AALBORG UNIVERSITY
STUDENT REPORT

Institut for Datalogi,

Aalborg University
Elma Lagerlöfs Vej 300,
9220 Aalborg Øst
http://www.aau.dk

Title:
Secure API Development Life Cycle

Theme:
Master Thesis

Project Period:
Winter Semester 2024

Guidance counselor
Marios Anagnostopoulos
mariosa@es.aau.dk

Student:
Jan Andersen
jande19@student.aau.dk

Copies: 1

Number of pages: 55

Date of Completion:
January 15, 2024

Abstract

Due to the increased use of APIs, and especially web APIs, and the importance of these to all parts of society, there is a need for a shift from security in digital solutions being something , which is incorporated after a product, so that it becomes an integral part of the entire software development life cycle.

However ,since the complexity of software increases significantly, including with the use of many 3rd party libraries, integrations, both between internal systems in companies, but also when doing business (B2B), it is necessary that more and better control comes with the software development process.

Some believe that the solution is to buy technological solutions that can scan for all kinds of vulnerabilities and, with a few clicks, get them fixed. Unfortunately, it is not that easy. This despite, vendors of the products, claim that this is the case.

Very few people are aware of which components are included in the solutions that they themselves help to develop. This is critical, because without knowledge of this, how will the responsible people be able to mitigate the risks that are in the products?

Furthermore, companies can no longer rely on the traditional perimeter security, that is, where there was no need to communicate with services on the Internet, and where everything was "secure" on the internal network.

With Web APIs, customers and other consumers have direct access to the company's data, and if these APIs are not sufficiently secured, this access can also be misused by hackers to gain access to potentially sensitive data.

To increase the security of the solutions that are developed and avoid that the time to be able to deliver is longer than necessary there is a need to work better and closer together, and to get away from a "silo approach", where each person does the work required, without considering other stakeholders.

The purpose of this thesis is to investigate what it will require to have Web API security being an integral part of the SDLC, including having threat model part of the pross, as well as the various kind of test, SAST and DAST, without slowing down, the time to deliver.

Acronym Key and Glossary Terms

ABAC	Attribute-based access control
ALTS	Application Layer Transport Security
API	Application Programming Interface
ASPM	Application Security Posture Management
AST	Application security testing
B2B	Business-to-Business
BOLA	Broken Object Level Authorization
CAPTCHA/ ReCAPTCHA	Completely Automated Public Turing-test to tell Computers and Humans Apart
CDN	content delivery network
crAPI	Completely ridiculous API
CSPM	Cloud Security Posture Management
DAST	Dynamic Application Security Testing
DevSecOps	Development, Security & Operations
DOS	Denial of Service
DREAD	Damage potential, Reproducibility, Exploitability, Affected users, Discoverability
False positive	An event is reported despite the everything works
IDE	Integrated Development Environment
HSM	Hardware Security Module
HSTS	HTTP Strict Transport Security
HTTPS	Hypertext Transfer Protocol Secure
JWT	Jason Web Token
Log4J	Java library
Log4Shell	Software vulnerability in Apache Log4j
MFA	Multifactor authentication (MFA)
OAuth 2.0	Open Authorization
OIDC	OpenID Connect
OWASP	Open Worldwide Application Security Project
OSS	Open-source software
OTP	One Time Password
PII	Personally Identifiable Information
RBAC	Role-Based Access Control
REST	Representational State Transfer
SAST	Static Application Security Testing
SBOM	Software Bill of Materials
SDLC	Software Development Life Cycle
SLR	Systematic Literature Review
SOAP	Simple Object Access Protocol

SSO	Single Sign On
SSRF	Server-Side Request Forgery
STRIDE	Spoofing Tampering Repudiation, Information disclosure, Denial of Service, Elevation of privileges
TLS	Transport Layer Security
WAAP	Web application and API protection
WAF	Web application firewall

INTRODUCTION

1.1 MOTIVATION

Digital solutions are emerging faster than ever, and the demand to ensure their availability and security is becoming even more critical due to their importance to all parts of society. In fact, it is no longer enough to focus on the products/solutions that are developed, but companies must be managed as if it was a software company to remain relevant [1] [2].
Because of the rapid digital transformation and increased connectivity to the Internet, individuals, enterprises of any size and authorities are significantly more exposed to cyber-attacks [3].
There can be many explanations for why companies are compromised, but according to a study conducted by Contract Security, it is not hackers or other malicious actors who pose the biggest threat, rather is it vulnerabilities in the software that is used, and thereby developed (in good faith) by software developers. In fact, according to the study, nearly 50% =of all compromises are due to vulnerabilities that were ultimately created by software developers [4] [5].
Another study, by Akami, who provides content delivery network (CDN), Security Solutions other cloud services showed that by October 2018, 83% of all web traffic were based on Application Programming Interface (API) calls, and this number is growing [6].
In essence, an API can be described as a mechanism that enables software components to communicate with each other over a network, using a set of definitions and protocols using a common language that they both understand, without any user interaction and manages interactions between applications, data and devices, and makes it possible to transfer data between systems [7].

They are an enabler for organizations to not only improve/mature existing business offerings, but also to seek new business opportunities faster than ever before, by enabling enterprises to expose services, provide access to company data in a "modern" way and even create integration to other companies. It also exposes business services or assets to developers, who are building applications.
They are vital for businesses in any industry, and the importance of APIs should not be seen only from a technical point of view, but as an organizational strategy of how to do business.
For developers, they can consume APIs and use them to develop new apps, without having to start from scratch [8].

Enterprises can use APIs to grow their company, like Netflix did when they made their API publicly available, and developers started to create third-party apps, which in the end, gave Netflix more customers [9].

However, as the usage of APIs are increasing, so are attacks against these, perhaps even to become the most frequent attack vector, securing APIs from the wide range of emerging and evolving threats is critical for business success. [10].

Due to this, API security is becoming even more important, not just for security professionals, but also for developers, architects, and business stakeholders, given the proliferation and application of APIs in modern application and integration architecture.

According to research by Google in 2022, 53% of organizations decided to delay the rollout of a new service or application, due to concerns about API security. 77% of organizations who had a security incident, due to API security, had to delay a rollout of new software [11].

If vulnerabilities to APIs (and other software components) were identified in the early phase of the Software Development Lifecycle (SDLC), it could help to avoid the delay of a rollout of a software release, but how can a mature SDLC prevent a slow development process, and still being able to identify vulnerabilities?

The research within this thesis aims to answer how adopting SDLC practices within the software development process, throughout the API lifecycle, can help developers and other stakeholders to develop secure APIs.

1.2 PROBLEM DEFINITION

As an API is becoming an even more critical component in the software application architecture, ensuring the security of APIs is becoming even more important [12].
However, despite significant investments in both technology and people to increase security and to protect against cyber-attacks, even companies who spend billions in are still breached by insecure APIs [13].

Traditional security controls are seemingly not applicable for API', so what must be done to improve the security of APIs throughout the entire SDLC? [14] [15]?

The main problem to be answered is:

> **RQ:** "*How can API security be improved by following a Secure Software Development Lifecycle (SDLC) approach?*

This question will be further extended with the following:

> **RQ.1:** "*From a process point of view – without slowing down the development process*"?
>
> **RQ.2:** "F*rom a technical point of view – including enabling technology to perform various security related activities – what to be aware of"*?

This thesis will not only investigate various technology stacks, which can help solve a specific purpose, but also focus on Threat Modeling to identify potential threats during the design phase.

The research approach will be the Multivocal Literature Review (MLR). This is a kind of a Systematic Literature Review (SLR) which includes material from several sources, including, but not limited to: videos, blog posts, non-research documents etc. The reason for choosing this method is that for the research topic of this report, what is available as research material for reference is rather limited [12].

1.3 SCOPE

The scope of this study is the research for improving security for APIs throughout the SDLC process, and with a primary focus on the architecture/design phase and application security testing.
This will include the creation of a Threat Model, and application security testing.

Beside the focus on when application security testing technology will not be enough to identify potential weaknesses, potential solutions for mitigations will be investigated.

Both the development and operational phases will be discussed, but the scope is not to develop an API, nor is it to establish an API gateway – even though, some of the findings potentially could lead to future recommendation. Should any such be identified, each of them will be discussed in theory.

Any technology, which are being used to generate results, must have coverage for OWASP API TOP 10 [16]. Throughout the report, a vulnerable API: Completely ridiculous API (CRAPI) will be used [17]. There will be no considerations about the quality of the technology, which is being used, even though there could be situations where Open-source software (OSS) or software which can be used without any financial cost, have limitations compared to commercial software.

1.4 DATA COLLECTION

The process of gathering and analyzing relevant data for this study has mainly been focused on published academic research papers and published books with the focus of Secure Software development, API design, API-management, API-Security, Threat Modeling, Application Security Testing, and papers about "Secure by design, "Defense in Dept" and "Zero Trust".

The sources from which the literature has been collected are primarily from Aalborg University Library and searches via Google scholar, Books, articles, and other non-research literature has mainly been collected through publishers, "trusted" sources such as, but not limited to: OWASP (Open Web Application Security Project) and NIST (National Institute of Standards and Technology). In addition to this, searching the Internet, using browsers such as Microsoft Bing and Google Chrome for relevant content has also been done.

1.5 DATA ANALYSIS

Experimental studies have been done using crAPI (completely ridiculous API) which is vulnerable by design, GitHub, a developer platform, used for storing source code, executing build and security testing using products from Semgrep and Snyk for static application security testing [18] [19] and for Dynamic application security testing, Burp Suite community edition and Apisec Free API Security assessment tool were used [20] [21].

1.6 API SECURITY LIFECYCLE

The objective of this thesis is to investigate how to design and test an API to be secure throughout the entire SDLC. Originally the scope was about security testing of APIs, as several vendors claim to have coverage for OWASP API top 10 risks, yet many breaches are caused by vulnerabilities in APIs.

However, after having done some research, applying both static and dynamic application security testing on a few vulnerable APIs, it became clear that,

none of the tools being used, had full coverage, especially when testing for authentication and authorization vulnerabilities, rather generating several false positive.

Furthermore, without a ferrule understanding of the scope of the solution, the huge amount of data returned from a security scan can be difficult to prioritize as other mitigating controls may be implemented making the finding less critical.

