API Security and Project Analysis Report

Table of Contents

Introduction to API Security

API security is the practice of protecting Application Programming Interfaces (APIs) from attacks. As APIs are the backbone of modern applications, securing them is critical. Key concepts in API security include:

- **Authentication:** Verifying the identity of a user or application. Common methods include API keys, OAuth 2.0, and JWT.
- **Authorization:** Determining what an authenticated user is allowed to do.
- **Data Encryption:** Protecting data in transit (using TLS/HTTPS) and at rest.
- **Rate Limiting:** Preventing abuse by limiting the number of requests an entity can make.
- **Input Validation:** Preventing injection attacks by validating all incoming data.

Common vulnerabilities include broken authentication, broken object-level authorization (BOLA), excessive data exposure, and security misconfigurations.

API Endpoint Documentation

MoMo SMS Transactions API Documentation

Authentication

This API uses HTTP Basic Authentication. The credentials are `admin:password123`.

The `Authorization` header must be included in all requests: `Authorization: Basic YWRtaW46cGFzc3dvcmQxMjM=`

Endpoints

Get All Transactions

- **Description:** Retrieves a list of all transactions.
- **Method:** `GET`
- **Path:** `/transactions`
- **Request:** `http GET /transactions HTTP/1.1 Host: localhost:8000 Authorization: Basic YWRtaW46cGFzc3dvcmQxMjM=`
- **Success Response:**
  - **Code:** `200 OK`
  - **Body:** `json [{ "Id": "TxID1", "transaction_type": "deposit", "amount": 2000, "sender": "Jane Smith (*********013) on your mobile money account", "receiver": "My Account", "timestamp": "2024-05-10 16:30:51", "raw_body": "You have received 2000 RWF from Jane Smith (*********013) on your mobile money account at 2024-05-10 16:30:51. Message from sender: . Your new balance:2000 RWF. Financial Transaction Id: 76662021700." }, { "Id": "TxID2", "transaction_type": "payment", "amount": 1000, "sender": "Me", "receiver": "Jane Smith", "timestamp": "2024-05-10 16:31:39", "raw_body": "TxId: 73214484437. Your payment of 1,000 RWF to Jane Smith 12845 has been completed at 2024-05-10 16:31:39. Your new balance: 1,000 RWF. Fee was 0 RWF.Kanda*182*16# wiyandikishe muri poromosiyo ya BivaMoMotima, ugire amahirwe yo gutsindira ibihembo bishimishije." }]`
- **Error Response:**
  - **Code:** `401 Unauthorized`
  - **Body:** `Unauthorized`

Get Transaction by ID

- **Description:** Retrieves a single transaction by its ID.
- **Method:** `GET`
- **Path:** `/transactions/{id}`
- **Request:** `http GET /transactions/TxID1 HTTP/1.1 Host: localhost:8000 Authorization: Basic YWRtaW46cGFzc3dvcmQxMjM=`
- **Success Response:**
  - **Code:** `200 OK`

- **Body:** `json { "Id": "TxID1", "transaction_type": "deposit", "amount": 2000, "sender": "Jane Smith (*********013) on your mobile money account", "receiver": "My Account", "timestamp": "2024-05-10 16:30:51", "raw_body": "You have received 2000 RWF from Jane Smith (*********013) on your mobile money account at 2024-05-10 16:30:51. Message from sender: . Your new balance:2000 RWF. Financial Transaction Id: 76662021700." }`

- **Error Responses:**
  - **Code:** `401 Unauthorized`
  - **Body:** `Unauthorized`
  - **Code:** `404 Not Found`
  - **Body:** `json { "error": "Not found" }`

Create Transaction

- **Description:** Creates a new transaction.
- **Method:** `POST`
- **Path:** `/transactions`
- **Request:**
  - **Headers:** `Content-Type: application/json Authorization: Basic YWRtaW46cGFzc3dvcmQxMjM=`
  - **Body:** `json { "transaction_type": "payment", "amount": 2500, "sender": "Me", "receiver": "John Doe", "timestamp": "2025-09-02T14:30:00", "raw_body": "Your payment of 2500 RWF to John Doe has been completed." }`

