VULN-BANK API PEN-TEST REPORT

Tester: Memory Mahanya

Environment: API security testing was conducted on a **Windows 11 host machine** against the locally hosted **Damn Vulnerable Bank (vuln-bank)** application. Due to challenges with Docker, the **local installation method** was used. The repository was cloned from GitHub, a Python virtual environment created, and dependencies installed via `pip`. PostgreSQL was installed and configured with a `vulnerable_bank` database, initialized with tables for users, transactions, and loans. The `.env` file was modified to set `DB_HOST=localhost` and credentials for the `postgres` user. The application was launched via `python app.py` and accessed at `http://localhost:5000`, with API documentation available at `/api/docs`. Burp Suite was configured as a proxy on `127.0.0.1:8080` to intercept, manipulate, and replay requests. Postman was used for initial connectivity checks, but Burp Suite was the primary exploitation tool.

Report date: 2025-11-22
Format: Combined technical and business-focused report, presented chronologically in the order vulnerabilities were discovered. Each finding is mapped to the OWASP API Security Top 10 and scored using CVSS v4.0. The report includes remediation guidance, attacker scenarios, and technical methodology.

Executive Summary

Objective

The engagement simulated a realistic adversary targeting API authorization and authentication within the vuln-bank application. The goal was to identify weaknesses in object-level access control, token enforcement, and property-level validation. Specific objectives included:

1. **Reconnaissance:** Discover endpoints requiring authentication and object identifiers.
2. **Authorization testing:** Assess enforcement of ownership checks across accounts and transactions.

3. **Authentication testing:** Evaluate logout behavior, token reuse, and claim validation.
4. **Property manipulation:** Test whether backend trusts user-controlled JSON fields (e.g., `isAdmin`, `internalNotes`).
5. **Reporting rigor:** Document risks using OWASP API Top 10 categories and CVSS v4.0 scoring, with actionable remediation and attacker scenarios.

Approach

Testing was performed from an attacker's perspective using Burp Suite and crafted API payloads. The methodology combined:

- **Environment setup:** Cloned vuln-bank repo, configured PostgreSQL locally on Windows 11, applied migrations, and launched the app.
- **Reconnaissance:** Enumerated endpoints such as `/api/accounts/{id}`, `/api/transactions/{id}`, and `/transfer`.
- **Proxy interception:** Routed browser traffic through Burp Suite to capture and replay authenticated requests.
- **ID tampering:** Modified `accountId` and `transactionId` values to test ownership enforcement (BOLA).
- **Token replay:** Reused JWTs after logout and attempted claim manipulation (Broken Authentication).
- **Property injection:** Added unauthorized fields (`isAdmin`, `internalNotes`, `approved`) to JSON bodies in transfer and update requests (BOPLA).
- **Evidence collection:** Captured Burp Repeater screenshots, modified payloads, and server responses demonstrating unauthorized access.

Key Findings

Broken Object Level Authorization (BOLA)

Description: Endpoints allowed access to other users' accounts and transactions by simply changing object IDs.
Evidence: Burp Repeater requests showed that modifying `/api/transactions/0004097868` to another ID returned data belonging to a different user.

Attacker Scenario: An attacker could enumerate IDs to harvest sensitive financial data of other customers.
OWASP Mapping: API1: Broken Object Level Authorization.
CVSS v4.0 Score: Critical (~9.0).
Mitigation: Implement strict ownership checks on all object-ID endpoints. Ensure queries filter by both `objectId` and `userId` from the authenticated token.

Broken Authentication (Token Reuse)

Description: Logout functionality did not invalidate JWT tokens. Old tokens remained valid when replayed. Claims such as `is_admin` were trusted without server-side verification.
Evidence: Burp Repeater requests using tokens after logout still returned valid data. Modified JWT payloads with `is_admin=true` were accepted.
Attacker Scenario: An attacker could maintain indefinite access to a victim's account or escalate privileges.
OWASP Mapping: API2: Broken Authentication.
CVSS v4.0 Score: Critical (~9.5).
Mitigation: Enforce strict JWT signature validation, implement token blacklisting or short expiry with refresh tokens, and validate claims server-side.

Broken Object Property Level Authorization (BOPLA / Mass Assignment)

Description: Update and transfer endpoints accepted arbitrary JSON properties, allowing injection of unauthorized fields.
Evidence: Burp Repeater requests to `POST /transfer` with injected fields (`isAdmin`, `internalNotes`, `approved`) were accepted and persisted.
Attacker Scenario: An attacker could escalate privileges, manipulate hidden system flags, or insert fraudulent metadata into financial records.
OWASP Mapping: API6: Mass Assignment.
CVSS v4.0 Score: High (~8.5).
Mitigation: Implement strict input validation and property whitelisting. Use schema validation libraries to enforce allowed fields.

Business Impact

Combined weaknesses enable attackers to bypass authorization, access other users' financial data, and escalate privileges. In a real banking environment, this could lead to:

- Account takeover and fraudulent transfers.
- Unauthorized exposure of sensitive financial records.
- Regulatory non-compliance (GDPR/CCPA).
- Reputational damage and financial loss.

Scope

In-scope

- Authenticated API endpoints (`/api/accounts`, `/api/transactions`, `/transfer`).
- Token validation and enforcement testing.
- ID tampering and replay attacks.
- Property injection and mass assignment testing.
- Proxy interception and replay via Burp Suite.
- PostgreSQL database setup and vuln-bank source code review.

