Security

case studie

S67-3

2021

Inhoud

[The case 2](#_Toc68084870)

[Onderzoeksvragen 4](#_Toc68084871)

[Strategie 5](#_Toc68084872)

[Resultaten 6](#_Toc68084873)

[Wat zijn de meest voorkomende security risico’s in een softwareapplicatie? 6](#_Toc68084874)

[Welke technieken zijn er beschikbaar om de security risico’s te beperken of elimineren? 6](#_Toc68084875)

[Injection 6](#_Toc68084876)

[Broken Authentication 7](#_Toc68084877)

[Sensitive Data Exposure 7](#_Toc68084878)

[XML External Entities (XXE) 8](#_Toc68084879)

[Broken Access Control 8](#_Toc68084880)

[Security Misconfigurations 9](#_Toc68084881)

[Insecure Deserialization 9](#_Toc68084882)

[Using components with known vulnerabilities 10](#_Toc68084883)

[Insufficient Logging & Monitoring 10](#_Toc68084884)

[Hoe kunnen deze technieken tijdens het agile ontwikkelen worden toegepast 11](#_Toc68084885)

[Conclusie 12](#_Toc68084886)

[Security binnen het VLL project 13](#_Toc68084887)

[OWASP 13](#_Toc68084888)

[Bronnen 15](#_Toc68084889)

# The case

Author: Leon Schrijvers

Date: January 28, 2021

Recently, a small development agency ('Fast Campaigns') has landed a large, high profile data driven project. For the company, this is a great opportunity, as until now they have only worked on small, short-living web applications (mostly Wordpress based marketing campaign websites). Because the developers are very excited to work on this project, they start programming right away!

The team is pushing features fast, and it looks like they are even able to deliver on time! One week before the deadline of the project, one of the team members reads a news article about a security data breach in a well known government website ([RIVM Infectieradar](https://nos.nl/artikel/2336416-lek-in-rivm-coronasite-gegevens-van-gebruikers-makkelijk-in-te-zien.html)). The next day, during the daily standup meeting, he tells the team about this and asks: "Shouldn't we have done something related to security?". The team members start to mumble indistinctively which makes the project manager very nervous.

Because the deadline of the project is in less than a week, he quickly decides to hire a Pentester to check if the project contains any major security risks. Luckily, a Security Consultant in his network is available and is able to start his security analysis the next day. The consultant starts with a Black Box pentest, and it doesn't take long to come up with a huge list of possible security threats. The consultant decides to dive deeper, and after performing a White Box pentest, it becomes clear that the development team haven't put any effort into developing a secure project. They only focused on delivering functionality as fast as possible.

The list of security issues and resulting rework is huge and cannot be ignored. This results in a massive overrun of the budget of the project. Needless to say, the project manager is not amused (understatement). The development team now have learned (in an unfortunate, expensive way) that security isn't something that you can do afterwards. The Security Consultant tells the team about Security by Design and why it is important to keep security into mind during all phases of the software development lifecycle, especially when processing (personal) data.

The consultant gives the development team some pointers to begin with (e.g. security requirements & threat analysis, secure design & implementation and security verification & testing). Furthermore, he explains that the subject of security is very broad, so that they will be even more topics to explore. To process all this new information in a structured way, he encourages the team to come up with a plan in which they can define what security related activities should always be done in the different phases of the software development lifecycle for future projects.

##### Case study goal

Your task is to come up with a 'Security by Design Plan', in which is defined what security related activities should be done in the different phases of the software development lifecycle. When doing this, try to be as concrete as possible (think about concrete security threats, mitigations, design decisions, implementation guidelines, and so on), so that it adds as much value as possible to future projects.

Part of the plan is technology selection: select supporting solutions that suitable for (parts of) a secure design and implementation. Make clear what alternatives you considered and what your selection criteria were.

##### Purpose for the group project

Think about what knowledge gained in this case study can be translated to group project:

* Write the plan in such a way that it can be directly used in the group project
* Add security related tasks to the product backlog (make sure your PO is involved in this process)
* Decide what supporting technology can also be used for your group project and motivate why.

