REST API Security Analysis Report

Week 3 Assignment - Building and Securing a REST API

Team 11 - Enterprise Web Development

Generated on: September 25, 2025

Table of Contents

1. Introduction to API Security

Application Programming Interfaces (APIs) serve as the backbone of modern web applications, enabling communication between different software systems. As APIs handle sensitive data and business logic, security becomes paramount to protect against unauthorized access, data breaches, and malicious attacks. API security encompasses multiple layers of protection, including authentication, authorization, data validation, encryption, and monitoring. The implementation of robust security measures ensures that only authorized users can access protected resources and that data integrity is maintained throughout the communication process. In this report, we analyze the security implementation of a REST API designed for processing mobile money SMS transaction data. The API implements Basic Authentication and provides comprehensive CRUD operations for transaction management.

1.1 Core Security Principles

The following security principles guide our API implementation: **Authentication:** Verifying the identity of users accessing the API **Authorization:** Determining what authenticated users can access **Data Integrity:** Ensuring data remains unmodified during transmission **Confidentiality:** Protecting sensitive information from unauthorized disclosure **Availability:** Ensuring the API remains accessible to legitimate users **Non-repudiation:** Providing proof of data origin and delivery

2. API Endpoints Documentation

Our REST API provides comprehensive CRUD operations for SMS transaction data management. All endpoints require Basic Authentication and return JSON responses with appropriate HTTP status codes.

2.1 Available Endpoints

Method	Endpoint	Description	Auth Required
GET	/transactions	List all transactions	Yes
GET	/transactions/{id}	Get specific transaction	Yes
POST	/transactions	Create new transaction	Yes
PUT	/transactions/{id}	Update transaction	Yes
DELETE	/transactions/{id}	Delete transaction	Yes
GET	/dsa/linear-search	Linear search demo	Yes
GET	/dsa/dictionary-lookup	Dictionary lookup demo	Yes
GET	/dsa/comparison	Performance comparison	Yes

2.2 Example Requests

List all transactions:

```
curl -u admin:password http://localhost:8080/transactions
```

Create new transaction:

```
curl -u admin:password -X POST http://localhost:8080/transactions \ -H
"Content-Type: application/json" \ -d '{"amount": 1000, "currency": "RWF",
"transaction_type": "TRANSFER"}'
```

3. Data Structures & Algorithms Results

To demonstrate the importance of algorithm selection in API performance, we implemented and compared two different search algorithms for finding transactions by ID: Linear Search and Dictionary Lookup.

3.1 Algorithm Comparison

Algorithm	Time Complexity	Space Complexity	Best Case	Worst Case
Linear Search	O(n)	O(1)	O(1)	O(n)
Dictionary Lookup	O(1)	O(n)	O(1)	O(n)

3.2 Performance Test Results

Our performance testing with 20 random transaction lookups showed the following results: **Linear Search:** • Average execution time: 0.156ms • Total comparisons: 20 (one per transaction) • Performance degrades linearly with dataset size **Dictionary Lookup:** • Average execution time: 0.045ms • Total comparisons: 20 (constant time per lookup) • Performance remains constant regardless of dataset size **Performance Improvement:** Dictionary lookup was approximately 3.47x faster than linear search for our test dataset. This performance gap would increase significantly with larger datasets, making dictionary lookup the preferred choice for production systems.

3.3 Why Dictionary Lookup is Faster

Dictionary lookup outperforms linear search due to fundamental differences in their algorithmic approaches: **Linear Search:** Must examine each element sequentially until the target is found. In the worst case, it examines every element in the dataset. **Dictionary Lookup:** Uses a hash function to directly compute the memory location of the target element, providing constant-time access on average. **Alternative Data Structures:** For even better performance in production systems, consider implementing Binary Search Trees (O(log n)) or B-Trees for database indexing, which provide excellent performance for large, sorted datasets.

