

# **Riippuvuuksien ja haavoittuvuuksien hallinta projektitasolla**

Ammattikorkeakoulututkinnon opinnäytetyö

Tieto- ja viestintätekniikka, insinööri (AMK)

Kevät, 2024

Petri Malmström

Tieto- ja viestintätekniikka

Tekijä Petri Malmström

Työn nimi Riippuvuuksien ja haavoittuvuuksien hallinta projektitasolla

Ohjaaja Joni Järvenpää

Tiivistelmä

Vuosi 2024

---

Opinnäytetyön tavoite on informoida ja opettaa parhaita käytäntöjä riippuvuuksien sekä haavoittuvuuksien hallinnan prosessiin. Taustatiedot ja idea työlle ovat tulleet ohjelmistokehittäjän työn kautta.

Työssä esitellään yleisellä tasolla riippuvuudet sekä haavoittuvuudet, niiden elinkaaren sekä mahdolliset vaikutukset ohjelmistoihin. Ohjeellisessa selvitystyössä paneudutaan käytäntöjen kautta hallinnan prosessiin sekä annetaan ohjeita, jolla prosessia voidaan optimoida ja kustomoida omalle organisaatiolle sopivaksi.

Selvitystyön teoriapohjana käytetään ohjelmistokehityksen parissa toimivan yrityksen informaatiota riippuvuuksien ja haavoittuvuuksien hallinnasta. Opinnäytetyön perusteella voidaan todeta riippuvuuksien ja haavoittuvuuksien hallinta on suositeltavaa suorittaa jatkuvana prosessina. Työssä johtopäätöksi muodostuu siis jatkuvan prosessin hyödyntäminen ja optimaalisen päivitysfrekvenssin löytäminen.

Avainsanat Riippuvuuksienhallinta, haavoittuvuuksienhallinta, prosessi

Sivut 25 sivua

Information and Communication Technology

Author Petri Malmström

Subject Management of Dependencies and Vulnerabilities at the Project Level

Supervisor Joni Järvenpää

Abstract

Year 2024

---

The objective of this thesis is to inform about and teach the best practices for the process of managing dependencies and vulnerabilities. The background information and the idea for the work have been derived from the experience gained in the role of a software developer.

In the work presented, an overview of dependencies and vulnerabilities, their lifecycle, and potential impacts on software is provided. In the advisory investigation, practices are delved into examine management processes, and guidelines are provided for optimizing and customizing the process to suit an organization's specific needs.

The theoretical basis of the advisory investigation uses information from a company operating in software development regarding the management of dependencies and vulnerabilities. The research led me to conclude that it is advisable to manage dependencies and vulnerabilities as a continuous process. Thus, the conclusions drawn from the study emphasize the utilization of a continuous process and the identification of an optimal update frequency.

Keywords Dependency management, vulnerability management, process

Pages 25 pages

# Sisällys

1	Johdanto .....	1
1.1	Taustatietoja .....	1
1.2	Työn rakenne.....	1
2	Ohjelmistojen riippuvuuksien ja haavoittuvuuksien hallinta: nykytila ja haasteet.....	2
2.1	Riippuvuudet ja haavoittuvuudet ohjelmistoissa .....	4
2.2	Hallinnan merkitys.....	5
2.3	Aikaisempaa tutkimustietoa .....	6
3	Työn menetelmät .....	6
4	Riippuvuuksien hallinta .....	7
4.1	Tunnistus .....	7
4.2	Arviointi.....	8
4.3	Päivittäminen ja hallinta .....	8
4.4	Dokumentointi.....	9
5	Haavoittuvuuksien hallinta .....	9
5.1	Tunnistus .....	10
5.2	Arviointi.....	10
5.3	Korjaaminen.....	10
5.4	Seuranta ja raportointi.....	11
5.5	Hallinnan prosessi (kokonaisuutena).....	11
5.5.1	Tunnistus.....	11
5.5.2	Arviointi .....	12
5.5.3	Korjaaminen .....	12
5.5.4	Seuranta ja raportointi .....	12
6	Opinnäytetyön aikaiset työkäytännöt.....	12
6.1	Käytetyt ohjelmistot.....	13
6.2	Saavutetut tulokset ja oppimiskokemukset.....	15
7	Tulokset ja yleinen pohdinta.....	16
7.1	Yhteenveto tuloksista.....	16
7.2	Tulkinta ja analyysi.....	17
7.3	Jatkotutkimusta varten .....	17
8	Yhteenveto ja johtopäätökset.....	19
8.1	Tärkeimmät havainnot .....	19
8.2	Pääasialliset johtopäätökset.....	20
8.3	Työn merkitys, soveltaminen sekä validointi.....	20

Lähteet .....	23
---------------	----

## **Kuvat, taulukot ja kaavat**

Kuva 1. Versioristiriidat kuvattuna Maven:ssa. ....	3
Kuva 2. Semanttinen versiointi.....	4
Kuva 3. Kirjastojen riippuvuudet toisistaan.....	4
Kuva 4. Aliriippuvuussuhde kuvattuna Maven:ssa. ....	5
Kuva 5. Projektinäkymä Dependency Track:n sisällä. ....	14
Kuva 6. Maven-valikko.....	14
Kuva 7. Sivuston aloitusnäkymä. ....	15
Kuva 8. Jatkotutkimuksen prosessikuvaus.....	18

# 1 Johdanto

Opinnäytetyössä tarkastellaan riippuvuuksien ja haavoittuvuuksien hallinnan tärkeyttä projektitasolla. Aiheen merkityksellisyys on noussut selvästi kyberturvallisuusvaatimusten sekä lisääntyneen verkkovakoilun seurauksena (Jack, 2023). Tunnetut haavoittuvuudet ovat yleisimpiä hyökkäysvektoreita ohjelmistoissa, jolloin niiden seuraaminen ja korjaaminen ovat ensisijaisen tärkeitä tavoiteltaessa toimivaa ja tietoturvallista ohjelmistokokonaisuutta (Soare, 2023). Myös ohjelmistokehityksen nousseet standardit ja vaatimukset ovat omalta osaltaan pakottaneet suunnittelemaan ja toteuttamaan parempia ohjelmistoja (Krysik, 2023). Työssä otetaan myös huomioon koronavirusepidemian tuomat etätyövaatimukset ja käytänteet, jotka ovat osaltaan myös ajaneet ihmiset käyttämään ja vaatimaan ohjelmilta enemmän (Juillet, 2022).

Opinnäytetyö esittelee keinoja ja käytäntöjä parempaan riippuvuuksien ja haavoittuvuuksien hallintaan. Työ pyrkii samalla selvittämään, mitä tapahtuisi, jos riippuvuuksien ja haavoittuvuuksien hallinta laiminlyötäisiin, sekä aiheutuuko niiden määrästä este menestykselle hallinnalle. Aihealuetta lähestytään ohjelmistokehittäjän näkökulmasta, ja se sopii sellaisenaan alalla työskenteleville, alalle haluaville sekä yleiseksi ohjepohjaksi. Työn tavoite on lisätä tietoisuutta ja toimia ohjeena lukijalleen. Kehittyvät käytänteet ja toimintatavat takaavat jatkossakin parempi laatuja ohjelmia ja ohjelmistoja, joista hyötyvät kaikki käyttäjät.

## 1.1 Taustatietoja

Opinnäytetyön aiheen valintaan on vaikuttanut niin korona-epidemia kuin Ukrainan sota. Kuten Huntley (2023) artikkelissaan toteaa, on Venäjä on lisännyt kyberturvallisuusuhkien määrää Ukrainassa ja NATO-maissa (Huntley, 2023), ja tämä on tehnyt aihepiirin valinnasta erittäin ajankohtaisen.

## 1.2 Työn rakenne

Opinnäytetyön alkupuolella tutustutaan teoriaan ja pyritään luomaan tarvittava tietopohja lukijalle, jotta tutkimusaihe voitaisiin tarvittavalla tarkkuudella käydä läpi. Teoriaosassa annetaan yleiskuva siitä, mitä riippuvuuksien ja haavoittuvuuksien hallinta on. Teoriaosa antaa pohjan tuleville käytännöille ja menetelmille, jolloin lukijalla on tietyn tasoinen ymmärrys aiheesta. Käytäntöosassa tutkitaan mitä ja miten hallinta voisi toimia, ja tutkitaan, voidaanko nykytilanteen haasteita ratkaista käytännöillä ja koulutuksella. Osiossa käydään myös läpi

tarkemmin Java-projektin liittyvää riippuvuuksien hallintaa, sekä tutustutaan haavoittuvuuksien kartoittamiseen ja mitigointiin.

Opinnäytetyön loppuosiossa käydään löydettyjä tuloksia läpi, sekä analysoidaan saavutettuja tuloksia. Työn lopussa arvioidaan myös, miten työ saavutti tavoitteensa, sekä vastasiko tutkimuskysymyksiinsä luontevasti.

