Opdracht 3 bevat twee node applicaties. Voordat deze uitgevoerd kunnen worden, moet het commando “npm install” uitgevoerd worden in de “assignment3” folder van het project.

In de “server” folder is de Tweeter applicatie te vinden. Deze applicatie kan uitgevoerd worden door in de “server” folder het commando “node index.js” uit te voeren. Vervolgens is de applicatie beschikbaar in de browser op “http://localhost:3000”. Om berichten te kunnen versturen kan ingelogd worden met de username “user1” met wachtwoord “password1”, of username “user2” en wachtwoord “password2”.

In de “attacker” folder is een server te vinden die verantwoordelijk is voor het serveren van schadelijke code en het ontvangen van gestolen gebruikersgegevens. De applicatie kan uitgevoerd worden door in de “attacker” folder het commando “node attacker.js” uit te voeren.

# Risicoanalyse

|  |  |  |
| --- | --- | --- |
| **Bedreiging** | **Locatie** | **Toelichting** |
| Cross Site Scripting | Browser Client 🡨 🡪 Web Server | De webserver zuivert de input van data niet, waardoor schadelijke input (code) toegelaten wordt door de webserver. |
| Persistent Cross Site Scripting | Web Server 🡨 🡪 Memory | Aangezien de webserver de input die persistent wordt opgeslagen niet zuivert, kan schadelijke code persistent opgeslagen worden. Deze kan vervolgens teruggestuurd worden naar meerdere gebruikers. |

# Penetration Test

Zodra een gebruiker is ingelogd, kan hij/zij een bericht verzenden naar de webserver. De gebruiker heeft hier de mogelijkheid om JavaScript te injecteren. Dit kan echter niet rechtstreeks door een <script> tag te injecteren met schadelijke code. Scripts in de DOM worden namelijk uitgevoerd zodra de pagina wordt verwerkt. Zodra de pagina verwerkt is wordt deze geserveerd aan de gebruiker. Een <scripts> tag die na het verwerken nog geïnjecteerd wordt, zal niet worden uitgevoerd (Arora, 2014). Een attack die wel werkt is het “onError” attribuut van een <img> tag. De waardes van zogenaamde event handlers, zoals “onError” worden nog wel uitgevoerd zodra de bijbehorende events getriggerd worden (Arora, 2014). Wanneer een image niet gevonden kan worden (de waarde in het “src” attribuut is niet valide), wordt de waarde van het “onError” attribuut uitgevoerd. De waarde van het “onError” attribuut kan een script zijn (OWASP, XSS Filter Evasion Cheat Sheet, 2018). Een gebruiker kan de volgende code als message posten om een persistent XSS attack uit te voeren.



De hacker post de bovenstaande code als message naar de server. Deze message wordt vervolgens door de server opgeslagen. Iedere andere gebruiker die de berichten van de server opvraagt, krijgt vervolgens dit bericht. De bovenstaande code, het bericht, wordt dus weergegeven en getoond aan de gebruiker. De browser probeert een image in te laden, maar het pad naar de image is niet valide, waardoor de browser het schadelijke script in het “onError” attribuut uitvoert. Het schadelijke script in de “onError” voegt een nieuw HTML <script> element toe aan de body van de DOM. Het “src” attribuut van het <script> element verwijst naar het schadelijke JavaScript bestand dat op de server draait van de hacker. Die code wordt vervolgens uitgevoerd zodra het “onError” event wordt uitgevoerd, wat immers altijd gebeurd aangezien de “src” verwijst naar een niet bestaand image.

Het script credentials.js van de hacker overwrite de signIn functie. Wanneer dit schadelijke script is uitgevoerd, zal de signIn functie (die uitgevoerd wordt zodra een gebruiker inlogt) ook de inloggegevens en cookie van de gebruiker versturen naar de server van de hacker. Deze gegevens worden vervolgens in de console gelogd. De hacker weet vervolgens de inloggegevens van een gebruiker, en kan hier misbruik van maken.

# Mitigation

Het probleem is dat de message die gebruiker posten, worden ingeladen in de DOM via de volgende code:



Het probleem is de functie “innerHTML”. Deze functie ondersteunt, zoals de naam al zegt, het toevoegen van HTML. Indien de “message.message” dus bestaat uit HTML, ziet de website dit ook als HTML. Deze HTML wordt dan dus ook uitgevoerd. Indien “message.message” dus de schadelijke code bevat uit het vorige hoofdstuk (wat HTML bevat), dan wordt die ook uitgevoerd. We spreken hier van XSS. De vorm van XSS is persistent, aangezien de schadelijke code persistent wordt opgeslagen door de webserver. Iedere gebruiker die de chat opent, en de berichten ontvangt, krijgt ook de opgeslagen schadelijke code te zien. En bij iedere gebruiker wordt deze schadelijke code ook uitgevoerd.

