Introduction

Overview

APIs are crucial for enabling integration and interaction between systems, but they pose significant security risks. Protecting APIs is essential to safeguard sensitive data, prevent unauthorized access, and ensure system integrity. Key security practices include robust authentication, input validation, encryption, and regular security testing. Implementing these measures helps maintain the reliability and trustworthiness of APIs while complying with regulatory standards.

This document is a web security documentation on an API with user sign up, login and logout functionalities and its security assessment.

To ensure maximum security, Semgrep tool was used for identifying security vulnerabilities, vulnerabilities were grouped according to the OWASP Top 10 vulnerabilities and security measures were taking to remediate vulnerabilities.

API Security Assessment

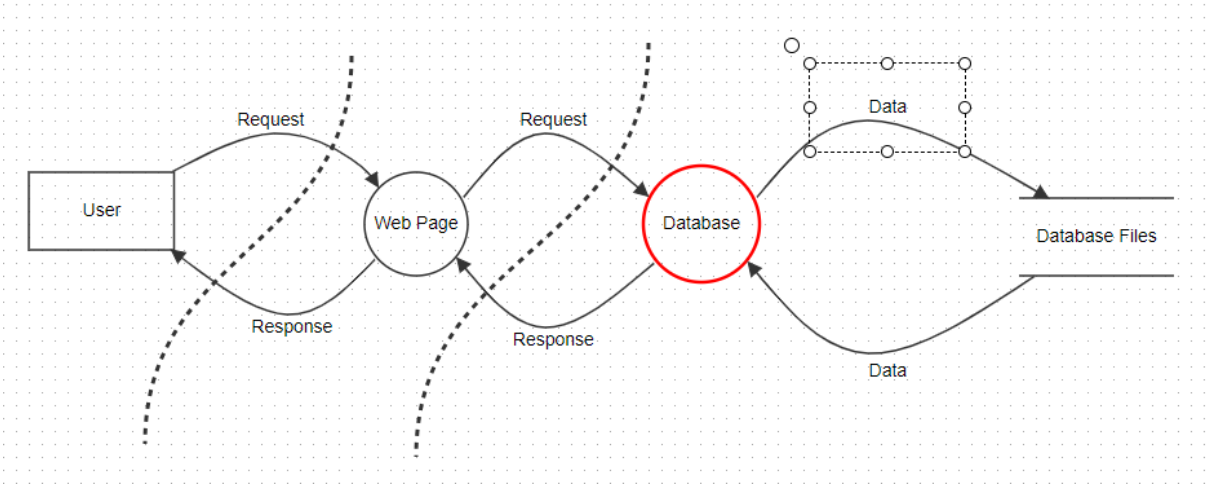

Figure 1 Level 1 data flow diagram of API

1. Semgrep Detected Vulnerabilities and Remediation

1. User Model

```
@Entity
@Data
@AllArgsConstructor
@NoArgsConstructor
public class User {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;
    private String username;
    private String password;
    private String role;
```

Figure 2 Code Snippet of User Entity

- **Security Concern**: Storing passwords in plain text is a critical security vulnerability.
- **Remediation**: Use a `PasswordEncoder` (like BCryptPasswordEncoder) to securely hash passwords before storing them in the database. Below is the code snippet that handles the password hashing.

```
public User saveUser(User user) {
    user.setPassword(passwordEncoder.encode(user.getPassword()));
    return userRepository.save(user);
}
```

Figure 3 Code Snippet of function that handle password Hashing

Security Analysis with STRIDE

The table below sites examples of Spoofing, Tampering, Repudiation, Information Disclosure, Denial of Service and Elevation of Privilege sited in the project and how it was addressed. This table was derived based on the data flow diagram of the API.

User (Actor)

A user trying to login.

