OWASP ResearchDocument

Mickey en Matthijs

Inhoud

[ResearchMethodes 3](#_Toc60824323)

[Onderzoeksvraag 4](#_Toc60824324)

[Hypothese 4](#_Toc60824325)

[Wat zijn onze grootste valkuilen qua security binnen ons project? 5](#_Toc60824326)

[Inleiding 5](#_Toc60824327)

[Security 5](#_Toc60824328)

[Owasp 5](#_Toc60824329)

[1. Injection 5](#_Toc60824330)

[2. Broken Authentication 6](#_Toc60824331)

[3. Sensetive data exposure 7](#_Toc60824332)

[4. Xml external entities 8](#_Toc60824333)

[5. Broken acces control 9](#_Toc60824334)

[6. Security Misconfiguration 10](#_Toc60824335)

[7. Cross-site Scripting xss 10](#_Toc60824336)

[8. Insecure deserialization 11](#_Toc60824337)

[9. Using components with known vulnerabilities 12](#_Toc60824338)

[10. Insufficient logging and monitoring 13](#_Toc60824339)

[Wat: 13](#_Toc60824340)

[Hoe: 13](#_Toc60824341)

[Security test 14](#_Toc60824342)

[Sanitization and security 14](#_Toc60824343)

[XSS Sanitization loading in a htmlsnippet 14](#_Toc60824344)

[XSS Sanitization Url loading 16](#_Toc60824345)

[conclusie 18](#_Toc60824346)

[Best Good and bad practices 19](#_Toc60824347)

[Good 19](#_Toc60824348)

[Gateway 19](#_Toc60824349)

[activemq 19](#_Toc60824350)

[mongodb 19](#_Toc60824351)

[Bad 20](#_Toc60824352)

[Cors 20](#_Toc60824353)

[Consul 20](#_Toc60824354)

[Authentication token 20](#_Toc60824355)

[Code review 21](#_Toc60824356)

[Owasp Zap 21](#_Toc60824357)

[Voordelen Zap 21](#_Toc60824358)

[Gebruik van Zap 21](#_Toc60824359)

[Passive mode 21](#_Toc60824360)

[Spider 22](#_Toc60824361)

[Active Mode 23](#_Toc60824362)

[Owasp resultaat 25](#_Toc60824363)

[Conclusie 26](#_Toc60824364)

[Conclusie 27](#_Toc60824365)

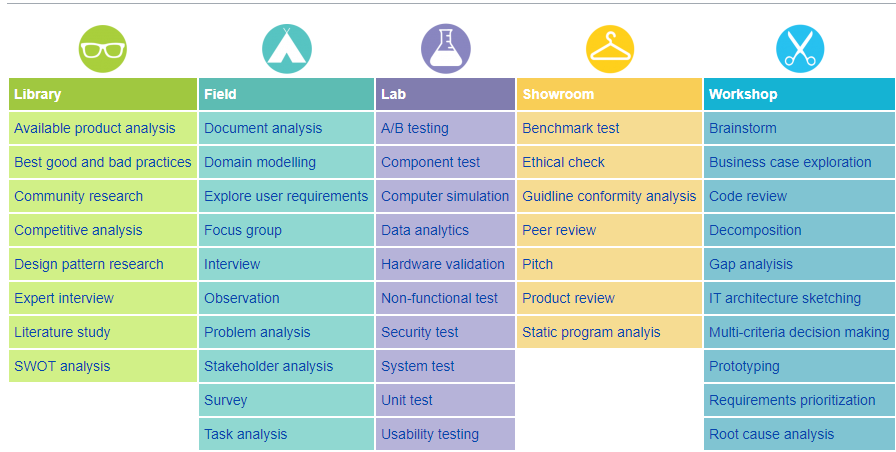
[Nabespreking 27](#_Toc60824366)

# ResearchMethodes

Binnen ons project is het verstandig om bepaald onderzoek te doen naar verschillende aspecten die misschien gevoelig kunnen zijn. Hierbij kun je denken aan een stuk security en het afweren van aanvallers of aan het schalen van je project als er meerdere gebruikers komen.

Om dit onderzoek van start te laten gaan hebben we eerst een onderzoeksvraag nodig over het onderwerp dat we willen behandelen. Nadat we de onderzoeksvraag hebben gesteld hebben we nog een hypothese nodig om ons uiteindelijke antwoord mee te vergelijken en dus te kijken of we gelijk hadden en of we meer te weten zijn gekomen.

Na deze stappen kunnen we gebruik maken van de verschillende researchmethodes om zo meer te weten te komen over ons project en om dichter bij het antwoord op de vraag te komen.



(bron 1)

Hierboven staan de methodes beschreven. Elke soort heeft zijn eigen voor en nadelen maar om de vraag juist te kunnen beantwoorden is het handig als we een aantal uit verschillende categorieën pakken zodat we meer vlakken raken op het gebied van onze research.

Nadat al het onderzoek voltooid is kun je de onderzoeksvraag beantwoorden en dus een conclusie trekken. Aan de hand van deze conclusie ben je hopelijk beter geworden rondom het onderwerp en weet je meer hierover.

# Onderzoeksvraag

**Wat zijn onze grootste valkuilen qua security binnen ons project?**

## Hypothese

Wij denken dat de grootste security valkuilen liggen bij het niet hebben van de authorization tokens dus blijven gebruikers altijd ingelogd. Het andere grote probleem is het hebben van Sensetive data exposure, wij maken veel gebruik van api’s en omdat we dit niet hebben kan iedereen bij deze api’s.

# Wat zijn onze grootste valkuilen qua security binnen ons project?

## Inleiding

Onze applicatie draait met een microservices architectuur, dit zorgt ervoor dat er al snel valkuilen zitten binnen de applicatie zelf. Wij maken namelijk een game waar currency een belangrijk doel van het spel is. Om deze reden is het al snel gevoelig voor cheaters en hackers die snel extra currency op hun account willen zetten. Hierdoor moeten wij onze applicatie volledig beschermen tegen dit soort acties.

