Melany Winklaar, Jordy Samson, Jeroen Hoebert & Aaik Oosters

UCloud

Hoe veilig ben jij

Inhoudsopgave

[Inleiding 2](#_Toc532471736)

[A1:2017-Injection 2](#_Toc532471737)

[A2:2017-Broken Authentication 3](#_Toc532471738)

[A3:2017-Sensitive Data Exposure 4](#_Toc532471739)

[A4:2017-XML External Entities (XXE) 5](#_Toc532471740)

[A5:2017-Broken Access Control [Merged] 6](#_Toc532471741)

[A6:2017-Security Misconfiguration 7](#_Toc532471742)

[A7:2017-Cross-Site Scripting (XSS) 8](#_Toc532471743)

[A8:2017-Insecure Deserialization 9](#_Toc532471744)

[A9:2017-Using Components with Known Vulnerabilities 10](#_Toc532471745)

[A10:2017-Insufficient Logging & Monitoring 11](#_Toc532471746)

# Inleiding

Voor het vak security hebben wij de opdracht gekregen om een OWASP top-10 webapplicatie te bouwen. Wij hebben hier ons best voor gedaan en zijn gekomen tot vele verschillende toepassingen om dit project OWASP top-10 valide te krijgen.

Het project is terug te vinden onder de volgende link:

<https://github.com/aaikoosters/blok12-security>

Wij hebben een UCloud gemaakt. Hierin is het mogelijk dat er foto’s ge-upload worden. Deze foto’s zijn voor jou en voor jou alleen. Aan ons de kunst om dit af te handelen en op te lossen.

# A1:2017-Injection

* Gebruik SQL-commando’s om SQL-injection te voorkomen
* Valideer binnenkomend data

# A2:2017-Broken Authenticatie

* Geen zwakke wachtwoorden toestaan (nog implementeren in devolop)
* ~~Data encrypten voordat verstuurd wordt~~ (toegepast)
* Maximaal aantal login pogingen (nog implementeren in devolop)

# A3:2017-Sensitive Data Exposure

* ~~Geen standard encryptie toepassen. Gebruikt up to date encryptie sleutels en algoritmes~~ (toegepast)
* We hashen het password
* **We gebruiken standaard php**

# A4:2017-XML External Entities (XXE)

Voor dit onderdeel zijn we op het oog valide. We hebben geen XML entities draaien waardoor dit voor nu veilig is. Maar op het moment dat wij wel XML-bestanden hadden gehad dan hadden we het op de volgende manier proberen af te vangen.

Dit kan gecontroleerd worden door verschillende tools. Die kunnen controleren op de boomstructuren binnen de XML lagen. Worden tags naar verwachting ook weer gesloten.

Daarnaast wordt er vanuit PHP een standaard meegeleverd die checkt of dit valide is. Dit is de XMLReader. Deze standaard is waar bekend veilig te gebruiken. We zouden het daarom op die manier toegepast hebben.

Daarnaast wil je XML tijdig valideren omdat het problemen op je websites kan tonen die afgevangen hadden kunnen worden.

# A5:2017-Broken Access Control [Merged]

* Zorg da teen specifiek iemand gegevens kunnen aanpassen aan de database Zorg dat alle toegangscontrole gemonitord en gelogd worden
* Invalideer JWT-tokens wanneer gebruikers uitloggen (toegepast?)

# A6:2017-Security Misconfiguration

Een van de eisen is geen HTTP te gebruiken. Wij wilden dit gaan verhelpen door SSL-certificaten te installeren op onze server. Echter door het gebrek aan een ‘live’ omgeving hebben wij dit proces overgeslagen. Het voegt in deze situatie niks toe. Het voegt niks toe omdat jij als gebruiker en enige bent die toegang heeft tot jouw omgeving.

Maar wat als we wel online waren gegaan? Dan hadden we dit uiteraard geïmplementeerd. Echter is het ding van SSL-certificaten zelf signed zijn. Dit zorgt voor schijnveiligheid. Het zorgt voor vertrouwen bij de client. Echter is SSL meer het idee van een hek op het land. Waar ja langs kan rijden zonder extra moeite

**Jordi scanners**

# A7:2017-Cross-Site Scripting (XSS)

* Een framework die beschermt tegen XSS
* Input valideren door middel van antiforgerytokens
* Niet vertrouwde http requests niet afhandelen

Voor het controleren van de Cross Site Scripting hebben wij in dit project een image uploaden. Dit gebeurt alleen voor jou. Dit is op het eerste oog geen schadelijk onderdeel voor onze applicatie. Voor de image hebben wij dit onderdeel proberen af te handelen voor de eerste stappen met:

Stap 1: Checken op de extensie. Het is alleen mogelijk om bestanden met de extensie JPEG en PNG te uploaden.

Stap 2: Aan het bestand vragen wat voor type het bestand is. Dit mag alleen JPEG en PNG zijn

Stap 3: Daarna is er een PHP-methode die het formaat van de foto opvraagt om te controleren of dit wel een ‘bestaand’ bestand is.

Stap 4: Waarna de bovenstaande stappen geslaagd zijn wordt de afbeelding opnieuw gebouwd zodat alle ‘slechte’ code er uitgefilterd wordt en alleen het plaatje ge-upload zal worden.

Daarnaast hebben we de input geprobeerd te valideren door gebruik te maken van antiforgerytokens **JEROEN SCHRIJFT DIT VERDER UIT**

# A8:2017-Insecure Deserialization

* Logging toevoegen zodat je kan traceren wat er is gebeurt
* Gaat jeroen doen en uitleggen

# A9:2017-Using Components with Known Vulnerabilities

* ~~Gebruik alleen officiële componenten~~ (toegepast)
* ~~Geüpdatet versie van componenten~~ (toegepast)

# A10:2017-Insufficient Logging & Monitoring

* Zorg dat logs gegenereerd worden zodat deze logs door een gecentraliseerd logbeheersysteem gebruikt kan worden.
* Het systeem zo monitoren dat er een alarm af gaat wanneer verdacht gedrag wordt gedecreteerd
* Zorg dat er een plan van herstel beschikbaar is indien iets fout is