2022

OWASP Rapportage

SAMIR ZALMAY

Contents

Inleiding 3

Injection 4

- Wat is SQL injection? 4
- Hoe komt dit tot stand? 4
- Zijn er specifieke acties nodig voor mijn applicatie? 4

Broken authentication 5

- Wat is broken authentication? 5
- Hoe komt dit tot stand? 5
- Zijn er specifieke acties nodig voor mijn applicatie? 5

Sensitive Data Exposure 6

- Wat is sensitive data exposure? 6
- Hoe komt dit tot stand? 6
- Zijn er specifieke acties nodig voor mijn applicatie? 6

XML External Entities (XXE) 7

- Wat zijn XML external entities? 7
- Hoe komt dit tot stand? 7
- Zijn er specifieke acties nodig voor mijn applicatie? 7

Broken Acces Control 8

- Wat is broken acces control? 8
- Hoe komt dit tot stand? 8
- Zijn er specifieke acties nodig voor mijn applicatie? 8

Security Misconfiguration 9

- Wat is security misconfiguration? 9
- Hoe komt dit tot stand? 9
- Zijn er specifieke acties nodig voor mijn applicatie? 9

Cross-Site Scripting (XSS) 10

- Wat is cross-site scripting? 10
- Zijn er specifieke acties nodig voor mijn applicatie? 10

Insecure Deserialisation 11

- Wat is insecure deserialisation? 11
- Hoe komt dit tot stand? 11
- Zijn er specifieke acties nodig voor mijn applicatie? 12

Het gebruiken van Componenten met bekende kwetsbaarheden 13

- Wat is het precies? 13
- Hoe komt dit tot stand? 13

Zijn er specifieke acties nodig voor mijn applicatie? 13
Insufficient Logging & Monitoring 13
Wat is het precies? 13
Hoe komt dit tot stand? 13
Zijn er specifieke acties nodig voor mijn applicatie? 13
Bronnen 14

Inleiding

Tegenwoordig is veiligheid erg belangrijk in de applicaties die gebouwd worden. In de afgelopen jaren zijn veel bedrijven aangevallen, doordat de veiligheid van de applicaties niet in orde waren. Een voorbeeld hiervan is Valve. Valve is een organisatie die niet alleen spellen maakt, maar ook platforms zoals Steam, die door miljoenen mensen wordt gebruikt tegenwoordig. In 2018 heeft iemand aan Valve gerapporteerd dat er met behulp van SQL injection codes voor spellen gegenereerd kunnen worden. Hierdoor zouden mensen de spellen gratis kunnen verkrijgen op een illegale manier. Valve heeft de desbetreffende persoon beloond met 25.000 dollar, aangezien hij een behoorlijk groot veiligheidsrisico heeft ontdekt voor het bedrijf. Dit maakt dus duidelijk dat een bedrijf veel geld kan verliezen door bepaalde veiligheidsrisico's.[1][2]

Hierom wordt in dit rapportage de OWASP Top 10 veiligheidsrisico's besproken. Er wordt gekeken naar hoe veilig mijn applicatie is aan de hand van de top 10. Hierdoor kunnen risico's gevonden worden. Aan de hand van de gevonden risico's kunnen oplossingen bedacht en geïmplementeerd worden, zodat de veiligheid van de webapplicatie verbeterd. De volgende categorieën worden besproken in dit document:

- Injection,
- Broken Authentication,
- Sensitive Data Exposure,
- XML External Entities (XXE),
- Broken Access Control,
- Security Misconfiguration,
- Cross-site Scripting XSS,
- Insecure Deserialization,
- Using Components with Known Vulnerabilities,
- Insufficient Logging & Monitoring.

Injection

Wat is SQL injection?

Een injection kwetsbaarheid is wanneer een aanvaller schadelijke gegevens opstuurt naar een applicatie. De schadelijke gegevens worden dan als onderdeel van een command of query gestuurd, die de applicatie opvangt. Als de aanvaller dit voor elkaar krijgt, dan wordt kwaadaardig code uitgevoerd in de applicatie. Hierdoor voert de applicatie opdrachten uit of worden gegevens onthult, die normaal gesproken niet bereikbaar zijn.

Hoe komt dit tot stand?

De meest voorkomende oorzaak, waardoor SQL injection plaatsvindt, is het niet filteren, valideren en opschonen van de input van een gebruiker door een webapplicatie. Webapplicaties waarbij gebruikers een account kunnen creëren en inloggen zullen hier altijd op moeten letten, omdat het invoeren van gebruiksgegevens vereist is zodat de applicatie kan functioneren. Helaas zullen er altijd mensen zijn die hier misbruik van willen maken. Hierbij wordt er geprobeerd om kwaadaardige gegevens mee te sturen tijdens het inloggen. Ook is er een ander belangrijk oorzaak waardoor SQL injection tot stand kan komen. Dit heeft betrekking tot het niet implementeren van parameterisation.