- **Success Response:**
  - **Code:** `201 Created`
  - **Body:** `json { "Id": "TxID1692", "transaction_type": "payment", "amount": 2500, "sender": "Me", "receiver": "John Doe", "timestamp": "2025-09-02T14:30:00", "raw_body": "Your payment of 2500 RWF to John Doe has been completed." }`

- **Error Responses:**
  - **Code:** `400 Bad Request`
  - **Body:** `json { "error": "Invalid JSON" }`
  - **Code:** `401 Unauthorized`
  - **Body:** `Unauthorized`

Update Transaction

- **Description:** Updates an existing transaction.
- **Method:** `PUT`
- **Path:** `/transactions/{id}`
- **Request:**
  - **Headers:** `Content-Type: application/json Authorization: Basic YWRtaW46cGFzc3dvcmQxMjM=`
  - **Body:** `json { "raw_body": "Updated raw body" }`
- **Success Response:**
  - **Code:** `200 OK`
  - **Body:** `json { "message": "record UpdatedTxID1" }`
- **Error Responses:**
  - **Code:** `400 Bad Request`
  - **Body:** `json { "error": "Invalid JSON" }`
  - **Code:** `401 Unauthorized`
  - **Body:** `Unauthorized`
  - **Code:** `404 Not Found`
  - **Body:** `json { "error": "Not found" }`

Delete Transaction

- **Description:** Deletes a transaction by its ID.
- **Method:** `DELETE`
- **Path:** `/transactions/{id}`
- **Request:** `http DELETE /transactions/TxID1 HTTP/1.1 Host: localhost:8000 Authorization: Basic YWRtaW46cGFzc3dvcmQxMjM=`
- **Success Response:**
  - **Code:** `200 OK`
  - **Body:** `json { "message": "Record DeletedTxID1" }`
- **Error Responses:**
  - **Code:** `401 Unauthorized`
  - **Body:** `Unauthorized`
  - **Code:** `404 Not Found`
  - **Body:** `json { "error": "Not found" }`

Error Codes

- `400 Bad Request`: The request body is not valid JSON.
- `401 Unauthorized`: The `Authorization` header is missing or invalid.
- `404 Not Found`: The requested resource (transaction or endpoint) does not exist.

Security Note

Basic Auth sends base64-encoded credentials; it's not encrypted. Always use HTTPS and consider JWT or OAuth2 for stronger auth and better token management.

DSA Comparison Results

The `dsa/dsa_compare.py` script was executed to compare the performance of linear search versus dictionary lookup.

- **Records:** 1691
- **Sample IDs:** 20
- **Linear search total time for 20 searches:** 0.000033 sec
- **Dict lookup total time for 20 searches:** 0.000010 sec
- **Dict lookup is faster by factor:** 3.39

These results demonstrate that dictionary lookup is significantly faster for this use case.

Reflection on Basic Auth Limitations

While simple to implement, Basic Authentication has significant limitations, especially for production environments:

- **Credentials Sent in Plain Text (without HTTPS):** Credentials are only Base64 encoded, not encrypted. Without HTTPS, they can be easily intercepted.
- **Vulnerability to Brute-Force Attacks:** Basic Auth is susceptible to brute-force attacks. Measures like rate limiting are necessary to mitigate this risk.
- **No Session Management:** Credentials must be sent with every request, which is inefficient and does not support session management or Single Sign-On (SSO).
- **Limited Granular Control:** It is difficult to implement fine-grained access control. It typically grants all-or-nothing access.

- **Poor User Experience:** Repeated credential prompts can be frustrating for users.

For these reasons, more secure authentication methods like **OAuth 2.0** or **JWT (JSON Web Tokens)** are recommended for production APIs.