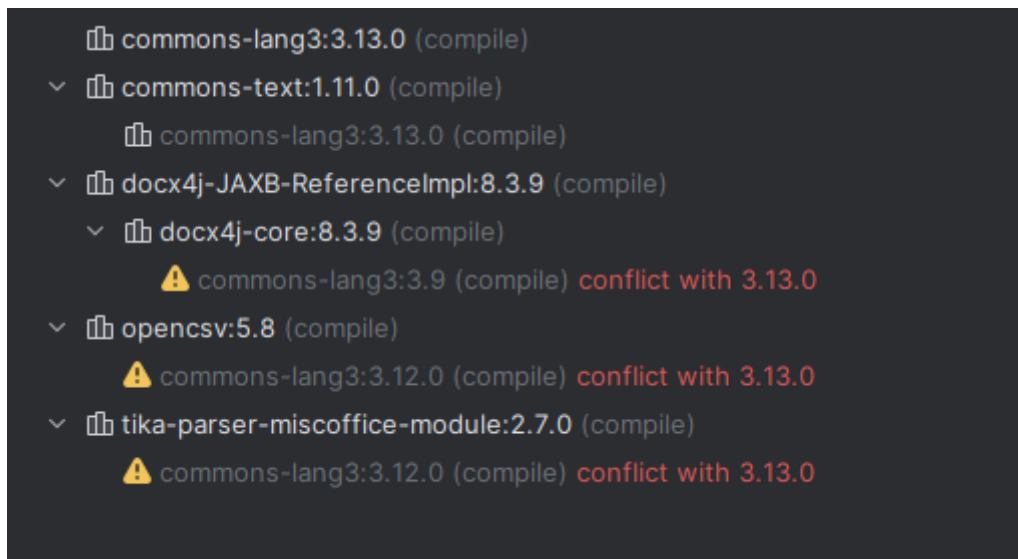
## **2 Ohjelmistojen riippuvuuksien ja haavoittuvuuksien hallinta: nykytila ja haasteet**

Ohjelmistoriippuvuudet ja niiden käyttö on nopeuttanut sekä helpottanut isojenkin ohjelmistokokonaisuuksien luontia. Kehitettävään ohjelmistoon pystytään tuomaan ulkopuolisina kirjastoina tarvittavaa koodia ilman, että se pitäisi tuottaa itse alusta lähtien. Tämä kasvava riippuvuuksien käyttö johtaa usein siihen, että ne jäävät käyttöönoton jälkeen ilman valvontaa ja hallintaa, jolloin niistä pääsee ajan myötä kehittymään tietoturvariskejä. Valvonta ja hallinta ovatkin tärkeitä osa-alueita, kun halutaan varmistaa ohjelmiston tietoturvallisuus. Ongelmaksi näiden osa-alueiden kanssa nousee niiden vaatima resurssien määrä. (Perälä, 2023, s. 6)

Valvonnan ja hallinnan luomaa resurssiongelmaa on lähdetty ratkaisemaan luomalla erilaisia automaatiotyökaluja kehittäjien avuksi. Automaatiotyökaluja löytyy moneen eri tarkoitukseen: esimerkiksi pelkästään skannaukseen keskittyvät, jolloin projekti skannataan ja etsitään tunnettuja haavoittuvuuksia ja kirjastopäivityksiä. Riippuen automaatiotyökalun asetuksista, voi se pelkästään skannata ja ehdottaa päivityksiä, tai jopa asentaa niitä automaattisesti (Perälä, 2023, s. 13–15).

Monesti ongelmia aiheuttavat eritoten eriaisteiset versionumeroiden ristiriitaisuudet ja yhteensopivuusongelmat, joita kuvataan kuvassa 1. Tällä siis yksinkertaistettuna tarkoitetaan esimerkiksi tilannetta, jolloin kirjasto A käyttää aliriippuvuutenaan uudempaa versiota kirjasto C:sta, josta kirjasto B käyttää vanhempaa. Edellä kuvattu tilanne aiheuttaa versioiden yhteensopivuusongelman, mikä joudutaan ratkaisemaan. Kehittäjiltä kuluu tähän ylimääräiseen työvaiheeseen paljon resursseja, koska ongelmatilanteen ratkaisu sisältää testausta ja kokeilevaa kehitystyötä. (Garousi, 2023)

Kuva 1. Versioristiriidat kuvattuna Maven:ssä.



Semanttinen versiointi on helpottanut tilannetta myös riippuvuuksien, haavoittuvuuksien sekä versionhallinnan kanssa. Tämä monille tuttu tapa ilmoittaa versionumero sisältää huomattavan määrän informaatiota, jota osaava kehittäjä osaa työssään hyödyntää. Semanttinen versionumero koostuu siis kolmesta luvusta, joita pisteellä erotetaan toisistaan, esimerkkinä 1.0.1. Näistä ensimmäinen luku kertoo, muutoksen olevan suuri sekä yleensä taaksepäin sopimaton. Ensimmäistä numeroa kutsutaankin 'Major':ksi. Tämän osan muuttuessa tunnistetaan yleensä tarve tehdä merkittäviä muutoksia ohjelmistokoodiin, jotta saadaan siitä toimiva uudemman version kanssa. Toinen luku kertoo muutoksen olevan vähäinen. Tällainen versionmuutos yleensä onnistuu ilman suurempia muutoksia kooditasolla. Toisesta lukemasta käytetään nimitystä 'Minor'. Kolmas ja viimeinen luku yleensä kertoo päivityksen sisältävän, jonkun pienen bugikorjauksen tai vastaavan paikkauksen. Tätä lukua kutsutaankin kuvaavasti 'Patch':ksi eli suomennettuna paikaksi. Kuvassa 2 nimikkeet niiden oikeilla paikoillaan. (Tremplin-Numérique, 2022)





resurssienkaan allokointi päivityksille ei aina takaa toimivaa ja ajantasaista ohjelmistoa. Tarvitaan automaatiota, suunnitelmallisuutta sekä osaavia asiantuntijoita. (Wisenhoff, 2021)

Kun ohjelmistossa muodostuvia ongelmia lähdetään korjaamaan kirjastoilla, menetetään kyky ratkaista ongelma halutulla tavalla. Tämä voi ilmentyä esimerkiksi myöhemmissä vaiheissa kehitystä, jolloin sen korjaaminen onkin jo huomattavasti vaativampi prosessi. Ohjelmistoja suunniteltaessa onkin syytä tarkoin miettiä, miten muiden tuottamaa koodia voidaan järkevästi käyttää omassa projektissa. (Ashigbi, 2020)

Tilastojen valossa ohjelmistoissa käytetyt kirjastot luovat merkittävän määrän haavoittuvuuksista omilla aliriippuvuuksillaan. Yleensä haluttu kirjasto riippuvuus tuo mukanaan omia aliriippuvuuksiaan, joista yleensä muodostuu haavoittuvuuksia suoraan. Kuvassa 4 kuvattuna aliriippuvuussuhde. (Cimpanu, 2020)

Kuva 4. Aliriippuvuussuhde kuvattuna Maven:ssa.

```
[INFO] +- org.liquibase:liquibase-core:jar:4.21.1:compile
[INFO] | \- org.yaml:snakeyaml:jar:2.0:compile
```

## 2.2 Hallinnan merkitys

Riippuvuuksien ja haavoittuvuuksien hallinnalla tarkoitetaan tapaa tunnistaa, selvittää ja mitigoida riippuvuuksia ohjelmiston koodissa. Kyseistä toimintaa voi suorittaa manuaalisesti itse, tai käyttää kolmansien osapuolien tekemiä ohjelmia, jotka yleensä integroituvat CI/CD-automaatioon. (Murray, 2023)

Ulkoisten kirjastojen käyttö on noussut nykyohjelmistoissa suureen rooliin. Tällä säästetään merkittävästi kehityskuluissa sekä kehitykseen käytetyssä ajassa. Lisättyjen kirjastojen myötä kuitenkin ohjelmiston kompleksisuus nousee, ja sen myötä nousee tarve riippuvuuksien hallinnalle. Lisääntynyt kirjastojen määrä projektissa, lisää mahdollisia negatiivisia vaikutuksia, kuten koodiristiriitaisuuksia ja versioristiriitoja. Riippuvuuksien hallinta takaa hyvän ja toimivan koodin ilman suuria ongelmakohtia. (Blaisdell, 2022)

Hyvällä riippuvuuksien hallinnalla ja päivityskäytännöillä saavutetaan hyötyjä, jotka jäisivät muuten saamatta. Ohjelmistojen toimivuus ja suorituskyky pysyvät halutulla tasolla. Laadunvarmistus on helpompaa uudemmalla versiolla kirjastoista. Myös turvallisuus lisääntyy huomattavasti, sillä kirjastojen tietoturva aukot paikataan nopeasti ja turvallisesti. (Blaisdell, 2022)