# Onderzoeksvragen

*“Welke technieken kunnen er gebruikt worden om de veiligheid in een softwareapplicatie te verzekeren?”*

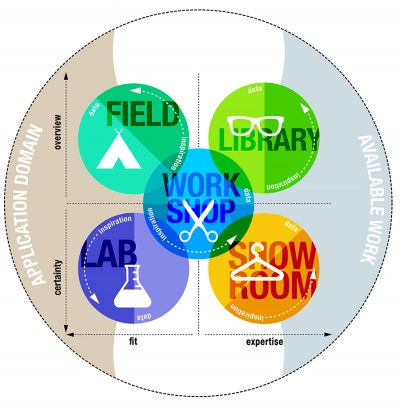
Om deze hoofdvraag goed te kunnen beantwoorden, is het belangrijk om eerst te onderzoeken tegen welke security risico’s er ‘verdedigd’ moet worden en welke technieken hiervoor beschikbaar zijn.

* Wat zijn de meest voorkomende security risico’s in een softwareapplicatie?
  + Welke technieken zijn er beschikbaar om de security risico’s te beperken of elimineren?
* Hoe kunnen deze technieken tijdens het agile ontwikkelen toegepast worden?

Naast een samenvatting van de antwoorden op de onderzoeksvragen in de vorm van een conclusie, zullen we ook bespreken welke van de technieken we als groep al in onze projecten toepassen en welke technieken we in de toekomst in onze projecten willen toepassen.

# Strategie

Het verrichten van het onderzoek zal gebeuren met behulp van het DOT Research framework. Voor het gebruik van het framework is het van belang dat er verschillende methodes worden gebruikt. Deze methodes komen uit verschillende strategieën en worden toegepast om je onderzoek te valideren.



Om de onderzoeksvraag te beantwoorden hebben wij gekozen voor de volgende strategieën met daarbij de volgende methodes.

|  |  |  |  |
| --- | --- | --- | --- |
| Field | Library | Workshop | Lab |
| Focus group | Community research | Decomposition | Security test |
|  | Available product analysis | Brainstorm |  |
|  | Best good and bad practices |  |  |

# Resultaten

## Wat zijn de meest voorkomende security risico’s in een softwareapplicatie?

Om dit te onderzoeken hebben we gebruik gemaakt van de openbare informatie van OWASP. Hierbij hebben we gekeken naar de top 10 kwetsbaarheden van 2021. Deze staan hieronder in het rijtje:

*“*

1. *Injection*
2. *Broken Authentication*
3. *Sensitive Data Exposure*
4. *XML External Entities (XXE)*
5. *Broken Access Control*
6. *Security Misconfigurations*
7. *Cross-Site Scripting (XSS)*
8. *Insecure Deserialization*
9. *Using Components with Known Vulnerabilities*
10. *Insufficient Logging and Monitoring*

*” - (OWASP Top 10 security, 2021)*

Deze 10 punten willen we allemaal langsgaan hieronder. Hierbij een korte beschrijving met wat het precies is en welke technieken gebruikt kunnen worden tijdens het ontwikkelproces om de applicatie hiervan te beveiligen.

## Welke technieken zijn er beschikbaar om de security risico’s te beperken of elimineren?

### Injection

Vaak denken mensen bij injection dat het hier gaat om SQL queries. Echter gaat het bij injection bij meer dan dit. *“Injection vulnerabilities are often found in SQL, LDAP, XPath, or NoSQL queries, OS commands, XML parsers, SMTP headers, expression languages, and ORM queries.” - (OWASP, 2017)*

Hieronder is een lijst opgesteld met mogelijke technieken om dit injection kwetsbaarheden te voorkomen:

* Input validation
* Parametrized queries
* Stored procedures
* Escaping
* Access rights

Een andere manier om injection te voorkomen is door het gebruik van een ORM (Object-Relational Mapper). Deze ORM’s zijn ver ontwikkelde tools waarin de bovenstaande punten meestal al ingebakken zitten. Afhankelijk van de taal waarin er gewerkt wordt zijn er andere ORM’s beschikbaar.

