justin van de laar

Abstract

In dit document ga Ik de volgende vraag onderzoeken. Welke type of security risk is het meest gevaarlijk voor mijn applicatie?.

Security Research document

Inhoud

[Inleiding 2](#_Toc74991910)

[Woord verwijzing lijst 2](#_Toc74991911)

[Voorwoord 2](#_Toc74991912)

[Leeswijzer 2](#_Toc74991913)

[Welke security risk is het meest gevaarlijke voor de applicatie? 3](#_Toc74991914)

[Wat voor security risks zijn er? 3](#_Toc74991915)

[SQL injections 3](#_Toc74991916)

[XSS Cross side scripting 3](#_Toc74991917)

[Cross site request forgery (CRSF) 4](#_Toc74991918)

[Broken Authencation 4](#_Toc74991919)

[Why is security important for the application 5](#_Toc74991920)

[How do we secure the application better? 6](#_Toc74991921)

[Libarary research 6](#_Toc74991922)

[Field research 6](#_Toc74991923)

[Lab research 6](#_Toc74991924)

[Workshop research 6](#_Toc74991925)

[Showroom Research 7](#_Toc74991926)

[Conclusie 7](#_Toc74991927)

[Bronnen 8](#_Toc74991928)

[Verwijzingen 8](#_Toc74991929)

[Images Bronnen 8](#_Toc74991930)

# Inleiding

## Woord verwijzing lijst

CRSF = Cross site request forgery  
XSS = Cross side scripting.  
SQLIJ = SQL injections   
BA = Broken Authencation

## Voorwoord

Om te beginnen zal ik een korte introductie gegeven naar het document. Het document is bedoeld een inzicht te geven welke security risk het meest gevaarlijke voor de applicatie zal zijn. Zelf heb ik gekozen voor deze vraag met de reden dat me applicatie uit 4 componenten bestaat. Verder lijkt het me zelf ook best interessant dit te onderzoeken.

## Leeswijzer

Er is geen speciale leeswijzer voor dit document

# Welke security risk is het meest gevaarlijke voor de applicatie?

## Wat voor security risks zijn er?

Sinds de applicatie bestaat uit 4 componenten zoals uitgelegd in het voorwoord is er kans op een security risk groter dan verwacht. Als zichtbaar in figuur 1.1 bestaat de applicatie uit 4 onderdelen waarvan ik invloed heb op de gebieden die oranje zijn. Wat meteen opvalt is dat er heel veel calls worden gemaakt tussen componenten.

Dus een van de meeste opvallende die als eerste naar mind komt is Cross side scripting. Verder vormen van risk voor de applicatie die kunnen voorkomen zijn SQL injections, ook CROSS-SITE Request Forgery (CSRF), Als laatste Broken Authenticatie. Als gedachte gang lijkt mij Cross site scripting het meest gevaarlijkste voor de applicatie. Ik zal even verder uitleggen wat de impact kan zijn op de applicatie voor iedere security risk.

### SQL injections

SQL injecties zijn een van de meest voorkomende websites risk op het moment van het maken van dit document. SQL injecties zijn een manier waarop een attacker via query’s rommelt met de applicatie. Deze query’s hebben vaak toegang tot de database. Door de tempering van de attacker krijgt de attacker vaak data te zien die de attacker niet hoort te zien. Om een klein voorbeeld te geven:

Laten we uitgaan van de applicatie in figuur 1.1. Stel je voor dat de user alle songs wilt ophalen in de database waarvan de artiest Hatsune Miku is. Dus dan krijg je de call naar de backend die een query doet op de volgende database:  
**SELECT \* FROM Songs WHERE artiest = ‘Hatsune Miku’.**

Dit resulteert dat de query het volgende doet:

1. Vraagt de database alles terug te geven voor de artiest Hatsune Miku -> (\*)
2. Alles van de songs table

Deze query bevat alleen geen verdediging tegen SQL Injecties. Een manier waarop de gebruiker het kan beïnvloeden is als volgt:  
**SELECT \* FROM Songs ‘--’ WHERE artiest = ‘Hatsune Miku’**

Dit resulteert in het feit dat alles achter ‘--’ dit een comment wordt. Zo technische gezien wordt de query dan uitgevoerd dat alles van Songs wordt opgehaald. De query zou nog verder kunnen worden misbruikt als volgt:  
**SELECT \* FROM Songs WHERE artiest = “Hatsune Miku’ OR 1=1**