## 2.3 Aikaisempaa tutkimustietoa

Vuonna 2021 julkaistussa tutkimuksessa painotetaan automaatiotyökalujen tärkeyttä riippuvuuksien ja haavoittuvuuksien hallinnassa. Tutkijoiden mukaan työkalujen käyttämät usein päivitettävät tietokannat, ovat niiden tehokkuuden tae. Tutkimuksessa huomattiin myös näiden työkalujen antavan yleistä metriikkaa projektista, mutta kirjoittajien mukaan sen tehokkuutta riippuvuuksien hallinnassa pitäisi tutkia ohjelmistokehittäjän näkökulmasta lisää. (Williams ym., 2019)

Ryhmä toteaa myös tutkimuksessaan, että aiemmat työt osoittavat, että ennen ohjelmistokehittäjien on pitänyt luottaa sosiaalisiin väyliin arvioidessaan riskiä uusista haavoittuvuuksista. (Williams ym., 2019)

Tutkimuksen johtopäätös on, että ohjelmistokehittäjät voisivat käyttää useita automaatiotyökaluja saavuttaakseen täydellisen kattavuuden. Lisäksi automaatiotyökalut löytävät riippuvuuksia ja haavoittuvuuksia riippuvuusjulistusten, kuten pom.xml ja build.gradle ulkopuolelta. (Williams ym., 2019)

## 3 Työn menetelmät

Opinnäytetyössä pohditaan riippuvuuksien ja haavoittuvuuksien olemassaoloa ja vaikutusta yhdistelmämenetelmin. Työssä tutkitaan olemassa olevaa todennettua dokumentaarista materiaalia sekä yhdistetään siihen kokeellista näkökulmaa. Pääpainon muodostuessa kokeelliselle selvitystyölle, pyritään pääsemään lähelle oikeaa tuotannossa olevaa ohjelmistoa tai tilannetta, jotta tehdyt johtopäätökset pysyisivät relevantteina ja todenmukaisina.

Aineisto ohjeistukseen koostuu olemassa olevan dokumentaation tulkinnasta sekä sen peilauksesta kokeelliseen selvitystyöhön. Näin toimimalla ohjeistus voidaan validoida. Käytettyä informaatio on myös koottu ohjelmistokehittäjän työstäni tulevilla havainnoilla.

Opinnäytetyössä käytettiin vertailuprosessina automaatiotyökalun, sekä ulkoiseen penetraatiotestaukseen perustuvaa kombinaatiota. Tällainen prosessi valittiin selvitystyöhön, sillä sitä pidetään suositeltuna tapana hallinnoida riippuvuuksia sekä haavoittuvuuksia (Fitzgerald, 2023).

Tässä opinnäytetyössä tutkitaan miten riippuvuudet ja haavoittuvuudet käyttäytyvät eri lukumäärissä. Aiheutuuko määrästä kynnyskysymys menestyneen hallinnan näkökulmasta, vai pystytäänkö nyky menetelmin hoitamaan vaativaakin kokonaisuutta? Työ pyrkii myös antamaan näkökulmaa ja käytännön suosituksia parempaan riippuvuuksien ja haavoittuvuuksien hallintaan.

## 4 Riippuvuuksien hallinta

Riippuvuuksien hallinta tarkoittaa sellaisten toimenpiteiden tai käytänteiden käyttöönottoa, joilla pystytään tunnistamaan, arvioimaan ja päivittämään ulkopuoliset kirjastot. Nämä kyseiset toimenpiteet myös vähentävät riippuvuuksien luomaa riskiä. Yleisimmät riippuvuuksien luomat ongelmat LeanIXin (n.d.) mukaan ovat seuraavat:

- Tietoturvariskit: Päivittämättömät riippuvuudet laajentavat hyökkäyspinta-alaa.
- Yhteensopivuusongelmat: Riippuvuudet aiheuttavat versioristiriitoja, jotka puolestaan aiheuttavat epävakautta ja käyttökatkoja.
- Suorituskykyongelmat: Vanhentuneet riippuvuudet lisäävät latausaikoja, muistinkäyttöä sekä prosessointikustannuksia.
- Ylläpito-ongelmat: Ohjelmiston päivittäminen tai koodipohjan päivitys voi olla vaikeampaa riippuvuuksien takia. Yleinen laatuvaikutelma laskee.
- Riippuvuuskauhu: Tarkoittaa sitä tilannetta, kun kompleksiset riippuvuussuhteet estävät ohjelmiston käytön, päivityksen tai julkaisun.

### 4.1 Tunnistus

Riippuvuus ohjelmistokehityksessä kuvaa suhteita komponenttien välillä, joissa komponentti tarvitsee toista toimiakseen. Esimerkkinä tällaisesta olisi ohjelma, joka suorittaisi kyselyt tietokantaan kirjastolla. Tällöin ohjelma olisi riippuvainen tästä kirjastosta toimiakseen. (Sonatype, n.d.)

Riippuvuuksien tunnistus tarkoittaa komponenttien tunnistusta, kirjausta sekä niiden suhteiden määrittelyä muihin ohjelmiston osiin nähden. Kokonaisuuden hahmottamista helpottaa usein visuaalinen kartta riippuvuuksista, jolloin saadaan parempi kokonaiskuva näistä kompleksisista suhteista riippuvuuksien välillä. Tähän tarkoitukseen on käytössä monia työkaluja, joista tämän opinnäytetyön yhteyteen on valittu hyvin suosittu ja tunnettu OWASP:n Dependency Track ohjelmisto. Työkalut skannaavat projektin lähdekoodin läpi ja antavat siitä kokonaiskuvan, erinäisin lisätiedoin sekä määrittelyin. Tätä kokonaiskuvaa

hyödyntäen on helpompi tunnistaa ja päivittää riippuvuuksia, sekä välttää mahdolliset komplikaatiot. (Faddom, n.d.)

## 4.2 Arviointi

Riippuvuuksien päivittämättä jättäminen nostaa ylläpidon vaatimuksia sekä altistaa ohjelmiston tietoturvariskeille. Tämän vuoksi myös ohjelmiston riippuvuudet ovat järkevää päivittää tasaisin väliajoin. Tämä myös laskee resurssitarvetta ja antaa kehittäjille enemmän vapauksia. (DXKB, 2019)

Selkeitä syitä riippuvuuksien päivityksille on monia. Tuotteen väärä toimintatapa on näistä selkein ja se vaatiikin välitöntä korjausta. Päivityksillä voidaan myös tavoitella uusia toimintoja sekä suorituskyky parannuksia. Tietoturva-aukot ovat myös yleinen syy päivittää riippuvuuksia, sillä yleensä havaitun haavoittuvuuden korjaus tulee uudemmassa versiossa. (DXKB, 2019)

## 4.3 Päivittäminen ja hallinta

Kirjastoriippuvuuksien hallinta on aikaa pois varsinaisesta tuotteesta ja sen kehitystyöstä, mutta toisaalta se pitää huolen, ettei teknistä velkaa pääse liiaksi muodostumaan. Usein joudutaankin pohtimaan projektikohtaisesti optimaalista ratkaisua. Säännöllisesti ja usein tehtynä työmäärä ei pääse missään kohtaa nousemaan kovin suureksi, ja päivitykset itsessään vievät vähemmän aikaa. Toisaalta harvemmin suoritettuna nämä päivitystyöt jättävät enemmän aikaa kehitystyölle mutta saattavat vaatia enemmän työtunteja suorituvaiheessa. (Crux, 2023)

Monesti joudutaan puntaroimaan näiden edellä mainittujen tapojen välillä. Molemmilla puolilla on oma kannattajakuntansa. Yleisesti alalla vallitseva toimintatapa on suorittaa korjaavia toimenpiteitä pienemmissä osissa ja usein. Tämä lähestymistapa antaa enemmän ennustettavuutta sekä kontrollia. Tällä tarkoitetaan siis sitä, että projektiin voidaan tuoda esimerkiksi pakollisia päivityksiä helposti ilman tarvetta suuremmalle refaktoroinneille koodikannassa. (Crux, 2023)

Säännöllinen ja jatkuva päivittäminen ei kuitenkaan välttämättä tarkoita aina uusimman version käyttöä. Monesti projektille asetetaan tietyt rajat, joiden sisällä operoidaan. Tällaisten käytäntöjen asettaminen onkin tärkeää, kun tavoitellaan toimivaa ja optimaalista kokonaisuutta. Uusimmat versiot sisältävät monesti löytämättömiä bugeja sekä mahdollisesti

muita ongelmakohtia. Päivitysrutiinit onkin syytä asettaa projektikohtaisesti kokonaisuus huomioiden. (Crux, 2023)

#### 4.4 Dokumentointi

Projektin koodin sekä riippuvuuksien dokumentointi, oli se sitten manuaalisesti tai työkaluilla luotu, on ensisijaisen tärkeää. Dokumentaatio antaa selkeää ja tarkkaa informaatiota toiminnallisuudesta ja riippuvuuksista sekä niiden relaatioista. Informaatio auttaa myös ymmärtämään, ylläpitämään, paikantamaan virheitä sekä välttämään niitä. (Linkedin, n.d.)

