Designing secure APIs

Laurenz van de Poll

Software Semester 3

Contents

[Abstract 3](#_Toc132922153)

[Introduction 4](#_Toc132922154)

[Research Question 4](#_Toc132922155)

[Background/Literature Review 5](#_Toc132922156)

[What are the best practices for designing secure APIs for web applications? 5](#_Toc132922157)

[How can access control be implemented for different user roles (e.g., owner, service provider) in the API? 6](#_Toc132922158)

[What measures can be taken to protect against common API security threats, such as SQL injection, cross-site scripting, and CSRF attacks? 6](#_Toc132922159)

[How can rate limiting and request validation be used to mitigate API abuse and ensure service availability? 6](#_Toc132922160)

[How can input validation and output encoding be implemented in the API to prevent injection attacks and data leaks? 6](#_Toc132922161)

[How can API versioning be managed to maintain backward compatibility and security updates without disrupting the user experience? 6](#_Toc132922162)

[Research Methodology 6](#_Toc132922163)

[Results 6](#_Toc132922164)

[Discussion 6](#_Toc132922165)

[Conclusion 7](#_Toc132922166)

[References 7](#_Toc132922167)

[Appendices (if applicable) 7](#_Toc132922168)

## Abstract

A brief summary of your research, including its purpose, methodology, and main findings.

## Introduction

V Introduce the topic of secure API design and implementation in web applications.

V Present the main research question and sub-questions.

V Explain the relevance of the research to your project or the course

Provide a brief overview of the report structure

In recent years, web applications have become an integral part of our daily lives, offering a wide range of services and functionality. One of the critical components of modern web applications is the Application Programming Interface (API), which serves as an intermediary between different software applications, enabling them to communicate with each other. As web applications become increasingly interconnected and handle sensitive user data, the importance of secure API design and implementation has grown significantly.

Secure API design and implementation involves creating APIs that are resilient to potential security threats and vulnerabilities while ensuring data privacy and integrity. By adhering to best practices and applying robust security measures, developers can reduce the risk of unauthorized access, data breaches, and other security incidents that may compromise the functionality and user trust in web applications. In the context of a car maintenance tracking web application, secure API design and implementation is crucial to protect users personal information, vehicle data, and maintenance records.

This research report aims to explore the topic of secure API design and implementation in web applications, focusing on the challenges and best practices associated with creating secure APIs for a car maintenance tracking web application. By addressing the main research question and its sub-questions, this report seeks to provide valuable insights and recommendations for implementing secure APIs in the context of the car maintenance tracking project.

### Research Question

How can secure API design and implementation be achieved in a car maintenance tracking web application?

#### Sub-questions

1. What are the best practices for designing secure APIs for web applications?
2. How can access control be implemented for different user roles (e.g., owner, service provider) in the API?
3. What measures can be taken to protect against common API security threats, such as SQL injection, cross-site scripting, and CSRF attacks?
4. How can rate limiting and request validation be used to mitigate API abuse and ensure service availability?
5. How can input validation and output encoding be implemented in the API to prevent injection attacks and data leaks?
6. How can API versioning be managed to maintain backward compatibility and security updates without disrupting the user experience?

## Background/Literature Review

Review relevant literature and theories

Identify gaps in knowledge and how your research addresses them

The purpose of this literature review is to explore existing knowledge and best practices in secure API design and implementation for web applications, specifically in the context of a car maintenance tracking web application. This review will cover key aspects of API security, including best practices for designing secure APIs, implementing access control for different user roles, addressing common security threats, rate limiting and request validation, input validation and output encoding, and managing API versioning. By examining the relevant literature, this review aims to provide a comprehensive understanding of secure API design and implementation, which will serve as a foundation for the subsequent sections of this research report.

### What are the best practices for designing secure APIs for web applications?

IDs: 3, 5, 9, 11, 12, 13

#### Security Principles

Secure API design should follow these key principles:

* Least Privilege
* Fail-safe Defaults
* Economy of Mechanism
* Complete Mediation
* Open Design
* Separation of Privilege
* Least Common Mechanism
* Psychological Acceptability

#### The CIA Triad

Benchmarking information systems security involves three key factors: Confidentiality, Integrity, and Availability.

* Confidentiality: Protect data from unintended recipients during storage and transit using encryption (e.g., TLS, HTTPS, message-level encryption).
* Integrity: Ensure data correctness and trustworthiness, as well as detect unauthorized modifications. Preventive and detective measures should be employed for data in transit and at rest.
* Availability: Ensure that the system is always accessible to legitimate users, protecting it from illegal access attempts and potential attacks.

#### Authentication and Authorization

Control access to the API through proper authentication and authorization mechanisms.