Based on this, the scope was changed to investigate how a set of guidelines/recommendations using modern technology, could be used throughout the SDLC, to help creating secure WEB APIs.

FUNDAMENTALS

1.7 API FUNDAMENTALS

Several types of APIs exist, all of which provide the same core functionality, which is to serve as an interface for communication between systems. It can be thought of as a set of defined rules which enables software to communicate, either locally on a device, or over a network using a common language that they both understand [22]. Communication between an API provider and consumer are defined in an API contract and requires non- human interaction [7] [23]. APIs can work in multiple networks, however, the most widespread type of API to be used within a network is a Web API, which can be utilized using the HTTP/HTTPS protocol [24].

1.8 TYPE OF APIS

Web APIs plays a significant role in this even more connected society, and thus the importance of proper design should not be underestimated. Regardless of whether an API is to be consumed internally in an organization, or in the public Internet, solid API design is a foundation for success or the opposite [25].
However, as several different types of APIs exist, each having a specific purpose, it is important to understand them, the use cases to which they are designed for, to make correct choices for any given solution and the risks associated with them [26].

1.8.1 OPEN API

An open API, also known as a Public API, is open and available for anyone to use. Organizations can assign developers, both internal to an organization and externally, to have access to application data while prohibiting access to the actual source code. The more accessible and well-documented an API is, chances of its success increase significantly, both in relation to a business perspective developer attention. However, there are security risks associated with exposing a company's back-end systems directly on the Internet allowing access to everyone through a firewall. Furthermore, it must be expected that the number of API calls will increase significantly. One mitigation would be to limit the number of requests users who aren't paying for consuming the API, using rate limiting, and require authentication – e.g., using JSON Web Token (JWT), OAuth or an API key [7] [26] [27].

1.8.2 PARTNER API

A Partner API can be described as an interface which are used by organizations to make data available for other companies/partners, as well as to gain access to other companies' data and service offerings, allowing for creating unique features, using a partner's resources business-to-business (B2B). A Partner API should be restricted to specific users/companies, while limits for requests may vary depending on the consumer. For any request, proper authorization, and authorization, must be applied [7] [26] [27].

1.8.3 INTERNAL API

Internal APIs are only available for usage within an organization, and its functionality are designed for specific use-case e.g., process automation, data transfer between systems or providing developer employees access to data; Business-to-Employee (B2E). Depending on the data exposed, this may not require any authentication [26] [28].

1.8.4 COMPOSITE API

A composite API combines/consolidates several APIs into a single interface, which a unified view of from different data sources. This integration simplifies data access and offers developers efficient coding practices, as they do not have to write separate code for every individual API.
Access control to the various APIs included in a request depends on how it is handled in the various APIs.
Use cases for a composite API can be B2E, Application-to-Application, B2B and B2C [26] [28] [29].

1.9 API PROTOCOLS AND ARCHITECTURES

In essence, a protocol is a set of rules for formatting, processing, and exchanging data between systems.
Several different API protocols exists, e.g., GraphQL, gRPC ("google Remote Procedure Call") SOAP (simple object access protocol) or REST (Representational State Transfer), each of which includes standards and processes that an API uses for communication and exchange of data. All with strengths and weaknesses. Despite that an API can utilize multiple API protocols, however, the impact of selecting an appropriate API architecture, can impact the success and/or adoption of an API, and therefore, even though this thesis will not focus on specific API protocols, but rather aim for providing guidelines for securing any type of API despite their differences, a short description of the once already mentioned, will be done [30] [31] [32] [33].

1.9.1 REST (REPRESENTATIONAL STATE TRANSFER)

Rather than being an actual protocol, REST is a set of architectural constraints / design style, used to create HTTP APIs, and which can be implemented in a variety of ways. REST applications use HTTP methods like GET, POST, DELETE, and PUT. Presented in 2000 in his dissertation, Thomas Roy Fielding specified that, any API, which follows the six guiding principles: Uniform Interface, Client-Server, Stateless, Cacheable, Layered System and Code on Demand or constraints of the RESTful architecture is a REST API [34] [35].

1.9.2 GraphQL (GRAPH QUERY LANGUAGE)

GraphQL is a query language for APIs, developed by Facebook and made open-sourced in 2015. Following the six constraints of REST APIs, GraphQL is to be considered RESTful. furthermore, GraphQL also have the advantage of being query-centric, as it is designed to work in an equivalent way similarly to a query language such as, Structured Query Language (SQL). Using GraphQL, data from multiple sources can be requested and returned in a single request [36] [37].

1.9.3 GRPC (GOOGLE REMOTE PROCEDURE CALL)

Developed by Google in 2015, gRPC is based on remote procedure calls (RPC), allowing a client application to call and execute a method on a remote server as if it were a local object. This is especially beneficial when creating distributed systems or working with a microservice architecture. gRPC uses the HTTP/2 protocol for communication, which is a major revision of the original HTTP protocol [38] [39].

1.9.4 SOAP (SIMPLE OBJECT ACCESS PROTOCOL)

First introduced in 1998, SOAP was designed as a message specification for exchanging information between systems and applications. Besides HTTP, SOAP also supports the following transfer protocols: TCP, SMTP, FTP. Despite new types of APIs – e.g. GraphQL, and the fact that SOAP is losing popularity, SOAP is still being used in many industries, and will continue to be relevant going forward [40] [41].

SOFTWARE DEVELOPMENT LIFE CYCLE (SDLC)

Ensuring quality and security within a software product has for several years been a challenge for companies delivering the digital products as well as for the developers writing the software. It continues to be. The same is true for APIs. Traditionally, security has been an "after thought", decoupled from the development phase. Dedicated security teams would test a product for vulnerabilities after the software has been developed, perhaps even without understanding the architecture in detail [42].
This became true for the U.S government in the 1970's, when they realized that basic penetration testing wouldn't be sufficient to identify both quality defects and/or security problems in the solutions. To this, several limitations were discovered - including the (limited) knowledge that the teams who performed the test had, but also, the limited time to perform various tests.
However, as one source of compromise is due to vulnerability within the software being used, it is essential to integrate security throughout the product development process. Trying to overcome the challenges, the U.S. government concluded that to achieve secure and reliable solutions, the development process should be managed in a more rigorous and systematic way [43].
Concepts such as, Defense in depth, Secure by Default etc. to mention a few, gained traction for an increasing number of companies within the private sector, such as, but not limited to Microsoft and RedHat, with the purpose of building security into the design of their solutions [44] [45]. However, as many standards and guidelines related to secure software development are created with high-level and declarative content, it is a challenge for the developers to implement security in the products. Furthermore, missing, or limited information-sharing across development and operations teams prevents organizations getting the most value out of the resources in which they have invested. This goes for both human resources, but also technology purchased for securing the products [43] [46].

1.10 DEVSECOPS

Integrating security throughout the entire API lifecycle will require collaboration between the stakeholders involved in the process.
DevSecOps (Development, Security & Operations), is a practice of having security integrated into the entire software development lifecycle, with the aim of being able to detect security and/or design issues in applications, as early as possible [47].
It can be seen as an enhancement of DevOps which addresses the need for continuously to integrate security across the SDLC, enabling teams to deliver secure and robust solutions - without impacting the time to deliver [48] [49].

In the context for API security, following a DevSecOps approach, the intent is to amalgamate all phases, starting from the plan/design → development → test → deployment → operations → retirement into one unified approach. This is to ensure that APIs are both designed and developed with security in mind, based on the potential risks identified, e.g., data privacy concerns, but also to have it deployed efficiently, and managed effectively throughout the entire lifecycle.

1.11 OWASP TOP 10 API SECURITY RISKS 2023

Created by OWASP (Open Worldwide Application Security Project), a nonprofit foundation, the Top 10 API Security Risks 2023, has been developed with a focus on strategies and solutions regarding API Security, and to help raise awareness and solutions for a better understanding of how to mitigate the unique vulnerabilities and security risks associated with APIs [16].

Establishing and enforcing coverage for the API Top 10 API Security Risks 2023 within the SDLC, indicates that security is not just a high-level guideline, rather it shows commitment to follow and implement industry best practices for secure development.

Despite numerous of documentation exists. about OWASP API top 10, each of them, will be explained in high-level, as tests will be done for investigating if/how tradiational application security testing (ASP) technology, will be able to identify vulnerabilities within APIs.

1.11.1 API1:2023: BROKEN OBJECT LEVEL AUTHORIZATION

Description:
BOLA (Broken Object Level Authorization), is an authorization vulnerability, allowing access to resources that should otherwise be restricted to unauthorized individuals.
This can be due to insecure coding practices, including failing to properly validate input or verifying permissions before granting access to an object.

Impact:
A successful BOLA attack could lead to some of the following:

Attack:	**Consequence:**
Account takeover	An individual gains full control of a legitimate account and uses it for malicious purposes
Data disclosure to unauthorized individuals.	Companies could get a fine for breaching of the responsibility to properly ensure the security of personal data
Unauthorized access	Account takeover, data manipulation, deleting of data

Scenarios:
Without sufficient access controls, an API endpoint will not be able to validate that users are only able to access resources belonging to them.
An example to this, is the following, which is a direct reference to a record in a database -e.g.:

https://janswebite.org/users/1234
If it is possible for an individual to modify the number to something as:
https://janswebite.org/users/1235,
and access data belonging to someone else, the solution is vulnerable to BOLA, as no validation was done prior to providing access.

Prevention:
Enforce proper authorization at every request. HTTP is a stateless protocol, meaning that one request does not have any relation with other succeeding requests.
Implement authorization mechanism to check if the logged-in identity has access to perform the requested action on a record in every function that uses an input from the client to access a record in the database.
Use random and unpredictable values as globally unique identifier (GUID) for record IDs [16] [50] [51].

1.11.2 API2:2023: BROKEN AUTHENTICATION

Description:
The process of authenticating identities must be considered broken if, for example, an API endpoint fails to use a modern and strong authentication method, or if the control used is poorly designed and/or implemented incorrectly. Regardless of the reason, compromising the ability to identify a client will compromise the overall API security.

Impact:
The impact of broken authentication can be devastating if malicious users are able to compromise a high-privilege account – e.g., a "domain admin" account. In addition to this, another consequence could be to have personal data leaked or perform actions on behalf of the compromised user.

