigning rule ownership across tenants (`src/backend/Controllers/driverShares.controller.js`)

Änderungen seit letztem Lauf (Code)

Erstlauf bzw. voller Scan — keine Vorwoche zum Vergleich. Alle offenen Findings gelten als neu.

> Wie Tony Stark in der Höhle: aus wenig wird ein stabiles System. 🔧 *(Marvel)*

Technische Details

- Gescannt am: 2026-05-30T09:10:58.143555+00:00
- Commit: `9ef5df114bc7f02f17e73aa0855a902ac71f7a2d`
- Dateien: 124 · Sprachen: JavaScript

🆕 New Findings

1. HIGH SQL injection via unvalidated object keys in car-earnings update ⚡ QUICK WIN

- **ID:** `d3d9a18e8ff396a8` *(ausblenden: `adler ignore <projekt> d3d9a18e8ff396a8`)*
- **Verifikation:** – nicht live getestet
- **Status:** new | **Quelle:** llm | **Kategorie:** injection | **Confidence:** 0.88 | **FP-Risiko:** low | **Fix-Aufwand:** low
- **CWE/OWASP:** CWE-89 / A03:2021 - Injection
- **Lebenszyklus:** 🆕 neu diese Woche
- **Wo:** `src/backend/Models/carsEarnings.model.js:179-198, src/backend/Controllers/carsEarnings.controller.js:110-123, src/backend/routes/carsEarnings.routes.js:42`
- **Funktionen:** updateCarEarning, updateCarEarningController
- **Impact:** An authenticated user can inject arbitrary SQL fragments via JSON keys, rewriting the SET/WHERE clause of the UPDATE. This permits modifying arbitrary columns of arbitrary rows across all tenants (e.g. zeroing or inflating every car's earnings), regardless of ownership. (Stacked queries are blocked because mysql2's multipleStatements defaults to false, but single-statement injection into this UPDATE is fully controllable.)
- **Erreichbarkeit:** PUT /cars-earnings/:id -> updateCarEarningController (routes/carsEarnings.routes.js:42) -> updateCarEarning(id, req.body). Reachable by any request holding a valid Firebase token for a user present in the users table (the only gate, applied globally in server.js:36).
- **Begründung:** updateCarEarningController passes the raw parsed JSON body straight into updateCarEarning as `fields` (controller line 113). updateCarEarning then iterates Object.entries(fields) and concatenates the attacker-controlled object KEY directly into the SQL string at line 184 (`updates.push( ${key} = ? )`) and line 193 (`UPDATE cars_earnings SET ${updates.join(', ')} WHERE id = ?`). Only the value is parameterized; the column-name/key is not validated against an allowlist and is interpolated verbatim. Every other update model in the codebase (updateDriver, updateEarning, updateArchive, updateFleet) uses a hard-coded column allowlist; this one does not.

Betroffener Code:

```
179: export async function updateCarEarning(id, fields) {
180:   const updates = [];
181:   const values = [];
182:
183:   for (const [key, value] of Object.entries(fields)) {
184:     updates.push(`${key} = ?`);
185:     values.push(value);
186:   }
...
192:   const [result] = await pool.query(
193:     `UPDATE cars_earnings SET ${updates.join(', ')} WHERE id = ?`,
194:     values
195:   );

// controller: carsEarnings.controller.js
112:     const { id } = req.params;
113:     const result = await updateCarEarning(id, req.body);
```

Proof of Concept / Verifikation:

```
curl -X PUT http://localhost:3000/cars-earnings/1 -H "Authorization: Bearer <validIdToken>" -H "Content-Type: application/json" -d '{"total_earnings = 0 WHERE 1=1 -- -": 0}'  =>  executed SQL becomes `UPDATE cars_earnings SET total_earnings = 0 WHERE 1=1 -- - = ? WHERE id = ?` which sets total_earnings=0 for EVERY row in cars_earnings (all tenants). Vary the injected key to target other columns/rows.
```

- **Remediation:** Do not derive column names from request keys. Map incoming fields through an explicit allowlist of permitted column names (as done in updateDriver/updateEarning), and reject or ignore unknown keys. e.g. `const`

