[Inleiding 2](#_Toc516778603)

[Performance 2](#_Toc516778604)

[Repository pattern 2](#_Toc516778605)

[Cache 2](#_Toc516778606)

[Asynchroon 3](#_Toc516778607)

[Watch history 3](#_Toc516778608)

[Minimaliseren van database Locking 3](#_Toc516778609)

[Queuing 3](#_Toc516778610)

[Security 4](#_Toc516778611)

[1. Injection 4](#_Toc516778612)

[2. Broken authentication 4](#_Toc516778613)

[3. Sensitive data exposure 4](#_Toc516778614)

[4. XML External Entities (XXE) 4](#_Toc516778615)

[5. Broken access control 4](#_Toc516778616)

[6. Security misconfiguration 4](#_Toc516778617)

[7. Cross site scripting (XSS) 4](#_Toc516778618)

[8. Insecure Deserialization 5](#_Toc516778619)

[9. Using components with known vulnerabilities 5](#_Toc516778620)

[10. Insufficient logging & Monitoring 5](#_Toc516778621)

[Resultaten Netsparker security test voor de implementatie van security verbeteringen 5](#_Toc516778622)

[Resultaten OWASP ZAP security test voor de implementatie van security verbeteringen 5](#_Toc516778623)

[Oude top 10 issues 7](#_Toc516778624)

[Cross site request forgery (CSRF) 7](#_Toc516778625)

[Insecure cryptographic storage 7](#_Toc516778626)

[Failure to restrict URL access 7](#_Toc516778627)

[Insufficient transport layer protection 7](#_Toc516778628)

[Unvalidated redirects and forwards 7](#_Toc516778629)

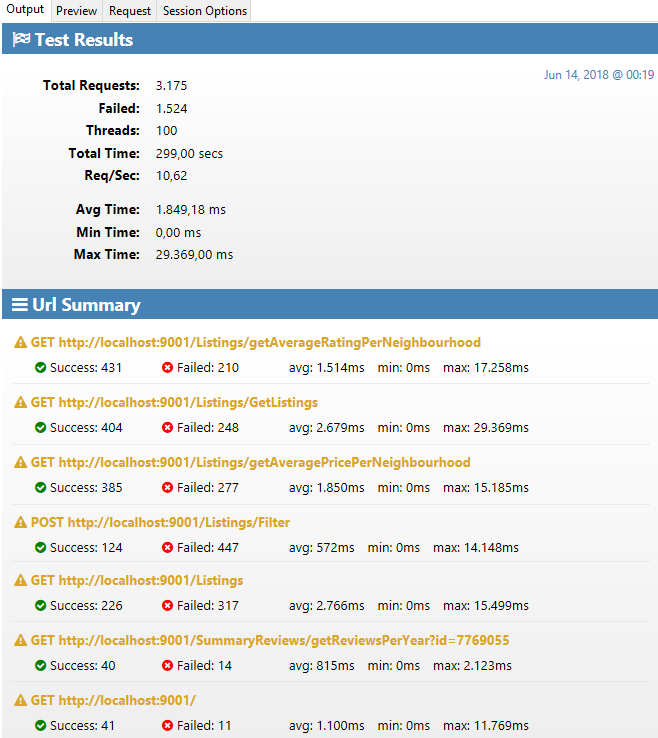
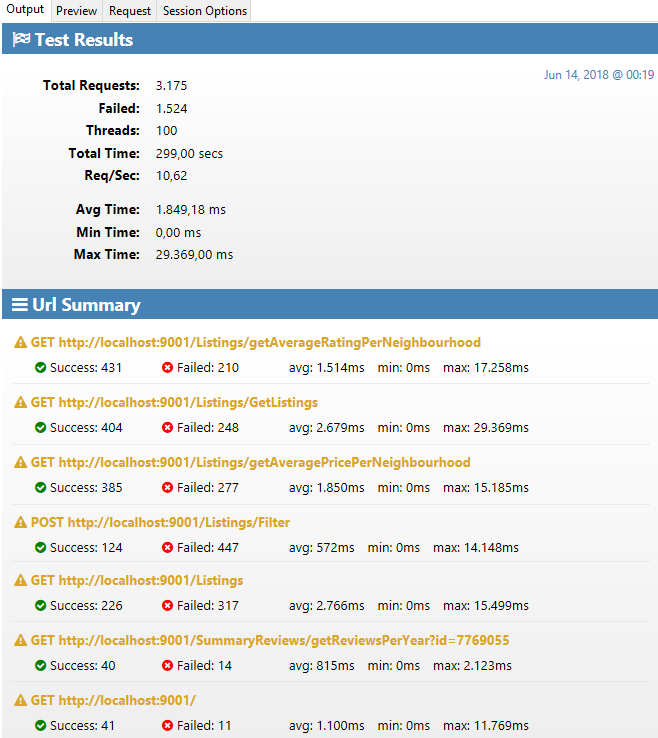
# Inleiding