Attack:	**Consequence:**
Weak passwords – e.g., "abc123":	Several attack techniques for breaking the password such as a "rainbow table" could be used to get access to a system.
Weak cryptography- e.g., SHA1 or MD5 hashing algorithms	Confidentiality is broken

Scenarios:
Passwords or secrets, being an API key, are stored improperly – e.g., outside a Hardware Security Module (HSM). Rather it is stored in a version control system and is by accident being committed to a public repository or sent to members in the team by email.

Prevention:
Don't use default passwords.
Ensure proper key management (both for storing and rotating keys)
Implement weak password checks.
Never trust, always verify (Zero trust principle)
Use Principle of Least Privilege
Whenever possible, multi-factor authentication must be implemented.
[16] [50] [51] [52] [53]

1.11.3 API3:2023: BROKEN OBJECT PROPERTY LEVEL AUTHORIZATION

Description:
Exposing more data than required, or allowing unauthorized permissions to objects, when allowing users access to an APIs, it is important to not only verify if the authentication request is valid – verifying permissions for each object is crucial. Compared to GraphQL, REST API often returns more data than required.

Impact:
Unauthorized access to sensitive data, could result in both having data disclosed, tampering of data / loss of integrity. Furthermore, it could result in privileges escalation.

Attack:	**Consequence:**
Mass Assignment:	Allowing a user to escalate privileges or edit object properties.

Scenarios:
Allowing a user to update the title of a e-book, which is blocked for reading:

```
PUT /api/books/Book-About-Api-Security


{
  "description": "Great book to read"
}
```

Could be changed by a user, using the following:

```
PUT /api/books/Book-About-Api-Security


{
  "description": "Great book to read"
  "blocked": false
}
```

Without proper validation for verifying whether, the user should have access to the object, hence it is possible to allow/deny access for their own.

Prevention:
Validate permissions at the object level.
Define and return only the properties of an object required in a request.
Implement technology, which are capable of to detect "suspicious behaviour"/patterns. [54] [55]

1.11.4 API4:2023: UNRESTRICTED RESOURCE CONSUMPTION

Description:
Without limitations for consuming resources, an API provider is in risk of becoming either a victim for a Denial of Service (DoS) attack or to be in a situation with an additional expenditure.

Impact:
The impact of Unrestricted Resources Consumption can be that end-users will get a timeout, when consuming a solution, with the result, that another provider will be selected. Another consequence could be the additional processes being executed, which could lead to an increased operational cost. Either in terms of CPU usage, or for storage, if no limitation is configured for the size of files to be uploaded. Being unavailable prevents an API from serving legitimate requests.

Attack:	**Consequence:**
Botnet	Business is unavailable. Increased billing.

Scenarios:
Multiple concurrent requests from both legitimate and/or infected devices can lead to performance degradation or even unavailability.

Prevention:
Rate limiting should be fine-tuned based on the business needs. E.g., only allow a public facing-API to be invoked a certain number of times, within a period.

Throttling could be implemented, defining the frequency of how often a single API client/user can execute a given operation.

Enforce a maximum value of data on incoming parameters.

[16] [51] [56]

1.11.5 API5:2023: BROKEN FUNCTION LEVEL AUTHORIZATION

Description:
Broken Function Level Authorization allows an authorized user access to an endpoint of function. This occurs when an API endpoint isn't configured with the appropriate permissions. Complexity within the authentication scheme could be one reason for this.

Impact:
Being able to exploit this, an attacker could get access to resources (including administrative functions), belonging to someone else.

Attack:	**Consequence:**
Account takeover	An individual gains full control of a legitimate account and uses it for malicious purposes
Data disclosure to unauthorized individuals.	Companies could get a fine for breaching of the responsibility to properly ensure the security of personal data
Unauthorized access	Account takeover, data manipulation, deleting of data

Scenarios:
Utilizing a non-administrative account to successfully conduct an action which normally would requires administrative privileges, is evidence of a broken function-level authorization.

Prevention:
Follow the principle of least privilege.
Have a clear separation between admin and non-administrative functions.

[16] [51] [57]

1.11.6 API6:2023: UNRESTRICTED ACCESS TO SENSITIVE BUSINESS FLOWS

Description:
APIs vulnerable to this risk expose a business flow—such as buying a ticket to a concert or sporting event—without considering how that functionality could do harm if automated. This could be by design, and not necessarily a technical issue.

Impact:
Buying tickets for a concert, either a person, or even an automated process using a headless browser, can prevent others from buying tickets.

Attack:	**Consequence:**
DOS	Business unavailable due to overloading the API with requests.
Excessive resources are required to keep up with the requests	Additional cost/expenses to the business

Scenarios:

A car rental app has a referral program where its users can invite friends and acquaintances to join, and for each person who has joined the app, they get a bonus. This bonus can later be used as cash to rent cars. A person with bad intentions can exploit this flow by automating the registration process, where each new user adds credit to the attacker, who then afterwards can then rent cars without having to pay for it.

Prevention:
Implement a human detection solution, e.g., CAPTCHA or ReCAPTCHA (Completely Automated Public Turing test to tell Computers and Humans Apart).
Device fingerprinting: deny service such as headless browsers)
Identify and protect business flows which could harm the business, if being misused.
[16] [58] [59] [60]

1.11.7 API7:2023: SERVER-SIDE REQUEST FORGERY

Description:
In a Server-Side Request Forgery (SSRF) attack, it is possible for an attacker to abuse functionality on the server to get access to internal resources. Despite the resources are not exposed through a firewall, but only available internally, however a vulnerable webserver may have access to resources inside a firewall, which allows an attacker to abuse the web application to read the internal resources.

Impact:
SSRF can be used to bypass security controls and thereby gain unauthorized access to internal resources within a company, including sensitive data.

Attack:	**Consequence:**
Exploiting a vulnerable webserver	By exploiting a vulnerable web server, it is possible for an attacker to make the server establish a connection to an internal service, which could result in a company leaking sensitive data

Scenarios:
the attacker might cause the server to make a connection to internal-only services within the organization's infrastructure. In other cases, they may be able to force the server to connect to arbitrary external systems, which potentially could lead to sensitive data being exfiltrated.

Prevention:
Input validation: One of the most important steps in mitigating SSRF vulnerabilities is to validate and sanitize user input.
[16] [56] [61]

1.11.8 API8:2023: SECURITY MISCONFIGURATION

Description:
Security flaws Configuration can be the result of several things, including insecure configurations, missing or misconfigured HTTP headers, unpatched systems, all of which can be exploited by an attacker.

Impact:
The impact of a misconfiguration can be many, but an example could be that an attacker can cover for actions committed by modifying and/or deleting log files stored in a directory without access control.

Attack:	**Consequence:**
Use default credentials	An attacker could easily get access to a solution
Unpatched systems	Can result in unauthorized access to sensitive data
Ransomware	Exploit vulnerabilities to encrypt data

Scenarios:
All API requests, including tokens providing admin access, are being logged and stored in a logfile, which is publicly available on the Internet. Anyone, who can find the share, is able to read the information, and get access to the keys.

Prevention:
Using a baseline, implement and enforce a repeatable hardening process for all systems.
Ensure to have an up to date, asset inventory – including every API endpoint.
Retire systems, which are no longer in use.
Always use encrypted communication – also for internal systems, or solutions, which are sending non-sensitive data.
Never use default settings.

1.11.9 API9:2023: IMPROPER INVENTORY MANAGEMENT

Description:
Improper asset management is due to missing or no API documentation or properly managing them. This can be developers, who do not document the APIs they develop.

Impact:
Incorrect asset management is when there are multiple versions of the same API, API1 and API2, but the original API1 is not shut down, despite API2 being used. These older versions - often without maintenance and updates, tend to use weaker security requirements and can be simple to exploit.

Attack:	**Consequence:**
Brute-force	A flow within one of Facebook's API, allowed third-party apps to access the private data of millions of users without consent. Having visibility and control of the API, this potentially could have been avoided.

Scenarios:
According to OWASP, one scenario is about a social media network who did manage to implement rate-limiting to block brute force attacks, however, it was implemented as a separate component rather than being integrated in the code. In the case of a beta API host would host the same API, rate limiting would be bypassed, hence a brute-force attack would be possible.

Prevention:
Implement an inventory of all APIs, endpoints, API hosts and versions.
Retire old/obsolete APIs.
All API versions must be under version control.
[62] [63]

1.11.10 API10:2023: UNSAFE CONSUMPTION OF APIS

Description:
Insecure consumption of APIs is when back-end implementations, which have external integration, accepts user-controlled inputs from external party carried over APIs without applying any proper validations.

Impact:
The impact can vary; however, successful exploitation could lead to sensitive data being exposed to unauthorized users.

Attack:	**Consequence:**
Integration	Caused by Unsafe consumption of APIs in Log4j, the Log4Shell vulnerability allowed users to execute arbitrary code on numerous web services, using the Apache Log4j logging library, potentially leading to allowing an attacker with full control of the system.

Scenarios:
An API blindly trust input from other external APIs, which could lead to vulnerabilities in the consuming application.

Prevention:
Only allow API communication over a secure channel (HTTPS).
Never trust, but always validate and sanitize input from integrated APIs. Do avoid, to blindly follow redirects.
[16] [64]

1.12 API SECURITY BASELINE

The cost of solving a security vulnerability is far more expensive once a product has been released, compared to, having security as an integral part [65]. Depending on the use case, being an API, which should be used for an integration either internally in an organization or with an external company, or made public available for external developers to consume, rather than having to define requirement for each API, based on a risk assessment, a baseline should be established, making it clear, what security controls are required.

When defining the security baseline, it is important to distinguish between risks associated with being an API provider and an API consumer.

Guidelines for securing APIs, throughout the entire life cycle of an API, presented and published in the article: *Security Guidelines for Providing and Consuming APIs,* aim to provide guidance for both scenarios: Ingress API exposure (provider), and Egress access (consumer) [66].

1.13 EXPOSURE OF API (INGRESS)

An Inbound API connection can be accessible, both to internal and external consumers, e.g., a developer, access to read or modify data stored in the internal network.