`ALLOWED = {totalEarnings:'total_earnings', ...}; for (const [k,v] of Object.entries(fields)) if (ALLOWED[k]) { updates.push( ${ALLOWED[k]} = ? ); values.push(v); }`.
- **Zuerst gesehen:** 2026-05-30T09:10:58.143555+00:00

2. HIGH Any authenticated user can delete any user account (cascading data destruction)

⚡ QUICK WIN

- **ID:** `f307c1f5bd1c4ab3` *(ausblenden: `adler ignore <projekt> f307c1f5bd1c4ab3`)*
- **Verifikation:** – nicht live getestet
- **Status:** new | **Quelle:** llm | **Kategorie:** authorization | **Confidence:** 0.85 | **FP-Risiko:** low | **Fix-Aufwand:** low
- **CWE/OWASP:** CWE-862 / A01:2021 - Broken Access Control
- **Lebenszyklus:** 🆕 neu diese Woche
- **Wo:** `src/backend/Controllers/user.controller.js:49-58, src/backend/Models/user.model.js:24-31, src/backend/routes/user.routes.js:9`
- **Funktionen:** removeUser, deleteUserHandler, deleteUser
- **Impact:** A single low-privileged authenticated user can permanently delete any other user's account and, via ON DELETE CASCADE, all of that tenant's drivers, earnings, and financial archive records — full destruction of other tenants' data.
- **Erreichbarkeit:** DELETE /users/:id (routes/user.routes.js:9) -> removeUser -> deleteUserHandler -> deleteUser. Gated only by the global verifyFirebaseToken middleware (server.js:36); any valid account suffices.
- **Begründung:** removeUser takes the target user id straight from req.params and deletes it with no check that it equals the caller (req.user.localId) and no role/admin concept exists anywhere in the codebase (grep for role/admin/permission finds only firebase imports). Every other write in the app is meant to be per-user (the app exposes dedicated *OfUser endpoints scoped to req.user.localId), confirming users are independent tenants. drivers.sql/archives.sql/driver_shares.sql declare `ON DELETE CASCADE` on the user FK, so deleting a user also wipes that tenant's drivers, earnings and archives.

Betroffener Code:

```
49: export async function removeUser(req, res) {
50:   try {
51:     const { id } = req.params;
52:     const result = await deleteUserHandler(id);
53:     res.json(result);

// model user.model.js
24: export async function deleteUser(id){
25:     try{
26:         await pool.query('DELETE FROM users WHERE id=(?)', [id]);
```

Proof of Concept / Verifikation:

```
Authenticate as user A, then: curl -X DELETE http://localhost:3000/users/2 -H "Authorization: Bearer
<A_idToken>"  => returns {"message":"User deleted successfully"} and cascades, deleting user 2 and all their
drivers/earnings/archives.
```

- **Remediation:** Restrict deletion to self (compare req.params.id to req.user.localId) or introduce an admin role and enforce it. For self-service deletion, ignore the path param entirely and delete req.user.localId.
- **Zuerst gesehen:** 2026-05-30T09:10:58.143555+00:00

3. HIGH Broken object-level authorization: cross-tenant modification and deletion of resources by ID

- **ID:** `b0d8a5526c602f65` *(ausblenden: `adler ignore <projekt> b0d8a5526c602f65`)*
- **Verifikation:** – nicht live getestet
- **Status:** new | **Quelle:** llm | **Kategorie:** authorization | **Confidence:** 0.85 | **FP-Risiko:** low | **Fix-Aufwand:** medium
- **CWE/OWASP:** CWE-639 / A01:2021 - Broken Access Control
- **Lebenszyklus:** 🆕 neu diese Woche
- **Wo:** `src/backend/Controllers/driver.controller.js:84-113, src/backend/Controllers/earning.controller.js:151-196, src/backend/Controllers/archives.controllers.js:167-195, src/backend/Controllers/fleet.controller.js:58-83, src/backend/Controllers/car.controller.js:156-212, src/backend/Controllers/driverShares.controller.js:82-110`

