# **Security by design Kwetter project**

Risk Analysis

Risicoanalyse is het proces van het identificeren en analyseren van mogelijke problemen die een negatieve impact kunnen hebben. Dit proces wordt gedaan om organisaties te helpen deze risico's te vermijden of te beperken. Het uitvoeren van een cyberbeveiliging risicoanalyse helpt bij het identificeren, beheren en beveiligen van gegevens, informatie en activa die kwetsbaar kunnen zijn voor een mogelijke cyberaanval.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Bedreiging** | **Type** | **Risico** | **Impact** | **Security level** |
| **Gebruikersgegevens lek** | Leak/Theft | Medium | High | High |
| **Admin inloggegevens lek** | Leak/Theft | Medium | High | High |
| **onveilige connectie met 3th party** | Vulnerability | Medium | Medium | Medium |
| **Outdated libraries of SDK** | Vulnerability | Medium | Medium | Medium |

# **Threat analysis**

Er wordt een threat analyse gemaakt om de mogelijke bedreigingen en hun potentiële schade voor een organisatie of project in kaart te brengen. In onderstaande tabel zijn de mogelijke bedreigingen voor het project opgesomd en hun mogelijke motivatie en misbruik activiteiten gedefinieerd.

|  |  |  |
| --- | --- | --- |
| **Threat Actor** | **Motivation** | **Misuse activities** |
| **Rogue medewerker** | Bemachtigen van gegevens, Blackmail material, stalking | Access |
| **Cybercriminelen** | Bemachtigen van gegevens, Blackmail material, stalking | Access, hacking tools |
| **Script Kiddies** | Learning, Fun, Thrill | Hacking tools |
| **Overheid** | Locaties, informatie, gegevens | Group hacking, Hired hackers |
| **Andere organisaties/concurrentie** | Bemachtigen gegevens, blackmail, faishiment | Hacking, access, rogue medewerker, hacker groep |
| **3rd party** | Bemachtigen data, blackmail, wraak | Hacking, access, hacking groep |
| **Onbekend** | Dit is een onbekende gebruiker die mogelijk in de toekomst een rol kan spelen | Dit is een onbekende gebruiker die mogelijk in de toekomst een rol kan spelen |

Misuse cases diagram

Een misbruik case diagram wordt gebruikt om mogelijke bedreigingen in kaart te brengen en zien hoe deze kunnen worden uitgevoerd op het project dat zal worden gemaakt. Het misbruik geval heeft de gebruikers en de functies die ze kunnen uitvoeren, en de dreiging actoren met de dreigingen die ze kunnen uitvoeren op de bestaande functies.

# 

# **OWASP top 10**

De OWASP top 10 is een lijst met de top tien meest voorkomende security kwetsbaarheden. Sinds dat onze applicatie een webapplicatie is zullen we kijken naar de OWASP top 10 voor webapplicaties. We zullen per kwetsbaarheid in de OWASP kijken wat er gedaan kan worden om de kwetsbaarheid te voorkomen in de applicatie.

1. **Injectie**

De meest gebruikte injectie om ongewenst toegang te krijgen door hackers, is de SQL injectie. Met een SQL injectie voert de hacker een query direct uit op de database, waarmee diegene data verkrijgen of zichzelf toegang verlenen. Het kan ook zijn dat mede door een SQL injectie de hele applicatie niet meer werkt en alle data verloren gaat.

De bekendste injectiemethoden zijn:

* SQL injectie
* OS commando injectie
* LDAP injectie

**Injectie binnen het project**

Om te voorkomen dat hackers gebruik kunnen maken van injectie binnen het connected airlines project hebben we een aantal stappen genomen. Zo heeft elke backend een eigen database met stored procedures. Hierdoor staat de query los van de code en kan deze op databaseniveau aangepast worden. Ook worden alle stored procedures aangeroepen door middel van zogenaamde “geparametriseerde queries”. Dit zorgt ervoor dat eventuele parameters niet in de query worden meegenomen die daardoor kwaadaardig kan zijn.

Aan de frontend kan er gebruik worden gemaakt van invoervalidatie. Invoervalidatie zorgt ervoor dat de invoer type, de lengte, het formaat, enzovoort goed is. Alleen de waarde die validatie doorstaat, kan worden verwerkt. Het helpt eventuele opdrachten die in de invoerreeks zijn ingevoegd, tegen te gaan. In zekere zin is het vergelijkbaar met kijken wie er klopt voordat de deur wordt geopend.

