Risicoanalyse/Beveiligingsanalyse
Esther Adjei Mensah
Novi Hogeschool BV/Erik Mols

Samenvatting

Met dit document wil ik in kaart brengen wat de mogelijke risico's zijn op basis van de gegeven kennis voor het maken van deze applicatie. Het betekent niet dat ik voor elke beveiligingsrisico een oplossing heb of zal toepassen en dat zul je in de realiteit bij de maak van elk informatiesysteem moeten afwegen. Het betekent ook niet dat als ik een beveiligingsrisico hier niet behandel, dat de doorzetting van de maak van de applicatie niet door zou gaan.

Verder breng ik in kaart m.b.v. de bekende dreigingen (STRIDE, OWASP Top-10) welke risico's er bestaan, hoeveel impact deze zullen hebben en de mogelijke oplossingen die er bestaan.

Fig.1

#		Score	Weging
	Secure Software		**20%**
	De student heeft kennis en inzicht in een basisset aan maatregelen die genomen kunnen worden om de inbouw van kwetsbaarheden bij de bouw van software te voorkomen. De student kan afhankelijk van de situatie passende maatregelen selecteren om de basisveiligheid van software te maximaliseren.		
	De student toont aan mogelijke kwetsbaarheden in applicaties te kunnen herkennen, opsporen en tegenmaatregelen te treffen. Tevens kan de student software security toepassen in alle fasen van softwareontwikkeling.		

(Hogeschool NOVI B.V., z.d.)

Risicoanalyse/Beveiligingsanalyse

De risicoanalyse heeft tot doel per project zwakheden in de beveiliging of opzet van de software te vinden en te onderkennen. (Bron: Edhub, Grip op Secure Software, 4.3, alinea 1) Hiervoor gebruik ik de uitgangspunten van STRIDE als aanleiding. Ook tijdens het ontwikkelen van de software houd ik hier rekening mee.

Het eindresultaat van de risicoanalyse is een lijst met dreigingen die relevant worden geacht voor de IT-middelen binnen de scope en inzicht in de ernst van deze dreigingen. (Bron: Edhub, Grip op Secure Software, 4.3, alinea 15) Dit zijn de onderdelen met dreigingen die ik relevant acht voor de maak van deze software.

Lijst van dreigingen:

- Spoofing
- Tampering
- Repudiation
- Information disclosure
- Denial of Service (DoS)
- Elevation of privilege

Het doel van de risicoanalyse is het in een zo vroeg mogelijk stadium identificeren en begrijpen van risico's en het benoemen van mitigerende beveiligingseisen. (Bron: Edhub, Grip op Secure Software, 4.3, alinea 16) * (Hogeschool NOVI B.V., z.d.)

Inschatting van dreigingen via STRIDE
De analysemethode STRIDE is ontwikkeld door Microsoft. Dit is een 'threat assessment'. Er wordt een decompositie uitgevoerd, waarna per relevante component de gevoeligheid voor dreigingen wordt geanalyseerd. (Bron: Edhub, Grip op Secure Software, 4.3, alinea 9) (Hogeschool NOVI B.V., z.d.)

De naam STRIDE is een afkorting van de benamen van zes categorieën aan dreigingen, namelijk:

- Spoofing (misbruik van de gebruikersidentiteit, namelijk zich als een ander voordoen);
- Tampering (schending van de Integriteit);
- Repudiation (weerlegbaarheid);
- Information disclosure (schending van de privacy of het lekken van data);
- Denial of Service (DoS) (on-beschikbaarheid);
- Elevation of privilege (misbruik van bevoegdheden)

Op basis van deze dreigingen voer ik de risico/beveiligingsanalyse uit.