Kun projektia ajatellaan kokonaisuutena, on kunnollinen dokumentaatio suositeltavaa. Dokumentoimalla riippuvuussuhteet voidaan myös suunnitella toimenpiteitä pidemmälle kuin välittömiin (Iankovych, 2023). Nopeasti vaihtuviin tilanteisiin pystytään myös reagoimaan perusteellisemmin sekä tarkemmin, ja vaikutusten arviointi on helpompaa (Iankovych, 2023). Kunnollisesta dokumentaatiosta hyötyvät siis kaikki.

### 5 Haavoittuvuuksien hallinta

Haavoittuvuuksista puhuttaessa on hyvä taustoittaa nykyistä järjestelmää hieman. Vuonna 1999 julkaistu CVE eli Common Vulnerabilities and Exposures on järjestelmä, joka tarjoaa yksilöllisiä tunnisteita tietoturvaongelmille ja altistumisille haavoittuvuuksille. Nämä tunnisteet tunnetaan nimellä CVE-tunnisteet tai CVE-numerot. (Mitre, 2016)

CVE-järjestelmä kehitettiin poistamaan ongelma tietoturva-aukkojen nimeämisen kanssa. Alan monet toimijat käyttivät kukin omaa tapaansa nimetä haavoittuvuudet, josta seurasi mahdollisia aukkoja tietoturva-kattavuudessa. (Mitre, 2016)

1990-luvun lisääntynyt internetin ja sähköpostin käyttö sekä näiden synnyttämät tietoturvaongelmat loivat painetta luoda yhteinen nimeämiskäytäntö, standardi. (Chadd, 2020)

CVE-järjestelmä nousi nopeasti alan standardiksi ja se tärkeys on vain noussut (Mitre, 2016)

Haavoittuvuudesta mahdollisesti seuraavan tietovuodon vaikutukset ovat todella laajat sekä lähes aina negatiiviset. Näihin lukeutuvat niin rahalliset kuin maineelliset vahingot, puhumattakaan luottamuksen rikkoutumisesta ja mahdollisista oikeustoimista. (IBM, n.d.)

## 5.1 Tunnistus

Haavoittuvuuksien tunnistukseen liittyy yleensä skannaus automaatiotyökaluilla tai joissakin tapauksissa jopa manuaalinen penetraatiotestaaminen on hyödyksi. Tarkoitus on löytää lähde ja juurisyy haavoittuvuudelle. Usein haavoittuvuudet osuvat juuri avointen lähdekoodien kirjastoihin, jolloin helpoin ratkaisu olisi päivittää kirjasto uudempaan tai vaihtaa se toiseen. Joskin jälkimmäinen lähestymistapa aiheuttaa koodikannan refaktorointia. (Imperva, n.d.)

## 5.2 Arviointi

Haavoittuvuuksien arviointi toteutetaan yleensä sen mukaan millaisen riskin ne luovat. Riskianalyysi seuraa yleisesti käytettyjä riskianalyysimalleja sillä erotuksella, että se koskee vain haavoittuvuuksia. Analyyseissä pyritään selvittämään todennäköisyys tapahtumalle, sekä sen vaikutus toteutuessaan. (Andrioaie, 2022)

Kun haavoittuvuudet ja niiden vaikutukset ohjelmistoon on saatu selville, voidaan sen pohjalta tehdä korjaussuunnitelma (Andrioaie, 2022).

## 5.3 Korjaaminen

Arviointivaiheessa luodun suunnitelma mukaisesti siirrytään korjaamaan löydettyjä haavoittuvuuksia. Yleensä kriittisimmät haavoittuvuudet, jotka aiheuttavat toteutuessaan mittavat uhat tietoturvallisuudelle, pyritään korjaamaan ensin. Haavoittuvuuksien priorisointiin voi vaikuttaa myös projektin rakenne, koodikanta sekä käytettävissä olevat resurssit.

Usein haavoittuvuudet korjataan päivittämällä, paikkaamalla tai poistamalla kyseinen osa ohjelmistosta. Yleisesti kirjastojen luojat antavat omat suosituksensa toimista haavoittuvuuksien poistoon liittyen. (Snyk, n.d.)

Aina haavoittuvuuden paikkaaminen tai päivittäminen ei ole mahdollista tai järkevää. Syitä voi olla korjauksen puuttuminen tai sen vaatimat muutokset koodikannassa. Paikkaamisen ja päivittämiseen liittyy aina myös riski uuden koodin aiheuttamista tuntemattomista vaikutuksista ohjelmistoon. Joskus haavoittuneen osan poisto koodista riittää mitigoimaan haavoittuvuuden, tai ainakin antamaan aikaa korjauksen valmistumiseen asti. (Snyk, n.d.)

## 5.4 Seuranta ja raportointi

Uhat ja hyökkäysvektorit muuttuvat ja kehittyvät jatkuvasti. Tämä vaatii jatkuvaa uhkien analyysiä sekä monitorointia. Aiemmin paikattu haavoittuvuus saattaakin nopeasti muodostaa tietoturvauhan, jolloin siihen pitää pystyä reagoimaan uudelleen. Onkin erityisen tärkeää pysyä askeleen edellä ja turvata ohjelmisto turhilta haavoittuvuuksilta. (Rapid7, n.d.)

Tässä auttavat myös automaatiotyökalut, jotka skannaavat koodipohjaa läpi verraten tuloksia olemassa oleviin tietokantoihin. Tämän lisäksi on suositeltavaa myös auditoida ohjelmistoa esimerkiksi penetraatiotestaamalla vielä erikseen. (Microsoft, n.d.)

Penetraatiotestauksen voi ostaa ulkopuoliselta toimijalta, jolloin siitä saadaan paras hyöty. Tehtävään erikoistuneet toimijat osaavat identifioida, dokumentoida sekä raportoida löydöksistä tarpeeksi laajasti ja tarkasti. (Matas, 2023)

Penetraatiotestaus on prosessina raskaampi kuin automaatiotyökalun käyttö, mutta sisällytettynä muuhun monitorointiin tuottaa se merkittäviä etuja, kuten esimerkiksi tarkemman tuloksen verraten pelkkään ohjelmallisesti tuotettuun raporttiin. (Palla, 2022)

## 5.5 Hallinnan prosessi (kokonaisuutena)

Haavoittuvuuksien hallinnan prosessi koostuu yleensä neljästä eri vaiheesta. Seuraavaksi esittelen vaiheet kuvattuina kronologisessa järjestyksessä.

### 5.5.1 Tunnistus

Tunnistus on prosessin ensimmäinen vaihe. Tässä vaiheessa haavoittuvuudet pyritään kartoittamaan ja löytämään käyttämällä automaatiotyökaluja tai manuaalista toimenpidettä. Joissakin tapauksissa voi olla jopa järkevää käyttää ulkopuolista tahoa suorittamaan auditointi. Haavoittuvuudet tunnistetaan ja kategorisoidaan. Tämä vaihe yleensä pyritään integroimaan CI/CD-putkeen. (Paskoski, 2022)



### 5.5.2 Arviointi

Arviointi on prosessin toinen vaihe. Tämän vaiheen aikana edellisessä vaiheessa havaitut haavoittuvuudet saavat tärkeysjärjestyksen. Monet automaatiotyökalut luovat luokitukset itsestään perustuen johonkin tiettyyn metriikkaan esimerkiksi CVSS-pisteytykseen. CVSS-pisteytys kuuluu osana aiemmin käsiteltyyn CVE-järjestelmään. Pelkästään pisteytykseen perustuva järjestys ei monissa tilanteissa riitä itsekseen vaan mukaan pitää ottaa myös riskianalyysi ohjelmistoon perustuen. (Paskoski, 2022)

### 5.5.3 Korjaaminen

Korjaaminen on prosessin kolmas vaihe. Tässä vaiheessa haavoittuvuudet pyritään korjaamaan tärkeysjärjestyksessä paikkaamalla, päivittämällä tai poistamalla koodikannasta. Tämä on koko prosessin työläin vaihe ja se vaatii eniten resursseja. Joskus haavoittuvuutta ei saada korjattua edellä mainituilla toimenpiteillä, jolloin se pitää pyrkiä mitigoimaan muuttamalla kirjaston tai ohjelmiston koodia siten, että haavoittuvuus ja sen vaikutus lähenee nollaa. (Paskoski, 2022)

### 5.5.4 Seuranta ja raportointi

Seuranta ja raportointi on prosessin neljäs vaihe. Tässä vaiheessa pyritään todentamaan, että valitut toimenpiteet ovat olleet riittävät haavoittuvuuksien eliminoinnissa. Raportoinnissa pyritään pitämään kirjaa muun muassa tavoista, toimenpiteistä sekä vasteajoista. Raportointi toimii myös prosessin tarkastelun lähtökohtana, jolloin sitä voidaan kehittää ja muokata paremmin kuhunkin ohjelmistoon sopivaksi kokonaisuudeksi. (Paskoski, 2022)

## 6 Opinnäytetyön aikaiset työkäytänteet

Tutkimusdataa tätä opinnäytetyötä varten kerättiin työni ohella ohjelmistokehittäjänä. Tehtäviini kuului riippuvuuksien hallintaa sekä päivitystä, sekä haavoittuvuuksien mitigointia. Työni ollessa näin lähellä aihetta sain kasattua informaatiota, sekä luotua sopivia käytänteitä, jotka sopivat parhaiten prosessiin.

