Beveiliging en kwetsbaarheden op het web

Stef Oostveen

September 2018

1 Veilige Wachtwoorden

1.1 Inleiding

De meesten van jullie hebben al gehoord dat het belangrijk is om een veilig wachtwoord te kiezen voor aanmeldingen op websites of computers. Maar wanneer is een wachtwoord veilig? En hoe lang duurt het voor een hacker om je wachtwoord te raden? Het lijkt logisch dat een langer en ingewikkelder wachtwoord moeilijker te raden is. Een wachtwoord onthouden dat lijkt op `$rH81(fv*BGh`, laat staan meerdere daarvan is een onmogelijke opgave. We gaan eens kijken wat we daar aan kunnen doen.

1.2 In de praktijk

1.2.1 Wachtwoorden opslaan

Zoals jullie hebben gezien tijdens het maken van de PHP opdrachten is na het registreren van een klant op een website, het wachtwoord van die klant gewoon zichtbaar voor de beheerder! Voor de meeste websites, bijvoorbeeld die van Google, Apple of in Wordpress is dit echter niet het geval omdat de wachtwoorden daar versleuteld worden opgeslagen (zie het hoofdstuk 'hashing'), zoals hopelijk ook bij jouw website. Maar we weten natuurlijk niet van te voren hoe netjes de beheerder van een website te werk gaat. Misschien worden de wachtwoorden wel kant-en-klaar leesbaar in de database opgeslagen. In dat laatste geval hebben inbrekers het makkelijk als ze ooit toegang krijgen tot de database. Daarom is het belangrijk om voor elke site een ander wachtwoord te nemen; als een hacker dan van de ene website je wachtwoord achterhaalt, zijn je andere accounts toch nog veilig.

1.2.2 Wachtwoorden raden

Stel nou voor dat een beheerder van een website de wachtwoorden van zijn klanten veilig heeft opgeslagen. Een hacker heeft geen toegang tot de wachtwoorden, dus hij moet nu iets anders verzinnen. Hij zou natuurlijk gewoon kunnen proberen om het juiste wachtwoord van jouw account te raden. Dit principe noemen we *Brute forcing*.

Hoe lang duurt het om een gemiddeld wachtwoord te raden? Gelukkig (voor de hacker) hoeft hij dat niet zelf te doen. Een computer is heel goed in staat verschillende wachtwoorden uit te proberen. Als voorbeeld kunnen we kijken naar de pincode van je bankpas. Een pincode is 4 karakters lang en elk karakter kan 10 verschillende waardes aannemen (van 0 t/m 9). Dat betekent dat er $10^4 = 10000$ verschillende mogelijke combinaties zijn. Peanuts voor een computer: een normaal beestje kan al 3 miljoen mogelijkheden per seconden produceren. Dat betekent dat jouw laptop een pincode heeft geraden in $10^4/3000000 = 0.003s$.

Opdracht 1

> Laten we eens kijken naar het meest voorkomende wachtwoord. Helaas is dat 'password'...
>
> **a)** Hoeveel mogelijke combinaties heeft dit wachtwoord?
>
> **b)** Hoe lang duurt het om dit wachtwoord te raden?
>
> Zoals je kunt zien heeft een eenvoudige computer alle mogelijke combinaties binnen een acceptabele tijd afgewerkt.
>
> **c)** Op welke manieren kunnen we het een hacker moeilijker maken?

1.3 Oplossingen

We kunnen dus wel stellen dat het bedenken van een ingewikkeld wachtwoord belangrijk is. En dat wachtwoord moet ook nog eens verschillen van andere wachtwoorden. Dat maakt het onthouden lastig. Een *password manager* kan

ons daarbij helpen. De meeste password managers zijn beschikbaar als een plugin voor je browser en als app op je telefoon. - password manager - ander wachtwoord voor andere sites - twofactor auth