1.13.1 DESIGN PHASE

The d

ID	Control	Description
1	Threat Modeling and Countermeasures	Designing for security, having a threat modeling process integrated into the SDLC, when developing an API, is crucial for being able to assess the possible threats and vulnerabilities as well as the likelihood that weaknesses that are identified will be exploited. A threat model can also help prioritize which weaknesses need to be mitigated first. A threat model must not be considered static, rather be included in every sprint, or whenever significant changes are made to the API.
2	Idendity management	Never trust, always verify. All requests, on all API endpoints, should require the client to authenticate and authorize independent of the API endpoint or object being accessed. Multifactor authentication (MFA) should be enforced, when possible. Preferably, the authorization service, is decoupled from the API itself, but managed independently. Use of least privileged access should be enforced. When possible, the use of secure protocols such as API keys, JWT, OpenID Connect (OIDC) and OAuth2 should be used. Legacy authentication such as Basic authentication, should be prevented, and only used as an exception, for legacy applications.
3	Secure communication	Communication is secured by enforcing HTTPS between an HTTP client and a Web API. self-

		signed certificates must not be used in production environments.
4	Input and output validation	Data received, (including request parameters), and sent by an API should be validated and sanitized. Avoid operational dependencies between systems by having input validation decoupled from the application, but specified in a format that can be reviewed, e.g., an OpenAPI Specification. The API contract defines the expectations for how the API should work, including but not limited to input and output formats.

1.13.2 DEVELOPMENT PHASE

5	Implement rate limeting.	Implement rate limiting and throttling policies to prevent abuse due to excessive requests against an API.
6	Secure consuming of 3rd party components.	Preferably combined with a process for onboarding and utilizing open-source components, libraries should be checked for vulnerabilities, licenses terms and operational risks. The latest stable version of components should always be used if possible. A Software Bill of Materials (SBOM), including software components used to build an API, should be available.
7	Storage of Application Secrets	Encryption keys and application secrets must be stored in a secure location such as a vault or Hardware Security Module (HSM). The development phase should include a process for Secret detection and remediation, to prevent sensitive data such as secrets, passwords, and keys from entering the code repository. Access keys and other secrets should be rotated periodically. Key and secret management, should always be done taken the most care due to the criticality in case of a compromise.
8	Token Strength	Strong algorithms for securing API keys or JWT tokens are essential due to data privacy and security, hence a "state-of-the-art" encryption algorithm should always be used.

9	Input Validation/ Output Encoding	Appropriate input validation and output encoding should be done. For instance, only accept input which strictly conform to an API specification.
10	Error Handling	API implementations should not expose issues occurring in the system, including: - service failure - permission issues Rather a generic error and default error code could be returned: 100-199: Informational 200-299: Success 300-399: Redirection 400-499: Client error 500-599: Server error [67]
11	Protection of Testing/Staging Environments	Using a baseline configuration, the same security requirements enforced for production environments should be required for other environments which processes data that could pose a risk in the event of a data breach.

1.13.3 TESTING PHASE

12	Penetration Testing and Continuous Vulnerability Scanning	To identify vulnerabilities within an API, a Pentest, performed by an independent individual, should be conducted, prior to releasing for production. Different types of tests exist – including: **White-box testing**: Internal information, e.g., source-code and documentation is fully available access to the tester. **Gray-box testing**: Partial access to relevant internal information for being able to do a test. **Black-box testing**: No knowledge about the target is known to the tester.
13	Code Review	Prior to release, code review must be performed, either manually, or automated by applying Application Security Testing (AST) technology.

14	Application and Secret scanning	Prior to code deployment and based on security testing processes and requirements, using appropriate automated application security testing technology and secret detection should be performed.

1.13.4 IMPLEMENTATION PHASE

15	Cryptography	To safeguard data processed by an API, consider: **Data in transit**: TLS/HTTPS helps to prevent network communication from being intercepted, However, only strong cipher suites should be used - never consider deprecated and insecure cipher suites. Confidentiality is about keeping data secret from anyone other than those who have a legitimate purpose for accessing it. However, **Data at Rest** and **Message integrity** should be considered as well.
	Detection / Monitoring	A process for detecting suspicious API activity and responding to this, should be implemented. Designed, not only to protect Web applications and APIs, is a Web Application and API protection (WAAP).
16	Exposed Network Interfaces	Deny by default. No network ports or services should be exposed otherwise required. Care should be taken for APIs with admin privileges.
17	Session Termination	Depending on the sensitivity of the data, a timeout value should be configured to the shortest time possible, as active sessions are a target to an attacker [68].
18	System Updates	For all components used in a solution, including but not limited to infrastructure components, open-source libraries etc. must be updated on a regular basis. Teams responsible for individual components should sign up to receive notifications from the vendor, or similar, if/when a vulnerability is identified, hence actions is required.

1.13.5 LOGGING AND MONITORING PHASE

19	Application Security Posture Management	Due to complexity, it can be difficult to prioritize security risks associated with APIs and the associated infrastructure. Security Posture Management (ASPM) and Cloud Security Posture Management (CSPM), are 2 different solutions, which aim to help secure applications and cloud infrastructure [69].
20	Audit and logging	Any system must be capable of logging events due to several reasons, including compliance, legal, security and root-cause analysis. HTTP access logs should be saved in a separate location, and only as an exception should sensitive data, e.g., API keys be stored. Should this be required, the stoarge for these logfiles, should be further protected, and only accessible from a compliant device, and with the use of either SSO or MFA.

1.14 CONSUMER OF AN API (EGRESS)

1.14.1 DESIGN PHASE

1	Threat Modeling and Countermeasures	While most security controls should be implemented, by the API provider, having a threat model created for outgoing traffic is just as important, for being able to assess the possible threats and vulnerabilities as well as the likelihood that weaknesses that are identified will be exploited. For instance, a compromised account could be used to exfiltrate data, and having this threat documented, also could indicate which mitigations to take.

1.14.2 DEVELOPMENT PHASE

2	Storing Secrets (Digital Safe)	Encryption keys and secrets must be stored in a secure location such as a vault or Hardware Security Module (HSM). Access keys and other secrets should be rotated periodically.

1.14.3 TESTING PHASE

3	Application Security Scanning and Secrets Scanning	Prior to building the product, the source code must be tested for vulnerabilities. Several integrated development environments (IDE), have this integrated, meaning fast feedback to the developers. The scanning solution, must be finetuned, to avoid to many false positive.

1.14.4 IMPLEMENTATION PHASE

4	TLS Valid Certificate	Independent on the data being transferred, encrypted communication must be enforced, using HTTPS, with a strong cipher suite.
5	Session Termination	The value for terminating inactive sessions must be within the shortest time possible.
6	Destination IP and Port Limitation	External connection made by service accounts should be restricted to specific URLs and whitelisted in a firewall.

1.14.5 LOGGING AND MONITORING PHASE

7	Continuous Monitoring	To be able to react to alerts, it is important that security risks and vulnerabilities are continuously evaluated.
8	Detection and Response	People, processes, and technology should be available for ongoing detection and have the capability to act accordingly.
9	Documentation	Ensure documentation exist and is updated throughout the life cycle of an API.

THREAT MODELING

Threat modeling is the process of analyzing a system to identify and evaluate security issues and weaknesses in a system, so appropriate actions can be taken to mitigate those, as early as possible in the SDLC.

As a process for designing for security, solutions which are developed with security in mind from the beginning, are to be more secure, as the identified attacks are considered, and the necessary security controls are included to prevent them.

In the context of a Web API, a Threat model could help communicate what product is being developed, the sensitivity of the data being processed, potential integrations, who will consume the API, and how? Are users able to provide input, which should be sanitized, before entering the system? Have the required security headers been enabled? Has HTTPS been enforced on all API endpoints using a strong cipher suite? What about rate limiting? Perhaps paid customers should have more "bandwidth" than non-paying customers. Have granular permissions been considered – e.g., Attribute-based access control (ABAC) or Role-Based Access Control (RBAC)? [70] [71].

1.15 SCOPE FOR A THREAT MODEL

crAPI, a solution, which is vulnerable-by-design, is used for the purpose of creating a threat model.
The architecture and description, which are used for the threat model, are all from the GitHub page of crAPI [72].
The focus will be on ingress communication and identities.

1.16 THREAT MODELING METHODOLOGY

1.16.1 STRIDE

STRIDE, short for: (Spoofing, Tampering, Repudiation, Information Disclosure, Denial of Service, and Elevation of Privilege) is a methodology for categories representing potential attack vectors, which can be exploited by threat actors.

Spoofing: Refers to an attack, where someone or something is impersonating a legitimate user or process to gain access to resources, to which they elsehow would be unauthorized.

Tampering: Is about unauthorized alterations of data or systems, such as modifying data in transit, due to insecure communication protocol.

Repudiation: Denying claims/unproven actions that has taken place, e.g., removal of data.

Information Disclosure: Involves exposing potential sensitive information to unauthorized parties, due to inadequate permissions.
Denial of Service: Service unavailability, preventing valid users to use the services provided by a system.

Elevation of Privilege: Is when a person or process gains extended permissions, allowing for actions to be done on resources, which elsehow would be denied.

The following table maps each threat to the corresponding security property [73] [74].

Threat	**Security property**
Spoofing	Authentication
Tampering	Integrity
Repudiation	Non-Repudiation
Information Disclosure	Confidentiality
Denial Of Service	Availability
Elevation Of Privileges	Authorization

1.17 THREAT MODELING AN API USING STRIDE

Following the Four Question Framework by Adam Shostack [75]:

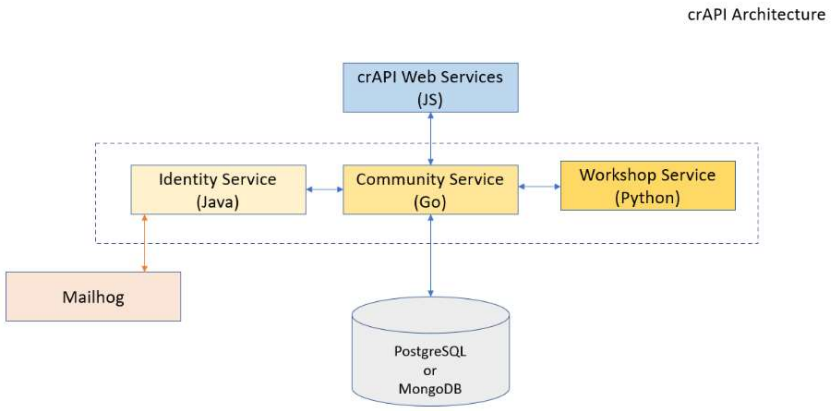
- What are we working on?
- What can go wrong?
- What are we going to do about it?
- Did we do a good job?