Zijn er specifieke acties nodig voor mijn applicatie?

Aangezien ik Entity Framework gebruik in ASP.NET Core wordt er al tegen SQL injection gewerkt. Dit komt omdat er gebruik wordt gemaakt van Linq. Linq is niet gevoelig voor traditionele SQL-injection aanvallen. Dit komt omdat Linq queries niet opgebouwd zijn uit string manipulatie of concatenatie. [43][44]

Broken authentication

Wat is broken authentication?

De kwetsbaarheden van authenticatie en sessiebeheer. Applicaties kunnen kwetsbaar zijn wanneer de authenticatie en sessiebeheer niet goed geïmplementeerd zijn. Aanvallers kunnen hierdoor toegang krijgen tot accounts en zelfs gegevens die ze normaal gesproken niet zouden mogen bekijken.

Hoe komt dit tot stand?

Broken authentication kan tot stand komen door een aantal punten:

- Het niet tegengaan van geautomatiseerde aanvallen (Brute-forcing, credential stuffing, etc.),
- Een andere oorzaak is het toestaan van zwakke wachtwoorden van gebruikers,
- Het weergeven van bijvoorbeeld session id's in bijvoorbeeld de url.
- Het gebruik van wachtwoorden, waarbij de wachtwoorden als tekst of bijvoorbeeld als zwakke hash gebruikt worden.
- Het niet implementeren van Session ID switching, nadat een gebruiker is ingelogd.
- Het te lang behouden van dezelfde session, nadat een sessie is verlaten of het inactief behouden van een session met een lange periode.
- Het niet gebruiken van multi-factor authenticatie.
- Etc.

Een voorbeeld van broken authentication is hieronder te vinden.

Example #2: Application session timeouts aren't set properly.

A user uses a public computer to access an application. Instead of selecting "logout" the user simply closes the browser tab and walks away. An attacker uses the same browser an hour later, and the user is still authenticated.

Figuur 1: Voorbeeld van een oorzaak die leidt tot broken authentication.

Zijn er specifieke acties nodig voor mijn applicatie?

Aangezien ik zelf een Authentication-Service heb opgezet, zijn er nog een aantal risico's die voorkomen zoals, slechte wachtwoorden en te lang geldig van JWT-token. Dit kan opgelost worden door in de frontend ervoor te zorgen dat wachtwoorden een minimale lengte met verschillende karakters heeft. Ook heb ik in de backend ervoor gezorgd dat de JWT-tokens maximaal 20 minuten geldig zijn, echter heb ik nog geen Refreshtokens toegevoegd. Automatische aanvallen worden verweerd door het gebruiken van Ocelot Gateway, waarbij Ratelimiting wordt toegepast. Wat ook nog toegevoegd zou kunnen worden in het project is multi-factor authenticatie. [8][9][10]

Sensitive Data Exposure

Wat is sensitive data exposure?

In de naam staat eigenlijk al wat dit probleem precies inhoudt. Wanneer een webapplicatie niet goed gevoelige informatie beveiligd. Denk hierbij aan bijvoorbeeld e-mails, adressen, postcodes, bank informatie, geboortedatums, enzovoorts.

Hoe komt dit tot stand?

De grootste oorzaken waardoor gevoelige/persoonlijke data blootgesteld wordt:

- Data overdracht gebeurt in tekst formaat,
- Gevoelige data wordt opgeslagen als tekst aan de serverkant,
- Het gebruik maken van zwakke crypto-grafische algoritmen, zoals SHA-1, MD5, etc.
- Het gebruiken van zwakke/standaard crypto-grafische sleutels,
- Het niet gebruiken van SSL of HTTPS beveiliging op webpagina's die data opslaan,
- Het niet gebruiken van gehashde én gesaltte wachtwoorden.

Een aantal voorbeelden van de oorzaken zijn hieronder weergegeven.

Example #3: The password database uses unsalted hashes to store everyone's passwords

Risk

- A file upload flaw allows an attacker to retrieve the password file. All of the unsalted hashes can be exposed with a rainbow table of precalculated hashes.

Figuur 2: Voorbeeld van het niet hashen én salten van een wachtwoord.[11]

Login form on non-HTTP page in one of Chaturbate's assets allows for a Man-in-the-Middle (MITM) attack where user credentials can be intercepted by an eavesdropping user. The publicly disclosed report can be viewed on HackerOne here.

Figuur 3: Voorbeeld van het gebruik maken van een non-HTTP pagina.