## Security

Er zijn ontzettend veel manieren om je applicatie te beveiligen en dit maakt het zo moeilijk voor ons omdat wij de juiste vormen van de applicatie moeten gaan vinden. Uiteraard willen we de focus leggen op het beveiligen van de frontend omdat hier de gebruikers naartoe zullen komen. Maar ook het beveiligen van de backend en voornamelijk de currency en de wachtwoorden is van groot belang. De currency wordt gekocht met echt geld of langzaam verdient en lokt dus al snel hackers naar zich toe. Om wat meer research te gaan doen naar deze security methodes maken wij gebruik van de OWASP lijst waarin verschillende security risks staan beschreven.

## Owasp

### Injection

#### Wat:

Injectie fouten zoals SQL injectie vinden plaats wanneer er geen data maar een commando als data wordt verstuurd, zo kunnen aanvallers een hoop data achterhalen.

#### Hoe:

##### Input Validation:

Kijken en verifiëren of de input van de gebruiker wel mag volgens het programma, hierbij moet je bijvoorbeeld denken aan whitelisten van data.

##### Parameterized queries:

Hiermee kun je het verschil zien tussen input van een gebruiker en een script of commando, hierdoor kun je dus voorkomen dat er überhaupt een script geactiveerd kan worden op de applicatie.

##### Stored procedures

Met stored procedures groepeert het programma in een bepaald plan van executie, je hoeft dan niet iedere keer een bepaalde query aan te maken maar haalt dan een voor gegenereerde op.

### Broken Authentication

#### Wat:

Applicaties met authenticatie worden vaak verkeerd geimplementeerd waardoor aanvallers mogelijk session keys, wachtwoorden en gebruikers kunnen achterhalen.

#### Hoe:

##### Multi factor authentication

Dit is waarbij de applicatie een andere applicatie vraagt om de gegevens nog een keer te controleren, hierdoor voorkom je makkelijk brute force attacks tegen gaan

##### Hashing

Hier hashen wij het wachtwoord waardoor deze niet makkelijk te decrypten is.

##### Login timeout

Als de gebruiker x aantal keer fout heeft ingelogd moet er een timeout komen zodat niemand een brute force attack zou kunnen gebruiken

##### Strong password check

Het controleren of een wachtwoord sterk genoeg is voorkomt het gebruik van rainbow lists tegen de applicatie.

##### Session keys

Genereert willekeurige session id’s per gebruiker om te detecteren wanneer er ergens anders is ingelogd.

### Sensetive data exposure

#### Wat:

Veel api’s verbergen belangrijke data niet zoals financiele gegevens, aanvallers veranderen deze data of stelen het om dingen zoals fraude te kunnen plegen.

#### Hoe:

##### Verschillende wachtwoorden

Het gebruik van verschillende wachtwoorden per account kan ervoor zorgen dat als er een wachtwoord gelekt wordt dat een aanvaller niet toegang heeft tot alles.

##### Https

Gebruik maken van een https verbinding zorgt ervoor dat we veilig data versturen en dat een aanvaller dit niet zomaar kan vangen.

##### GateWay

Een gateway geeft een extra laag security bovenop onze applicatie die ervoor zorgt dat niet iedereen zomaar alle apis kan aanroepen.

### Xml external entities

#### Wat:

Vooral oude XML processors evalueren alles in xml documenten waardoor aanvallers deze kunnen toesturen om zo aanvallen binnen in het systeem te krijgen.

#### Hoe:

##### Uitschakelen Parsing libraries

Het uitschakelen van de functies die in de parsing xml libraries zitten zorgt ervoor dat xml niet meer blootgesteld wordt aan potentiele aanvallers.

##### Json

Het gebruik van minder complexe gegevensindelingen kan er al voor zorgen dat xml niet blootgesteld wordt.

##### White listing

Het gebruik van een whitelist invoervalidatie betekend dat alleen gevalideerde gebruikers(of admins) xml bestanden aan kunnen passen en niet zomaar willekeurige gebruikers.

### Broken acces control

#### Wat:

Restricties van wat een gebruiker wel en niet mag doen worden vaak slecht gecontroleerd, hierdoor kan iemand die niet geautoriseerd is makkelijk op plekken komen waar hij niet hoort te zijn en of kunnen komen.

#### Hoe:

##### Toegang

Om broken access te kunnen voorkomen is het weigeren van toegang door het zetten van ‘private’ erg gemakkelijk

##### Gebruik controleLijsten

Het gebruik van controle en rollenlijsten zorgt ervoor dat niet elke gebruiker dezelfde rechten heeft, een admin kan bijvoorbeeld op een pagina waar een gebruiker niet op kan

##### Verbergen

Het verbergen van functies en variabelen net zoals sommige modellen en controllers kan er al voor zorgen dat een aanvaller niet zomaar in een applicatie kan kijken.

### Security Misconfiguration

#### Wat:

Een verkeerde configuratie van de beveiliging is het meest voorkomende probleem. Dit is meestal het gevolg van onveilige standaardconfiguraties, verkeerd geconfigureerde HTTP-headers en uitgebreide foutmeldingen met gevoelige informatie.

#### Hoe:

##### DebugginG uitschakelen

Het uitschakelen van debugging zorgt ervoor dat een aanvaller niet binnen de code kan kijken.

##### Standaard accounts en wachtwoorden uitschakelen

Het uitschakelen van testwachtwoorden zorgt ervoor dat niet iedereen kan inloggen als een admin.

##### Server initialiseren

De server initialiseren om alleen ingelogde mensen toe te laten werkt tegen aanvallers die op een server proberen te komen

### Cross-site Scripting xss

#### Wat:

Met XSS kunnen aanvallers scripts uitvoeren in de browser van het slachtoffer die gebruikerssessies kunnen over nemen, websites kunnen beschadigen of de gebruiker kunnen omleiden naar kwaadwillende sites.