Kwalificering Risico's

De kwalitatieve risicoanalyse gaat uit van scenario's en situaties. Hierbij worden de kansen dat een dreiging werkelijkheid wordt ingeschat op basis van vuistregels en waarschijnlijkheid. De kwantitatieve risicoanalyse probeert op basis van risicowaardering te berekenen hoe groot de kans is dat een dreiging een incident wordt (Bron: pinkelephant.nl Risicoanalyse)

Voorbeeld:

Fig 2. Kwalificering van dreigingen, volgens Edhub

Risico (per dreiging)			
	Omvang van de schade door een dreiging		
Kans van optreden	Laag	Midden	Hoog
Laag	Laag	Laag	Midden
Midden	Laag	Midden	Hoog
Hoog	Midden	Hoog	Hoog

Afbeelding bron: (Edhub, Grip op Secure Software, 4.3

(Hogeschool NOVI B.V., z.d.)

De lijst met dreigingen:

- Spoofing (misbruik van de gebruikersidentiteit, namelijk zich als een ander voordoen);
- Tampering (schending van de Integriteit);
- Repudiation (weerlegbaarheid);
- Information disclosure (schending van de privacy of het lekken van data);
- Denial of Service (DoS) (on-beschikbaarheid);
- Elevation of privilege (misbruik van bevoegdheden)

De bovengenoemde dreigementen kunnen met behulp van authenticatie en autorisatie worden bestreden. Het Spring Security framework heeft hiervoor geschikte hulpmiddelen. (Marcobehler, 2020). Met uitzondering van Spoofing en Denial of service. In het geval van dit project wordt de applicatie alleen gedraaid op de local host van de beoordelaar, en/of via aan cloud-dienst. Als dit het geval is regelt de cloud-dienst de bescherming tegen Denial of service. Dit besef (DoS) is ook meegenomen in de kwalificering “kans van optreden.
Spofing kun je tegengaan door gebruik te maken van autorisatie door key-exchage.

Er bestaat een kans dat een hacker een van de bovengenoemde dreigementen kan uitvoeren. Hoe groot is dan de impact op gebruikers of het bedrijf?
Hieronder volgt dan de kwalificering voor de kans van optreden en de impact bij DIT project.
De kans van optreden is berekend op basis van de kennis die ik tijdens de bootcamp heb opgedaan. Het is een schatting. De impact is berekend op basis van de gevoeligheid van de persoonsgegevens en of het probleem schadeloos op te lossen valt.
Ik had graag de opdrachtgever (Arjen Wiersma) erbij willen betrekken om in dit document op te nemen wat de mogelijke impact voor hem (of een bedrijf) zou betekenen. Nu doe ik het in plaats daarvan met mijn gezond verstand omdat het niet even kan.

*Deze risicoanalyse is uiteraard net als de verschillende analyses en studies in deze map toegepast op de specifieke context van het te ontwikkelen systeem.

	Laag	Midden	Hoog
Hoog	Elevation of privilege	Tampering Spoofing	
Midden	Information Disclosure		
Laag	Denial of Service	Repudiation	

Impact →

Kans van optreden →

Spoofing(IP adressen)

Kans van optreden midden omdat: IP adressen unieke identificatienummers zinj waarmee een netwerkkaart kan worden geïdentificeerd. Maar ze kunnen wel door hackers worden nagebootst of vervalst.

Impact hoog omdat: Omdat een host in communicatie denkt te zijn met een andere host, terwijl die eigenlijk in communicatie is met de hacker “de man-in-the-middle”. Ook speelt er een rol in hoe nuttig of gevoelig de gevraagde informatie is. In dit geval gaat de informatie over de gebruiker en de verwerking van zijn demos. (Dit schat ik in als niet gevoelig)

Mogelijke oplossingen:

- Gebruik van authenticatie met key-exchange. (Bijv IP-SEC)
- Applicatie door een securityspecialist laten testen/meebouwen

Tampering

Kans van optreden midden omdat: Ik niet aandachtspunten van security tegen tampering weet. Ik kan namelijk niet met 100% zekerheid kan zeggen dat dit volledig gedekt kan worden.

Wel vertrouw ik erop dat je door middel van het goed configureren van security m.b.v. een sucurity-framework ver op weg komt.

Impact hoog omdat: Tampering is het opzettelijk manipuleren, vernietigen of veranderen van data. Dit kan overal in het informatiesysteem gebeuren en kan leiden tot een groot verlies.

