MoMo SMS Analytics - REST API Security Report

Team: Codeizzy

Date: February 2, 2026

Members: David Achibiri, Manuelle Ackun, Rhoda Umutesi

Table of Contents

---

1. Introduction to API Security

API security protects Application Programming Interfaces from unauthorized access and malicious attacks. For our MoMo SMS Analytics system, securing the API is critical because it exposes sensitive financial transaction data including user information, transaction amounts, and payment history.Key Security Principles

- **Authentication:** Verifying user identity before granting access. Our API uses Basic Authentication requiring valid credentials with each request.
- **Authorization:** Controlling what authenticated users can do. Currently all authenticated users have full CRUD access to transactions.
- **Confidentiality:** Protecting data from unauthorized disclosure. Best achieved through HTTPS encryption (not implemented in our development version).
- **Data Integrity:** Ensuring data isn't tampered with. Implemented through input validation and proper error handling.

Common API Security Threats

- **Broken Authentication:** Weak passwords, credential theft
- **Injection Attacks:** SQL injection, malicious input
- **Excessive Data Exposure:** Returning more data than necessary
- **Man-in-the-Middle Attacks:** Intercepting unencrypted communications
- **Lack of Rate Limiting:** API abuse and denial of service

Our Security Implementation

Security Measure	Status
Basic Authentication with credential verification	Implemented

401 Unauthorized responses for invalid credentials	Implemented
Input validation and JSON parsing	Implemented
Proper HTTP status codes	Implemented

2. API Endpoints Documentation

Base URL: `http://localhost:8000`

Authentication: Basic Auth (Username: `admin`, Password: `momosmsanalysis`)GET /transactions

Description: List all SMS transactions

Request:

```
GET /transactions HTTP/1.1
Host: localhost:8000
Authorization: Basic YWRtaW46bW9tb3Ntc2FuYWx5c2lz
```

Response (200 OK):

```
[
  {
    "id": 1,
    "transaction_type": "RECEIVED",
    "amount": 50.00,
    "sender": "0201234567",
    "receiver": "0241234567",
    "timestamp": "2024-01-15T10:30:00",
    "reference": "TXN20240115001"
  }
]
```

Error Codes: 401 (Unauthorized)

GET /transactions/{id}

Description: Retrieve a single transaction by ID

Request:

```
GET /transactions/1 HTTP/1.1
Host: localhost:8000
Authorization: Basic YWRtaW46bW9tb3Ntc2FuYWx5c2lz
```

Response (200 OK):

```
{
  "id": 1,
  "transaction_type": "RECEIVED",
  "amount": 50.00,
  "sender": "0201234567",
  "receiver": "0241234567"
}
```

Error Codes: 401 (Unauthorized), 404 (Not Found), 400 (Bad Request)

POST /transactions

Description: Create a new transaction

Request:

```
POST /transactions HTTP/1.1
Host: localhost:8000
Authorization: Basic YWRtaW46bW9tb3Ntc2FuYWx5c2lz
Content-Type: application/json

{
  "transaction_type": "SENT",
  "amount": 75.00,
  "sender": "0241234567",
  "receiver": "0554567890"
}
```

Response (201 Created):

```
{
  "id": 3,
  "transaction_type": "SENT",
  "amount": 75.00,
  "sender": "0241234567",
  "receiver": "0554567890"
}
```

Error Codes: 401 (Unauthorized), 400 (Bad Request)

PUT /transactions/{id}

Description: Update an existing transaction

Request:

```
PUT /transactions/1 HTTP/1.1
Host: localhost:8000
Authorization: Basic YWRtaW46bW9tb3Ntc2FuYWx5c2lz Content-Type:
application/json
```

```
{
  "amount": 60.00
}
```

Response (200 OK):

```
{
  "id": 1,
  "transaction_type": "RECEIVED",
  "amount": 60.00,
  "sender": "0201234567",
  "receiver": "0241234567"
}
```

Error Codes: 401 (Unauthorized), 404 (Not Found), 400 (Bad Request)

DELETE /transactions/{id}

Description: Delete a transaction

Request:

```
DELETE /transactions/1 HTTP/1.1
Host: localhost:8000
Authorization: Basic YWRtaW46bW9tb3Ntc2FuYWx5c2lz
```

Response (200 OK):

```
{
  "message": "Transaction deleted"
}
```

Error Codes: 401 (Unauthorized), 404 (Not Found)

---

3. DSA Comparison ResultsImplementation Details

We implemented and compared two search algorithms for retrieving transactions by ID:

- **Linear Search (O(n)):** Sequentially scans through the list comparing each transaction ID until found.
- **Dictionary Lookup (O(1)):** Stores transactions in a hash table (Python dictionary) where transaction ID is the key, enabling direct access.