#### Hoe:

##### Hoe vaak komt XSS vulnerabilities voor?

XSS-kwetsbaarheden komen veel voor, en XSS is waarschijnlijk de meest voorkomende kwetsbaarheid op het gebied van webbeveiliging.

##### Hoe vaak komt XSS attacks voor?

Het is moeilijk om betrouwbare gegevens te krijgen over echte XSS-aanvallen, maar er wordt waarschijnlijk minder vaak misbruik van gemaakt dan andere kwetsbaarheden.

##### Wat is het verschil tussen XSS and CSRF?

XSS houdt in dat een website kwaadaardig JavaScript retourneert, terwijl CSRF een slachtoffergebruiker ertoe aanzet om acties uit te voeren.

##### Wat is het verschil tussen XSS and SQL injection?

XSS is een kwetsbaarheid aan de clientzijde die zich richt op andere toepassingsgebruikers, terwijl SQL-injectie een kwetsbaarheid aan de serverzijde.

##### Hoe voorkom je XSS in PHP?

Filter uw invoer met een white list met toegestane tekens en gebruik typetips of typecasting. Ontsnap aan uw uitvoer met htmlentitiesen ENT\_QUOTES voor HTML-frameworks, of JavaScript Unicode-escapes voor JavaScript-frameworks.

##### Hoe voorkom je XSS in Java?

Filter uw invoer met een witte lijst met toegestane tekens en gebruik een bibliotheek zoals Google Guava om uw uitvoer HTML-coderen voor HTML-contexten, of gebruik JavaScript Unicode-escapes voor JavaScript-contexten.

### Insecure deserialization

#### Wat:

Onveilige deserialisatie leidt vaak tot uitvoering van externe code. Dit kan worden gebruikt om aanvallen uit te voeren, waaronder injection -aanvallen en aanvallen met privilege escalation aanvallen.

#### Hoe:

###### HOE VOORKOM JE iNSECURE DESERIALIZATION:

1. Het implementeren van integriteitscontroles zoals digitale handtekeningen op alle geserialiseerde objecten om het knoeien met gegevens te voorkomen.
2. Het isoleren en uitvoeren van code die indien mogelijk deserialiseert in omgevingen met weinig bevoegdheden.
3. Het registreren van uitzonderingen en fouten bij deserialisatie, zoals wanneer het inkomende type niet het verwachte type is, of de deserialisatie genereert uitzonderingen.
4. Inkomende en uitgaande netwerkverbindingen van containers of servers die deserialiseren, beperken of bewaken.
5. Monitoring van deserialisatie, waarschuwen als een gebruiker constant deserialiseert.

### Using components with known vulnerabilities

#### Wat:

Componenten, zoals libraries, frameworks en andere software modules, worden uitgevoerd met dezelfde rechten als de applicatie.  Als een kwetsbare component wordt misbruikt, kan een dergelijke aanval leiden tot ernstig gegevensverlies of serverovername.

#### Hoe:

##### Voorkomen insuddicient logging

###### HANDMATIGE UPDATES

Een optie is om geen componenten te gebruiken die u niet hebt geschreven. De meeste libraries en frameworks creëren geen kwetsbaarheidspatches voor oude versies. In plaats daarvan lost u het probleem eenvoudig op in de volgende versie. Upgraden naar deze nieuwe versies is dus een manier om dit te voorkomen.

###### GEBRUIK HDIV

Detectietools voor kwetsbare software van HDIV bevorderen een simpelere aanpak, waarbij softwareafhankelijkheid wordt geanalyseerd, zowel tijdens de bouwtijd als tijdens runtime, om gemakkelijk kwetsbare stukjes software te detecteren die moeten worden vervangen door nieuwere versies. De tools zijn ontworpen om de hele levenscyclus van applicaties te dekken:

### Insufficient logging and monitoring

### Wat:

Onvoldoende logging en monitoring, in combinatie met ontbrekende of oneffectief integration met incident response, stelt aanvallers in staat om systemen verder aan te vallen.

### Hoe:

###### Voorkomen van insuddicient logging and monitoring

1. Zorg ervoor dat alle aanmeldingsfouten, fouten bij toegangscontrole en validatiefouten aan de serverzijde kunnen worden gelogd met voldoende gebruikerscontext om verdachte accounts te identificeren.
2. Zorg ervoor dat logboeken worden gegenereerd in een indeling die gemakkelijk kan worden gebruikt door gecentraliseerde oplossingen voor logboekbeheer.
3. Zorg ervoor dat transacties met een hoge waarde een audit trail hebben met integriteitscontroles om manipulatie of verwijdering te voorkomen.
4. Zorg voor effectieve monitoring en alarmering zodat verdachte activiteiten tijdig worden gedetecteerd en beantwoord.

# Security test

Het eerste en waarschijnlijk ook het belangrijkste onderdeel van het testen van de security zelf, hierbij kunnen we hopelijk gemakkelijk te zien krijgen wat er wel en niet getest is en tegelijkertijd een beeld te schetsen van security problemen in onze applicatie.

## Sanitization and security

### XSS Sanitization loading in a htmlsnippet

cross-site scripting ook wel XSS genoemd zijn injectie aanvallen, waarbij kwaad willige scripts worden geïnjecteerd in het slachtoffer zijn website. Gelukkig heeft angular build in Sanitization and security, Die voorkomt dat scripts binnen de HTML geinjecteert kunnen worden.

Dit is een demo project waarbij ik een XSS Attack simuleer.

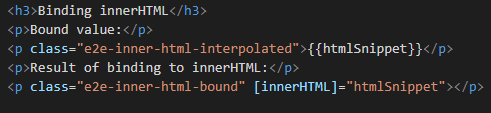
Dit is de ExampleXSSAttackComponent.ts file. Hier in laad ik html snippet in met een Script tag.



