



API Security Lab

Bachelor Thesis
Documentation

Department of Computer Science
OST – University of Applied Sciences
Campus Rapperswil-Jona

Spring Term 2024

Authors Corsin Salutt
Thajakan Thirunavukkarasu
Advisor: Ivan Bütler
External Co-Examiner: Dr. Benjamin Fehrensén
Internal Co-Examiner: Prof. Dr. Frieder Loch

Contents

I	Abstract	1
	Acronyms	6
	Glossary	8
II	Technical report	9
1	Introduction	10
1.1	Initial situation	10
1.2	Project scope	10
1.3	OST Hacking-Lab	11
2	Research	12
2.1	API and history	12
2.1.1	Definition	12
2.1.2	What is an API?	12
2.1.3	Simple Object Access Protocol	12
2.1.4	Web Services Description Language	13
2.1.5	API styles	13
2.2	API Security in context	14
2.2.1	Information security	14
2.2.2	Network security	15
2.2.3	Application security	15
2.3	Identifying threats	15
2.4	Defensive mechanism	16
2.5	Token-based authentication and OAuth 2.0	17
2.5.1	Token-based authentication	17
2.5.2	OAuth2	18
2.5.3	OAuth scopes	19
2.5.4	OAuth grant types	19
2.6	OpenID Connect	21
2.7	Microservice	21

2.8	OWASP Top 10 API Security Risks 2023	22
2.8.1	Broken Object Level Authorisation	22
2.8.2	Broken Authentication	22
2.8.3	Broken Object Property Level Authorization	22
2.8.4	Unrestricted Resource Consumption	23
2.8.5	Broken Function Level Authorization	23
2.8.6	Unrestricted Access to Sensitive Business Flows	23
2.8.7	Server Side Request Forgery	23
2.8.8	Security Misconfiguration	23
2.8.9	Improper Inventory Management	24
2.8.10	Unsafe Consumption of APIs	24
3	Lab evaluation	25
3.1	Approach	25
3.2	Lab idea taxonomy	25
3.3	Lab idea collection	26
3.3.1	Lab ideas	26
3.3.2	OAuth2 vulnerabilities	30
3.4	Decision matrix	32
3.4.1	Criteria	32
3.4.2	Outcome	35
4	Proof of concept	36
4.1	Lab PoC: API enumeration and reconnaissance	36
4.1.1	Objectives	36
4.1.2	Prerequisites:	36
4.1.3	Equipment/Software	36
4.1.4	Setup steps	37
4.1.5	Successful PoC	38
4.1.6	PoC status	38
4.2	Lab PoC: Implementing logging	41
4.2.1	Objectives	42
4.2.2	Prerequisites	42
4.2.3	Equipment/Software	42
4.2.4	Setup steps	42
4.2.5	Successful PoC	43
4.2.6	PoC status	43
4.3	Lab PoC: OWASP Coraza WAF	46
4.3.1	Objectives	46
4.3.2	Prerequisites	46
4.3.3	Equipment/Software	46
4.3.4	Setup steps	46
4.3.5	Successful PoC	47
4.3.6	PoC status	47

4.4	Lab PoC: OAuth2 vulnerabilities	48
4.4.1	Objectives	48
4.4.2	Prerequisites	48
4.4.3	Equipment/Software	48
4.4.4	Setup steps	48
4.4.5	Successful PoC	49
4.4.6	PoC Status	49
4.5	Lab PoC: Implementing API rate limiting and throttling	50
4.5.1	Objectives	50
4.5.2	Prerequisites	50
4.5.3	Equipment/Software	50
4.5.4	Setup steps	51
4.5.5	Successful PoC	51
4.5.6	PoC Status	51
4.6	Lab PoC: Implementing input validation and sanitization	52
4.6.1	Objectives	52
4.6.2	Prerequisites	52
4.6.3	Equipment/Software	52
4.6.4	Setup steps	52
4.6.5	Successful PoC	53
4.6.6	PoC Status	53
5	Lab documentation	57
5.1	Lab structure	57
5.2	Lab access	58
5.3	Generic Hacking-Lab resources	58
5.3.1	Dynamic nginx Multi-Docker	58
5.3.2	Theia Web IDE	59
5.4	API Security: API enumeration and reconnaissance	60
5.4.1	Descriptive information	60
5.4.2	Lab development	61
5.4.3	Lab solution	62
5.5	API Security: Implementing logging	63
5.5.1	Descriptive information	63
5.5.2	Lab development	64
5.5.3	Lab solution	65
5.6	API Security: OWASP Coraza WAF	66
5.6.1	Descriptive information	66
5.6.2	Lab developement	67
5.6.3	Lab solution	69
5.7	API Security: OAuth2 vulnerabilities	70
5.7.1	Descriptive information	70
5.7.2	Lab developement	71

5.7.3	Lab solution	73
5.8	API Security: Rate limiting	74
5.8.1	Descriptive information	74
5.8.2	Lab developement	75
5.8.3	Lab solution	78
5.9	API Security: Input validation and sanitization	79
5.9.1	Descriptive information	79
5.9.2	Lab development	80
5.9.3	Lab solution	81
6	Quality Measures	82
6.1	General lab requirements	82
6.2	Test concept	82
6.2.1	Roles and responsibilities	83
6.3	Defined test cases	84
6.3.1	Test cases: API enumeration and reconnaissance	84
6.3.2	Test cases: Implementing logging	85
6.3.3	Test cases: OWASP Coraza WAF	86
6.3.4	Test cases: OAuth2 vulnerabilities	88
6.3.5	Test cases: Rate limiting	90
6.3.6	Test cases: Input validation and sanitisation	92
6.4	Public testing	94
6.4.1	Tools	94
6.4.2	Questions	95
6.4.3	Overall feedback	95
6.4.4	Feedback improvements	96
7	Results	98
8	Conclusion and outlook	99
8.1	Conclusion	99
8.2	Outlook	99

Part I
Abstract

Abstract

APIs (Application Programming Interfaces) are integral to modern software development and digital transactions, facilitating communication and data exchange between diverse systems. However, their widespread use has made them prime targets for cyber-attacks. Many APIs are developed rapidly without sufficient security measures, leading to vulnerabilities such as weak authentication, data exposure, inadequate logging, and poor error handling. The bachelor thesis aims to develop labs in API security for future OST Hacking-Lab students to raise awareness of risks.

The research phase extensively examined API history, styles, and security fundamentals. Key areas such as threat identification, authentication methods and the OWASP Top 10 API Security Risks 2023 were explored. This foundational research informed the collection and categorization of lab ideas, which were then evaluated using a decision matrix based on feasibility, educational value, and expandability criteria.

A proof of concept (PoC) phase validated the feasibility of each lab, followed by iterative improvements based on detailed feedback from usability testing. Participants evaluated the labs on setup difficulty, usability, design, and realism, leading to enhancements that ensured an effective learning experience.

The project successfully developed six labs covering most OWASP Top 10 API Security risks. Each lab provided hands-on experience identifying and mitigating these vulnerabilities through practical exercises using tools in a containerized environment.

To enhance the educational value, future expansions could include additional labs to cover remaining OWASP risks and specialized areas like Cloud Provider APIs and advanced OAuth2 authentication flows.

Management summary

Problem statement

In modern software development and digital transactions, Application Programming Interfaces (APIs) play a crucial role. They enable different software systems to communicate and share data. However, due to their widespread use, APIs have become prime targets for cyberattacks. Many APIs are built quickly without enough focus on security, making them vulnerable to threats such as unauthorized access, data breaches, and service disruptions. Common security issues include weak authentication, exposure of sensitive data, insufficient logging, and poor error handling.

Given the importance of APIs, it is essential to address these security challenges. This project aims to develop practical API Security labs for students at the OST Hacking-Lab. These labs will provide hands-on experience with real-world scenarios, helping students learn how to secure APIs effectively. The labs will cover critical topics such as authentication methods, rate limiting, input validation, and secure API implementation. This practical approach will prepare students for the development and usage of APIs.

Approach & technology

The foundation of the thesis was built upon deep research into various aspects of API Security. This research included the history and evolution of APIs, different API styles and their security implications, and critical factors necessary for ensuring API Security. Methods for identifying API threats were explored. Various authentication methods, including token-based authentication and OAuth2, were examined. Detailed workings of OAuth2 flows were analyzed, and the OWASP Top Ten Security Risks 2023, which are listed below in Table 1, were reviewed.

OWASP Top 10 API Security Risks – 2023	
API1	Broken Object Level Authorization
API2	Broken Authentication
API3	Broken Object Property Level Authorization
API4	Unrestricted Resource Consumption
API5	Broken Function Level Authorization
API6	Unrestricted Access to Sensitive Business Flows
API7	Server-Side Request Forgery (SSRF)
API8	Security Misconfiguration
API9	Improper Inventory Management
API10	Unsafe Consumption of APIs

Table 1: OWASP Top 10 API Security Risks – 2023

Building on this research, a wide array of lab ideas was collected. These ideas were organized within a framework to improve comparison and provide an overview of the categories they cover, ensuring a wide variety of labs. A set of criteria for the decision matrix was defined to evaluate and select suitable lab ideas systematically. These criteria included the knowledge required to build them, the risk of being orphaned, access to online resources for help, expandability for future labs, and interest for students and lecturers.

The six labs with the highest scores were selected using the decision matrix, and each was subjected to a PoC analysis. This PoC phase involved developing a basic version of each lab to test core concepts and functionalities. It was crucial to identify potential challenges and refine the lab design before full-scale development.

After development, the labs underwent a usability testing phase involving a group of test participants. Participants provided valuable feedback on setup difficulty, usability, general difficulties encountered, lab design, and realism. This feedback was used to make iterative improvements, enhance the setup process, clarify instructions, and refine lab scenarios to ensure an effective learning experience.

Results

The project successfully developed six labs, each addressing a distinct vulnerability from the OWASP Top 10 API Security Risks 2023. These labs covered key risks such as Broken Authentication, Broken Object Property Level Authorization, Unrestricted Resource Consumption, Server-Side Request Forgery, Security Misconfiguration, and Unsafe Consumption of APIs.

The following labs were developed:

- **Implementing logging:** Students learn how to implement robust logging mechanisms to monitor API activity and detect potential security incidents.
- **Enumeration and reconnaissance:** This lab teaches students how to recognize and prevent unauthorized information gathering from APIs.
- **OAuth2 vulnerabilities:** Students are taught to analyze and identify an improper authentication implementation and exploit it to gain unauthorized access.
- **OWASP Coraza WAF:** This lab configures and uses the OWASP Coraza WAF to protect an application from various attack vectors.
- **Rate limiting:** Students learn how to implement rate limiting to prevent abuse of API endpoints.
- **Input validation and sanitization:** This lab emphasizes the importance of validating and sanitizing input to prevent injection attacks and other malicious activities.

The developed labs were categorized into three types: two tool-based, three implementation-based, and one lab focused on exploiting vulnerabilities from an attacker's perspective.

The labs were designed to be engaging and interactive, providing a structured learning experience. Each lab used Docker containers to ensure isolated and consistent environments, making the setup process straightforward and repeatable. Feedback from participants indicated high satisfaction with the labs' setup difficulty, usability, general challenges, design, and realism.

Looking ahead, the project could expand by creating additional labs to cover the remaining OWASP Top 10 API Security Risks. Focusing on specific areas such as cloud provider APIs and delving deeper into OAuth2 authentication flows could further enhance the educational value. This future work would ensure a comprehensive coverage of API Security, addressing evolving cybersecurity challenges effectively.

Acronyms

- API** Application Programming Interface. 10
- COBRA** Common Object Request Broker Architecture. 13
- CSRF** Cross-Site Request Forgery. 21
- DoS** Denial-of-service. 23
- EJB** Enterprise Java Bean. 13
- HTTP** Hypertext Transfer Protocol. 13
- IDE** Integrated Development Environment. 42
- IT** Informatin technology. 11
- JWT** JSON Web Token. 21
- NIST** National Institute of Standards and Technology. 12
- OIDC** OpenID Connect. 21
- OWASP** Open Worldwide Application Security Project. 22
- PKCE** Proof Key of Code Exchange. 19
- PoC** Proof of concept. 10
- RBAC** Role Based Access Control. 27
- REST** Representational State Transfer. 13
- RMI** Remote Method Invocation. 13
- RPC** Remote Procedure Call. 13

SMTP Simple Mail Transfer Protocol. 13

SOAP Simple Object Access Protocol. 12

SQL Structured Query Language. 30

SSRF Server Side Request Forger. 23

TCP Transmission Control Protocol. 13

UDP User Datagram Protocol. 13

URL Uniform Resource Locator. 36

UUID Universally Unique Identifier. 11

W3C World Wide Web Consortium. 12

WAF Web Application Firewall. 15

WSDL Web Services Description Language. 13

XML Extensible Markup Language. 13

XSS Cross-Site-Scripting. 30

Glossary

Hacking-Lab Hacking-Lab is an online ethical hacking, computer network and security challenge (CTF) and training platform used by individuals, enterprises, universities, educational entities and armed forces.. 10

Microservice Microservice is an architectural style that structures an application as a collection of services that are Independently deployable and Loosely coupled. 14

ModSecurity ModSecurity is an open-source web application firewall module that is widely used to enhance the security of web applications and protect them from various attacks. It operates as an Apache or Nginx module, providing real-time monitoring, logging, and access control capabilities to HTTP requests and responses passing through a web server. . 46

OWASP ModSecurity Core Rule Set The OWASP ModSecurity Core Rule Set is a set of generic attack detection rules for use with ModSecurity or compatible web application firewalls. The CRS aims to protect web applications from a wide range of attacks, including the OWASP Top Ten, with a minimum of false alerts. The CRS provides protection against many common attack categories, including SQL Injection, cross-site scripting, Local File Inclusion, etc. . 46

Rainbow table A rainbow table is a database that is used to gain authentication by cracking the password hash. It is a precomputed dictionary of plaintext passwords and their corresponding hash values that can be used to find out what plaintext password produces a particular hash. . 40

RESTful A RESTful API is an architectural style for an application programming interface that uses HTTP requests to access and use data. That data can be used to GET, PUT, POST and DELETE data types, which refers to reading, updating, creating and deleting operations related to resources. . 12

Part II

Technical report

Chapter 1

Introduction

The following chapter gives a brief introduction to the initial situation and the scope of the bachelor thesis.

1.1 Initial situation

APIs play a crucial role in software development and digital business transactions. However, with the increasing spread and importance of APIs, there is also a growing threat landscape.

This bachelor thesis aims to develop an API Security Lab curriculum for future students of the OST using the Hacking-Lab. This curriculum will provide practical exercises that allow students to apply theoretical concepts in a hands-on environment, simulating real-world scenarios and challenges.

By completing these interactive tasks, future OST students will be better equipped to identify and mitigate API Security risks, securely develop and consume APIs, and contribute to a more secure digital ecosystem. This project can potentially enhance the practical skills and knowledge of future OST Hacking-Lab students, preparing them for the growing demand for expertise in API Security.

1.2 Project scope

The scope of this bachelor thesis is to evaluate labs for the OST Hacking-Lab platform. Future students should gain an in-depth insight into the topic based on the labs developed. The first part of this project will cover research and evaluation of topics regarding API Security. The second part will be about selecting the topics based on criteria and PoCs. The final part will be the lab development.

1.3 OST Hacking-Lab

The Hacking-Lab is an online education platform dedicated to the realm of cybersecurity. It contains practice-oriented labs, which are organised in an event covering a specific IT security subject. The challenges include clear instructions outlining the tasks and deliverables for a successful lab completion. The submission can be a flag, which is a random UUID hidden in the lab exercises or a write-up, a report documenting the problem-solving process or answers to the lab questions.

This bachelor thesis does not include a detailed explanation of the Hacking-Lab and its components. They are prerequisites for those who read and continue work on this thesis.

Chapter 2

Research

This chapter covers the research done on and around API Security during this thesis. It also briefly introduces APIs and their historical usage before delving deep into the security aspects.

2.1 API and history

The following section covers the definition of API and the historical usage of it.

2.1.1 Definition

According to the National Institute of Standards and Technology (NIST) the term API is defined as follows:

A system access point or library function with a well-defined syntax accessible from application programs or user code to provide well-defined functionality [1].

2.1.2 What is an API?

Software applications historically leveraged APIs as linked libraries, gaining access to functions and procedures. Nowadays, most APIs are available over the internet as RESTful web services [2].

An API serves as a boundary within a software system, offering a defined set of operations for other components or systems to utilise. It handles client requests, which can be user interfaces or other APIs. The API may interact with different APIs to fulfil its tasks [2].

2.1.3 Simple Object Access Protocol

Simple Object Access Protocol (SOAP) is an important messaging standard defined by the World Wide Web Consortium (W3C) and its member editors that helped introduce the widespread use of web services, also called APIs [3] SOAP uses an Extensible Markup

Language (XML) data format to declare its request and response messages, relying on XML Schema to enforce the structure of its payloads. It consists of three parts [4, 5]:

- an envelope, which defines the message structure and how to process it
- a set of encoding rules for expressing instances of application-defined data types
- a convention for representing procedure calls and responses

SOAP has three major characteristics:

- extensibility
- neutrality (SOAP can operate over any protocol such as Hypertext Transfer Protocol(HTTP), Simple Mail Transfer Protocol(SMTP), Transmission Control Protocol(TCP), User Datagram Protocol(UDP))
- independence (SOAP allows for any programming model)

2.1.4 Web Services Description Language

Web Services Description Language(WSDL is an XML-based language used to describe the functionality offered by a web service. It defines the interface of a web service by specifying the operations it supports, the input and output parameters for each operation, and the protocol and data format used for communication. Its usage has declined in recent years, particularly with the rise of REST APIs and more modern approaches to describing APIs [2].

2.1.5 API styles

Various approaches have emerged in API design regarding providing access to remote functionalities. These approaches include [2]:

- **Remote Procedure Cal(RPC) APIs:** Expose procedures or functions callable by clients over a network connection. They resemble local procedure calls but often require specific client libraries. The gRPC framework from Google is an example of a modern RPC approach. The older SOAP framework, which uses XML for messages, is still widely deployed.
- **Remote Method Invocation(RMI):** A variant of RPC using object-oriented techniques, allowing clients to call methods on remote objects as if they were local. Technologies like Common Object Request Broker Architecture(COBRA) and Enterprise Java Bean(EJB)s were popular but have declined due to the complexity of their use.
- **Representational State Transfer(REST) Style:** Developed by Roy Fielding, it emphasises standard message formats and a few generic operations to reduce client-API coupling. Hyperlinks are used for navigation, reducing the risk of client breakage as the API evolves.

- **Specialised querying APIs:** Some APIs focus on efficiently filtering large datasets, such as SQL databases or frameworks like GraphQL from Facebook. They provide a few operations with a complex query language, granting clients significant control over data retrieval [6].
- **WebSocket:** A communication protocol providing full-duplex communication channels over a single TCP connection, often used for real-time applications such as chat applications and online gaming [7].

Different API styles are preferable in varying environments. For instance, in an Microservice architecture, an efficient RPC framework can minimise the overhead of API calls, which is suitable when an organisation has control over clients and servers, allowing easy distribution of new stub libraries. Conversely, for widely used public APIs, the REST style, employing formats like JSON, enhances interoperability with different client types [2].

2.2 API Security in context

API Security lies in several security disciplines, but it can be restricted into three most important areas as shown in the Figure 2.1:

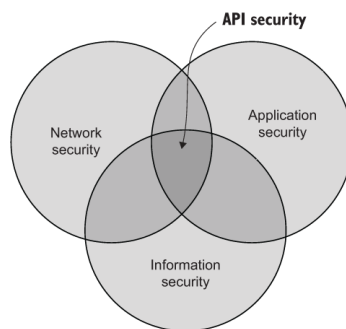


Figure 2.1: API Security fields [2]

- **Information security:** Focuses on safeguarding information throughout its life-cycle, including creation, storage, transmission, backup, and destruction.
- **Network security:** Addresses the protection of data transmitted over networks and guards against unauthorised access to the network infrastructure.
- **Application security:** Ensures that software systems are constructed and operated to resist attacks and prevent misuse.