In het geval van al-beoordeelde demo’s zou dit feitelijk niet erg zijn. Demo’s die zijn doorgestuurd naar Don Diablo (gebeurt via een automatisch gegenereerde email) bevatten data als de contactgegevens en de demo zelf op dat moment. Hieruit valt dus nog accurate informatie uit te halen voor de DJ(het einddoel). Alhoewel, de demo wordt waarschijnlijk door de applicatie zelf ge-served en loopt ook het risico om verloren te gaan.

Juiste informatie over doorgestuurde demos, en nog-te-beoordelen zijn het meest belangrijk. In het geval van de “nog-te-beoordelen” demo’s zou al deze informatie

En de bijgehouden ‘al beoordeelde demo’s’ kunnen in geval van vernietiging niet meer worden bijgehouden

Mogelijke oplossingen:

- Veilig bouwen van de applicatie.
- Applicatie door een security-specialist laten testen/meebouwen.

(Mogelijke oplossingen om de impact te verlagen bij voorkomen van tampering)

- Regelmatig back-ups van de database maken.
- Regelmatig back-ups van de audiobestanden maken. (Buiten de applicatie zodat die niet benaderd en gemanipuleerd kunnen worden vanuit de (kwetsbare) applicatie)

Repudiation

Kans van optreden laag omdat: De software aan de server-side gebouwd zal worden met allerlei validatie’s om een post met geldige en volledige gegevens (in een database) toe te staan maken.

Er speelt onder andere door de gehele applicatie authenticatie en autorisatie plaats van Spring Security, om bepaalde handelingen te tracken zoals: wie, wat heeft gedaan.

Bij de BO-side wordt bijvoorbeeld in de front-end gevisualiseerd wie welke demo heeft beoordeeld. En heeft de admin een overzicht van de belangrijkste gegevens uit de database.

De applicatie zal voorzien moeten van logging om gebeurtenissen bij te houden.
Impact midden omdat: Bij voorkomen, kan de informatie tot een graad niet meer betekenisvol zijn. Het wordt namelijk niet inzichtelijk wie wat wanneer heeft gedaan.
Er zal weer tijd ingestoken moeten worden om deze bug te fixen.
Mogelijke oplossingen: Software op een manier bouwen dat het altijd na te gaan is wie welke handelingen heeft verricht. En dat geen incomplete data in de database kan.

Information Disclosure

Kans van optreden midden omdat: Ik van plan ben om niet alle data ge-encrypt op te slaan in de database. Alleen de wachtwoorden wil ik encrypten.

Als er gevoelige on-ge-encrypte data in het database staat, is dat voor leden van het productieteam (ik) een probleem. Omdat er dan vertrouwde informatie wordt gezien.
In dit project wordt er geen vertrouwelijke informatie van gebruikers gevraagd.

Het optreden van Information disclosure door middel van SQL Injection is wel laag, mits er geen gebruik gemaakt wordt van ”Strings” met SQL query’s in de code. (Baeldung, 2020). Ook hier geld dat de opgeslagen data niet gevoelig is, en de wachtwoorden moeten worden ge-encrypt.

Voor dit project ben ik van plan een in-memory database te gebruiken. Dus hoef ik niet te denken aan een veilige fysieke locatie voor de database.
Impact midden omdat: De opgeslagen informatie van de gebruikers zijn niet gevoelig of waardevol. Gegevens als: geboortedatum, geboorteplaats, woonplaats, bankgegevens, BSN of soortgelijk gevoelige data worden niet gevraagd. Waarmee het stelen van een identiteit niet succesvol kan worden verricht.

Mocht een hacker bij deze informatie komen heeft hij toegang tot de voornaam, achternaam, artiesten-naam, het emailadres en de geuploadde demos. Over deze demo's beschikt natuurlijk auteursrechten. Maar blijven "eigendom van Hexagon" zoals de opdrachtgever heeft aangegeven.

Wachtwoorden zijn ge-encrypt.