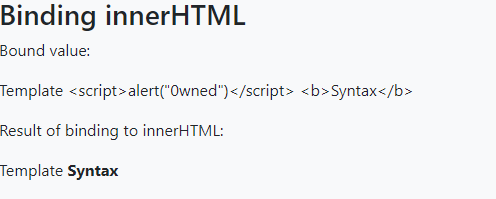
In dit voorbeeld als het niet word gestopt word er alleen maar een Alart melding in de console getoond. Maar in een echte situatie zou het een kwaad willig script kunnen zijn.



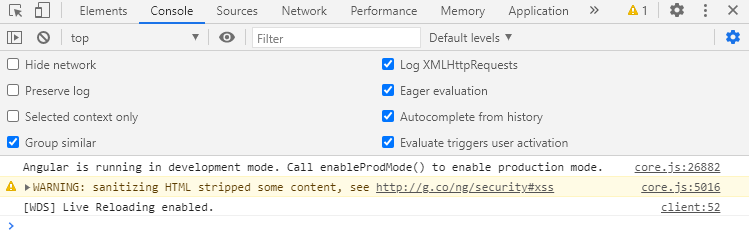
Dit laad ik vervolgens in de Html via een innerHTML.



Maar zo als het resultaat laat zien word dit door angular gezien en word dit niet uit gevoerd.



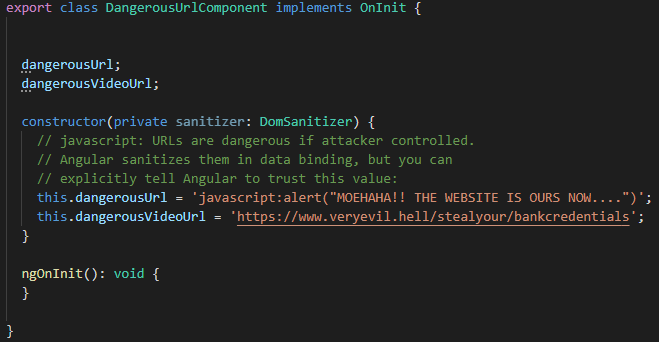
De console laat zelf zien dat de sanitizing de script tag verwijderd heeft.



### XSS Sanitization Url loading

Angular Biedt ook bescherming tegen het inladen en erheen routen van mogelijke gevaarlijke URL’s. in sommige applicaties moeten uitvoerbare code bevatten bijvoorbeeld een <iframe>van een bepaalde URL weergeven.

Dit is een demo project waarbij ik een aanvaller simuleer die de user naar een andere URL wil laten routen. Dit project bevat 2 URLS waarvan een URL die schadelijke code bevat die word uitgevoerd door het klikken van een knop. En een die een video toont van een onbekende URL.



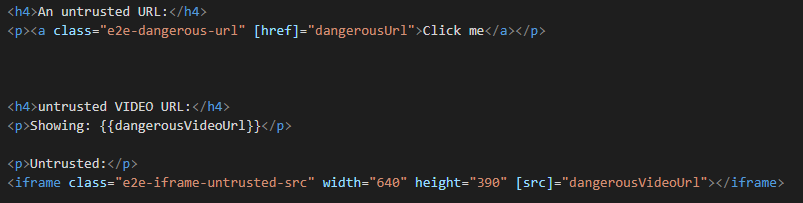
Deze URL voert een kwaadwillige script uit.



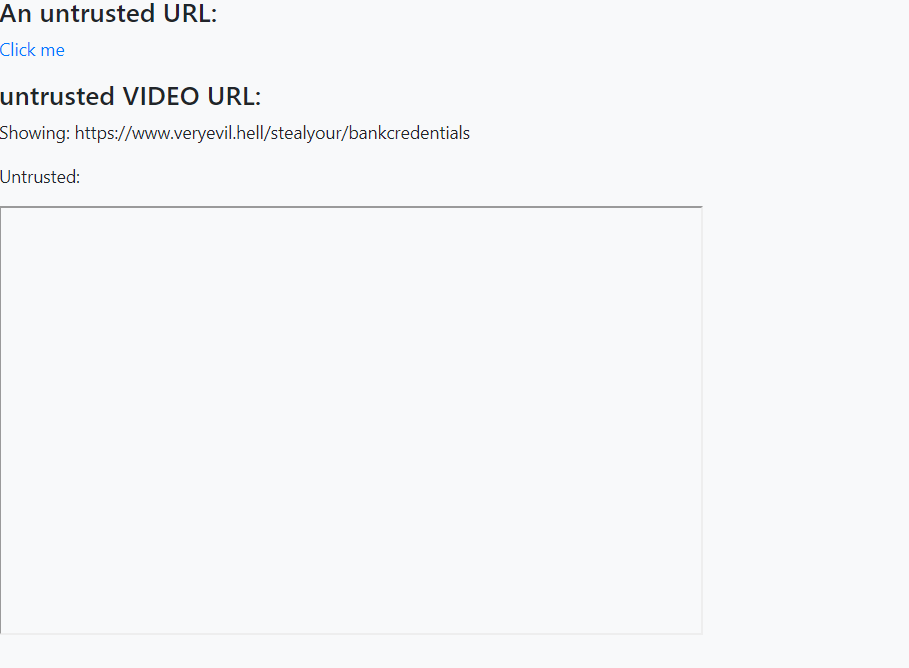
Deze URL kan de user mogelijk naar een kwaadwillige website leiden.



