26 september 2022



onderzoek: hashing/salting

Boorsma,Rob R.A.

v01

Inhoud

[Versiebeheer 3](#_Toc115340104)

[Inleiding 4](#_Toc115340105)

[1. Huidige situatie 5](#_Toc115340106)

[2. De gewenste situatie 5](#_Toc115340107)

[2.1 Doel 5](#_Toc115340108)

[2.2 Eisen 5](#_Toc115340109)

# Versiebeheer

|  |  |  |  |
| --- | --- | --- | --- |
| Persoon | Datum | Notitie | Versie |
| Rob Boorsma | 26-09-2022 | 1e Versie document | V01 |
| Rob Boorsma | 29-09-2022 | Toevoeging huidige Situatie&Doel | V02 |
| Rob Boorsma | 02-10-2022 | Informatie toegevoegd van onderzoek | V03 |
| xxxx | xxxx | xxx | Xxx |

# Inleiding

In dit document, word mijn onderzoek en mijn conclusies gedocumenteerd, betreffende het veilig opslaan van persoonsgegevens in een database. Denk bijvoorbeeld aan hashing & salting.

Dit onderzoek is gemaakt na aanleiding van de opdracht van het Ontdekstation.

Het Ontdekstation is bezig met een applicatie genaamd meetjestad.nl. Er worden door middel van verschillende type sensor data opgemeten en opgeslagen. Denk aan data zoals de luchtvochtigheid en de temperatuur. Het is de bedoeling dat alle burgers deze data eenvoudig kunnen inzien.

Burgers moeten dus ook kunnen inloggen in deze omgeving, maar dit moet natuurlijk op een veilige manier gebeuren. Hoe gaan deze gegevens veilig opgeslagen worden. En mocht er toch een inbraak zijn, hoe zorgen we voor dat er geen inbraak gemaakt kan worden in de accounts van deze burgers?

De hoofdvraag van het onderzoek is:

* Hoe kan je op een veilige manier persoonsgegevens van website gebruikers opslaan in een database.

Hierbij kunnen een aantal deelvragen bedacht worden:

* Is het mogelijk om een random salt te generate door middel van een algaritme?
* Welke hashing method is het beste voor wachtwoorden opslaan?
* Hoe blijft een user veilig ingelogd zonder dat het data verkeer uit te lezen is?

# 1. Huidige situatie

Momenteel is er in de front end een inlog pagina waarmee je je gegevens kan invullen. Er word echter nog niets gedaan met deze gegevens.

Er word gecontroleerd of er in het veld mailadres een @ gevonden is en of er na de @ nog tekst staat. Er word echter niet gevalideerd of het ook daadwerkelijk een mail is. Ook zijn er geen eisen gesteld aan het wachtwoord. Er kan dus eenvoudig een heel zwak wachtwoord gemaakt worden.

# 2. De gewenste situatie

## 2.1 Doel

Het doel van de opdracht is te zorgen dat alle klanten veilig gebruik kunnen maken van de Applicatie meetjestad.nl.

Dit houd in dat we een aantal stappen moeten nemen om te voorkomen dat deze accounts gemakkelijk gehackt kunnen worden. Hoe worden deze gegevens veilig opgeslagen zodat je de privacy van de gebruikers kan waarborgen? Meetjestad is een publieke applicatie die door iedereen gebruikt mag worden, veiligheid is daarom een top prioriteit.

Welke gegevens kunnen niet in de database als plain tekst staan? En hoe zorgen we er voor dat je veilig ingelogd kan blijven op de pagina, zonder dat deze gegevens uit te lezen zijn door een hacker?

## 2.2 Eisen

Hieruit kan geconcludeerd worden dat er een aantal eisen zijn om aan dit doel te voldoen

* Wachtwoorden moeten voldoen aan een bepaald beleid.
* Wachtwoorden moeten hashed/salted in de database opgeslagen worden
* Een gebruiker moet veilig ingelogd kunnen blijven op de website zonder dat de gegevens uit te lezen zijn.

# 3.0 Onderzoek

## 3.1 Vooraf

De applicatie word gebouwd in Java Spring Boot backend + JavaScript React Front end.

In dit onderzoek word dus gekeken naar welke oplossing het beste toegepast kan worden op deze situatie.

## 3.2 Veilig wachtwoord

Elke site heeft ze nodig, een wachtwoord om in te loggen. Wanneer een wachtwoord niet veilig is, kan deze zo uitgelekt worden. En kan dit account eenvoudig overgenomen worden. Om dit te voorkomen moet er een wachtwoord beleid zijn. Dit zijn eisen waar het wachtwoord aan moet voldoen, anders kan er geen account aangemaakt worden.