2 SQL Injection

2.1 Inleiding

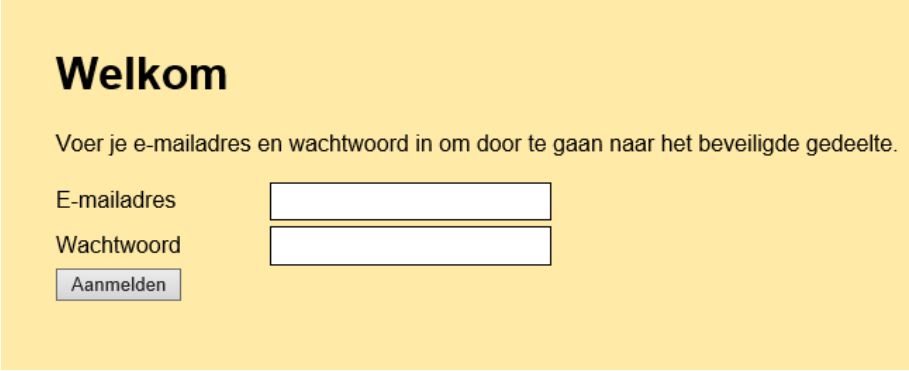
Tot nu toe heb je een aantal PHP-pagina's gemaakt en gezien waarop een bezoeker de mogelijkheid krijgt om in te loggen. Dit bestond vaak uit drie delen:

- Een HTML-formulier waarop een bezoeker zijn gebruikersnaam of e-mailadres en wachtwoord invult;
- Een verwerkpagina opgebouwd in PHP die controleert of de bezoeker de juiste gegevens heeft ingevuld.
- Een pagina met 'gevoelige' inhoud, die beschermd dient te worden.

Meestal staan de gebruikersnaam en het wachtwoord van een geregistreerde bezoeker opgeslagen in een *databank*. Wanneer de bezoeker zich probeert aan te melden wordt in de databank gezocht naar een combinatie van gebruikersnaam en wachtwoord.

Opdracht 2

> Bekijk nu het onderstaande codevoorbeeld 2.1. Figuur 1 is een screenshot van de webpagina. De verbinding met de database is al opgebouwd in `config.php`.
>
> **a)** Beredeneer aan de hand van de PHP-code wat er gebeurt als er een verkeerd wachtwoord of e-mailadres wordt ingevuld.
>
> **b)** Moet het e-mailadres uniek zijn in de databank? Zo ja/nee, waarom?
>
> **c)** Probeer in eigen woorden uit te leggen wat de aanhalingstekentjes (per aanhalingsteken!) in de `$sql`-variabele betekenen op regel 11.



Figuur 1: Aanmeldformulier

Listing 2.1: login.php

```php
<?php
        include("../config.php");
        session_start();

        if($_SERVER["REQUEST_METHOD"] == "POST") {

                $email = $_POST['email'];
                $wachtwoord = $_POST['wachtwoord'];

                // laad resultaten uit SQL
                $sql = "SELECT id FROM gebruikers WHERE email = '$email' and wachtwoord = '$wachtwoord'";
                $result = mysqli_query($db,$sql) or die(mysqli_error($db));
                $row = mysqli_fetch_array($result,MYSQLI_ASSOC);
                $count = mysqli_num_rows($result);

                if($count == 1) {
                        // stel de session in en stuur door naar de beveiligde pagina
                        $_SESSION['id'] = $row['id'];
                        header("location: index.php");
                        }else {
                        // geef een melding over verkeerde email/ww
                        $error = "Gebruikersnaam of wachtwoord is verkeerd";
                }
        }
?>
<html>
        <head>
                <title>Login</title>
                <link rel="stylesheet" type="text/css" href="../style.css">
        </head>

        <body>
                <h1>Welkom</h1>
                <p>Voer je e-mailadres en wachtwoord in om door te gaan naar het beveiligde gedeelte.</p>
                <form action="" method="post">
                        <label>E-mailadres</label><input class="textbox" type="text" name="email" /><br/>
                        <label>Wachtwoord</label><input class="textbox" type="password" name="wachtwoord" /><br/>
                        <input type="submit" value = "Aanmelden"/><br />
                </form>
                <?php
                        if(isset($error)){
                                echo "<div class='errorblock'>" . $error . "</div>";
                        }
                ?>
        </body>
</html>
```

2.2 In de praktijk

Op `http://informatica-cals.nl/soostveen/injection/` kunnen we een live voorbeeld bekijken van de webpagina. In de database staan twee gebruikers:

Naam	Gebruikersnaam	Wachtwoord
Margaret	`margaret@example.com`	`W8woord`
Winston	`winston@example.com`	???