Deze opdracht gaat over het populaire hotel internetplatform "AirBNB. Door de aanhoudende stroom met gebruikers is door de founders besloten om de applicatie stabieler op te zetten en deze te laten draaien op een cloud-platform. Omdat het werken met cloud oplossingen betrekkelijk nieuw is, hebben zij mijn hulp ingeschakeld voor de ontwikkeling en implementatie van een nieuwe webapplicatie. Onderzoek heeft opgeleverd dat het Google Cloud Platform en ASP.Net MVC voor deze toepassing de voorkeur hebben. Een belangrijk aspect van deze applicatie is dat het streeft naar een goede prestatie(snelheid), schaalbaarheid en veiligheid.

# Performance

In dit hoofdstuk zal aan de hand van load test resultaten beschreven worden welke performance verbeteringen doorgevoerd worden met hierop volgend nieuwe resultaten van dezelfde load testen. Op deze manier zal aangetoond worden dat deze applicatie ontwikkeld is met een van de voornaamste eisen dat de performance weinig te wensen over laat.

## Lokale test resultaten voor performance verbeteringen



Zoals te zien op bovenstaande afbeelding zijn dit de resultaten van een loadtest met 50 users die browsen op de AirBNB website. Deze resultaten zijn van voor de doorvoering van onderstaande performance verbeteringen. Hier duurde het gemiddelde request nog 2,571 seconden. Bovendien faalde 547 van de in totaal 2423 request door een timeout, dit is een failrate van 22,58%.

Hier nog beschrijven dat er steeds meer gefaalde requests zijn waardoor de avg time false positive naar beneden gaat. EN duidelijk benoemen dat dit de voor resultaten zijn. Hieronder dan de zelfde load tests draaien met geoptimaliseerde performace en dan een conclusie schrijven met vergelijkingen. Ook beschrijven welke acties ondernomen zijn om de performance te verbeteren.

## Repository pattern

Er is besloten om het repository pattern te gebruiken. Hiervoor heb ik vooral gekozen omdat je standaard SQL functies niet zelf hoeft te implementeren op deze manier. Bovendien is dit te testen d.m.v. unit tests in plaats van integration tests.

## Cache

Vanuit het oogpunt gericht op performance is het weinig moeite voor heel veel performance winst om caching toe te passen. Om deze reden heb ik besloten caching toe te passen in de applicatie. Het toepassen van caching zal gebeuren bij database queries die er lang over doen om uitgevoerd te worden.

Als caching methode zal de in memory cache gebruikt worden, dit houdt in dat het ram geheugen gebruikt zal worden als opslag hiervoor. Dit werkt snel, maar kan al snel vervelend worden als er niet genoeg ram beschikbaar is op de server. Hiervoor zou het beter zijn om bijvoorbeeld een redis cache te gebruiken, echter wordt dit nog niet ondersteund in combinatie met MVC 5.

De cache zal bijvoorbeeld gebruikten worden voor de resultaten van een slome query, mits deze resultaten niet vaak veranderen. Een voorbeeld waar het toegepast wordt is bij de query om populaire films van altijd en van de afgelopen twee weken op te halen. Deze resultaten veranderen niet snel maar om dit te berekenen kost wel veel tijd, dit komt doordat er gejoined(samengevoegd uit meerdere tabellen), gecount, geselecteerd en gesorteerd wordt in de query. Het lijkt mij slim om deze resultaten 24 uur in de cache te laten leven, met als reden om elke 24 uur nieuwe films toe te voegen en/of bijwerken. En dat de beheerder deze pagina’s dan één keer opent zodat deze resultaten opnieuw in de cache gezet worden en de gebruikers de komende 24 uur deze pagina’s snel kunnen bekijken.