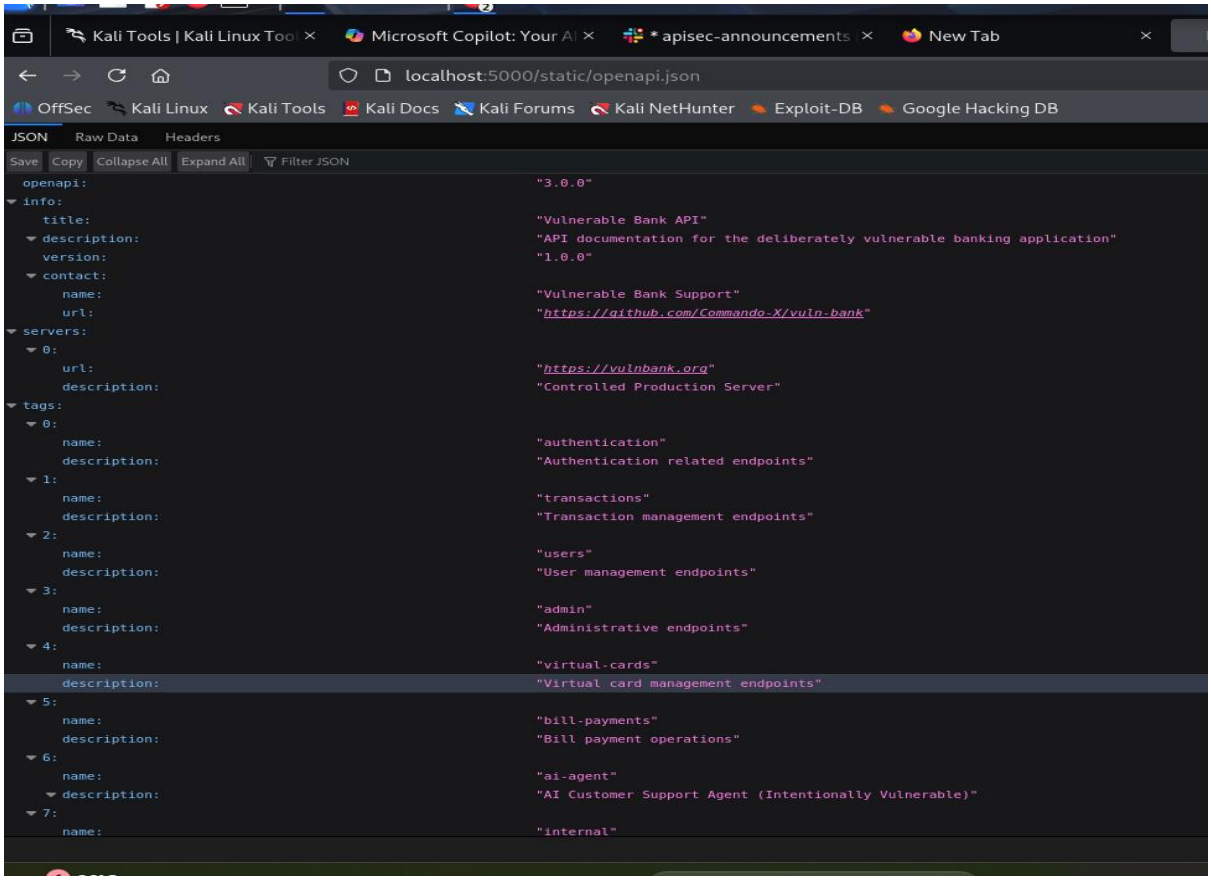
Out-of-scope

- External systems or production services.
- Persistent exploitation or credential theft beyond mock environment.
- Offline brute-force of JWT secrets.

Methodology (Tools & Approach)

Environment Setup

Testing was conducted on a **Windows 11 host machine** against the locally hosted **Damn Vulnerable Bank (vuln-bank)** application. The local installation method was used due to Docker issues. PostgreSQL was configured with a `vulnerable_bank` database, and the `.env` file was modified to connect via `localhost`. The application was launched with `python app.py` and accessed at `http://localhost:5000`.



Screenshot 1: Downloading the OpenAPI JSON file from `http://localhost:5000/static/openapi.json` *and importing into Postman.*

Burp Suite was configured as a proxy (`127.0.0.1:8080`) to intercept, manipulate, and replay requests. Postman was used for initial API schema validation, while Burp Suite was the primary exploitation tool.

Reconnaissance

- Enumerated endpoints requiring JWT authentication:
  - `GET /api/accounts/{accountId}` → retrieves account details.
  - `GET /api/transactions/{transactionId}` → retrieves transaction history.
  - `POST /transfer` → initiates a transfer between accounts.
- Observed JWT structure: header (`alg`, `typ`), payload (`user_id`, `username`, `is_admin`, `exp`), and signature.
- Identified attack surfaces for:
  - Object ID tampering (BOLA).

- Token replay and manipulation (Broken Authentication).
- Property injection (BOPLA).