Van Margaret, ons eigen account, zijn de gegevens bekend, maar van Winston weten we alleen het e-mailadres. We kunnen dus inloggen op het account van Margaret. Probeer maar.

Opdracht 3

We stappen nu in de schoenen van een inbreker en gaan proberen of we ook op het account van Winston kunnen inloggen, zonder dat we zijn wachtwoord weten of een aanpassing maken in de PHP-code. Daarvoor zijn een aantal stappen noodzakelijk, en we beginnen door te kijken naar wat we kunnen invoeren in de velden.

a) In principe kunnen we elk karakter invullen in het e-mailadres veld. Wat gebeurt er als je een enkel aanhalingsteken (') invult? Probeer met behulp van het codevoorbeeld te beredeneren hoe dit komt.

Blijkbaar is SQL niet erg vergevingsgezind als we gaan rommelen met de aanhalingstekens. Om het probleem te verduidelijken kijken we naar wat er gebeurt met de `$sql`-variabele als we braaf de juiste gegevens invullen:

```
1 SELECT id FROM gebruikers WHERE email = 'margaret@example.com' AND wachtwoord = 'W8woord'
```

Als we nu in plaats daarvan het enkele aanhalingsteken invullen komt er het volgende te staan:

```
1 SELECT id FROM gebruikers WHERE email = ''' AND wachtwoord = 'W8woord'
```

Het extra aanhalingsteken bij email zorgt ervoor dat SQL denkt dat de ingevoerde waarde na het tweede aanhalingsteken is opgehouden (een lege ingevoerde waarde weliswaar) en raakt ernstig verward als er daarna een derde aanhalingsteken wordt gevonden omdat er een volgende instructie wordt verwacht. Helaas zijn we er daarmee nog niet. We kunnen nu wel het einde van de ingevoerde waarde bepalen met het aanhalingsteken, maar SQL probeert de rest van de code nog gewoon uit te voeren en geeft als gevolg een error. Als we een manier vinden om de rest van de SQL-code voortijdig te beëindigen kunnen we proberen zoiets als dit te maken:

```
1 SELECT id FROM gebruikers WHERE email = 'winston@example.com'-- AND wachtwoord = 'W8woord'
```

Opdracht 4

We gaan weer even terug naar de PHP-code. In PHP kunnen we ingewikkelde stukjes code becommentariëren met behulp van # en // zoals ook gebeurd is in het codevoorbeeld.

a) Wat komt er op het scherm van een gebruiker te staan als het onderstaande stukje PHP-code wordt uitgevoerd? Hoe komt dit?

```
1 <?php
2         $a = 5;
3         $b = 8;
4         $c = 2;
5         echo $a + $b # + $c;
6 ?>
```

b) Zoek op welke mogelijkheden MySQL heeft om commentaar te plaatsen bij stukjes code.

Feitelijk hebben we dankzij de commentaar functie een methode gevonden om de rest van een SQL regel te negeren en hebben daarmee het missende puzzelstukje in handen.

Opdracht 5

> **a)** Kunnen we nu de SQL-code zo omvormen dat er een regel komt te staan die als emailadres `winston@example.com` gebruikt, de invoerwaarde netjes afsluit, en de rest van de regel negeert?

2.2.1 Toolbox

We nemen de ingrediënten voor onze SQL-injection even door:

'	Opent en sluit de waarde voor een variabele
#	Start het commentaar en beëindig de uitvoerbare SQL code.
--	Gelijk aan #. **Let op:** SQL verwacht na dit teken altijd een spatie.

Met behulp van deze tekens kunnen we onze eigen code of waardes in een invulveld plaatsen en door SQL laten uitvoeren. Laten we eens kijken of we met deze tools op zak een regel kunnen maken die ons naar binnen laat als Winston. We beginnen eerst met het invullen van het e-mailadres. Vervolgens sluiten we de waarde van de variabele `email` af met een aanhalingsteken. De rest van de SQL code kunnen we negeren, dus plaatsen we `--` gevolgd door een enkele spatie:

```
1 winston@example.com'␣--␣
```

PHP maakt van de ingevulde waarde vervolgens de volgende regel

```
1 SELECT id FROM gebruikers WHERE email = 'winston@example.com' -- ' AND wachtwoord = '
    W8woord'
```