Dit resulteert die alles van songs opgehaald wordt waar van de artiest Hatsune Miku is of wanneer 1 is gelijk aan 1. Sinds dit altijd waar is wordt alles terug gegeven van artiest. In dit geval verandert er niet veel maar in grotere applicaties kan dit leiden tot data die de gebruiker niet hoort te zien

### XSS Cross side scripting

Cross side scripting wordt meestal gebruikt om kwaadaardige scripts in de users browsers uitvoeren. Deze scripts zijn meestal javascript die de attacker gebruikt. Meestal is het geval dat de attacker geen controle heeft over de website en de kwaadaardige scripts worden meestal boven op een legitieme website uitgevoerd. Alleen hoe werkt XSS nu? Laat me een voorbeeld geven. Een note er zijn meerdere soorten cross side scripting. Voorbeelden zijn te vinden in de bron XSS. Meestal kunnen deze worden opgedeeld in 3 categorieën.

#### Refelected XSS

In geval we spreken van Refelected XSS dan hebben we het over een malicious script die niet wordt bewaart op de server. Allen de malicious script is zichtbaar in de websites zijn resultaten.

#### Stored XSS

Het tegenovergestelde van reflected XSS. In dit geval wordt de malicious script gesaved door een server exploit op de server zelf. Stored XSS hebben meestal een groter bereik dan Reflected XSS.

#### DOM XSS

Dit soort XSS komt zelden voor. Alleen bij dit soort types XSS wordt de DOM aangepast. Verder wordt de code van de website niet aangeraakt.

#### XSS Example

Als in het geval bij SQL injecties is het heel erg lastig om een concreet voorbeeld te geven van XSS. Dit met de reden omdat er zoveel taken zijn van XSS. Er is wel een meest voorkomend voorbeeld of een geval wat vaker voorkomt. Het voorbeeld is als volgt:

Laten we zeggen dat we een basic search input hebben als volgt:  
**<input type=”search” value=””>**

Dit geval is een mooi voorbeeld want dan kan de attacker het volgende doen om een XSS aanval uit te voeren.  
**<input type=”search” value=” “/> <script>StealCookies()</script>**

In dit voorbeeld wordt de search bar misbruikt. Attackers gebruiken dit vaker om een users te misleiden naar een malicious pagina waar dan de rest van de actie gebeurt. Search bars zijn meestal XSS points op de website die kunnen worden nagekeken.

### Cross site request forgery (CRSF)

Cross site request forgery is voor mij ook een nieuw website security risk waar van ik niks wist, of alleen de naam niet van wist het concept ben ik tegen gekomen. CRSF technische gezien houdt in dat er een actie wordt uitgevoerd die de gebruiker.

In figuur 2.1 kan je een hedendaagse mooi voorbeeld zien dat iedereen kent. We hebben allemaal wel is de mail ontvangen dat je Bank kaart vervallen is of dat je paypal een issue heeft. Als figuur 2.1 laat zien is wat er gebeurt op het moment dat we ervoor vallen. Het gebeurt vaker dan je denkt want we krijgen vaak paniek en stress ervan en denken niet na een stap.

Sinds Spotify is een van de componenten die wordt gebruikt is het risk van neen CRSF toch dichterbij. Veel mensen hebben gevoelige data op Spotify dus de risco is er dat er dus voorgaat komen dat hackers toch gaan proberen om aan die gevoelige data te komen. Ook hoewel CRSF lastig te voorkomen is het toch belangrijk het te proberen

### Broken Authencation

Broken Authencation is een risico voor de applicatie. Hoewel de spotify API aardig secure is en het idee verder dan de gebruikers naam te gebruiken in de applicatie, kan het voorkomen dat er ergens Broken Authencation voor komt. BA ook wel komt vaak voor in kleinere delen. Zoals het niet hashen van paswoorden, Geen multiple Authencation voor de application, etc. De lijst kan zo doorgaan maar BA is meestal een issue voor de developers meer dan de gebruiker zelf. BA ligt eigenlijk vaker buiten dan hand van de gebruiker

## Why is security important for the application

Wat ik probeer te bereiken met me onderzoek en waarom ik voor dit onderwerp heb gekozen is om te zien wat is nu een groot risico voor de applicatie. Je hebt veel verschillende security risico’s op de heden daagse web, ook bestaat de kans dat morgen er een compleet andere risico tevoorschijn komt die een impact krijgt op de applicatie.

Zo het is belangrijk te onderzoeken wat een gevaar vormt in de applicatie, in plaats van te onderzoeken wat een risico vormt buiten de applicatie. Dit zeg ik met de reden dat je beter kan beschermen van binnen uit dan van buiten af.