Een ander voorbeeld waar caching handig kan zijn is bij de pagination. De query voor pagination kost ook veel resources omdat je bij een lijst met alle films in de database een count moet uitvoeren. Hier zal ongeveer 380.000 uitkomen met de database zoals die aangeleverd is. In dit geval zal het ook geen probleem zijn dat er soms met gedateerde data gewerkt wordt. In het ergste geval, als er veel films toegevoegd of verwijderd zijn sinds er een nieuwe versie in de cache is gezet, zal er één pagina missen of teveel zijn in de pagination balk. Dit is verwaarloosbaar omdat het onwaarschijnlijk is dat iemand de zoekfuncties negeert en helemaal met de pagination balk naar de laatste pagina gaan om daar een film aan te klikken.

De performance winst door caching zullen vastgelegd worden door middel van loadtest resultaten. Dezelfde test zal twee keer uitgevoerd worden, één keer voordat caching geïmplementeerd is en één keer hierna.

## Asynchroon

Het ophalen van een lijst met meeste populaire films door middel van een linq query is erg heftig en kost veel tijd/resources. Om deze reden heb ik dit resultaat gecached zoals hierboven vermeld staat. Echter zal de tijd nog te lang zijn naar mijn mening. Hierdoor wilde ik dit ophalen asynchroon laten gebeuren. Zodat er al wel een pagina te zien is tijdens het wachten, in plaats van een wit pagina totdat deze query actie klaar is. Echter dit is nog niet mogelijk in DNX 5, en zal dus nog niet geïmplementeerd worden.

## Watch history

Ik heb ervoor gekozen om de watch date uit de tabel ‘watchhistory’ aan te passen naar het type DataTime, dit is nodig omdat het van type date was en in C# is hier geen ondersteuning voor. Ik heb geprobeerd om de datetime in te vullen met de time leeg, alleen dan probeert hij hier 0000 in te vullen en dan lukt het niet in de DB. Ook heb ik geprobeerd het te converten naar een string, maar dit pikte de db ook niet op.

## Minimaliseren van database Locking

Het verminderen van database locking geeft een performance boost, het houdt in dat er in plaats van één database is, om naar te schrijven en van te lezen, er twee zijn. Er zal er een geoptimaliseerd moeten worden voor schrijf operaties en de ander voor lees operaties. Dit wordt niet toegepast binnen dit project gezien het opzetten van een tweede database en zorgen voor persistentie tussen beiden teveel tijd en dus geld kost.

## Queuing

Een queue is een flexibele buffer om taken, verkregen van webservers (in de vorm van messages) te verdelen over workers. De response op een message wordt in een response queue gestopt en met andere workers teruggegeven. Dit zorgt ervoor dat de applicatie persistenter wordt, de messages zijn duurzaam en zullen niet verwijderd worden.

Voordelen van het toepassen hiervan zijn dat load balancing automatisch wordt toegepast, en dat dit een lage koppeling heeft.

Nadelen zijn dat het tijd kost om te implementeren, je performance overhead krijgt, het complexiteit toevoegt aan het project en dat er een extra server opgezet moet worden, wat extra geld kost.

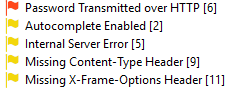
Deze nadelen zorgen ervoor dat dit voor dit project niet handig is om te gebruiken. Vooral dat er een tweede server nodig is maakt het onhaalbaar.

# Security

Voor de security van deze applicatie zal minstens verdedigd worden tegen de top 10 van de OWASP risico lijst. Ik zal kort beschrijven hoe dit in zijn werk gaat binnen AirBNB.

Veel van de top 10 bedreigingen worden al opgelost binnen het gebruikte .NET Core MVC framework, als dit zo is beschrijf ik hoe ze dit oplossen. Als dit niet het geval is zal ik beschrijven hoe dit het best op te lossen is en hoe ik dit in mijn applicatie verwerkt heb.

Hiernaast laat ik in dit hoofdstuk de resultaten van zowel de OWASP zap security test zien als de resultaten van de Netsparker security test. Deze zullen beschreven worden bij de bijbehorende kopjes van de OWASP top 10 als deze hieronder te groeperen zijn, zoniet zullen er extra kopjes voor gemaakt worden onderaan dit hoofdstuk.



## Injection

Injection is het laten uitvoeren van een database query op een plek waar dit niet de bedoeling is. Bijvoorbeeld bij een slecht beveiligde login pagina op een website waar je in plaats van inloggen database queries kan uitvoeren.

Om dit tegen te gaan zal ik gebruik maken van LINQ method based queries in plaats van SQL.

