**Week 4 - Web Application Development**

Lasse Benninga & Andre Nanninga

**Opgave 1: Beveiliging theorie**

**1. Bestudeer de owasp password storage cheat sheet. Hierin worden vier tips gegevens voor de opslag van wachtwoorden. Beschrijf deze vier en geef van elk een voor- en nadeel.**

1) Limiteer niet de toegestaande karakters voor een wachtwoord en zet een lange maximume lengte voor wachtwoorden.

De beste wachtwoorden zijn lang en bevatten karakters uit een zo groot mogelijke karakterset. Wanneer de karakterset en lengte gelimiteerd wordt is het bruteforcen van wachtwoorden makkelijker.

Extreem lange wachtwoorden kunnen echter leiden tot een DOS (Denial Of Service). Wanneer een extreem lang wachtwoord gehasht wordt kan dit erg veel CPU vragen waardoor andere requests mogelijk niet CPU tijd krijgen de ze nodig hebben.

2) Gebruik een gebruiker-unieke *salt* bij het hashen van wachtwoorden.

Door een gebruiker-unieke *salt* (een set van random bytes) te gebruiken bij het hashen wordt het gebruik van *rainbow tables* omzeilt. Een *rainbow table* is een dataset met veel voorkomende wachtworden en de daarbij horende hashes. Door een unieke *salt* toe te voegen zijn *rainbow tables* waardeloos omdat er voor elk wachtwoord een nieuwe table gemaakt zou moeten worden, dan wordt het dus een bruteforce attack welke veel meer tijd kost.

Een klein nadeel is dat het opslaan van deze gebruiker-unieke *salt* wel extra dataopslag vraagt.

3) Gebruik een cryptografische hashfunctie met een work factor.

Een cryptografische hashfunctie met een work factor is een hash functie waarbij met een work factor bepaald kan worden hoe lang de computatie duurt. Deze kan bijvoorbeeld ingesteld worden dat het bij het inloggen het hash 1 seconden duurt. Dit is een kort genoeg dat de gebruiker niet z'n geduld verliest maar te lang voor een aanvaller om te bruteforcen. Een bruteforce aanval waarbij er bij elk wachtwoord een vertraging van 1 seconden voorkomt duurt veel te lang om succesvol te zijn.

Een nadeel is dat work-factors niet overal onderstuend worden en ook niet elke soort applicatie passen. Voor sommige applicaties is een vertraging van 1 seconden bij het verificeren erg ongewenst.

4) Ontwerp je authenticatie systeem voor eventuele inbraken.

Zorg er voor dat bestaande authenticatie ongeldig wordt, forceer dat alle gebruikers voor alle mogelijke ingangen zich opnieuw moeten authenticeren.

Gebruik een nieuwe (sterke) authenticatie schema. Zorg dat het nieuwe schema niet het oude overschrijft met er naast staat.

Wanneer een gebruik weer inlogt forceer deze dan een nieuw wachtwoord in te voeren en sla deze op met het nieuwe schema. Verwijder dan het oude wachtwoord.

**2. Geef argumenten waarom een dergelijke beveiliging niet altijd noodzakelijk is.**

De sterkte van beveiliging voor een applicatie is afhankelijk van de hoeveelheid en de waarde van de persoonsgegeven die opgeslagen worden. Voor een website die enkel een lijst met films die jij leukt vindt bij houdt en daar informatie over geeft is het minder belangrijk dat deze de sterkste beveiliging heeft (hoewel het altijd een waardevolle toevoeging is). Voor websites zoals Google of Facebook is het juist heel erg belangrijk omdat zij beschikking hebben over heel veel waardevolle persoonsgegevens.

**3. Wat is een SQL injection en op welke manier kan dat gebruikt worden om een database te kraken?**

Een SQL injectie is het aanpassen van een SQL query door bepaalde data mee te sturen naar met een request. Met deze data wordt de query aangepast en kan een aanvallen (mogelijk) alles doen met een database wat hij wil.

Een voorbeeld van een SQL injectie zou zijn:

SELECT id FROM users WHERE username="$username" AND password="$password";

Wanneer een normale gebruiker inlogt met de gebruikersnaam 'Piet' en het wachtwoord 'welkom' wordt de bovenstaande query:

SELECT id FROM users WHERE username="piet" AND password="welkom";

Een aanvaller zou echter kunnen 'inloggen' met de gebruikersnaam 'Piet' en het wachtwoord 'xx" OR 1=1--'. Hiermee wordt de query:

SELECT id FROM users WHERE username="piet" AND password="xx" OR 1=1--";

De query is hiermee genoeg aangepast dat het wachtwoord niet meer juist hoeft te zijn om op een account in te loggen.

**4. Op welke manier kan query parameterization gebruikt worden om sql-injectie tegen te gaan? Geef aan op welke manier dit in het toegewezen framework toegepast wordt.**

Query parameterization is scheiden van de query en de data. De bovenstaande wordt:

SELECT id FROM users WHERE username=? AND password=?;

De vraagtekens geven aan dat hier data komt welke *niet* onderdeel zijn van de werking van de query.

Voor ons framework (python, flask) defineren we eerst onze *stored procedure* in de MySQL database. Voor de bovenstaande query ziet dat er als volgt uit:

PROCEDURE `sp\_login` (  
 IN p\_username VARCHAR(20),   
 IN p\_password VARCAR(200)  
)  
BEGIN  
 SELECT id FROM users WHERE username = p\_username AND password = p\_password;  
END$$

Vervolgens roepen we de *stored procedure* aan met de onderstaande python code:

conn = mysql.connect()  
cursor = conn.cursor()  
cursor.callproc('sp\_login', (username, password))  
result = cursor.fetchone()

**5. Wat is session-hijacking en op welke manier kan dit gebruikt worden voor identity-theft?**

Sessies worden over het algemeen bij gehouden met behulp van cookies. Wanneer een gebruiker inlogt krijgt hij een sessie cookie die bij alle volgende requests worden meegestuurd. Zo blijft de gebruiker ingelogd en hoeft er niet voor elke request geauthentiseerd worden.

Wanneer een kwaadwillend persoon deze cookie kan kopieren en in zijn eigen computer zet kan hij op de sessie van de eerste gebruiker de website bezoeken. Hij heeft dan toegang tot het account van de gebruiker en kan zo zijn identiteit stelen.

**6. Bekijk het onderstaande code-fragment. Welk gevaar schuilt er in dit fragement en op welke manieren is dat te omzeilen?**

...

**7. Bestudeer het artikel Seven steps for building a secure web-application. Geef van de zeven stappen die hierin genoemd worden elk een voor- en een nadeel. Is het wenselijk om deze stappen in elke web-applicatie in te bouwen?**

Query Parameterisation :

Voordeel: Veiliger omdat er geen kwaadwillende code direct geinjecteerd kan worden in de Query.  
Nadeel: Slechtere performance.

Secure password storage :

Voordeel: Wachtwoorden kunnen minder snel worden gestolen.  
Nadeel: Langzaam bij het encoden van het wachtwoord.