RESEARCH REPORT

Wat zijn technieken om mijn webapplicatie te beveiligen?

Luka Spaninks

INHOUDSOPGAVE

1. INLEIDING

1.1 OMSCHRIJVING VAN DIT DOCUMENT

Dit document dient als onderzoeksverslag over welke veiligheid risico's veel voorkomen bij een webapplicatie en de bijhorende technieken om deze te verminderen/weg te halen.

1.2 LEESWIJZER

In dit document maak ik gebruik van de STARR-methode. Dit houdt in dat ik de volgorde: Situatie, Taak, Actie en Resultaat aanhoud. Ook maak ik gebruik van het DOT framework, dat betekent dat ik 3 strategieën gebruik om onderzoek te verrichten

2.1 DE HUIDIGE SITUATIE

Ik ben dit semester gestart met het bouwen en ontwerpen van een webapplicatie genaamd: Meet a Stranger. Mijn applicatie heeft als voornaamste doel om willekeurige gebruikers met elkaar te verbinden via video- en textchat. Deze applicatie maakt ook gebruik van opgeslagen data. Deze data mag niet zomaar voor iedereen beschikbaar zijn.

2.2 HET PROBLEEM VAN DE HUIDIGE SITUATIE

Het probleem is dan ook dat ik niet weet hoe ik mijn webapplicatie kan beveiligen tegen gebruikers met slechte bedoelingen.

2.3 DE ONDERZOEKSVRAAG DIE HIER BIJ HOORT

De onderzoeksvraag luidt: Wat zijn technieken om mijn webapplicatie te beveiligen?

3. HOOFD- EN DEELVRAGEN

In dit document zal ik mijn best doen om de volgende vragen te beantwoorden: **Wat zijn technieken om mijn webapplicatie te beveiligen?**

1. Wat zijn de OWASP-top 10?
2. Welke kwetsbaarheden zijn voor mijn use case mogelijk een risico?
3. Wat zijn mogelijke oplossingen voor deze risico's?
4. Wat zijn JSON Web tokens?
5. Hoe implementeer ik JSON Web tokens?

4. TAKEN

De volgende strategieën ga ik gebruiken om de hoofd- en deelvragen te beantwoorden:

Library: Best, good and bad practices

Bij de best, good and bad practices strategie wil ik gaan kijken naar de bevindingen van andere mensen. Ik wil weten wat je absoluut niet moet doen en wat juist wel. Ik denk dat deze strategie heel handig zal zijn voor mijn onderzoek omdat het bij security ook belangrijk is om te weten wat er fout kan gaan.

Field: Problem Analysis

Bij de problem analysis wil ik gaan uitzoeken wat nou het probleem is van bijvoorbeeld de veiligheid risico's. Ik ga er dus dieper op in en wil ook graag weten waarom iets onveilig is.

Workshop: Prototyping

Bij prototyping wil ik zelf een techniek toepassen die de beveiliging bevorderd van mijn applicatie.

WAT ZIJN DE OWASP-TOP 10

De OWASP-top 10 (Open Web Application Security Project) zijn de 10 meest voorkomende kwetsbaarheden in een webapplicatie. Deze worden regelmatig bijgehouden en geüpdatet door verschillende veiligheidsexperts over de hele wereld.

1. Injection

Injection is een aanval waarbij de hacker zijn eigen code injecteert in een programma. Deze code wordt dan uitgevoerd en kan ervoor zorgen dat er data wordt gestolen, een service wordt uitgeschakeld of zelfs het hele systeem overgenomen kan worden. Ze komen nog vaak voor in oudere systemen en zijn vaak het gevolg van een slechte user input validatie. De meest voorkomende injection attacks zijn Cross-site Scripting en SQL injection.

2. Broken Authentication

Bij broken authentication wordt er misbruik gemaakt van een slecht geïmplementeerd inlog systeem. Zo kan een hacker bijvoorbeeld een script gebruiken om heel veel gebruikersnamen en wachtwoorden te proberen zonder dat er een limiet op het aantal pogingen zit. Ook kan het zijn dat het systeem een gebruiker toelaat om een zwak wachtwoord te gebruiken. Dit kan ervoor zorgen dat de hacker heel makkelijk een wachtwoord raadt.