## Broken authentication

Broken authentication zal geen probleem zijn door het Identity framework die de wachtwoorden van gebruikers hashed. En hetzelfde geldt voor session management dit wordt afgevangen door het forceren van https.

## Sensitive data exposure

Https en auto fill forms

## XML External Entities (XXE)

## 5. Broken access control

Dit zijn directe referenties vanuit de link naar variabelen in de code of records uit de database. Als hier dan ook geen goede controle op zit of het wel de goede persoon is die dit aanvraagt kan het gebeuren dat mensen gegevens kunnen inzien die ze niet zouden mogen inzien.

Bij de watchhistory, wat gevoelige informatie is, heb ik het bewust gedaan zonder parameters. Ik controleer gewoon wie er is ingelogd en laat van die gebruiker de history zien. Bovendien heb ik op sommige actions dat alleen een admin dit kan doen, dus niet dat de gebruiker alleen ingelogd hoeft te zijn maar deze moet ook de rol van admin hebben. Denk hierbij aan editmovie en dergelijken.

## 6. Security misconfiguration

Dit gaat over of de software up to date is, op de server/db of alle onnodige dingen disabled zijn en dergelijken.

## 7. Cross site scripting (XSS)

XSS is het injecteren van code op plekken waar dit niet bedoeld is. Denk aan een inlog formulier.

Het MVC 5 framework encode alle html die als output wordt gegeven. Dit dient als beveiliging tegen cross site scripting. Het is dan ook niet nodig om hier nog verdere protectie aan toe te voegen tenzij het voor een functie nodig is om bijvoorbeeld @html.raw te gebruiken. Dit is een tag helper die ervoor zorgt dat de html niet ge-encode wordt. Dan zou je het wel op moeten vangen.

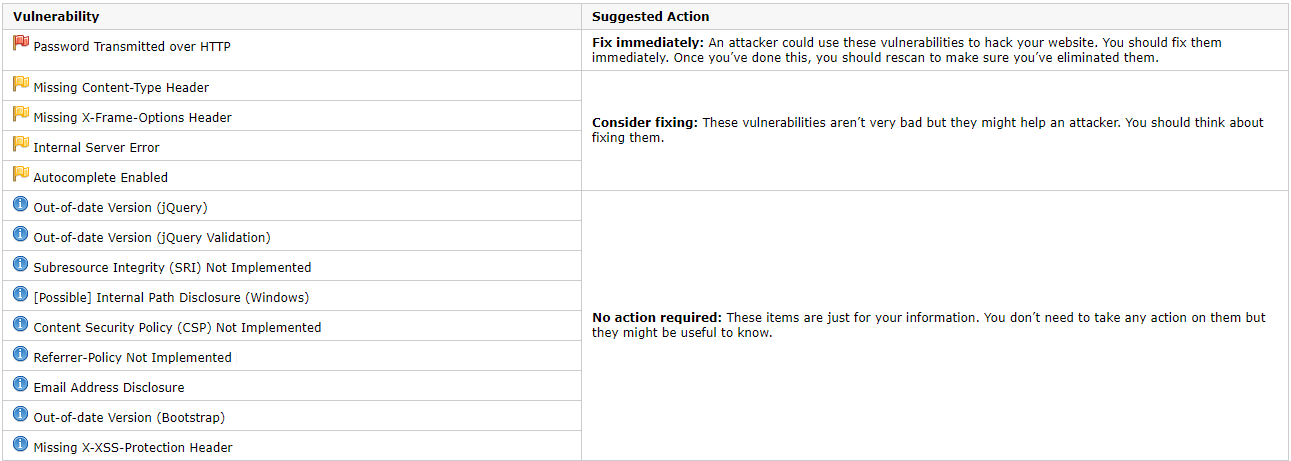
Verdere protectie zou anti-xss kunnen zijn alleen dit wordt nog niet ondersteund in mvc 5. Hiernaast kan je door middel van een reguliere expressie controleren of de input een html element bevat.

## 8. Insecure Deserialization

## 9. Using components with known vulnerabilities

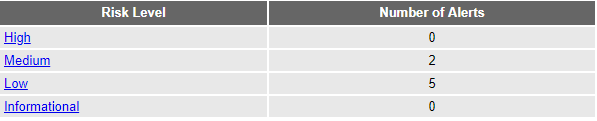
## 10. Insufficient logging & Monitoring

## Resultaten Netsparker security test voor de implementatie van security verbeteringen



Het versturen van wachtwoorden over http is zeker not done, ik heb dan ook besloten de applicatie op HTTPS te laten draaien.

