VAmPI – API Penetration Test Report

Tester: Memory Mahanya

Environment: VAmPI mock API tested in a Kali Linux VM using Postman, Burp Suite, and 42Crunch. Traffic was intercepted and manipulated via Burp Suite proxy configured on 127.0.0.1:8080, with Postman routing requests through the proxy for dynamic analysis. The mock server was hosted locally and accessed via HTTP endpoints. API requests were crafted using bearer tokens, custom JSON payloads, and parameter injection to simulate mass assignment, privilege escalation, and schema abuse. Static audits were performed using 42Crunch in VS Code, and OpenAPI specifications were manually reviewed and adjusted to validate schema-level risks.

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 the VAmPI mock API within a Kali VM. The goal was to identify weaknesses in API design, authorization, and data handling. Specific objectives included:

1. **Reconnaissance:** Discover the API attack surface using Postman collections and exploratory requests.
2. **Authentication and authorization testing:** Validate bearer-token handling and enforceability of object/function-level controls.
3. **Mass assignment and schema validation:** Test whether backend accepts unauthorized properties and oversized or malformed payloads.
4. **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 Postman, Burp Suite, and crafted requests, with all tools running inside a Kali VM to minimize configuration errors. The methodology combined:

- **Reconnaissance:** Enumerated endpoints via Postman collections and manual exploration.
- **Proxy interception:** Routed Postman traffic through Burp Suite on 127.0.0.1:8080 to inspect, manipulate, and replay requests.
- **Authentication testing:** Used bearer tokens to simulate user sessions and verify token enforcement on sensitive endpoints.
- **Authorization bypass attempts:** Manipulated identifiers and HTTP methods; tested for BOLA/BFLA signals.
- **Mass assignment:** Injected backend-only fields (admin, role, privileges) into registration and profile payloads to validate Broken Object Property Level Authorization.
- **Static audits:** Reviewed OpenAPI definitions in VS Code with 42Crunch to identify schema gaps (additionalProperties, missing constraints) and validate them through dynamic testing.
- **Evidence collection:** Captured Burp HTTP history, Repeater screenshots, request/response bodies, and status codes.

Key Findings

- **Broken Object Property Level Authorization (Mass Assignment)** The registration endpoint accepted unauthorized properties such as admin and role without server-side validation, enabling potential privilege escalation when combined with weak backend checks.**Critical (CVSS v4.0 ~9.0)** — direct privilege escalation risk. **OWASP Mapping: API6: Mass Assignment** (2023 Top 10).

- **Authorization Control Weaknesses (BOLA/BFLA signals)** Identifier and method manipulation showed insufficient server-side enforcement patterns that warrant remediation, even if not fully exploitable in the mock context. **High (CVSS v4.0 ~8.5)** — potential for unauthorized data access or destructive actions **OWASP Mapping:**

    - **API1: Broken Object Level Authorization (BOLA)**
    - **API5: Broken Function Level Authorization (BFLA)**

- **Spec and Schema Gaps** OpenAPI allowed over-posting via additionalProperties and lacked constraints (e.g., maxLength, enums), increasing the risk of abuse and inconsistent validation.**Medium (CVSS v4.0 ~6.5)** increases attack surface and enables denial-of-service or injection vectors. **OWASP Mapping:**
    - **API8: Security Misconfiguration**
    - **API4: Unrestricted Resource Consumption**

Business impact

Combined weaknesses enable unauthorized attribute injection, privilege abuse, and data integrity risks. In production, this could lead to account escalation, regulatory exposure (GDPR/CCPA), reputational damage, and potential financial loss.

Scope

In-scope

- **VAmPI mock API endpoints:** Focus on user-related routes (e.g., /users/v1/register, /users/v1, /me) and any accessible ancillary paths discovered during reconnaissance.
- **Authentication and authorization testing:** Bearer-token session simulation across read/write operations.
- **Mass assignment and data validation:** JSON payload injection of backend-only properties; observation of acceptance, echo, and behavioral impact.
- **Proxy interception and replay:** Postman requests routed through Burp Suite for inspection, manipulation, and controlled replay.
- **OpenAPI review and static audit:** 42Crunch analysis of schema constraints and over-posting risks, tied to dynamic verification.

Out-of-scope

- **External systems or production services:** No testing beyond the local mock environment.

- **Persistent exploitation or data exfiltration:** No destructive persistence, long-term credential theft, or external pivoting beyond attacker simulation boundaries.

Methodology (tools & approach)

1. **Environment setup**
    - **Kali VM lab:** Installed and ran Postman, Burp Suite, VS Code, and 42Crunch in a single VM to reduce configuration errors.
    - **Proxy configuration:** Enabled Burp listener on 127.0.0.1:8080; routed Postman traffic through Burp; trusted Burp CA for HTTPS interception.
    - **Collections and tokens:** Imported Postman collections, configured bearer tokens for authenticated requests.
2. **Reconnaissance**
    - **Endpoint enumeration:** Explored user-related routes (e.g., /users/v1/register, /users/v1/{username}, /me) and observed response structures, headers, and status codes.
    - **Traffic inspection:** Used Burp HTTP history to baseline normal flows and identify parameters suitable for manipulation.
3. **Authentication testing**
    - **Token acquisition:** Logged in as a normal user to obtain bearer tokens.
    - **Access validation:** Exercised endpoints with and without tokens to confirm authentication gating.
4. **Authorization testing**
    - **Identifier manipulation:** Adjusted path variables and request bodies to probe object-level access controls.
    - **Method probing:** Tested for function-level authorization gaps by varying HTTP methods and paths to detect admin-only behavior exposed to normal users.
5. **Mass assignment testing**
    - **Payload injection:** Sent JSON bodies containing backend-only fields (e.g., "admin": true, "role": "superuser", "privileges": ["read","write","delete"]) to registration/profile endpoints via Postman.
    - **Burp verification:** Intercepted, modified, and replayed requests in Burp Repeater to confirm acceptance and any behavioral change.