2.2.1 Information security

In Information security, the security goals are defined, and potential threats are identified. This process involves analysing potential risks and vulnerabilities associated with

the APIs. Additionally, techniques for protecting APIs using access control mechanisms are explored, ensuring that only authorised users or systems can interact with information security. It also delves into securing sensitive information through applied cryptography and encryption. Mastering these concepts equips individuals to establish robust security measures to protect APIs and the data they handle [2].

2.2.2 Network security

In the context of network security, basic tools are used to protect APIs on the network layer, such as web application firewalls (WAF), load-balancers, and reverse proxies. These tools play different roles in keeping APIs safe. Additionally, secure communication protocols like HTTPS [2] are used.

2.2.3 Application security

In application security, secure coding techniques are implemented. Additionally, common software security vulnerabilities are studied to mitigate risks effectively. Understanding how to securely store and manage system and user credentials for API access is also a crucial aspect [2].

2.3 Identifying threats

The protection of APIs is strongly related to the understanding of the environment. Every environment where the API is located has different potential threats. The set of threats considered relevant to the API is known as the threat model, and the process of identifying them is called threat modelling [2].

There are many ways to do threat modelling, but the general process is as follows [2]:

1. Draw a system diagram showing the main logical components of your API.
2. Identify trust boundaries between parts of the system. Everything within a trust boundary is controlled and managed by the same owner, such as a private data centre or a set of processes running under a single operating system user.
3. Draw arrows to show how data flows between the various parts of the system.
4. Examine each component and data flow in the system and identify threats that might undermine your security goals in each case. Pay particular attention to flows that cross trust boundaries.
5. Record threats to ensure they are tracked and managed.

2.4 Defensive mechanism

There are several countermeasures to protect APIs. The most common security mechanisms are listed below [2].

- Encryption: Safeguards data from unauthorised access during transmission from the API to a client and when stored in a database or filesystem. It also prevents unauthorised modification of data by attackers.
- Authentication: Verifies the identity of users and clients, ensuring they are who they claim to be.
- Access Control (Authorisation): Ensures that every request to the API is appropriately permitted based on predefined criteria.
- Audit Logging: Records all API operations, promoting accountability and facilitating effective API monitoring.
- Rate-Limiting: Prevents any individual user or group from monopolising resources, thus ensuring fair access for all legitimate users.

The Figure 2.2 shows how these processes are layered as a series of filters a request passes through. Each stage can also be outsourced by external components such as API gateway [2].

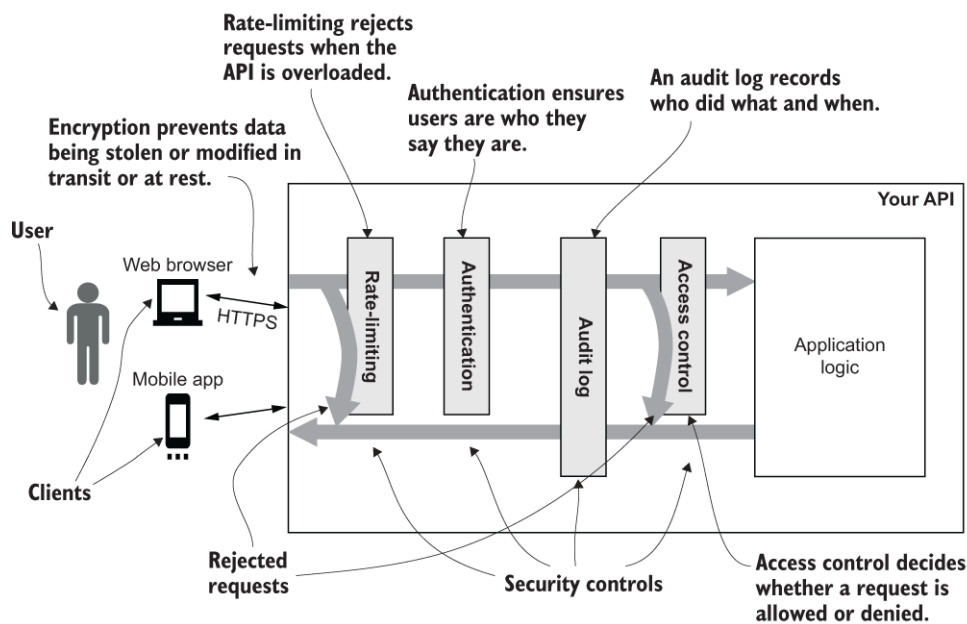


Figure 2.2: Defensive mechanism APIs [2]

2.5 Token-based authentication and OAuth 2.0

Token-based authentication and OAuth 2.0 are essential for securing modern APIs. Token-based authentication allows clients to authenticate once and use a token for subsequent requests, enhancing security and efficiency. OAuth 2.0 is a framework that enables third-party applications to access user resources without exposing user credentials. This section covers the principles of token-based authentication and OAuth 2.0, including token generation, usage, and validation, as well as the various OAuth flows and their roles in secure authorisation [8].

2.5.1 Token-based authentication

Token-based authentication is a popular method used to secure access to APIs.

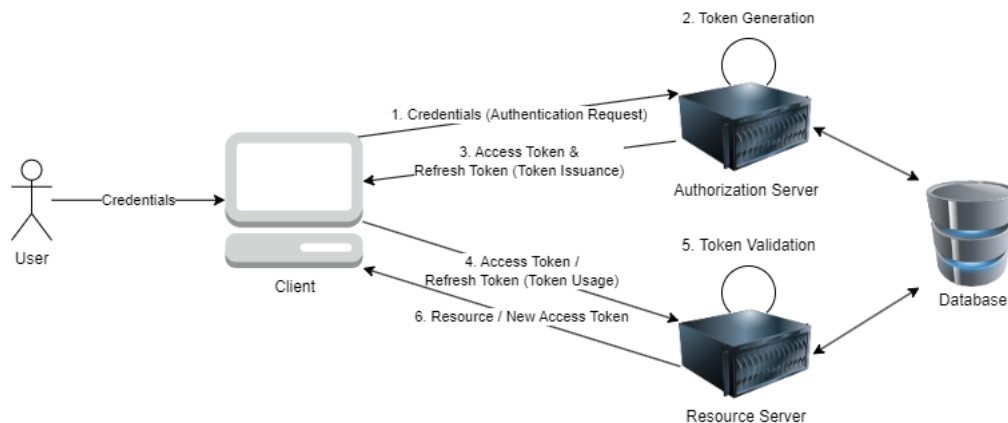


Figure 2.3: Token-based authentication

The following steps describe the procedure of token-based authentication for accessing an API as in the Figure 2.3 [9]:

1. **Authentication:** When a user or client application wants to access an API, they must first authenticate themselves. Instead of providing their username and password with each request, they typically exchange those credentials for a token.
2. **Token generation:** Upon successful authentication, the server generates a token. This token is a unique string of characters that proves the user's identity and authorisation to access specific resources. Tokens are usually generated using cryptographic algorithms and can contain information such as the user's identity, expiration time, and scope of access.
3. **Token issuance:** The server then returns the token to the client, typically in response to the authentication request. The client stores this token securely, often in memory or a secure storage mechanism like the browser's local storage or a mobile device's safe storage.

4. **Token usage:** The client includes the token in the header or as a parameter of subsequent API requests it makes to the server. This token serves as a credential that proves the client's identity and authorisation to access protected resources.
5. **Token validation:** The server validates the token with each request to ensure that it has not expired and that a trusted party issued it.
6. **Send resource:** The resource server sends a response to the requested resources.

2.5.2 OAuth2

OAuth 2.0 serves as a framework for granting third-party applications access to resources on behalf of a user without requiring the user to disclose their credentials directly to the application. It's widely used in scenarios where users want to share their information stored on one platform with another platform or application. Instead of sharing passwords, users authenticate through a trusted provider, like Google or Facebook, and then authorise specific access to their data for third-party applications. This protocol ensures a secure, standardised method for applications to access limited user data without compromising security [10].

OAuth defines four roles [10]:

- **Resource owner:** An entity capable of granting access to a protected resource. When the resource owner is a person, it is referred to as an end-user.
- **Resource server:** The server hosting the protected resources is capable of accepting and responding to protected resource requests using access tokens.
- **Client:** An application making protected resource requests on behalf of the resource owner and with its authorisation. The term "client" does not imply any particular implementation characteristics (e.g., whether the application executes on a server, a desktop, or other devices).
- **Authorization server:** The server issues access tokens to the client after successfully authenticating the resource owner and obtaining authorisation.

Access tokens

An OAuth Access token serves as a string the OAuth client utilises to request resources from the resource server. These tokens don't adhere to a specific format, and different OAuth servers employ various formats for their access tokens [11].

Several key properties of access tokens are integral to OAuth's security model [11]:

- Access tokens aren't meant to be read or understood by the OAuth client.

- They don't convey user identity or other user-related information to the OAuth client.
- Access tokens should solely be utilised for making requests to the resource server. Furthermore, ID tokens shouldn't be employed to make requests to the resource server.

Refresh tokens

An OAuth refresh token is a string, which the OAuth client uses to obtain a new access token without requiring the user's involvement. Notably, a refresh token is restricted from granting the client access beyond the scope of the initial authorisation. It allows authorisation servers to use short-lived access tokens without user intervention when the tokens expire [11].

2.5.3 OAuth scopes

Scope in OAuth 2.0 restricts an application's access to a user's account. The application can request one or more scopes, which are then displayed to the user for consent. The access token issued to the application is then limited to the granted scopes [11].

2.5.4 OAuth grant types

OAuth grant type defines the exact sequence of steps in the OAuth process. It also defines how the client application communicates with the OAuth service at each stage and how the token is sent. OAuth grant types are often called "OAuth flows" [12].

OAuth services must be set up to support specific grant types before client applications can establish the respective flows. In the initial authorisation request to the OAuth service, the client application indicates the desired grant type [12].

Various grant types exist, each presenting distinct levels of complexity and security implications. Following the two most common grant types, "authorisation code", "implicit (legacy)", and Proof Key of Code Exchange (PKCE) are described [12].

Authorisation code grant type

The client application and OAuth service use several browser-based HTTP requests to initiate the flow. The user will be asked whether they consent to the requested access. After they accept it, the client application is granted an authorisation code. This code is used to exchange with the OAuth service to get the access token, which can be used to make API calls [12]. The full flow can be seen in the Figure 2.4.

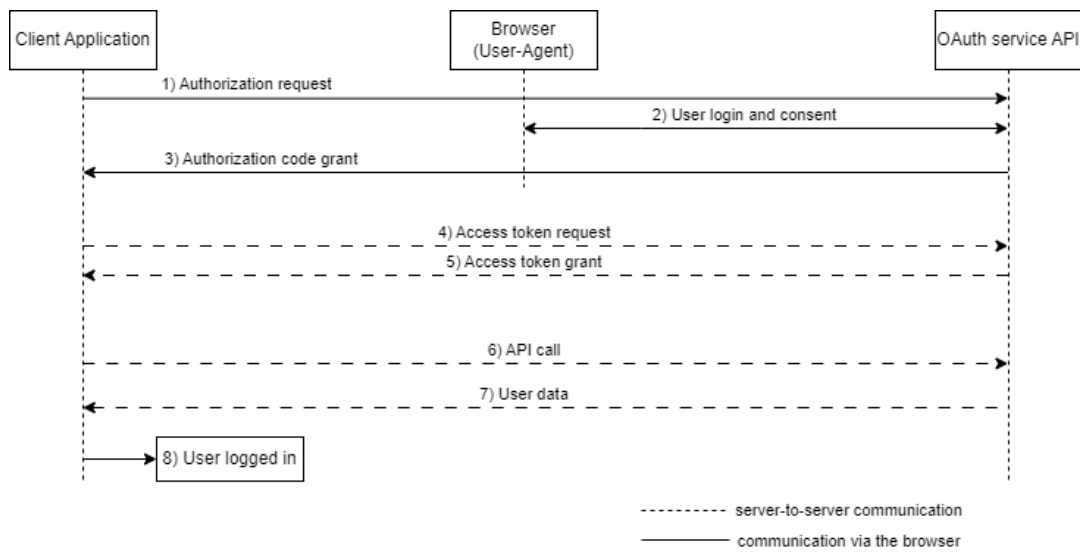


Figure 2.4: Authorization code flow

Implicit grant type

The Implicit grant type gives the client application the access token right after the user agrees on the consent. This way it skips getting an authorisation code first, as you can see in the Figure 2.5. However, it's not as secure as the other grant types because it relies only on browser redirects for communication. This makes it easier for attackers to access the access token and the user's data [12].

Also, according to the OAuth community, it is defined as a legacy grant type.

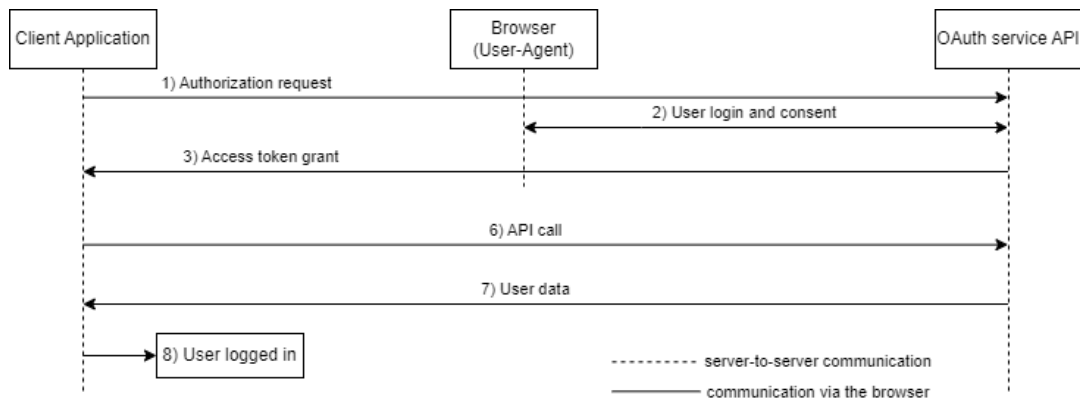


Figure 2.5: Implicit flow

PKCE

The PKCE extends the authorisation code grant type flow. It was evaluated for the implicit flow of single-page applications or native apps. These apps are vulnerable to Cross-Site Request Forgery(CSRF) attacks since they are considered public clients and cannot securely store their client credentials. The client creates a random code verifier and the corresponding hash, both will be included in the authorisation request. The authorisation server validates the verifier during the token exchange process to ensure the authenticity of the request and to mitigate any interceptions [12]. The full flow can be seen in the Figure 2.6.

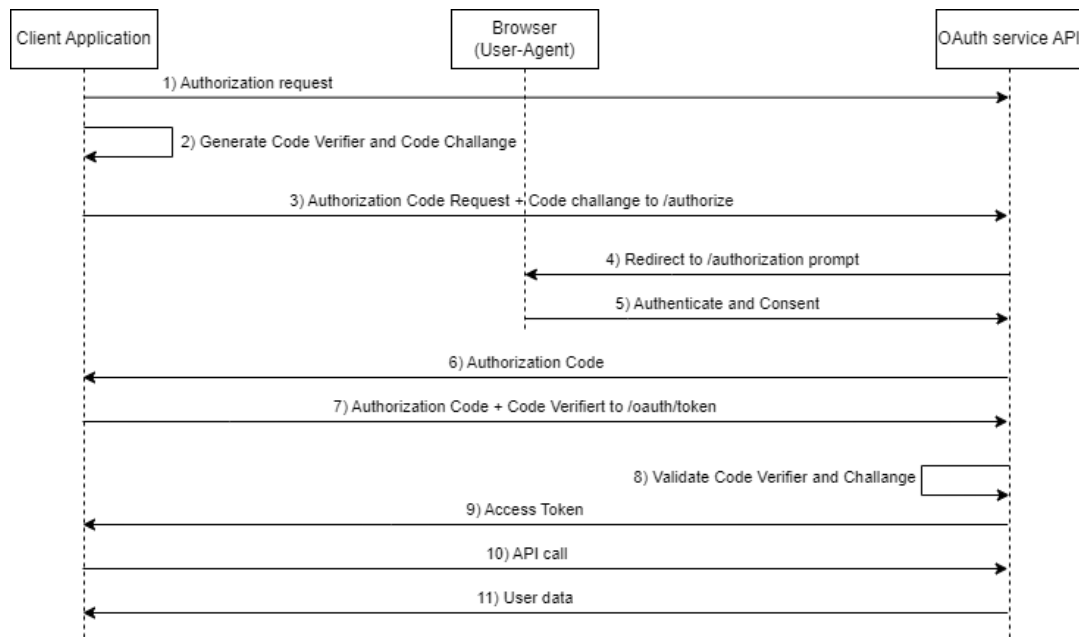


Figure 2.6: Authorisation code flow with PKCE

2.6 OpenID Connect

OpenID Connect (OIDC) is an authentication protocol built on top of OAuth2. It allows clients to verify end-users identities based on the authentication performed by an authorisation server and obtain basic profile information about the user. It uses JSON web tokens (JWT), which can be obtained using flows conforming to the OAuth2 specifications [13].

2.7 Microservice

Microservices are a software development approach where applications are structured as a collection of loosely coupled services. Each service is focused on a specific business

capability and can be independently developed, deployed, and scaled. In the context of APIs, Microservices often expose their functionality through well-defined interfaces, allowing other services to interact with them. This promotes modularity and flexibility, enabling agile development and easier maintenance. Microservices architecture facilitates the creation of complex systems by breaking them into smaller, manageable components that small teams can develop and maintain. Overall, Microservices enhance scalability, resilience, and agility in building and evolving modern software systems [14].

2.8 OWASP Top 10 API Security Risks 2023

The primary goal of the Open Worldwide Application Security Project (OWASP) API Security Top 10 is to educate those involved in API development and maintenance, for example, developers, designers, architects, managers, or organizations [15].

2.8.1 Broken Object Level Authorisation

Broken object-level authorisation refers to a vulnerability where an API lacks proper authorisation checks at the object level, allowing unauthorised access to resources. Even though the API may authenticate users correctly, it fails to adequately enforce access controls on individual objects or data within the system. This could lead to attackers gaining unauthorised access to sensitive information or performing actions they shouldn't be allowed to, such as viewing or modifying the data of other users [16].

2.8.2 Broken Authentication

Broken Authentication refers to a vulnerability where authentication mechanisms within an API are compromised or improperly implemented, leading to unauthorised access to resources or sensitive data. This vulnerability can arise from weak passwords, insufficient session management, or flawed authentication protocols. Attackers exploit these weaknesses to impersonate legitimate users, gaining unauthorised access to accounts or sensitive information. Broken authentication can result in various security threats, including account takeover, identity theft, and unauthorised data disclosure [16].

2.8.3 Broken Object Property Level Authorization

Broken Object Property Level Authorization refers to a security vulnerability where an API lacks proper authorisation checks at the property level of objects, allowing unauthorised access to specific attributes or fields within those objects. While the API may enforce access controls at the object level, it fails to restrict access to individual properties or data fields within those objects. Attackers can exploit this vulnerability to access sensitive data within objects they're not authorised to view or modify, potentially leading to data leakage or unauthorised manipulation [16].

2.8.4 Unrestricted Resource Consumption

Unrestricted Resource Consumption is a vulnerability where an API does not appropriately limit the amount of resources a user can consume, leading to excessive utilisation and potential Denial of Service (DoS). This vulnerability can occur due to inefficient algorithms, lack of rate limiting, or inadequate resource management. Attackers exploit this weakness by sending many requests or consuming excessive resources, causing the system to become overwhelmed and unresponsive. Unrestricted Resource Consumption can result in service degradation or complete downtime, impacting the availability and performance of the API for legitimate users [16].

2.8.5 Broken Function Level Authorization

Broken Function Level Authorization is a vulnerability where an API fails to properly enforce access controls at the function or endpoint level, allowing unauthorised users to execute certain functionalities. This occurs when the API assumes that authentication alone is sufficient to determine access rights without further validation. Attackers exploit this weakness to access restricted functionalities or perform actions they are not authorised to execute. Broken Function Level Authorization can lead to unauthorised data manipulation, privilege escalation, or other security breaches [16].