- **Funktionen:** modifyDriver, removeDriver, modifyEarning, removeEarning, removeEarningsInRangeOfDriver, editArchive, removeArchive, updateFleetController, deleteFleetController, updateCarController, deleteCarController, editDriverShare, removeDriverShare, updateCarEarningController, deleteCarEarningController, addEarning, addArchive
- **Impact:** Any authenticated tenant can tamper with or delete other tenants' drivers, earnings, financial archives (payouts/tax/SV), fleets, cars, payout rules and car-earnings by guessing/iterating sequential integer IDs — data integrity loss, financial-record falsification, denial of service, and silent ownership takeover of cars.
- **Erreichbarkeit:** PUT/DELETE on /drivers/:id, /earnings/:id, /earnings/monthly/:id, /archives/:id, /fleets/:id, /cars/:id, /driver-shares/:id, /cars-earnings/:id (and POST /earnings, POST /archives accepting arbitrary driverId). Gated only by the global verifyFirebaseToken (server.js:36).
- **Begründung:** All of these handlers take a resource id from req.params and call an update/delete model that filters only by primary key (`WHERE id = ?`) with no `user_id` ownership constraint and no preceding ownership lookup. The models (e.g. updateDriver/deleteDriver in driver.model.js, updateFleet/deleteFleet in fleet.model.js, deleteArchive/updateArchive, deleteEarning/updateEarning) never scope by owner. The app clearly intends per-user isolation (parallel *OfUser endpoints exist that DO filter by req.user.localId, e.g. getAllDriversOfUser, getMonthlyEarningsOfDriversOfUser), so these unscoped variants are a real isolation gap. updateCarController is aggravated: it sets user_id = req.user.localId (car.controller.js:160,179), so editing another tenant's car transfers ownership of it to the attacker.

Betroffener Code:

```
// driver.controller.js modifyDriver / removeDriver
86:     const { id } = req.params;
87:     const updates = req.body;
89:     const result = await updateDriver({ id, ...updates });
...
107:     const { id } = req.params;
108:     await deleteDriver(id);

// fleet.controller.js updateFleetController / deleteFleetController
60:     const { id } = req.params;
63:     const result = await updateFleet(id, name, colour);
...
77:     const { id } = req.params;
78:     await deleteFleet(id);

// car.controller.js updateCarController (also reassigns ownership)
160:    const userId = req.user.localId;
173:    const result = await updateCar(
...
179:      userId ?? existingCar.userId
```

Proof of Concept / Verifikation:

```
As any logged-in user: curl -X DELETE http://localhost:3000/drivers/5 -H "Authorization: Bearer <token>" (deletes another tenant's driver + cascaded earnings/archives). Ownership takeover: curl -X PUT http://localhost:3000/cars/9 -H "Authorization: Bearer <token>" -H 'Content-Type: application/json' -d '{"name":"x"}' then GET /cars/ user shows car 9 now belongs to the attacker. Tamper: curl -X PUT http://localhost:3000/archives/3 ... -d '{"actualPayout":0,"isFinished":1}'.
```

- **Remediation:** Enforce ownership on every by-id mutation: either add `AND user_id = ?` (directly for user-owned tables; via JOIN drivers/cars for child tables earnings/archives/cars_earnings) using req.user.localId, or load the object first and 403 if its owner != req.user.localId. Centralize this in middleware/helpers so all resource routes are covered.
- **Zuerst gesehen:** 2026-05-30T09:10:58.143555+00:00

4. HIGH Broken object-level authorization: cross-tenant disclosure of financial and personal data

- **ID:** `3dd008b45e06a23e` *(ausblenden: `adler ignore <projekt> 3dd008b45e06a23e`)*
- **Verifikation:** – nicht live getestet
- **Status:** new | **Quelle:** llm | **Kategorie:** authorization | **Confidence:** 0.83 | **FP-Risiko:** low | **Fix-Aufwand:** medium
- **CWE/OWASP:** CWE-639 / A01:2021 - Broken Access Control
- **Lebenszyklus:** 🆕 neu diese Woche
- **Wo:** `src/backend/Controllers/archives.controllers.js:77-85, src/backend/Controllers/archives.controllers.js:127-141, src/backend/Models/archives.model.js:67-99, src/backend/Controllers/earning.controller.js:57-65, src/backend/Controllers/driver.controller.js:29-66, src/backend/Controllers/user.controller.js:21-29`