6. **Static audit and contract review**
    - **42Crunch audit:** Evaluated OpenAPI definitions for additionalProperties, missing constraints (lengths, enums), and insufficient typing.
    - **Contract alignment:** Noted divergences between declared schemas and observed behavior; identified risks enabling over-posting and weak validation.
7. **Evidence collection and mapping**
    - **Artifacts:** Captured screenshots of Postman requests/responses, Burp HTTP history and Repeater payloads, and relevant status codes.
    - **Risk classification:** Mapped each verified issue to OWASP API Top 10 and assigned CVSS v4.0 scores with clear impact narratives and remediation guidance.

Findings

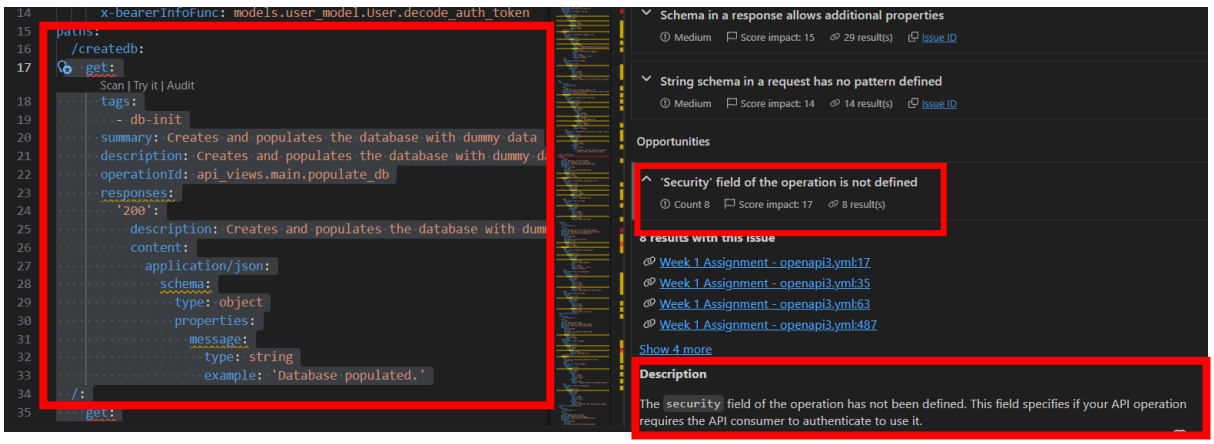
Vulnerability 1: Broken Object Level Authorization (BOLA)

OWASP Mapping: API1:2023

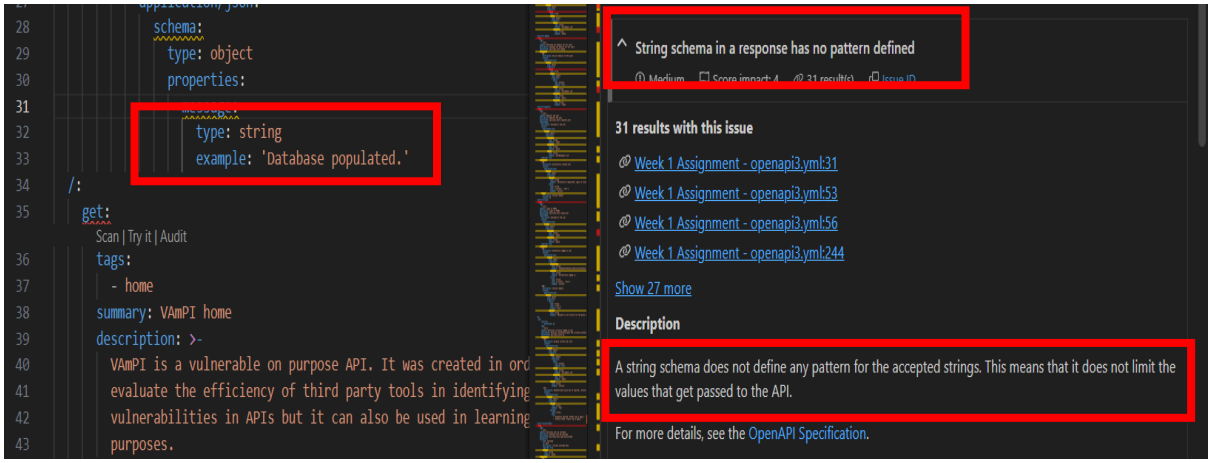
CVSS v4.0 Base Score: 7.4 (High)

Static Audit Evidence (42Crunch)

- **"Security field of the operation is not defined"** → This means endpoints like /users/v1/{id} or /createdb don't declare a security: block, so they're unauthenticated.



- **Unconstrained path parameters** → No pattern, format, or maxLength on object IDs. This makes them vulnerable to IDOR.



