

# Improper RLS Configuration

Admin Account Exposure via Unauthenticated Access

HIGH SEVERITY

DATE	TARGET
December 24, 2025	kafaatkg.netlify.app
BACKEND	PENETRATION TESTER
Supabase (PostgREST API)	Amr Saeed

## Executive Summary

During the security assessment of the target application, it was discovered that sensitive administrative account data is accessible without proper Row Level Security (RLS) enforcement. While other database tables enforce strict RLS policies to restrict data access, the `profiles` table exposes critical administrator information to unauthenticated users utilizing the `anon` role.

## Target & Methodology

### Target Specifications

- Application:** kafaatkg.netlify.app
- Backend:** Supabase (PostgREST API)
- Authentication:** Supabase Auth (JWT – anon role)

### Methodology

- Manual testing using Burp Suite.
- Direct interaction with the Supabase REST API.
- Analysis of OpenAPI specifications for endpoint enumeration.
- No authentication bypass or credential brute-force techniques were employed.

## Step-by-Step Discovery

### 1 Login Endpoint Analysis

The application utilizes Supabase Auth. A test request was sent to verify endpoint behavior:

```
POST /auth/v1/token?grant_type=password HTTP/2
Host: xwfgvpnkffdfojkwutij.supabase.co
Content-Type: application/json
Apikey: <anon_api_key>
Authorization: Bearer <anon_jwt>

{
  "email": "admin@test.com",
  "password": "123123"
}
```

**Result:** Invalid credentials were correctly rejected.

## 2 API Enumeration via OpenAPI

Accessing the OpenAPI specification revealed multiple exposed database tables:

```
GET /rest/v1/ HTTP/2
Host: xwfgvpnkffdfojkwutij.supabase.co
```

**Identified Tables:** profiles, students, attendance\_records, classes.

## 3 Accessing Admin Profile Without Authentication

Using the anon JWT token, a request was sent to retrieve all records from the profiles table:

```
GET /rest/v1/profiles?select=* HTTP/2
Host: xwfgvpnkffdfojkwutij.supabase.co
Apikey: <anon_api_key>
Authorization: Bearer <anon_jwt>
```

**Response:**

```
[
  {
    "id": "e598b6f6-ea23-43ea-8374-06ac35620134",
    "email": "***REDACTED***",
    "full_name": "كفاءات الأعمال",
    "role": "admin",
    "status": "active",
    "super_admin": true
  }
]
```

## 4 Comparison With Other Tables

Table	Query	Response	Status
students	GET /students?select=*	[]	Secured
attendance_records	GET /attendance_records?select=*	[]	Secured
profiles	GET /profiles?select=*	[{...admin_data...}]	Vulnerable

These tables correctly enforce RLS policies, returning empty arrays for unauthenticated requests.

## Root Cause Analysis

The profiles table does not properly enforce Row Level Security. The administrative account is accessible to unauthenticated users and those possessing the anon role. This indicates an inconsistent and incomplete RLS configuration across the database schema.

## Impact

- Exposure of Admin Role:** Critical PII is publicly accessible.
- Targeted Phishing:** Attackers can craft specific social engineering attacks.
- Account Takeover:** Facilitates brute-force or reset-token attacks.
- System Compromise:** May lead to full system compromise if chained with other vulnerabilities.

## Recommended Fix

- Enable RLS on the profiles table:** Ensure RLS is turned ON.
- Apply Strict Policies:** Users should only view their own profile (`auth.uid() = id`).
- Prevent Anon Access:** Explicitly deny the anon role from accessing sensitive data.
- Audit:** Review all tables for consistent RLS enforcement.

---

**Author:** Amr Saeed | **GitHub:** [RYD3R-0](#)

Confidential Report | Generated for Educational Purposes