- **Funktionen:** fetchAllArchives, fetchArchiveById, fetchArchiveByDriverId, getPayoutHistoryController, listMonthlyArchivesOfDrivers, fetchAllEarnings, listMonthlyEarnings, fetchEarningsByDriver, fetchEarningsByDriverInGivenMonth, fetchAllDrivers, fetchDriverById, fetchDriverByUberId, fetchAllUsers, getFleetByIdController, getCarByIdController, getCarsByFleetIdController, getCarByUberIdController, getCarByPlateNumberController, fetchAllDriverShares, fetchDriverShareById
- **Impact:** Any authenticated tenant can read all other tenants' drivers, earnings, payout/financial archives, fleets, cars, payout rules and the full user list — mass cross-tenant data disclosure including financial and PII data, plus user/ID enumeration to feed the write/delete attacks.
- **Erreichbarkeit:** GET /archives, /archives/:id, /archives/:id/payout-history, /archives/monthly, /earnings, /earnings/monthly, /earnings/driver/:id, /drivers, /drivers/details/:id, /users, /fleets/:id, /cars/:id, /cars/fleet/:fleetId, /driver-shares, /driver-shares/:id. Gated only by global verifyFirebaseToken (server.js:36).
- **Begründung:** These read endpoints return data either with no owner filter at all (getAllArchives/getAllEarnings/getAllDrivers/getAllUser have no WHERE clause) or filtered only by the requested primary key (getArchiveById, getDriverById, getFleetById, getCarById, getDriverShareById, getEarningsByDriver). None constrain by req.user.localId. Archive rows expose sensitive financials (actual_payout, bank_payout, fuel_card/cash, sv_abgaben, tax, justification); driver rows expose uber_id, sv_abgaben, debts; users list exposes all account names/ids for enumeration. The presence of scoped *OfUser counterparts confirms per-tenant confidentiality is the intended model.

Betroffener Code:

```
// archives.controllers.js
77: export async function fetchAllArchives(req, res) {
79:     const archives = await getAllArchives();
80:     res.status(200).json(archives);
...
129:     const { id } = req.params;
130:     const archive = await getArchiveById(id);

// archives.model.js getAllArchives (no WHERE / owner filter)
67: export async function getAllArchives() {
69:     const [rows] = await pool.query(
70:       `SELECT ... FROM archives`
```

Proof of Concept / Verifikation:

```
As any logged-in user: curl http://localhost:3000/archives -H "Authorization: Bearer <token>" returns every tenant's financial archive rows; curl http://localhost:3000/archives/3 -H "Authorization: Bearer <token>" returns an arbitrary archive's payout/bank/tax detail; curl http://localhost:3000/users returns all user ids+names.
```

- **Remediation:** Scope every read to the caller: add `AND user_id = ?` (directly or via JOIN drivers/cars) using req.user.localId, or verify ownership after fetch and return 404/403 otherwise. Remove or restrict the unscoped list endpoints (getAllArchives/getAllEarnings/getAllDrivers/getAllUser) or gate them behind an admin role.
- **Zuerst gesehen:** 2026-05-30T09:10:58.143555+00:00

5. HIGH Real [redacted] auth/session tokens [redacted] committed to the git repository (gitignore ineffective)

- **ID:** `0f65aa6535fcf5cb` *(ausblenden: `adler ignore <projekt> 0f65aa6535fcf5cb`)*
- **Verifikation:** – nicht live getestet
- **Status:** new | **Quelle:** llm | **Kategorie:** secret | **Confidence:** 0.85 | **FP-Risiko:** low | **Fix-Aufwand:** medium
- **CWE/OWASP:** CWE-540 / A05:2021-Security Misconfiguration
- **Lebenszyklus:** 🆕 neu diese Woche
- **Wo:** [redacted]
  `cars/`
- **Funktionen:** runScraper, runScraperCars
- **Impact:** Anyone with read access to the repository (it has a GitHub remote and external-contributor PRs, e.g. merge from fmsystem64-png/stage) can clone it [redacted]. If any captured session_token/web-session token is still valid, it can be replayed against [redacted]
- **Erreichbarkeit:** git clone (or browse the repo) -> open src/backend/*.json. No application runtime involved; the sensitive data is in version control.