3. Sensitive Data Exposure

Sensitive Data Exposure is een probleem waarbij een hacker data tussen een gebruiker en een server kan onderscheppen. Dit probleem kan voorkomen bij publieke netwerken. Een hacker kan zelf een netwerk opzetten die precies lijkt op die van bijvoorbeeld McDonalds. Als de data tussen server en gebruiker niet wordt geëncrypt dan is dit voor die hacker gewoon zichtbaar.

4. XML External Entities

Dit is een aanval waarbij een hacker een XML parser commando’s kan geven om gevoelige data vrij te geven. Dit probleem kan voorkomen bij XML omdat het een functie heeft die lokale bestanden kan lezen. Ook kan het externe bestanden laden en post requests maken. Dit maakt een endpoint die XML accepteert al snel gevoelig voor dit soort aanvallen. Bij JSON is dit niet mogelijk omdat deze taal niet zulke functies heeft.

5. Broken Access Control

Bij Broken Access Control kan een hacker bij endpoints waarvoor hij niet bevoegd is. Het kan zijn dat het role systeem slecht is geïmplementeerd of dat er een kwetsbaarheid zit in een url met parameters.

6. Security Misconfiguration

Security Misconfiguration komt vaak voor en is niet meer dan slechte configuratie van een applicatie. Een voorbeeld is dat een ontwikkelaar een pagina voor gebruikers probeert te verbergen door hem een moeilijke naam te geven in plaats van security te implementeren. Het is vaak het resultaat van de default configuratie te gebruiken.

7. Cross-Site Scripting

Bij Cross-Site Scripting kan een hacker code plaatsen in een url of website wat dan vervolgens wordt uitgevoerd bij andere gebruikers. Deze code kan dan bijvoorbeeld data stelen van een slachtoffer en deze vervolgens doorsturen aan de hacker. Zo worden cookies gestolen waarmee de hacker zich vervolgens kan voordoen als de gebruiker.

8. Insecure Deserialization

Bij insecure Deserialization wordt er misbruik gemaakt van data die wordt gedeserialiseerd zonder enige validatie. Zo kan een hacker bijvoorbeeld de prijs van een product aanpassen zonder dat het systeem een controle doet.

9. Using Components With Known Vulnerabilities

Bij het gebruik van een library of framework kan er soms een kwetsbaarheid in zitten. Een hacker maakt hier dan misbruik van en kan zo heel veel systemen raken.

10. Insufficient Logging and Monitoring

Als een systeembeheerder zijn systeem niet goed genoeg in de gaten houdt kan dit ervoor zorgen dat een kwetsbaarheid pas heel laat wordt opgemerkt. Dit geeft hackers genoeg tijd om hier misbruik van te maken. Gemiddeld wordt een kwetsbaarheid pas na 200 dagen opgepakt.

WELKE KWETSBAARHEDEN ZIJN VOOR MIJN USE CASE MOGELIJK EEN RISICO

Om deze vraag te beantwoorden ga ik de OWASP-top 10 nog een keer bekijken en geef ik een beschrijving per aanval/kwetsbaarheid in hoeverre dit mijn applicatie treft.

1. Injection

Een applicatie is kwetsbaar voor SQL injection als de systeembeheerder dynamische queries toelaat. In mijn Rest api valideer ik eerst de gebruiker input en sta ik geen dynamische queries toe. Daarnaast maak ik ook gebruik van een ORM waardoor ik zelf niet eens queries hoef te schrijven. Dit maakt SQL injection bijna onmogelijk in mijn use case.

Kwetsbaarheid niveau: -

2. Broken Authentication

Broken Authentication is een probleem voor mijn applicatie. Het is mogelijk dat een hacker ongestoord een account kan bruteforcen.

Kwetsbaarheid niveau: hoog

3. Sensitive Data Exposure

Ook deze kan een probleem worden voor mijn applicatie als ik niet goed oplet.