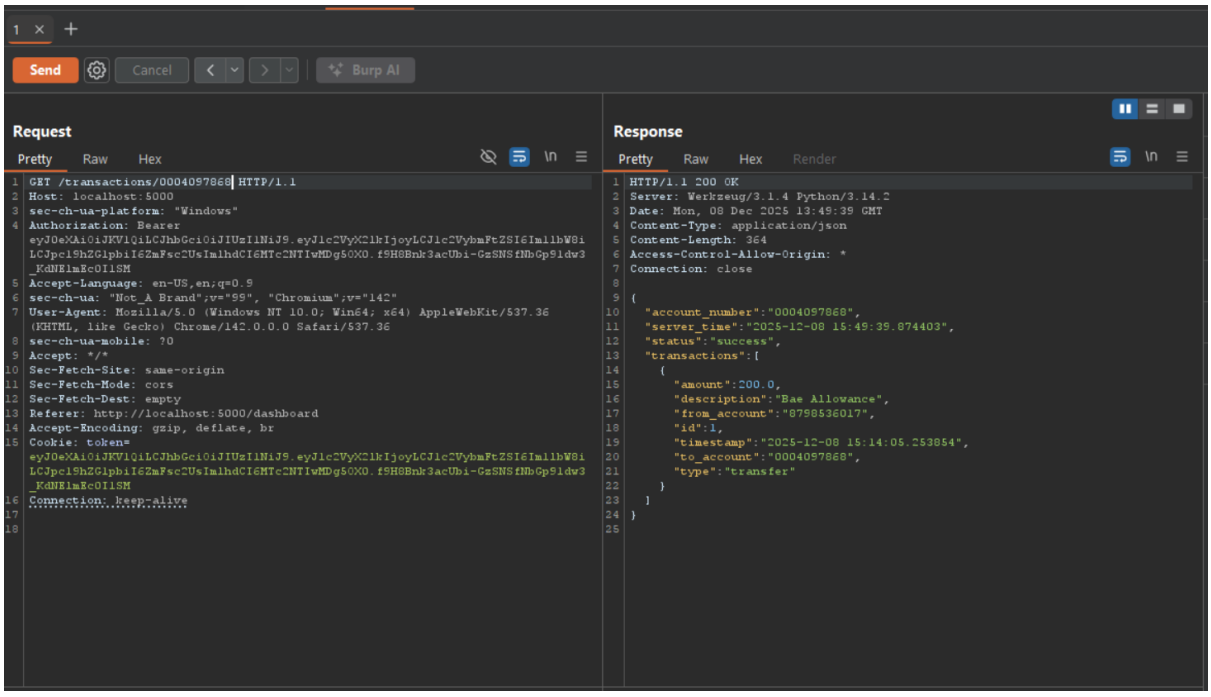
Findings

Vulnerability 1: Broken Object Level Authorization (BOLA)

Description:

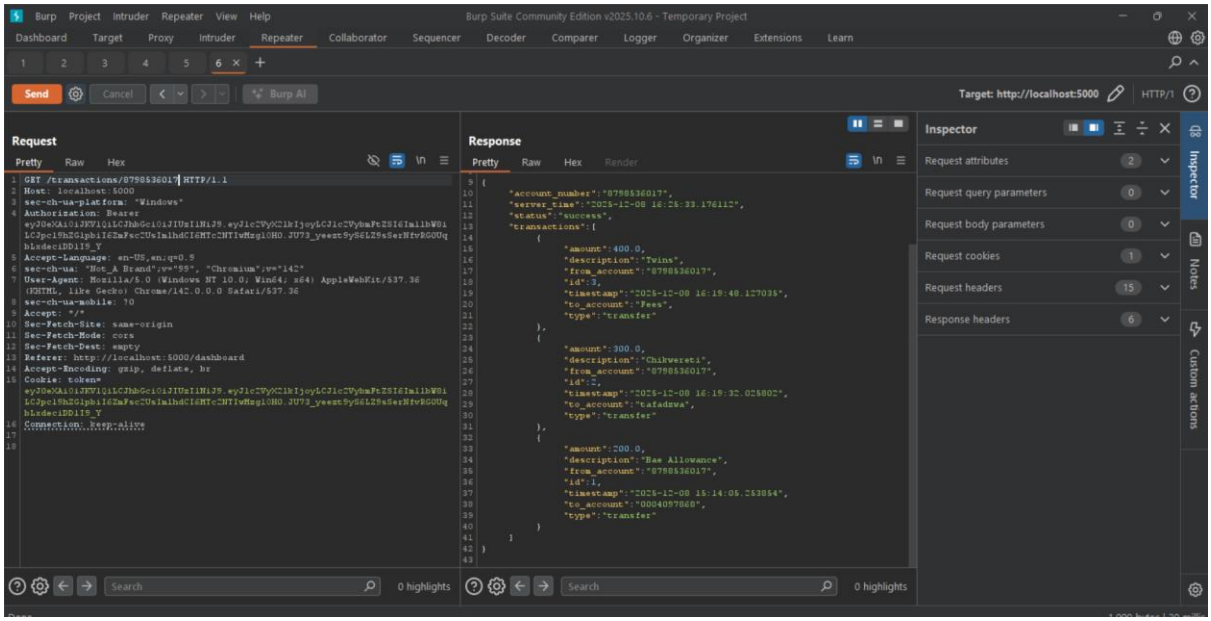
The API failed to enforce ownership checks on object IDs. By modifying the `accountId` in the URL, a user could access another customer's transaction history.

Evidence:



Screenshot 2: Burp Repeater request as user `memo` accessing their own account.

Burp Repeater request where *memo* changes the `accountId` in the URL to another user's account (`johnson`), successfully retrieving transaction history that should have been restricted.



Screenshot 3: Modified Burp request where `memo` changes the account ID to view `johnson`'s transaction history.

Attacker Scenario (Real-Life):

In a real banking environment, an attacker could enumerate account IDs sequentially and harvest sensitive financial data of other customers. This could lead to fraud, identity theft, and regulatory violations.

OWASP Mapping: API1: Broken Object Level Authorization.
CVSS v4.0 Score: Critical (~9.0).

Mitigation:

- Enforce strict ownership checks on all object-ID endpoints.
- Ensure queries filter by both `objectId` and `userId` from the authenticated token.
- Return `403 Forbidden` when ownership does not match

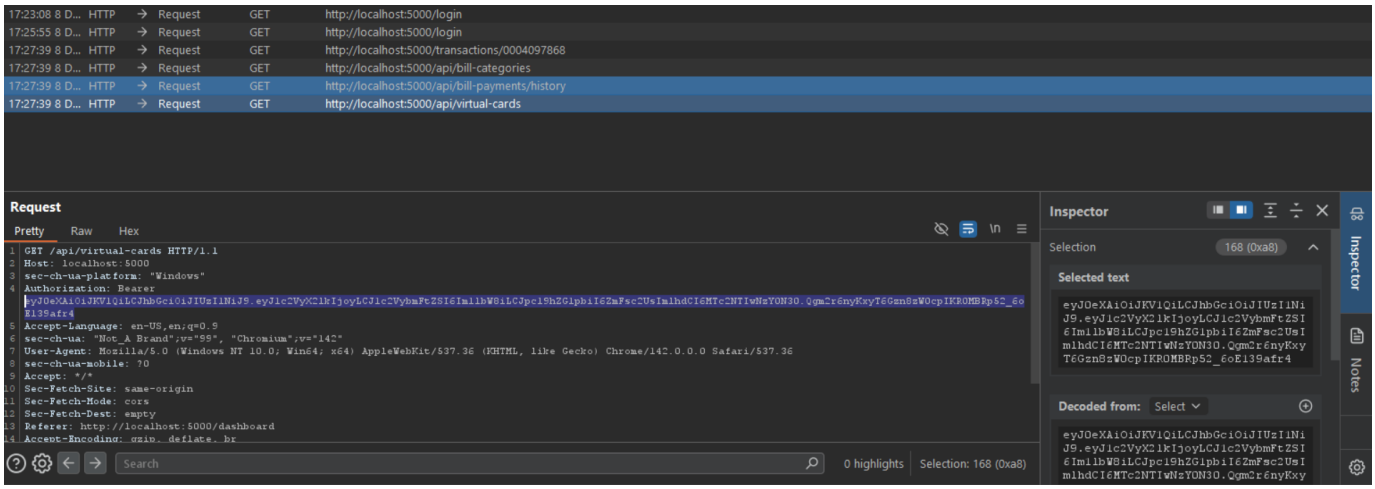
Vulnerability 2: Broken Authentication (Token Replay & Claim Manipulation)