2.8.6 Unrestricted Access to Sensitive Business Flows

Unrestricted Access to Sensitive Business Flows is a vulnerability in which an API allows unauthorised users to access critical business processes or workflows without proper authentication or authorisation. This vulnerability arises when sensitive operations within the API lack sufficient access controls, enabling unauthorised users to exploit them. Attackers can leverage this vulnerability to tamper with crucial business processes, manipulate sensitive data, or perform unauthorised actions, posing significant risks to the organisation's operations and security [16].

2.8.7 Server Side Request Forgery

Server Side Request Forgery (SSRF) is a vulnerability in which an attacker manipulates an API into making unauthorised requests to internal or external resources on behalf of the server or application. This occurs when the API allows the attacker to specify the destination of a request, and the server processes it without proper validation. Attackers exploit SSRF to access sensitive data, bypass firewalls, or perform reconnaissance on internal networks. SSRF can lead to data breaches, service disruptions, or unauthorised access to internal resources [16].

2.8.8 Security Misconfiguration

Security Misconfiguration is a vulnerability where an API is deployed or configured in a way that leaves it open to exploitation or unauthorised access due to insecure

settings, defaults, or oversight. This vulnerability can result from improper configuration of security settings, outdated software versions, or unnecessary exposure of sensitive information. Attackers exploit security misconfigurations to gain unauthorised access, manipulate data, or launch other attacks on the system. Security misconfigurations can lead to data breaches, service disruptions, or unauthorised access to resources [16].

2.8.9 Improper Inventory Management

Improper Inventory Management in the context of an API refers to a vulnerability where the API fails to adequately track and manage its available resources, endpoints, or dependencies. This vulnerability may result from outdated or inaccurate inventory records, lack of monitoring, or inadequate documentation of API components. Attackers can exploit this weakness to target outdated or vulnerable components, leading to security breaches, service disruptions, or unauthorised access to sensitive data [16].

2.8.10 Unsafe Consumption of APIs

Unsafe Consumption of APIs is a vulnerability where an application or service utilises APIs in a manner that endanger the security and integrity of the system. This can occur due to various factors, such as inadequate validation of input/output, lack of encryption, or reliance on deprecated or vulnerable APIs. Attackers exploit this vulnerability to manipulate data, inject malicious payloads, or gain unauthorised access to sensitive information transmitted through the API. Unsafe consumption of APIs can lead to data breaches, system compromises, or other security incidents [16].

Chapter 3

Lab evaluation

This chapter describes the process of the lab evaluation. It includes collecting and sorting project ideas, creating the decision matrix, and checking the lab feasibility with a proof of concept analysis.

3.1 Approach

The lab evaluation process consists of the following steps. Firstly, lab ideas are collected, sorted and categorised using a framework. Criteria that are important for future labs are then defined. Based on these criteria, a decision matrix is completed with the project advisor and the final lab ideas are sorted out. Before they are realised, these are checked for feasibility in the PoC.

3.2 Lab idea taxonomy

The lab idea framework provides a structured approach for organising potential lab exercises. This framework categorises lab ideas based on relevant factors so that the comparison of the ideas and decision-making is organised.

The following lab properties are defined:

- **ID** - Identifier for further linkings
- **Name** - Lab name
- **Type** - Lab mode if attacking or defending scenario.
 - Attacking
 - Defensive
- **Complexity** - Lab difficulty and time consumption

- Easy
- Medium
- Hard
- **Category** - Based on API Security basics, the five main categories
 - Encryption
 - Authentication
 - Authorisation
 - Audit logging
 - Rate limiting
- **Required skills** - Required skills needed for students
 - Security fundamentals
 - Programming & scripting
 - Network security

An example of an empty lab framework can be seen in the Table 3.1.

ID		Name	
Type	Complexity	Category	Required skill
Attacking	Easy	Encryption	Security fundamentals
		Authentication	
	Medium	Authorization	Programming & scripting
Defensive		Audit logging	
	Hard	Rate limiting	Network security

Table 3.1: Example lab framework

3.3 Lab idea collection

The following section contains the collection of the lab ideas, which will then be evaluated in the decision matrix.

3.3.1 Lab ideas

OAuth2 WebApp

This lab focuses on setting up OAuth2 for a basic WebApp using Keycloak. Students will learn to integrate OAuth2 for authentication and authorisation in web applications. This exercise emphasises practical implementation and understanding of the security benefits and challenges of OAuth2.

The corresponding lab framework can be seen in the Figure 3.2.

ID	1	Name	OAuth2 WebApp
Type	Complexity	Category	Required skill
Attacking	Easy	Encryption	Security fundamentals
		Authentication	
	Medium	Authorization	Programming & scripting
Defensive		Audit logging	
	Hard	Rate limiting	Network security

Table 3.2: Lab idea: OAuth2 WebApp

JWT & Role Based Access Control (RBAC)

In this lab, students will implement secure authentication and authorisation mechanisms for an API using JWT and RBAC. This lab teaches how to generate and validate JWTs and use RBAC to enforce fine-grained access control within an application.

The corresponding lab framework can be seen in the Figure 3.3.

ID	2	Name	JWT & RBAC
Type	Complexity	Category	Required skill
Attacking	Easy	Encryption	Security fundamentals
		Authentication	
	Medium	Authorization	Programming & scripting
Defensive		Audit logging	
	Hard	Rate limiting	Network security

Table 3.3: Lab idea: JWT & RBAC

OWASP Juice Shop

This lab involves selecting challenges regarding API Security in the OWASP Juice Shop. Students will engage with realistic scenarios to identify and mitigate security vulnerabilities in APIs using a popular open-source platform designed for security training [17]. The corresponding lab framework can be seen in the Figure 3.4.

ID	3	Name	OWASP Juice Shop
Type	Complexity	Category	Required skill
Attacking	Easy	Encryption	Security fundamentals
		Authentication	
	Medium	Authorization	Programming & scripting
Defensive		Audit logging	
	Hard	Rate limiting	Network security

Table 3.4: Lab idea: OWASP Juice Shop

Implementing logging

In this lab, students will learn how to implement logging for an API. They will understand the importance of audit logging, capturing relevant security events, and analysing logs to detect and respond to potential security incidents.

The corresponding lab framework can be seen in the Figure 3.5.

ID	4	Name	Implementing logging
Type	Complexity	Category	Required skill
Attacking	Easy	Encryption	Security fundamentals
		Authentication	
	Medium	Authorization	Programming & scripting
Defensive		Audit logging	
	Hard	Rate limiting	Network security

Table 3.5: Lab idea: Implementing logging

API rate limiting

Students will learn how to implement API rate limiting and throttling mechanisms to prevent abuse, ensure fair usage, and improve the security and performance of an API service. This lab highlights various strategies for controlling API usage and mitigating DoS attacks.

The corresponding lab framework can be seen in the Figure 3.6.

ID	5	Name	API rate limiting
Type	Complexity	Category	Required skill
Attacking	Easy	Encryption	Security fundamentals
		Authentication	
	Medium	Authorization	Programming & scripting
Defensive		Audit logging	
	Hard	Rate limiting	Network security

Table 3.6: Lab idea: API rate limiting

OWASP Coraza WAF

This lab focuses on API penetration testing and rule implementation with Coraza WAF. Students will explore how to use WAF to protect APIs from common attacks and how to analyse custom security rules.

The corresponding lab framework can be seen in the Figure 3.7.

ID	6	Name	OWASP Coraza WAF
Type	Complexity	Category	Required skill
Attacking	Easy	Encryption	Security fundamentals
		Authentication	
	Medium	Authorization	Programming & scripting
Defensive		Audit logging	
	Hard	Rate limiting	Network security

Table 3.7: Lab idea: OWASP Coraza WAF

API Security attacker view

This lab aims to help students understand common API Security vulnerabilities and how attackers exploit them. By simulating attack scenarios, students will gain insights into the attacker’s mindset and learn defensive strategies to protect APIs.

The corresponding lab framework can be seen in the Figure 3.8.

ID	7	Name	API Security attacker view
Type	Complexity	Category	Required skill
Attacking	Easy	Encryption	Security fundamentals
		Authentication	
	Medium	Authorization	Programming & scripting
Defensive		Audit logging	
	Hard	Rate limiting	Network security

Table 3.8: Lab idea: API Security from an attacker’s perspective

Tooling lab

This lab aims to familiarise students with standard API Security testing and assessment tools. Students will explore various security tools and their applications in identifying and mitigating API vulnerabilities.

The corresponding lab framework can be seen in the Figure 3.9.

ID	8	Name	Tooling lab
Type	Complexity	Category*	Required skill
Attacking	Easy	Encryption	Security fundamentals
		Authentication	
	Medium	Authorization	Programming & scripting
Defensive		Audit logging	
	Hard	Rate limiting	Network security

Table 3.9: Lab idea: Tooling lab

*No specific category addressable

3.3.2 OAuth2 vulnerabilities

In this lab, students will explore and exploit common OAuth2 vulnerabilities. They will learn about potential weaknesses in OAuth2 implementations and how attackers can exploit these vulnerabilities to compromise security.

The corresponding lab framework can be seen in the Figure 3.10.

ID	9	Name	OAuth2 vulnerabilities
Type	Complexity	Category	Required skill
Attacking	Easy	Encryption	Security fundamentals
		Authentication	
	Medium	Authorization	Programming & scripting
Defensive		Audit logging	
	Hard	Rate limiting	Network security

Table 3.10: Lab idea: OAuth2 vulnerabilities

JWT vulnerabilities

This lab focuses on discussing and exploiting common JWT vulnerabilities. Students will understand the pitfalls in JWT implementation and how to secure JWT-based authentication and authorisation mechanisms.

The corresponding lab framework can be seen in the Figure 3.11.

ID	10	Name	JWT vulnerabilities
Type	Complexity	Category	Required skill
Attacking	Easy	Encryption	Security fundamentals
		Authentication	
	Medium	Authorization	Programming & scripting
Defensive		Audit logging	
	Hard	Rate limiting	Network security

Table 3.11: Lab idea: JWT vulnerabilities

Input validation and sanitisation

Students will implement input validation and sanitisation techniques to prevent Structured Query Language (SQL) injection, Cross-Site-Scripting (XSS) attacks, and other common vulnerabilities. This lab emphasises the importance of validating and sanitising user inputs to maintain application security.

The corresponding lab framework can be seen in the Figure 3.12.

ID	11	Name	Input validation and sanitisation
Type	Complexity	Category	Required skill
Attacking	Easy	Encryption	Security fundamentals
		Authentication	
	Medium	Authorization	Programming & scripting
Defensive		Audit logging	
	Hard	Rate limiting	Network security

Table 3.12: Lab idea: Input validation and sanitisation

API enumeration and reconnaissance

To understand the API’s functionalities, students will perform API enumeration techniques such as endpoint discovery, parameter probing, and header analysis. This lab teaches an attacker the initial steps to discover API weaknesses.

The corresponding lab framework can be seen in the Figure 3.13.

ID	12	Name	API enumeration and reconnaissance
Type	Complexity	Category	Required skill
Attacking	Easy	Encryption	Security fundamentals
		Authentication	
	Medium	Authorization	Programming & scripting
Defensive		Audit logging	
	Hard	Rate limiting	Network security

Table 3.13: Lab idea: API enumeration and reconnaissance

API Logging & monitoring

In this lab, students will implement a monitoring mechanism for APIs to track and analyse security-related events. They will learn how to set up logging infrastructure, monitor API traffic, and respond to suspicious activities effectively.

The corresponding lab framework can be seen in the Figure 3.14.

ID	13	Name	API Logging & monitoring
Type	Complexity	Category	Required skill
Attacking	Easy	Encryption	Security fundamentals
		Authentication	
	Medium	Authorization	Programming & scripting
Defensive		Audit logging	
	Hard	Rate limiting	Network security

Table 3.14: Lab idea: API Logging & monitoring

3.4 Decision matrix

In this section, the decision matrix for the labs is evaluated. The criteria are described, and weights are set to address the interest.

A decision matrix, as you see in Figure 3.1, is a systematic approach used to evaluate and prioritise a set of options based on predefined criteria. This method involves listing topics in rows and evaluation criteria in columns, assigning weights to each criterion according to its importance. Each topic is then scored against the criteria, and the scores are multiplied by the weights to produce a weighted score for each option. The total weighted scores are calculated to determine the best topics. This structured technique helps in making objective, informed decisions by quantifying and comparing the worth of each topic, ensuring that all relevant factors are considered and balanced according to their significance.

The five highest total values are coloured green. This means that the topics Implement logging, Implement rate-limiting and throttling, OWASP Coraza WAF, OAuth2 vulnerabilities, Input validation and sanitization, API enumeration and reconnaissance and API monitoring are elected for further progression. Due to time constraints, the lab idea API monitoring is not considered for further development.

3.4.1 Criteria

The following criteria are defined.

C1: Know-How

Description:

The following criteria will be set based on the knowledge of the topic and whether there is experience in this area.

Scale:

- 3 - Already used and have knowledge about the topic.
- 2 - Have some experience with a related topic
- 1 - No experience with the topic

Weight: The weighting of this criterion is classified as normal and, therefore, given the value of 1.

Decision:

This criterion is classified as normal because it can be influenced. In this way, missing know-how can be worked on.

C2: Danger of being orphaned

Description:

How likely is it that the topic is orphaned?

Scale:

- 3 - It is very unlikely that the topic will be orphaned
- 2 - There is a chance that the topic is orphaned
- 1 - It is likely that the topic is orphaned

Weight: The weighting of this criterion is classified as normal and, therefore, given the value of 1.

Decision:

This criterion is classified as normal because it is often impossible to predict accurately how a topic will develop.

C3: Documentation & Community

Description:

How well is the tool or topic documented, and are there active communities for support and questions?

Scale:

- 3 - Documentation is comprehensive, up-to-date, and user-friendly, providing clear guidance and examples
- 2 - Documentation exists but may be incomplete or difficult to follow.
- 1 - Documentation is scarce or outdated, making it challenging to understand and use the tool or topic.

Weight: The weighting of this criterion is classified as normal and, therefore, given the value of 1.

Decision:

This criterion is normal because meaningful documentation and community are useful but unnecessary.

C4: Expandability

Description:

Assess how well the project idea aligns with the current threat landscape in IT security.

Scale:

- 3 - The topic allows easy expansion

2 - It is possible to expand

1 - There is no possibility for expansion

Weight: The weighting of this criterion is considered important and, therefore, given the value of 2.

Decision:

This criterion is considered necessary because extensibility enables long-term use of systems and promotes innovation, flexibility and adaptability, which is of great advantage in a fast-moving technological environment

C5: Student interest

Description:

How much interest do we have in the topic?

Scale:

3 - The project idea will likely generate high interest and enthusiasm.

2 - The project idea has the potential to pique the interest.

1 - The project idea is unlikely to generate significant interest.

Weight: The weighting of this criterion is considered important and, therefore, given the value of 2.

Decision: This criterion is considered important because it serves as a personal control element for students.

C6: Lecturer interest

Description:

How much interest does the lecturer have in the topic?

Scale:

3 - The project idea will likely generate high interest and enthusiasm.

2 - The project idea has the potential to pique the interest.

1 - The project idea is unlikely to generate significant interest.

Weight: The weighting of this criterion is considered important and, therefore, given the value of 2.

Decision:

This criterion is considered important because it serves as a personal control element for the supervisor.

3.4.2 Outcome

35

Decisionmatrix API Security Lab		1	1	1	2	2	2	Weight
		Know-How	Danger of being orphaned	Docmunetation & Community	Expandability	Student interest	Lecturer interest	Criteria
Student rating area	Oauth2 WebApp	Corsin	1	3	3	3	2	37
		Thajakan	2	3	3	3	3	
		Ivan					0	
Topics	JWT / RBAC	Corsin	1	3	3	3	2	36
		Thajakan	1	3	3	3	2	
		Ivan					1	
	OWASP Juice Shop	Corsin	2	2	3	2	3	34
		Thajakan	2	2	3	2	2	
		Ivan					1	
	Implementing logging	Corsin	2	3	3	3	3	41
		Thajakan	2	3	2	3	2	
		Ivan					2	
	Implementing API rate limiting and throttling	Corsin	2	3	2	3	3	38
		Thajakan	1	2	2	2	2	
		Ivan					3	
	OWASP Coraza WAF	Corsin	1	2	3	3	2	40
		Thajakan	1	2	3	3	3	
		Ivan					3	
	API Security from attacker's perspective	Corsin	1	2	2	1	3	27
		Thajakan	1	1	2	1	2	
		Ivan					2	
	Tooling lab	Corsin	2	3	3	2	3	35
		Thajakan	2	2	3	1	2	
		Ivan					2	
	OAuth2 vulnerabilities	Corsin	2	2	3	2	3	39
		Thajakan	3	2	3	2	3	
		Ivan					2	
	JWT vulnerabilities	Corsin	1	3	2	2	2	32
		Thajakan	1	3	2	2	2	
		Ivan					2	
	Input validation and sanitization	Corsin	2	3	2	2	3	38
		Thajakan	2	3	2	2	2	
		Ivan					3	
	API enumeration and reconnaissance	Corsin	2	3	2	3	3	43
		Thajakan	1	3	2	3	3	
		Ivan					3	
	API monitoring	Corsin	2	2	2	3	2	38
		Thajakan	2	2	2	3	2	
		Ivan					3	

Figure 3.1: Decisionmatrix

Chapter 4

Proof of concept

This chapter defines each lab PoC before the lab development. It is used to verify the feasibility and functionality of the lab idea.

4.1 Lab PoC: API enumeration and reconnaissance

The proof of concept for API enumeration and reconnaissance is defined as follows.

4.1.1 Objectives

This PoC setup aims to demonstrate the process of API enumeration and reconnaissance using Burp Suite. In advance, tools like "Fuzz Faster U Fool" could also be used, which could be beneficial to bring students knowledge about various tools. By conducting this PoC, the feasibility of identifying and exploring APIs, understanding their endpoints, and analyzing their security implications can be showcased.

An API endpoint is a Uniform Resource Locator (URL) that acts as the point of contact between an API client and an API server. API clients send requests to API endpoints to access the APIs functionality and data [18].

4.1.2 Prerequisites:

- Basic understanding of APIs and HTTP protocols.
- Access to a target environment with APIs for testing (e.g., a demo web application with exposed APIs).

4.1.3 Equipment/Software

- Burp Suite
- Web browser (preferably Firefox or Chrome)
- Target web application with exposed APIs,

4.1.4 Setup steps

- Install and configure Burp Suite:
 - Download and install BURP Suite on a local machine
 - Configure the proxy settings in a web browser to route traffic through Burp Suite
- Identify target environment:
 - Select a target web application that exposes APIs for testing purposes.
 - Ensure that the target environment is accessible and available for testing.
- Start and verify Burp Suite:
 - Launch Burp Suite on a local machine.
 - Verify that Burp Suite’s proxy is running and listening on a specified port
 - Ensure that a web browser is configured to use Burp Suite as the proxy
- Explore target application:
 - Navigate to the target web application using a web browser
 - Interact with different functionalities of the application to identify API endpoints
- Capture traffic:
 - Use Burp Suite’s interception feature to capture HTTP requests and responses between the web browser and the target application
 - Analyze the captured traffic to identify API endpoints, parameters, and payloads
- Reconnaissance and enumeration:
 - Utilize Burp Suite’s various tools and features to enumerate and explore the discovered API endpoints systematically.
 - Collect information such as endpoint functionality, supported HTTP methods, authentication mechanisms, input validation, and error handling
- Document findings:
 - Document the discovered API endpoints, along with relevant details such as endpoint URLs, parameters, expected input/output, and any identified vulnerabilities or security issues.

4.1.5 Successful PoC

A PoC for API enumeration and reconnaissance is considered successful when it effectively identifies API endpoints, conducts thorough reconnaissance, detects security issues and documents findings.