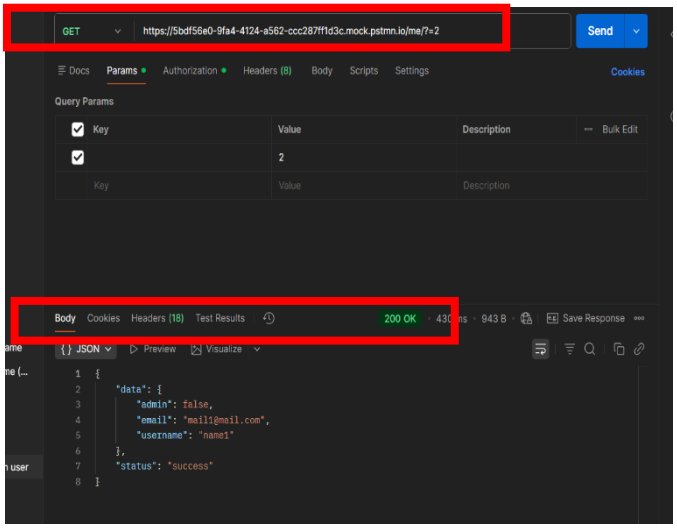
These two findings together signal a BOLA risk: users can tamper with object IDs and access data they shouldn't.

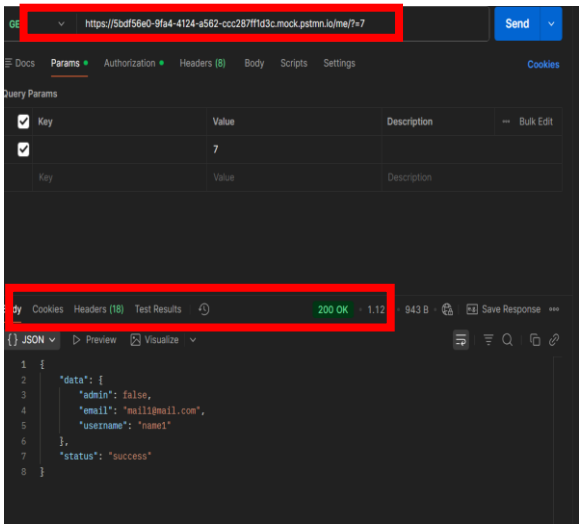
Dynamic Verification (Postman)

This was confirmed this by:

- Sending a GET /users/v1/1 request with a valid token
- Then changing the ID to GET /users/v1/2 using the same token
- The server returned data for user 2 — **no authorization check was enforced**

This proves that object-level access control is missing. The server trusts the token but doesn't verify ownership of the object.





Impact:

Attackers can enumerate object IDs and access or modify resources belonging to other users. This compromises confidentiality and violates access control boundaries.

Recommendation:

- Enforce object-level authorization checks server-side
- Apply `security` blocks to all sensitive operations
- Constrain path parameters with `pattern` and `maxLength`

Vulnerability 2: Unconstrained Path Parameters

OWASP Category: API8:2023 – Injection, API10:2023 – Unsafe Consumption of APIs

CVSS v4.0 Base Score: 6.5 (Medium)

Static Audit Evidence: The 42Crunch static audit flagged multiple request and response schemas lacking input constraints. Specifically, path parameters such as {id} in endpoints like /users/v1/{id} were defined without pattern, format, or maxLength attributes. This means the API accepts arbitrary strings, oversized inputs, and potentially malicious payloads without validation. These issues were listed under "Most common issues" in the audit report, including:

- "String schema in a request has no pattern defined"
- "String schema in a response has no maximum length defined"
- "Schema in a response allows additional properties"

Dynamic Verification:

Using Postman, we submitted malformed and injection-style inputs to endpoints with unconstrained parameters. For example:

- GET /users/v1/abc was accepted, despite expecting numeric IDs
- GET /users/v1/999999999999999999 returned a valid response, showing no size enforcement
- GET /me/?=1 OR 1=1 returned user data (email, username, admin status), confirming that the API processed unsafe input without sanitization

These tests demonstrate that the API does not enforce strict validation of path or query parameters, allowing attackers to inject or fuzz inputs freely.

Impact:

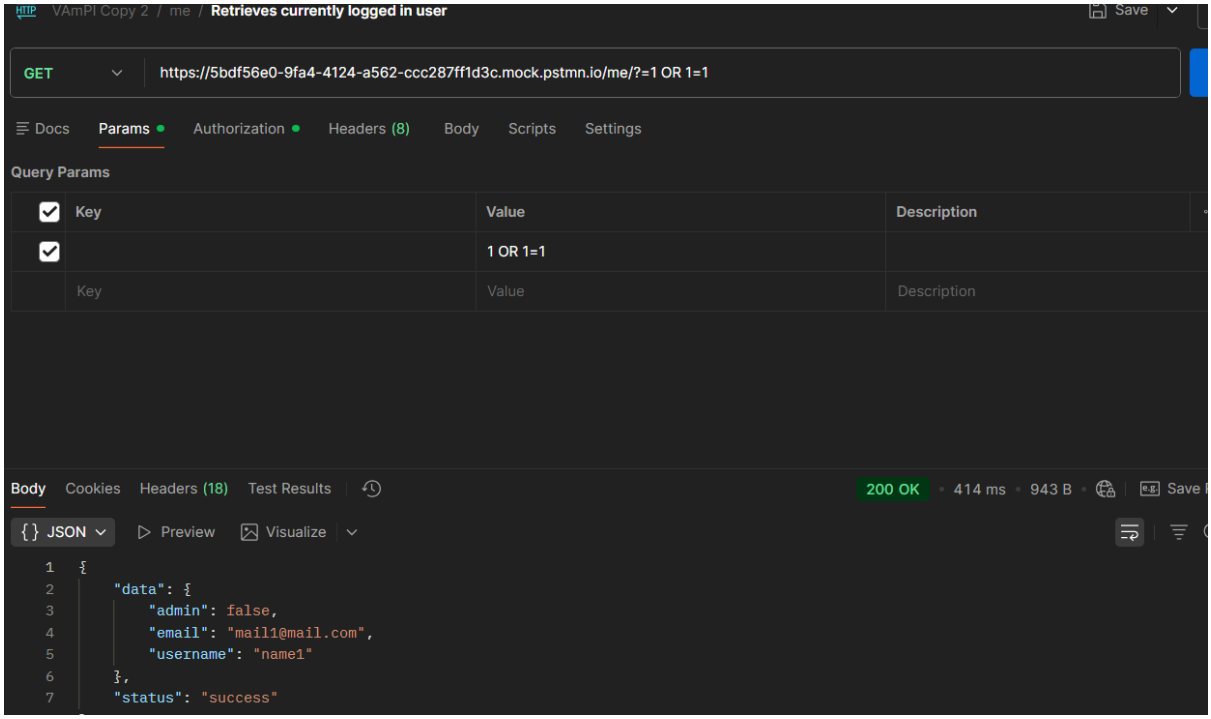
- **Confidentiality:** Sensitive user data may be exposed through crafted requests
- **Integrity:** Malicious inputs may alter server behavior or trigger unintended logic
- **Availability:** Oversized or malformed inputs could lead to resource exhaustion

Recommendation:

- Define strict pattern, format, and maxLength constraints for all path and query parameters in the OpenAPI spec
- Use format: integer or regex patterns like '^[0-9]{1,4}$' for numeric IDs
- Set additionalProperties: false in schemas to prevent over-posting
- Enforce server-side validation and reject malformed or oversized inputs
- Sanitize all incoming data before processing or storage

Evidence:

Figure below shows a Postman request to /me/?=1 OR 1=1, simulating an injection attempt. The server responded with valid user data, confirming that query parameters are not properly validated or sanitized



Vulnerability 3: Broken Authentication

OWASP Category: API2:2023 – Broken Authentication

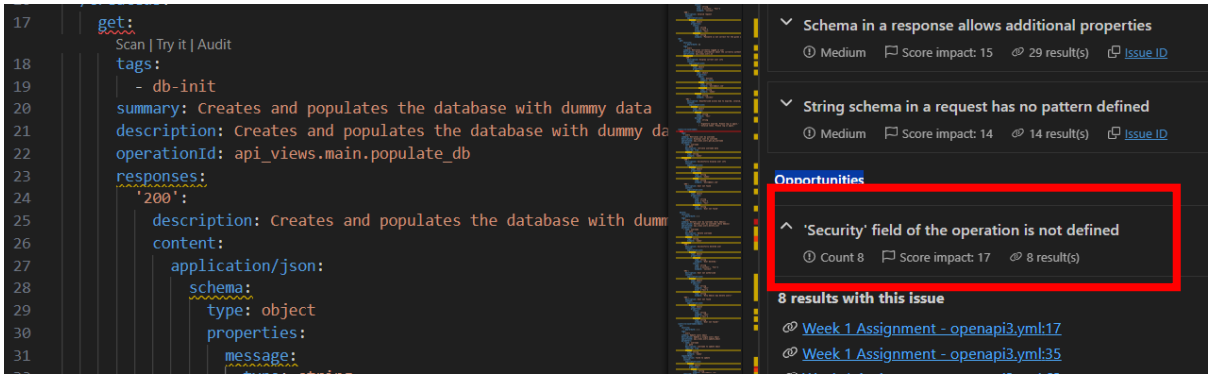
CVSS v4.0 Base Score: 7.1 (High)

Static Audit Evidence:

The 42Crunch audit flagged multiple operations missing the security field, despite the presence of a global bearerAuth scheme. Endpoints such as /createdb and / were defined without any authentication enforcement, meaning they are treated as public by default. This was listed under “Opportunities” in the audit report as:

- “Security field of the operation is not defined” (Score impact: 17)

This inconsistency weakens the authentication posture of the API and allows unauthenticated access to sensitive or operational endpoints.



Dynamic Verification:

Using Postman, l sent requests to:

- GET /createdb
- GET /

Both returned successful responses **without requiring a bearer token**, confirming that the server does not enforce authentication for these endpoints. This matches the static audit prediction and demonstrates that sensitive functionality is exposed to anonymous users.