Een veilig wachtwoord bestaat uit een aantal eisen:

* Minimaal 12 karakters
* Hoofdletters
* Kleineletters
* Cijfers
* Speciale tekens

Ook mogen woorden uit je email niet voorkomen in het wachtwoord en kan je beter je eigen naam niet gebruikt. (Muhlenberg, 2022)

Dit staat ook vermeld in de documentatie van Microsoft.

## 3.3 Hashing/Salting

### 3.3.1 Waarom?

(Nieuw)

Hashing is belangrijk om een wachtwoord veilig op te slaan. Dit houd in dat een wachtwoord niet uit te lezen is wanneer een database gehacked wordt. Hashing is geen encryptie! Maar wel een manier van cryptografie.

Encryptie kan je decrypten, door middel van een public key. Een hash is altijd one way.

Zo kan je van het woord: ICT.  
De SHA256 Hash: bc1ccfebaf4130bc34322a99fb2e80b0fb593d55cb1c181f8e84958d6a52d73b krijgen.

Het is echter onmogelijk om vanuit deze hash, terug te gaan naar het originele woord. Bij het inloggen word het ingevulde wachtwoord dus opnieuw gehashed. En word er vervolgens gecontroleerd of deze overeenkomt met de bekende hash die in de database staat.

Hashing alleen is echter niet veilig genoeg. Een hacker kan elk woord proberen te hashen en vervolgens zijn eigen table maken. ( Ook wel bekend als Rainbow Table ).

Hierom is het belangrijk dat een hashing method, langzaam is. Een hacker maakt vaak gebruik van brute-force attacks. Ook wel het uitproberen van meerdere wachtwoorden, heel snel achter elkaar. ( Dit gebeurd door een automatisch systeem). Als je in een uur tijd 100 wachtwoorden kan uitproberen. Is dat natuurlijk veiliger dan dat je 10.000 wachtwoorden kan uitproberen.

Verder is er een andere methode genaamd salting. Salting houd in dat je een waarde toevoegd aan het wachtwoord. Dus bijvoorbeeld:

* Gebruiker vult ICT in
* Systeem Voegt het woord THUIS toe ( Meestal is een SALT 32 karakters lang )
* Systeem maakt een Hash voor het woord ICTTHUIS

Hierdoor komt er dus weer een totaal andere hash uit. Dit betekent ook dat als user1, en user2 het zelfde wachtwoord hebben. En hierbij een ander salt gebruikt wordt, dat het hash totaal anders is. Zo kan een hacker dus nooit weten dat 2 mensen de zelfde wachtwoorden gebruiken.

(Gitconnected.com, 2022)

### 3.3.2 MD5

MD5 is niet veilig voor wachtwoord hashing! Zie onderstaand voorbeeld:

Graphical user interface, text

Description automatically generated

(MD5Decrypt, 2022)

(Nieuw)

MD5 is een snel hashing algaritme. Zoals eerder omschreven moet een hash algaritme traag zijn om wachtwoorden veilig op te slaan. Daarbij zijn er veel wachtwoorden bekend zoals in bovenstaande image te zien is.

### 3.3.3 SHA512

(Nieuw)

Hoewel SHA512 een goeie hash genereert is deze ook redelijk snel. Hiernaast zijn er voor passwords nog veel sterkere mogelijkheden. SHA512 is ook designed om snel te werken.

(Baeldung, 2022)

### 3.3.4 Password Hash Functions

(Nieuw)

BCrypt en PBDKF zijn password hash functions. Deze zijn gespecialiseerd om traag te zijn. Dit is juist om zo genoemde brute force attacks te voorkomen. Deze functies zijn ook configureerbaar. Computers worden steeds sneller, en deze functies kunnen dus ook zo geconfigureerd worden om ze veilig te houden.

Deze functions zijn ook intensive voor de CPU, ze vragen veel meer om aan een hash te komen. Dit is ook de reden dat deze een stuk trager worden.

Text

Description automatically generated  
Dit zijn de recommendations volgens de OWASP principles. (Owasp Deteact, 2022)

### 3.3.5 Salting

( WORK IN PROGRESS )

(Salting volgens OWASP via SecureRandom in java)

<https://docs.oracle.com/en/java/javase/11/docs/api/java.base/java/security/SecureRandom.html>

https://auth0.com/blog/adding-salt-to-hashing-a-better-way-to-store-passwords/