## Resultaten OWASP ZAP security test voor de implementatie van security verbeteringen



|  |  |
| --- | --- |
| **Medium (Medium)** | **X-Frame-Options Header Not Set** |
| Description | X-Frame-Options header is not included in the HTTP response to protect against 'ClickJacking' attacks. |
|  |  |

|  |  |
| --- | --- |
| **Medium (Medium)** | **Format String Error** |
| Description | A Format String error occurs when the submitted data of an input string is evaluated as a command by the application. |
|  |  |

|  |  |
| --- | --- |
| **Low (Medium)** | **Cross-Domain JavaScript Source File Inclusion** |
| Description | The page includes one or more script files from a third-party domain. |
|  |  |

|  |  |
| --- | --- |
| **Low (Medium)** | **X-Content-Type-Options Header Missing** |
| Description | The Anti-MIME-Sniffing header X-Content-Type-Options was not set to 'nosniff'. This allows older versions of Internet Explorer and Chrome to perform MIME-sniffing on the response body, potentially causing the response body to be interpreted and displayed as a content type other than the declared content type. Current (early 2014) and legacy versions of Firefox will use the declared content type (if one is set), rather than performing MIME-sniffing. |
|  |  |

|  |  |
| --- | --- |
| **Low (Medium)** | **Cookie No HttpOnly Flag** |
| Description | A cookie has been set without the HttpOnly flag, which means that the cookie can be accessed by JavaScript. If a malicious script can be run on this page then the cookie will be accessible and can be transmitted to another site. If this is a session cookie then session hijacking may be possible. |
|  |  |

|  |  |
| --- | --- |
| **Low (Medium)** | **Password Autocomplete in Browser** |
| Description | The AUTOCOMPLETE attribute is not disabled on an HTML FORM/INPUT element containing password type input. Passwords may be stored in browsers and retrieved. |
|  |  |

|  |  |
| --- | --- |
| **Low (Medium)** | **Web Browser XSS Protection Not Enabled** |
| Description | Web Browser XSS Protection is not enabled, or is disabled by the configuration of the 'X-XSS-Protection' HTTP response header on the web server |
|  |  |

# Oude top 10 issues

## Cross site request forgery (CSRF)

CSRF is een lastig principe, het komt erop neer dat een andere site een request doet naar de server van fletnix. Zodra de gebruiker ingelogd is kan de sessie worden gebruikt om authenticated acties uit te voeren. Doordat je een request doet naar de site waarop is ingelogd wordt er een cookie meegestuurd naar de destination site. Dit zorgt ervoor dat jouw malicious website authenticated taken uit kan voeren.

Om dit tegen te gaan zal gebruik maken van een anti forgery token. Hierdoor zal een random string gegeneerd worden in het form en in een cookie, deze worden beiden meegestuurd naar de server en worden vergeleken. Het werkt omdat de malicious website niet de waarde uit de cookie kan lezen en dus niet weet wat hij in het hidden form field moet zetten.

Wat ik zeker ga implementeren is het niet laten zien van de error messages op de published website. Op de debug omgeving laat ik dit wel aan, omdat dit makkelijk is om te ontwikkelen. Ook kan het handig zijn om een custom error page te maken en deze te tonen aan de eindgebruikers. Verder worden de servers onderhouden door Azure, hier is geen omkijken naar.

## Insecure cryptographic storage

Identity gaat veilig om met de persoonsgegevens. Het hashed de wachtwoorden.

## Failure to restrict URL access

Het kan soms voorkomen dat er bij een website alleen gekeken wordt of er is ingelogd en niet of de gebruiker de juiste rechten heeft.

Hiervoor ga ik Identity weer aan het werk zetten door verschillende rollen aan te maken en hierop te checken bij de verschillende pagina’s.

## Insufficient transport layer protection

Om hieraan te voldoen zal ik de gebruikers forceren om via https de website te bezoeken. http zal alleen beschikbaar zijn als deze website lokaal gedraait wordt, in debug mode.

## Unvalidated redirects and forwards

Om te beschermen op dit onderdeel zal ik ervoor zorgen dat er een check zit op redirects binnen de applicatie of deze wel lokaal zijn. Dit betekend of ze wel van mijn host komen en niet van een aanvallende partij.