Aangezien er wordt gewerkt met gevoelige informatie die kan leiden tot rampzalige gevolgen als het maar zo onder het tapijt te schuiven. Ook als we terug kijken op figuur 1.1 kunnen we zien dat de Front end eigenlijk het belangrijkste punt is in de applicatie.

Sinds de Front end communiceert met de back end en de spotify API kunnen we ervanuit gaan dat daar de security het meest moet worden getest. Als stakeholders wil je natuurlijk weten dat de applicatie goed beveiligt is vooral vanwege de gevoelige data.

Over wat voor data spreken we dan dat dingen als XSS en CRSF een probleem gaan vormen. Meeste mensen zullen een email hebben gekoppeld aan spotify, daarbij zullen sommige bijvoorbeeld inloggen via een alternatieve methode voor spotify. Als laatste gebruiken sommige spotify ook op hun mobiel. Data die in ergste geval kan leiden tot serieuze schade. Vandaar dat het belangrijk is om uit te zoeken wat een gevaar punt is in de applicatie.

# How do we secure the application better?

Denkende hoe je de applicatie securde kan maken of antwoord vinden op de vraag van wat vormt een risico voor de applicatie. Dit brengt je eigenlijk terug als eerst bij 2 gedachtes. Vooral library and field research. Libarary met de reden dat er al vaker is gekeken naar dit soort security risks en dus andere mensen het al is eerder hebben gemaakt en gedaan. Dit brengt me ook bij het tweede deel namelijk field research.

Field research is zeker belangrijk in dit geval. Een mooi voorbeeld is om zeker websites als OWASP te blijven volgen voor een geval er een nieuwe groot website risico komt. Op deze website zijn vaak links te vinden naar voorbeelden en verwijzingen hoe je het kan voorkomen.

Ik zal kort ff uitleggen hoe het plan zal werken voor iedere onderwerp:

## Libarary research

Libarary research is waarschijnlijk een van de belangrijkste factoren in de onderzoek. Aangezien we er al eerder mee bezig zijn geweest als developers. Groote kans dat er dus al iets voor is wat de boven opgenoemde security risico`s kan voorkomen.

De strategie is als volgt te gaan kijken naar wat er al bestaat en voorbeelden te gaan maken en kijken of de voorbeelden als verwachting te gaan werken. Dit moet uiterlijke 1 dag of max 2 dagen in beslag gaan nemen.

Aan de hand van de resultaten kunnen we overstappen naar het volgende doel van het onderzoek wat eigenlijk deels hand in hand gaat met dit deel van de research.

## Field research

Field research houdt simpel in dat je opzoekt gaat naar je eind gebruikers. Dus denk eraan gebruikers die het bijvoorbeeld interessant vinden om te zien waar ze naar luisteren op Spotify. Van wat zijn hun behoefte bij het gebruik van zo applicatie en wat vinden zij belangrijk kwa beveiliging.

De strategie is eigenlijk best simpel interview nou is een paar gebruikers die je applicatie gaan gebruiken. Vraag is naar hun wensen voor de applicatie en ook vragen naar de wensen rond om security. Vanuit daar zullen er een aantal eisen naar voorkomen die we weer kunnen gebruiken in de applicatie.

Met de info van field en Libarary research kunnen we nu naar de volgende strategie

## Lab research

Lab research is een klein deel in dit plan. Het houd technische gezien in dat er de mogelijkheid is om dingen te testen van of concepten van je product, of van het eind product. Dus in dit part is voor de strategie mogelijke om dingen te kunnen testen en te zien of het werkt naar gedachte.

Vanuit hier gaan we naar de 1 na laatste stap. Voor ons idee voor onderzoek

## Workshop research

Voor we ons prototypes en ideeën brengen naar de stake holder moet het nog wel worden getest. Dit kan worden samen gevoegd met de show room. Dus de strategie voor de workshop research sectie is als volgt.

Prototyping en testing 2 dingen die common staan hier. In dit deel van het onderzoek moet worden gekeken of de verwachting werken naar wens en of er niet iets moet worden aangepast. Mocht er iets cruciaal worden aangepast dan is het terug naar de teken tafel en het van af vooraan weer uitproberen op de Workshop research.

Als het is allemaal werkt en er geen cruciale dingen zijn gevonden die geen aandacht nodig hebben dan kunnen we het idee en product tonen

## Showroom Research

De final step, de laatste stretch. Op dit moment is het waarschijnlijk zover dat je een product klaar is om op een test server te gezet te worden of dat je toevallige een release moet doen voor je stakeholder.

