**DEMONSTRATION OF THE RISKS OF WEAK API AUTHENTICATION AND PROPOSED SOLUTIONS**

Project submitted to the SRM University – AP, Andhra Pradesh

for the partial fulfillment of the requirements to award the degree of

**Bachelor of Technology**

In

**Computer Science and Engineering School of Engineering and Sciences**

Submitted by

**K. Sarada Reddy AP22110010389**



Under the Guidance of

**Bhaskara Santhosh Egala,**

**Assistant professor, Dept.of CSE**

**SRM University–AP Neerukonda, Mangalagiri, Guntur**

**Andhra Pradesh – 522 240**

**April, 2025**

**1. Introduction**

In today’s digital age, APIs (Application Programming Interfaces) serve as the backbone of web and mobile applications. They enable different software systems to communicate and share data. However, the increasing reliance on APIs has made them a frequent target for attackers, especially when authentication mechanisms are weak or misconfigured.

This project focuses on the security risks associated with weak API authentication and provides practical solutions using modern security techniques to protect APIs from unauthorized access and data breaches.

**2. Importance of API Security**

* **Core Component**: APIs are essential for seamless integration between services and systems.
* **Prime Targets**: Attackers often exploit APIs due to poor authentication configurations.
* **Security Risks**: Weak authentication can lead to severe vulnerabilities including data theft, impersonation, and unauthorized access.

**3. Types of API Authentication Methods**

| **Method** | **Description** | **Security Level** |
| --- | --- | --- |
| API Key | Static token passed in headers; easy to steal | Low |
| Basic Auth | Uses base64 encoded credentials; insecure | Low |
| OAuth 2.0 | Token-based and widely used by Google, Facebook etc. | High |
| JWT | Digitally signed, self-contained tokens | High |

Each method has pros and cons, and choosing the right one depends on the application's sensitivity and security requirements.

**4. Problem Identification: Widespread API Vulnerabilities**

* **Lack of Proper Authentication**: Many public APIs still use outdated or no authentication mechanisms.
* **Susceptibility to Attacks**: Such APIs are exposed to impersonation, data manipulation, service abuse, and even financial theft.

These vulnerabilities can compromise not just user data, but also the credibility and security posture of the organization.

**5. Project Objectives**

1. **Demonstrate Exploitation**: Showcase real-world scenarios where weak API authentication can be abused.
2. **Implement Authentication**: Use simple authentication mechanisms such as API keys or token verification in a sample API.
3. **Highlight Improvements**: Evaluate and compare the system’s behavior before and after implementing authentication.

**6. Tools and Environment**

* **Programming Language**: Python
* **Web Framework**: Flask
* **API Testing Tool**: Postman
* **Editor**: Any text editor (e.g., VS Code)

A Python file (api\_demo.py) was created to simulate the API behavior with both public and protected endpoints.

**7. Observations: Weak API Authentication in Action**

* **Unauthorized Access**: APIs without authentication allow intruders to access private data.
* **Data Breaches**: Exposing user data can violate privacy laws.
* **Account Takeovers**: Attackers may hijack sessions or perform actions as legitimate users.
* **Other Risks**:
  + **Replay Attacks**: Reusing previous requests to exploit the system.
  + **Token Theft**: Stolen tokens can be reused by attackers.

**8. Solutions for Enhancing API Security**

| **Solution** | **Description** |
| --- | --- |
| OAuth 2.0 / JWT | Implement strong, token-based authentication |
| HTTPS | Always secure data transport with encryption |
| Token Expiry | Limit token lifespan to reduce risk of misuse |
| Flask-Limiter | Apply rate limiting to prevent brute-force attacks |
| Input Validation | Ensure all incoming data is properly sanitized |
| Monitoring & Logging | Maintain logs for auditing and anomaly detection |