4.1.6 PoC status

The PoC successfully demonstrated the feasibility of enumerating and conducting reconnaissance on the Juice Shop API using Burp Suite Community Edition within a Windows 11 environment. Through systematic analysis, several API endpoints were identified, along with associated parameters and potential security vulnerabilities. The tools used in Burp were Target, Proxy, Intruder and Repeater. The following topics were successfully addressed:

- Brute Force: Using the API for Brute Force attacks on user passwords is possible.
Description:
 1. Start Burp Target and open a new browser
 2. Go to the Juice Shop and log in with a standard user
 3. The login request can be seen in Figure 4.1:

Host	Method	URL	Params
http://localhost:3000	GET	/styles.css	
http://localhost:3000	GET	/Materialcons-Regular.woff2	
http://localhost:3000	GET	/rest/admin/application-configurati...	
http://localhost:3000	POST	/rest/user/login	✓
http://localhost:3000	GET	/rest/basket/6	
http://localhost:3000	GET	/rest/products/search?q=	✓
http://localhost:3000	GET	/api/Quantitys/	

Request

Pretty Raw Hex ln ≡

```

1 POST /rest/user/login HTTP/1.1
2 Host: localhost:3000
3 Content-Length: 43
4 sec-ch-ua: "Chromium";v="123", "Not:A-Brand";v="8"
5 Accept: application/json, text/plain, */*
6 Content-Type: application/json
7 sec-ch-ua-mobile: ?0
8 User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64)
  AppleWebKit/537.36 (KHTML, like Gecko) Chrome/123.0.6312.58
  Safari/537.36
9 sec-ch-ua-platform: "Windows"
10 Origin: http://localhost:3000
11 Sec-Fetch-Site: same-origin
12 Sec-Fetch-Mode: cors
13 Sec-Fetch-Dest: empty
14 Referer: http://localhost:3000/
15 Accept-Encoding: gzip, deflate, br
16 Accept-Language: de-DE,de;q=0.9,en-US;q=0.8,en;q=0.7
17 Cookie: language=en; welcomebanner_status=dismiss; cookieconsent_status
  =dismiss; continueCode=
  Mw8RL5Xbj6PnOmByiYaJKWGkxfpJuBphoOt2p0p137Q9xEozNveVg2ZrDkq4
18 Connection: close
19
20 {
  "email": "test@test.ch",
  "password": "testt"
}

```

Figure 4.1: API login request

- Request is sent to Intruder
- In the payload, the email is modified, and a payload marker is set in place of the password as shown in Figure 4.2. The payload marker is a placeholder and gets replaced with a word from a wordlist with each request issued by the Intruder.

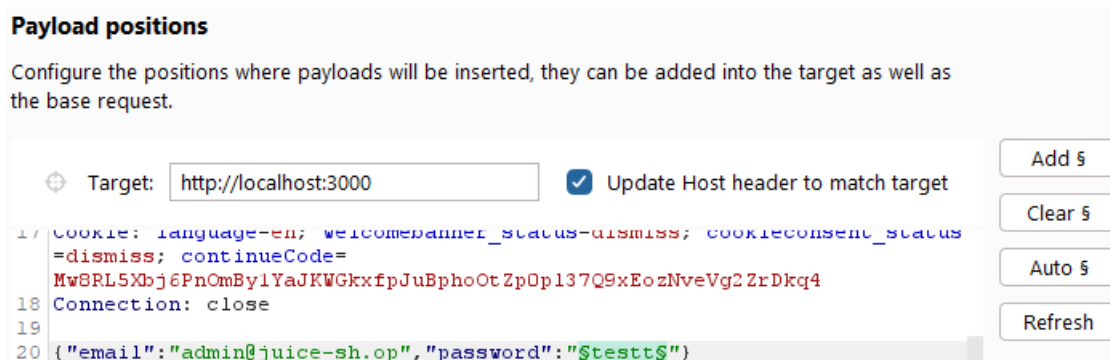


Figure 4.2: Payload positions

- Switch to payloads and paste a wordlist at the payload settings. In this example, the top 100 passwords of 2017 are used. Example payload can be seen in Figure 4.3.

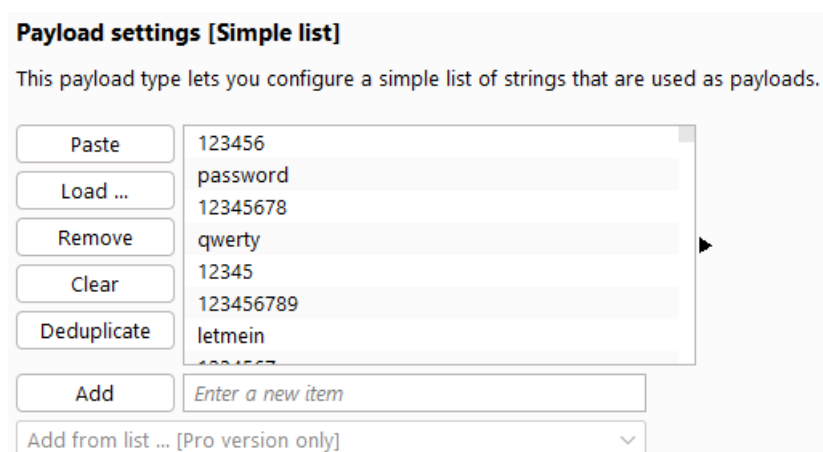


Figure 4.3: Payload settings

- Start attack and deselect status codes 4XX. After a few minutes, a request with status code 200 appears, which shows the admin user's password.
- Request manipulation: different API vulnerabilities can be exploited through request manipulation.
 - SQLInjection: an SQL injection responds to the user's JWT. The user password can then be found by comparing the password hash inside the JWT with a Rainbow table.
- Start Burp Target and open a new browser

4.2.1 Objectives

This proof of concept setup aims to demonstrate the process of implementing logging mechanisms within an API to enhance monitoring. By conducting this proof of concept, the feasibility of effectively integrating logging functionality into APIs and understanding its importance in API Security can be showcased.

4.2.2 Prerequisites

- Basic understanding of APIs and HTTP protocols
- Familiarity with a programming language commonly used for API development
- Access to a development environment for API implementation and testing

4.2.3 Equipment/Software

- IDE or text editor for coding (e.g., Visual Studio Code, IntelliJ IDEA)
- Web server environment (e.g., Node.js) for hosting the API
- Logging framework/library for the chosen programming language

4.2.4 Setup steps

- Select logging framework:
 - Choose a logging framework/library suitable for the programming language and environment used for API development
- Integrate logging framework:
 - Install and configure the selected logging framework/library within the API codebase
- Configure logging levels:
 - Customize logging levels for different components of the API to ensure appropriate granularity
- Format log entries:
 - Define a structured format for log entries to include essential metadata such as timestamp, client IP address, request method, endpoint URL, and response status code
- Deploy and test logging functionality:
 - Deploy the API to a local or test server environment

- Send test requests to the API using various scenarios to trigger different types of log events
 - Verify that log entries are generated as expected and contain relevant information
- Documentation and reporting:
 - Document the steps followed to implement logging within the API, including configuration settings, code snippets, and sample log entries

4.2.5 Successful PoC

A PoC for implementing logging is considered successful when it effectively implements logging mechanisms and provides comprehensive coverage of relevant events.

4.2.6 PoC status

The PoC successfully shows the logging implementation for a straightforward REST API.

```

1  const express = require('express');
2  const { createLogger, format, transports } = require('winston'); // Using
   Winston for logging
3  const { combine, timestamp, printf } = format;
4
5  const port = 8000;
6
7  const app = express();
8
9  // Define log format
10 const logFormat = printf(({ level, message, timestamp }) => {
11   return `${timestamp} ${level}: ${message}`;
12 });
13
14 // Create logger instance
15 const logger = createLogger({
16   format: combine(
17     timestamp(),
18     logFormat
19   ),
20   transports: [
21     new transports.Console(),
22     new transports.File({ filename: 'api.log' })
23   ]
24 });
25
26 // Middleware to log incoming requests
27 app.use((req, res, next) => {
28   logger.info(`${req.method} ${req.url}`);
29   next();
30 });

```



```

31
32
33 app.get('/', (req, res) => {
34   res.send('Get Request Received!');
35 })
36
37
38 app.post('/', (req, res) => {
39   res.send('Post Request Received!');
40 })
41
42 app.put('/', (req, res) => {
43   res.send('Put Request Received!');
44 })
45
46
47 app.delete('/', (req, res) => {
48   res.send('Delete Request Received!');
49 })
50
51 // Middleware to log outgoing responses
52 app.use((req, res, next) => {
53   res.on('finish', () => {
54     logger.info(`${res.statusCode} ${res.statusMessage}; ${res.get('
Content-Length') || 0}b sent`);
55   });
56   next();
57 });
58
59 // Error handling middleware
60 app.use((err, req, res, next) => {
61   logger.error(err.stack);
62   res.status(500).send('Internal Server Error');
63 });
64
65
66 app.listen(process.env.port || port, async() => {
67   console.log('Server Running on PORT: ${port}');
68 });

```

Code Description:

1. Dependencies:

- express: This imports the Express.js framework, which creates the server and handles HTTP requests.
- winston: This imports the Winston logging library, which provides a flexible and extensible logging mechanism for Node.js applications.

2. Logger configuration:

- The code creates a logger instance using `createLogger()` from Winston.
 - The logger is configured with a custom log format using the `combine()` function from Winston's format module. The log format includes a timestamp, log level, and message.
 - Two transports are added to the logger: one for logging to the console (`transports.Console()`) and another for logging to a file (`transports.File()`). The file transport is configured to log to a file named `api.log`.
3. Express app setup:
- An instance of the Express application is created using `express()`.
 - Middleware is added to the Express app to log incoming requests. This middleware function logs each incoming request's HTTP method (`req.method`) and URL (`req.url`).
 - Route handlers are defined for the various HTTP methods (GET, POST, PUT, DELETE). Each route handler sends a response with a message indicating the type of request received.
4. Middleware for outgoing responses:
- Another middleware function is added to the Express app to log outgoing responses. This middleware listens for the `finish` event on the response object (`res`). When the response is finished, it logs the response status code (`res.statusCode`), status message (`res.statusMessage`), and content length (`res.get('Content-Length')`) if available.
5. Error handling middleware:
- Error handling middleware is added to the Express app to log errors. If an error occurs during the processing of a request, this middleware logs the error stack trace (`err.stack`) and sends a 500 Internal Server Error response.
6. Server listening:
- The Express app starts listening on the specified port (8000) or the port specified by the `process.env.PORT` environment variable. Once the server is running, a message is logged to the console indicating the port on which the server is running.

4.3 Lab PoC: OWASP Coraza WAF

OWASP Coraza is a Web Application Firewall framework that supports ModSecurity's seclang language and is compatible with OWASP ModSecurity Core Rule Set. It is designed to enhance the security of web applications by providing real-time monitoring, logging, and protection against a wide range of common web attacks. Coraza operates as a module for web servers such as Apache and Caddy.

4.3.1 Objectives

This PoC aims to demonstrate the integration of OWASP Coraza with the Caddy server to protect the backend. This will involve setting up a basic backend website, protecting it by OWASP Coraza rules, and validating that the rules effectively mitigate common API Security threats.

4.3.2 Prerequisites

- Basic understanding of API Security concepts
- Familiarity with Caddy server and OWASP Coraza

4.3.3 Equipment/Software

- Docker
- Caddy server with OWASP Coraza module
- Backend (like juice shop)

4.3.4 Setup steps

- Setup caddy docker
- Configure OWASP Coraza module for caddy
- Configure Caddyfile
 - Define the rules for API Security (OWASP core rule set)
- Test API with and without OWASP Coraza:
 - Send requests to the API endpoint with and without OWASP Coraza enabled. Verify that OWASP Coraza effectively mitigates security threats and blocks malicious requests.
- View audit logs

4.3.5 Successful PoC

The PoC is considered successful when:

- The integration of OWASP Coraza with the Caddy server was successfully achieved.
- OWASP Coraza effectively mitigates common API Security threats
- Malicious requests are promptly blocked while legitimate requests continue to flow seamlessly, preserving API functionality
- Comprehensive logs provided by OWASP Coraza demonstrate its ability to detect and block security threats effectively.

4.3.6 PoC status

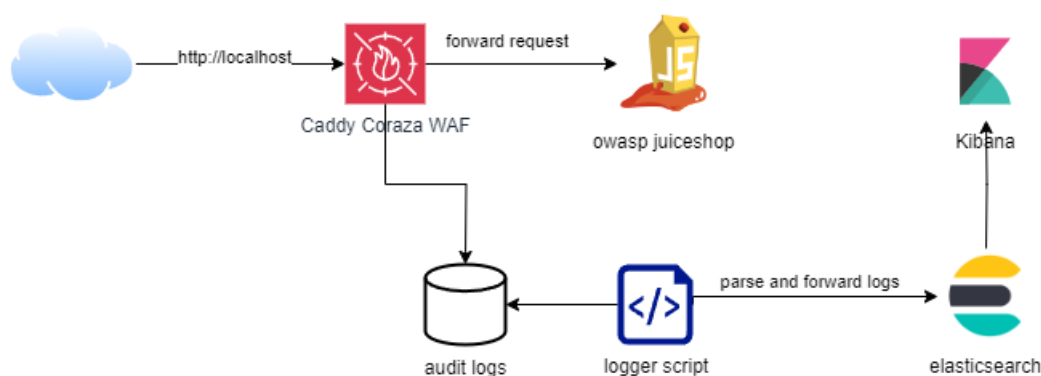


Figure 4.6: OWASP Coraza WAF PoC architecture

As shown in the Figure 4.6 the PoC was successfully implemented using docker. The setup included a Caddy, Juice Shop, Elasticsearch, Kibana and a Python logging script. The Caddy proxy was serving as a WAF by enabling the Coraza module. It also included the OWASP ModSecurity Core Rule Set. The Caddy Coraza intercepted incoming requests reverse proxy before they were forwarded to the Juice Shop. With this method, no security features need to be implemented for the backend application, which in our case was the Juice Shop. The dropped traffic from the WAF was audited in a log, which was forwarded through a Python script to an Elasticsearch instance. The elasticsearch index was then visualised through Kibana, and the audit logs were available over the web interface.

4.4 Lab PoC: OAuth2 vulnerabilities

The proof of concept for OAuth2 vulnerabilities is defined as follows.

4.4.1 Objectives

This PoC aims to assess OAuth 2.0 security vulnerabilities within a setup involving Keycloak as an identity provider and Nextcloud as the client application. This involves identifying and testing potential vulnerabilities of OAuth 2.0 authentication and authorization mechanisms [19] [20].

4.4.2 Prerequisites

- Understanding of OAuth 2.0 framework and its security implications
- Familiarity with Keycloak and Nextcloud setup and configuration

4.4.3 Equipment/Software

- Docker
- Keycloak server installed and configured as the OAuth 2.0 provider
- Nextcloud instance set up as the OAuth 2.0 client
- OAuth 2.0 client application for testing (Postman)
- Vulnerability scanning and testing tools (e.g., OWASP ZAP, Burp Suite)

4.4.4 Setup steps

- Configure Keycloak:
 - Set up the Keycloak server as the OAuth 2.0 provider. Configure realms, clients, and user accounts within Keycloak.
- Integrate Nextcloud with Keycloak
 - Configure Nextcloud to use Keycloak as the OAuth 2.0 backend for authentication and authorization.
- Test OAuth 2.0 authentication flow:
 - Use OAuth 2.0 client applications to test the authentication flow between Nextcloud and Keycloak. Verify that users can authenticate securely and obtain access tokens.
- Test for OAuth 2.0 vulnerabilities:

- Utilize vulnerability scanning and testing tools to identify potential OAuth 2.0 vulnerabilities such as authorization code leakage, token manipulation, insufficient scope validation, etc.

4.4.5 Successful PoC

The PoC is considered successful when:

- A successful integration of Nextcloud with Keycloak as the OAuth 2.0 backend.
- OAuth 2.0 authentication and authorization flows are functioning correctly.
- Vulnerability testing identifies and addresses any OAuth 2.0 vulnerabilities, ensuring the security of the authentication and authorization mechanisms.

4.4.6 PoC Status

During the PoC, it was determined that the integration of Nextcloud with Keycloak as an OAuth 2.0 provider does not offer an easy attack surface. Therefore, a simple Node.js with express was developed, which allows the vulnerabilities to be tailored to the lab without making the topic complex. The Node.js application is used for further development, and the PoC is considered successful because of the vulnerabilities that can be used.

4.5 Lab PoC: Implementing API rate limiting and throttling

The proof of concept for Implementing API rate limiting and throttling is defined as follows.

4.5.1 Objectives

This PoC aims to implement the suggested properties for a secure API regarding OWASP Top 10 API Security Risks - API4:2023 Unrestricted Resource Consumption [16]. The following limits are mentioned in the OWASP API Security Top 10:

- Execution timeouts
- Maximum allocable memory
- Maximum number of file descriptors
- Maximum number of processes
- Maximum upload file size
- Number of operations to perform in a single API client request (e.g. GraphQL batching)
- Number of records per page to return in a single request-response
- Third-party service providers' spending limit

4.5.2 Prerequisites

- Understanding of API Security principles and best practices
- Familiarity with chosen API development framework and environment: Node.js with Express

4.5.3 Equipment/Software

- Text editor or IDE for API development
- API development framework and libraries
- API testing tools (e.g., Postman, curl)

4.5.4 Setup steps

- Choose Rate Limiting/Throttling Library: Select a library compatible with your chosen API development framework that provides rate limiting and throttling functionalities.
- API Endpoint Configuration: Integrate the chosen library into API code.
- Define properties from OWASP API Security Top 10
- Test properties

4.5.5 Successful PoC

The PoC is considered successful when:

- Rate limiting and throttling are successfully integrated within the API code.
- API endpoints enforce defined rate limits and quotas.
- Testing demonstrates that exceeding limits results in appropriate error responses.
- The API remains functional and responsive within configured limits, mitigating potential resource exhaustion issues.

4.5.6 PoC Status

The implementation of the PoC was successful. Implementing the API properties mentioned in the OWASP API Security TOP 10 - API4:2023 Unrestricted Resource Consumption was possible. Nevertheless, some API properties shouldn't be restricted to the application layer and should be limited to the underlying layer (e.g., maximum allocable memory). The tests created for each applicable API property could also be tested using the Test framework. The implementation of the properties also did not affect the base functionality of the API.

4.6 Lab PoC: Implementing input validation and sanitization

The proof of concept for Implementing input validation and sanitization is defined as follows.

4.6.1 Objectives

The primary objective of this proof of concept is to demonstrate the effectiveness of input validation and sanitization in a Node.js and Express environment. By integrating the express-validator middleware, this PoC aims to ensure that all incoming data is properly validated and sanitized, thereby enhancing the security and reliability of the application.

4.6.2 Prerequisites

- Basic knowledge of Node.js and Express
- Familiarity with JavaScript and middleware concepts
- Simple Node.js application

4.6.3 Equipment/Software

- IDE or text editor for coding (e.g., Visual Studio Code, IntelliJ IDEA)
- Web server environment (e.g., Node.js) for hosting the API
- Logging framework/library for the chosen programming language

4.6.4 Setup steps

- Select input validation framework:
 - Choose an input validation framework/library suitable for the programming language and environment used for API development
- Integrate input validation and sanitization:
 - Install and configure the selected input validation framework/library within the API codebase
- Configure input validation for input fields:
 - Customize input validation to fit different criteria and use cases.

4.6.5 Successful PoC

A PoC for implementing input validation and sanitization is considered successful when it effectively implements a mechanism to validate and sanitize input along with given criteria.