### Broken Authentication

Authenticatie is ook een groot probleem in systemen voor de veiligheid. “Session management is the bedrock of authentication and access controls, and is present in all stateful applications.” *- (OWASP, 2017b)*

Het gaat volgens OWASP dus niet alleen om het authentiseren zelf maar ook om de sessie. Dit is vooral een probleem in stateful applicaties en oudere websites geschreven in PHP.

De volgende acties kunnen ondernomen worden om dit risico te beperken:

* Multi-factor authenticatie
* Geen default credentials gebruiken, zeker niet voor admin accounts
* IP Checking
* Nonce *“Typically with a nonce, you have a per-page "token" so that each page checks that the last page's nonce matches what it has stored.” - (How safe are PHP session variables?, 2009)*
* Weak password checks
* Gebruik van reCAPTCHA of limiet/vertraging op herhaald foutieve inlogpogingen
* Gebruik van een server side sessie manager

### Sensitive Data Exposure

Gevoelige data is extra kwetsbaar in de gebruikers client of terwijl het onderweg is. “*Rather than directly attacking crypto, attackers steal keys, execute man-in-the-middle attacks, or steal clear text data off the server, while in transit, or from the user’s client, e.g. browser.” - (OWASP, 2017c)*

Om het lekken van gevoelige data te voorkomen kunnen de volgende maatregelen getroffen worden:

* Classificeer de verwerkte, verzonden en opgeslagen data in de applicatie. Bepaald vervolgens welke data gevoelig is.
* Sla gevoelige data niet onnodig op.
* Encrypt alle gevoelige data die wel opgeslagen wordt.
* Verzeker dat er up-to-date en sterke algoritmes worden gebruikt voor het encrypten van gevoelige data.
* Encrypt verzonden data in requests en maak gebruik van protocollen zoals TLS, HTTPS en HSTS.
* Voer geen caching uit voor responses die gevoelige data bevatten.
* Sla wachtwoorden op met sterke hashing en een delay factor zoals scrypt, bcrypt of PBKDF2.

### XML External Entities (XXE)

Veel oudere of slecht geconfigureerde XML-processors evalueren externe entiteitsreferenties binnen XML-documenten. Externe entiteiten kunnen worden gebruikt om interne bestanden openbaar te maken met behulp van de bestands-URI-handler, interne bestandsshares, interne poortscanning, uitvoering van externe code en denial of service-aanvallen.

OWASP raadt de volgende dingen aan om dit te voorkomen:

* Het gebruiken van minder complexe data formats zoals JSON
* Persoonlijke data niet serialiseren
* Werk altijd XML-processors bij en gebruik een checker die automatisch kijkt of er een nieuwe versie beschikbaar is
* Werk het SOAP protocol bij naar minimaal versie 1.2

### Broken Access Control

In veel applicaties wordt er wel goede authenticatie toegepast, maar blijft de autorisatie achter. Hackers kunnen dit misbruiken door toegang te krijgen tot unauthorized functies of data. Bijvoorbeeld het inzien of aanpassen van data van een andere gebruiker.

Om broken access control te voorkomen kunnen de volgende stappen ondernomen worden:

* Deny by default
* Model access controls moeten record ownership meenmenen
* Diable web server directory listing, verzeker file metadata en maak backups van data
* Log access control failures
* Rate limit API en controller toegang om automatische aanvallen te beperken
* Disable JWT tokens na het uitloggen

### Security Misconfigurations

Een foute configuratie kan op veel plekken plaats vinden. OWASP geeft de volgende voorbeelden: *“Security misconfiguration can happen at any level of an application stack, including the network services, platform, web server, application server, database, frameworks, custom code, and pre-installed virtual machines, containers, or storage.” -* (OWASP, 2017c)