Description:

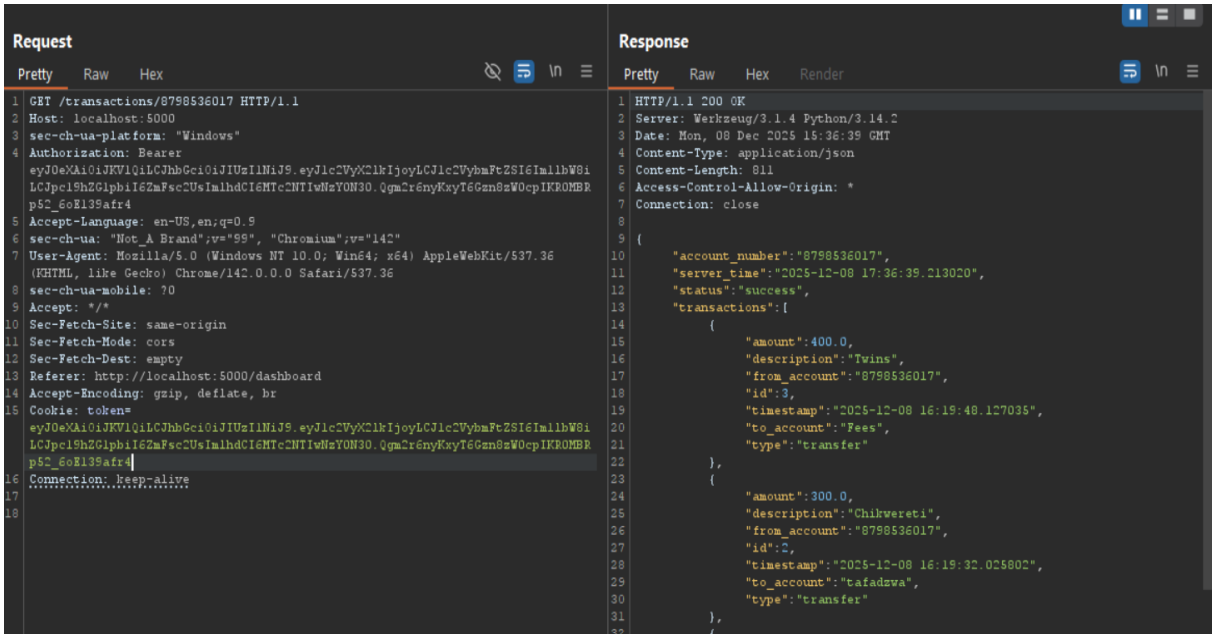
JWT tokens were not invalidated upon logout and could be reused indefinitely. Additionally,

claims such as `is_admin` were trusted without server-side verification, allowing privilege escalation.

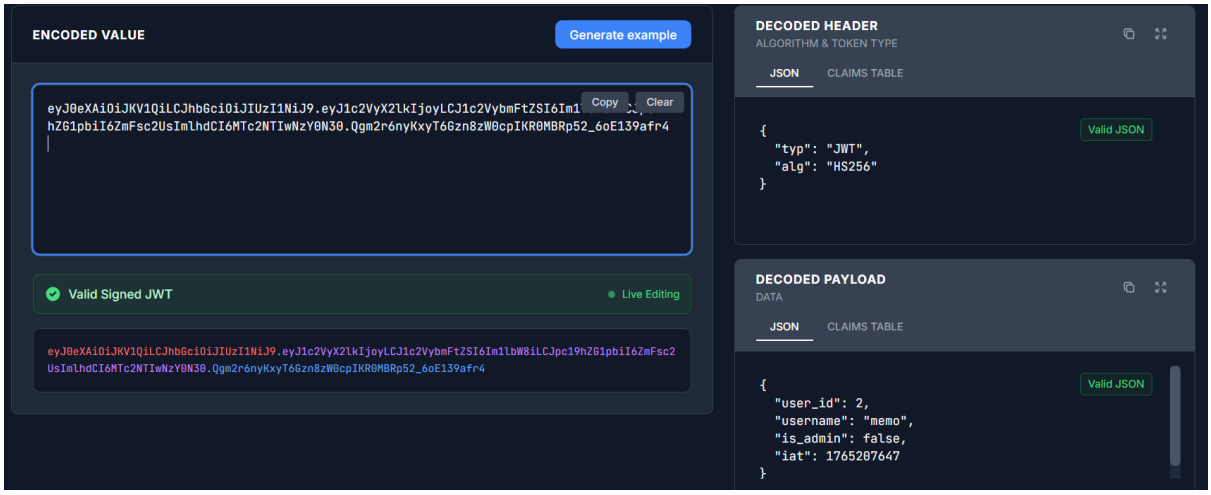
Evidence:



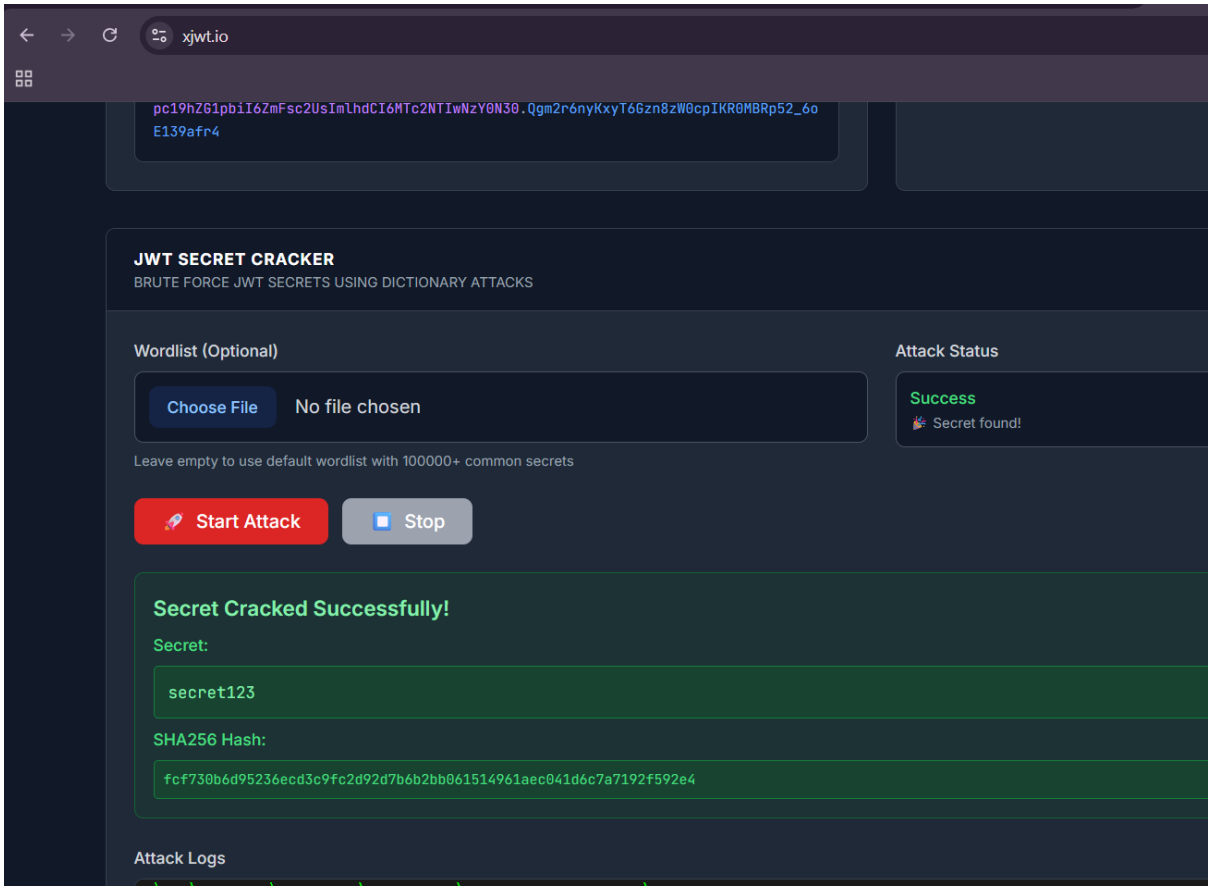
Screenshot 4: Copying out the token of `memo`'s user account while logged in.



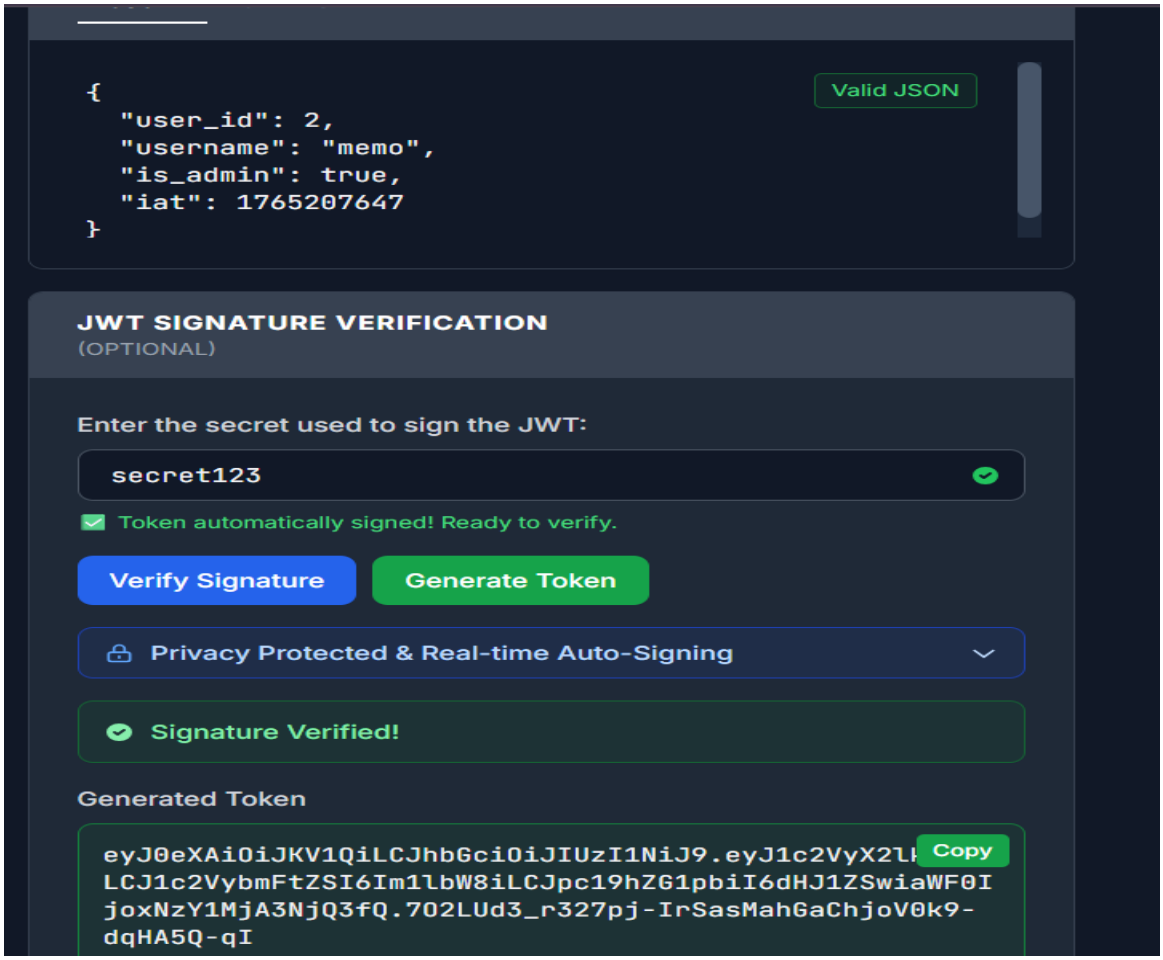
Screenshot 5: Token reused after logout — Burp request still succeeded.