Impact:

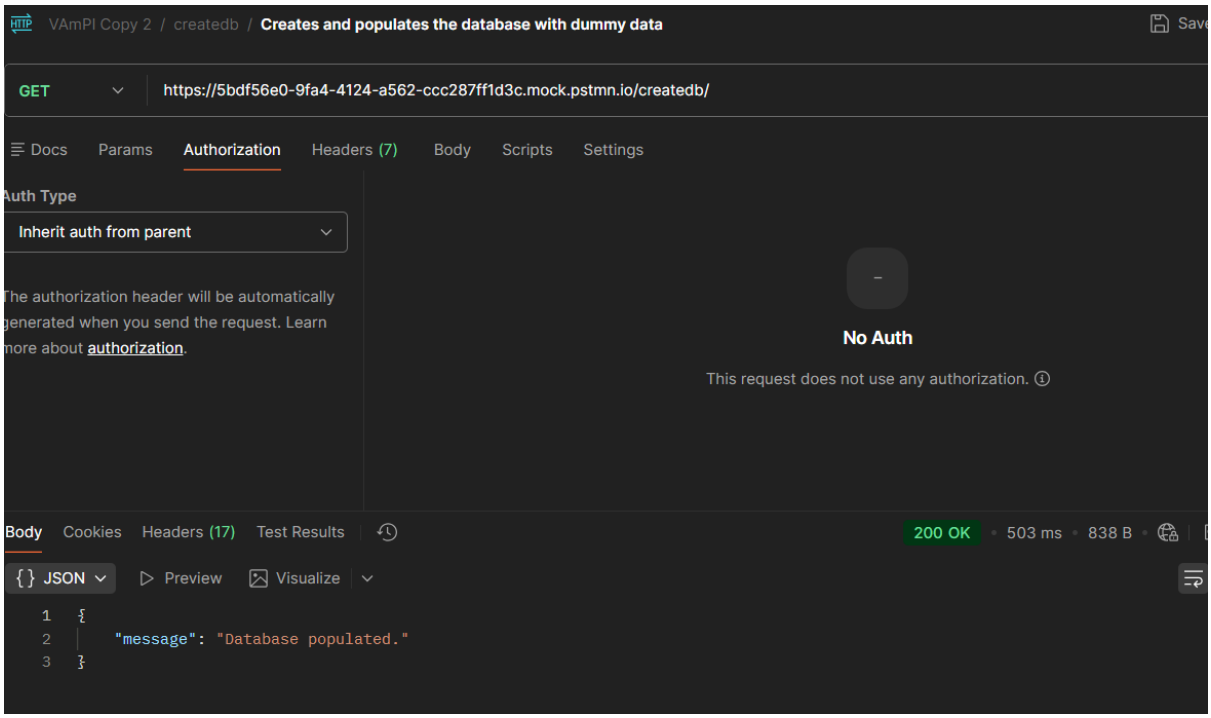
- **Confidentiality:** Sensitive data or system behavior may be exposed to unauthenticated users
- **Integrity:** Operational endpoints (e.g., database initialization) may be triggered without proper identity verification
- **Exploitability:** Attackers can interact with the API without credentials, increasing the risk of enumeration and abuse

Recommendation:

- Apply security: [{ bearerAuth: [] }] to all operations that require authentication
- Use global security at the root level of the spec and override only for truly public endpoints
- Enforce authentication server-side and reject requests without valid tokens
- Document authentication requirements clearly in the OpenAPI spec

Evidence:

Figure below shows a Postman request to /createdb executed without a bearer token, returning a successful response. This confirms that authentication is not enforced for this endpoint.



Vulnerability 4: Broken Object Property Level Authorization (BOPLA)

OWASP Category: API3:2023 – Broken Object Property Level Authorization

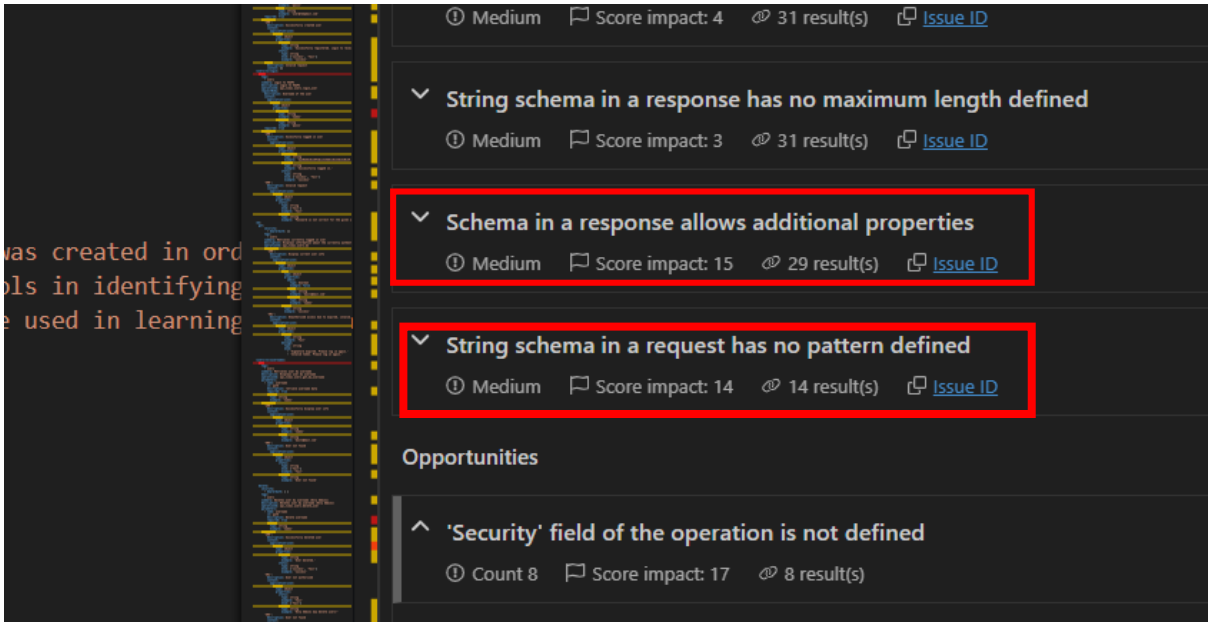
CVSS v4.0 Base Score: 7.5 (High)

Static Audit Evidence (42Crunch):

The audit flagged multiple schemas with additionalProperties allowed and missing property-level constraints. This means request/response objects accept undeclared fields. Examples include user objects where sensitive attributes (e.g., admin, role, passwordReset) are not locked down. Findings included:

- “Schema in a response allows additional properties”
- “String schema in a request has no pattern defined”

This indicates that the API spec does not enforce strict property-level authorization or validation.