4.6.6 PoC Status

The PoC successfully shows the input validation implementation for a simple JavaScript application.

```
1 import express from 'express';
2 import bodyParser from 'body-parser';
3 import { body, validationResult } from 'express-validator';
4 import Datastore from 'nedb';
5
6 const app = express();
7 const port = 3001;
8
9 const db = new Datastore({ filename: 'users.db', autoload: true });
10
11 app.use(bodyParser.json());
12 app.use(bodyParser.urlencoded({ extended: true }));
13
14 const userValidationRules = [
15   body('username')
16     .isAlphanumeric().withMessage('Username must be alphanumeric')
17     .isLength({ min: 3 }).withMessage('Username must be at least 3
18     characters long')
19     .trim().escape(),
20   body('email')
21     .isEmail().withMessage('Invalid email address')
22     .normalizeEmail(),
23   body('password')
24     .isLength({ min: 6 }).withMessage('Password must be at least 6
25     characters long')
26     .matches(/\d/).withMessage('Password must contain a number')
27     .trim().escape(),
28   body('birthdate')
29     .isDate({ format: 'YYYY-MM-DD' }).withMessage('Birthdate must be in
30     YYYY-MM-DD format')
31 ];
32
33 const validate = (req, res, next) => {
34   const errors = validationResult(req);
35   if (errors.isEmpty()) {
36     return next();
37   }
38
39   const extractedErrors = errors.array().map(err => {
40     return { [err.param]: err.msg };
41   });
42 }
```

```

40     return res.status(422).json({
41       errors: extractedErrors,
42     });
43   };
44
45   app.get('/', (req, res) => {
46     res.redirect('/register');
47   });
48
49   app.get('/register', (req, res) => {
50     res.send('
51       <form action="/register" method="post">
52         <label for="username">Username:</label><br>
53         <input type="text" id="username" name="username"><br>
54         <label for="email">Email:</label><br>
55         <input type="email" id="email" name="email"><br>
56         <label for="password">Password:</label><br>
57         <input type="password" id="password" name="password"><br>
58         <label for="birthdate">Birthdate (YYYY-MM-DD):</label><br>
59         <input type="text" id="birthdate" name="birthdate"><br><br>
60         <input type="submit" value="Register">
61       </form>
62     ');
63   });
64
65   app.post('/register', userValidationRules, (req, res) => {
66     const { username, email, password, birthdate } = req.body;
67     const errors = validationResult(req);
68     if (!errors.isEmpty()) {
69       return res.status(400).json({ errors: errors.array() });
70     }
71     const user = { username, email, password, birthdate };
72     db.insert(user, (err, newUser) => {
73       if (err) {
74         return res.status(500).json({ error: 'Failed to register user' });
75       }
76       res.send('User registered successfully');
77     });
78   });
79
80   if (process.env.NODE_ENV !== 'test') {
81     app.listen(port, () => {
82       console.log('Server is running on http://localhost:${port}');
83     });
84   }
85
86   export default app;

```

Code description

1. Dependencies

- `express`: Imports the Express.js framework to create the server and handle HTTP requests
- `body-parser`: Parses incoming request bodies in a middleware before your handlers, available under the `req.body` property
- `express-validator`: Provides a set of middleware for validating and sanitizing input data
- `nedb`: A lightweight JavaScript database for Node.js applications used for storing user data

2. Express app setup

- An instance of the Express application is created using `express()`
- Middleware is added to the Express app to parse JSON and URL-encoded request bodies using `body-parser.json()` and `body-parser.urlencoded({ extended: true })`

3. Validation rules

- **Username**: Must be alphanumeric, at least three characters long, and is sanitized to remove any unwanted characters
- **Email**: Must be a valid email address and is normalized
- **Password**: Must be at least 6 characters long, contain at least one number, and is sanitized to remove any unwanted characters
- **Birthdate**: Must be a valid date in the YYYY-MM-DD format

4. Middleware for validation

- A middleware function checks for validation errors using `validationResult()`. If errors are found, they are formatted and returned as a JSON response with a 422 status code. The request proceeds to the next middleware or route handler if no errors are found.

5. Routes

- `GET /`: Redirects to the `/register` route.
- `GET /register`: Serves as an HTML form for user registration, allowing users to input their username, email, password, and birthdate.
- `POST /register`: Handles form submissions. It validates and sanitizes input data using the predefined rules. If validation passes, the user data is inserted into the NeDB database. A 500 Internal Server Error response is returned if there is a database error. Otherwise, a success message is sent.

6. Database configuration

- A NeDB database is initialized with the filename `users.db`, and `autoload` is enabled to automatically load the database on startup.

7. Server listening

- The Express app listens on port 3001 (or a different port if specified by the `process.env.PORT` environment variable). Once the server is running, a message is logged to the console indicating the URL the server is accessible.

Chapter 5

Lab documentation

This chapter contains information about the developed resources in the Hacking-Lab and documentation for each lab created. Figure 5 shows an overview of the created labs.































#		Name	Categories	Level	Mode	Grading
1		 API Security: Implementing logging <small>c677a441-ca4e-47cc-9a16-781d1bccc1f1</small>		easy		
2		 API Security: Enumeration and reconnaissance <small>88be97cd-805b-4da3-b796-5d0269ed74cf</small>		medium		
3		 API Security: OAuth2 Vulnerabilities <small>80b957ed-1513-46c9-85a8-c067a998c6f0</small>		easy		
4		 API Security: OWASP Coraza WAF <small>bc69d9ac-309e-4055-a33c-059c1970df18</small>		medium		
5		 API Security: Rate limiting <small>4b590b0d-c5e2-4f05-e119-92f6362c1a0d</small>		medium		
6		 API Security: Input validation and sanitization <small>66bd4e25-675c-4246-907a-100d93ee8a04</small>		easy		

Figure 5.1: Hacking-Lab: Labs overview

5.1 Lab structure

The Hacking-Lab Challenge Generator [21] was used for the base structure of the lab instruction. It provides a base tool set of scripts, files and directories for the lab authors. Each lab has an introduction part at the beginning containing the following sections:

- Introduction - Describes the challenge
- Prerequisites - Student needs to solve the challenge
- Goal - Overall goal of the challenge

- Task - Overview of tasks
- Flag Format - Write-UP, Flag

The Table 5.1 will be included in all labs with the risks marked related to the challenge.

OWASP Top 10 API Security Risks – 2023	
API1	Broken Object Level Authorization
API2	Broken Authentication
API3	Broken Object Property Level Authorization
API4	Unrestricted Resource Consumption
API5	Broken Function Level Authorization
API6	Unrestricted Access to Sensitive Business Flows
API7	Server-Side Request Forgery (SSRF)
API8	Security Misconfiguration
API9	Improper Inventory Management
API10	Unsafe Consumption of APIs

Table 5.1: OWASP Top 10 API Security Risks – 2023 [16]

Lab images

The lab images were used from the Hacking-Lab Midjourney collection. <https://bit.ly/hl-images>

5.2 Lab access

The lab instructions are referenced over a link in the following documentation. To access the lab, it is needed to Sign-Up to <https://ost.hacking-lab.com> and access the lab event over the following redeem link <https://ost.hacking-lab.com/events/redeem/Y55W4-N2LJA-LBCS6-PA4G4>

5.3 Generic Hacking-Lab resources

The following resources were developed or improved for generic use and can be used for future labs.

5.3.1 Dynamic nginx Multi-Docker

The Hacking-Lab only allows one endpoint exposure in the default setup. The Multi-Docker resource is used to expose multiple services. It allows the exposure of up to 3 services as shown in the Figure 5.2 in one Docker Compose file. Unfortunately, it is handled manually and has no dynamic configuration over Environment variables or custom naming for services. [22]

Hi there

Please use the services below for this challenge!

- [Service 1](#)
- [Service 2](#)
- [Service 3](#)

Figure 5.2: Multi-Docker before

Hi there

Please use the services below for this challenge!

- [Keycloak](#)
- [Note App](#)

Figure 5.3: Multi-Docker after

To improve the User Experience and allow dynamic configuration of the exposed services, an updated version of the "nginx-multi-docker" was released. The improved resource named "dynamic-nginx-multi-docker" uses the environment variables provided in the resources Compose file to generate the nginx landing page for the lab services. Now, it is possible to configure the environment variable `URLS` to assign the hobo host-names to the according service name dynamically as shown in the Figure 5.3.

```
1 version: '2.3'
2
3 services:
4   nginx-multi-docker-hobo:
5     image: REGISTRY_BASE_URL/dynamic-nginx-multi-docker:stable
6     hostname: 'hobo'
7     networks:
8       - hobo
9     environment:
10      - "domainname=idocker.REALM_DOMAIN_SUFFIX"
11      - "SERVICES=hobo"
12      - URLS=Keycloak:hobo-1,Note App:hobo-2
13     cpus: 1
14     mem_limit: 128M
```

5.3.2 Theia Web IDE

A few labs require an IDE to solve the challenge. A "Theia Web IDE" resource was already available from previous labs. Unfortunately, the container had Node.js 12 running and an older Theia version, which was deprecated. Also, the previously used repository was archived in October 2022.[23]

Therefore, a new Docker image was created based on `node:22-alpine`. The base image needed to be alpine to install the `s6-overlay` [24], which is the default for Hacking-Lab resources to spin up services. Additionally, the build process of Theia had to be modified so that it is compatible with Alpine because Theia uses `node:18-bullseye` as the base image for the Docker build. At last, the PHP and PHP-FPM versions were upgraded.

5.4 API Security: API enumeration and reconnaissance

The Figure 5.4 shows the challenge overview on the Hacking-Lab platform from the lab API enumeration and reconnaissance containing the challenge image, OWASP Top 10 API Security Risks reference, challenge properties and resources.

The screenshot displays the challenge overview for 'API Security: Enumeration and reconnaissance' on the Hacking-Lab platform. It is divided into three main sections:

- Challenge:** Features a character image and a table of OWASP Top 10 API Security Risks - 2023. The table lists various risks, with API3 (Broken Object Property Level Authorization) and API9 (Improper Inventory Management) highlighted in green.
- Properties:** Shows challenge details such as Categories (API, Recon, Enum), Level (medium), Grading (A, B, C), Mode (Pentest), and Solution Status (GRADED 100%).
- Resources:** Lists available resources, including 'lab-api-enum-recon' (STOPPED) and 'endpoint-wordlist.txt'.

OWASP Top 10 API Security Risks - 2023	
API1	Broken Object Level Authorization
API2	Broken Authentication
API3	Broken Object Property Level Authorization
API4	Unrestricted Resource Consumption
API5	Broken Function Level Authorization
API6	Unrestricted Access to Sensitive Business Flows
API7	Server-Side Request Forgery (SSRF)
API8	Security Misconfiguration
API9	Improper Inventory Management
API10	Unsafe Consumption of APIs

Figure 5.4: API enumeration and reconnaissance challenge

The challenge can be accessed through the following link:

<https://ost.hacking-lab.com/events/1120/curriculumevents/1121/challenges/7798>

5.4.1 Descriptive information

Learning goal

In this lab, students learn how to use Burp Suite. The main functions used will be Target, Intruder and Repeater. A website that has intentionally built-in vulnerabilities will be investigated using these tools. Particular attention will be paid to the API interfaces and endpoints.

Lab duration

90 min

OWASP Top 10 API Security Risks relation

The website is examined with the help of Burp Suite. Vulnerabilities are also exploited. These vulnerabilities are affected by "Broken Object Property Level Authorization".

An attacker can supply specific API endpoints with incorrect values and exploit these vulnerabilities without proper privileges.

API3: Broken Object Property Level Authorization

Broken Object Property Level Authorization issues arise because authorization checks are not fine-grained enough to control access within an object's property level. This can lead to unauthorized access or modification of sensitive data, such as roles, permissions, payment information, or personal details.

API9: Improper Inventory Management

This vulnerability occurs when an application does not properly manage and validate inventory-related operations, such as item quantities and stock levels, leading to potential exploitation and business logic issues.

5.4.2 Lab development

Architecture

The challenge in the Hacking-Lab provides a web application with a Juice Shop as a resource. This web application can be accessed with the browser via HTTPS. In the lab, a pre-installed Kookarai with Burp Suite is recommended.

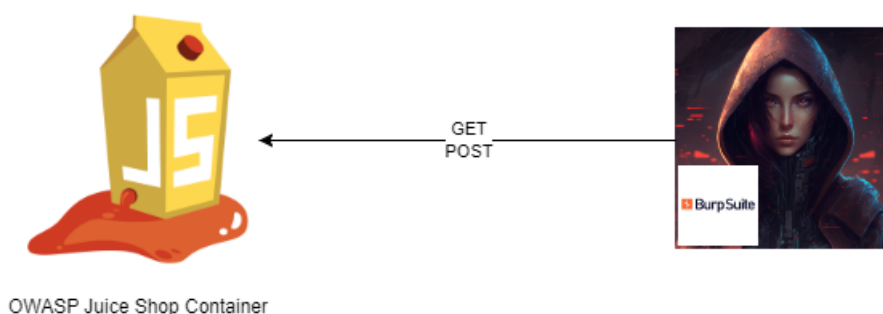


Figure 5.5: Architecture enumeration and reconnaissance [17] [25] [26]

Development

Enumeration and reconnaissance are major steps in the process of security testing. They serve to gather as much information as possible about the target system, enabling a deeper understanding and identifying potential vulnerabilities that could be exploited. This lab provides various tools and offers support with instructions to assist in the execution.

Burp Suite Community Edition

Burp Suite is a comprehensive web vulnerability scanner and security testing tool used by penetration testers and security professionals. It provides features for scanning,

mapping, and analyzing web applications to identify security vulnerabilities. Key components include the Proxy for intercepting and modifying web traffic, the Scanner for automated vulnerability detection, the Intruder for custom attacks, and the Repeater for manual testing of requests.

OWASP Juice Shop

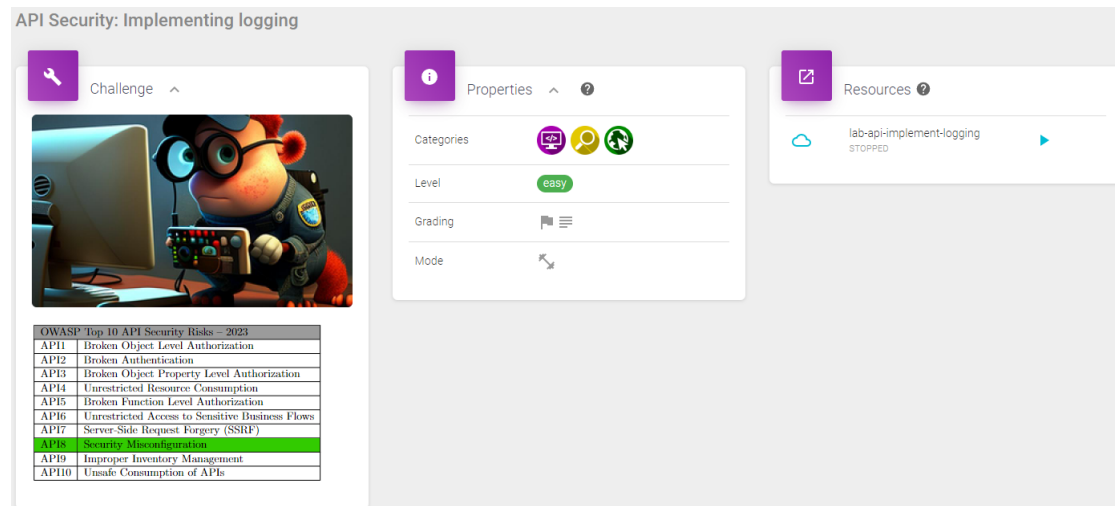
The OWASP Juice Shop is the example application that needs to be enumerated. Therefore, the existing `juice-shop` image is used.

5.4.3 Lab solution

The lab is considered complete after the student follows all the instructions and submits a flag and a report with the answers to the provided questions.

5.5 API Security: Implementing logging

The Figure 5.6 shows the challenge overview on the Hacking-Lab platform from the lab Implementing logging containing the challenge image, OWASP Top 10 API Security Risks reference, challenge properties and resources.



The screenshot displays the challenge overview for 'API Security: Implementing logging' on the Hacking-Lab platform. It is divided into three main sections:

- Challenge:** Features a character wearing glasses and a headset, and a table of OWASP Top 10 API Security Risks - 2023. The risk API8, Security Misconfiguration, is highlighted in green.
- Properties:** Lists the challenge's characteristics: Categories (API, OWASP, Security), Level (easy), Grading (points), and Mode (interactive).
- Resources:** Shows a resource named 'lab-api-implement-logging' with a 'STOPPED' status and a play button icon.

OWASP Top 10 API Security Risks - 2023	
API1	Broken Object Level Authorization
API2	Broken Authentication
API3	Broken Object Property Level Authorization
API4	Unrestricted Resource Consumption
API5	Broken Function Level Authorization
API6	Unrestricted Access to Sensitive Business Flows
API7	Server-Side Request Forgery (SSRF)
API8	Security Misconfiguration
API9	Improper Inventory Management
API10	Unsafe Consumption of APIs

Figure 5.6: Implement logging challenge

The challenge can be accessed through the following link:

<https://ost.hacking-lab.com/events/1093/curriculumevents/1122/challenges/7801>

5.5.1 Descriptive information

Learning goal

In this lab, students work on a lab that includes Theia IDE from Hacking-Lab [27]. This allows the editing of JavaScript files directly in the lab environment and running them afterwards. The goal is to implement and configure an Express middleware called Winston into an existing web application.

Lab duration

90 min

OWASP Top 10 API Security Risks relation

Maintaining comprehensive and secure logging practices can significantly enhance your ability to protect APIs, detect threats early, and respond effectively to security incidents.

API8: Security Misconfiguration

Proper logging can help identify security misconfigurations. Logs may reveal unusual or unauthorized access attempts, errors in configuration settings, or other indicators that an API is not properly secured. These misconfigurations can go unnoticed without adequate logging, leaving the API vulnerable.

5.5.2 Lab development

Architecture

The Figure 5.7 shows the lab architecture.

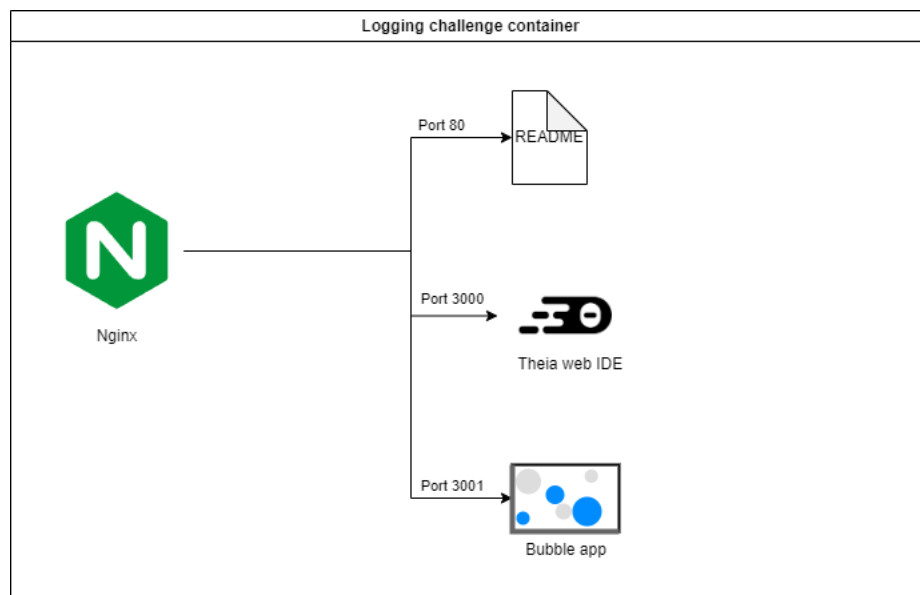


Figure 5.7: Architecture overview logging lab

Development

The lab container is based on the Theia Web IDE 5.3.2 resource and uses a nginx to expose the following services:

README

The README is the landing page when the student starts the container. It contains essential information about the lab services, how to use the Theia Web IDE and how to start the Bubble app.

Theia Web IDE

Theia Web IDE provides the web IDE for the students to develop and start the Bubble sample app.

Bubble app

The Bubble app is a simple JavaScript application. It provides two API endpoints. One is called when a user presses the "create-bubble" button. This creates a bubble with a random counter between 1 and 5. The user can press on the bubbles, which decreases the affected counter by 1. As soon as the counter reaches 0, the bubble is deleted. This is executed by the "delete-bubble" endpoint. Following Figure 5.8 shows the app.

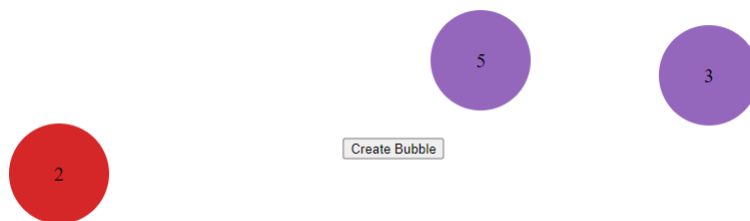


Figure 5.8: Bubble app

5.5.3 Lab solution

The lab is considered complete after the student follows all the instructions and submits a report that contains JavaScript code.