Een extra maatregel om de impact naar 'laag' te veranderen zou zijn om het emailadres, de achternaam én het adres naar de demo ge-encrypt op te slaan in de database. Maar zonder deze extra maatregel is de impact 'midden'. De kans is klein dat dit in deze versie van het project gaat worden toegepast.

Een ander ding om rekening mee te houden is de media. Als bekend wordt dat er een informatie-lek is kan dit mogelijk schade verrichten aan de naam van het bedrijf. Maar dit hangt er vanaf hoe belangrijk deze opgeslagen data is.
Mogelijke oplossingen:
- Code schrijven om SQL injectie te voorkomen
- Gevoelige data encrypten.
- Mocht alle informatie gevoelig worden gevonden: dan alle data encrypten (Dit kost wel veel productietijd)

DoS
Kans van optreden laag omdat: Er kunnen twee scenario's zijn:
A. Het project dient alleen gedraaid te worden op de localhost van de beoordelaar
B. Het project wordt gehost op AWS.

In het eerste geval zou dat betekenen dat de 'hacker' zijn eigen systeem (die als server dient) tijdelijk buiten gebruik zet. Ik zie de kans van optreden daarom als laag ondanks dat ik gezien de tijd en opdracht geen custom oplossing tegen DoS ga hanteren.

Voor beginners is het handig om kwesties als DoS aan de Cloud over te laten. Cloud-gebaseerde applicaties kunnen schadelijk of verdacht verkeer absorberen voordat het zijn doelwit bereikt. (Dobran, 2018)
Impact laag omdat: In het eerste geval zou dat betekenen dat de 'hacker' zijn eigen systeem (die dient als server) tijdelijk buiten gebruik zet.

In het tweede geval zou dat betekenen dat het systeem tijdelijk niet meer te bereiken is. Als het goed is kan Amazon dit voorkomen, maar ook vrijwel direct oplossen. Voor de medewerkers en gebruikers wordt het dan een kwestie van wachten, maar er gaat geen informatie verloren.
Mogelijke oplossingen:
- Gebruik maken van cloud-oplossingen
- Veilige netwerk-architectuur ontwerpen en toepassen
- Netwerk security hanteren
- Denial of service response plan
(Dobran, 2018)

Elevation of priviledge
Kans van optreden laag omdat: Omdat je als developer met een security framework (bijv. Spring) goed kan aangeven welke restricties bepaalde rollen hebben. Het Spring Security Framework is bij

goed gebruik betrouwbaar. (Huysamen, 2011)
Impact hoog omdat: Omdat er schade in alle vormen kan ontstaan als een ongeautoriseerde gebruiker toegang krijgt tot functies die niet voor dat type gebruiker zijn bedoeld. In het ergste geval kun je al je data kwijt-raken of kunne andere gebruikers de storende effecten daarvan merken
Mogelijke oplossingen:
- Applicatie tegen kwetsbaarheden bouwen en op de juiste manier gebruik maken van het security-framework
- toepasssen van onder andere tests
- Regelmatige back-ups van de database maken

OWASP

Ook al werd er GÉÉN uitleg gegeven over OWASP in Edhub en is dit woord niet één keer benoemd in de gegeven leerlijn “Secure Software” (ga maar na https://edhub.novi.nl/study/learnpaths/236/courses), heb ik toch heb ik het initiatief genomen om even op te zoeken wat het is, want ondanks dat dit nooit is behandeld, ben ik hier toch op beoordeeld voor het ontbreken van dit stukje informatie. Dus bij deze.

OWASP staat voor Open Web Application Security Project en is een open source-project rond computerbeveiliging. Individuen, scholen en bedrijven delen via dit platform informatie en technieken. (Wikipedia, 2019)

Er bestaat een zogeheten OWASP top-10 die erg hot blijkt te zijn. Eind 2017 heeft het Owasp een nieuwe top 10 gepubliceerd van de grootste en meest voorkomende risico's binnen webapplicaties. Deze top 10 is een belangrijk hulpmiddel voor webontwikkelaars om kwetsbaarheden te identificeren en te voorkomen.