* Authentication: Verify the identity of a user or application, either through direct or brokered authentication.
* Authorization: Validate the actions an authenticated user or application can perform within the system boundary. Implement Role-Based Access Control (RBAC) or Attribute-Based Access Control (ABAC) to define the required authorization levels.

#### Additional Best Practices for Secure API Design

* Max Retry and Jail Safety Mechanism
* Strict Input Validation
* Avoid Auto-Incrementing IDs
* Protect Sensitive Endpoints
* Limit Requests
* Encrypt Everything
* Remove Components with Vulnerabilities
* Add Timestamps in Requests
* Block Large Requests
* IP-Based Throttling
* Rate Limit Throttling

### How can access control be implemented for different user roles (e.g., owner, service provider) in the API?

IDs: 5, 6, 12, 13

In this section of the literature review we present strategies and best practices for implementing access control in APIs.

#### Access control models

* Role-Based Access Control (RBAC): Assign users to roles, and manage permissions at the role level.
* Attribute-Based Access Control (ABAC): Define access policies based on user attributes, resource attributes, and environmental factors.

#### Function level authorization

* Implement a permission-based approach to restrict access to specific operations.
* Ensure code is easy to audit and understand for maintaining robust authorization policies.

#### Centralizing authorization policies

* Utilize a policy decision point, such as Open Policy Agent, to manage complex authorization rules.
* Address broken object level authorization by simplifying endpoint code, improving maintainability, and facilitating policy auditing.

#### Authentication and tokens

* Employ OAuth tokens for authentication in APIs.
* Choose between reference tokens (immediately revocable, may impact performance) and self-contained tokens (JSON Web Tokens or JWT, can be verified independently) based on application requirements.

By implementing these access control models, authorization techniques, and authentication strategies, developers can effectively manage different user roles in APIs and ensure that only authorized users have access to specific functions and data. This comprehensive approach to access control provides a secure and manageable system, protecting sensitive information while granting appropriate access to users.

### What measures can be taken to protect against common API security threats, such as SQL injection, cross-site scripting, and CSRF attacks?

IDs: 3, 4, 5, 7, 13

In this section, we explore the various measures that can be implemented to safeguard against prevalent API security threats, including SQL injection, cross-site scripting (XSS), and Cross-Site Request Forgery (CSRF) attacks. The discussion outlines best practices for creating secure APIs in web applications.

#### Protecting against CSRF attacks

* Use short-lived authorization codes to limit the attacker's window of opportunity.
* Implement state parameters in OAuth 2.0 to validate requests and prevent unauthorized access.
* Employ PKCE to secure the exchange of authorization codes and access tokens.

#### Mitigating XSS attacks

* Encrypt API key authentication to prevent unauthorized access to APIs.
* Convert script code execution requests into plain text, preventing execution in the browser.
* Validate and scrub user input of HTML tags, JavaScript tags, and SQL statements to avoid code injection.

#### Preventing SQL injection attacks

* Validate user input data and restrict parameter values to a whitelist of expected values.
* Use prepared statements with bind variables to prevent unauthorized SQL commands.
* Implement schema validation to avoid manipulation of SQL schema elements.

#### CSRF defense strategies

* Implement CSRF tokens to validate requests originating from legitimate sources.
* Utilize the double submit cookie strategy to compare hidden field values with stored cookies.
* Check request origins to ensure they come from authorized clients.
* Leverage the SameSite cookie property to restrict cookie usage to the same domain.

In conclusion, by employing a combination of these best practices and strategies, developers can effectively protect their web applications from common API security threats such as CSRF, XSS, and SQL injection attacks. This comprehensive approach ensures a more secure environment for web applications and their users.

### How can rate limiting and request validation be used to mitigate API abuse and ensure service availability?

IDs: 5, 11, 13

### How can input validation and output encoding be implemented in the API to prevent injection attacks and data leaks?

IDs: 12, 13

### How can API versioning be managed to maintain backward compatibility and security updates without disrupting the user experience?

IDs: 9

## Research Methodology

Describe the research design (e.g., quantitative, qualitative, mixed-methods)

Explain the data collection methods and sample selection

Discuss the data analysis techniques

## Results

Present the findings of your research

Use appropriate visuals (e.g., tables, figures, charts) to support your results

## Discussion

Interpret the results and relate them to the research question

Discuss the implications of your findings for your project or the course

Address any limitations of your research and suggest future research directions

## Conclusion

Summarize the main findings of your research

Restate the significance of your research for your project or the course

## References

List all the sources cited in your research report, following a specific citation style (e.g., APA, MLA, or Chicago)

## Appendices (if applicable)

Include any additional information or materials that support your research but are not essential for understanding the main report (e.g., interview transcripts, questionnaires)