Number	Title	Type	Priority	Status	Score	Description	Mitigations
1	Spoofing(User->webpage)	Spoofing	Medium	Mitigated		Issue: Username Enumeration: Attackers might be able to discover valid usernames by testing different login credentials against the web page	Implement CAPTCHA and Rate Limiting: Protect against automated login attempts and brute-force attacks by limiting the number of login attempts and using CAPTCHA to distinguish between human users and bots. Error Messages: Ensure error messages do not reveal whether a username or password is incorrect.
2	Repudiation(User->Webpage)	Repudiation	Medium	Mitigated		Issue: Unaccountable Actions: Users might perform actions (e.g., login attempts or data modification) and later deny having done so.	Logging and Monitoring: Implement detailed logging of user actions, especially around authentication and data access. Ensure logs are tamper-proof and monitored regularly. Audit Trails: Maintain audit trails that capture critical events such as login attempts, data changes, and access requests.

Web Page (Process)

Number	Title	Type	Priority	Status	Score	Description	Mitigations
3	Information Disclosure(Webpage->User)	Information disclosure	Medium	Mitigated		Sensitive Data Exposure: Sensitive data might be exposed in responses from the web page, such as through error messages or data leaks.	Error Message Management: Ensure error messages do not disclose sensitive information. Provide generic messages that do not indicate the specific nature of any error. Data Encryption: Encrypt sensitive data both in transit and at rest to protect against unauthorized access.
4	Elevation of Privilege	Elevation of privilege	Medium	Mitigated		Unauthorized Access: Attackers might exploit vulnerabilities in the web page to gain elevated privileges and access the database.	Role-Based Access Control (RBAC): Implement RBAC to ensure that users only have access to the data and actions they are authorized for. Secure Authentication: Use strong authentication mechanisms (e.g., multi-factor authentication) and enforce strong password policies.
5	Tampering	Tampering	Medium	Mitigated		SQL Injection: If inputs from the web page are not properly sanitized, attackers might be able to inject SQL commands, leading to unauthorized data manipulation. Session Hijacking: Attackers could intercept session tokens between the web page and the database.	Input Validation: Validate and sanitize all inputs to prevent SQL injection attacks. Use HTTPS: Encrypt data transmission between the web page and the database to protect against session hijacking. Secure Session Management: Implement secure session tokens with appropriate expiration and use HttpOnly, Secure, and SameSite cookie attributes.
10	Denial of Service	Denial of service	Medium	Mitigated		Service Overload: Attackers might overwhelm the web page with excessive requests, causing the service to become unavailable.	Rate Limiting: Implement rate limiting to control the number of requests a user can make in a given time period. Load Balancing: Use load balancing to distribute incoming requests across multiple servers, reducing the impact of DoS attacks. CAPTCHA: Use CAPTCHA to mitigate automated attack attempts.

Database (Process)

Number	Title	Type	Priority	Status	Score	Description	Mitigations
6	Tampering(Datahase->Database Files)	Tampering	Medium	Mitigated		SQL Injection: If inputs from the web page are not properly sanitized, attackers might be able to inject SQL commands, leading to unauthorized data manipulation. Session Hijacking: Attackers could intercept session tokens between the web page and the database.	Access Control: Implement strict access control mechanisms to ensure that only authorized users and services can modify database files. Data Encryption: Encrypt sensitive data stored in database files to protect against unauthorized access.

Number	Title	Type	Priority	Status	Score	Description	Mitigations
7	Information disclosure(Database->Webpage)	Spoofing	Medium	Open		Data Leaks: Sensitive information might be leaked through the SQL query responses or database error messages.	SQL Query Management: Ensure SQL queries and responses do not expose sensitive data. Use parameterized queries to prevent data leaks. Data Masking: Implement data masking for sensitive data returned to the web page.

Key Takeaway of Web Security Fundamental Concepts:

Web security fundamentals focus on protecting web applications and their users from various threats. Key concepts include:

1. **Authentication and Authorization**: Ensuring that users are who they claim to be (authentication) and granting them access only to resources they are permitted to use (authorization).
2. **Data Protection**: Implementing encryption for data in transit and at rest to safeguard sensitive information from unauthorized access and tampering.
3. **Input Validation**: Validating and sanitizing user inputs to prevent injection attacks and other vulnerabilities.
4. **Secure Communication**: Using HTTPS to secure communication channels between users and servers, preventing eavesdropping and man-in-the-middle attacks.
5. **Error Handling and Logging**: Properly managing errors and maintaining logs to detect and respond to potential security incidents while avoiding exposure of sensitive information.