**9. Code**

**from flask import Flask, request, jsonify**

**app = Flask(\_name\_)**

**AUTH\_TOKEN = "securetoken123" # basic hardcoded token**

**@app.route('/public-data')**

**def public\_data():**

**return jsonify({"user": "guest", "data": "This is public info"})**

**@app.route('/secure-data')**

**def secure\_data():**

**token = request.headers.get('Authorization')**

**if token != AUTH\_TOKEN:**

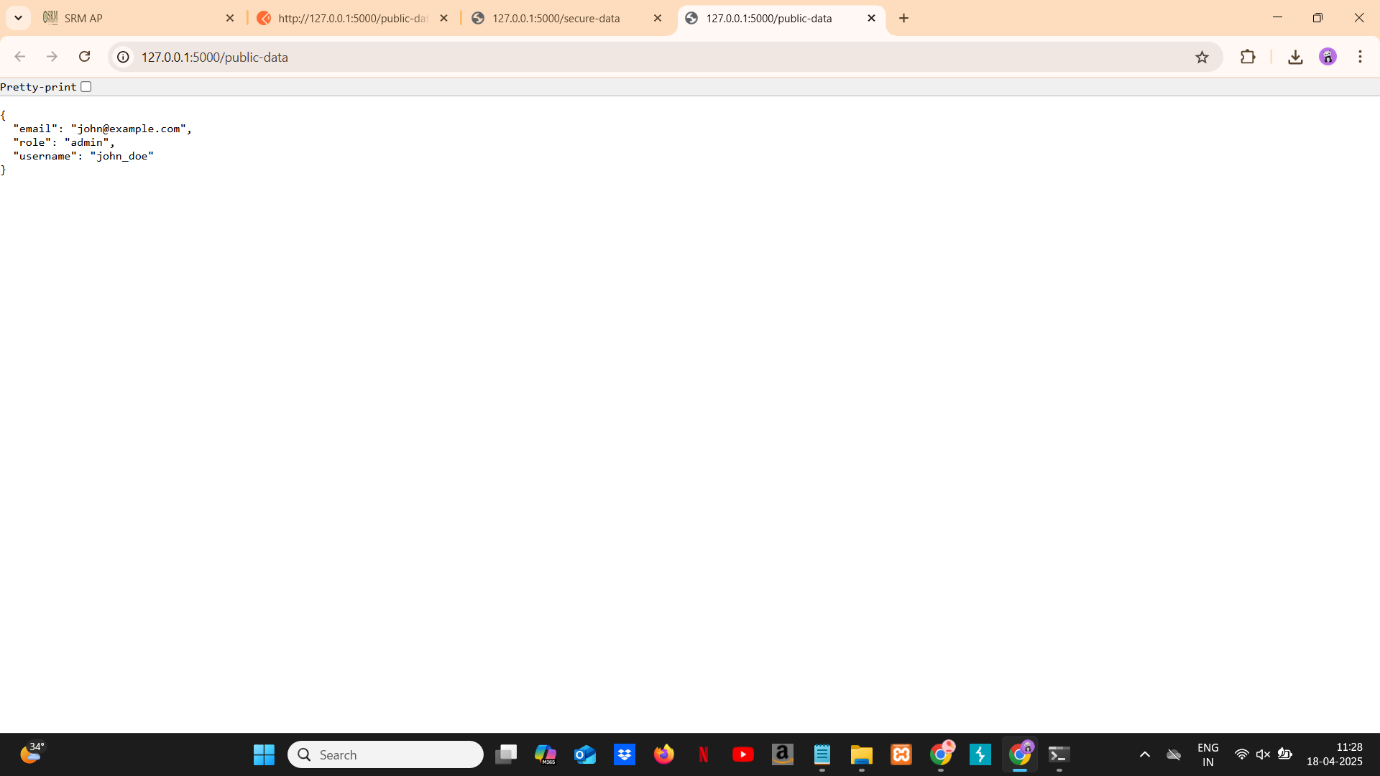
**return jsonify({"error": "Unauthorized"}), 401**