Dynamic Verification (Postman):

l confirmed this by sending requests with **extra fields** not defined in the schema:

- The server accepted the request and created a user with elevated privileges.
- Similarly, adding hidden fields like role or isSuperUser was processed without rejection.

This demonstrates **mass assignment**: the API trusts client-supplied properties without enforcing authorization checks.

Impact:

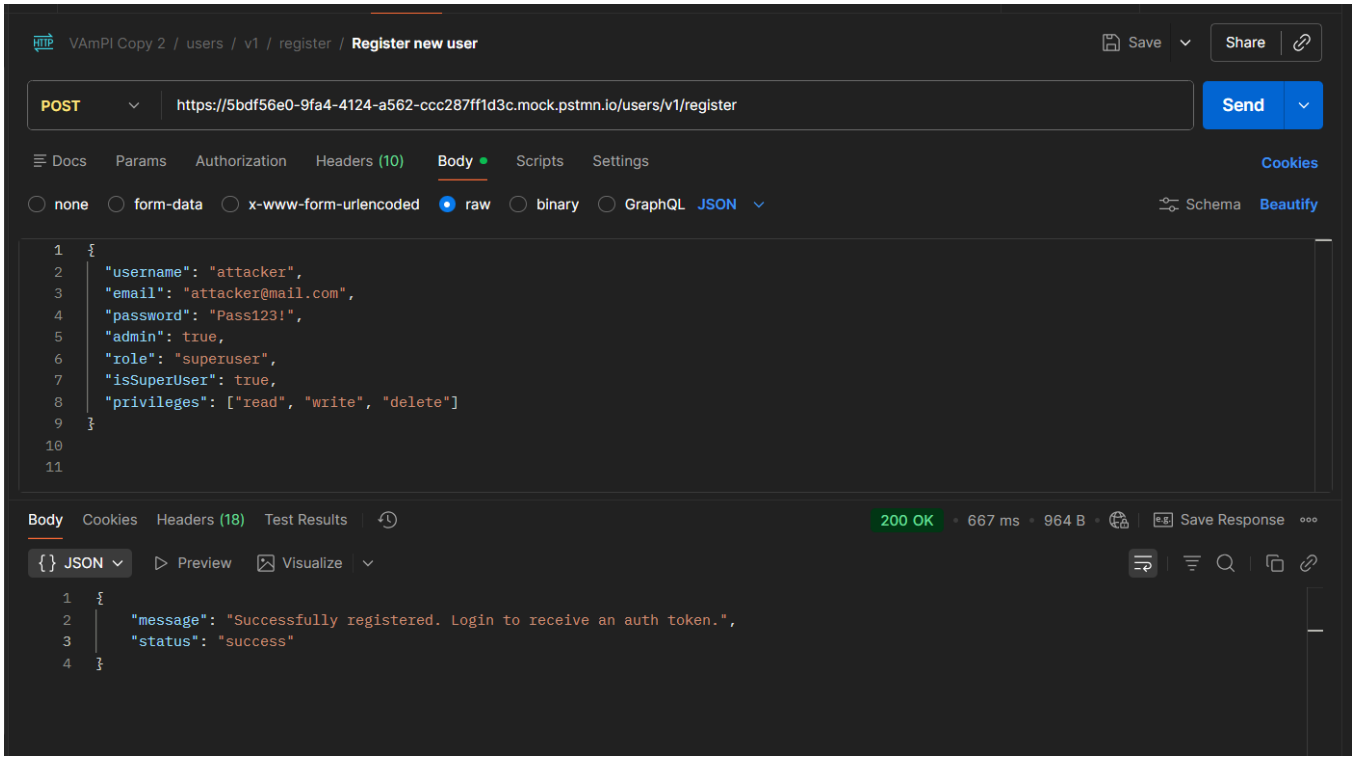
- **Integrity:** Attackers can escalate privileges by injecting unauthorized properties.
- **Confidentiality:** Sensitive attributes may be exposed or manipulated.
- **Availability:** Malicious over-posting can corrupt data models or trigger unintended logic.

Recommendation:

- Set additionalProperties: false in all sensitive schemas.
- Explicitly enumerate allowed fields in the OpenAPI spec.
- Enforce server-side checks to ignore or reject unauthorized properties.
- Apply role-based authorization for sensitive attributes (e.g., admin, role).