Om dit te voorkomen kan “innerHTML” vervangen worden door de functie “textContent” of “innerText”. Deze twee methodes ondersteunen namelijk geen HTML, maar enkel tekst. Indien “message.message” in dit geval de schadelijke code bevat, ziet de browser dit als tekst. De HTML wordt dus niet uitgevoerd en de schadelijke JavaScript kan dus ook niet geïnjecteerd worden (OWASP, DOM Based XSS Prevention Cheat Sheet, 2017).

Daarnaast kan input bij de server als aan de browser kant ook verder gezuiverd worden door de betreffende input te escapen naar strings, overal waar data wordt opgehaald uit de HTML, en waar data in de HTML wordt gezet. Bovenstaande mitigation is hier een vorm van (OWASP, DOM Based XSS Prevention Cheat Sheet, 2017).

# https://i.gyazo.com/8a6b9ee2bda0dd62b01564caa4eb72a4.pnghttps://i.gyazo.com/067f274f3b7ab12d92d03074a12f90c3.pnghttps://i.gyazo.com/cddb4d0941d93cc535075bb83a54c9d5.pngCode Review

Hierboven de relevante onderdelen van de OWASP Code Review checklist getest. Het onderdeel dat vooral betrekking heeft op onze gebruikte mitigation is het vijfde onderdeel, “Input Validation”. Untrusted data wordt doormiddel van de mitigation gevalideerd op het type en formaat. Zo kan er geen JavaScript geïnjecteerd worden, om XSS te voorkomen. OWASP geeft echter aan dat er ook gevalideerd moet worden op de lengte van data. Zo voorkom je een buffer overflow. Een buffer overflow treedt op wanneer een programma meer gegevens naar een vast geheugen blok probeert te schrijven, dan de hoeveelheid gegevens het geheugen blok kan bevatten. Dit kan gebruikt worden voor XSS, hier hebben wij echter een mitigation voor toegevoegd. Het kan echter ook gebruikt worden voor een Denial of Service attack. Een buffer overflow kan een programma namelijk laten crashen. Een mogelijke mitigation is inderdaad de lengte van input data limiteren (TechTarget, n.d.).

Enkele aanbevelingen is ten eerste het toepassen van input validation op alle mogelijke plekken. Overal waar data in de HTML wordt gezet, en eruit wordt gehaald. Daarnaast ook op de server waar de data wordt weggeschreven naar een persistente opslag (laatste onderdeel review). Beveiligde verbindingen (HTTPS) en encryptie van cookie en inloggegevens zijn vanzelfsprekend aanbevelingen die voor iedere applicatie van groot belang zijn.

# Penetration Test 2

De applicatie is nu veilig tegen een persistent XSS-attack. Er zijn echter nog vele andere bedreigingen waar de applicatie niet tegen beveiligd is. Persistent XSS blijft ook altijd een bedreiging voor het systeem. Een systeem is nooit helemaal veilig. Er kunnen bedreigingen zitten in externe packages, of er kunnen bedreigingen zijn die je over het hoofd ziet. Ook kunnen er in de toekomst nog bedreigingen ontstaan, die nu nog niet bekend zijn, voor zowel de white als black hat hackers.

# References

Arora, R. (2014, juni 12). *Why are scripts injected through innerHTML not executed whilst onerror and other on<event> attributes on elements are? - Google XSS Challenge 2*. Opgehaald van Security StackExchange: https://security.stackexchange.com/questions/60861/why-are-scripts-injected-through-innerhtml-not-executed-whilst-onerror-and-other

OWASP. (2017, november 25). *DOM Based XSS Prevention Cheat Sheet*. Opgehaald van OWASP: https://www.owasp.org/index.php/DOM\_based\_XSS\_Prevention\_Cheat\_Sheet

OWASP. (2018, April 7). *XSS Filter Evasion Cheat Sheet*. Opgehaald van OWASP: https://www.owasp.org/index.php/XSS\_Filter\_Evasion\_Cheat\_Sheet

TechTarget. (sd). *Application attacks Definitions*. Opgehaald van TechTarget: https://searchsecurity.techtarget.com/definitions/Application-Attacks-Buffer-Overflows-Cross-Site-Scripting