Kwetsbaarheid niveau: hoog

4.XML external entities

Deze zal geen probleem worden in mijn applicatie, ik maak namelijk gebruik van JSON.

Kwetsbaarheid niveau: -

5. Broken Access Control

Dit is mogelijk een probleem voor mijn applicatie. Ik zal security moeten implementeren wil ik dit oplossen.

Kwetsbaarheid niveau: hoog

6. Security misconfiguration

Security misconfiguration is een valkuil waar ik ook in kan trappen. Ik zal dus alles goed moeten bijhouden en testen.

Kwetsbaarheid niveau: mogelijk hoog

7. Cross Site Scripting

Aangezien ik meerdere inputs gebruik in mijn website is het mogelijk om geraakt te worden door zo een aanval.

Kwetsbaarheid niveau: mogelijk hoog

8. Insecure Deserialization

Ik ga er niet vanuit dat ik mijn applicatie hiervoor gevoelig is. Zolang ik zoveel mogelijk data valideer (backend) kan dit niet zo snel mis gaan.

Kwetsbaarheid niveau: laag

9. Using Components With Known Vulnerabilities

Het is mogelijk dat ik hiervoor kwetsbaar ben, ik maak namelijk gebruik van javascript libraries en het spring framework.

Kwetsbaarheid niveau: hoog

10. Insufficient Logging and Monitoring

Ook dit is iets waar ik nog iets mee kan doen.

Kwetsbaarheid niveau: -

WAT ZIJN MOGELIJKE OPLOSSINGEN VOOR DEZE RISICO'S?

Eerder heb ik een aantal aanvallen genoemd die zonder security met gemak mijn systeem kunnen hacken. Om deze kwetsbaarheden op te lossen zijn er een aantal stappen die ik kan nemen.

Als eerste zorgt het gebruik maken van een security framework al voor een afname in kwetsbaarheden. Omdat ik het spring framework gebruik in mijn RestAPI is het wel zo handig om Spring security te gebruiken. Doormiddel van spring security is het mogelijk om bijvoorbeeld het aantal inlog pogingen te begrenzen. Dit helpt al bij het aanpakken van de Broken Authentication Kwetsbaarheid. Ook is een mogelijkheid multi-factor authentication en de eisen verhogen voor een wachtwoord.

https://www.codejava.net/frameworks/spring-boot/spring-security-limit-login-attempts-example

Sensitive Data Exposure is op te lossen door wachtwoorden te hashen en gebruik te maken van het https-protocol. Ook moet ik oppassen met wat ik beschikbaar maak voor de gebruikers qua data. Een tip is dat het belangrijk is dat ik belangrijke informatie niet in cookies of localstorage plaats. Deze kunnen namelijk gehackt worden.

Broken Access Control moet met spring security voorkomen kunnen worden, zolang ik de autorisatie goed configureer.

Cross Site Scripting kwetsbaarheden kunnen gevonden worden doormiddel van tools of handmatig testen. Als ik het handmatig wil testen moet ik dus een test element maken met javascript erin verwerkt. Deze moet ik dan plakken in een input en als deze niet wordt uitgevoerd zit er heel waarschijnlijk geen kwetsbaarheid. Hetzelfde is van toepassing voor URL-parameters, maar vooralsnog heb ik dit niet in mijn applicatie.

Om Using Components with Known Vulnerabilities op te lossen zal ik op tijd mijn libraries moeten updaten.