Each question will be used for creating a threat model for a Web API.

1.17.1 WHAT ARE WE WORKING ON?

As a starting point, creating a diagram of the solution being built is a great way of presenting it to the relevant stakeholders.

The below diagram shows, at a high-level, the components included in the solution, and part of the business logic. *See Figure 1.A: Architecture of crAPI* for further information.



At this stage, potential things which can go wrong, such as: "How are the compoennts protected from each other"? "Are they communicating in a secure way"? Based on the system design, a threat model ,or models depending on the details required, can be created.

Using a Data Flow Diagram, adding trust boundaries to the components in use, visialize how the data flows through the solution, permant stoarge as well as internal and external boundaries. Each are they represented with one of the following icons:

Figure 2.A: Icons used to create the threat model.

The threat model created for this can be found in the appendix.
See figure 3.A: crAPI threat model.

1.17.2 WHAT CAN GO WRONG?

Spoofing

Threat / "Victim"	**Mitigation**
User	Authenticity
Website	Authenticity

Target for crAPI	**Description:**
1. Authorized user 2. Employee 3. Web site	Valid credentials are leaked, e.g., via. phising None or improper certificate implementation.

Countermeasures for Spoofing:

- Authentication enforcement on every API endpoint and request.
- Use secure and unique authentication tokens, e.g., JWT or OAuth 2.0.
- Input validation
- MFA
- Rate Limiting and Throttling to avoid brute force
- Enforce least privilege principles
- HTTPS, with a strong cipher suite.
- HSTS security header to only allows HTTPS traffic.
- Logging and monitoring, e.g., network behavior anomaly detection.

Tampering

Threat / "Victim"	**Mitigation**
Community	Integrity
Mailhog	Integrity
MongoDB	Integrity
Website	Integrity

Target for crAPI	**Description:**
1. Authorized user of Community, Mailhog and MongoDB 2. Data stored on Website.	Broken user authentication, allowing access to else restricted data. E.g., One Time Password (OTP) for reset of password.

Countermeasures for Tampering:
- Encrypt sensitive information.
- Applied Principle of Least Privilege.
- Input validation
- Authentication enforcement.
- Authorization enforcement.
- Logging and monitoring.

Repudiation

Threat / "Victim"	**Mitigation**
Users	Non-repudiation

Target for crAPI	**Description:**
1. Users within community	A user reveals the location of a car belonging to someone claims not to have done so.

Countermeasures for Repudiation:
- Authentication enforcement.
- Logging and monitoring.

Information Disclosure

Threat / "Victim"	**Mitigation**
Community Identity Mailhog MongoDB PostgreSQL Users Workshop	Confidentiality

Target for crAPI	**Description:**
1. Any valid user	Any user within the system is at risk of having data exposed. Furthermore, internal data are also at risk of being leaked, hence the company owning the solution could get a fine for improperly protecting personal identifiable information (PII).

Countermeasures for Information Disclosure:
- Authentication enforcement on every API endpoint and request.
- Use secure and unique authentication tokens, e.g., JWT or OAuth 2.0.
- Secure coding
- Application security testing
- Encrypt sensitive information.
- Input validation
- MFA
- Enforce least privilege principles
- HTTPS, with a strong cipher suite.
- HSTS security header to only allows HTTPS traffic.
- Logging and monitoring.

Denial Of Service (DOS)

Threat / "Victim"	**Mitigation**
Community Mailhog Website	Availability

Target for crAPI	**Description:**
1. Any valid user – including the owner(s) of the solution.	This type of attack aims to overwhelm a system, e.g., a Web server or application by sending a large volume of requests, to consume server resources or drain the server's capacity, making it unavailable. NB: This does not have to be malicious, but could be due to misconfiguration, or limited resources.

Countermeasures for Denial of Service (DOS):
- Logging and monitoring.
- Web Application Firewall (WAF), capable of detecting malicious traffic and blocking it.

Elevation of Privileges

Threat / "Victim"	Mitigation
Identity MongoDB PostgreSQL Website Workshop	Authorisation

Target for crAPI	Description:
1. Any valid user – with extended permissions – e.g., system administrators or. Domain admins.	Elevation of Privilege is when a user or application gains permissions that should not be available to them. It could be due to the credentials of an administrator has been leaked to the public.

Countermeasures for Elevation of Privileges

- Authorization enforcement.
- Enforce least privilege principles
- Logging and monitoring.
- MFA
- Proper key and secret management.

1.17.3 WHAT ARE WE GOING TO DO ABOUT IT?

Based on the findings, it should be decided how to address and prioritize between them. Several options are available, including mitigate each of them, having the functionality removed, so it no longer poses a threat, make it the responsibility of the developers to remove the bug in the code, or simply just accept it [76].

As this is about securing Web APIs, and having it being part of the SDLC, the focus will be on identity management and application security testing with OWASP Top 10 API security risks.

SAST:
Using SAST, none of the tools used (used with default settings), were able to identify all the top 10 Risks.

However, improper TLS and missing TLS were found, and resolved, so all communication are encrypted.
See figure 4.A: SNYK SAST overall findings.

Furthermore, hardcoded credentials were only found in one of the solutions.
See figure 5.A: SNYK hardcoded password finding.

The number of false positive, is difficult to answer, due to limited knowledge about the solution, and already implemented mitigations, however, a comparison has been made, with no rule configured, were Snyk identified 55 vulnerabilities and Semgrep had 104 findings.
See figure 6.A: Compare Snyk and Semgrep for number of security findings.

DAST:
For automated API security testing, the free version of Apisec.ai was used. The only thing which was to be done was uploading an OpenAPI specification for the project, and tests for all Top 10 risks were conducted. *See figure 7.A: APISEC OWASP API Top 10 coverage.*
For the *Analysis Results, See figure 8.A: APIsec Spec Analysis Results.*

1.17.4 DID WE DO A GOOD JOB?

A threat model has been created, so identifying risks in the solution, and thereby being able to improve the security within it, and the APIs, should be seen as a good start. One thing which would improve on the model, would be to have conducted a risk analysis using the DREAD, (Damage potential, Reproducibility, Exploitability, Affected users, Discoverability) methodology, and then prioritize the findings based on the risk score.

API SECURITY TESTING TOOLS

Due to the uniqueness of a Web API compared to traditional Web solutions, and the wide spread of Web APIs in particular, the importance of security testing is not only important, but also further complicated. Therefore, various solutions have been investigated, including static code analysis, as well as dynamic code analysis.

1.17.5 STATIC APPLICATION SECURITY TESTING (SAST)

While static code analysis has full coverage of the source code, the tests are executed, without running the solution, hence it did provide several false positive. This indicates that, despite the value it did provide, a significant amount of time is required to have it properly fine-tuned.
One advantage is the integration into the IDE, which provides fast feedback to developers. The tests made in this thesis were conducted using the free version of Snyk.IO and Semgrep.dev. While both claim to have coverage for OWASP API top 10 risks, none of them were able to provide this. Nevertheless, both technologies still have an important role in the SDLC, combined with other technologies.

1.17.6 DYNAMIC APPLICATION SECURITY TESTING (DAST)

Unlike static code analysis, then Dynamic application security testing, are executing test cases, on a running solution. Thereby there are very few false positive, however, as there is no access to source code, it cannot be assured that, all possible ways data travers the solution is being tested. The tests made in this thesis were conducted using the free version Apisec.ai and Burp Suite community edition. However, due to the manual work and limited functionality included in this version, Burp Suite was rather quickly considered out of scope.

DISCUSSION AND CONCLUSION

1.18 DISCUSSION

People, process and technology, in that order.
Answering the research question, "*How can API security be improved by following a Secure Software Development Lifecycle (SDLC) approach*", the answer would be exactly that. No technology can be used out-of-the-box and resolve all the security issues in an API.

Proper and useful documentation and guidance for the teams who design, develop, implement and operate APIs, should be available, for them to consume. High-level documents, being guidelines or standards, stating that APIs should be developed or operated securely, do not provide value. It might if the solutions were less complex.

The technology available for API security, in general, is in the early stages, while maturing. However, as security issues often are created during the development phase, it is critical to be capable of identifying them, prior to release. Including business logic flaws.

Without an architectural overview of the APIs in scope, how data exposed through an API travers through systems, integrations between systems and their dependencies, chances are that not every component is included when performing a threat model and/or risk assessment.

Following the "Shift Left" approach [77], security testing is conducted early and throughout the development phase. Having security issues detected, and other design flaws identified earlier, the cost of having them resolved is significantly cheaper. Additional benefits with this approach could be improved and optimized processes in the software development process.

1.19 CONCLUSION

Close collaboration between stakeholders is required if the API security is to be improved significantly. Rather than considering an API as a software solution, it should be seen as a product which requires to be designed with security in mind, but also operated with care. APIs are critical for business success and should be managed accordingly.

Part of the reason why an API is vulnerable could be due to the many different types which exist, each with strengths and weaknesses. When to use what?
Going forward, research of different threat modeling methodologies of Web APIs could be interesting.

Works Cited

[1] J. P. Carvalho, "every-business-software-business," quidgest.com, [Online]. Available: https://quidgest.com/en/articles/every-business-software-business/.

[2] C. U. Orji, "CLOUD API SECURITY AUDIT - An Extensive Approach to API Assessment -," Aalborg University Electronics and IT, 2021.

[3] C. Teglers, "cybersikkerhed-saadan-ser-trusselsbilledet-ud," kromannreumert.com, 28 juni 2023. [Online]. Available: https://kromannreumert.com/viden/artikler/cybersikkerhed-saadan-ser-trusselsbilledet-ud.

[4] H. G. S. C. Michael Howard, Designing and Developing Secure Azure Solutions, Microsoft Press, 2022.

[5] P. Spencer, "assessing-appsec-implications," contrastsecurity.com, 20 May 2020. [Online]. Available: https://www.contrastsecurity.com/security-influencers/assessing-appsec-implications.