Om sensitive data exposure te voorkomen moet er dus gebruikt gemaakt worden van het versleutelen van gegevens, het minimaliseren van het gegevensoppervlak, het gebruik maken van de nieuwste versleutelingsalgoritmes, etc.

Zijn er specifieke acties nodig voor mijn applicatie?

In mijn backend maak ik gebruik van BCrypt om de wachtwoorden te hashen en te salten. Voor dit project heb ik geen Certificaat kunnen maken, waarbij ik ook gebruik zou maken van HTTPS en SSL beveiliging. Dit zou nog toegevoegd moeten worden om dit probleem te verweren. Ook wordt er gebruik gemaakt van DTO's (data transfer objects). Dit zorgt ervoor dat de interne structuur van de data niet blootgesteld wordt. Ook kan er met behulp van DTO's data verborgen gehouden worden. Je stuurt alleen de data mee die nodig is en niet meer. [8][11][12][13][44]

XML External Entities (XXE)

Wat zijn XML external entities?

XXE aanvallen richten zich op slecht geconfigureerde XML-processors, waardoor interne bestanden en gedeelde bestanden in de verkeerde handen terecht komen. Ook kunnen deze aanvallen gebruikt worden om externe code uit te voeren in applicaties of voor DoS-aanvallen. Een entiteit, in de context van XML, is een mechanisme die vervangingswaardes definieert. Je kunt het zien als een variabele die in een script gedefinieerd wordt. Deze entiteiten kunnen intern of extern aangegeven worden. XXE aanvallen mikken op alleen de externe entiteiten. Hierom moet er in dit geval alleen op de externe entiteiten gelet worden.

Hoe komt dit tot stand?

Webapplicaties en op XML gebaseerde web services kunnen kwetsbaar zijn voor een XXE aanval als één of meer van de volgende punten aanwezig zijn:

- De applicatie staat XML overdracht toe die vervolgens door een XML-processor uitgelezen wordt. Dit komt met name voor wanneer de applicatie de bron van het verzoek niet valideert bij het overdragen van een XML bestand.
- Als document type definities (DTD's) zijn ingeschakeld, waarbij de applicatie gevaarlijke gegevens toestaat binnen het identificatiesysteem van een entiteit.
- Als de applicatie SAML gebruikt.
- Als de applicatie een verouderde versie van SOAP (Simple Object Access Protocol) gebruikt. De verouderde versies zijn kwetsbaar wanneer XML entiteiten direct naar een framework worden gestuurd.

Hieronder is een voorbeeld weergegeven van een XXE aanval, waarbij de elementen 'bar' vervangen worden door 'world'. Het kan eruitzien alsof het niet kwaadaardig is, maar een aanvaller kan XML entiteiten gebruiken om 'denial of service' (DoS) te veroorzaken. Dit is een cyber-aanval waardoor een applicatie/netwerk onbruikbaar wordt voor haar gebruikers. Dit wordt ook wel 'Billion Laughs attack' genoemd. Hierdoor raakt dus het geheugen van een XML-processor overbelast.

Request

```
POST http://example.com/xml HTTP/1.1

<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE foo [
  <!ELEMENT foo ANY>
  <!ENTITY bar "World">
]>
<foo>
  Hello &bar;
</foo>
```

Response

```
HTTP/1.0 200 OK

Hello World
```

Figuur 4: Voorbeeld van een XXE aanval.[18]

Zijn er specifieke acties nodig voor mijn applicatie?

Aangezien mijn applicatie JSON gebruikt voor dataoverdracht heeft de applicatie geen XML overdracht nodig. Wel moet ik nog uitzoeken of er XML parsers aanwezig zijn in mijn applicatie, die externe entiteiten en DTD-verwerking toestaan. Ik moet ook even kijken of ik gebruik maak van een verouderde versie van SOAP.[8][15][16][17][18][19][45]

Broken Acces Control

Wat is broken acces control?

Acces control zorgt ervoor dat gebruikers geen functionaliteiten kunnen uitvoeren, waarvan ze geen rechten hebben. Fouten hierin kunnen bijvoorbeeld leiden tot het ongeoorloofd openbaar maken van informatie. Ook kan dit leiden tot het wijzig of zelfs vernietigen van gegevens. Ten slotte kan een aanvaller gebruik maken van functionaliteiten die hij/zij normaal gesproken niet zou kunnen gebruiken binnen een applicatie. In het ergste geval kan de aanvaller toegang krijgen tot de rechten van een administrator.

Hoe komt dit tot stand?

