API Security Assessment Report

Target Environment: `https://dev.[redacted].app/`
Assessment Date: August 2025
Assessment Type: Blackbox and Logical Review
Reviewer: BOLUJO EXCELLENT

ImmuniWeb Result Grade - F,[redacted]

Executive Summary

This report outlines critical vulnerabilities identified in the current implementation of [redacted]s API endpoints. The encryption mechanisms, authentication flaws, and insecure endpoint design expose the system to risks such as unauthorized data access, user impersonation, session hijacking, and mass service disruption. These vulnerabilities could be exploited by attackers to compromise confidentiality, integrity, and availability of the system.

Key Findings

1. Deterministic Encryption in Use

Observation: The encryption service at `https://dev.[redacted].app/decrypt/encrypt` produces the same ciphertext for the same input.
Example:

Input: {"id": "474"}
Output: i855963Mb/T2FMOq788mJOLr7rGxNMfpTgDXdOX+quuXTWqXNMDLcqZ7OoT4k86k

- **Risk:** This strongly indicates the use of **deterministic encryption or encoding**, likely AES in ECB mode or a non-randomized scheme.
- **Implication:** Patterns in input are preserved in ciphertext, making the system vulnerable to:
    - Frequency analysis
    - Plaintext-ciphertext mapping
    - Replay or forgery attacks

- **Severity:** HIGH

2. Encryption Endpoint Lacks Authentication

- **Endpoints:**
    - `POST https://dev.[redacted].app/decrypt/encrypt`
    - `POST https://dev.[redacted].app/decrypt`
- **Observation:** These endpoints appear to be publicly accessible without requiring any form of authentication or access control.
- **Risk:** An attacker with knowledge of input patterns (e.g., user IDs) can abuse this to:
    - Encrypt arbitrary inputs and spoof tokens or payloads
    - Decrypt sensitive data if responses are not strictly limited
- **Severity:** CRITICAL

3. Insecure Logout Endpoint (Unauthenticated)

- **Endpoint:** `https://dev.[redacted].app/passengerapi114/index/?type=passenger_logout`
- **Payload Example:** `{ "id": "474" }`
- **Observation:** This endpoint:
    - Accepts user ID directly in the body
    - Requires **no token or session** validation
    - Allows **logout of any user** with just the ID
- **Risk:**
    - **Authorization Bypass:** No user ownership validation

    - **ID Enumeration Attack:** Numeric IDs (e.g., 1–10000) can be brute-forced
    - **Denial of Service (DoS):** Scripted attacks can force-logout users en masse
- **Real-World Scenario:** Malicious script logs out 1000+ users via ID enumeration
- **Severity:** CRITICAL

4. Lack of Access Control Across API

- **Observation:**
    - API requests do not enforce identity or session binding
    - Backend trusts client-supplied IDs or payloads
- **Implication:**
    - One user can impersonate another
    - No protection against privilege escalation or impersonation
- **Severity:** HIGH

5. Unsecured API Design Patterns

- **Observation:** Current API routes use:
    - Query parameters for logic control (`?type=passenger_logout`)
    - Redundant and insecure method (`GET/POST mix for unsafe operations`)
- **Risk:** This violates RESTful best practices and makes code behavior harder to audit or secure
- **Severity:** MEDIUM

6. Trip tracking - Screen for when trip not found

Mitigation Recommendations

1. Use Strong, Randomized Encryption

- Replace deterministic encryption with **AES-GCM or AES-CBC with random IVs**
- Never use ECB mode
- Ensure encryption is only used for internal system purposes, not as authentication or authorization

2. Protect All Sensitive Endpoints with Authentication

- Require **OAuth2 / JWT** for all encryption/decryption, logout, and sensitive operations
- Validate token for every request:
    - Token signature
    - Expiry
    - Scope/permissions

3. Apply Proper Authorization Checks

- Do not accept user IDs or sensitive info in body or query without verification
- Validate that the token owner is the user performing the action:
    - Match token subject to user ID in request
    - Reject mismatches or unverified sessions

4. Refactor API Endpoints

- Use RESTful path: `POST /v1/passenger/logout`
- Remove need for passing user ID — derive from the token
- Implement session invalidation securely

5. Implement Rate Limiting and Abuse Prevention

- Rate-limit API usage per IP or user token
- Monitor for brute-force patterns
- Add CAPTCHA or bot protection to sensitive or abuse-prone endpoints

6. Enable Logging and Audit Trails

- Log every sensitive operation with user, IP, timestamp, and action
- Enable traceability for security investigations and accountability

7. Secure All API Traffic

- Enforce HTTPS with TLS 1.2+
- Validate that encryption endpoints are internal or require API keys at minimum

Potential Impact of No Action

Risk	Possible Impact
Unauthorized Access	Attackers perform actions on behalf of others
Denial of Service	Users are mass logged out or service degraded
Data Exposure	Confidential payloads guessed or decrypted
Lack of Accountability	No tracking of malicious usage or breach analysis
Compliance Failures	Violations of GDPR, PCI-DSS, or internal standards

Summary of Key Remediations

Area	Action
Encryption	Switch to AES-GCM or CBC with random IVs
Authentication	Require tokens for *all* sensitive endpoints

Authorization	Validate user identity for all actions
Endpoint Design	Refactor to use RESTful routes, avoid exposing user IDs
Security Controls	Add rate limits, audit logging, bot protection

Final Notes

The current architecture lacks the foundational controls necessary to ensure secure handling of user data and operations. Immediate attention is required to prevent exploitation. The outlined mitigation steps provide a clear roadmap toward securing the system and protecting both the platform and its users.