Benchmark Results

Test Parameters:

- Dataset: 1,691 transactions from modified_sms_v2.xml
- Number of searches: 100 random lookups
- Environment: Python 3.x

Metric	Linear Search	Dictionary Lookup	Improvement
Total Time	6.35 ms	0.041 ms	154.9x faster
Avg per Search	0.0635 ms	0.000409 ms	155.25 x faster
Avg Comparisons	~845	1	845x fewer
Time Complexity	O(n)	O(1)	Constant time

Dictionary Creation: ~0.5 ms one-time costWhy Dictionary Lookup is Faster

Linear Search Process:

```
Search for ID 500:
→ Check transaction[0]
→ Check transaction[1]
→ ...
→ Check transaction[499] ✓
Result: 500 comparisons
```

Dictionary Lookup Process:

```
Search for ID 500:
→ Compute hash(500)
→ Access dict[500] directly ✓
Result: 1 operation
```

Dictionary lookup uses a hash function to compute the memory address directly, eliminating the need to iterate through elements. This provides constant O(1) time regardless of dataset size.Scalability Analysis

Dataset Size	Linear Search	Dictionary Lookup
100 records	~50 comparisons	1 comparison
1,691 records	~845 comparisons	1 comparison
10,000 records	~5,000 comparisons	1 comparison

Conclusion: Linear search performance degrades proportionally with dataset size, while dictionary lookup remains constant.Alternative Data Structures

- **Binary Search Tree (BST) - O(log n):**
- For 1,691 records: ~11 comparisons
- Good for sorted data and range queries

- More complex than dictionary ● **Trie (Prefix Tree) - O(k):**
- Excellent for prefix searches
- Useful for autocomplete features
- High memory overhead

Recommendation: Dictionary lookup is optimal for our API's `GET /transactions/{id}` endpoint, providing the best balance of speed (O(1)), simplicity, and low memory overhead.

---

4. Basic Authentication LimitationsWhat is Basic Authentication?

Basic Authentication encodes credentials (username:password) in base64 and sends them in the Authorization header:

```
"admin:momosmsanalysis" → Base64 encode →
"YWRtaW46bW9tb3Ntc2FuYWx5c2lz"
Header: "Authorization: Basic YWRtaW46bW9tb3Ntc2FuYWx5c2lz"
```

Critical Security Flaws1. Not Encrypted

Base64 is **encoding**, not **encryption**. Anyone can decode it:

```
Python
base64.b64decode("YWRtaW46bW9tb3Ntc2FuYWx5c2lz"
)
# Returns: "admin:momosmsanalysis"
```

Risk: Credentials transmitted in plain text. Vulnerable to Man-in-the-Middle attacks, especially on public WiFi.2. Credentials Sent with Every Request

Unlike token-based systems, credentials must be included in **every single request**, increasing exposure risk with each API call.3. No Expiration

Credentials remain valid indefinitely until manually changed. No automatic expiration means compromised credentials can be exploited indefinitely.4. Vulnerable to Brute Force

No built-in protection against automated password guessing attacks. Attackers can try unlimited password combinations.5. Poor Scalability

- Single shared password for all users in our implementation
- Cannot distinguish between users
- No role-based access control
- Difficult to audit who performed which actions

Stronger AlternativesJWT (JSON Web Tokens) - Recommended

How it works:

1. User logs in with credentials once
2. Server returns a signed token with expiration
3. Client includes token in subsequent requests
4. Token expires automatically (e.g., 1 hour)

Advantages over Basic Auth:

- Credentials only sent during login
- Automatic expiration (configurable)
- User identification embedded in token
- Supports role-based permissions
- Cannot be tampered with (cryptographic signature)
- Revocable via token blacklist

Example Token Structure:

```
Header.Payload.Signature
 Payload
contains:
{
  "user_id": 1,
  "username": "david",
  "role": "admin",
  "exp": 1707234000  // Expiration timestamp
}
```

Feature	Basic Auth	JWT	OAuth 2.0
Security	Weak	Strong	Very Strong
Expiration	No	Yes	Yes
Revocation	Difficult	Easy	Easy
User Identity	No	Yes	Yes
Scalability	Poor	Good	Excellent
Complexity	Simple	Moderate	Complex