Omdat configuratie vaak afgestemd moet worden op het systeem is het soms lastig om hier zwart op wit te bepalen hoe je ervoor kan zorgen dat je configuratie juist is van tevoren. Wel zijn er adviezen en richtlijnen die gevolgd kunnen worden:

* Automatische scanners voor het detecteren van verkeerde configuraties
* Automatische scanners die kijken op verouderde of legacy configuraties
* Een standaard maken voor een applicatie en deze zo kort mogelijk houden
* In je framework geen overbodige of ongebruikte libraries gebruiken welke een eigen configuratie hebben

Cross-Site Scripting (XSS)

XSS laat hackers scripts uitvoeren op de browser van het slachtoffer waardoor de sessie overgenomen kan worden, gevoelige data gestolen kan worden of het slachtoffer naar kwaadaardige websites gestuurd kan worden.

De onderstaande punten kunnen toegepast worden om XSS te voorkomen:

* Gebruik van frameworks die XSS automatisch escapen
* Escapen van onbetrouwbare HTTP requests
* Enable een Content Security Policy (CSP)

### Insecure Deserialization

Insecure Deserialization Is het deserialiseren van objecten welke aangepast kunnen zijn welke weer impact hebben op het systeem. Een voorbeeld hiervan is dat op een PHP-website bij het inloggen een cookie maakt van de gebruiker. Door deze op te vangen en de role aan te passen van “user” naar “admin” kan een aanvaller zichzelf of anderen administrator privileges geven op het systeem.

De enigste manier om dit volledig te voorkomen is om geen geserialiseerde objecten van onbekende bronnen te vertrouwen of om enkel primitieve data types te gebruiken. Als dit niet mogelijk is zijn de adviezen als volgt:

* Implementeren van integriteit checks door bijvoorbeeld digitale handtekeningen
* Het monitoren en loggen met daarbij ook een waarschuwing als een gebruiker herhaaldelijk deserialisatie probeert toe te passen

### Using components with known vulnerabilities

Dit is een aannemelijk concept zelfs voor mensen die zelf geen ontwikkelaars zijn. Toch gaat dit in de praktijk volgens OWASP toch nog vaak fout. Dit komt vaak vooral omdat ontwikkelaars naarmate een project groter wordt niet meer goed weten welke componenten en dependencies allemaal in het project verspreid zitten. Het advies is om hier altijd over na te blijven denken en meldingen niet te negeren waarbij wordt aangegeven dat iets verouderd is of binnenkort vervangen gaat worden.

Daarnaast zijn de volgende globale adviezen van toepassing:

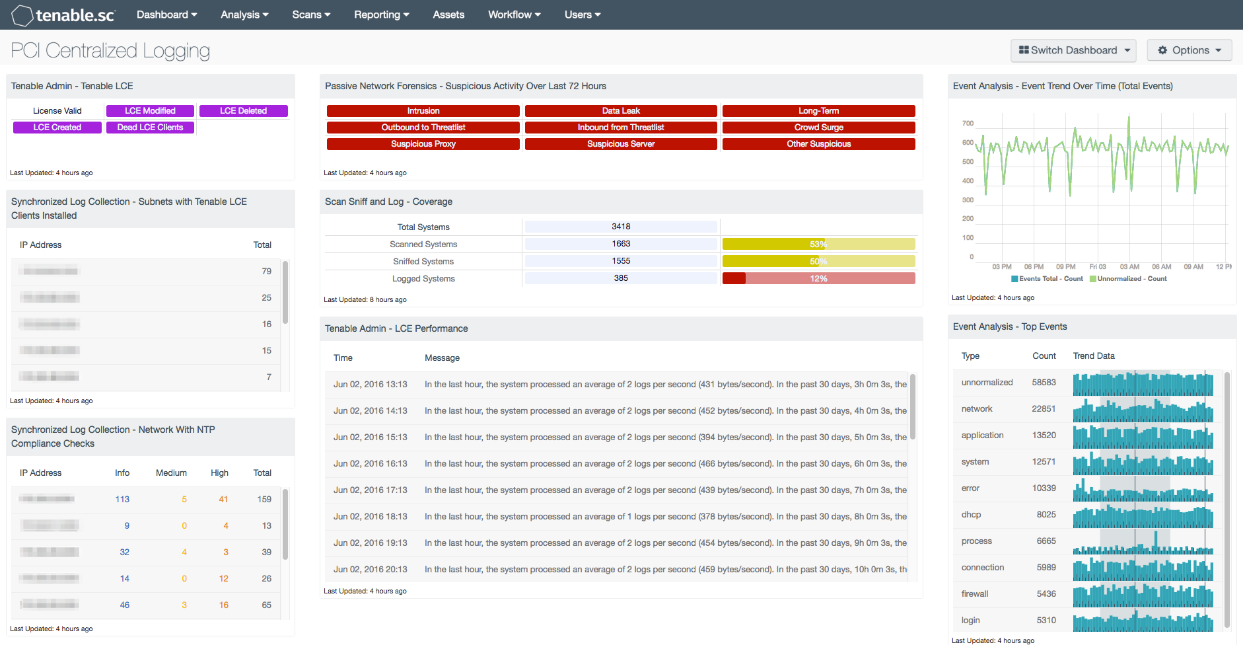
* Verwijder ongebruikte dependencies, componenten bestanden en documentatie
* Blijf geautomatiseerd inventariseren welke versies een systeem gebruikt in de verschillende frameworks en libraries. Hierbij kunnen tools zoals retire.js en DependencyCheck gebruikt worden
* Installeer alleen componenten via officiële kanalen. Dus niet handmatig via bijvoorbeeld een zip bestand gedownload op een willekeurige website

### Insufficient Logging & Monitoring

Als laatste is volgens OWASP het een groot probleem dat applicaties vaak niet voldoende loggen en/of monitoren. Dit is zowel belangrijk om live aanvallen te detecteren als achteraf kunnen bepalen wat voor schade een hack heeft aangericht. Algemene adviezen hiervoor zijn als volgt:

* Zorg ervoor dat alle inlogpogingen gelogd zijn. Bij voorkeur ook een geautomatiseerde melding bij verdachte activiteiten. Een concreet voorbeeld is om een sms te laten sturen als er binnen 1 minuut 5 mislukte aanmeldpogingen geweest zijn van een administrator account.
* Zorg ervoor dat logs een bepaalde format volgen binnen een organisatie zodat de logs van meerdere systemen ook leesbaar zijn op 1 centraal punt
* Zorg ervoor dat audit logs ook worden bijgehouden. Denk hierbij vooral aan een administrator account die een database drop query uitvoert of vergelijkbare scenario’s
* Denk van tevoren na over procedures die gevolgd moeten worden op alerts die uit de monitoring service voorkomen

Er zijn meerdere gecentraliseerde logging tools op de markt. Het advies is om hier als ontwikkelaar onderzoek naar te doen om te zien welke het beste zou werken in de omgeving. Een voorbeeld van een centrale logging is te zien in de afbeelding hieronder.



## Hoe kunnen deze technieken tijdens het agile ontwikkelen worden toegepast

Tijdens het agile kunnen veel dingen halverwege de ontwikkelingsfase veranderen, een ervan is security. Wat voor security risico’s je hebt in een applicatie ligt aan de functies die je bied met je applicatie, dit betekent dat je voor je gaat programmeren goed moet nadenken welke functies welke risico’s meenemen.

Door vooraf te kijken welke risico’s er in je applicatie zullen zijn kun je een concreet plan maken over welke technieken er toegepast moeten worden om deze risico’s te verwijderen. Door het agile werken is het natuurlijk wel mogelijk om terug te gaan naar een oudere module om hier nog security toe te voegen, maar in het geval van security is het beter om van tevoren te bekijken welke security measures ingebouwd moeten worden om tijd te besparen.