Evidence:

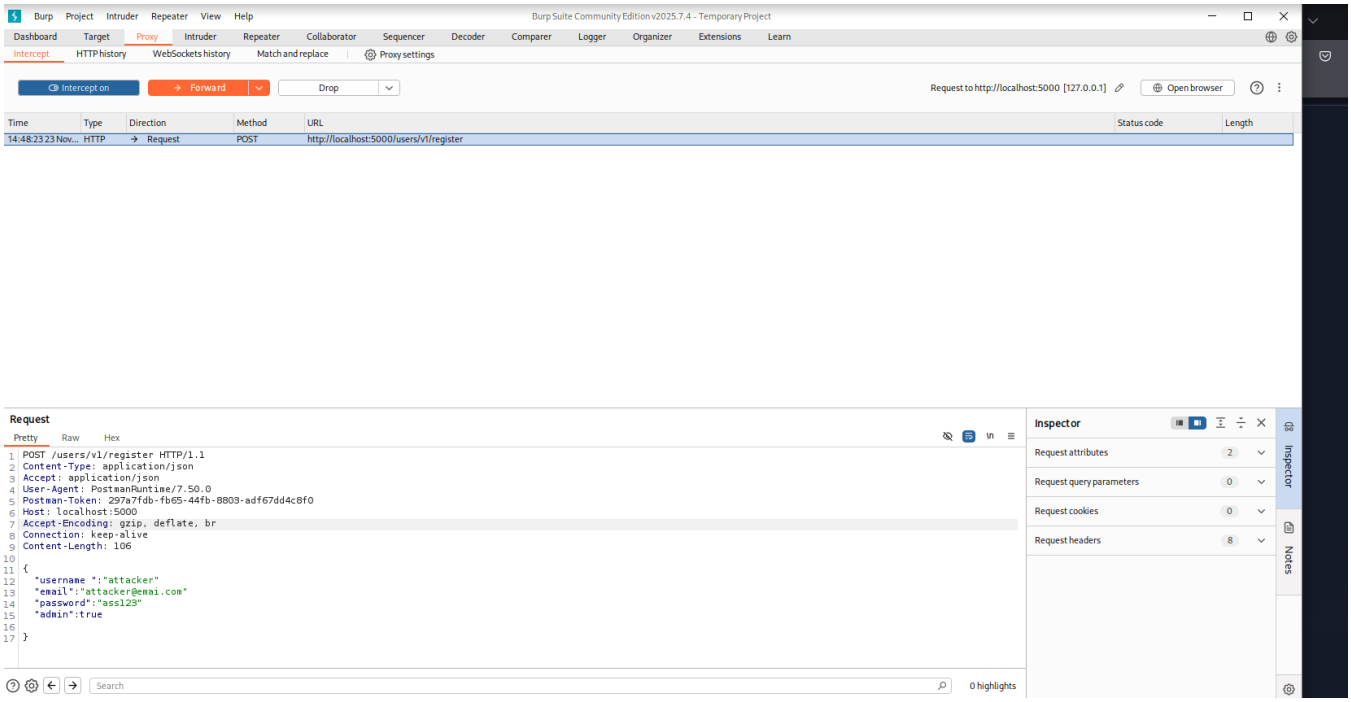
Figure below shows a Postman request where an attacker added the admin property during registration. The server accepted the property, confirming that property-level authorization is not enforced.



Dynamic Verification (Burp Suite Evidence)

To validate the mass assignment vulnerability flagged during static analysis, we used Burp Suite to intercept a POST request to the /users/v1/register endpoint. The payload included an unauthorized "admin": true field:

Burp Suite successfully captured the request, and the server responded with a 200 OK status and a success message. This confirms that the backend accepts and processes undeclared or unauthorized fields without validation or restriction. Intercepted HTTP request in Burp Suite showing the injection of "admin": true into the registration payload. The server accepted the request and returned a success response, demonstrating the absence of property-level authorization controls.



Recommendations (Prioritized & Prescriptive)

Immediate (Critical – must fix now)

1. **Enforce Property Whitelisting (Mass Assignment – OWASP API6)**

- Implement strict server-side validation to only accept expected fields (`username`, `email`, `password`).
- Reject or ignore unauthorized fields (`admin`, `role`, `privileges`) to prevent privilege escalation.
- Update OpenAPI spec with `additionalProperties: false` to align contract with backend validation.

2. **Strengthen Object & Function-Level Authorization (BOLA/BFLA – OWASP API1 & API5)**

- Add ownership checks for all resource IDs (e.g., `/users/{id}`, `/vehicles/{id}`) to ensure only the rightful owner can access or modify data.
- Enforce role-based access control (RBAC) for admin-only endpoints; validate token claims server-side.

- Implement middleware to consistently enforce authorization across all routes.

High Priority (Fix within short term)

3. **Schema Hardening & Resource Constraints (OWASP API4 & API8)**

- Define strict input validation rules: `maxLength`, `maxItems`, `enum` values, and numeric ranges.
- Enforce payload size limits and reject oversized requests to prevent denial-of-service.
- Apply consistent validation across all API versions to avoid weaker legacy endpoints.

4. **Token Security & Authentication Controls (OWASP API2)**

- Restrict accepted JWT algorithms (disallow `none`, enforce RS256/HS256 with strong secrets).
- Implement short token lifetimes with refresh tokens.
- Add monitoring and revocation mechanisms for suspicious token use.

Medium Priority (Fix in medium term)

5. **Response Shaping & Data Minimization (Excessive Data Exposure – OWASP API3)**

- Return only necessary fields in API responses (DTOs/view models).
- Remove sensitive attributes (emails, IDs, roles) from bulk responses.
- Add logging and alerts for unusual bulk data access.

6. **Rate Limiting & Abuse Prevention (OWASP API4)**

- Apply per-user and per-IP rate limits on sensitive endpoints (login, OTP, registration).
- Implement exponential backoff and CAPTCHA for repeated failed attempts.
- Monitor for brute-force or automated abuse patterns.