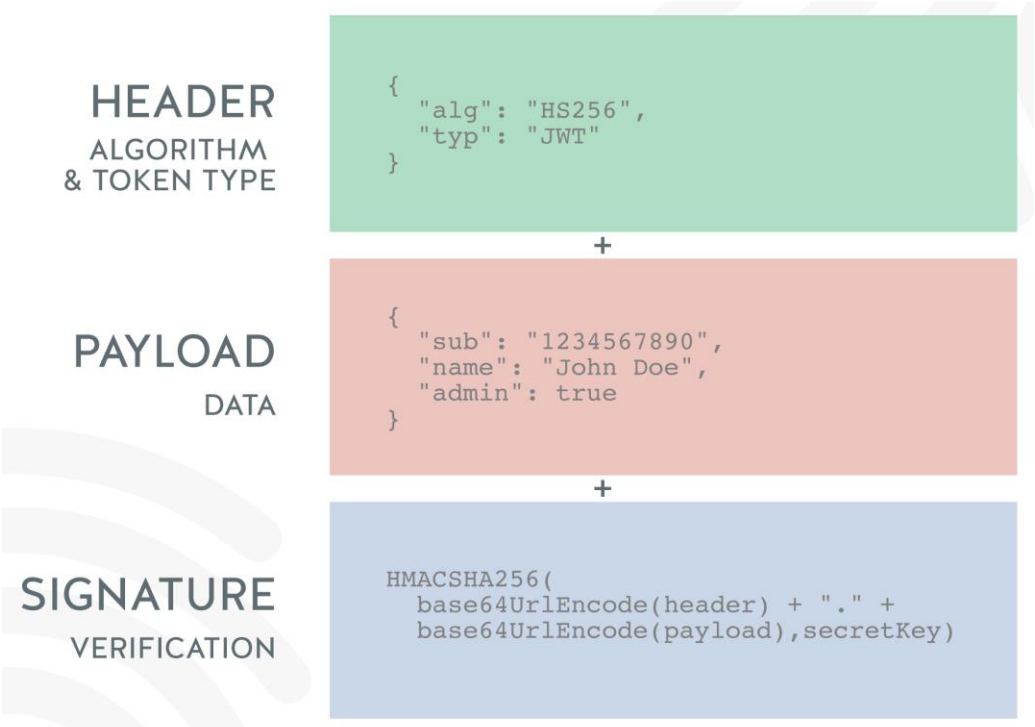
Insufficient Logging and monitoring, hiervoor kan ik in mijn backend een logger toevoegen. Deze houdt dan bij wat er allemaal binnenkomt en welke veranderingen er aan mijn database worden gemaakt.

Een goed overzicht staat op de website: https://owasp.org/www-project-top-ten/

WAT ZIJN JSON WEB TOKENS

Om mijn RestAPI te beveiligen tegen onbevoegde gebruikers moet ik een vorm van autheticatie implementeren. Een manier om dit te doen is doormiddel van tokens. Een token wordt naar een gebruiker gestuurd als hij/zij succesvol heeft ingelogd. Met deze token kan een gebruiker gegevens opvragen/versturen waarvoor hij/zij bevoegd is. Een makkelijkere oplossing zou zijn om de gebruikersnaam en het wachtwoord direct in de header mee te geven, maar dan zit je met het probleem dat dit heel onveilig is (sensitive data exposure)

In mijn applicatie wil ik gebruik maken van JSON Web Tokens. JSON Web Token is een standaard voor het maken van een token (string) die data bevat. Deze data bestaat uit informatie (payload) zoals de rol van de gebruiker. Ook heb je nog de header en de signature. De header bevat informatie over het type token, jwt in mijn geval, en het algoritme waarmee de token is geëncrypt of gesigned. Een signature zorgt ervoor dat de informatie niet zomaar veranderd kan worden. De server kijkt dan of de signature overeenkomt met de payload. Als dat niet zo is, dan is de jwt niet geldig. Een JSON Web Token kan gelezen worden door iedereen die hem bemachtigd.