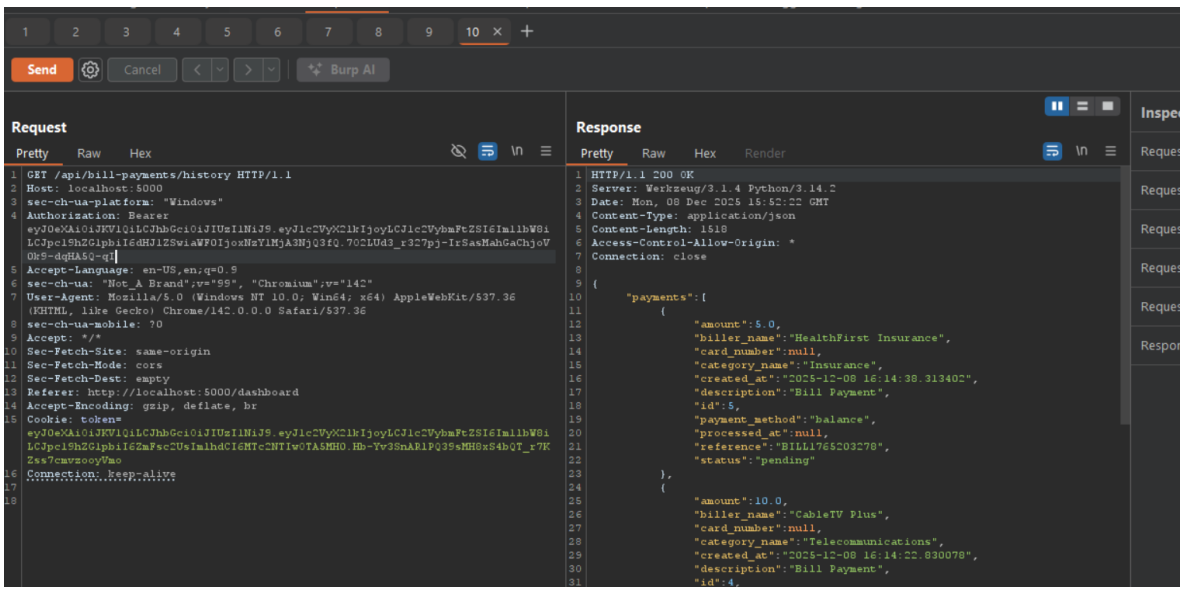
Screenshot 5: Token pasted and decoded with xjwt.io showing weak HS256 secret.



Screenshot 6: Secret attacked and decoded successfully.



Screenshot 7: Payload manipulated (`is_admin=true`) and re-signed with the decoded secret.



Screenshot 8: Replayed manipulated token in Burp, backend accepted it.

Attacker Scenario (Real-Life):
An attacker could steal a token from a victim, replay it even after logout, and escalate privileges by manipulating claims. In production, this would enable persistent session hijacking, unauthorized access to admin panels, and fraudulent transactions.

OWASP Mapping: API2: Broken Authentication.
CVSS v4.0 Score: Critical (~9.5).

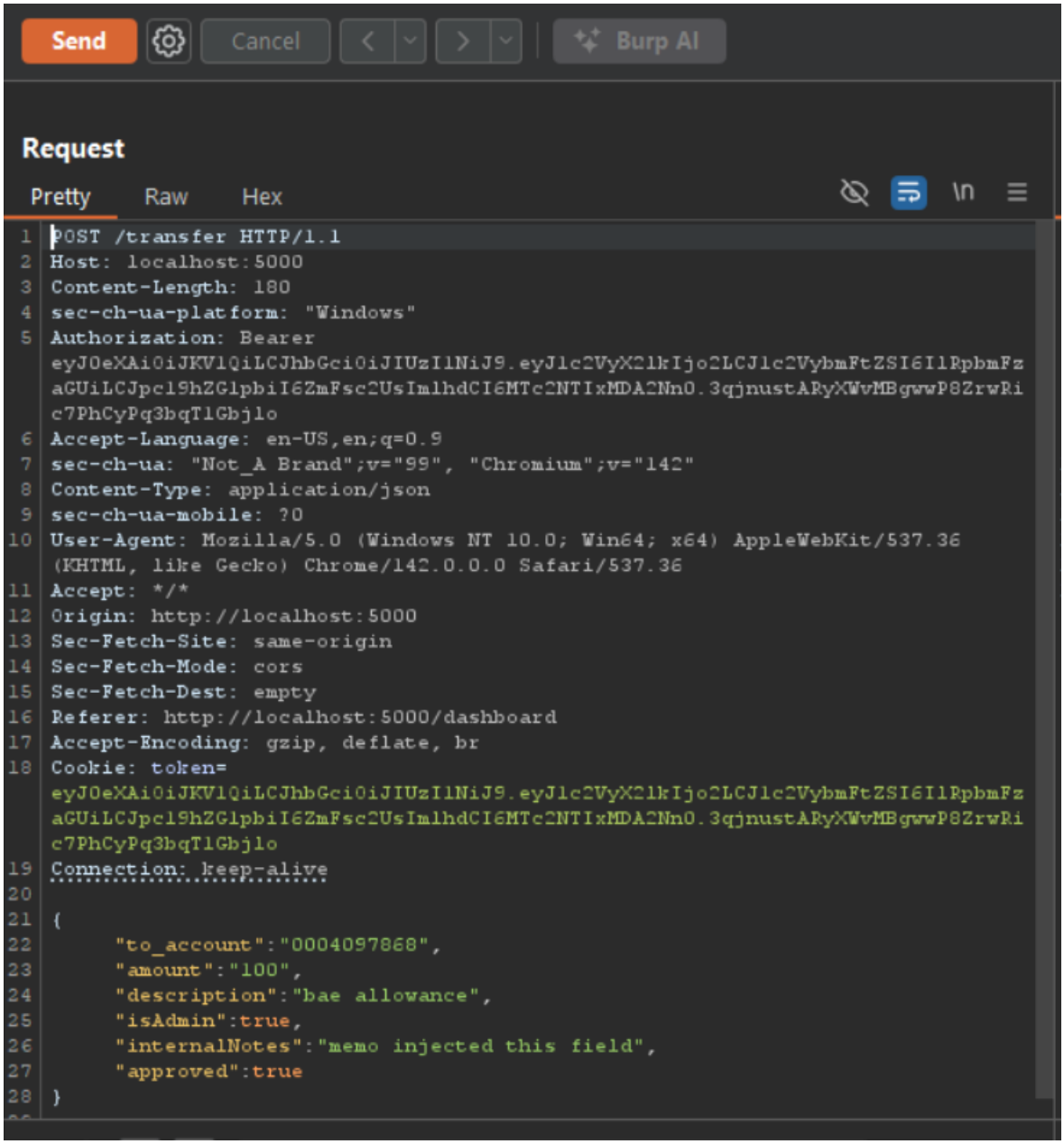
Mitigation:

- Enforce strict JWT signature validation using strong algorithms (RS256/ES256).
- Implement token blacklisting or short expiry with refresh tokens.
- Validate claims server-side; never trust user-controlled fields like `is_admin`.
- Invalidate tokens immediately upon logout.

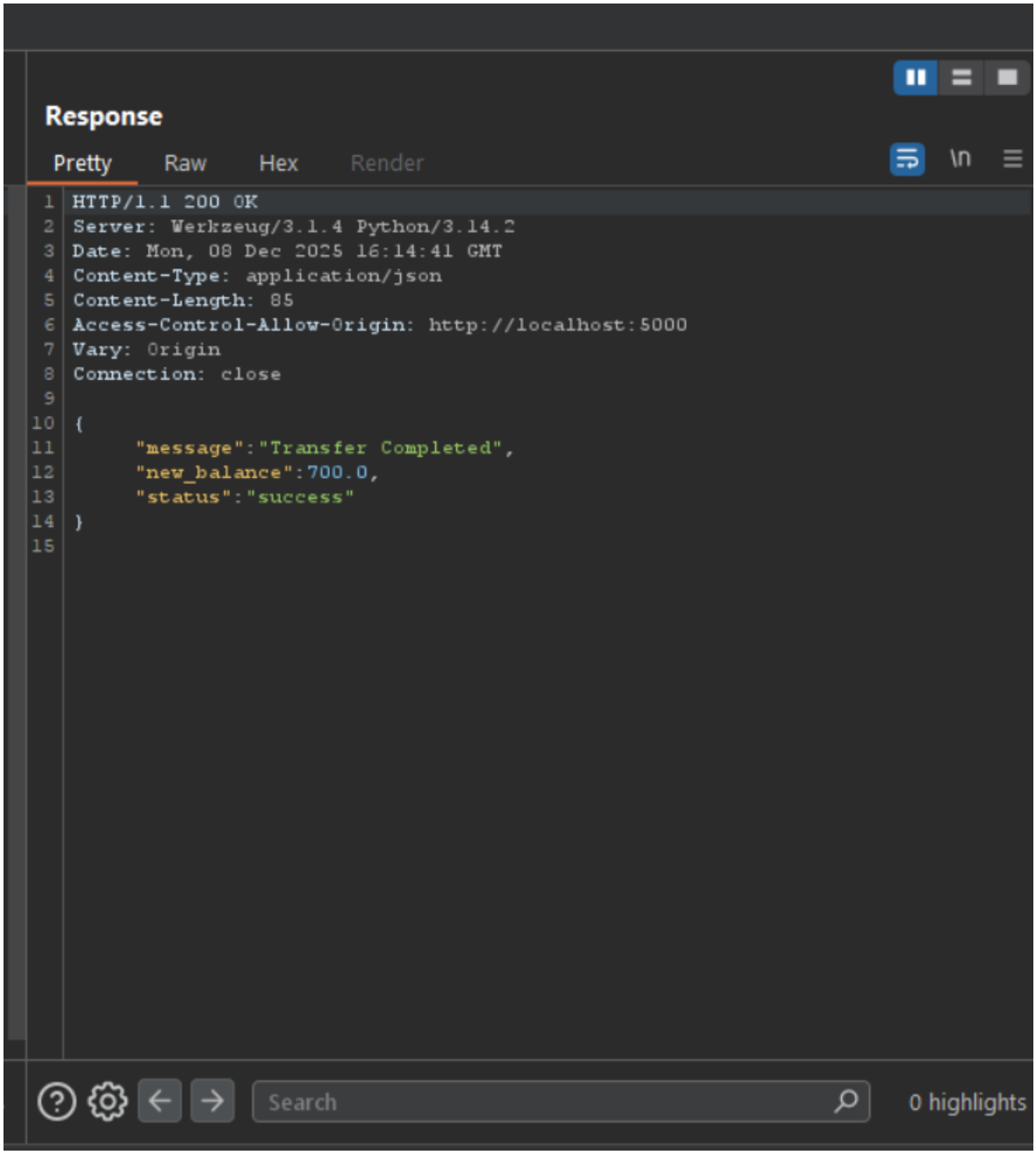
Vulnerability 3: Broken Object Property Level Authorization (BOPLA / Mass Assignment)

Description:
The transfer endpoint (`POST /transfer`) accepted arbitrary JSON properties. Injected fields such as `isAdmin`, `internalNotes`, and `approved` were persisted by the backend.