De strategie hier is te zien of onze research idee werkt tegen het werk dat er al is. Daarnaast het product laten testen door bijvoorbeeld mede developers is ook een goed idee om te kijken of XSS werkt en CRSF. Daarnaast gaan we nog de applicatie testen tegen genereal guide Lines voor XSS and CRSF om te zien of ze daar ook aan voor doen.

# Conclusie

Aan het begin van het onderzoek dacht ik al dat XSS een groot gevaar ging vormen voor de applicatie aangezien je werkt met aardige gevoelige data. Alleen ben ik toch na mening dat CRSF ook een aardig grote impact kan hebben. Alleen ligt daar wel een issue mee XSS is een manier die ik kan preventen en CRSF niet.

Na het onderzoek lijkt het toch dat de risico voor CRSF groter is dan XSS. De reden waarom is dat CRSF een manier is die ligt bij verantwoording van de gebruiker. Ik kan dingen als SQL injecties, XSS en Broken Authencation voorkomen maar CRSF niet. Daarom met die reden trek ik de conclusie dat CRSF de grootste risico vormt voor de applicatie.

# Bronnen

# Verwijzingen

bassi, B. (2021, January 04). *6 common website securit vulnerabilties*. Retrieved from commonplaces: https://www.commonplaces.com/blog/6-common-website-security-vulnerabilities/

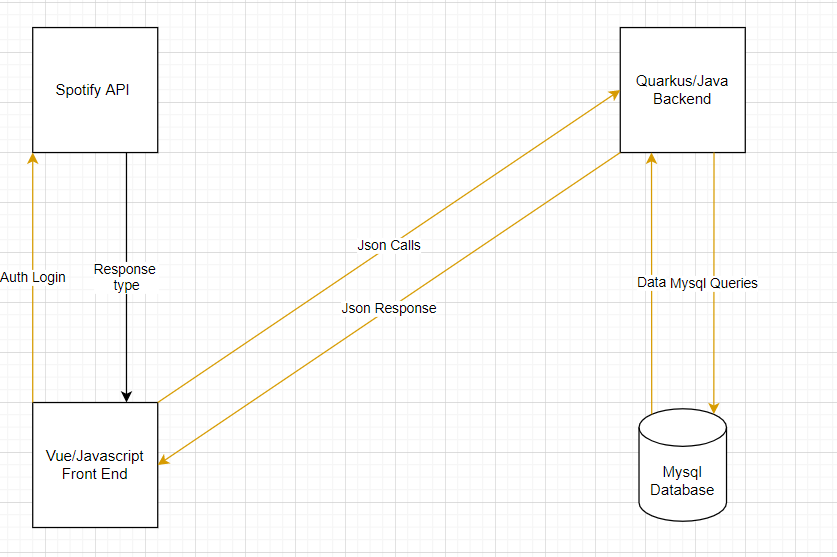
*Cross-side Scripting (XSS) Attacks*. (2019, November 7th). Retrieved from sucuri.net: https://sucuri.net/guides/what-is-cross-site-scripting/

*OWASP Top ten Web application Security Risks*. (n.d.). Retrieved from owasp: https://owasp.org/www-project-top-ten/

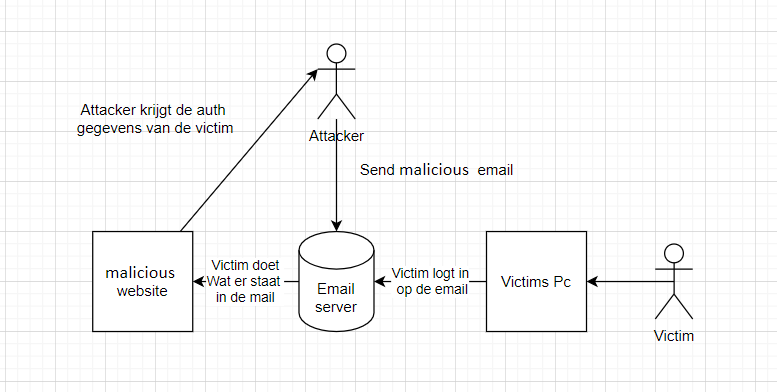
*What is SQL injection* . (n.d.). Retrieved from potswigger : https://portswigger.net/web-security/sql-injection

*XSS and test attacks*. (2021, June 6). Retrieved from softwaretesting help: softwaretestinghelp.com/cross-site-scripting-xss-attack-test/

## Images Bronnen



Figuur 1.1 An general overview of the application



Figuur 2.1 Example of CRSF