[6] A. Press, "state-of-the-internet-security-retail-attacks-and-api-traffic," akamai.com, 2 2019. [Online]. Available: https://www.akamai.com/newsroom/press-release/state-of-the-internet-security-retail-attacks-and-api-traffic.

[7] B. De, "API Management," in *An Architect's Guide to Developing and Managing APIs for Your Organization*, Apress Media, 2017.

[8] G. team, "maps," Google, 2023. [Online]. Available: https://developers.google.com/maps.

[9] R. Lawler, "netflix-api-shutdown," techcrunch.com, 13 6 2014. [Online]. Available: https://techcrunch.com/2014/06/13/netflix-api-shutdown/.

[10] M. Campbell, "api-security-predictions-2022-the-good-the-bad-and-the-scary," nonamesecurity.com, 30 12 2021. [Online]. Available: https://nonamesecurity.com/blog/api-security-predictions-2022-the-good-the-bad-and-the-scary/.

[11] G. Cloud, "API Security: Latest Insights & Key Trends," Google Cloud, 2022.

[12] D. Barahona, "business-logic-vulnerabilities," apisec, 10 04 2022. [Online]. Available: https://www.apisec.ai/blog/business-logic-vulnerabilities.

[13] K. Wagner, "facebook says it has spent 13 billion on safety security," bloomberg.com, 21 9 2021. [Online]. Available:

https://www.bloomberg.com/news/articles/2021-09-21/facebook-says-it-has-spent-13-billion-on-safety-security#xj4y7vzkg.

[14] Alexandra, "what-is-sdlc," stackify.com, 10 3 2023. [Online]. Available: https://stackify.com/what-is-sdlc/.

[15] reblaze, "api-security-vs-traditional-web-security," reblaze.com, [Online]. Available: https://www.reblaze.com/wiki/api-security/api-security-vs-traditional-web-security/.

[16] owasp.org, "0x10-api-security-risks/," owasp.org, 2023. [Online]. Available: https://owasp.org/API-Security/editions/2023/en/0x11-t10/.

[17] m. jose, "crAPI," github.com, 2023. [Online]. Available: https://github.com/OWASP/crAPI.

[18] S. team, "semgrep-code," semgrep, [Online]. Available: https://semgrep.dev/products/semgrep-code/.

[19] S. team, "snyk-code," snyk.io, [Online]. Available: https://snyk.io/product/snyk-code/.

[20] p. team, "communitydownload," portswigger.net, [Online]. Available: https://portswigger.net/burp/communitydownload.

[21] A. team, "product#scan," apisec.ai, [Online]. Available: https://www.apisec.ai/product#scan.

[22] G. B. D. W. Daniel Jacobson, APIs: A Strategy Guide, O'Reilly Media, Inc., 2011.

[23] c. team, "what-is-an-api-contract," criteria.sh, 23 5 2023. [Online]. Available: https://criteria.sh/blog/what-is-an-api-contract.

[24] t. team, "what-is-web-api," tutorialsteacher.com, [Online]. Available: https://www.tutorialsteacher.com/webapi/what-is-web-api.

[25] A. Lauret, The Design of Web APIs, New York: Manning Publications, 2019.

[26] k. team, "different-api-types-and-use-cases," konghq.com, [Online]. Available: https://konghq.com/learning-center/api-management/different-api-types-and-use-cases.

[27] M. F. R. H. O. M. M. R. E. S. Marius Aharonovich, "Security Guidelines for Providing and Consuming APIs," Israeli chapter of the Cloud Security Alliance (CSA), 2021.

[28] J. Juviler, "types-of-apis," blog.hubspot.com/, 16 1 2023. [Online]. Available: https://blog.hubspot.com/website/types-of-apis.

[29] J. Simpson, "nordicapis.com," nordicapis, 15 3 2022. [Online]. Available: https://nordicapis.com/6-types-of-apis-open-public-partner-private-composite-unified/.

[30] G. team, "https://www.howtographql.com/advanced/1-server/," howtographql, [Online]. Available: https://www.howtographql.com/advanced/1-server/.

[31] T. P. Team, "rest-api-examples," postman, 18 6 2023. [Online]. Available: https://blog.postman.com/rest-api-examples/.

[32] s. Team, "SOAP vs REST 101: Understand The Differences," soapui, [Online]. Available: https://www.soapui.org/learn/api/soap-vs-rest-api/.

[33] g. team, "about," gRPC, [Online]. Available: https://grpc.io/about/.

[34] R. T. Fielding, "Architectural Styles and the Design of Network-based Software Architectures," p. 180, 2000.

[35] r. team, "restfulapi.net," restfulapi.net, [Online]. Available: https://restfulapi.net/.

[36] g. team, "rest-api-architectural-constraints," geeksforgeeks.org, 1 6 2023. [Online]. Available: https://www.geeksforgeeks.org/rest-api-architectural-constraints/.

[37] C. Ball, Hacking APIs : breaking web application programming interfaces, San Francisco: no Starch Press, 2022.

[38] I. Grigorik, "performance-http2," web.dev, [Online]. Available: https://web.dev/articles/performance-http2.

[39] L. Ryan, "principles," grpc, 8 9 2015. [Online]. Available: https://grpc.io/blog/principles/.

[40] B. Woodring, "soap-api-for-saas," prismatic.io, 15 3 2023. [Online]. Available: https://prismatic.io/blog/soap-api-for-saas/.

[41] H. Bell, "what-is-a-soap-api," nonamesecurity.com, 8 6 2023. [Online]. Available: https://nonamesecurity.com/learn/what-is-a-soap-api/.

[42] A. team, "what-is/sdlc/," amazon.com, [Online]. Available: https://aws.amazon.com/what-is/sdlc/.

[43] S. Kang and S. Kim, "CIA-level driven secure SDLC framework for integrating security," 2021.

[44] M. team, "sdl," Microsoft, [Online]. Available: https://www.microsoft.com/en-us/securityengineering/sdl.

[45] J. Kelly and D. Sastre, "security-design-security-principles-and-threat-modeling," redhat, 23 2 2023. [Online]. Available: https://www.redhat.com/en/blog/security-design-security-principles-and-threat-modeling.

[46] M. MAZYAR, "devops-the-ultimate-way-to-break-down-silos," devops.com/, 18 12 2018. [Online]. Available: https://devops.com/devops-the-ultimate-way-to-break-down-silos/.

[47] O. team, "www-project-devsecops-guideline/latest," owasp, [Online]. Available: https://owasp.org/www-project-devsecops-guideline/latest/.

[48] S. Dunn, "From-devoops-to-devsecops," SANS, 23 2 2023. [Online]. Available: https://www.sans.org/blog/from-devoops-to-devsecops/.

[49] S. team, "what-is-devsecops," synopsys, [Online]. Available: https://www.synopsys.com/glossary/what-is-devsecops.html.

[50] C. Ball, "owasp-api-security-top-10-and-beyond/categories/2152491879/posts/2166897970," university.apisec.ai, [Online]. Available: https://university.apisec.ai/products/owasp-api-security-top-10-and-beyond/categories/2152491879/posts/2166897970.

[51] L. Tal, "owasp-top-10-vulnerabilities/api-security-top-10/," snyk.io, [Online]. Available: https://snyk.io/learn/owasp-top-10-vulnerabilities/api-security-top-10/.

[52] b. team, "rainbow-table-attack," beyondidentity, [Online]. Available: https://www.beyondidentity.com/glossary/rainbow-table-attack.

[53] D. simp, I. S. Jackson, i. and X. , "understand-default-user-accounts," microsoft, 19 5 2023. [Online]. Available: https://learn.microsoft.com/en-us/windows-server/identity/ad-ds/manage/understand-default-user-accounts.

[54] S. Uniyal, "api-security-series-part-3-hands-on-guide-to-reducing-excessive-data-exposure-253826363204," medium.com, 19 2 2021. [Online]. Available: https://shailanchal.medium.com/api-security-series-part-3-hands-on-guide-to-reducing-excessive-data-exposure-253826363204.

[55] O. team, "Mass_Assignment_Cheat_Sheet.html," owasp, [Online]. Available: https://cheatsheetseries.owasp.org/cheatsheets/Mass_Assignment_Cheat_Sheet.html.

[56] "owasp-api-security-top-10-and-beyond," .apisecuniversity.com, [Online]. Available: https://www.apisecuniversity.com/courses/owasp-api-security-top-10-and-beyond.

[57] D. K. Paxton-Fear, "owasp-api/broken-function-level-authorization," traceable.ai, [Online]. Available: https://www.traceable.ai/owasp-api/broken-function-level-authorization.

[58] A. Kiskyte, "what-is-headless-browser," oxylabs.io, 21 11 2023. [Online]. Available: https://oxylabs.io/blog/what-is-headless-browser.

[59] i. team, "what-is-a-device-fingerprint-and-what-is-it-used-for," incognia, [Online]. Available: https://www.incognia.com/the-authentication-reference/what-is-a-device-fingerprint-and-what-is-it-used-for.

[60] d. team, "aptcha-vs-recaptcha-whats-the-difference," datadome, [Online]. Available: https://datadome.co/bot-management-protection/captcha-vs-recaptcha-whats-the-difference/.

[61] p. team, "ssrf," portswigger.net, [Online]. Available: https://portswigger.net/web-security/ssrf.

[62] P. Dughi, "/owasp-top-10-api-improper-inventory-managemen," barracuda.com, 8 8 2023. [Online]. Available: https://blog.barracuda.com/2023/08/08/owasp-top-10-api-improper-inventory-management.

[63] O. team, "0xa9-improper-inventory-management," OWASP, [Online]. Available: https://owasp.org/API-Security/editions/2023/en/0xa9-improper-inventory-management/.

[64] F. Wortley, F. Allison and C. Thompson, "log4j-zero-day," lunasec.io, 9 12 2021. [Online]. Available: https://www.lunasec.io/docs/blog/log4j-zero-day/.

[65] K. J. Higgins, "the-cost-of-fixing-an-application-vulnerability," darkreading.com, 11 5 2009. [Online]. Available: https://www.darkreading.com/cyber-risk/the-cost-of-fixing-an-application-vulnerability.