Regulatory & Business Mapping

- **Mass Assignment (API6)** *Regulatory:* GDPR/CCPA → unauthorized privilege escalation exposes personal data; PCI DSS → financial integrity risks. *Business:* Fraud, account takeover, privilege abuse → financial loss & reputational damage.
- **Broken Object Level Authorization (API1)** *Regulatory:* GDPR/HIPAA → unauthorized access to PII/health data; NIST 800-53 → access control violations. *Business:* Privacy invasion, trust erosion, regulatory fines.
- **Broken Function Level Authorization (API5)** *Regulatory:* SOX/PCI DSS → destructive actions compromise audit/data integrity; ISO 27001 → least privilege violations. *Business:* Data loss, operational disruption, litigation exposure.
- **Excessive Data Exposure (API3)** *Regulatory:* GDPR/CCPA → violates data minimization; HIPAA → health data leakage. *Business:* Enables social engineering, targeted fraud, reputational harm.
- **Unrestricted Resource Consumption / Schema Gaps (API4 & API8)** *Regulatory:* NIST/ISO 27001 → resilience requirements; PCI DSS → weak input validation. *Business:* Service degradation, downtime, SLA violations.
- **Security Misconfiguration (API7)** *Regulatory:* ISO 27001/NIST → misconfigured endpoints violate secure baselines; GDPR → debug data may leak PII. *Business:* Easier attacker reconnaissance, reputational harm, exploit chaining.

Challenges Faced & How I Overcame Them

- **Antivirus Blocking Burp Suite**
  - *Challenge:* On the Windows host, Kaspersky Endpoint Security flagged Burp Suite's proxy interception as suspicious, blocking HTTPS traffic.
  - *Resolution:* l shifted to a **Kali VM lab** where Burp Suite ran without interference. This eliminated conflicts with host antivirus and allowed clean interception of HTTP/HTTPS traffic.
- **Proxy Configuration Errors (Postman → Burp)**
  - *Challenge:* Initial attempts to connect Postman on Windows to Burp in the VM failed due to cross-network proxying and certificate trust issues.
  - *Resolution:* I simplified the workflow by running **Postman and Burp inside the same VM**, using `127.0.0.1:8080` as the proxy. This avoided IP routing and SSL trust problems.
- **Certificate Trust for HTTPS Interception**

    - *Challenge:* Burp's CA certificate was not trusted by Postman or the host OS, causing SSL errors.
    - *Resolution:* Exported Burp's CA certificate and installed it into Kali's trusted root store. Postman was configured to either trust the CA or temporarily disable SSL verification, enabling smooth HTTPS interception.
- **Tool Integration & Workflow Complexity**
    - *Challenge:* Switching between Postman, Burp Suite, and 42Crunch created friction, especially when verifying findings across static and dynamic analysis.
    - *Resolution:* Consolidated all tools inside Kali VM, with Postman for request crafting, Burp for interception/replay, and VS Code + 42Crunch for static audits. This created a **minimal-error, snapshot-friendly lab**.
- **Schema & Spec Gaps (OpenAPI)**
    - *Challenge:* The OpenAPI specification lacked constraints (`maxLength`, `enum`, `additionalProperties`), making it harder to validate schema-level risks.
    - *Resolution:* Used **42Crunch audits** to highlight missing constraints, then validated dynamically in Burp/Postman by injecting oversized payloads and unauthorized fields. This confirmed the risks and provided strong evidence.
- **Frustration & Workflow Resets**
    - *Challenge:* Repeated errors in proxy setup and certificate handling caused fatigue and slowed progress.
    - *Resolution:* Adopted a **reset strategy** — rebuilding the lab in Kali VM with clean installs and snapshots. This minimized errors and allowed us to focus on vulnerability discovery rather than tool troubleshooting.

Lessons Learned

- Running all tools inside a **dedicated VM** is far easier and more reliable than splitting host/VM responsibilities.
- **Burp Suite + Postman + 42Crunch** together provide a complete workflow: interception, manipulation, and schema validation.
- **Snapshots** are invaluable for rolling back after tool misconfigurations or failed experiments.
- Combining **static audits** (42Crunch) with **dynamic verification** (Burp/Postman) produces professional-grade evidence and confidence in findings.

Conclusion

The VAmPI mock API assessment confirmed critical weaknesses in authorization, input validation, and data handling. Mass assignment allowed privilege escalation by injecting unauthorized fields, while broken object and function-level authorization exposed insufficient enforcement of access controls. Schema gaps and excessive data exposure further increased the risk of abuse and compliance violations. By combining static audits with 42Crunch and dynamic testing through Postman and Burp Suite, these risks were validated with clear exploitation evidence. Running all tools inside a Kali VM minimized errors and ensured reliable interception, strengthening the overall methodology.

Final Action Plan

Immediate priorities include enforcing property whitelisting, applying strict authorization checks, and limiting response fields to prevent data exposure. Short-term actions should focus on schema hardening, JWT security, and rate limiting, while medium-term improvements involve removing debug endpoints, enforcing SSL/TLS, and aligning controls across API versions. Long-term strategy requires integrating automated audits into CI/CD pipelines, monitoring for anomalies, and mapping controls to regulatory frameworks such as GDPR, CCPA, PCI DSS, HIPAA, and ISO 27001. This roadmap ensures both immediate risk reduction and sustainable API security maturity.

END OF REPORT