Dit kan ook op twee manieren worden gedaan. Als alle ontwikkelaars verstand van security hebben kunnen alle ontwikkelaars hun eigen code beveiligen. Als dit niet het geval is, en het team afhankelijk is van één specialist binnen het team, dan is het misschien beter om deze ontwikkelaar alle security van het project te laten beheren en testen.

De conclusie waar we tot komen is dus ook dat de beste manier om dit aan te pakken als volgt is:

Eerst een overzicht van risico’s maken, deze verdelen onder ontwikkelaars of een dedicated securityspecialist aanwijzen. En tot slot als er tijdens het ontwikkelen nieuwe risico’s naar boven komen kunnen deze nog worden gemaakt door de agile werk ethiek.

Vervolgens moet in het onderhoud van de applicatie ook regelmatig gecheckt worden of alles nog wel veilig is. Dit betekent dat tijdens het ontwikkelen hier ook al over nagedacht moet worden. Zo moet er nog een monitoring service of tool zijn welke na deployment ook blijft bijhouden of er geen known vulnerabilities bekend zijn geworden in componenten die het systeem gebruikt.

# Conclusie

Hier willen we terugkomen op onze hoofdvraag. Deze luidde als volgt:

*“Welke technieken kunnen er gebruikt worden om de veiligheid in een softwareapplicatie te verzekeren?”*

Er zijn velen technieken die gebruikt kunnen worden om tijdens het ontwikkelproces al rekening te houden met de beveiliging van de applicatie. Veel kan opgevangen worden door centrale logging en monitoring. Dit kan op veel vlakken zoals:

* Het monitoren van verouderde componenten
* Het monitoren van nieuwe known vulnerabilities in gebruikte componenten
* Het loggen van alle aanmeldpogingen
* Het monitoren van mislukte aanmeldpogingen
* Het loggen van audit handelingen

Naast het loggen is het ook belangrijk om een concreet plan te maken waar rekening wordt gehouden met alle risico’s die mogelijk zijn binnen het project en hoe we deze risico’s kunnen tegen gaan. Dit gecombineerd met logging en monitoring zou op zichzelf ervoor moeten zorgen dat de security van de applicatie zodanig goed is dat merendeels van de aanvallen ineffectief zouden zijn op het systeem.

Door deze technieken toe te passen (de technieken beschreven in het OWASP-rapport onderaan dit document) zal de applicatie goed beveiligd zijn, wel is er nog de mogelijkheid om een pen test uit te voeren om deze technieken te testen.

Dit allemaal samengevoegd zou goed beveiliging moeten bieden voor de applicatie en bijna alle aanvallers afslaan.

# Security binnen het VLL project

## OWASP

*Injection*

Injections in het groepsproject worden voorkomen door het gebruik van ORM’s (object relational mapper). Een ORM controleert de inputs en queries die uitgevoerd moeten worden op karakters die een injection zouden kunnen starten. Een ORM vervangt deze karakters/vervormd ze zodat injections niet werken.

*Broken Authentication*

Authenticatie zal geregeld gaan worden door 1 specifieke service in het groepsproject. Sessies en Credentials worden door een extern system geregeld. Zo bevat ons project minder verantwoordelijkheid omtrent authenticatie en wordt dit gedaan door een system wat daar zich op focust.

*Sensitive Data Exposure*

Data communicatie zal binnen een Kubernetes cluster gaan. De communicatie is dus in een gesloten omgeving. Data zou wel gelekt kunnen worden door calls niet te authenticaten. Dit kan gebeuren door menselijke fouten. Data die gecommuniceerd wordt gaat zo min mogelijk gevoelige data delen door gebruik te maken van data transfer objecten.

*XML External Entities (XXE)*

Het groepsproject maakt geen gebruik van XML of externe XML objecten. Communicatie gaat via REST met hulp van JSON objecten. Deze JSON objecten worden automatisch gegenereerd op basis van de modellen die gedefinieerd zijn in code.