- Als een aanvaller gebruik maakt van een API-aanvalstool, waarmee acces control omzeilt wordt. Of wanneer acces control omzeilt wordt door het aanpassen van een URL of een interne applicatie state.
- Als de applicatie toestaat dat de primary key van de ene gebruiker aangepast kan worden naar de gegevens van een andere gebruiker.
- Het opvorderen van privileges van een gebruiker. Denk hier bijvoorbeeld aan het gebruik maken van admin rechten, terwijl er ingelogd is met een gebruikers account.
- Het manipuleren van metagegevens. Denk hierbij aan het opnieuw afspelen of het manipuleren van een JWT (JSON Web Token) access control token. Of het manipuleren van een cookie of een verborgen veld, waardoor de rechten van de aanvaller verhoogd kunnen worden.
- Het fout configureren van CORS (Cross-Origin Resource Sharing), waardoor ongeautoriseerde API-toegang toegestaan wordt.

Zijn er specifieke acties nodig voor mijn applicatie?

Acces control is alleen effectief, wanneer het gebruikt wordt in een vertrouwde server of server loze API. Hierdoor kan een aanvaller de Access control en metadata niet wijzigen. Mijn applicatie maakt op het moment gebruik van één URL, zodat CORS kwetsbaarheden niet voorkomen in mijn applicatie. Ook staat mijn applicatie niet toe dat gebruikers primary keys kunnen wijzigen. Wel zijn er een aantal andere zaken die hierboven genoemd zijn waar ik aandacht aan moet besteden. [8][20][21][22][23][24]

Security Misconfiguration

Wat is security misconfiguration?

Dit is tegenwoordig één van de meest voorkomende kwetsbaarheden van webapplicaties. Het verkeerd configureren van standaard configuraties, onvolledige configuraties, verkeerd geconfigureerde HTTP-headers of foutmeldingen die (teveel) informatie weergeven kunnen oorzaken zijn voor kwetsbaarheden in deze categorie. Aanvallers kunnen hierdoor kwetsbaarheden vinden van webapplicaties.

Hoe komt dit tot stand?

Hieronder zijn een aantal oorzaken weergegeven:

- Het niet hebben van een firewall in een applicatie, of het mis configureren van een firewall.
- Het toestaan van debuggen van een applicatie door gebruikers.
- Incorrecte instellingen om folders te openen/zien.
- Het gebruiken van standaard accounts en wachtwoorden voor administratieve tools/hardware.
- Het tonen van error berichten, die teveel informatie kunnen bevatten. Denk hier bijvoorbeeld aan paths van bepaalde bestanden, configuratie details, versie informatie, etc.
- Het niet versturen van security headers door de server.

Zijn er specifieke acties nodig voor mijn applicatie?

Om dit probleem te voorkomen moet er ten eerste altijd gekeken worden of er updates zijn voor dependencies, plug-ins, etc. van de applicatie. Deze updates moeten altijd zo snel mogelijk toegepast worden. De updates kunnen namelijk de veiligheid van een applicatie bevorderen. Dit hou ik zelf altijd strak in de hand. Om de week check ik of er updates zijn voor bepaalde onderdelen in mijn applicatie. Ik moet zelf nog even uitpuzzelen hoe goed beveiligd de headers in mijn applicatie zijn. Met behulp van Selenium, of een ander soortgelijk framework, kan er gekeken worden naar misconfiguraties in een applicatie. Ik maak er op het moment zelf geen gebruik van, maar tijdens dit onderzoek heb ik mezelf er wel mee kunnen informeren. [25][26][27][28]

Cross-Site Scripting (XSS)

Wat is cross-site scripting?

Cross-site scripting staat een aanvaller toe om scripts uit te voeren op de internet browser van een slachtoffer. Hierdoor kan de aanvaller de sessie van een gebruiker kapen, door cookies, session ID's enzovoorts te krijgen. Ook kan de inhoud van een webpagina gewijzigd worden door de aanvaller of een gebruiker kan omgeleid worden naar een kwaadaardige site. Er zijn een aantal veelvoorkomende type XSS aanvallen:

- Opgeslagen XSS:
  Hierbij is een script geïnjecteerd en opgeslagen op een vaste locatie in een server. Een voorbeeld hiervan is een forum bericht. Stel dat er een script is geïnjecteerd in één van de posts, die gelezen wordt door meerdere gebruikers. De gebruikers vragen de post van de server aan en worden vervolgens beïnvloed door de XSS aanval.
- Reflecterende XSS:
  Wanneer een geïnjecteerde script in een verzoek wordt gereflecteerd naar de response van een server. Een veel voorkomend voorbeeld is een zoek functie. De zoekfunctie kan een response van de server verkrijgen, die vervolgens de kwaadaardige input van een gebruiker reflecteert of de output van de informatie die wordt verkregen bij het zoeken.
- DOM gebaseerde XSS:
  In dit type aanval wordt er gebruik gemaakt van de DOM (Document Object Model). Een voorbeeld hiervan is hieronder te vinden.

