Nick Van der Meer
[Email address]

Version	Changes
0.1	Initiation Document, adding risk assessment, owasp reasoning and conclusion

Table of Contents

OWASP TOP 10

De Owasp top 10 bestaat uit de volgende principes:

- Broken Acces Control
- Cryptographic Failures
- Injection
- Insecure Desgin
- Security Misconfiguration
- Vulnerale and Outdated Components
- Identification and Authentication Failures
- Software and Data Integrity Failures
- Security Logging and Monitoring Failures
- Server Side Request Forgery

OWASP Reasoning

In the following section I will explain what the principle is and how it applies to my application

Broken Acces Control

“Access control enforces policy such that users cannot act outside of their intended permissions. Failures typically lead to unauthorized information disclosure, modification, or destruction of all data or performing a business function outside the user's limits. Common access control vulnerabilities include examples like: “Bypassing access control checks by modifying the URL (parameter tampering or force browsing), internal application state, or the HTML page, or by using an attack tool modifying API requests. “

In mijn microservices applicatie wordt hier rekening mee gehouden door geen User ID’s mee te geven via parameters in een request. Ten eerste wordt er gebruik gemaakt van UUID’s. Door hier gebruik van te maken en niet van integers, wordt het al een stuk moeilijker om de ID van een andere gebruiker te raadden. Daarnaast wordt alleen het ID opgeslagen in een JWT-token. Zodra hiermee geknoeid wordt, zoals bijvoorbeeld het ID veranderen, wordt de token ongeldig verklaard en is hij niet meer geldig.

Cryptographic Failures

The first thing is to determine the protection needs of data in transit and at rest. For example, passwords, credit card numbers, health records, personal information, and business secrets require extra protection, mainly if that data falls under privacy laws, e.g., EU's General Data Protection Regulation (GDPR), or regulations, e.g., financial data protection such as PCI Data Security Standard (PCI DSS).

In de microservices applicatie wordt hier rekening mee gehouden op een aantal verschillende vlakken. Het eerste is het encrypten van gevoelige informatie zoals de wachtwoorden. Hiervoor is gebruik gemaakt van Bcrypt.

Daarnaast is door de architectuurkeuze van microservices de mogelijkheid en de keuze gemaakt om profiel data en accountdata te scheiden in compleet verschillende databases. Zo heeft de ene database bijvoorbeeld email, wachtwoord en de profiel database NAW-gegevens. Mocht er dan om wat voor een reden dan ook een datalek zijn, zal niet alle data aan elkaar gekoppeld kunnen worden.

Als laatste is er een functie ingebouwd om alle gegevens van een gebruiker te kunnen verwijderen. Volgens de GDPR moeten alle Europese gebruikers de kans hebben om persoonsgegevens te verwijderen van een website. In mijn applicatie is hiervoor ook een mogelijkheid om alle persoonsgegevens te verwijderen.

Insecure Design

Insecure design is a broad category representing different weaknesses, expressed as “missing or ineffective control design.” Insecure design is not the source for all other Top 10 risk categories. There is a difference between insecure design and insecure implementation. We differentiate between design flaws and implementation defects for a reason, they have different root causes and remediation. A secure design can still have implementation defects leading to vulnerabilities that may be exploited. An insecure design cannot be fixed by a perfect implementation as by definition, needed security controls were never created to defend against specific attacks. One of the factors that contribute to insecure design is the lack of business risk profiling inherent in the software or system being developed, and thus the failure to determine what level of security design is required.

Ik heb mijn applicatie gemaakt aan de hand van de standaarden voor softwareontwikkeling en de verwachtingen die mijn collega's hebben voor het ontwikkelen van een applicatie. Ik heb een microservices architectuur gebruikt. In de microservices heb ik een 3-laags ontwerp gebruikt met interfaces die de lagen verbinden. Ik heb de solide principes gebruikt tijdens het ontwerpen van mijn applicatie. In de productie omgeving heb ik per microservice een database gebruikt in het kader van scalability

Security Misconfiguration

The application might be vulnerable if the application is, for example:

- Error handling reveals stack traces or other overly informative error messages to users.
- The software is out of date or vulnerable (see Principle 6: Vulnerable and outdated components)
-

De applicatie gebruikt .net 7 als backend taal. De foutafhandeling gebeurt via http-statuscodes. Alleen de statuscode wordt beantwoord met een request, de frontend (in dit geval een react-applicatie) toont de foutmelding.

Vulnerale and Outdated Components

Voor zowel de react app als alle .net microservices heb ik de meest recente versies van alle componenten gebruikt om alle mogelijkheden van kwetsbare of onveilige/outdated pakketten te omzeilen.

Identification and Authentication Failures

Confirmation of the user's identity, authentication, and session management is critical to protect against authentication-related attacks. There may be authentication weaknesses if the application:

- Permits automated attacks such as credential stuffing, where the attacker has a list of valid usernames and passwords.
- Permits brute force or other automated attacks.
- Permits default, weak, or well-known passwords, such as "Password1" or "admin/admin".
- Uses weak or ineffective credential recovery and forgot-password processes, such as "knowledge-based answers," which cannot be made safe.
- Uses plain text, encrypted, or weakly hashed passwords data stores (see A2: Cryptographic Failures).
- Has missing or ineffective multi-factor authentication.
- Exposes session identifier in the URL.
- Reuse session identifier after successful login.
- Does not correctly invalidate Session IDs. User sessions or authentication tokens (mainly single sign-on (SSO) tokens) aren't properly invalidated during logout or a period of inactivity.

De applicatie gebruikt meerdere vormen van identificatie en authenticatie. Elke functionaliteit/component die gevoelige gegevens/gebruiker gerelateerde functies verwerkt, is beveiligd met een JWT Token. Dit token wordt gegenereerd door een geldige inlogpoging en verloopt binnen een bepaalde tijd. Daarnaast is er een signature toegevoegd aan het token. Deze moet overeenkomen om de geldigheid te verifiëren. In mijn geval gaat dit om een string van 20 random tekens om de veiligheid van de signature te vergroten. Dit token is nodig voor interactie met de backend. Voor deze toepassing lag de nadruk echt op het JWT Token systeem. De applicatie heeft bijvoorbeeld geen functionaliteit om je wachtwoord te wijzigen, controleert niet op de "sterkte" van een wachtwoord of Multi-factor authenticatie. Zodra men een tool als bijvoorbeeld keycloak gebruikt, zit deze functionaliteit er wel bij, maar valt voor dit project buiten de scope

Security Logging and Monitoring Failures

Returning to the OWASP Top 10 2021, this category is to help detect, escalate, and respond to active breaches. Without logging and monitoring, breaches cannot be detected. Insufficient logging, detection, monitoring, and active response occurs any time:

- Auditable events, such as logins, failed logins, and high-value transactions, are not logged.
- Warnings and errors generate no, inadequate, or unclear log messages.
- Logs of applications and APIs are not monitored for suspicious activity.
- Logs are only stored locally.
- Appropriate alerting thresholds and response escalation processes are not in place or effective.
- Penetration testing and scans by dynamic application security testing (DAST) tools (such as OWASP ZAP) do not trigger alerts.
- The application cannot detect, escalate, or alert for active attacks in real-time or near real-time.

De applicatie monitord of logt bepaalde data transfers binnen de applicatie niet actief.