Opdracht 6

> **a)** Als alles goed verlopen is, zouden we nu moeten kunnen inloggen als Winston. Probeer maar uit!
>
> **b)** Speelt dit probleem ook op je eigen webpagina?
>
> **c)** Bedenk een manier waarbij we kunnen inloggen zonder dat we de inlognaam weten

2.3 Theorie

SQL-Injectie ontstaat doordat de invoer van gebruikers niet (goed) wordt gecontroleerd voordat die aan de database wordt doorgestuurd. Aanhalingstekens in SQL, de boosdoeners in dit scenario, worden gebruikt om de waardes van *Strings* te openen en af te sluiten. Een een kwaadwillende gebruiker kan zelf ook aanhalingstekens invoegen en zo zelf bepalen wanneer de waarde wordt gesloten. Dat is natuurlijk niet de bedoeling.

2.4 Oplossing

De remedie tegen dit probleem is door de variabelen die we willen meegeven aan SQL (in ons geval email en wachtwoord) eerst voor te bereiden en te controleren op aanhalingstekens. Als er aanhalingstekens worden gevonden, wordt er een backslash voorgezet. Dit noemen we *escapen*. Gelukkig heeft PHP dit al voor ons ingebakken:

```
1 // escape de waarden ontvangen via POST
2 $email = mysqli_real_escape_string($db, $_POST['email']);
3 $wachtwoord = mysqli_real_escape_string($db, $_POST['wachtwoord']);
```

Nu is dit formulier niet meer vatbaar voor SQL-Injectie. Er zijn ook andere methoden om te voorkomen dat sql-injection plaatsvindt, zoals het gebruik van prepared statements. De ondersteuning van verschillende methoden is echter afhankelijk van de PHP-versie die we gebruiken. Let wel, sql-injection is niet alleen van toepassing op PHP. Ook andere programmeertalen die met databases werken zijn vatbaar!

Opdracht 7

> **a)** Beveilig je eigen webwinkel tegen sql-injection.

3 Hashing

3.1 Inleiding

Met SQL-Injectie is het niet alleen mogelijk om direct in te loggen zonder het wachtwoord te weten van een persoon, maar vaak is het ook mogelijk om een complete database in handen te krijgen. Dat is echter niet de enige methode om bij de database te komen; soms wordt er een zwakheid gevonden op andere plekken op de server en weet een hacker zo bij de database te komen. Wat de methode ook is, als een aanvaller bij de database komt, heeft hij in de huidige situatie toegang tot de gebruikersnamen en de wachtwoorden. Dat is natuurlijk niet de bedoeling.

3.2 Oplossing

We moeten dus op zoek gaan naar een manier om de wachtwoorden onleesbaar te maken, maar we willen ze ook kunnen controleren om iemand in te kunnen loggen. Hiervoor hebben we *hashing* tot onze beschikking.

Bij het hashen van een wachtwoord wordt dit wachtwoord met behulp van wat ingewikkelde wiskunde omgezet in een lange regel tekst (zie figuur 2). Deze regel kan niet meer terug worden gezet naar het wachtwoord. Hashen is dus eenrichtingsverkeer. Het hashen van het wachtwoord levert echter wel elke keer dezelfde regel tekst op, en dus kunnen we twee hashes met elkaar vergelijken. Er zijn meerdere hashing-algoritmen, bijvoorbeeld SHA-256 uit het voorbeeld.

Figuur 2: Hashing

3.2.1 Rainbow Tables

Probleem opgelost? Niet helemaal. Hoewel het bijna niet te doen is om een hash terug te krijgen naar een normaal wachtwoord, hebben hackers tabellen gemaakt met daarin heel veel bekende wachtwoorden en bijbehorende hashes. Zo'n tabel heet een *rainbow table*.

Opdracht 8

> Op `https://crackstation.net/` is zo'n rainbow table te doorzoeken
>
> **a)** Kan je het wachtwoord achterhalen dat hoort bij de hash
> `ADDB0F5E7826C857D7376D1BD9BC33C0C544790A2EAC96144A8AF22B1298C940` ?

3.2.2 Salting

4 Versleuteling

5 WiFi

6 Malware