1. **Authenticatie fouten**

Wanneer je een webapplicatie met een gebruikersaccount hebt, dan is het zeer aannemelijk om te zeggen dat er een authenticatiesysteem wordt gebruikt. Het terugsturen van foutmeldingen die een hacker potentieel kunnen helpen bij het vaststellen of een gebruikersaccount bestaat, vormt een behoorlijk risico. Door een fout implementatie van deze authenticatiesysteem kunnen gebruikersgegevens zoals gebruikersnamen en wachtwoorden buitgemaakt worden.

**Authenticatie binnen het project**

In het project wordt er gebruik gemaakt van een authenticatie systeem. Door het gebruik van generale foutmeldingen en het min mogelijk gebruik maken van het laten zien van gevoelige data proberen wij de kans op gebroken authenticatie zo veel mogelijk te verminderen. Onze authenticatie draait ook volledige op de server side. Hierbij worden de inloggegevens maar een keer verstuurd en alleen maar op de server side verwerkt. Dit kan de kans op het achterhalen van gegevens verkleinen, zeker als er een kwetsbaarheid zou zitten aan de client side.

1. **Gevoelige data**

Vrijwel alle webapplicaties werken met zeer gevoelige data. Het is dus belangrijk om deze goed af te schermen, door middel van een goede encryptie. Inmiddels hebben wij hier ook een wetgeving dat ons verplicht om goed met gevoelige data om te gaan, genaamd de AVG wetgeving.

**Gevoelige data binnen het project**

Er zijn een aantal manieren om te zorgen dat gevoelige data van gebruikers van de applicatie niet zomaar in de handen van kwaadwillenden kan vallen. Zo is het belangrijk om alleen data op te slaan die relevant is voor de werking van de applicatie. Wanneer telefoonnummers niet gebruikt worden in de applicatie moeten deze ook niet worden opgeslagen. Data die niet wordt opgeslagen kan ook niet worden gelekt.

Ook is het belangrijk dat data die wel opgeslagen wordt, versleuteld wordt. Door gebruik te maken van nieuwe encryptietechnieken kan een hacker nog steeds niks met de data ook bij een volledige lek van de database. Door te zorgen dat de wachtwoorden aan bepaalde eisen voldoen worden aanvallen als brute-force ook een stuk minder bruikbaar.

1. **Externe XML entities (XXE)**

Hackers kunnen door middel van een XML bestand zichzelf toegang verlenen tot bestanden en programma’s op het besturingssysteem van de server. Dit komt met name door gedateerde XML verwerkers. Het zou ook mogelijk kunnen zijn dat de hacker data aan koppelingen, zoals bijvoorbeeld met een boekhoudpakket, door kan sturen met foutieve data en daardoor geld kan overmaken naar een bankrekening.

**Externe XML entities binnen het project**

Om te zorgen dat deze aanval onbruikbaar wordt is het nodig om eventuele XML parsers binnen het project up to date te houden en goed in te stellen. Het is namelijk mogelijk binnen een xml parser alle externe entiteiten te blokkeren. Hiermee worden xml-bestanden die door een gebruiker worden ingevoerd gewoonweg niet behandeld. Deze manier van preventie kan ook bepaalde Denial of Services aanvallen voorkomen, bijvoorbeeld *Billion Laughs*.

1. **Rollen en rechtensysteem kwetsbaarheden**

Wanneer een gebruiker foutieve rechten krijgt bij zijn rol, kan de gebruiker eventueel meer handelingen verrichten dan gewenst is. Het is daarom erg belangrijk dat het rollen- en rechtensysteem goed op elkaar afgestemd en getest wordt.

**Rollen en rechtensysteem kwetsbaarheden binnen het project**

Het is belangrijk dat de verschillende rollen binnen een systeem goed beschreven staan. Zo moet een administrator veel meer mogen dan een “gewone” gebruiker. Dit moet getest worden met unit- en systeemtesten. Dit voorkomt dat een gewone gebruiker bij functionaliteiten komt waar hij of zij niet bij hoort te mogen.

Ook moeten de endpoint van een API goed afgebakend worden met het gebruik van tokens. Dit wordt binnen ons project dan ook gedaan. Bij elke afgebakende API call wordt eerst gecheckt of de gebruiker een valide token heeft met de juiste rechten. Dit voorkomt ook rechtstreekse calls naar de API, aangezien een valide token alleen uitgegeven kan worden door de API.