5.6 API Security: OWASP Coraza WAF

The Figure 5.9 shows the challenge overview on the Hacking-Lab platform from the lab OWASP Coraza WAF containing the challenge image, OWASP Top 10 API Security Risks reference, challenge properties and resources.

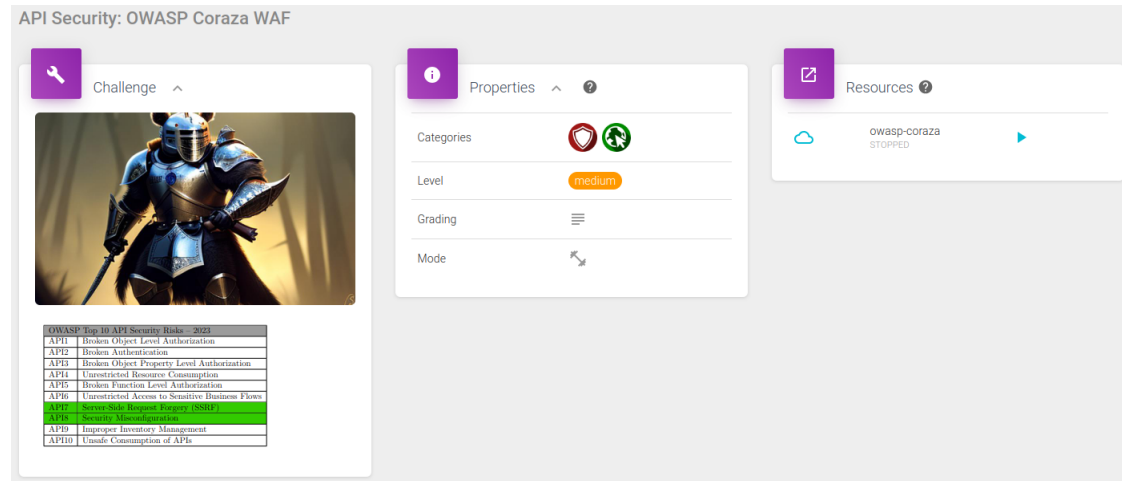


Figure 5.9: OWASP Coraza WAF challenge

The challenge can be accessed through the following link:

<https://ost.hacking-lab.com/events/1120/curriculumevents/1121/challenges/7800>

5.6.1 Descriptive information

Learning goal

The lab aims to introduce students to OWASP Coraza WAF within the Caddy server setup. Through practical exercises, participants will grasp how Coraza WAF defends web applications from typical vulnerabilities. They'll also gain firsthand experience configuring, deploying, and assessing its effectiveness against web API Security threats.

Lab duration

60 min

OWASP Top 10 API Security Risks relation

OWASP Coraza is a web application firewall that can help mitigate API Security risks outlined in the API7: Server Side Request Forgery and API8: Security Misconfiguration.

API7: Server Side Request Forgery

OWASP Coraza helps mitigate SSRF by:

- Input Validation: Enforcing strict validation rules for user-supplied URLs.
- Whitelisting/Blacklisting: Allowing only trusted domains and blocking malicious ones.
- Disabling HTTP Redirections: Blocking redirection responses.
- Media Type Enforcement: Ensuring only expected media types are accepted.
- URL Parsing: Implementing robust URL parsing and validation.
- Rate Limiting: Limiting the number of sensitive requests.
- Logging/Monitoring: Logging suspicious activity and alerting administrators.

API8: Security Misconfiguration

OWASP Coraza helps mitigate Security Misconfiguration by:

- Secure configurations such as TLS encryption.
- Proper CORS (Cross-Origin Resource Sharing) policies.
- Restricted HTTP methods.
- Sanitized error handling practices.

5.6.2 Lab development

Architecture

The Figure 5.10 shows the lab architecture.

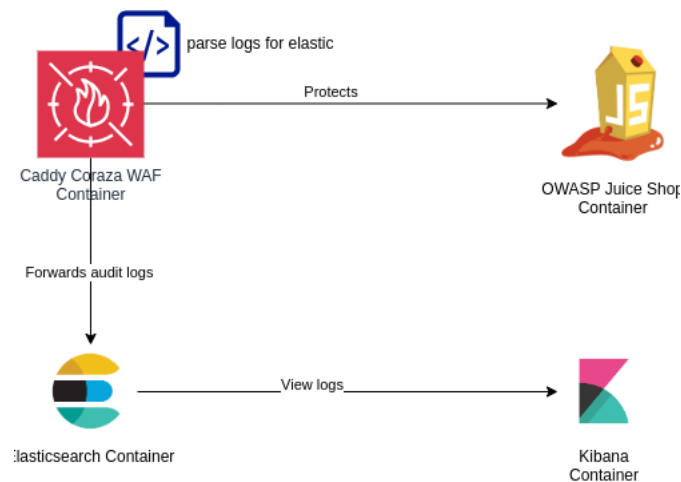


Figure 5.10: Architecture overview OWASP Coraza WAF lab

Development

The Medium article inspired the lab idea: OSS WAF stack using Coraza, Caddy, and Elastic [28].

The lab consists of five containers: Dynamic nginx Multi-Docker, Coraza Caddy, Elasticsearch, Kibana and OWASP Juice Shop.

Dynamic nginx Multi-Docker

This container provides a simple nginx landing page containing all the links to the lab resources since the Hackin-Lab link only allows the reference of one URL for each resource.

Coraza Caddy

OWASP Coraza is configured over a caddy module to provide caddy web application firewall capabilities. The custom Dockerfile builds caddy with the Coraza WAF enabled, installs the s6-overlay and copies the needed files. This contains the Caddyfile, which loads the Coraza WAF modules and enables the audit log and `logger.py`, a custom script to watch the audit logs, parse and send them to the Elasticsearch instance.

Elasticsearch

The Elasticsearch container uses the default image `elasticsearch:7.17.20` and is used to collect the logs from the Coraza Caddy WAF.

Kibana

Kibana uses the default `kibana:7.17.20` image and is used to visualize the logs from the WAF.

OWASP Juice Shop

The OWASP Juice Shop is the example application that needs to be protected. Therefore, the existing `juice-shop` image is used.

Compose file

The following Compose file is deployed. (Filtered to only image and environment variable)

```
1 version: '2.3'
2
3 services:
4   nginx-multi-docker-hobo:
5     image: REGISTRY_BASE_URL/dynamic-nginx-multi-docker:stable
6     environment:
7       - URLS=JuiceShop:hobo-1,Kibana:hobo-3
8
9   owasp-coraza-waf-hobo-1:
10    image: REGISTRY_BASE_URL/multi-docker-coraza-caddy:stable
11    environment:
12      - "BACKEND_APPLICATION=hobo-4.i.vuln.land"
```

```
13     - "BACKEND_APPLICATION_PORT=3000"
14
15     owasp-coraza-waf-hobo-2:
16         image: REGISTRY_BASE_URL/multi-docker-coraza-kibana:stable
17
18     owasp-coraza-waf-hobo-3:
19         image: REGISTRY_BASE_URL/demo-docker-shell-chatgpt-3-sa:stable
20
21     owasp-coraza-waf-hobo-4:
22         image: REGISTRY_BASE_URL/juice-shop:stable
```

Flow

Any incoming vulnerable request to the OWASP Juice Shop will be captured and blocked by the Coraza Caddy WAF instance. The log script will detect the audit logs generated by the blocked requests, which process and send the data to the Elasticsearch instance. Finally, Kibana will be able to visualize the logs for the student and read the coraza index from Elasticsearch.

5.6.3 Lab solution

The lab is considered complete after the student follows all the instructions and submits a report with the answers to the provided questions.

5.7 API Security: OAuth2 vulnerabilities

The Figure 5.11 shows the challenge overview on the Hacking-Lab platform from the lab OAuth2 vulnerabilities containing the challenge image, OWASP Top 10 API Security Risks reference, challenge properties and resources.

The screenshot displays the challenge overview for 'API Security: OAuth2 Vulnerabilities' on the Hacking-Lab platform. It is divided into three main sections:

- Challenge:** Features a cartoon character in a police uniform and a table of OWASP Top 10 API Security Risks - 2023. The table lists various risks, with API2 (Broken Authentication) highlighted in green.
- Properties:** A panel showing challenge details: Categories (Security, API), Level (easy), Grading (Points), Mode (Interactive), and Solution Status (GRADED 20%).
- Resources:** A panel listing related resources: 'lab-oauth2-vulnerabilities-multidocker' (STOPPED) and 'lab-oauth2-vulnerabilities-client.json'.

OWASP Top 10 API Security Risks - 2023	
API1	Broken Object Level Authorization
API2	Broken Authentication
API3	Broken Object Property Level Authorization
API4	Unrestricted Resource Consumption
API5	Broken Function Level Authorization
API6	Unrestricted Access to Sensitive Business Flows
API7	Server-Side Request Forgery (SSRF)
API8	Security Misconfiguration
API9	Improper Inventory Management
API10	Unsafe Consumption of APIs

Figure 5.11: OAuth2 Vulnerabilities challenge

The challenge can be accessed through the following link:

<https://ost.hacking-lab.com/events/1120/curriculumevents/1121/challenges/7799>

5.7.1 Descriptive information

Learning goal

The primary goal of this challenge is to identify and exploit a vulnerability in the OAuth2 authentication flow to impersonate another user. By achieving this, the students will:

- Learn how misconfigurations or incorrect implementations in authentication can be exploited.
- Understand the importance of secure coding practices in authentication mechanisms.
- Gain practical experience in analyzing and exploiting web application vulnerabilities.

Lab duration

60min

OWASP Top 10 API Security Risks relation

The lab focuses on API2: Broken Authentication because it targets a vulnerability in the OAuth2 implementation used for user login within a web application.

- API2: Broken Authentication deals with weaknesses in how APIs identify and verify users.
- The lab specifically mentions analyzing the OAuth2 login flow, a core function of user authentication in APIs.
- The goal is to exploit a flaw and impersonate another user, bypassing the intended access control mechanism. This indicates a serious failure in the authentication process.

The lab highlights how a broken authentication implementation allows unauthorized access to the API by impersonating a user "john".

5.7.2 Lab development

Architecture

The Figure 5.12 shows the lab architecture.



Figure 5.12: Architecture overview OAuth2 vulnerabilities lab

Development

The lab consists of three containers: Dynamic nginx Multi-Docker, Notes application and Keycloak.

Dynamic nginx Multi-Docker

This container provides a simple nginx landing page containing all the links to the lab resources since the Hackin-Lab link only allows the reference of one URL for each resource.

Notes App

This Node.js application is a simple notes app that uses Keycloak for user authentication and Express for handling HTTP requests."It uses sessions to handle user state and JWTs to secure communications."

When a user accesses the root route (/), the application checks if they are authenticated. If not, it redirects them to Keycloak for authentication. Upon successful login, Keycloak redirects back to the application with an authorization code. The application exchanges this code for access tokens in the callback route (/callback). These tokens are stored in the session to maintain the authenticated state. The figure 5.13 shows the login page of the Notes App.

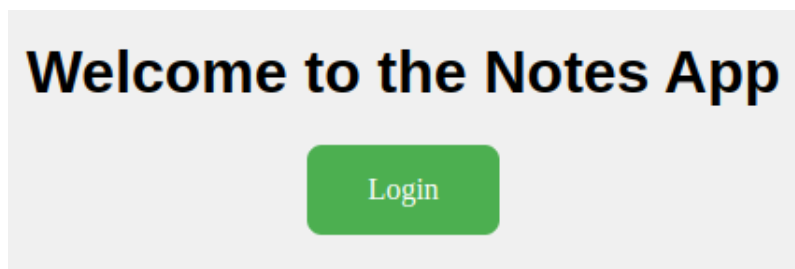


Figure 5.13: Notes App: Welcome page

Once authenticated, users are greeted with a personalized welcome page, which prompts them to view or add notes. The application decodes the JWT token to extract user information, ensuring the session contains the correct user data. This allows the application to display user-specific notes and provide an interface for adding new notes.

The application manages notes by retrieving and displaying user-specific notes stored in an in-memory object. Users can add new notes through the interface as shown in the Figure 5.14, which are saved to this in-memory store. This ensures that each user can manage their notes independently.

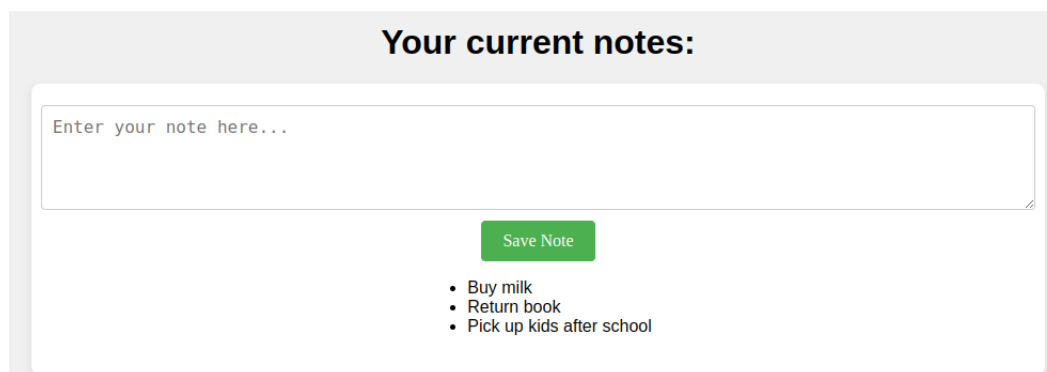


Figure 5.14: Notes App: Notes

Access control is enforced throughout the application. Only authenticated users can access routes that involve note management. This is checked by verifying the presence of access tokens in the session before allowing access to sensitive routes.

```

1 app.post('/login', async (req, res) => {
2   const { username, accessToken } = req.body;
3
4   if (username && accessToken) {
5     ACTIVE_USER = username;
6     res.send('Login successful!');
7   } else {
8     res.send('Error: no username or accessToken is defined');
9   }
10 });

```

The vulnerability in the POST /login route is primarily due to the insecure handling of user authentication. Specifically, the route sets the active user (ACTIVE_USER) based solely on the provided `username` and `accessToken` without proper validation. This allows any user who sends a POST request with arbitrary values for `username` and `accessToken` to impersonate any user. This lack of proper authentication checks can lead to unauthorized access and potential session hijacking, where an attacker can access another user's notes and personal data without proper authorization.

Keycloak

Keycloak is used in this application to manage user authentication and authorization securely. It handles the login process, providing a centralized and secure way to authenticate users. When users attempt to log in, they are redirected to Keycloak, which verifies their credentials and issues the authorization token. The application then uses the token to maintain user sessions and verify identities.

5.7.3 Lab solution

The lab is considered solved after the student has successfully impersonated the user "john", where the flag is stored in his notes as seen in the Figure 5.15 and answered the provided questions in the write-up.

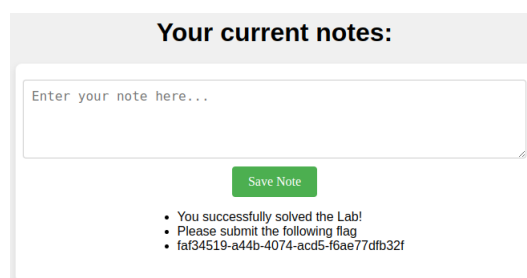


Figure 5.15: Notes App: Solution

5.8 API Security: Rate limiting

The Figure 5.16 shows the challenge overview on the Hacking-Lab platform from the lab Rate limiting containing the challenge image, OWASP Top 10 API Security Risks reference, challenge properties and resources.

The screenshot displays the challenge overview for 'API Security: Rate limiting' on the Hacking-Lab platform. It is divided into three main sections:

- Challenge:** Features a character holding a laptop and a table of OWASP Top 10 API Security Risks (2022). The risk 'API5 Unrestricted Resource Consumption' is highlighted in green.
- Properties:** Lists the challenge's characteristics: Categories (API, Security), Level (medium), Grading (Flag), and Mode (Puzzle).
- Resources:** Shows a resource named 'lab-api-rate-limiting' with a 'STOPPED' status and a play button.

OWASP	Top 10 API Security Risks - 2022
API1	Broken Object Level Authorization
API2	Broken Authentication
API3	Broken Object Property Level Authorization
API4	Unrestricted Resource Consumption
API5	Broken Function Level Authorization
API6	Unrestricted Access to Sensitive Business Flows
API7	Server-Side Request Forgery (SSRF)
API8	Security Misconfiguration
API9	Improper Inventory Management
API10	Unsafe Consumption of APIs

Figure 5.16: Rate limiting challenge

The challenge can be accessed through the following link:

<https://ost.hacking-lab.com/events/1120/curriculumevents/1121/challenges/7885>

5.8.1 Descriptive information

Learning goal

The primary goal of this challenge is to secure a basic Library app by implementing best practices to mitigate Unrestricted Resource Consumption vulnerabilities. By achieving this, students will:

- Learn how to prevent excessive resource usage in APIs.
- Understand the importance of rate limiting and resource management.
- Gain practical experience in applying security measures to protect web APIs.

Lab duration

90min

OWASP Top 10 API Security Risks relation

The lab is built on API4:2023 Unrestricted Resource Consumption and directly handles the API recommendation for this field. It focuses on each API property mentioned in the OWASP Top 10 API Security Risks and provides information on how and why it needs to be considered.

5.8.2 Lab development

Architecture

This Figure 5.23 shows the lab architecture.

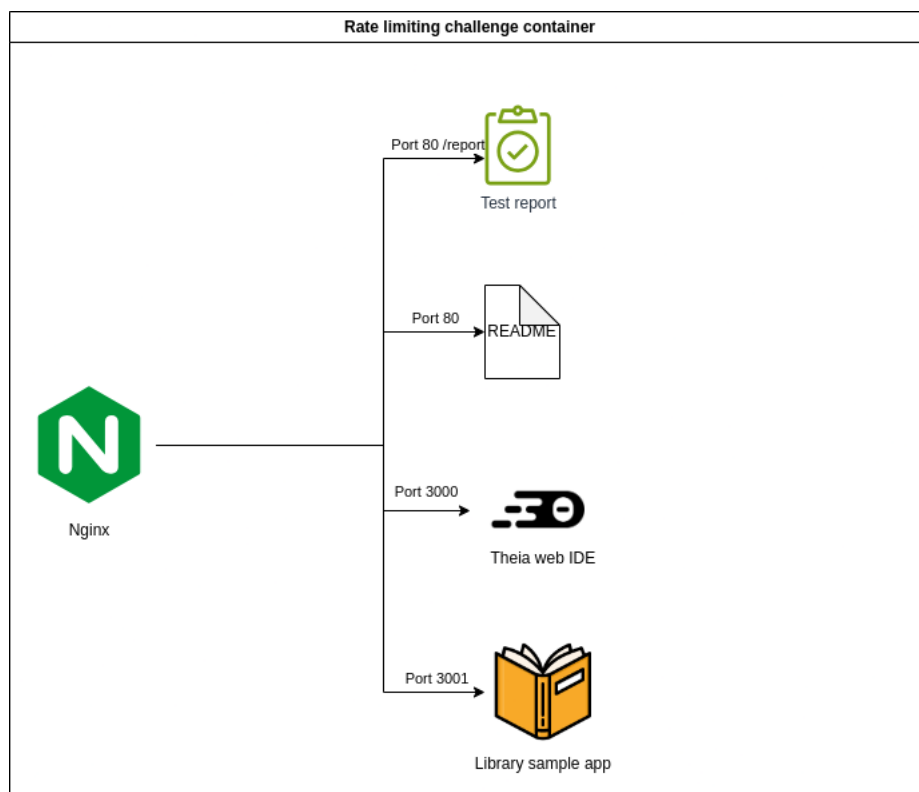


Figure 5.17: Architecture overview Rate limiting lab

Development

The lab container is based on the Theia Web IDE 5.3.2 resource and uses an nginx to expose the following services:

README

The README is the landing page when the student starts the container. It contains

important information about the lab services, how to use the Theia Web IDE, how to start the Library sample app and how the Test report page works.

Theia Web IDE

Theia Web IDE provides the web IDE for the students to develop and start the Library sample app.