Het eerste risico is dezelfde als in 2013 en 2010: **injectie**. Injectie is een algemene term voor het risico dat ontstaat wanneer commando's in een instructietaal (bijvoorbeeld SQL, Bash of LDAP) onveilig gemixt worden met gebruikersinvoer. Wanneer een aanvaller tekens met een speciale betekenis (‘metakarakters’) invoert, kan deze vervolgens de betekenis van dit commando aanpassen en vaak ernstige schade aanrichten. Het bekendste voorbeeld van zo'n aanval is SQL-injectie, waarbij een aanvaller eigen queries kan laten uitvoeren en daarmee de inhoud van de database kan uitlezen of wijzigen. Hoewel SQL-injectie al bijna twintig jaar een groot probleem is, komen we het tijdens beveiligingsonderzoeken steeds minder vaak tegen. Ontwikkelaars zijn tegenwoordig goed op de hoogte van deze beruchte aanval en web-frameworks zijn er standaard op ingericht om dit risico te vermijden. Dat geldt echter niet persé voor NoSQL-injectie.

Het tweede risico is **authenticatiefouten**: voorkomen dat aanvallers andermans account overnemen blijft in het algemeen ingewikkeld. Trends zoals stateless sessiebeheer, single-sign on en authenticatiemethoden die ook geschikt zijn voor apps, introduceren nieuwe risico's, waar ontwikkelaars mogelijk minder bekend mee zijn.

Het derde risico is het **lekken van gevoelige data**, bijvoorbeeld **door een gebrek aan versleuteling of (sterke) gebruikersauthenticatie**. Ook worden tegenwoordig gevoelige gegevens nog vaak, per

ongeluk, publiek toegankelijk gemaakt. Een pluspunt is wel dat het gebruik van transportversleuteling, door middel van https, enorm is toegenomen.

Het vierde risico op de lijst is: **het toestaan van externe xml-entiteiten**. Dit is een oud probleem dat aanvallers toestaat bestanden op de server (en het netwerk achter de firewall) uit te lezen door een, bij nader inzien niet zo handige, feature uit te buiten in (verouderde) software die xml-berichten uitleest.

Het vijfde risico is **falende toegangscontroles**. Een klassiek probleem dat bij moderne webtechnologieën ook erg vaak voorkomt. Het ontbreken van autorisatiecontroles is ook een typische kwetsbaarheid dat lastig is om op te testen, of om te vinden met automatische tools.

Het zesde risico luidt: **configuratiefouten**. Software is vaak standaard ingesteld zodat het makkelijk in gebruik te nemen is, maar niet persé op een manier die veilig is. Er wordt aangeraden om ervoor te zorgen dat de belangrijke security-functies altijd aan staan, er geen gebruik wordt gemaakt van standaardwachtwoorden en debug-functies uit te zetten op de productieomgeving.

Het zevende risico is **cross-site-scripting**, het overnemen van andermans browsersessie door scripts in een webpagina te injecteren, is flink gedegradeerd: van nummer drie in 2013 tot nummer zeven in de nieuwste editie. Deze kwetsbaarheid kan nog steeds catastrofaal zijn, maar is vanwege mitigaties van web-frameworks moeilijker uit te buiten dan voorheen. Dit soort beveiligingsmaatregelen worden echter vaak expliciet uitgeschakeld wanneer ze in de weg zitten, en ze beschermen ook niet tegen alle mogelijke vormen van cross-site-scripting.

De top 10 wordt afgesloten met **respectievelijk onveilige deserialisatie**: Het mogelijk maken voor een aanvaller om een getransporteerd programmeertaal-object aan te passen, waardoor de server overgenomen kan worden.
Het **gebruik van software met bekende kwetsbaarheden** (tegenwoordig gemakkelijk uit te buiten met hacker-tools waarmee op verouderde software gescand kan worden).
En **onvoldoende logging en monitoring**. Het laatste risico erkent dat je niet kunt aannemen dat preventieve maatregelen afdoende zijn: het is ook belangrijk om aanvalspogingen te kunnen detecteren, adequaat erop te kunnen reageren, en een plan te hebben voor wanneer het misgaat.