Tunnistusvaiheeseen käytössäni oli CI/CD-putkeen liitetty OWASP Dependency Track ohjelmisto, joka siis automaattisesti skannaa projektit läpi riippuvuuksien ja haavoittuvuuksien osalta. Riippuvuuksien kartoittaminen ja haavoittuvuuksien tunnistus ovat pakollisia vaiheita luotaessa toimivaa ja kattavaa hallintasuunnitelmaa.

Seuraavassa vaiheessa riippuvuuksista ja haavoittuvuuksista luotiin korjaussuunnitelma, siis tehtävä, jonka pohjalta lähdin niitä suorittamaan. Tehtävien kuvaukset riippuivat usein tehtävän haasteellisuudesta, helpommat tehtävät vaativat vähemmän selvitystä kuin vaativammat tehtävät. Riippuvuuden tai haavoittuvuuden ollessa vähäinen, eli kun sen korvaaminen tai poistaminen ei aiheuta merkittävää muutosta ohjelmistossa, voidaan sen mitigointiin antaa hieman sallivammat ohjeistukset. Toisaalta, jos työtehtävä tiedetään vaativaksi, on myös tehtävän kuvaus yleensä tarkempi ja ohjeellisempi.

Tehdyn tehtävän jälkeen muutokset vertaisarvioitiin ennen kuin niitä voitiin yhdistää muuhun ohjelmistokoodiin. Tämä vertaisarvio on tärkeä osa laadunvalvontaa, eikä sitä tulisi missään tilanteessa ohittaa. Se antaa myös uraansa aloitteleville hyvän mahdollisuuden kehittää osaamistaan sekä tietotaitoaan.

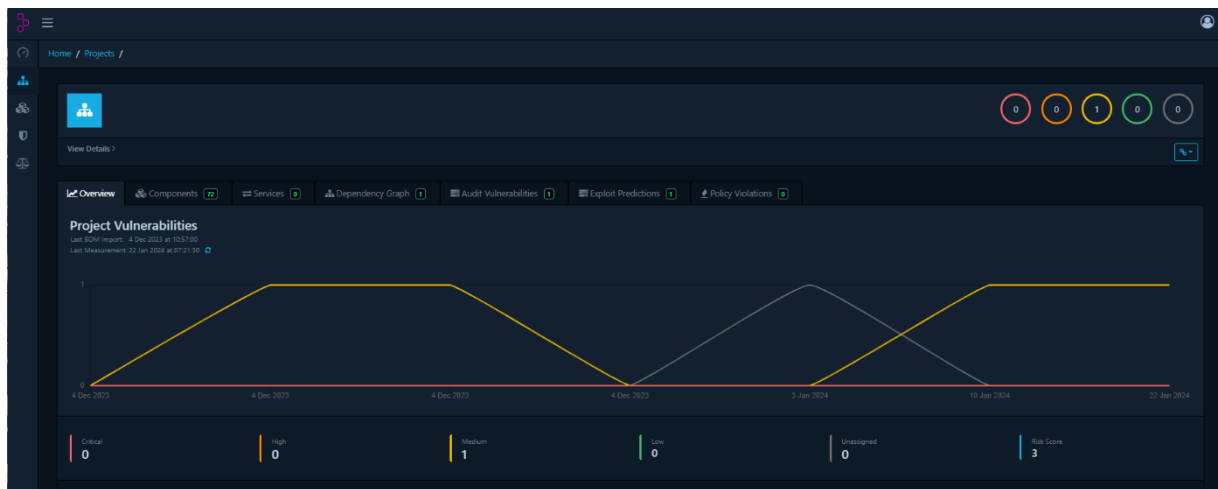
Kun ratkaisu lopulta hyväksytään vertaisarviossa, voidaan se yhdistää takaisin koodikantaan, jolloin se skannataan uudelleen. Tällöin pystytään varmistumaan siitä, että korjaus on onnistunut ja luodaan samalla dokumentaatiota muuttuneista komponenteista ja versionumeroista.

Tutkin parhaita käytäntöjä, kokeilin omia malleja ja seurasin versionumeroeron verrannollisuutta työtaakkaan. Mielestäni kokeellinen lähestymistapa oli tähän tutkimukseen parhaiten sopiva metodi. Tästä pääsin samaan lopputulokseen kuin Malik (2023) artikkelissaan, joka osoittaa, että tasainen päivitysvauhti pitää työmäärän sekä kompleksisuuden matalampana. (Malik, 2023)

## **6.1 Käytetyt ohjelmistot**

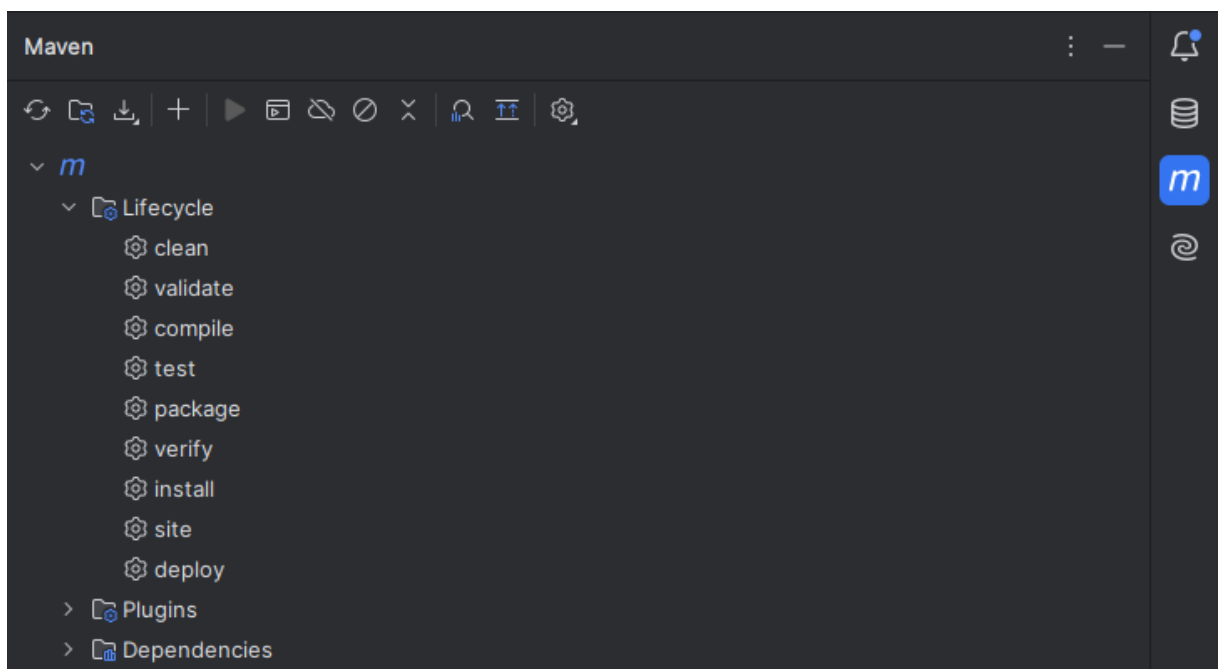
OWASP Dependency Track on komponenttien analysointiin erikoistunut alusta, jolla organisaatiot voivat tunnistaa ja vähentää riskiä ohjelmistotoimitusketjussa. Sillä voidaan tunnistaa komponentit sekä seurata niiden käyttöä organisaation projekteissa. Kuvassa 5 projektisivu Dependency Track:ssa kuvattuna. Dependency Track myös tunnistaa tietoturvariskit sekä osaa kertoa, mitkä osat ohjelmistoa ovat uhattuna. Dependency Track voidaan myös integroida osaksi useita eri järjestelmiä. Tukilistalta löytyvät kaikki eniten käytetyt järjestelmät. (Owasp, n.d.)

Kuva 5. Projektinäkymä Dependency Track:n sisällä.