Library sample app

The library application, as shown in Figure 5.18, allows users to lend and return books. The application provides several endpoints for interacting with the book database but lacks important resource management features.

The Library API consists of the following main functionalities:

- Uploading books: Users can upload books through a JSON file or directly in the request body.
- Lending books: Users can lend books by specifying the book ID.
- Returning books: Users can return lent books by specifying the book ID.
- Viewing books: Users can view the list of books.

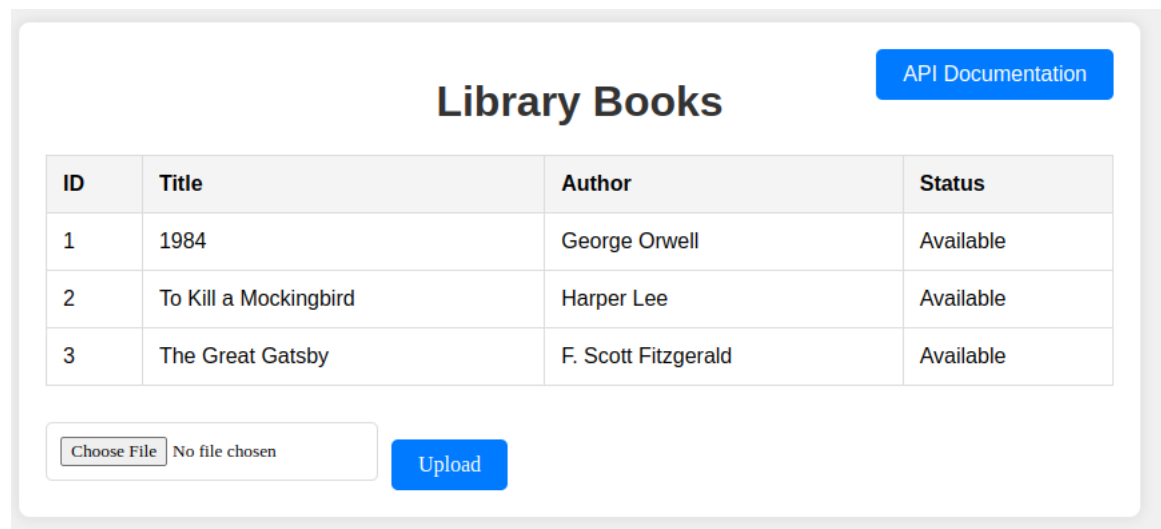


Figure 5.18: Library sample app

It uses Swagger [29] to document the API endpoints and provide base information about how to consume the API with an execution possibility. The following Figure 5.19 shows a Swagger overview.

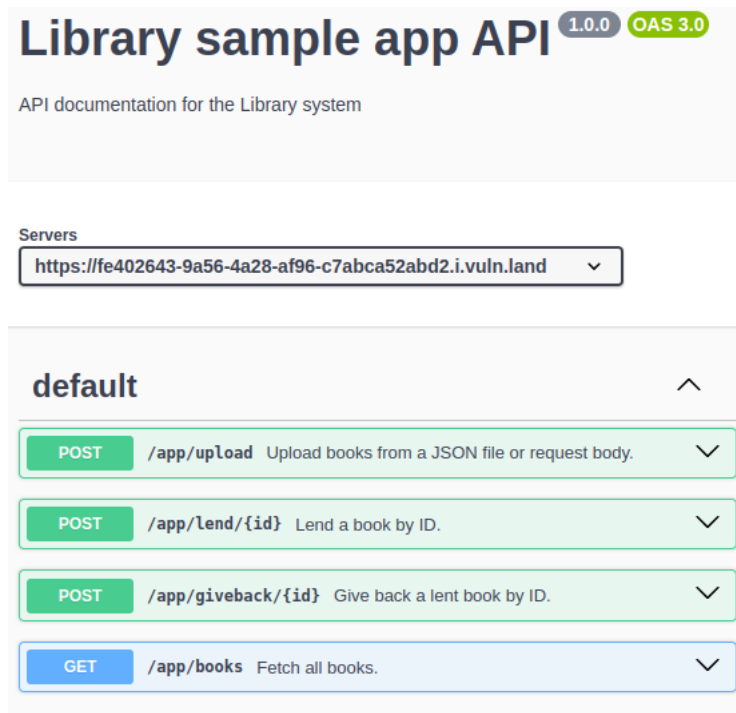


Figure 5.19: Library sample app

Test report

Jest, a JavaScript Testing framework, handles the code validation. Several assertions were created with `supertest` to verify each API property. The report page is generated by `jest-html-reporter` and exposed through `nginx`.

A shell script runs every minute to execute the tests and generate the report for the students. When all tests are successful, the dynamic flag will be imported into the HTML of the report page. Following a Figure 5.20 with unsuccessful tests.

Library App Tests

Started: 2024-06-09 13:51:02

Suites (1)

0 passed
1 failed
0 pending

Tests (7)

1 passed
6 failed
0 pending

~ /opt/nodejs/tests/example.test.js				7.565s
Test execution timeout	should return 503 status code if the request times out	failed	6.002s	
Maximum upload file size	should return 400 when file size exceeds limit	failed	0.03s	
Book upload limit	should reject requests with more books 10 books	failed	0.015s	
Book upload limit	should accept requests with a number of books below 9	passed	0.005s	
Pagination tests	should return the first page of books by default	failed	0.007s	
Pagination tests	should return the second page of books	failed	0.004s	
Rate Limiting	should allow up to 100 requests per 15 minutes	failed	0.195s	

Figure 5.20: Library app: Test report

5.8.3 Lab solution

The lab is considered solved after the student has successfully implemented all API properties from the API4:2023 Unrestricted Resource Consumption and submitted the flag after all tests were successful. Following a Figure 5.21 with all tests successful.

Maximum upload file size	should return 400 when file size exceeds limit	passed	0.03s
Book upload limit	should reject requests with more books 10 books	passed	0.014s
Book upload limit	should accept requests with a number of books below 9	passed	0.004s
Pagination tests	should return the first page of books by default	passed	0.006s
Pagination tests	should return the second page of books	passed	0.004s
Rate Limiting	should allow up to 100 requests per 15 minutes	passed	0.226s

CONGRATS YOU SOLVED THE LAB, PLEASE COMMIT THE FOLLOWING FLAG: 4ab9a724-1386-47d4-ba62-53642a20af4a

Figure 5.21: Library app: Test report solution

5.9 API Security: Input validation and sanitization

The Figure 5.22 shows the challenge overview on the Hacking-Lab platform from the lab Input validation and sanitization containing the challenge image, OWASP Top 10 API Security Risks reference, challenge properties and resources.

The screenshot displays the challenge overview on the Hacking-Lab platform. It is divided into three main sections:

- Challenge:** Features a character image and a table of OWASP Top 10 API Security Risks - 2023. The table lists risks from API1 to API10, with API10 'Unsafe Consumption of APIs' highlighted in green.
- Properties:** A panel showing challenge details: Categories (with icons for API, OWASP, and Security), Level (easy), Grading (with icons for a list and a document), Mode (with a key icon), and Solution Status (GRADED 20%).
- Resources:** A panel showing a resource titled 'lab-api-implement-input-validation' with a 'STOPPED' status and a play button icon.

Figure 5.22: Input validation and sanitization challenge

The challenge can be accessed through the following link:

<https://ost.hacking-lab.com/events/1120/curriculumevents/1121/challenges/7886>

5.9.1 Descriptive information

Learning goal

The goal of this lab is to implement input validation and sanitization. Therefore, a simple input form for user registration is provided. Since students must meet specific criteria for a validation service to pass successfully, they must correctly implement the Express middleware `express-validator`.

Lab duration

90 min

OWASP Top 10 API Security Risks relation

This lab is about implementing input validation and sanitization, which serves as a countermeasure for unsafe API consumption. By learning this, students can prevent vulnerabilities such as injection attacks, data corruption, and unauthorized access, thereby

ensuring the security and integrity of their applications when interacting with external APIs.

API10: Unsafe Consumption of APIs

This risk occurs when applications consume APIs without adequately handling the input or output, leading to various security issues such as injection attacks, data leaks, or compromised systems.

5.9.2 Lab development

Architecture

This Figure 5.23 shows the lab architecture.

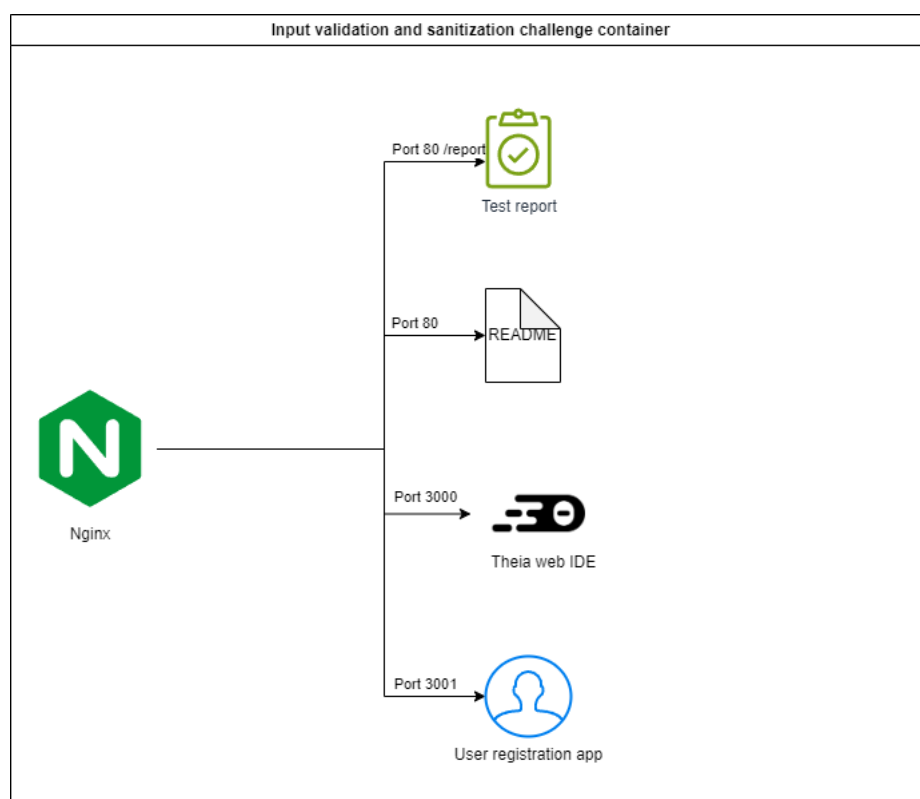


Figure 5.23: Architecture overview input validation lab

Development

The lab container is based on the Theia Web IDE 5.3.2 resource and uses a nginx to expose the following services:

README

The README is the landing page when the student starts the container. It contains essential information about the lab services, how to use the Theia Web IDE and how to start the user registration app.

Theia Web IDE

Theia Web IDE provides the web IDE for the students to develop and start the user registration sample app.

User registration app

The user registration app provides input fields that can be implemented with express-validator. This allows specific input validation and sanitization. See Figure 5.24



Username:

Email:

Password:

Birthdate (DD-MM-YYYY):

Figure 5.24: User registration app

5.9.3 Lab solution

The lab is considered complete after the student follows all the instructions and the validation service can complete all tests. The student must then submit a flag and his code.

Chapter 6

Quality Measures

The following chapter contains requirements and test concept descriptions for the evaluating labs afterwards.

6.1 General lab requirements

The following general requirements are defined for the labs.

- **Consumer** - The target audience of the lab should be computer science students who want to learn about topics in API Security.
- **Language** - English will be the general language for all lab instructions.
- **Grading** - The lab should have a Write-Up, Flag or Write-Up + Flag as submission to validate the challenge.
- **Platform** - The lab components should be fully compatible and functional on the Kookarai Pentesting Linux virtual machine.

6.2 Test concept

This section defines the test concept of how the labs will be tested. It describes the test plan and defines the requirements of the test cases.

Testing approach

The testing will be done in three phases.

- **Author testing** - The lab author defines the test cases before developing the lab and tests them afterwards.
- **Partner testing** - The team partner of the author will do this testing to verify the outcome of the Test cases.

- **Public testing** - This testing will be done by a 3rd party user, which has no relation to the topic. Hereby, the focus lies on the user experience and general understanding of the lab.

Testing Schedule

The tests will be done immediately after finishing the lab and will follow the phases defined in the testing approach section.

Test deliverables

- **Author testing:** The test report will be the deliverable for this phase.
- **Partner testing:** The test report will be the deliverable for this phase.
- **Public testing:** The feedback form will be the deliverable for this phase.

6.2.1 Roles and responsibilities

- **Test manager (Thajakan Thirunavukkarasu)** - Is responsible for monitoring the test and verifying that all the phases are validated properly with the corresponding quality.
- **Tester (Corsin Salutt & Thajakan Thirunavukkarasu)** - Are responsible for performing the tests at the author testing and partner testing phase.
- **User acceptance tester** - Provide feedback on usability, functionality, and overall user experience.

Test cases

The test cases are defined using the following template.

TC-ID	Unique identifier for each test case
Lab	Lab name, where it belongs to
Description	Description of the test case
Precondition	Any preconditions that must be met before the test case can be executed
Test steps	Step-by-step instructions for executing test case
Expected result	The expected outcome or behaviour of the test
Environment	Details of the test environment
Priority	high/medium/low
Status	pass/fail/NA
Notes	Any additional notes

Table 6.1: Test Case template

6.3 Defined test cases

This chapter includes all the test cases defined for the individual labs.

6.3.1 Test cases: API enumeration and reconnaissance

TC-ID	TC-1-1
Lab	API enumeration and reconnaissance
Description	API-enum-recon resource can be started and is accessible
Precondition	User is logged in to the HL
Test steps	1. Start the API-enum-recon docker resource 2. Access the webpage of the resource
Expected result	Resource can be started and accessed
Environment	The test environment is the Hacking-Lab Challenge page.
Priority	high
Status	pass/fail/NA
Notes	-

Table 6.2: TC-1-1: API enumeration and reconnaissance

TC-ID	TC-1-2
Lab	API enumeration and reconnaissance
Description	Burp can interact with the HL resource
Precondition	HL resource is running and accessible. Kookarai is started and Burp is open
Test steps	1. Go to Burp Target and open a new browser 2. Access the FQDN of the HL resource
Expected result	Traffic can be seen in Burp. Burp can send and retrieve requests from the resource.
Environment	The test environment is the Hacking-Lab Challenge page and Kookarai OS.
Priority	high
Status	pass/fail/NA
Notes	-

Table 6.3: TC-1-2: API enumeration and reconnaissance

6.3.2 Test cases: Implementing logging

TC-ID	TC-2-1
Lab	Implementing logging
Description	Readme is accessible
Precondition	User is logged in to the HL
Test steps	1. Start the Implementing logging docker resource 2. Access the webpage of the resource, which should show the lab readme page
Expected result	Readme page can be accessed
Environment	The test environment is the Hacking-Lab Challenge page.
Priority	high
Status	pass/fail/NA
Notes	-

Table 6.4: TC-2-1: Implementing logging

TC-ID	TC-2-2
Lab	Implementing logging
Description	Theia instance is accessible
Precondition	User is logged in to the HL
Test steps	1. Start the Implementing logging docker resource 2. Access the webpage of the resource, which should show the labs readme page 3. From the readme page, access the Theia instance
Expected result	Theia instance is accessible
Environment	The test environment is the Hacking-Lab Challenge page.
Priority	high
Status	pass/fail/NA
Notes	-

Table 6.5: TC-2-2: Implementing logging

TC-ID	TC-2-3
Lab	Implementing logging
Description	Theia instance works properly
Precondition	User is logged in to the HL
Test steps	1. Access the Theia instance 2. Open, edit and save files in the root directory 3. Start the terminal and execute npm start
Expected result	Theia instance works properly
Environment	The test environment is the Hacking-Lab Challenge page.
Priority	high
Status	pass/fail/NA
Notes	-

Table 6.6: TC-2-3: Implementing logging

6.3.3 Test cases: OWASP Coraza WAF

The test cases are defined using the following template.

TC-ID	TC-3-1
Lab	OWASP Coraza WAF
Description	Exposed services in multi-service docker are available over the browser.
Precondition	Multi-service docker resource is deployed and linked to the challenge
Test steps	1. Start the coraza-waf multi-service docker resource 2. Navigate to the nginx page exposing the services from the multi-service setup 3. Click on all Service1 and Service2
Expected result	Service 1 should open up the vulnerable backend application, and Service2 should open Kibana.
Environment	The test environment is the Hacking-Lab Challenge page.
Priority	high
Status	pass/fail/NA
Notes	-

Table 6.7: TC-3-1: OWASP Coraza WAF

TC-ID	TC-3-2
Lab	OWASP Coraza WAF
Description	Logger script does create the elastic index automatically.
Precondition	Multi-service docker resource is deployed and linked to the challenge
Test steps	1. Start the coraza-waf multi-service docker resource 2. Run a vulnerable curl command on Service1 3. Check in Kibana if the index is automatically created.
Expected result	Under Kibana settings, it should show the "coraza" index.
Environment	The test environment is the Hacking-Lab Challenge page.
Priority	high
Status	pass/fail/NA
Notes	-

Table 6.8: TC-3-2: OWASP Coraza WAF

TC-ID	TC-3-3
Lab	OWASP Coraza WAF
Description	Audit logs are enabled on caddy.
Precondition	Multi-service docker resource is deployed and linked to the challenge
Test steps	1. Start the coraza-waf multi-service docker resource 2. Run a vulnerable curl command on Service1 3. Create a dashboard in Kibana 4. Check if audit logs are visible in the Kibana dashboard
Expected result	Audit logs from caddy should be visible on newly created Kibana dashboard.
Environment	The test environment is the Hacking-Lab Challenge page.
Priority	high
Status	pass/fail/NA
Notes	-

Table 6.9: TC-3-3: OWASP Coraza WAF

TC-ID	TC-3-4
Lab	OWASP Coraza WAF
Description	Backend Application can be changed
Precondition	Multi-service docker resource is deployed and linked to the challenge
Test steps	1. Go to Hacking-Lab resource manager to the owasp-coraza resource 2. Configure another backend image for the application 3. Configure the Caddy BACKEND_APPLICATION and BACKEND_APPLICATION_PORT env var in the caddy service 4. Redeploy the challenge 5. Start the OWASP-coraza resource in the challenge
Expected result	When navigating to the Service1, the new backend application should be available.
Environment	The test environment is the Hacking-Lab Challenge page.
Priority	low
Status	pass/fail/NA
Notes	-

Table 6.10: TC-3-4: OWASP Coraza WAF

6.3.4 Test cases: OAuth2 vulnerabilities

TC-ID	TC-4-1
Lab	OAuth2 Vulnerabilities
Description	Multi-Docker is working and correctly linked to the application on the nginx services page.
Precondition	Multi-service docker resource is deployed and linked to the challenge
Test steps	1. Start the "lab-oauth2-vulnerabilities-multidocker" resource. 2. Wait until the service is started. 3. Navigate to Keycloak and Note App. 4. Check if they are running.
Expected result	Keycloak and Note App are linked correctly and running.
Environment	The test environment is the Hacking-Lab Challenge page.
Priority	high
Status	pass/fail/NA
Notes	-

Table 6.11: TC-4-1: OAuth2 Vulnerabilities

TC-ID	TC-4-2
Lab	OAuth2 Vulnerabilities
Description	Keycloak client import and login function.
Precondition	Multi-service docker resource is deployed and linked to the challenge
Test steps	1. Start the "lab-oauth2-vulnerabilities-multidocker" resource. 2. Wait until the service is started. 3. Import the keycloak client config as declared in the lab instructions. 4. Test the login with the admin user in the Notes App
Expected result	Logged in as the admin user in the Notes App
Environment	The test environment is the Hacking-Lab Challenge page.
Priority	high
Status	pass/fail/NA
Notes	-