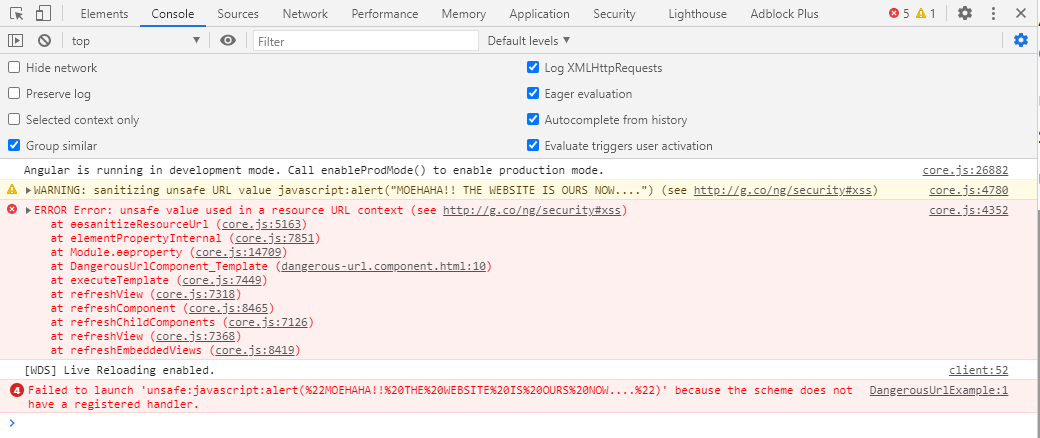
Dit is de HTML waar de URL’s worden ingeladen via een href en een src.



Angular toned de website.



En de kwaad willige URL’s worden geblokkeerd.



## conclusie

Angular heeft een build in beveiliging tegen XXS attacks, wat bij het implementeren van een webapplicatie erg handig is. Na dat ik meerdere test had uit gevoerd om het systeem te omzeilen blokkeerde Angular me nog steeds. In het kort ik ben blij dat we als front-end framework Angular gekozen hebben.

# Best Good and bad practices

Om zelf een beter beeld te krijgen met wat goed en slecht is qua security willen we een good and bad practices maken. Hierbij willen we graag laten zien wat er goed en slecht is voor het implementeren van beveiliging om zo ook zelf na te gaan denken wat er fout is bij security.

Voor de good en bad practices willen we graag gaan kijken binnen ons eigen project, we willen graag dieper duiken in de componenten die we slecht en goed vinden qua beveiliging zodat we een beter beeld krijgen bij de qualiteit van ons project.

## Good

### Gateway

Het gebruiken van een gateway tussen de front-end en de back-end heeft veel voordelen.

Het bied bescherming tegen overmatig gebruik en misbruik. Het bied ook als een hub voor authenticatie en beveilig checks. Ook Analyseren-en monitoring tools kunnen makkelijker worden toegevoegd. Het is vooral een erg goede optie als je met een microservice architectuur werkt. Wanneer microservices vervallen of worden vervangen zullen de users de services nog steeds op dezelfde plek kunnen vinden.

Voor ons project gebruiken wij de C# library Ocelot, de reden is dat Ocelot veel mogelijkheden biedt op het uitbreiding van onze app. Het toevoegen van authenticatie en het verbinden met bijv. een discovery service (Consul)is erg makkelijk.

### activemq

Eigenlijk is het hebben van een messagebroker al een good practice in zichzelf, een messagebroker zorgt er namelijk voor dat de berichten die verstuurd zijn altijd aan komen, zelfs als een api of service halverwegen stopt. Dit zou dramatisch kunnen zijn als de broker er niet tussen zat omdat er dan berichten verloren zouden kunnen gaan.

#### Security

ActiveMq zorgt direct niet zelf voor security, maar als je deze goed op hebt gezet kan het gemakkelijk ook voor security zorgen. We zorgen ervoor dat alle data via Dto’s gaat dus als iemand er iets in zou sturen wordt dit eruit gefilterd door active mq zelf, hier wordt dus niks mee gedaan. Ook om requests te versturen heb je een wachtwoord nodig van activemq zelf, dit kun je niet zomaar aanroepen omdat we alleen de requests manager buiten de firewall hebben staan en niet de User interface. Dit bij elkaar zorgt voor een veilige omgeving voor api’s om requests naar elkaar te versturen.

### mongodb

De integratie van mongodb heeft ons ook al een stap in de juiste richting gezet qua security. Door deze repository compleet onzichtbaar te maken voor de gebruiker kan er niemand ook misbruik maken hiervan.

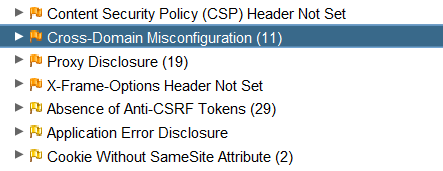
Mongodb zelf werkt met een url die je in moet voegen om de connectie te maken, mensen die deze url niet weten zullen ook nooit weten waar de data vandaan komt en hebben heir dus weinig aan.

In tegenstelling tot sql databases die gevoelig zijn voor sql injection is Mongodb dit niet of in ieder geval een stuk minder. Ook omdat we de requests via Http doen en controleren met Dto’s zal er ook nooit zomaar een commando door onze api’s heen komen en zorgen voor onnodige problemen.

## Bad

### Cors

Een van de ‘bad practices’ die we nog in ons project hebben zitten is onze cors policy. Aan het begin van ons project hebben we veel moeite gehad met deze ingebouwde beveiliging en hebben deze dus uitgeschakeld. Hierdoor is wel het probleem veroorzaakt dat de applicatie en de api’s minder goed zijn beveiligd. In Owasp zap kun je ook met een active scan zien dat hierbij problemen kunnen komen kijken.



Hier kun je al zien dat op onze frontend alleen al 11 problemen zijn ontstaan binnen het cors probleem. Het is dus duidelijk verstandig om deze functie beter erin te zetten omdat we nu iedereen toelaten, als we dit beveiligen met een eigen sleutel per api of door alleen bepaalde ip’s toe te laten in de applicatie zal de applicatie hier beter tegen zijn beveiligd.

### Consul

Opzich is consul een prima discovery service en er zit ook een hele hoop in. Je hebt health checks en kan altijd de url zien van api die het nodig heeft. Hier komt automatisch ook het probleem kijken. Omdat de discovery service dusdanig gebouwd is dat de requests via dezelfde port gaan als de User interface kun je (als je de Url weet) gemakkelijk alle url’s van de microservices achterhalen.