[redacted]

Betroffener Code:

Proof of Concept / Verifikation:

- **Remediation:**

- **Zuerst gesehen:** 2026-05-30T09:10:58.143555+00:00

6. MEDIUM Cross-tenant payout-rule bleed and tampering via unscoped driver_shares JOIN in earnings calculation ⚡ QUICK WIN

- **ID:** `28b6857325c232ad` *(ausblenden: `adler ignore <projekt> 28b6857325c232ad`)*
- **Verifikation:** – nicht live getestet
- **Status:** new | **Quelle:** llm | **Kategorie:** authorization | **Confidence:** 0.6 | **FP-Risiko:** medium | **Fix-Aufwand:** low
- **CWE/OWASP:** CWE-639 / A01:2021-Broken Access Control
- **Lebenszyklus:** 🆕 neu diese Woche
- **Wo:** `src/backend/Models/earning.model.js:109-123, src/backend/Controllers/earning.controller.js:128-137`
- **Funktionen:** getEarningsByDriverInGivenMonth, fetchEarningsByDriverInGivenMonth
- **Impact:** Any authenticated user can POST /driver-shares with a common shift_type (e.g. 'Vollzeit', 'Tag', 'Nacht') and an inflated percentage / fuel_limit. Because the settlement query takes MAX across all tenants, those attacker values are applied to OTHER tenants' drivers when their monthly settlement (Abrechnung) is computed, inflating the computed driver-share percentage and fuel limit and potentially causing the victim operator to over-pay drivers. Even without a malicious actor, any two tenants who happen to use the same shift_type label silently corrupt each other's payout figures.
- **Erreichbarkeit:** Authenticated GET /earnings/monthly/driver/:id?startDate=&endDate= (router earning.routes.js:39) -> fetchEarningsByDriverInGivenMonth -> getEarningsByDriverInGivenMonth. The driver_shares rows are populated via authenticated POST /driver-shares.
- **Begründung:** The payout computation joins the per-tenant `driver_shares` table only on `rule_type` and `shift_type` — there is NO `user_id` predicate tying the rule rows to the driver's owner. `COALESCE(MAX(ds.fuel_limit),0)` and `COALESCE(MAX(dm.percentage),0)` therefore aggregate fuel_limit/percentage across EVERY tenant's rules that share the same shift_type label. `driver_shares.user_id` exists and is used to scope ownership elsewhere (getDriverSharesOfUser at driverShares.model.js:69-89, createDriverShare sets user_id from req.user.localId), but it is ignored here. This is distinct from object-level read authorization on the driver row: it is an integrity flaw in the calculation logic itself that affects even a user's own authorized data.

Betroffener Code:

```
109:    COALESCE(MAX(ds.fuel_limit), 0) AS fuelLimit,
110:    COALESCE(MAX(dm.percentage), 0) AS driverSharePercentage
...
116:LEFT JOIN driver_shares ds
117:    ON ds.rule_type = 'fuel_rule'
118:   AND ds.shift_type = d.shift_type
120:LEFT JOIN driver_shares dm
121:    ON dm.rule_type = 'revenue_rule'
122:   AND dm.shift_type = d.shift_type
123:   AND es.nettoRevenue >= dm.revenue_threshold
```

Proof of Concept / Verifikation:

```
1) As attacker (any logged-in user) create a poisoned rule:
curl -X POST http://localhost:3000/driver-shares -H 'Authorization: Bearer <attackerToken>' -H 'Content-Type:
application/json' -d '{"shiftType":"Vollzeit","ruleType":"revenue_rule","percentage":99,"revenueThreshold":0}'
curl -X POST http://localhost:3000/driver-shares -H 'Authorization: Bearer <attackerToken>' -H 'Content-Type:
application/json' -d '{"shiftType":"Vollzeit","ruleType":"fuel_rule","fuelLimit":999999}'
2) Victim (different tenant) has a driver whose shift_type is 'Vollzeit'. Now GET /earnings/monthly/driver/
<victimDriverId>?startDate=2026-05-01&endDate=2026-05-31 returns driverSharePercentage=99 and fuelLimit=999999
(the cross-tenant MAX), corrupting the victim's settlement.
```

- **Remediation:** Scope the driver_shares joins to the driver's owner, e.g. add `AND ds.user_id = d.user_id` and `AND dm.user_id = d.user_id` to both joins (or pass the requesting user's id and filter on it). Apply the same owner-scoping to every driver_shares lookup used in calculations.
- **Zuerst gesehen:** 2026-05-30T09:10:58.143555+00:00