*Broken Access Control*

Broken Access Control kan voorkomen als een webpagina alleen kijkt of een gebruiker is ingelogd, maar niet goed kijkt of de ingelogde gebruiker wel mag doen waar hij mee bezig is. Een voorbeeld hiervan is inloggen en daarna de data van een andere gebruiker veranderen.

Om dit tegen te gaan is het belangrijk dat een systeem niet alleen kijkt of een gebruiker is ingelogd. Maar ook of de ingelogde gebruiker de actie waar hij mee bezig is wel mag doen. Bijvoorbeeld kijken of het wachtwoord wat de gebruiker wil aanpassen ook daadwerkelijk van dezelfde gebruiker is.

*Security Misconfigurations*

Dit is een kwestie van zorgvuldig werken. Een van de belangrijkste en simpelste manieren is er voor zorgen dat modules/code die niet meer wordt gebruikt worden verwijderd, en dat de bestaande code en configuratie up-to-date blijft met alles.

*Cross-Site Scripting (XSS)*

XSS kan erg moeilijk zijn om tegen te verdedigen, maar gelukkig zijn hier op vandaag erg veel voorbeelden om te gebruiken. In onze app zal veel data worden weergeven die door gebruikers is geüpload. Dit betekent dat onze app ook heel erg kwetsbaar is voor XSS aanvallen.

De beste manier om dit tegen te gaan is om te zorgen dat we zo veel mogelijk user input nakijken op XSS aanvallen. Dit is te doen door onder andere haakjes en andere speciale tekens te weigeren/escapen uit de user input. Als dit niet goed wordt gedaan kan het er voor zorgen dat elke gebruiker die op een bepaalde pagina komt, bijvoorbeeld geredirect kan worden naar de aanvaller hun website.

*Insecure Deserialization*

Insecure deserialization kan worden tegengehouden door onder andere geen serialized objecten aan te nemen van onbekende bronnen. Door dit te doen is de kans dat een aanvaller deze soort aanval pleegt veel kleiner, dit is niet al te moeilijk toe te passen en is een kwestie van configuratie in de applicaties.

*Using Components with Known Vulnerabilities*

Een van de makkelijkste security risico’s om te vermijden, dit doen we dan ook in het groepsproject door veel modules zelf te schrijven, en als we externe modules gebruiken deze van te voren eerst goed te bestuderen om zeker te zijn dat deze ook daadwerkelijk veilig zijn.

*Insufficient Logging and Monitoring*

Logging kan worden gebruikt om verschillende security aanvallen te voorkomen en te herkennen. In het groepsproject kan dit gedaan worden met de kafka logging functie. Op deze manier kunnen we alle traffic naar de backend monitoren en verdachte activiteit flaggen.

Door dit te doen kan het zo zijn dat sommige aanvallen voor dat deze door komen al kunnen worden tegengehouden. Ook kunnen de logs worden gebruikt om de beveiliging van de applicatie te verbeteren in de toekomst.

# Bronnen

*How safe are PHP session variables?* (2009, 25 juli). Stack Overflow. https://stackoverflow.com/questions/1181105/how-safe-are-php-session-variables#:%7E:text=4%20Answers&text=Sessions%20are%20significantly%20safer%20than,own)%2C%20and%20using%20a%20nonce

OWASP. (2017a, maart). *A1:2017-Injection | OWASP*. Owasp.org. https://owasp.org/www-project-top-ten/2017/A1\_2017-Injection

OWASP. (2017b, juni). *A2:2017-Broken Authentication | OWASP*. https://owasp.org/www-project-top-ten/2017/A2\_2017-Broken\_Authentication

*OWASP Top 10 security*. (2021). Imunniweb OWASP Top 10. <https://www.immuniweb.com/resources/owasp-top-ten/>

OWASP. (2017c, juni). *A6:2017-Security Misconfiguration | OWASP*. <https://owasp.org/www-project-top-ten/2017/A6_2017-Security_Misconfiguration>

OWASP. (2017d, augustus). *A8:2017-Insecure Deserialization | OWASP*. https://owasp.org/www-project-top-ten/2017/A8\_2017-Insecure\_Deserialization