1. **Foutieve configuratie**

Een ontwikkelaar gebruikt configuratiebestanden om zijn server en frameworks in te stellen, die absoluut niet inzichtelijk mogen zijn voor het openbare publiek. Het komt verrassend vaak genoeg voor dat een ontwikkelaar zijn configuratie niet op orde brengt voor het besturingssysteem van de server en van de gebruikte frameworks, wat de hacker in staat kan stellen om toegang te krijgen.

**Foutieve configuratie binnen het project**

Binnen ons project is het met name belangrijk om de servers goed af te sluiten en alleen poorten open te stellen die nodig zijn voor de werking van de applicatie. Het komt wel eens voor dat te veel poorten open staan waardoor bepaalde protocollen uitgevoerd kunnen worden op een server die dan weer kwetsbaarheden kunnen blootstellen.

1. **Cross-Site Scripting (XSS)**

Als een webapplicatie onvoldoende validatie toepast op de te verwerken data, kan het voorkomen dat een hacker code injecteert in de webapplicatie. Hiermee kan de hacker bijvoorbeeld onveilige bestanden plaatsen op de computer van een andere gebruiker, maar ook de webapplicatie volledig vernietigen.

**Cross-Site Scripting (XSS) binnen het project**

XSS is een van de meest voorkomende en dus belangrijke manieren voor een hacker om een applicatie binnen te dringen. Dit gebeurt met name door invoervelden niet te valideren waardoor een kwaadwillende Javascript code uit kan voeren voor elke gebruiker. Wanneer een gebruiker een account maakt en hier als naam <script>alert(“Ik ben een hacker”)</script> invult en dit niet gevalideerd wordt, krijgt iedere gebruiker een pop-up venster te zien wanneer de naam van die specifieke gebruiker geladen wordt. Dit kan makkelijk voorkomen worden door invoervalidatie en wordt in de meeste frameworks al automatisch geïmplementeerd. Ook binnen ons project zorgt React.JS ervoor dat XSS niet kan worden uitgevoerd.

1. **Onveilige serialisatie**

Er zijn webapplicaties die objecten omzetten naar tekst, al voor deze in de database opgeslagen worden. Wanneer een webapplicatie hiervan gebruik maakt en niet voldoende controles uitvoert op de ingevulde data, kan de hacker commando’s uitvoeren op het besturingssysteem van de gebruikte server.

**Gebruik van serialisatie in het project**

In ons project maken we geen gebruik van serialisatie, er worden geen bestanden zoals afbeeldingen, video’s of tekstbestanden opgeslagen in een database of file server met behulp van serialisatie. Hierdoor is dit niet van toepassing op ons project.

1. **Gebruik van software met bekende kwetsbaarheden**

Vaak maakt een ontwikkelaar gebruik van open-source frameworks of stukken software dat door andere ontwikkelt worden. Als deze niet tijdig geüpdatet worden door de makers en/of de gebruikers, kunnen veiligheidsrisico’s benut worden door hackers.

**Gebruik van software in het project**

In het project gebruiken wij verschillende libraries of software pakketten. De pakketten die we gebruiken voor bijvoorbeeld hashing, proberen we zoveel mogelijk officiele of pakketten die regelmatig upgedate worden te gebruiken. Door pakketten te gebruiken die officieel zijn (zoals van java) of die goede ondersteuning krijgen is de kans op (langdurige) kwetsbaarheden erg klein.

1. **Onvoldoende logging en monitoring**

Feitje, een lek wordt gemiddeld pas na 200 dagen gemeld bij de makers van de webapplicatie. Als de makers van de webapplicatie onvoldoende of zelfs helemaal geen logbestanden bijhoudt, waarin vastgehouden wordt wat er precies gebeurd binnen de applicatie kan het heel lang duren voor de makers het lek boven water krijgen.

Met behulp van monitoring heeft een ontwikkelaar tevens veel sneller in de gaten of een webapplicatie nog wel bereikbaar is en kan hierop beter reageren wanneer dit niet het geval is.

**Monitoring in het project**

Binnen het project proberen we zo veel mogelijke belangrijke events te monitoren. Hiermee kunnen we belangrijke data vastleggen die mogelijk bugs, kwetsbaarheden of hacks kunnen aantonen. In het project worden activiteiten gelogd die te maken hebben met het verkrijgen van (gevoelige) data, hieruit kan wanneer er afwijkende patronen zijn een mogelijk hack, kwetsbaarheid of rogue medewerker uitkomen. Hiernaast gebruiken we de logging ook om errors te registreren waarmee we mogelijk bugs of kwetsbaarheden kunnen vinden.