Example of a DOM-based XSS flaw

Before the bug was resolved, Bugzilla error pages were open to DOM-based XSS attacks in which arbitrary HTML and scripts could be injected using forced error messages.[18]

Figuur 5: Voorbeeld DOM gebaseerde XSS.[31]

```
http://www.example.com/test.html#<script>alert(1)</script>
```

Figuur 6: Voorbeeld DOM gebaseerde XSS, waarbij een triviale XSS-payload wordt gebruikt.[25]

Kortom, Opgeslagen en reflecterende XSS komen voor wanneer een applicatie gebruikers input bewaard, waarbij de input niet is geautoriseerd, gevalideerd of niet wordt gecodeerd. De op DOM gebaseerde XSS aanvallen komen voor bij Javascript frameworks, single-page applicaties en API's die dynamische input/gegevens van gebruikers bevatten.

Zijn er specifieke acties nodig voor mijn applicatie?

Om dit probleem te verhelpen, kan er gezocht en getest worden op XSS kwetsbaarheden. Een tool die hiervoor gebruikt kan worden is Burp Suite's web vulnerability scanner. Ook kan handmatig hierop getest worden, waarbij bij elke inputveld op de applicatie een simpele Unique input krijgen. Hierbij moet geïdentificeerd worden op welke locaties de input returned wordt als HTTP response.

Dus om dit probleem tegen te gaan moet er gebruik worden gemaakt van een combinatie van het valideren, filteren, coderen en het gebruik maken van escape codes op de input van gebruikers. Ook is er een cheat sheet gemaakt door OWASP, waarop gedetailleerd wordt verteld over methodes die gebruikt kunnen worden om XSS tegen te gaan. Ook kan er gebruik worden gemaakt van frameworks

die automatisch XSS ontvluchten dankzij het design. Denk hierbij aan Ruby, Rails of React JS. Ik moet zelf even kijken hoe Vuejs hierin vaart, aangezien ik Vuejs gebruik voor mijn applicatie. Ook moet er gebruik worden gemaakt van veilige headers.[25][[29][30][31][47]

Insecure Deserialisation

Insecure deserialization is a vulnerability in which an untrusted or unknown data is used to either inflict a denial of service attack (DoS attack), execute code, bypass authentication or further abuse the logic behind an application. Serialization is the process that converts an object to a format that can later be restored. Deserialization is the opposing process which takes data from a file, stream or network and rebuilds it into an object.

Serialized objects can be structured in text such as JSON, XML or YAML. Serialization and deserialization are safe, common processes in web applications. However, an attacker can abuse the deserialization process if left insecure. An attacker can inject hostile serialized objects to a web app, where the victim's computer would initialize deserialization of the hostile data. The attacker can then change the angle of attack, making insecure deserialization the initial entry point to a victim's computer.

Wat is insecure deserialisation?

Serialisatie is het proces waarbij een object vertaald wordt in een formaat dat opgeslagen kan worden op een schijf (als een bestand), of verzonden kan worden via streams of een netwerk. Het kan binaire of gestructureerde tekst zijn zoals XML, JSON, etc. Andersom bestaat er ook deserialisatie. Hierbij wordt het principe omgedraaid. Geserialiseerde data uit een bestand, stream of netwerk kan omgezet worden in een object. Een aanvaller kan hier gebruik van maken door de inhoud van wat er gedeserialiseerd wordt aan te passen. Hierdoor kunnen ze externe code uitvoeren via dat object/bestand (RCE, remote code execution). Het serialiseren van data wordt meestal gebruikt, wanneer er gebruik wordt gemaakt van databases, cache servers, bestands systemen, HTTP cookies, API authenticatie tokens, web services, enzovoorts.

Hoe komt dit tot stand?

Het komt dus tot stand wanneer een applicatie schadelijke en onbetrouwbare data deserialiseerd van een gevaarlijke bron.

CVE-2019-6503	Affects Chatopera, a Java app. Deserialization issue leads to remote code execution
CVE-2019-10068	Remote code execution in .NET app Kentico. One of the most recent vulnerabilities.
CVE-2018-7489	Remote code execution in systems that include Java Jackson XML functionality, similar to the example we provide below.
CVE-2018-6496, CVE-2018-6497	Unsafe deserialization leading to cross-site request forgery.
CVE-2018-19362	Can prevent normal operation of JBoss due to a XML Jackson vulnerability.
CVE-2018-1851	String formatting issues in Microsoft systems allows remote execution.
CVE-2017-9805	Related to Struts handling of XML deserialization, leads to remote execution. Very similar to the example described below.