(Tervoort, 2018)

Beveiligingsmaatregelen

Dit zijn de beveiligingsmaatregelen die ik verwacht te gebruiken in de applicatie. Dit baseer ik op de opgedane kennis die ik heb gedaan bij het bouwen van andere projecten:

- Bescherming tegen csrf
- Input validatie in max aantal, elk naar wat de developer verstandig lijkt te zijn om de applicatie niet te laten crashen.
- Rollen en Autorisatie
- Authenticatie

- Sessiebeheer
- Bescherming tegen spoofing van een emailadres door een bevestigings-email met code te sturen.
- Bescherming tegen crossite scripting

Conclusie

Belangrijkste conclusies zijn het volgende:
Spoofing, Tamering en Elevation of priviledge lijken de grootste uitdagingen om tegen te gaan bij de maak van de applicatie.
Verder heb ik de toegevoegde informatie over de O-wasp genoteerd.

Dit document was een combinatie van de stof in Edhub over het maken van de risico-analyse en extra informatie over Owasp. Ik hoop aan de hand van (o.a) dit document te hebben aangetoond dat ik kennis en inzicht heb in een basisset aan maatregelen die genomen kunnen worden om de inbouw van kwetsbaarheden bij de bouw van software te voorkomen.

Tijdens de bouw van de applicatie schrijf ik een verantwoordingsdocument met daarin de toegepaste security-maatregelen binnen alle fasen van de softwareontwikkeling.
Afhankelijk van de situatie selecteer ik passende maatregelen, en toon ik aan de mogelijke kwetsbaarheden te kunnen herkennen en opsporen.

#		Score	Weging
	Secure Software		**20%**
	De student heeft kennis en inzicht in een basisset aan maatregelen die genomen kunnen worden om de inbouw van kwetsbaarheden bij de bouw van software te voorkomen. De student kan afhankelijk van de situatie passende maatregelen selecteren om de basisveiligheid van software te maximaliseren.		
	De student toont aan mogelijke kwetsbaarheden in applicaties te kunnen herkennen, opsporen en tegenmaatregelen te treffen. Tevens kan de student software security toepassen in alle fasen van softwareontwikkeling.		

<u>Einde Risico-analyse</u>

Bronnenlijst

Baeldung. (2020, April 17). *SQL Injection and how to prevent it*. Opgehaald van baeldung.com: https://www.baeldung.com/sql-injection

Dobran, B. (2018, 09 10). *7 Tactics To Prevent DDoS Attacks & Keep Your Website Safe*. Retrieved from phoenixnap.com: https://phoenixnap.com/blog/prevent-ddos-attacks

Hogeschool NOVI B.V. (z.d.). *Eindopdracht Full Stack Developer Beoordelingscriteria.* Opgehaald van edhub.novi.nl: https://edhub.novi.nl/study/learnpaths/252/documents

Hogeschool NOVI B.V. (z.d.). *Proffessionele ontwikkeling.* Opgehaald van edhub.novi.nl: https://edhub.novi.nl/study/learnpaths/247/documents

Hogeschool NOVI B.V. (z.d.). *Risicoanalyse.* Opgehaald van edhub.novi.nl: https://edhub.novi.nl/study/courses/291/content/6725

Huysamen, N. (2011, 03 24). *How Secure is Spring Security?* Retrieved from Stackoverflow: https://stackoverflow.com/questions/5417757/how-secure-is-spring-security

Marcobehler. (2020, April 11). *Spring Security: Authentication and Authorization In-Depth*. Opgehaald van marcobehler.com: https://www.marcobehler.com/guides/spring-security

Tervoort, T. (2018, 01 19). *Owasp top 10 van web-risico's is vernieuwd*. Opgehaald van computable.nl: https://www.computable.nl/artikel/expertverslag/security/6280786/4573232/owasp-top-10-van-web-risicos-is-vernieuwd.html

Wikipedia. (2019, 11 2). *OWASP*. Opgehaald van Wikipedia.org: https://nl.wikipedia.org/wiki/OWASP