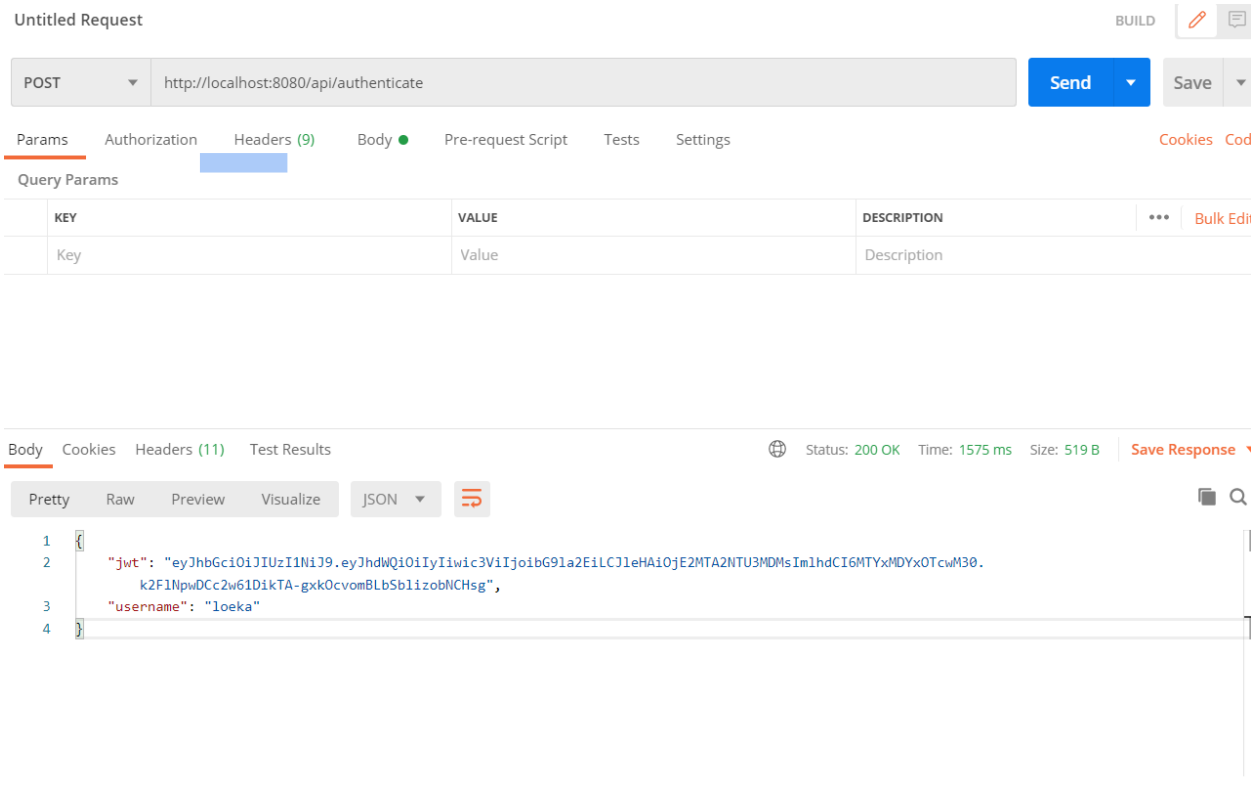
HOE IMPLEMENTEER IK JSON WEB TOKENS?

Zoals ik al eerder heb aangegeven maak ik gebruik van het spring framework. Daarom is het wel zo slim om spring security te gebruiken voor het implementeren van de webtokens. Ik heb gekozen om een YouTube tutorial te volgen van de gebruiker Java Brains. De video begint met de opzet van het project en het maken van een configuratie class. Deze class bepaalt eigenlijk welke endpoints toegankelijk zijn voor welke gebruikers. Ook voeg je toe dat je een json web token check wil doen voordat een controller endpoint wordt aangesproken end dat je geen gebruik wilt maken van een sessie (stateless).

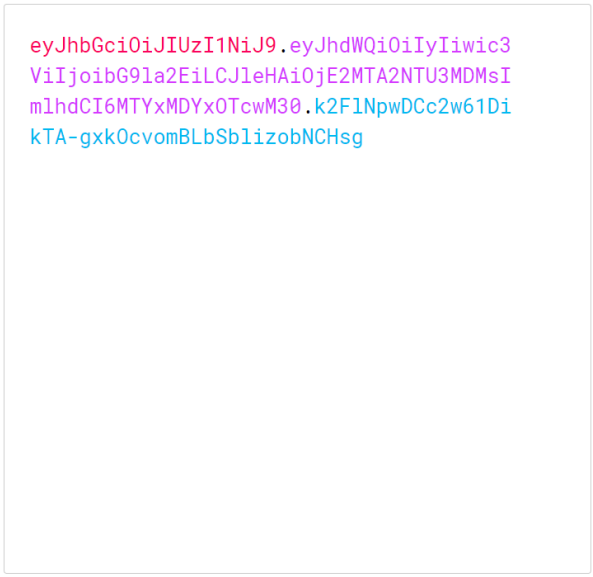
Dan wordt de UserDetailsService class aangemaakt. Deze haalt via een repository class de gebruiker op uit de database. Deze gebruikersclasse heet UserDetails en bevat informatie zoals het wachtwoord, de gebruikersnaam, de rol en informatie over de geldigheid en duur van de token.