Figuur 7: Recente incidenten, waarbij insecure deserialization voor zijn gekomen.[34]

Zijn er specifieke acties nodig voor mijn applicatie?

- Check bij deserializations of de data op een goede manier verwerkt wordt als input van een gebruiker in plaats van vertrouwde interne data.
- Check bij deserializations of de data, data is die je verwacht voordat het gebruikt wordt.
- Het gebruiken van een monitoring tool kan ook hierbij helpen. Deserializations kunnen hierdoor gemonitord worden. Wanneer er een probleem voorkomt, worden notificaties verstuurd.
- Het regelmatig uitvoeren van security scans. (CI/CD zou denk ik hierbij toegevoegd kunnen worden)

Ook zijn er een aantal mogelijkheden om dit soort type aanvallen te voorkomen:

- Het monitoren van deserialization processen.
- Het encrypten van serialization processen.
- Het niet accepteren van geserializeerde objecten van onbekende of onbetrouwbare bronnen.
- Het uitvoeren van deserialization code met gelimiteerde toegang tot de applicatie.
- Het gebruiken van een firewall. Een firewall kan namelijk onveilige deserialisation detecteren.

Het niet accepteren van geserialiseerde objecten van onbetrouwbare bronnen is één van de oplossingen om dit probleem tegen te gaan. Ook moet er niet gebruik worden gemaakt van serialisatie mediums die alleen primitieve data types toestaan. Om te weerhouden dat gevaarlijke objecten gecreëerd worden of om data wijzigingen tegen te gaan, kan er gebruik worden gemaakt van ‘integrity checks’, zoals digitale handtekeningen. Het isoleren van deserialisatie processen in een omgeving waar de privileges beperkt zijn wanneer mogelijk, is ook een manier om dit probleem tegen te gaan. Het verbieden of het monitoren van in- en uitgaande netwerk connecties van containers of servers waar deserialisatie wordt gebruikt.[25][32][33][34][35][48]

Het gebruiken van Componenten met bekende kwetsbaarheden

Wat is het precies?

In de titel staat eigenlijk precies wat het probleem deze categorie. Het komt dus voor dat applicaties gebruik maken van componenten, waarvan het bekend is dat het kwetsbaarheden bevat.

Hoe komt dit tot stand?

Wanneer developers geen kennis hebben over de versies van componenten en de informatie die ze bevatten en wanneer developers niet weten welke componenten er gebruikt worden door een applicatie, kan dit probleem voorkomen. Natuurlijk komt dit ook voor wanneer bepaalde software elementen in een applicatie kwetsbaarheden bevatten, niet ondersteund zijn of verouderd zijn. Kortom moet een developer kennis hebben over de componenten die zijn/haar applicatie bevat. Deze moeten up to date gehouden worden.

Zijn er specifieke acties nodig voor mijn applicatie?

Aangezien ik zowat continu mijn componenten check en up to date hou hoef ik niet veel te doen om dit probleem te voorkomen. Wel moet ik misschien nog even controleren of de gebruikte componenten geen kwetsbaarheden bevatten. [25][36][37][38][39]

Insufficient Logging & Monitoring

Wat is het precies?

Ook in deze categorie wordt er al in de naam weergegeven wat het probleem kan zijn. Een goede cyber beveiligde applicatie moet een effectieve SOC (security operation center) hebben. Hierbij wordt de applicatie krachtig en continu gelogd en gemonitord, zodat de gezondheid van de netwerk geëvalueerd wordt. Hierdoor kunnen potentiele bedreigingen op tijd geïdentificeerd worden.

Hoe komt dit tot stand?

- Wanneer belangrijke functionaliteiten zoals het inloggen, login pogingen en significante gegevens overdracht niet worden gelogd/gemonitord.
- Wanneer waarschuwingen en errors slechte of geen berichten genereren.
- Wanneer logs alleen lokaal worden opgeslagen.

Zijn er specifieke acties nodig voor mijn applicatie?

Zoals hierboven vermeld moet een applicatie een SOC hebben, waarbij de applicatie gelogd en gemonitord wordt. Ik maak gebruik van Prometheus en Grafana om de cluster te monitoren. Ik kan hierbij ook code in de backend schrijven om extra queries te maken voor Prometheus.