[66] O. Avenstein and S. G. Maor, "Security Guidelines for Providing and Consuming API," Cloud Security Alliance, 2021.

[67] J. Albano, "rest-api-error-handling-best-practices," baeldung.com, 8 1 2024. [Online]. Available: https://www.baeldung.com/rest-api-error-handling-best-practices.

[68] T. OWASP, "Session_Timeout," owasp, [Online]. Available: https://owasp.org/www-community/Session_Timeout.

[69] J. Peterson, "aspm-vs-cspm-key-differences," cycode.com, [Online]. Available: https://cycode.com/blog/aspm-vs-cspm-key-differences/.

[70] O. team, "what-is-role-based-access-control-rbac/," okta.com, 15 9 2023. [Online]. Available: https://www.okta.com/identity-101/what-is-role-based-access-control-rbac/.

[71] K. Casey, "attribute-based-access-control-abac," okta.com, 29 9 2020. [Online]. Available: https://www.okta.com/blog/2020/09/attribute-based-access-control-abac/.

[72] O. C. team, "crAPI_architecture.md#architecture-of-crapi-1," OWASP, [Online]. Available: https://github.com/OWASP/crAPI/blob/main/docs/crAPI_architecture.md#architecture-of-crapi-1.

[73] "uncover-security-design-flaws-using-the-stride-approach," microsoft.com, 10 7 2019. [Online]. Available:

https://learn.microsoft.com/en-us/archive/msdn-magazine/2006/november/uncover-security-design-flaws-using-the-stride-approach.

[74] o. T. (TM), "Threat_Modeling_Process," owasp, [Online]. Available: https://owasp.org/www-community/Threat_Modeling_Process.

[75] A. Shostack, "threat-modeling," shostack.org, [Online]. Available: https://shostack.org/resources/threat-modeling.

[76] A. shostack, "/threat-modeling#4steps," shostack.org, [Online]. Available: https://shostack.org/resources/threat-modeling#4steps.

[77] f. team, "shift-left-security," fortinet.com, [Online]. Available: https://www.fortinet.com/fr/resources/cyberglossary/shift-left-security.

[78] S. team, "introducing-threatcanvas-an-ai-powered-tool-to-automate-threat-modeling/," secureflag, 6 11 2023. [Online]. Available: https://blog.secureflag.com/2023/11/06.

[79] s. team, "secureflag team," secureflag, [Online]. Available: https://www.secureflag.com.

[80] G. LLC, "2022 Research Report," 2022. [Online]. Available: https://services.google.com/fh/files/misc/google_cloud_api_security_research_report.pdf.

[81] N. Kirtley, "threat-modeling.com," 24 7 2022. [Online]. Available: https://threat-modeling.com/pasta-threat-modeling/.

[82] "https://www.sciencedirect.com/science/article/abs/pii/S0950584918301939," [Online].

[83] H. Bell, "api-security-best-practices," nonamesecurity, 18 6 2023. [Online]. Available: https://nonamesecurity.com/learn/api-security-best-practices.

[84] B. Bhattacharya, "api-management-101-rate-limiting," Tyk.io, 3 1 2024. [Online]. Available: https://tyk.io/blog/api-management-101-rate-limiting/.

APPENDIX A

APPENDIX

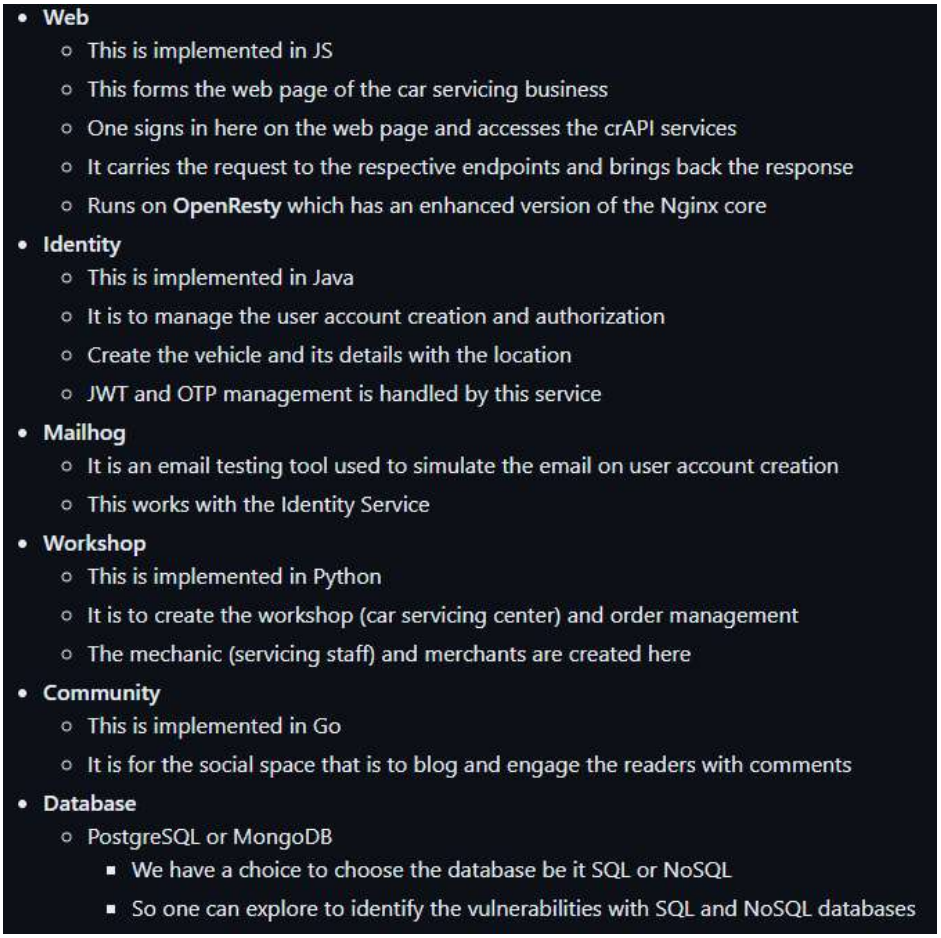


Figure 1.A: Architecture of crAPI.

Figure 2.A: Icons used to create the threat model.

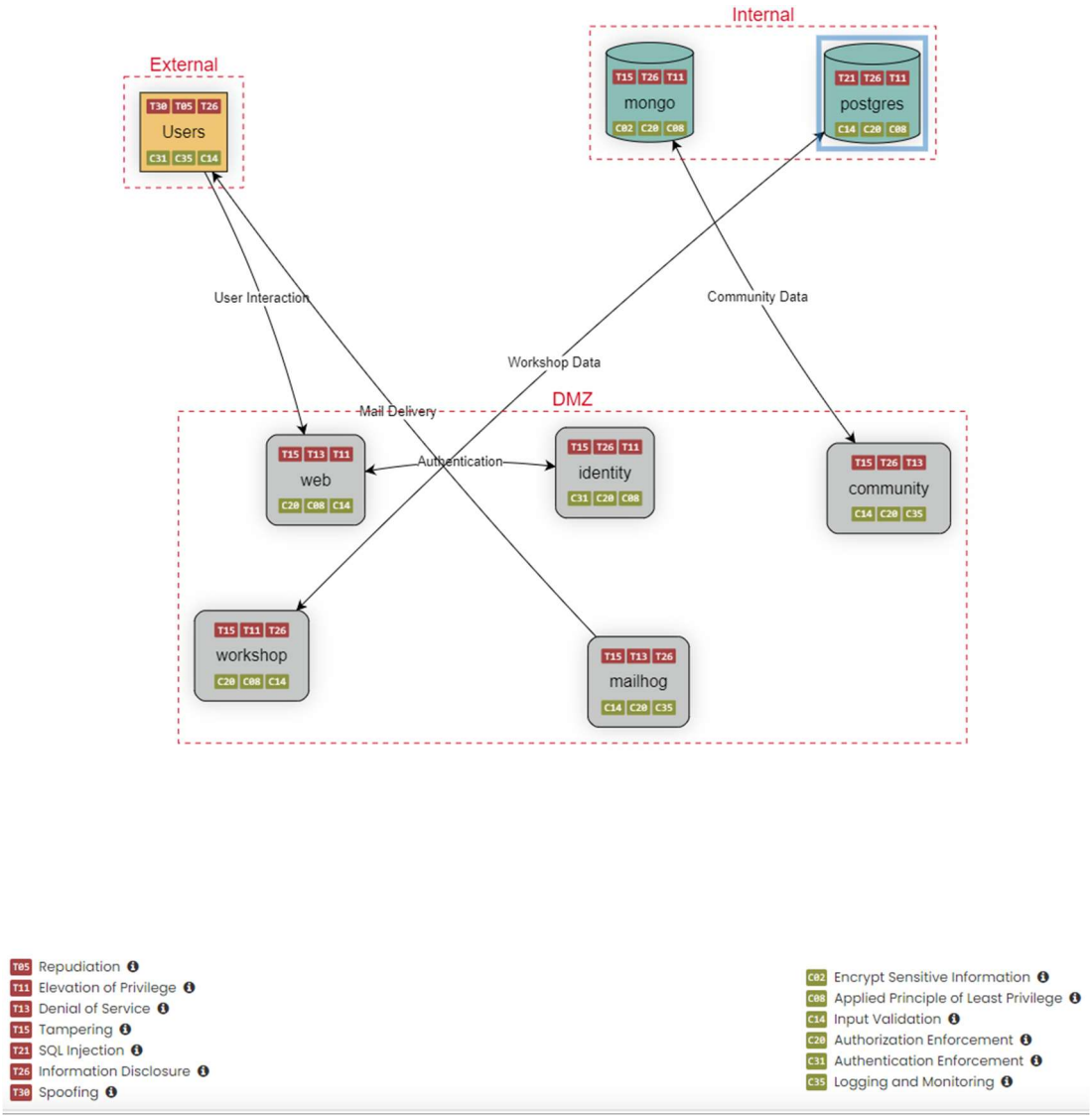


Figure 3.A: crAPI threat model. *The threat model has been created, based on the crAPI architecture drawing, by Jan Andersen, author of the thesis, using SecureFlag* [78] [79].