Hierna wordt de JwtUtil class toegevoegd. Deze zorgt voor het maken en valideren van een token. Ook kun je met deze class informatie uit de token halen. Als laatste wordt er ook nog een JwtFilter class aangemaakt. De JwtFilter class is verantwoordelijk voor onderscheppen en het goedkeuren van een token voordat een endpoint wordt aangesproken. Het jwt gedeelte is klaar, er hoeft alleen nog maar een authenticatie endpoint aangemaakt te worden die bij een juiste login de token aan de gebruiker geeft.

In mijn application.properties kan ik ook nog een sleutel definiëren die alleen ik ken. Deze sleutel zorgt ervoor dat mijn signature uniek is. Zo kan een gebruiker de jwt niet zomaar aanpassen. Als de sleutel wordt geraden of gestolen kan een hacker bij alle data, dus is het belangrijk om hem enigzins complex te maken.



Authenticatie controller Postman



Parsed JWT

Exp = expiration time

Iat = issued at

6. RESULTAAT

Dankzij het verrichte onderzoek ben ik me veel bewuster van de veiligheidsrisico's die komen kijken bij software. Ik heb geleerd wat goede en slechte gewoontes zijn bij het ontwerpen van een veilig systeem. Zo is het bijvoorbeeld belangrijk om te zorgen dat je data niet zomaar direct aangepast kan worden door gebruikers. Een server die de gebruikersinput valideert is dus een heel belangrijk middel om dit te beveiligen. Ook heb ik geleerd dat je voorzichtig moet zijn met de data die je heen en weer stuurt. Bepaalde data zoals wachtwoorden en andere gevoelige gebruikersgegevens wil je nooit toegankelijk maken voor onbevoegden. Daarom is het handig om zo min mogelijk client-side mee te geven. Daarnaast is het hashen van wachtwoorden verstandig voor in het geval van een database lek.

Als laatste heb ik ook nog geleerd wat JSON Web Tokens zijn en hoe je deze implementeert in een spring boot Rest API. Hiermee is het gelukt om authenticatie en autorisatie te implementeren.

7. REFLECTIE

Ik ben blij met het resultaat. Ik vind dat ik veel geleerd heb en zelfs dingen ben tegengekomen die ik niet verwacht had. Een voorbeeld is XML external entities, ik had geen idee dat XML bepaalde functionaliteiten heeft die voor kwetsbaarheden kunnen zorgen.

Als ik het onderzoek opnieuw zou moeten doen zou ik het waarschijnlijk vergelijkbaar doen met hoe ik het nu heb gedaan. Misschien zou het bij een volgend onderzoek interessant zijn in de praktijk te experimenteren met bepaalde kwetsbaarheden zoals XSS of SQL injection.

8. BRONNEN

- *https://owasp.org/www-project-top-ten/*
- *https://www.acunetix.com/blog/articles/injection-attacks*
- *https://medium.com/digitalfrontiers/5-easy-steps-to-improve-your-spring-boot-apps-security-ca4ffb536fa6*
- *https://youtu.be/rWHvp7rUka8*
- *https://jwt.io/introduction/*
- *https://www.youtube.com/watch?v=X80nJ5T7YpE&t=1924s&ab_channel=JavaBrains*
- *https://www.youtube.com/watch?v=gjm6VHZa_8s&t=887s&ab_channel=PwnFunction*