7. MEDIUM Mass assignment of ownership (user_id) on driver-share update allows reassigning rule ownership across tenants ⚡ QUICK WIN

- **ID:** `83287160689ff996` *(ausblenden: `adler ignore <projekt> 83287160689ff996`)*
- **Verifikation:** – nicht live getestet
- **Status:** new | **Quelle:** llm | **Kategorie:** authorization | **Confidence:** 0.68 | **FP-Risiko:** medium | **Fix-Aufwand:** low
- **CWE/OWASP:** CWE-915 / A01:2021-Broken Access Control
- **Lebenszyklus:** 🆕 neu diese Woche
- **Wo:** `src/backend/Controllers/driverShares.controller.js:84-87, src/backend/Models/driverShares.model.js:104-107`
- **Funktionen:** editDriverShare, updateDriverShare
- **Impact:** An authenticated user can change the owner (user_id) of any driver_share record: claim another tenant's payout rule by setting userId to their own id, or push a rule into a victim's account, polluting which rules the victim 'owns'. Because the rule set drives settlement configuration, this corrupts tenant data-ownership boundaries and combines with the existing missing-ownership-check to grant full takeover/transfer of rule records.
- **Erreichbarkeit:** Authenticated PUT /driver-shares/:id (driverShares.routes.js:22) -> editDriverShare -> updateDriverShare. No ownership check and no field allow-list between the body and the SQL.
- **Begründung:** createDriverShare always derives the owner from the trusted session (driverShares.controller.js:21 `const userId = req.user.localId`). The update path, however, spreads the raw request body into the model (`updateDriverShare({ id, ...updates })`) and updateDriverShare explicitly accepts a `userId` parameter and writes it to the privilege-relevant `user_id` column (driverShares.model.js:92-107). No allow-list filters the body, so a client-supplied `userId` overwrites the record's owner. This is the only update model in the codebase that accepts user_id from the spread body (updateDriver has user_id commented out; earnings/archives have no user_id column), so it is a unique mass-assignment weakness rather than a generic IDOR.

Betroffener Code:

```
controller editDriverShare:
84:    const { id } = req.params;
85:    const updates = req.body;
86:
87:    const result = await updateDriverShare({ id, ...updates });
model updateDriverShare:
104:  if (userId !== undefined) {
105:    fields.push("user_id = ?");
106:    values.push(userId);
107:  }
```

Proof of Concept / Verifikation:

```
curl -X PUT http://localhost:3000/driver-shares/5 -H 'Authorization: Bearer <attackerToken>' -H 'Content-Type:
application/json' -d '{"userId": 999}'
=> executes `UPDATE driver_shares SET user_id = ? WHERE id = ?` with [999, 5], reassigning rule #5 to user 999.
Set userId to the attacker's own localId to claim a victim's rule instead.
```

- **Remediation:** Do not pass user_id through from the request body. Remove `userId` from updateDriverShare's accepted fields (as already done for updateDriver), or build the update from an explicit allow-list (shiftType, revenueThreshold, percentage, fuelLimit, ruleType). Additionally enforce that the target row's user_id equals req.user.localId before updating.
- **Zuerst gesehen:** 2026-05-30T09:10:58.143555+00:00

8. LOW Database/internal error details returned to clients

- **ID:** `752ad6e885e0b0b4` *(ausblenden: `adler ignore <projekt> 752ad6e885e0b0b4`)*
- **Verifikation:** – nicht live getestet
- **Status:** new | **Quelle:** llm | **Kategorie:** logging | **Confidence:** 0.7 | **FP-Risiko:** low | **Fix-Aufwand:** low
- **CWE/OWASP:** CWE-209 / A05:2021 - Security Misconfiguration
- **Lebenszyklus:** 🆕 neu diese Woche
- **Wo:** `src/backend/Controllers/fleet.controller.js:23-25, src/backend/Controllers/car.controller.js:37-42, src/backend/Controllers/carsEarnings.controller.js:51-53`
- **Funktionen:** createFleetController, updateFleetController, deleteFleetController, getFleetByIdController, createCarController, updateCarController, createCarEarningController, updateCarEarningController
- **Impact:** Leaks internal/database error details (SQL state, driver/column messages, schema hints) that aid SQL-injection and schema mapping; information disclosure.
- **Erreichbarkeit:** Triggered by causing an error on the affected fleet/car/cars-earnings routes (e.g. malformed input or via the SQLi above), all behind the global auth middleware.
- **Begründung:** Several controllers serialize the caught error into the HTTP response. fleet.controller.js returns the whole `error` object; mysql2 error objects carry enumerable fields (code, errno, sqlState, sqlMessage) that get serialized, leaking DB error text and structure. car.controller.js and carsEarnings.controller.js return `error.message` (e.g. raw sqlMessage). This contrasts with the safe generic handler in server.js:53-56.