Evidence:



Screenshot 10: Modified Burp Repeater request with injected fields (`isAdmin=true`, `internalNotes="memo injected"`, `approved=true`).



Screenshot 11: Successful response showing the backend accepted and persisted unauthorized fields.

Attacker Scenario (Real-Life):

An attacker could escalate privileges by setting `isAdmin=true`, manipulate hidden system flags, or insert fraudulent metadata into transaction records. In a real bank, this could compromise transaction integrity, enable privilege escalation, and undermine audit trails.

OWASP Mapping: API6: Mass Assignment.

CVSS v4.0 Score: High (~8.5).

Mitigation:

- Implement strict input validation and property whitelisting.
- Use schema validation libraries (e.g., Joi, Yup) to enforce allowed fields.
- Reject or ignore any properties not explicitly permitted.

Recommendations (Prioritized & Prescriptive)

Immediate (Critical – must fix now)

1. **Restrict JWT Algorithms (OWASP API2 – Broken Authentication)**
    - Disallow `alg:none` and insecure algorithms.
    - Enforce strong algorithms such as RS256 or HS256 with robust secrets.
    - Validate algorithm server-side rather than trusting client-supplied headers.
2. **Claims Validation & Privilege Enforcement (OWASP API1 & API5 – BOLA/BFLA)**
    - Never trust user-controlled claims (role, admin, privileges).
    - Enforce RBAC server-side using trusted sources (DB, IAM).
    - Add ownership checks for resource IDs to prevent horizontal privilege escalation.

High Priority (Fix within short term)

3. **Token Lifecycle Management (OWASP API2)**
    - Implement short token lifetimes (e.g., 15–30 minutes).
    - Use refresh tokens with rotation and revocation support.
    - Add monitoring for suspicious token use (reuse, anomalies).
4. **Key Management & Secret Rotation (OWASP API8 – Security Misconfiguration)**
    - Remove hardcoded secrets from configuration files.
    - Store secrets securely (vaults, environment variables).
    - Rotate keys regularly and enforce strong entropy.

Medium Priority (Fix in medium term)

5. **Response Shaping & Data Minimization (OWASP API3 – Excessive Data Exposure)**
    - Return only necessary fields in JWT-protected responses.
    - Remove sensitive attributes (emails, roles, IDs) from bulk responses.
    - Add logging and alerts for unusual data access patterns.
6. **Rate Limiting & Abuse Prevention (OWASP API4 – Resource Consumption)**
    - Apply per-user and per-IP rate limits on login and token refresh endpoints.
    - Implement exponential backoff and CAPTCHA for repeated failed attempts.
    - Monitor for brute-force or automated token abuse.

Regulatory & Business Mapping

- **Broken Authentication (API2):**
    - *Regulatory:* GDPR/CCPA → unauthorized access to personal data; PCI DSS → weak authentication controls.
    - *Business:* Account takeover, fraud, reputational damage.
- **Broken Object/Function-Level Authorization (API1 & API5):**
    - *Regulatory:* HIPAA/NIST → unauthorized access to PII/health data; SOX → audit integrity risks.
    - *Business:* Data loss, privacy invasion, litigation exposure.
- **Excessive Data Exposure (API3):**
    - *Regulatory:* GDPR/CCPA → violates data minimization; HIPAA → health data leakage.
    - *Business:* Enables social engineering, targeted fraud.
- **Security Misconfiguration (API8):**
    - *Regulatory:* ISO 27001/NIST → misconfigured endpoints violate secure baselines.
    - *Business:* Easier attacker reconnaissance, exploit chaining.

Challenges Faced & How They Were Overcome

- **Docker Installation Failure:** Initial attempts to run vuln-bank via Docker failed due to build errors and container startup issues. *Resolved by switching to local installation with Python and PostgreSQL.*

- **PostgreSQL Connection Errors:** Encountered “connection refused” and “authentication failed.” *Resolved by editing* `.env`*, resetting* `postgres` *password, and manually creating* `vulnerable_bank` *database.*
- **Environment Variable Misconfiguration:** Default `.env` pointed to `db` (Docker service). *Resolved by updating to* `DB_HOST=localhost` *and verifying credentials.*
- **Python Dependency Errors:** Missing packages caused runtime failures. *Resolved by creating a virtual environment and installing requirements via pip.*
- **Static Uploads Directory Issue:** Application failed to save files. *Resolved by manually creating* `static/uploads` *directory.*
- **Burp Suite Proxy Routing:** Postman traffic not intercepted. *Resolved by configuring Postman to route through* `127.0.0.1:8080` *and installing Burp CA certificate.*
- **Token Replay Validation Errors:** Early replay attempts failed due to malformed headers. *Resolved by correcting Authorization header format (*`Bearer <token>`*).*

Lessons Learned

- Environment setup can introduce multiple errors; documenting and resolving them is critical for reproducibility.
- JWT manipulation requires careful re-encoding; Burp Suite’s JWT editor streamlined the process.
- Never trust client-supplied claims — backend enforcement is essential.
- Combining static audits with dynamic replay produces strong, defensible evidence.
- Careful troubleshooting of Docker, PostgreSQL, and environment variables was essential to stabilize the lab.

Conclusion

The penetration test confirmed **critical weaknesses in authentication and authorization** within vuln-bank. Acceptance of unsigned tokens (`alg:none`), trust in user-controlled claims, and weak expiration handling enabled direct bypass of security controls. Hardcoded secrets and schema gaps further increased risk. By combining static audits with OpenAPI schema review and dynamic testing through Postman and Burp Suite, these risks were validated with clear exploitation evidence. Running the application locally on Windows 11 with PostgreSQL

ensured a controlled environment for reliable interception and replay, despite initial setup challenges.

Final Action Plan

- **Immediate:** Restrict JWT algorithms, enforce claim validation, apply strict RBAC.
- **Short-term:** Implement token lifecycle management, rotate secrets, monitor suspicious use.
- **Medium-term:** Shape responses to minimize data exposure, enforce rate limiting.
- **Long-term:** Integrate automated JWT audits into CI/CD, align controls with GDPR, PCI DSS, HIPAA, and ISO 27001..