Het probleem ligt dan met name bij het feit dat de gateway dit ook gebruikt dus consul er nog voor ligt. Hierdoor krijg je dus de echte links naar de microservices en niet die van de gateway.

Hiernaast hebben we niet ver genoeg in Consul gedoken en weten we niet hoe we dit moeten beveiligen. Er staat ook op internet dat het opzich niet nodig is om een gebruikersnaam en wachtwoord te hebben voor deze service en dat dit ook veel moeite kost.

Om deze redenen is Consul niet heel veilig in zichzelf en is het niet handig om zo’n service maar alf te integreren zoals het bij ons project nu in elkaar zit.

### Authentication token

Authorization token bieden ontzettend veel voordelen binnen de applicatie op het gebied van beveiliging. Deze tokens zijn ‘decoupled’, dit houdt in dat het zowel op de user staat als in de backend waardoor er een extra laag beveiliging buiten het inloggen zit.

Een ander voordeel is dat deze tokens bescherming bieden tegen mogelijkke XSS en XSRF aanvallen die makkelijk gevaarlijke scripts injecteren door bugs in de beveiliging van de webapplicatie.

Momenteel hebben wij de authorization tokens nog niet geimplementeerd maar gebruiken wij alleen de id van de gebruiker. Hierdoor is de applicatie minder beveiligd en kunnen aanvallers direct toegang krijgen tot een stukje data van de backend. Dit is gevaarlijk en vandaar dat wij dit ook als bad-practice hebben beschreven binnen ons project.

# Code review

Als laatste willen we gaan kijken naar onze api’s en daarover een review maken. De beste manier om dit te kunnen doen is via sonarqube. Sonarqube is in het kort gezegd een static code analysis tool. Deze laat de code coverage, code smells en de security risks zien en hoewel we alleen de security nodig hebben is dit alsnog een uitstekende tool voor het maken van een code review op security basis.

## Owasp Zap

Ons eerste plan was om sonarqube te gaan gebruiken voor de code review, dit is een uitstekende keuze voor bugs en de basis controle, maar dit is niet handig voor als we willen gaan testen binnen een applicatie. Om dit wel te kunnen doen hebben we even gezocht naar een alternatief. Hier kwamen we bij de volgende video uit:

<https://www.youtube.com/watch?v=8fdh7CntpmE> . hierin werd beschreven waarom en hoe je Owasp kunt gebruiken, hier hebben we dus verder ingedoken omdat deze applicatie(net als sonarqube), ook een test rapport uitschrijft, deze kun je zelfs in je CiCd pipeline verwerken voor een geautomatiseerde test.

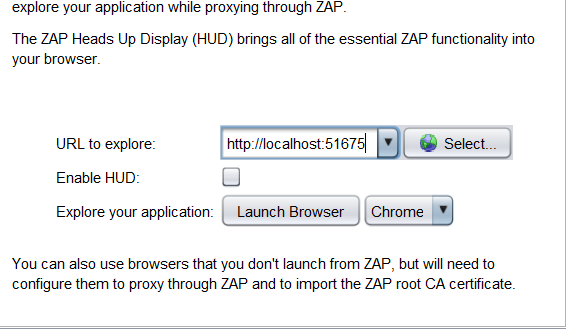
## Voordelen Zap

We hebben nog niet naar concurrentie gezocht voor een code review tool in het security opzicht maar zap kwam bij veel mensen als nr. 1 omdat deze veel ingebouwde tools heeft en het testrapport makkelijk te converteren naar een excel bestand.

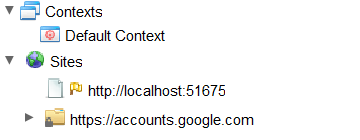
## Gebruik van Zap

### Passive mode

In passive mode ga je zelf door de applicatie heen en kijkt zap of er eventuele breach punten zijn in je applicatie, dit werkt doormiddel van een ‘spider’. Deze spider gaat elke url die jij in de gegenereerde browser invult controleren. Owasp valt de applicatie hierbij niet aan en kijkt hij alleen maar of er gevoeligheden zijn.



Op zap vullen we eerst de url in van de website die we willen testen, in dit geval is dat localhost omdat we onze applicatie lokaal aan het draaien zijn. Maar deze site kan net zo goed online staan.



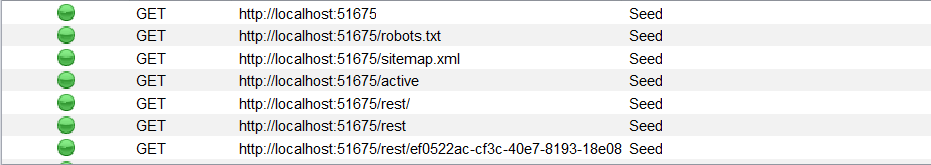
Op het moment dat we deze browser openen komt er links in de applicatie een file bij te staan van de site waar we naartoe zijn gegaan, in ons geval staat er direct een gele vlag bij. Dit houdt in dat er een minimale breach is ontstaan, zonder dat ik iets heb gedaan.

Als we meer willen weten over onze api en waar deze fout op gaat kunnen we de ‘spider’ gebruiken binnen owasp.

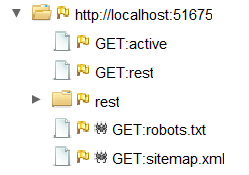
### Spider

Een spider in Owasp wordt gebruikt om nieuwe url’s te achterhalen op een website / api. Deze vind de spider voornamelijk in hyperlinks of als de gebruiker naar een andere url gaat.

Op het moment dat de spider een hyperlink vind kan deze automatisch weer gescand worden door owasp zelf en krijgen we daar weer de breaches weergegeven.