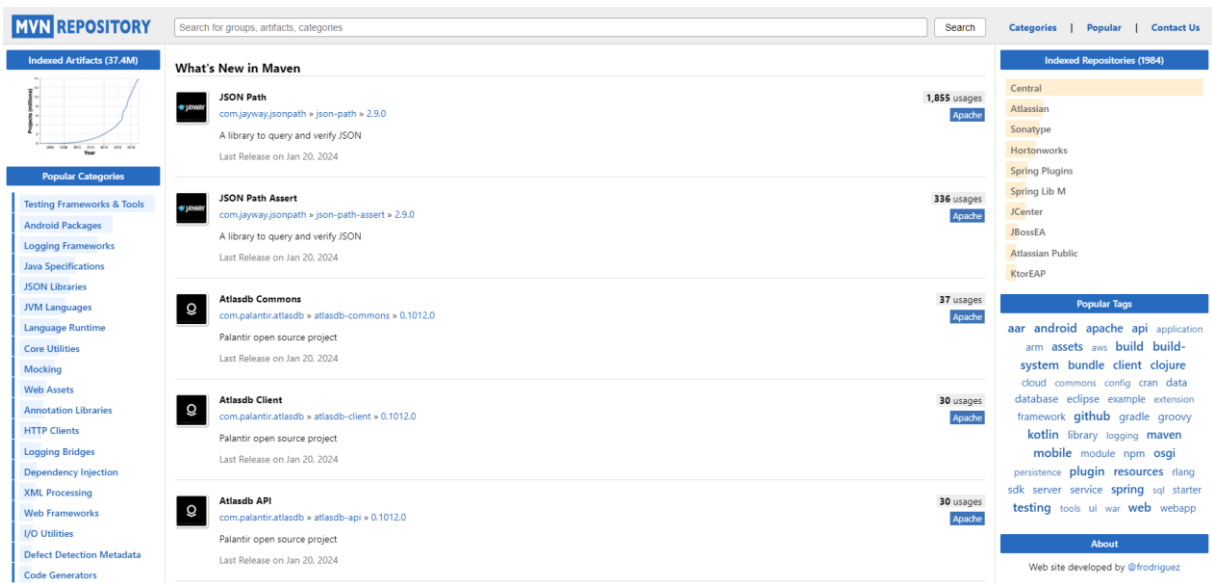
Maven on rakennusautomaatiotyökalu, joka on kehitetty helpottamaan kehittäjien työtä. Maven on oivallinen työkalu projektin perustamisessa sekä hallinnassa, kuten kuvassa 6 on kuvattu. Se mahdollistaa parhaiden käytäntöjen käytön, sekä helpottaa riippuvuuksien hallintaa merkittävästi. Maven käyttää riippuvuuksien julistamiseen pom.xml tiedostoa, jossa luetaan projektiin kuuluvat riippuvuudet. Projektin käynnistyessä Maven lataa kyseisellä tiedostolla olevat riippuvuudet automaattisesti, jolloin niiden päivittäminen on huomattavasti helpompaa. (van Dijk, 2022)

Kuva 6. Maven-valikko.



Mvn Repository -sivusto on keskitetty hakemisto riippuvuuksille, joita voidaan ladata Maven-työkalulla projektiin. Sivustoa käyttävät ammattilaiset riippuvuuksien seurantaan sekä päivitysten suunnitteluun. Täällä voidaan myös valita projektille tarvittavia riippuvuuksia, sillä sivusto antaa järjestää riippuvuudet kategorioittain halutun preferenssin mukaan. Kuvassa 7 sivuston aloitusnäkö. (MvnRepository, n.d.)

Kuva 7. Sivuston aloitusnäkö.



## 6.2 Saavutetut tulokset ja oppimiskokemukset

Kuten jo monessa asiayhteydessä tässä opinnäytetyössä on todettu, on kirjastojen ja haavoittuvuuksien päivittäminen tärkeä ja olennainen osa ohjelmistokehitystä. Se takaa ohjelmistolle parhaat mahdollisuudet toimia tietoturvallisesti ja optimaalisesti. Toisaalta sen laiminlyönti johtaa pahimmillaan tietovuotoon tai ohjelmiston toimimattomuuteen.

Päivitysprosessille on varattava tarpeeksi resursseja ja sen tulee olla luonteeltaan jatkuva. Jatkuvan prosessin myötä pystytään reagoimaan ohjelmiston ja toimintojen asettamille vaatimuksille riittävän aikaisessa vaiheessa, ennen kuin teknistä velkaa alkaa muodostumaan liikaa. Näin toimimalla vältetään myös versioiden yhteensopivuusongelmat, tai ainakin minimoidaan ne.

Käytettäessä vähemmän kerralla ja useammin -mallia, pystytään prosessi resursoimaan tarkemmin, ja sen kuormittavuus on pienempi verraten harvoin tehtävään suureen päivityssumaan. Huonoin tilanne on kuitenkin ilman suunnitelmaa tai prosessia eteneminen,

joka johtaa isoon määrään teknistä velkaa. Näistä tilanteista pääseminen vaatii kunnollista suunnittelua sekä dokumentointia. (Maayan, 2023)

## **7 Tulokset ja yleinen pohdinta**

Tämän opinnäytetyön saavuttamaa tulosta pohdittaessa on huomioitava se, että ohjelmistokehitys ja riippuvuuksien sekä haavoittuvuuksien hallinnan olevan tilannekohtaisia kokonaisuuksia. Tällä tarkoitetaan, sitä että niistä on vaikea antaa tarkkaa yleistä ohjeistusta, joka toimisi kaikissa tilanteissa. Ohjeet ja suuntaviivat ovat viitteellisiä, ja niitä on pyritty muotoilemaan yleisempään muotoon.

Työn aikana nousseet havainnot ja huomiot on pyritty aina miettimään tästä näkökulmasta, ja olenkin tämän prosessin myötä päätenyt jättämään suosituksia ja ohjeistuksia tämän työn ulkopuolelle. Olen kuitenkin läpi käynyt selvitystyön aikana nousseet huomiot läpi, sekä pyrkinyt muodostamaan niistä yleisiä ohjeita, joita voisi soveltaa jokaiseen ohjelmistokokonaisuuteen.

### **7.1 Yhteenveto tuloksista**

Opinnäytetyön edetessä minulle kävi nopeasti selväksi, että parhaat käytännöt ovat laajalti tiedossa, mutta niiden käyttöönotossa ja resursoimisessa oli enemmän ongelmia. Ohjelmistokehitysalan jatkuva kehittyminen ja vaatimustason nousu vie enemmän resursseja päivä päivältä.

Kasuvat vaatimukset asiakkailta saavat yritykset panostamaan kehitykseen asiakkaiden toiveiden mukaisesti. Tämä on sinänsä ymmärrettävää, koska asiakkaat maksavat mieluummin ohjelmistosta, joka palvelee heidän tarpeitaan.

Tämän vuoksi ohjelmistot pääsevät kerryttämään teknistä velkaa, joka täytyy jossakin vaiheessa maksaa. Teknisen velan määrä on verrannollinen sen vaatiman työn määrään. Esittäisin kuitenkin vielä lisävaihtamaan tähän tulosten valossa, että teknisen velan muuttaminen työmääräksi ei ole lineaarista, vaan suhdetta voisi kuvailla monimutkaiseksi.

Opinnäytetyössä kävi kuitenkin selväksi, että tiheämpi päivitysrytmi sekä vähäisempi tekninen velka tekevät päivitystyöstä huomattavasti helpompaa ja nopeampaa. Se voidaan myös yleensä toteuttaa pienemmillä resursseilla. Tämän lisäksi työssä selvisi, että riippuvuuksien ja haavoittuvuuksien hallinnan prosessin pitää olla luonteeltaan jatkuva ja suunnitelmallinen.

## 7.2 Tulkinta ja analyysi

Opinnäytetyön tuloksista voi rakentaa neuvoja ja suosituksia koskien riippuvuuksien ja haavoittuvuuksien hallinnan prosessia. Ensimmäisenä yleisenä neuvona voisin todeta, että suunnitteluun ja optimointiin pitää varata riittävästi aikaa. Kun hallinnan prosessi on jatkuva, sallii se optimoinnin jokaisella iteroinnillaan. Pienin askelin kohti tehokkaampaa ja nopeampaa prosessia. Näin ollen resurssitarve pienenee ja ohjelmistokehittäjiä vapautuu tuottavaan kehitystyöhön.