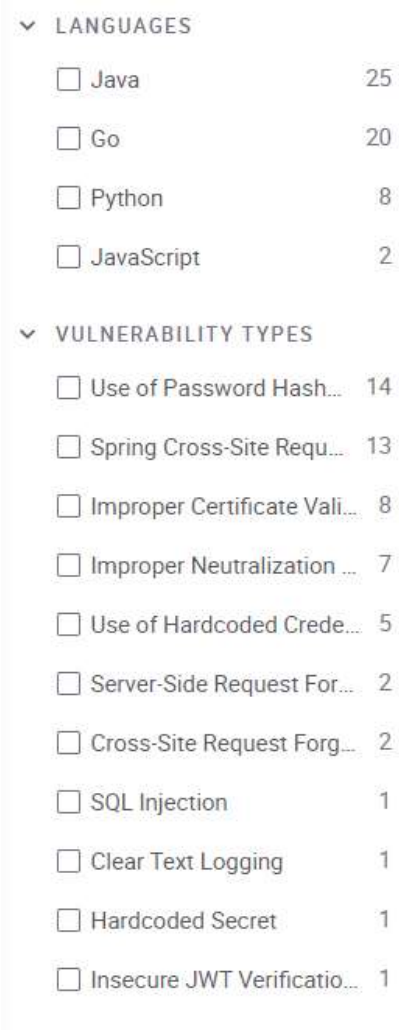


Figure 4.A: SNYK SAST overall findings. Source: [snyk.io]

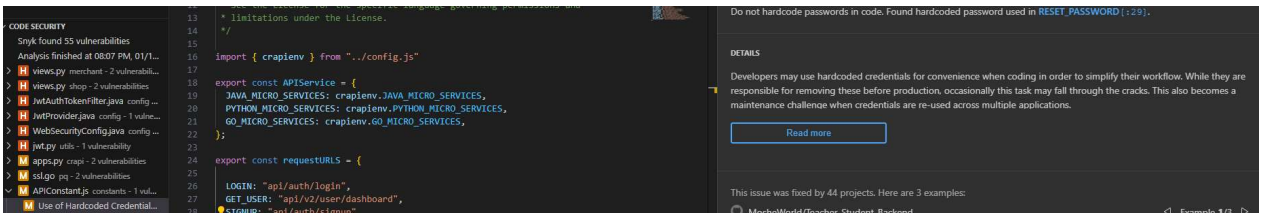


Figure 5.A: SNYK hardcoded password finding. Source: [snyk.io]

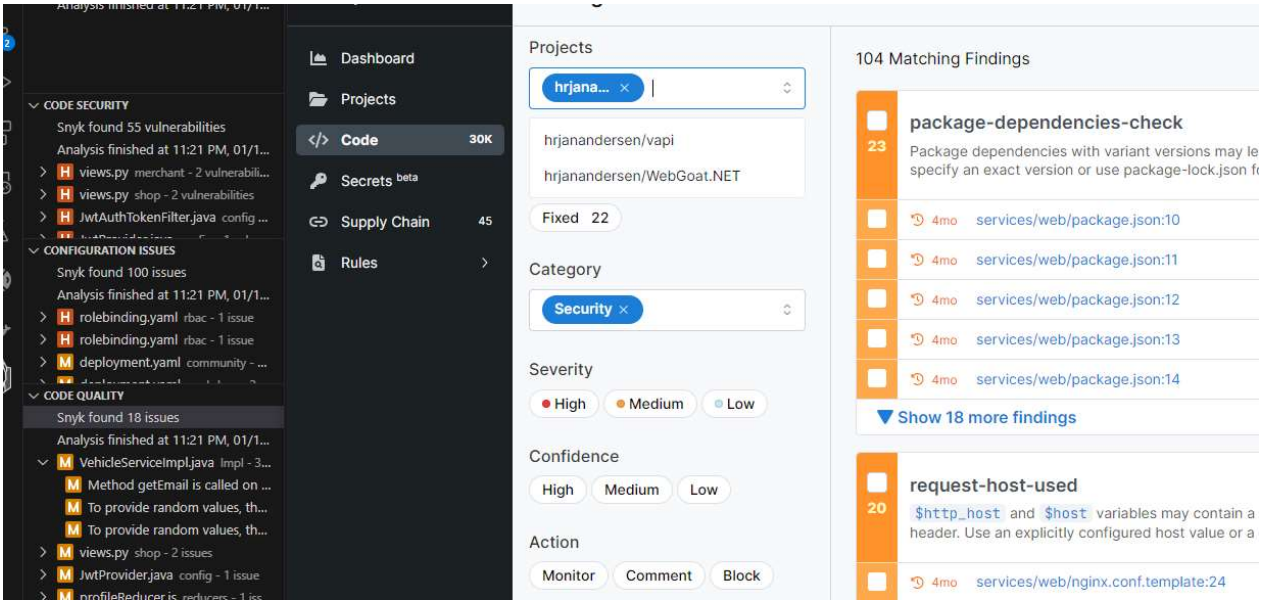


Figure 6.A: Compare Snyk and Semgrep for number of security findings. Source: snyk.io and semgrep.dev

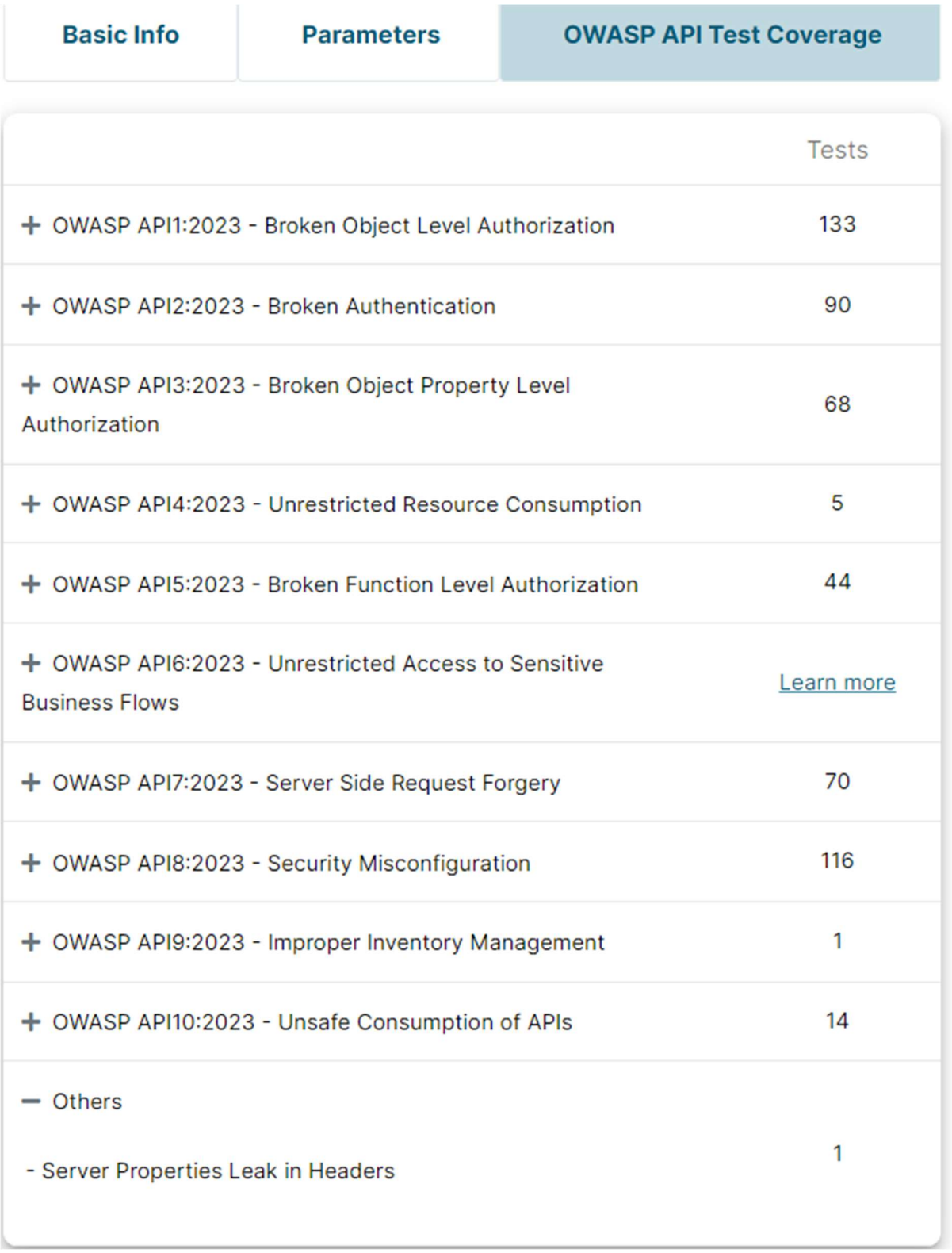


Figure 7.A: APISEC OWASP API Top 10 coverage. Source: https://www.apisec.ai/product#scan

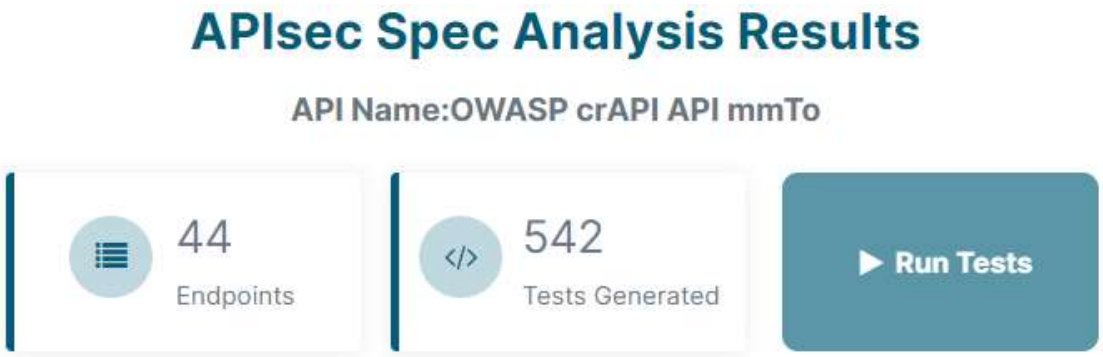


Figure 8.A: APIsec Spec Analysis Results. Source: https://www.apisec.ai/product#scan