Hierboven staat een sequence weergegeven van acties die ik zelf heb uitgevoerd, ik heb eerst **/rest** gedaan om alles op te halen en vervolgens **/{id}** erbij gezet om een specifiek dataobject aan te vragen. Het groene rondje geeft aan dat de spider daar op de site heeft kunnen komen.



Als je vervolgens weer naar het explorer scherm kijkt zie je dat er 2 onderdelen bij zijn gekomen van de spider (spin). Robot.txt en sitemap.xml.

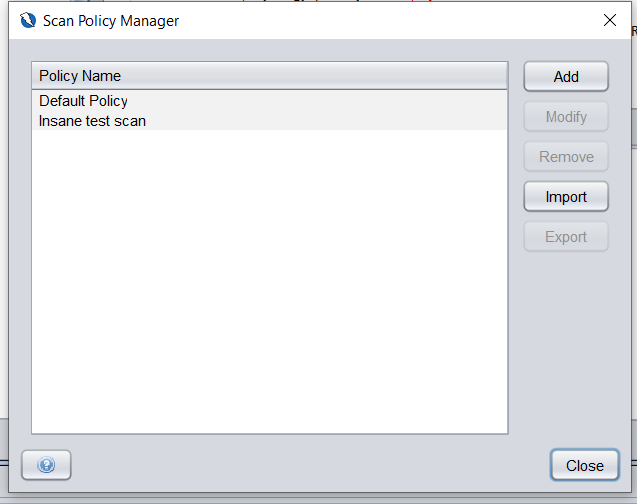
De **Robot.txt** is het configuratie gedeelte van de spider, hierin wordt aangegeven wat de spider allemaal wilt ophalen.

De **Sitemap.xml** is eigenlijk de plek waar alle mappings van de sites worden opgeslagen (de acties die ondernomen moeten worden).

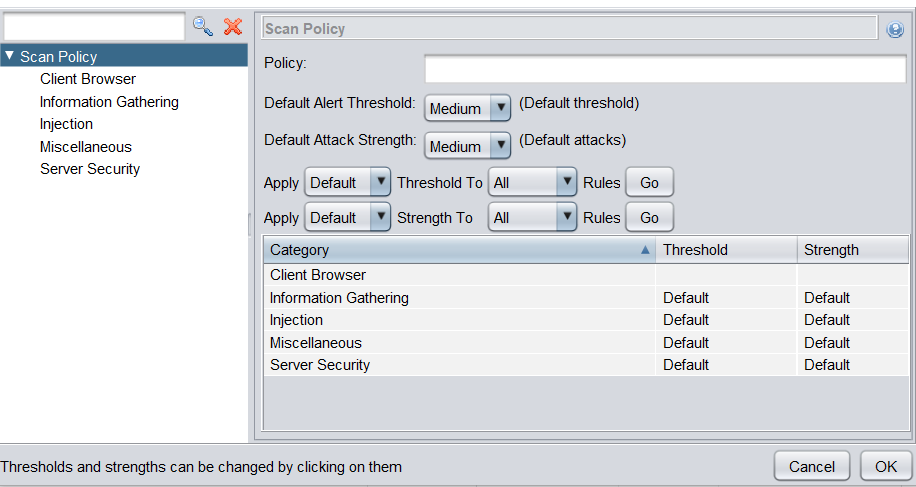
### Active Mode

Een active scan valt de webapplicatie echt aan, hierbij gaat owasp kijken hoe de applicatie misbruikt zou kunnen worden met fouten die in de database zelf staan. Hierbij kun je ook je eigen scripts toevoegen om zo een eigen scan te maken. Maar je kunt ook de voor gegenereerde gebruiken.

Als je wilt scannen moet je eerst een nieuwe policy aanmaken, dit doe je door naar Analyse in de toolbar te gaan en vervolgens op policy manager te klikken.



Hierin kun je tests aanpassen, importeren en aanmaken.



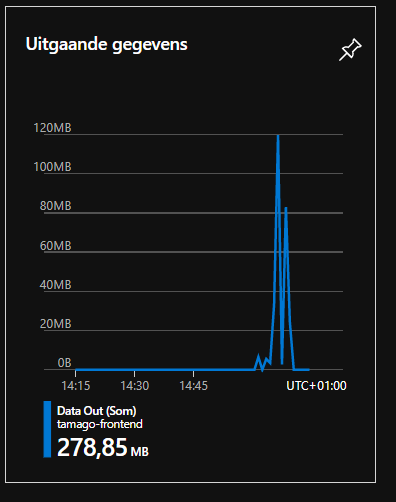
Als je er een wilt toevoegen krijg je het volgende scherm, hierin kun je aangeven welke sterkte van aanval je wilt en op wel aspect van de app je het wilt draaien.

Als je eenmaal de policy hebt gemaakt en een spider op de passive scan hebt laten werken heb je als het goed is al een lijst met de verschillende urls waar Owasp naartoe kan gaan. Als je vervolgens op active scan drukt via het attack menu kun je daar je policy selecteren en de active scan zijn gang laten gaan. (dit duurt een tijdje).

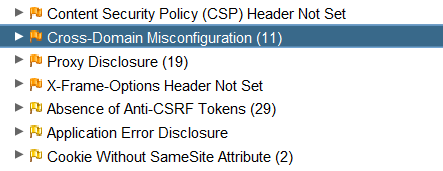
### Owasp resultaat



Een scan van onze eigen frontend resulteerde in 9223 requests die waren verstuurd en ook 72 nieuwe alerts.



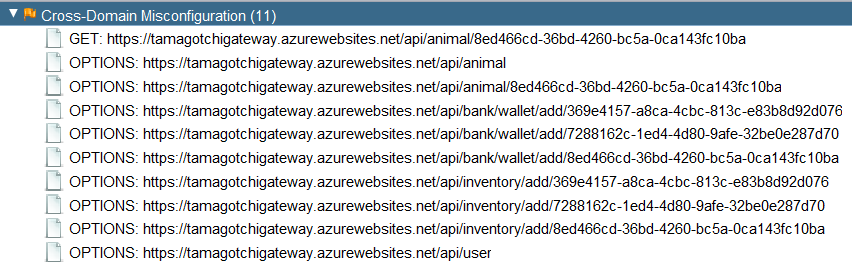
Ook zijn de uitgaande gegevens van onze azure app gigantisch omhoog gegaan door deze scan.