Betroffener Code:

```
// fleet.controller.js
23:   } catch (error) {
24:     res.status(500).json({ message: 'Error creating fleet', error });
25:   }

// car.controller.js
38:     res.status(500).json({
39:       message: 'Fehler beim Erstellen des Cars',
40:       error: error.message
41:     });
```

Proof of Concept / Verifikation:

```
Send a request that triggers a DB error to a fleet/car route (e.g. PUT /cars-earnings/1 with the injection
payload, or a body violating a constraint) and observe the JSON response includes `error`/`error.message`
containing the database message.
```

- **Remediation:** Return a generic message (as in server.js) and log details server-side only. Never serialize Error objects or error.message to clients in production.
- **Zuerst gesehen:** 2026-05-30T09:10:58.143555+00:00

9. LOW CORS enabled with wildcard origin for all API routes

- **ID:** `1268aeb5891ab0f6` *(ausblenden: `adler ignore <projekt> 1268aeb5891ab0f6`)*
- **Verifikation:** – nicht live getestet
- **Status:** new | **Quelle:** llm | **Kategorie:** misconfig | **Confidence:** 0.55 | **FP-Risiko:** medium | **Fix-Aufwand:** low
- **CWE/OWASP:** CWE-942 / A05:2021 - Security Misconfiguration
- **Lebenszyklus:** 🆕 neu diese Woche
- **Wo:** `src/backend/server.js:24`
- **Funktionen:** app

- **Impact:** Allows any website to interact with the API from the victim's browser context; primarily relevant if a token is ever exposed to page JS or auth is later moved to cookies. Mostly a hardening gap given current token-header auth.
- **Erreichbarkeit:** Global middleware applied to all routes (server.js:24).
- **Begründung:** cors() is used with no configuration, which sets Access-Control-Allow-Origin: * for every route. Any web origin can issue cross-origin requests to the API. Impact is limited here because authentication is a Bearer token in the Authorization header (not cookies), so the browser does not auto-attach credentials and a malicious origin cannot read another user's token without separate XSS/token theft; hence low severity rather than higher.

Betroffener Code:

```
24: app.use(cors());
```

Proof of Concept / Verifikation:

```
Confirm in code: cors() called without an origin allowlist. At runtime: any preflight/simple cross-origin request
receives `Access-Control-Allow-Origin: *`.
```

- **Remediation:** Configure an explicit origin allowlist, e.g. `app.use(cors({ origin: ['https://fms5561.de'], methods: ['GET','POST','PUT','DELETE'] }))`, and avoid wildcard origins in production.
- **Runtime-Validierung erforderlich:** ja
- **Zuerst gesehen:** 2026-05-30T09:10:58.143555+00:00

10. LOW

[redacted]

Betroffener Code:

[redacted]

Proof of Concept / Verifikation:

[redacted]

```
FROM users;` returns live session cookies in plaintext. Not directly exploitable without DB access or traffic
interception, so this is primarily a credential-handling/hardening weakness.
```

- **Remediation:** Encrypt the Uber cookie at rest (e.g. envelope encryption with a KMS-managed key) or store it in a dedicated secrets store; never return the Cookie field in user-profile API responses (exclude it from the getUserById projection used by fetchCurrentUser); scope its lifetime and rotate/invalidate on logout. Ensure DB at-rest encryption is enabled as defense-in-depth.
- **Zuerst gesehen:** 2026-05-30T09:10:58.143555+00:00