Ook maak ik gebruik van Ocelot Gateway. Ocelot bevat een logging interface die hiervoor gebruikt kan worden. Je moet wel oppassen dat het loggen via een console de performance van de applicatie kan beïnvloeden. ASP.NET core biedt haar gebruikers ook aan om Application Insights te gebruiken. Hiermee kan de applicatie gelogd worden. Hoe dit precies werkt moet nog uitgezocht worden.[25][40][41][42][49]

Bronnen

[1] *SQL injection Valve*. Geraadpleegd van:
[2] *Valve bug generated cd keys*. Geraadpleegd van: https://www.pcmag.com/news/valve-pays-hacker-20k-for-bug-that-generated-cd-keys
[3] *Wikipedia SQL injection*. Geraadpleegd van: https://en.wikipedia.org/wiki/SQL_injection
[4] *PortSwigger SQL injection.* Geraadpleegd van: https://portswigger.net/web-security/sql-injection
[5] *OWASP SQL injection.* Geraadpleegd van: https://owasp.org/www-project-top-ten/OWASP_Top_Ten_2017/Top_10-2017_A1-Injection
[6] *SQL injection and how to prevent it?* Geraadpleegd van: https://www.baeldung.com/sql-injection
[7] *How to fix SQL injection: JPA.* Geraadpleegd van: https://software-security.sans.org/developer-how-to/fix-sql-injection-in-java-persistence-api-jpa
[8] *OWASP Top 10: Real-World Examples (Part 1).* Geraadpleegd van: https://medium.com/@cxosmo/owasp-top-10-real-world-examples-part-1-a540c4ea2df5
[9] *Broken Authentication.* Geraadpleegd van: https://owasp.org/www-project-top-ten/OWASP_Top_Ten_2017/Top_10-2017_A2-Broken_Authentication
[10] *Broken Authentication OWASP Top 10.* Geraadpleegd van: https://hdivsecurity.com/owasp-broken-authentication
[11] *Norton Sensitive data exposure.* Geraadpleegd van: https://us.norton.com/internetsecurity-privacy-sensitive-data-exposure-how-its-different-from-data-breach.html
[12] *Hdiv Sensitive data exposure.* Geraadpleegd van: https://hdivsecurity.com/owasp-sensitive-data-exposure
[13] *Sitelock OWASP top 10, Sensitive data exposure.* Geraadpleegd van: https://www.sitelock.com/blog/owasp-top-10-sensitive-data-exposure/
[14] *ssllabs.com SSL Report: emal.hsbusiness.nl.* Geraadpleegd van: https://www.ssllabs.com/ssltest/analyze.html?d=emal.hsbusiness.nl&hideResults=on
[15] *XML External Entities (XXE) OWASP top 10.* Geraadpleegd van: https://owasp.org/www-project-top-ten/OWASP_Top_Ten_2017/Top_10-2017_A4-XML_External_Entities_(XXE)
[16] *XML External Entities (XXE) Processing.* Geraadpleegd van: https://owasp.org/www-community/vulnerabilities/XML_External_Entity_(XXE)_Processing
[17] *Hdiv, XML External Entities (XXE).* Geraadpleegd van: https://hdivsecurity.com/owasp-xml-external-entities-xxe
[18] *acunetix, What are XML External Entity (XXE) Attacks.* Geraadpleegd van: https://www.acunetix.com/blog/articles/xml-external-entity-xxe-vulnerabilities/
[19] *Wikipedia SOAP.* Geraadpleegd van: https://en.wikipedia.org/wiki/SOAP
[20] *Broken Acces Control OWASP Top 10.* Geraadpleegd van: https://owasp.org/www-project-top-ten/OWASP_Top_Ten_2017/Top_10-2017_A5-Broken_Access_Control
[21] *owasp.org, Broken Acces Control.* Geraadpleegd van: https://owasp.org/www-community/Broken_Access_Control
[22] *Packetlabs, Broken Acces Control: Hidden Exposure for Sensitive Data.* Geraadpleegd van: https://www.packetlabs.net/broken-access-control/
[23] *Hdiv, Broken Acces Control.* Geraadpleegd van: https://hdivsecurity.com/owasp-broken-access-control
[24] *Cors.* Geraadpleegd van: https://identityserver4.readthedocs.io/en/latest/topics/cors.html
[25] *OWASP Top 10: Real-World Examples (Part 2).* Geraadpleegd van: https://medium.com/@cxosmo/owasp-top-10-real-world-examples-part-2-3cdb3bebc976
[26] *OWASP Top 10, Security Misconfiguration.* Geraadpleegd van: https://owasp.org/www-project-top-ten/OWASP_Top_Ten_2017/Top_10-2017_A6-Security_Misconfiguration