Table 6.12: TC-4-2: OAuth2 Vulnerabilities

TC-ID	TC-4-3
Lab	OAuth2 Vulnerabilities
Description	Notes app functionality
Precondition	Multi-service docker resource is deployed and linked to the challenge
Test steps	1. Start the "lab-oauth2-vulnerabilities-multidocker" resource. 2. Wait until the service is started. 3. Import the keycloak client config as declared in the lab instructions. 4. Login as admin user 5. Take some notes
Expected result	Notes are listed and stored.
Environment	The test environment is the Hacking-Lab Challenge page.
Priority	medium
Status	pass/fail/NA
Notes	-

Table 6.13: TC-4-3: OAuth2 Vulnerabilities

TC-ID	TC-4-4
Lab	OAuth2 Vulnerabilities
Description	Impersonating user "john"
Precondition	Multi-service docker resource is deployed and linked to the challenge
Test steps	1. Start the "lab-oauth2-vulnerabilities-multidocker" resource. 2. Wait until the service is started. 3. Import the keycloak client config as declared in the lab instructions. 4. Login as admin user 5. Do a POST request using accessToken and username to the /login endpoint. 6. Navigate to the Notes App and refresh the site.
Expected results	Notes of the user "john" are listed, which include the Flag.
Environment	The test environment is the Hacking-Lab Challenge page.
Priority	high
Status	pass/fail/NA
Notes	-

Table 6.14: TC-4-4: OAuth2 Vulnerabilities

6.3.5 Test cases: Rate limiting

TC-ID	TC-5-1
Lab	Rate limiting
Description	Check if Theia is accessible and working.
Precondition	Multi-service docker resource is deployed and linked to the challenge
Test steps	1. Start the "lab-api-rate-limiting" resource. 2. Open the link to Theia from the README. 3. Change the name of one of the initial books. 4. Save and run <code>npm start</code> in the terminal. 6. Check the Library sample app web page.
Expected results	The applied changes should be visible on the Library web page.
Environment	The test environment is the Theia environment
Priority	high
Status	pass/fail/NA
Notes	-

Table 6.15: TC-5-1: Rate limiting

TC-ID	TC-5-2
Lab	Rate limiting
Description	Upload books to Library
Precondition	Resource is deployed and linked to the challenge
Test steps	1. Start the "lab-api-rate-limiting" resource. 2. Open the link to Theia from the README. 3. Start the terminal and run <code>npm start</code> 4. Upload a book through POST request and file.
Expected results	The books should be visible on the Library page.
Environment	The test environment is the Library sample app.
Priority	low
Status	pass/fail/NA
Notes	-

Table 6.16: TC-5-2: Rate limiting

TC-ID	TC-5-3
Lab	Rate limiting
Description	Test report page solution
Precondition	Resource is deployed and linked to the challenge
Test steps	1. Start the "lab-api-rate-limiting" resource. 2. Open the link to Theia from the README. 3. Copy and paste the lab solution from teacher grading. 4. Check the Test report page
Expected results	All tests should be passed.
Environment	Test report web page.
Priority	high
Status	pass/fail/NA
Notes	-

Table 6.17: TC-5-3: Rate limiting

TC-ID	TC-5-4
Lab	Rate limiting
Description	Students cannot access the test file.
Precondition	Resource is deployed and linked to the challenge
Test steps	1. Start the "lab-api-rate-limiting" resource. 2. Open the link to Theia from the README. 3. Try to open the test directory from the Theia workspace. 4. Try to open it from the terminal.
Expected results	It is not accessible from the GUI and terminal.
Environment	Test report web page.
Priority	medium
Status	pass/fail/NA
Notes	-

Table 6.18: TC-5-4: Rate limiting

6.3.6 Test cases: Input validation and sanitisation

TC-ID	TC-6-1
Lab	Input validation and sanitization
Description	Readme is accessible
Precondition	User is logged in to the HL
Test steps	1. Start the Implementing logging docker resource 2. Access the webpage of the resource which should show the labs readme page
Expected result	Readme page can be accessed
Environment	The test environment is the Hacking-Lab Challenge page.
Priority	high
Status	pass/fail/NA
Notes	-

Table 6.19: TC-6-1: Input validation and sanitization

TC-ID	TC-6-2
Lab	Input validation and sanitization
Description	Theia instance is accessible
Precondition	User is logged in to the HL
Test steps	1. Start the Implementing logging docker resource 2. Access the webpage of the resource, which should show the labs readme page 3. From the readme page, access the Theia instance
Expected result	Theia instance is accessible
Environment	The test environment is the Hacking-Lab Challenge page.
Priority	high
Status	pass/fail/NA
Notes	-

Table 6.20: TC-6-2: Input validation and sanitization

TC-ID	TC-6-3
Lab	Input validation and sanitization
Description	Theia instance works properly
Precondition	User is logged in to the HL
Test steps	1. Access the Theia instance 2. Open, edit and save files in the root directory 3. Start terminal and execute npm start
Expected result	Theia instance works properly
Environment	The test environment is the Hacking-Lab Challenge page.
Priority	high
Status	pass/fail/NA
Notes	-

Table 6.21: TC-6-3: Input validation and sanitization

TC-ID	TC-6-4
Lab	Input validation and sanitization
Description	Test report page solution
Precondition	Resource is deployed and linked to the challenge
Test steps	1. Start the "lab-api-rate-limiting" resource. 2. Open the link to Theia from the README. 3. Copy and paste the lab solution from teacher grading. 4. Check the Test report page
Expected results	All tests should be passed.
Environment	Test report web page.
Priority	high
Status	pass/fail/NA
Notes	-

Table 6.22: TC-6-4: Input validation and sanitization

TC-ID	TC-6-5
Lab	Input validation and sanitization
Description	Students cannot access the test file.
Precondition	Resource is deployed and linked to the challenge
Test steps	1. Start the "lab-api-rate-limiting" resource. 2. Open the link to Theia from the README. 3. Try to open the test directory from the Theia workspace. 4. Try to open it from the terminal.
Expected results	It is not accessible from the GUI and terminal.
Environment	Test report web page.
Priority	medium
Status	pass/fail/NA
Notes	-

Table 6.23: TC-6-5: Input validation and sanitization

6.4 Public testing

The public testing section explains the testing method, the tools used, and the testing phase results.

6.4.1 Tools

Microsoft Forms is used to collect feedback from the participants. It is easy to set up and collect feedback and integrates well with the Microsoft toolset.

The questions are kept general so they can be used throughout all labs. They are integrated into the instruction at the end of each lab with a link to the feedback form

to keep it simple and easy for all participants.

6.4.2 Questions

The following questions were asked:

1. How easy was it to set up the environment before the start of the labs? 1 difficult - 10 easy
2. How easily were you able to navigate through the lab instructions? 1 difficult - 10 easy
3. Did you encounter any difficulties while performing tasks in the lab environment? 1 a lot of problems - 10 no problems
4. How good were the API Security challenges designed? 1 not good - 10 very good
5. How supportive did you find the laboratory environment? 1 no support - 10 very good support
6. How adequate were the resources and tools provided for completing the tasks? 1 not good - 10 very good
7. How would you rate the realism of the scenarios presented in the lab? 1 not realistic - 10 very realistic
8. Were there any areas for improvement or suggestions you would like to provide for enhancing the lab experience?

6.4.3 Overall feedback

The following statistics show the overall feedback across all labs. Shown in Figure 6.1. The statistics come from 4 participants for each lab.

Question	Labs					
	Implement Logging	Enumeration And reconnaissance	OAuht2 Vulnerabilities	OWASP Coraza WAF	Rate limiting	Input validation And sanitization
1	9	9	9.5	7.75	8.5	8.75
2	7.75	8.75	9	7.75	8.75	8.5
3	7.5	8.5	8	6.75	9	8.25
4	7.25	9.5	8.5	6.33	9	8.75
5	6.75	9	8	6.5	8.75	8
6	7.75	9.75	8	7	8.75	8.75
7	9.25	7.5	8.5	8	8.75	8.5
Average	8	9	9	7	9	9

Figure 6.1: Overall feedback labs

The overall feedback was good for all the, except for some outliers, which were not extremely bad. The improvements made out of the feedback are discussed in the next section.

6.4.4 Feedback improvements

This section contains the improvement after the feedback result.

Enumeration and reconnaissance

The overall lab feedback was positive; therefore, no improvements were made.

Implement logging

The feedback is positive, but there is a recommendation to include an image guide to make it easier. Therefore, an image guide is now present in this lab.

Another feedback is to improve the grading system. Due to lack of time, another validation service isn't implemented.

OAuth2 vulnerabilities

The lab instructions were not improved because the overall rating regarding the instructions was positive.

The following improvements were mentioned in the free text field of question 9.

- "Hints are currently placed at the end of all steps. They would be more helpful to find them in the correct step."
- "noting against lab but better teachers grading would be good to have"

The hints were intentionally placed at the end of the instruction so that the students would not see them when they opened the task. So, there will be no changes to this feedback.

The teacher grading was improved for the second feedback with an example code of how the exploit post request works. Since there was no such example.

OWASP Coraza WAF

The lab instructions were improved based on feedback provided by the free back.

- "It would be nice if the service site could be more graphical. On Kibana setup the hint in number five would be better placed as an information in number 4. In Step three an example of a Log would be good to find it faster."

- "Very good lab to understand API / logs also one minor mistake in step 1 - Open Kibana by clicking Service 3 on nginx services page. you don't have anything named Service 3 now as you have renamed."

Regarding the first feedback, the static nginx services page was replaced with the Dynamic nginx multi-docker 5.3.1. Also the lab instruction were rearranged based on the feedback and additional pictures were added to help the students.

The second feedback was an adjustment that was forgotten after the Dynamic nginx change. It is fixed now.

Rate limiting

The following free text feedback was received for the lab:

- "A bit complex to understand due to the description."

To make the description more understandable, it was rewritten to be more precise.

Input validation and sanitisation

The lab was improved after the feedback that there is a partial problem with the execution of the lab.

- "‘npm start‘ gives permission issues on the ‘users.db‘ file. Please check."

The error is resolved, and the lab is not working.

Chapter 7

Results

As a result of this thesis, six labs were successfully created. Each lab covers a separate vulnerability of the OWASP Top 10 API Security Risks - 2023. The labs cover the following risks as you can see in the Table 7.1.

OWASP Top 10 API Security Risks – 2023		Lab
API1	Broken Object Level Authorization	
API2	Broken Authentication	OAuth2 Vulnerabilities
API3	Broken Object Property Level Authorization	Enumeration and reconnaissance
API4	Unrestricted Resource Consumption	Rate limiting
API5	Broken Function Level Authorization	
API6	Unrestricted Access to Sensitive Business Flows	
API7	Server-Side Request Forgery (SSRF)	OWASP Coraza WAF
API8	Security Misconfiguration	Implement logging, OWASP Coraza WAF
API9	Improper Inventory Management	Enumeration and reconnaissance
API10	Unsafe Consumption of APIs	Input validation and sanitization

Table 7.1: OWASP Risks covered with labs

The developed labs can be categorized into three types: tool-based, implementation-based, and labs focused on exploiting vulnerabilities from the attacker's perspective. Although the goal of developing six labs was met, three API Security Risks were not covered these are:

- API1:2023 - Broken Object Level Authorization
- API5:2023 - Broken Function Level Authorization
- API6:2023 - Unrestricted Access to Sensitive Business Flows

Chapter 8

Conclusion and outlook

This chapter covers the personal conclusion of this thesis and the outlook for future research based on this thesis.

8.1 Conclusion

This project aimed to develop and create labs on the Hacking-Lab platform regarding API Security. This was achieved using the OWASP Top 10 API Security Risks as a foundation for the lab development. The developed labs cover at least one Risk from the Top 10. The created labs allow students to apply theoretical concepts hands-on, simulating real-world scenarios and challenges.

The labs were created in a structured way so that the student was not overwhelmed by the challenge. Also, the appropriate hints were integrated to help the students without spoiling the solution.

Students gain a comprehensive understanding of the subject by combining tool-based and implementation-based exercises. Additionally, by covering seven of ten risks from the OWASP Top 10, the students will gain a broad knowledge of API Security by solving the labs.

8.2 Outlook

The future of this project could include the creation or improvement of labs to cover the remaining risks from the OWASP Top 10 API Security Risks 2023. Besides that, focusing on specific areas like cloud provider API would also be interesting as it becomes more and more relevant. Also, the OAuth2 authentication flow can be researched deeply as it becomes the de facto standard for authentication.

List of Figures

2.1	API Security fields [2]	14
2.2	Defensive mechanism APIs [2]	16
2.3	Token-based authentication	17
2.4	Authorization code flow	20
2.5	Implicit flow	20
2.6	Authorisation code flow with PKCE	21
3.1	Decisionmatrix	35
4.1	API login request	39
4.2	Payload positions	40
4.3	Payload settings	40
4.4	SQLInjection response	41
4.5	SQLInjection response token	41
4.6	OWASP Coraza WAF PoC architecture	47
5.1	Hacking-Lab: Labs overview	57
5.2	Multi-Docker before	59
5.3	Multi-Docker after	59
5.4	API enumeration and reconnaissance challenge	60
5.5	Architecture enumeration and reconnaissance [17] [25] [26]	61
5.6	Implement logging challenge	63
5.7	Architecture overview logging lab	64
5.8	Bubble app	65
5.9	OWASP Coraza WAF challenge	66
5.10	Architecture overview OWASP Coraza WAF lab	67
5.11	OAuth2 Vulnerabilities challenge	70
5.12	Architecture overview OAuth2 vulnerabilities lab	71
5.13	Notes App: Welcome page	72
5.14	Notes App: Notes	72
5.15	Notes App: Solution	73
5.16	Rate limiting challenge	74
5.17	Architecture overview Rate limiting lab	75
5.18	Library sample app	76

5.19	Library sample app	77
5.20	Library app: Test report	78
5.21	Library app: Test report solution	78
5.22	Input validation and sanitization challenge	79
5.23	Architecture overview input validation lab	80
5.24	User registration app	81
6.1	Overall feedback labs	95

List of Tables

1	OWASP Top 10 API Security Risks – 2023	4
3.1	Example lab framework	26
3.2	Lab idea: OAuth2 WebApp	27
3.3	Lab idea: JWT & RBAC	27
3.4	Lab idea: OWASP Juice Shop	27
3.5	Lab idea: Implementing logging	28
3.6	Lab idea: API rate limiting	28
3.7	Lab idea: OWASP Coraza WAF	29
3.8	Lab idea: API Security from an attacker’s perspective	29
3.9	Lab idea: Tooling lab	29
3.10	Lab idea: OAuth2 vulnerabilities	30
3.11	Lab idea: JWT vulnerabilities	30
3.12	Lab idea: Input validation and sanitisation	31
3.13	Lab idea: API enumeration and reconnaissance	31
3.14	Lab idea: API Logging & monitoring	31
5.1	OWASP Top 10 API Security Risks – 2023 [16]	58
6.1	Test Case template	83
6.2	TC-1-1: API enumeration and reconnaissance	84
6.3	TC-1-2: API enumeration and reconnaissance	84
6.4	TC-2-1: Implementing logging	85
6.5	TC-2-2: Implementing logging	85
6.6	TC-2-3: Implementing logging	86
6.7	TC-3-1: OWASP Coraza WAF	86
6.8	TC-3-2: OWASP Coraza WAF	87
6.9	TC-3-3: OWASP Coraza WAF	87
6.10	TC-3-4: OWASP Coraza WAF	88
6.11	TC-4-1: OAuth2 Vulnerabilities	88
6.12	TC-4-2: OAuth2 Vulnerabilities	89
6.13	TC-4-3: OAuth2 Vulnerabilities	89
6.14	TC-4-4: OAuth2 Vulnerabilities	90
6.15	TC-5-1: Rate limiting	90

6.16	TC-5-2: Rate limiting	91
6.17	TC-5-3: Rate limiting	91
6.18	TC-5-4: Rate limiting	92
6.19	TC-6-1: Input validation and sanitization	92
6.20	TC-6-2: Input validation and sanitization	93
6.21	TC-6-3: Input validation and sanitization	93
6.22	TC-6-4: Input validation and sanitization	94
6.23	TC-6-5: Input validation and sanitization	94
7.1	OWASP Risks covered with labs	98

Bibliography

- [1] NIST, “Nist glossary.” https://csrc.nist.gov/glossary/term/application_programming_interface, February 2024.
- [2] N. Madden, *API Security in Action*. Manning Publications, 2021.
- [3] “W3c.” <https://www.w3.org/>, February 2024.
- [4] K. Lane, “What is soap api.” <https://blog.postman.com/soap-api-definition/>, 2023. Accessed: 2024-06-13.
- [5] C. Miller, “From soap to rest: Tracing the history of apis.” <https://blog.treblle.com/from-soap-to-rest-tracing-the-history-of-apis/>, July 2023.
- [6] Dgraph, “The ultimate guide to graphql.” <https://dgraph.io/whitepaper/the-ultimate-guide-to-graphql>, n.d. Accessed: 2024-06-12.
- [7] MuleSoft, “What is an api?.” <https://www.mulesoft.com/resources/api/what-is-an-api>, n.d. Accessed: 2024-06-12.
- [8] M. Pollicove, “Ultimate guide to token-based authentication.” <https://www.pingidentity.com/en/resources/blog/post/ultimate-guide-token-based-authentication.html>, 2021.
- [9] A. by Okta, “What is oauth2.0?.” <https://auth0.com/intro-to-iam/what-is-oauth-2>, n.d.
- [10] IETF, “Rfc 6749.” <https://datatracker.ietf.org/doc/html/rfc6749>, October 2012.
- [11] OAuth, “OAuth 2.0.” <https://oauth.net/2/>, n.d. Accessed: 2024-06-12.
- [12] Auth0, “Application grant types.” <https://auth0.com/docs/get-started/applications/application-grant-types>, n.d. Accessed: 2024-06-12.
- [13] Auth0, “Openid connect protocol.” <https://auth0.com/docs/authenticate/protocols/openid-connect-protocol>, n.d. Accessed: 2024-06-12.
- [14] atlassian, “Microservices architecture.” <https://www.atlassian.com/microservices/microservices-architecture>, n.d. Accessed: 2024-06-13.

- [15] owasp, “Owasp api security top 10.” <https://owasp.org/API-Security/editions/2023/en/0x03-introduction/>, 2023. Accessed: 2024-06-13.
- [16] OWASP-API-Security-Project-team, “Owasp top 10 api security risks – 2023.” <https://owasp.org/API-Security/editions/2023/en/0x11-t10/>, June 2023.
- [17] B. Kimminich, “Juice shop logo.” https://raw.githubusercontent.com/bkimminich/juice-shop/master/frontend/src/assets/public/images/JuiceShop_Logo_400px.png.
- [18] T. P. Team, “What is an api endpoint?,” *What is an API endpoint?*, 2023.
- [19] A. S. Pedro Ruivo, “Keycloak.” <https://github.com/keycloak/keycloak>.
- [20] PortSwigger, “Oauth 2.0 authentication vulnerabilities.” <https://portswigger.net/web-security/oauth>.
- [21] I. Bütler, “Hacking-lab challenge generator.” <https://github.com/Hacking-Lab/generator-hl-challenge>.
- [22] I. Bütler, “Multiple docker resources.” <https://hacking-lab.atlassian.net/wiki/spaces/HLSD/pages/333971467/Hacking-Lab+Challenge+Development+Procedures>, November 2023.
- [23] J. Faltermeier, “theia-blueprint.” <https://github.com/eclipse-theia/theia-blueprint/tree/master>.
- [24] J. R. Laurent Bercot, “s6-overlay.” <https://github.com/just-containers/s6-overlay>.
- [25] Hacking-Lab, “Kookarai.” https://level1.idocker.hacking-lab.com/wp-content/uploads/2023/01/E1_beautiful_hacker_woman_bot_860f3a75-51b7-4a26-a8bd-78f746c9137f.png, n.d. Accessed: 2024-06-13.
- [26] PortSwigger, “Burp suite.” <https://portswigger.net/burp>, n.d. Accessed: 2024-06-13.
- [27] I. Bütler, “alpine-nginx-with-theia-web-ide-hl.” <https://github.com/Hacking-Lab/alpine-nginx-with-theia-web-ide-hl>.
- [28] J. Tosso, “Oss waf stack using coraza, caddy, and elastic.” Medium, 2023. Accessed: 2024-06-06.
- [29] Swagger, “Swagger.” <https://swagger.io/>.