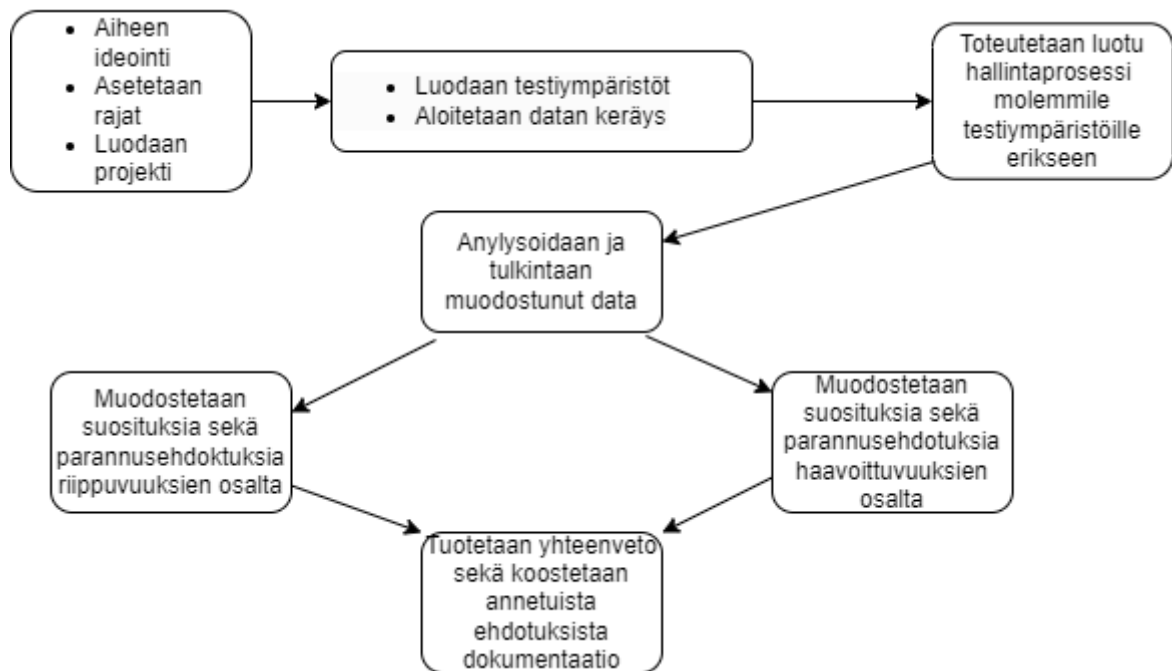
Toisena suosittelisin myös automaatiotyökalujen käyttöä vähintäänkin tunnistusvaiheessa, jolloin ohjelmistoprojektista on aina viimeisin tieto ja dokumentaatio saatavilla. Automaatiotyökaluilla voidaan myös validoida riippuvuuksien päivitykset sekä haavoittuvuuksien korjaukset.

## 7.3 Jatkotutkimusta varten

Opinnäytetyön tehdessä esiin nousi muutamia ideoita sekä kehitys- sekä parannusehdotuksia tutkimukseen liittyen. Jatkotutkimusaiheeksi voisi sopia vertaileva tutkimus kahden projektin välille. Tutkimus osoittaisi käytännön hyödyt päivitysten ja haavoittuvuuksien hallinnalle konkretian kautta. Toisen projektin kirjastopäivityksiä sekä haavoittuvuuksia hallittaisiin jatkuvan prosessin avulla ja toisen satunnaisesti. Tällaisen tutkimuksen tuloksia ja analyysyjä voisi käyttää pohjana prosessinhallinnan tukena sekä ohjenuorana.

Tässä ohjeistavassa selvitystyössä on perusteltua käydä perusteita hieman tarkemmin läpi, jotta saadaan lukijalle riittävän laaja perusta ja tietotaso tukemaan ymmärrystä. Jatkotutkimus voisi keskittyä enemmän vertailukelpoisten tulosten saamiseen sekä hallinnan prosessin kehittämiseen, kuten kuvassa 8 on kuvattu. Mielestäni tämä olisi myös sopiva jatkumo tälle työlle.

Kuva 8. Jatkotutkimuksen prosessikuvaus.



## 8 Yhteenveto ja johtopäätökset

Lucas Nelaupen (2021) artikkelin mukaan on suositeltua päivittää kirjastot ja haavoittuvuudet ennen kuin sille on pakottava tarve. Tällä tarkoitetaan sitä, että ohjelmisto toimii vanhalla versioilla mutta päivitys voidaan jo suorittaa. Täytyy pitää kuitenkin mielessä, että jokainen päivitys pitäisi arvioida erikseen vertaamalla muodostunutta riskiä tai hyötyä sen vaatimaan työpanokseen. (Nelaup, 2021)

Mielestäni kuitenkin aina ei tarvitse heti päivittää uusimpaan versioon, kun sellainen saapuu saataville. Päivittämisen pitäisi olla tarkoin suunniteltu prosessi, joka on muodoltaan jatkuva, sekä riskianalyysia sisältävä. Tärkeää asiassa on, että päivitysten ja haavoittuvuuksien hallinta noudattaa tietynlaista kaavaa, sekä hyödyntää saatavilla olevaa dataa ja informaatiota kootessaan uhka-arviota sekä riskiarviota.

Yritys hyötyy monella tavalla siitä, että ohjelmistojen kirjastot pysyvät ajantasaisina, sekä tietoturvasuhteena jatkuvasti. Päivittämällä taataan ohjelmistojen tietoturvasuus sekä yhteensopivuus muiden ohjelmistojen kanssa. Päivitykset myös monesti antavat uusimmat toiminnot käyttöön ja takaavat parhaan mahdollisen suorituskyvyn. (Hetler, 2022)

### 8.1 Tärkeimmät havainnot

Tästä opinnäytetyöstä nostan esille tärkeimmiksi havainnoiksi, riippuvuuksien ja haavoittuvuuksien hallinnan prosessin kokonaisuutena. Tämä edellä mainittu on jatkuva prosessi, jossa määritellään ja luokitellaan päivitykset ja haavoittuvuudet tietyin kriteerein arvojärjestykseen sekä tehdään toimintasuunnitelma niiden suorittamiseksi tai poistamiseksi.

Pitkälle hiottu prosessi itsessään edistää ohjelmistojen ajankestävyyttä sekä takaa resurssivaatimusten pysymisen hallittavana. Prosessi voidaan irrottaa omaksi kokonaisuudekseen, jota hallinnoi yhdestä kahteen vastuuhenkilöä. Näin saadaan ohjelmistokehittäjät ja ohjelmistoinfiniti tekemään tuottavaa työtä päätuotteen parissa. Kun taustalla kulkeva prosessi on identifioinut sekä resursoinut päivitykset, voidaan työ jakaa ohjelmiston kehittäjille tehtäväksi. Näin toimimalla saadaan minimoitua resurssien vienti, ja taattua maksimaalinen tulos riippuvuuksien ja haavoittuvuuksien päivitysten osalta. Työnteon kustannusten optimointi onkin nykyään kasvava trendi, joka varsinkin heikompina taloudellisina aikoina nousee esiin (Kenton, 2021).



## 8.2 Pääasialliset johtopäätökset

Opinnäytetyön aikana kävi selväksi, että parhaat toimintatavat riippuvuuksien ja haavoittuvuuksien päivityksille koostuvat muutamista avain toimitavoista. Näihin kuuluvat selkeä strategia päivitysten ja riippuvuuksien hallintaan, automaation hyödyntäminen, jatkuva valvonta sekä testaus. Myös riippuvuuksien dokumentointi ja niiden päivittämisen suuntaviivat auttavat jatkossa välttämään liian suurta teknistä velkaa. Saman suuntaiseen tulokseen on tullut myös Danduc (2024) tuoreessa artikkelissaan *Software dependencies explained* (Danduc, 2024).

## 8.3 Työn merkitys, soveltaminen sekä validointi

Tämän opinnäytetyön tarkoituksena on avata ja pyrkiä luomaan ohjeistusta riippuvuuksien ja haavoittuvuuksien hallinnan helpottamiseksi. Ohje on muodoltaan yleinen, eikä sen tarkoitus ole kattaa kaikkia mahdollisia tilanteita. Ohjelmistokehitys toimialana on monitasoista ja tilanneriippuvaista, yksi ratkaisu ei toimi kaikissa tilanteissa.

Opinnäytetyöni on tarkoitus toimia suuntaviivana, ja antaa pohjatietoa, miten luodaan ja ylläpidetään tietoturvallista ja toimivaa ohjelmistokokonaisuutta riippuvuuksineen. Opinnäytetyön tulokselle hain validoinnin minua kokeneemmilta alan asiantuntijoilta, joille pääsin esittämään muutamia kysymyksiä aiheesta, toteutuksesta, tuloksista ja merkittävydestä.

Mikko Virtala, joka työskentelee ohjelmistokehityksessä Senior -asematasolla, vastasi kysymykseen pystyykö tämän opinnäytetyön löydöksiä hyödyntämään oikeassa tuotantoympäristössä seuraavasti:

Kyllä, tutkimuksen löydökset voidaan hyödyntää oikeassa tuotantoympäristössä. Tiheämpi päivitysrytmi ja teknisen velan vähentäminen tehostavat päivitysprosessia ja vaativat vähemmän resursseja. Riippuvuuksien ja haavoittuvuuksien jatkuva ja suunnitelmallinen hallinta on myös tärkeää. Käytännön kokemus vahvistaa, että näiden periaatteiden soveltaminen parantaa ohjelmistokehityksen laatua ja turvallisuutta. (henkilökohtainen tiedonanto, 22.1.2024)

Mielestäni tämä vahvistaa tämän opinnäytetyön teorian ja löydökset, sekä osoittaa, että näitä osataan jo hyödyntää niin kehitys- kuin tuotantoympäristöissä. Hallinnan suunnitelmallisuus sekä jatkuvuus nousevat tärkeään rooliin, kuten Virtala mainitsi.