**return jsonify({"user": "admin", "data": "This is secure info"})**

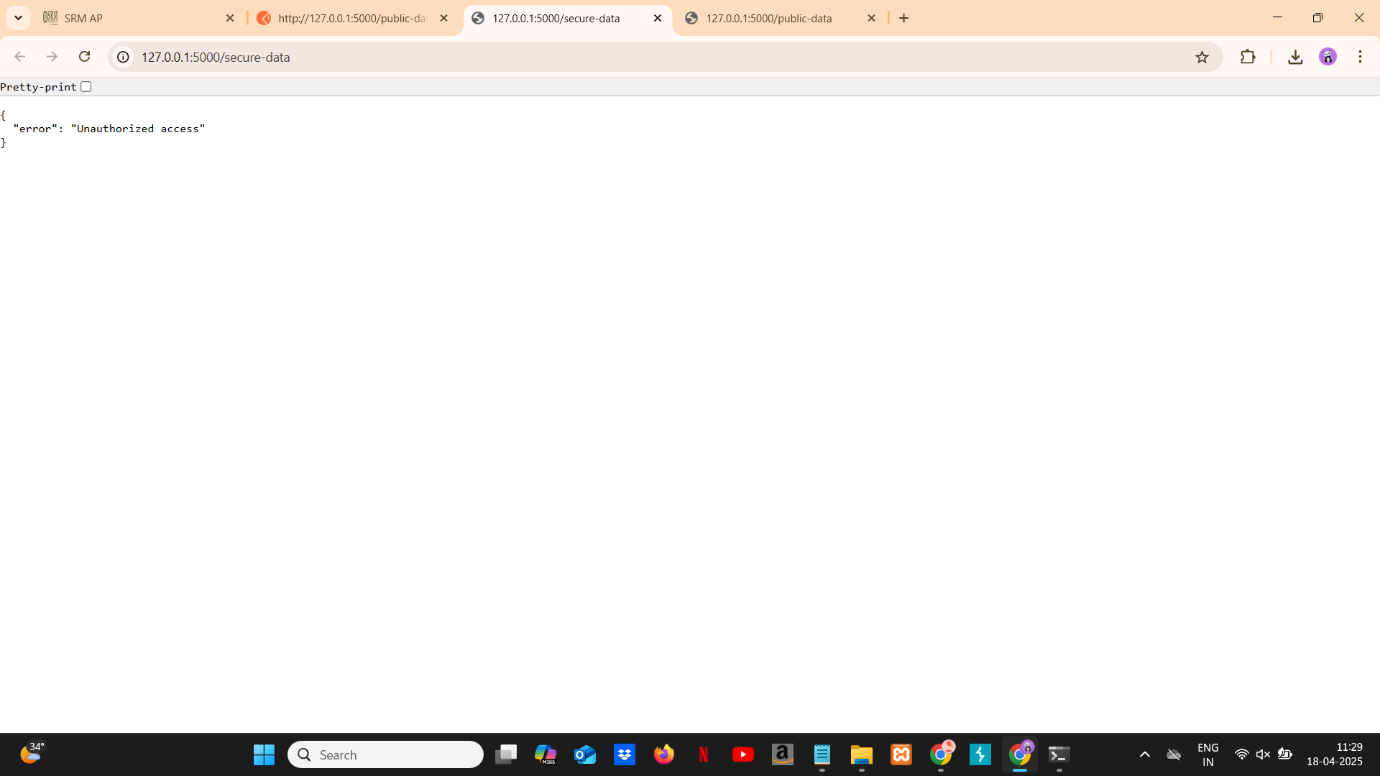
**if \_name\_ == '\_main\_':**

**app.run(debug=True)**

**PUBLIC ROUTE:**



**SECURE ROUTE:**



**10. Conclusion**

Securing APIs is crucial to maintaining the confidentiality, integrity, and availability of digital services. This project demonstrates how weak API authentication opens the door for security threats and proposes practical, effective solutions using Python and Flask.

**Key Takeaways**:

* Always implement strong authentication and secure transport.
* Regularly rotate and expire tokens.
* Use tools like Flask-Limiter for rate-limiting.
* Monitor and validate API activity continuously.