[27] *Hdiv, Security Misconfiguration.* Geraadpleegd van: https://hdivsecurity.com/owasp-security-misconfiguration
[28] *Tutorialspoint, Security Misconfiguration.* Geraadpleegd van: https://www.tutorialspoint.com/security_testing/testing_security_misconfiguration.html
[29] *OWASP top 10, Cross-Site Scripting (XSS).* Geraadpleegd van: https://owasp.org/www-project-top-ten/OWASP_Top_Ten_2017/Top_10-2017_A7-Cross-Site_Scripting_(XSS)
[30] *acunetix, Cross-site Scripting (XSS).* Geraadpleegd van: https://www.acunetix.com/websitesecurity/cross-site-scripting/
[31] *Wikipedia, Cross-site scripting.* Geraadpleegd van: https://en.wikipedia.org/wiki/Cross-site_scripting
[32] *OWASP Top 10, Insecure Deserialization.* Geraadpleegd van: https://owasp.org/www-project-top-ten/OWASP_Top_Ten_2017/Top_10-2017_A8-Insecure_Deserialization
[33] *acunetix, Insecure Deserialization.* Geraadpleegd van: https://www.acunetix.com/websitesecurity/cross-site-scripting/
[34] *Hdiv, Insecure Deserialization: Attack examples, mitigation and prevention.* Geraadpleegd van: https://hdivsecurity.com/bornsecure/insecure-deserialization-attack-examples-mitigation-and-prevention/
[35] *acunetix, What is insecure deserialization?* Geraadpleegd van: https://www.acunetix.com/blog/articles/what-is-insecure-deserialization/
[36] *Hdiv, Components with known vulnerabilities.* Geraadpleegd van: https://hdivsecurity.com/docs/using-components-with-known-vulnerabilities/
[37] *Security boulevard, Using components with known vulnerabilities.* Geraadpleegd van: https://securityboulevard.com/2020/04/using-components-with-known-vulnerabilities-2/
[38] *OWASP A9: Using Components with known vulnerabilities -Why you can't ignore it.* Geraadpleegd van: https://resources.whitesourcesoftware.com/blog-whitesource/owasp-a9-using-components-with-known-vulnerabilities
[39] *OWASP-Using components with known vulnerabilities.* Geraadpleegd van: https://owasp.org/www-project-top-ten/OWASP_Top_Ten_2017/Top_10-2017_A9-Using_Components_with_Known_Vulnerabilities
[40] *OWASP TOP 10, Insufficient logging & monitoring.* Geraadpleegd van: https://owasp.org/www-project-top-ten/OWASP_Top_Ten_2017/Top_10-2017_A10-Insufficient_Logging%252526Monitoring
[41] *Hdiv, Insufficient Logging & Monitoring.* Geraadpleegd van: https://hdivsecurity.com/owasp-insufficient-logging-and-monitoring
[41] *ImmuniWeb, OWASP Top 10: Insufficient Logging & Monitoring Security Vulnerability Practical Overview.* Geraadpleegd van: https://www.immuniweb.com/blog/OWASP-insufficient-logging-and-monitoring.html
[42] *Medium, How to prevent insufficient logging and monitoring.* Geraadpleegd van: https://medium.com/@javan.rasokat/owasp-appsensor-logging-and-monitoring-2518712ee0fe

[43] M. (2017, 30 maart). *Security Considerations (Entity Framework) - ADO.NET*. Geraadpleegd op 20 juni 2021, van https://docs.microsoft.com/en-us/dotnet/framework/data/adonet/ef/security-considerations#prevent-sql-injection-attacks
[44] Torris, A. (z.d.). *SQL Injection attacks in Entity Framework Core 2.0*. Geraadpleegd op 20 juni 2021, van https://adrientorris.github.io/entity-framework-core/SQL-Injection-attacks-in-Entity-Framework-Core-2-0.html
[45] Kanjilal, J. (2020, 15 juni). *How to use Data Transfer Objects in ASP.NET Core 3.1*. Geraadpleegd op 20 juni 2021, van https://www.infoworld.com/article/3562271/how-to-use-data-transfer-objects-in-aspnet-core-31.html

[46] W. (2021e, 10 januari). *Testing XXE Vulnerabilities In .NET Core*. Geraadpleegd op 20 juni 2021, van https://dotnetcoretutorials.com/2021/01/11/testing-xxe-vulnerabilities-in-net-core/
[47] *What is cross-site scripting (XSS) and how to prevent it? | Web Security Academy*. (z.d.). Geraadpleegd op 23 juni 2021, van https://portswigger.net/web-security/cross-site-scripting
[48] Gillis, A. S. (2019, 9 mei). *Insecure deserialization*. Geraadpleegd op 23 juni 2021, van https://searchsecurity.techtarget.com/definition/insecure-deserialization
[49] Kã¶Bler, N. (z.d.). *Keycloak Events Logging | Niko KÃ¶bler - Keycloak Expert, Software-Architect & Trainer*. Geraadpleegd op 23 juni 2021, van https://www.n-k.de/2020/12/keycloak-events-logging.html