11. LOW Hardcoded, client-side-only PIN gate ("1001") protecting the Fahrer and Earning-Generator pages

- **ID:** `937ae9da43a7e4e1` *(ausblenden: `adler ignore <projekt> 937ae9da43a7e4e1`)*
- **Verifikation:** – nicht live getestet
- **Status:** new | **Quelle:** llm | **Kategorie:** authorization | **Confidence:** 0.9 | **FP-Risiko:** low | **Fix-Aufwand:** medium
- **CWE/OWASP:** CWE-602 / A01:2021 - Broken Access Control
- **Lebenszyklus:** 🆕 neu diese Woche
- **Wo:** `src/frontend/src/pages/Fahrer.vue:9-18, src/frontend/src/pages/Fahrer.vue:26-29, src/frontend/src/pages/EarningGenerator.vue:14-28`
- **Funktionen:** checkPin, validatePin
- **Impact:** Any user who can load the SPA (or any authenticated user who should not see the driver roster / bulk earning-generator) can bypass the gate instantly: the secret is readable in the JS bundle and the gate is a client-side boolean. Because the protected data and bulk-create operations are already reachable directly via the API with only a Firebase token, the PIN provides a false sense of an extra authorization tier while granting no real protection.
- **Erreichbarkeit:** Navigate to /drivers (Fahrer.vue) or /earning-generator (EarningGenerator.vue) — both registered in src/frontend/src/router/index.js and reachable by any logged-in user. The PIN overlay is the only thing between the user and the protected content, and it is enforced solely in the browser.
- **Begründung:** Both pages implement an access gate entirely in client-side Vue code. The expected PIN '1001' is hard-coded in the shipped JavaScript (Fahrer.vue:10 and EarningGenerator.vue:21). The 'authenticated' state is stored in a forgeable sessionStorage flag ('fahrer_pin_authenticated'='true', 'eg_pin_valid'='1') that is read on mount (Fahrer.vue:27, EarningGenerator.vue:15) to decide whether to render the protected content (DriversTable / the scraping & bulk earning-import UI). There is no corresponding server-side check: the backend routes these pages call (e.g. GET /drivers, POST /earnings, POST /cars, GET /uber-scrape/...) are protected only by the Firebase token middleware (server.js:36, middleware/auth.js) and never validate this PIN. The PIN therefore enforces nothing on the server.

Betroffener Code:

```
Fahrer.vue
9: const checkPin = () => {
10:   if (pinInput.value === '1001') {
11:     sessionStorage.setItem('fahrer_pin_authenticated', 'true');
12:     isAuthenticated.value = true;
...
26: onMounted(() => {
27:   const authenticated = sessionStorage.getItem('fahrer_pin_authenticated') === 'true';
28:   isAuthenticated.value = authenticated;

EarningGenerator.vue
15:   if (sessionStorage.getItem('eg_pin_valid') === '1') {
16:     pinValid.value = true
...
20: function validatePin() {
21:   if (pin.value === '1001') {
22:     pinValid.value = true
23:     try { sessionStorage.setItem('eg_pin_valid', '1') } catch (e) {}
```

Proof of Concept / Verifikation:

```
1) Read the PIN straight from source/bundle: it is the literal '1001'. Enter 1001 in the overlay. OR 2) Bypass
without the PIN — open DevTools console on /drivers and run:
sessionStorage.setItem('fahrer_pin_authenticated','true'); location.reload();  (for /earning-generator use
sessionStorage.setItem('eg_pin_valid','1'); location.reload();). OR 3) Ignore the UI entirely and call the
```

- **Remediation:** Do not implement authorization in the client. Remove the hard-coded PIN. If these pages must be restricted to a sub-set of users, introduce a real server-side role/permission (e.g. an 'is_admin'/role column checked in the auth middleware or per-route) and enforce it on every backend endpoint the pages use; the frontend should only hide UI based on the role returned by the server, never gate on a shipped secret or a sessionStorage flag.
- **Zuerst gesehen:** 2026-05-30T09:10:58.143555+00:00

♻️ Regressions

Keine.

🔁 Changed Findings

Keine.

📂 Existing Open Findings

Keine.

Coverage and Limitations

- LLM-Review: aktiv, 11 Findings (Claude Code, read-only).
- Findings, die Runtime-Validierung brauchen: 1.
- SLA-Schwellen (Wochen): critical=1, high=2, medium=4, low=8, info=12.
- Historie aktiv: Status basiert auf Vergleich mit dem persistenten Ledger.