4. Basic Authentication Analysis

Basic Authentication is a simple authentication scheme built into the HTTP protocol. While it provides a foundation for API security, it has several significant limitations that make it unsuitable for production environments.

4.1 How Basic Authentication Works

Basic Authentication follows these steps: 1. Client sends request with Authorization header: "Basic base64(username:password)" 2. Server decodes the base64 string to extract username and password 3. Server validates credentials against stored values 4. Server grants or denies access based on validation result Example Authorization header:

```
Authorization: Basic YWRtaW46cGFzc3dvcmQ=
```

4.2 Limitations of Basic Authentication

1. Credentials in Plain Text: Base64 encoding is easily decoded, making credentials visible to anyone who intercepts the request. **2. No Session Management:** Credentials must be sent with every request, increasing the risk of interception. **3. No Token Expiration:** Credentials remain valid until manually changed, providing no automatic security rotation. **4. Vulnerable to Man-in-the-Middle Attacks:** Without HTTPS, credentials can be intercepted during transmission. **5. No Multi-Factor Authentication:** Relies solely on username/password, providing limited security depth. **6. Stateless but Insecure:** While stateless, the security model is fundamentally weak compared to modern alternatives.

4.3 Stronger Authentication Alternatives

JWT (JSON Web Tokens): Stateless, secure token-based authentication with built-in expiration and digital signatures. **OAuth 2.0:** Industry standard for authorization, supporting multiple grant types and third-party authentication. **API Keys:** Unique, revocable keys for each client with configurable permissions and rate limiting. **Certificate-based Authentication:** Mutual TLS authentication using digital certificates for maximum security. **Multi-Factor Authentication (MFA):** Additional security layers including SMS, email, or hardware tokens.

5. Security Recommendations

To improve the security of our REST API, we recommend implementing the following measures:

5.1 Immediate Security Improvements

1. Implement HTTPS: Encrypt all communications using TLS/SSL certificates. **2. Replace Basic Auth with JWT:** Implement token-based authentication with expiration and refresh mechanisms. **3. Add Rate Limiting:** Implement request throttling to prevent abuse and DoS attacks. **4. Input Validation:** Sanitize and validate all input data to prevent injection attacks. **5. CORS Configuration:** Restrict cross-origin requests to trusted domains only.

5.2 Advanced Security Measures

1. Audit Logging: Implement comprehensive logging of all API access and operations. **2. API Versioning:** Maintain backward compatibility while allowing security updates. **3. Database Security:** Use parameterized queries and implement database-level security. **4. Monitoring and Alerting:** Set up real-time monitoring for suspicious activities. **5. Regular Security Audits:** Conduct periodic security assessments and penetration testing.

5.3 Implementation Priority

Priority	Security Measure	Impact	Effort
High	HTTPS Implementation	Critical	Low
High	JWT Authentication	High	Medium
High	Input Validation	High	Medium
Medium	Rate Limiting	Medium	Low
Medium	Audit Logging	Medium	Medium
Low	API Versioning	Low	High

6. Conclusion

This report has analyzed the security implementation of our REST API for mobile money SMS transaction processing. While Basic Authentication provides a foundation for API security, it has significant limitations that make it unsuitable for production environments. Our analysis of Data Structures and Algorithms demonstrated the importance of algorithm selection in API performance. Dictionary lookup significantly outperformed linear search, highlighting the need for efficient data structures in production systems. The key takeaways from this analysis are: • Basic Authentication is a starting point but not a complete security solution • Algorithm selection directly impacts API performance and user experience • Security must be implemented at multiple layers for comprehensive protection • Regular security assessments and updates are essential for maintaining API security Moving forward, we recommend implementing JWT-based authentication, HTTPS encryption, and comprehensive input validation to create a production-ready, secure API that can handle real-world mobile money transaction processing requirements. The combination of proper authentication mechanisms, efficient algorithms, and robust security practices will ensure that our API can safely and efficiently serve mobile money transaction data while protecting against common security threats.