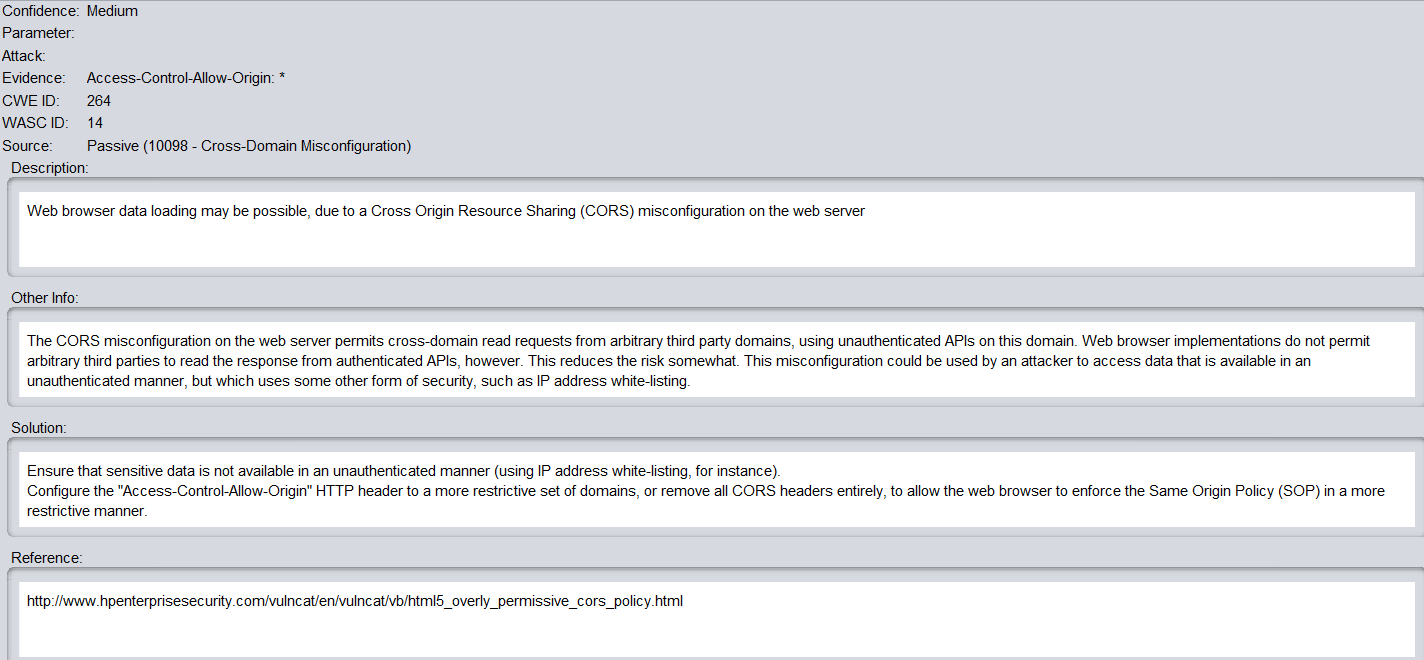
Bij onze alerts zitten veel dezelfde problemen zoals de proxy ‘Disclosure’ en de ‘Cross-domain misconfiguration’. Om deze te kunnen verhelpen kunnen we dieper in de problemen duiken.

#### Resultaat

Bij de alerts staan gelukkig ook waar ze vandaan komen, wat het inhoud en hoe je het zou kunnen verhelpen.



Bij ons cross domain alert kunnen we zien waar die allemaal in is voorgekomen, je kunt dus zien dat er veel requests hetzelfde probleem hebben.



Verder zien we zoals hierboven wat het probleem is, hoe groot het probleem is en hoe je het kan verhelpen. Dit probleem ligt met name aan CORS dus is voor ons nu niet van grote waarde omdat we dit risico al wisten, maar als je als bedrijf hier nog niet aan had gedacht kan dit van groot belang zijn.

## Conclusie

Owasp zap is een uitstekende keus als het gaat om code reviews ten opzichte van de security. Owasp zap schrijft een rapport uit waardoor wij gemakkelijk kunnen beoordelen of de applicatie goed genoeg is voor deployment of niet, voor ons heeft dit al een hele hoop geholpen met het vinden van security fouten en is gemakkelijk te gebruiken.

# Conclusie

We hebben veel onderzoek gedaan naar security en zijn achter een veel handige tools gevonden en deze ook kunnen implementeren om de beveiliging van een applicatie te verbeteren, maar wat is nou de beste manier van beveiligen?

Het belangrijkste van het beveiligen is dat je de kleine componenten niet vergeet, in ons project zijn dat voornamelijk de authentication tokens, de Cors policy en de Consul discovery service die momenteel zonder wachtwoord te bereiken is. Via Owasp zap zijn we erachter gekomen dan Cors een groter probleem is dan we dachten en ook vaker een probleem is. Ook via owasp kwamen we achter de authentication tokens die dus ook een probleem waren.

Het is dus voornamelijk belangrijk om te blijven testen en externe tools te gebruiken om achter deze kleine fouten te komen.

# Nabespreking

Het onderzoek ging erg goed, we kwamen achter een hoop tools en ingebouwd testmateriaal wat we voorheen nog niet wisten. Dit heeft ons niet alleen met het onderzoek geholpen maar vooral met de beveiliging van het project.

We hebben het onderzoek synchroon gedaan met het implementeren van de tools binnen het project waardoor we ook gelijk een stukje uit ervaring spraken en hierdoor de keuzes ook beter konden onderbouwen.

.