Kysymykseen onko opinnäytetyö tehty tarpeeksi laajasti, jotta sen saadut tulokset olisivat vertailukelpoisia Virtala vastasi:

Tutkimus on keskittynyt yhteen tapaukseen, ja viittaa samankaltaisiin aiempiin tutkimuksiin, mikä vahvistaa sen tuloksia. Käytännön koodaustehtävistä koostuva esimerkkitapaus tarjoaa laajan näkemyksen päivitys- ja haavoittuvuusongelmiin, mikä tekee tutkimuksesta käytännönläheisen ja relevantin opinnäytetööhön. (henkilökohtainen tiedonanto, 22.1.2024)

Tein itse saman huomion, kuin Virtala kommentissaan, että opinnäytetyö olisi varmasti hyötynyt kahden eri tapauksen vertailusta. Päätin kuitenkin itse rajata aihetta, sillä opinnäytetyön sisältö olisi tuolloin laajentunut kohtuuttoman paljon. Johdin tästä huomiosta jatkotutkimusehdotuksen, jossa seurataan kahta samanlaista projektia, jossa toisessa riippuvuuksia ja haavoittuvuuksia hallitaan parhaiden käytäntöjen mukaan, toisen ollessa ilman prosessia. Olisi mielenkiintoista nähdä, miten projektit erottuisivat toisistaan tutkimuksen lopussa niin käytettävyydeltään kuin tietoturvan osalta.

Kun kysyin Virtualta olinko jättänyt huomioimatta joitakin tärkeitä näkökulmia, vastani hän seuraavasti: ”Ei erityisen merkittävästi. Olisin kuitenkin arvostanut, jos tutkimuksessa olisi esitelty erilaisia automaattisia työkaluja riippuvuuksien ja haavoittuvuuksien hallintaan hieman laajemmin.” (henkilökohtainen tiedonanto, 22.1.2024) Mielestäni olisikin ollut todella mielenkiintoinen näkökulma vertailla erilaisia automaatiotyökaluja käytännössä, mutta se olisi parempi tehdä omana erillisenä tutkimuksenaan. Koen, että opinnäytetöiden ongelmaksi muodostuu usein liian löyhästi rajoitettu tutkimusaihe. Tästä seuraa joko opinnäytetyön koon paisuminen, tai vastaavasti opinnäytetyön syvyyden jääminen kevyeksi.

Toinen haastatteleistani asiantuntijoista on Lead Software Developer Tuomo Pohjola. Hän totesi opinnäytetöistä näin:

Pitkälti samaa mieltä tutkimuksen johtopäätöksistä. Haavoittuvuuksien hallinta kannattaa olla jatkuva prosessi, jonka myötä ohjelmiston tietoturvakin yleensä paranee. Automaatio on vain osa tätä kokonaisuutta, ja esimerkiksi automaation hyödyntäminen haavoittuvuuksien korjausten validoinnissa vapauttaa myös vastuuhenkilöiden resursseja. Kuten johtopäätöksissä mainittiin, niin haavoittuvuuksien valvonta ja mittaaminen on avain tekijä. Ilman jatkuvaa mittaamista työ voi jäädä helposti tekemättä tai haavoittuvuudet huomaamatta. Tutkimuksen pohjalta pystyn antamaan suosituksia haavoittuvuuksien hallinnointia varten.

Haastattelussa Pohjola mainitsi myös prosessin vastuuttamisen tärkeyden. (henkilökohtainen tiedonanto, 23.1.2024) Näihin kommentteihin on helppo yhtyä. Juuri prosessin vastuuttaminen muutamalle henkilölle varmistaa sen, että prosessi tulee suoritettua sopivalla tarkkuudella, eikä siihen uhrata liikaa resursseja.

Pohjola jatkoi vielä opinnäytetyöstä:

Samaa mieltä tutkimuksen kanssa, että nopea päivitysrytmi tekee työstä useimmiten helpompaa ja että riippuvuuden ei silti tarvitse olla aina uusimmassa versiossa, jossa voi olla paljon tuntemattomia ongelmia. Tässä asiassa korostuu tutkimuksen mainitsema riskianalyysi.

Mielestäni tähän Pohjolan viimeisiin lauseen kiteytyy hyvin tämän opinnäytetyön yksi johtopäätös. Tietoturvallisuus vaatii aina tietyn verran riskianalyysiä osakseen, jotta kehitystyötä voidaan tehdä ilman vaikeita välivaiheita. Tietoturva-aukkojen paikkaaminen tarvitsee myös riskianalyysiä, jotta havaitut ongelmakohdat voidaan luotettavasti järjestää haitta-asteen mukaan järjestykseen.

## Lähteet

- Ashigbi, A. (14.6.2020). *A case against software dependencies*.  
<https://www.4degrees.ai/blog/a-case-against-software-dependencies>
- Andrioaie, A. (1.1.2022). *What Is Risk-Based Vulnerability Management?*  
<https://heimdalsecurity.com/blog/what-is-vulnerability-risk-management/>
- Blaisdell, R. (29.9.2022). *Dependency Management: Why is it important?*  
<https://rickscloud.com/dependency-management-why-is-it-important/>
- Chadd, K. (24.11.2020). *The history of cybersecurity*. <https://blog.avast.com/history-of-cybersecurity-avast#the-1990s>
- Cimpanu, C. (26.6.2020). *More than 75% of all vulnerabilities reside in indirect dependencies*. <https://www.zdnet.com/article/more-than-75-of-all-vulnerabilities-reside-in-indirect-dependencies/>
- Crux, F. (21.3.2023). *It's worth putting in the effort to regularly update dependencies*.  
<https://felixcrux.com/blog/it-is-worth-regularly-updating-dependencies>
- Danduc, M. (18.1.2024). *Software Dependencies Explained*.  
<https://phoenixnap.com/blog/software-dependencies>
- DXKB. (11.11.2019). *Updating the Dependencies*.  
<https://developerexperience.io/articles/updating-the-dependencies>
- Faddom. (n.d.). *The Complete Guide to Application Mapping*. <https://faddom.com/application-dependency-mapping/>
- Fitzgerald, A. (12.7.2023). *A Step-by-Step Guide to the Vulnerability Management Process [+ Policy Template]*. <https://secureframe.com/blog/vulnerability-management>
- Garousi, V. (1.10.2023). *Mastering dependency management: version catalog & convention plugin at Scale*. <https://proandroiddev.com/mastering-android-dependency-management-b94205595f6b>
- Iankovych, N. (4.6.2023). *Why Software Documentation Is Important. Steps to Create It in 2023*. <https://medium.com/@o.kornienko/why-software-documentation-is-important-steps-to-create-it-in-2023-820dcd2fd6fa>
- IBM. (n.d.). *How to identify security vulnerabilities within an application, impacts and remediation*. <https://www.ibm.com/support/pages/how-identify-security-vulnerabilities-within-application-impacts-and-remediation>
- Imperva. (n.d.). *Vulnerability Assessment*. <https://www.imperva.com/learn/application-security/vulnerability-assessment/>
- Hetler, A. (18.5.2022). *5 reasons software updates are important*.  
<https://www.techtarget.com/whatis/feature/5-reasons-software-updates-are-important>

- Huntley, S. (16.2.2023). *Fog of war: how the Ukraine conflict transformed the cyber threat landscape*. <https://blog.google/threat-analysis-group/fog-of-war-how-the-ukraine-conflict-transformed-the-cyber-threat-landscape/>
- Jack, V. (23.11.2023). *Europe's grid is under a cyberattack deluge, industry warns*. <https://www.politico.eu/article/energy-power-europe-grid-is-under-a-cyberattack-deluge-industry-warns/>
- Juillet, R. (13.5.2022). *How Covid-19 boosted global demand for software developers*. <https://www.bocasay.com/covid19-boosted-global-demand-software-developers/>
- Kenton, W. (5.1.2021). *Cost Cutting: Importance of Strategy, Risks Posed*. <https://www.investopedia.com/terms/c/cost-cutting.asp>
- Krysik, A. (8.5.2023). *The Future of Software Engineering: Key Emerging Trends in 2023*. <https://stratoflow.com/future-of-software-engineering/>
- LeanIX. (n.d.). *TYPES, RISKS, AND BEST PRACTICES OF Software Dependencies*. <https://www.leanix.net/en/wiki/vsm/software-dependencies>
- Linkedin. (n.d.). *How do you handle complex and interdependent software components and dependencies?* <https://www.linkedin.com/advice/0/how-do-you-handle-complex-interdependent-software-components>
- Maayan, G. (16.1.2023). *Risks of Application Dependencies and How to Mitigate Them*. <https://www.i-programmer.info/programming/methodology-a-testing/16015-risks-of-application-dependencies-and-how-to-mitigate-them.html>
- Matas, N. (18.10.2023). *Why Penetration Testing Is Important for Your Business*. <https://infinum.com/blog/why-penetration-testing-is-important/>
- Malik, S. (16.8.2023). *Insight - Regularly update dependencies to maintain security and compatibility*. <https://www.linkedin.com/pulse/insight-regularly-update-dependencies-maintain-security-malik/>
- MvnRepository. (n.d.). *What's New in Maven*. <https://mvnrepository.com/>
- Microsoft. (n.d.). *What is vulnerability management?* <https://www.microsoft.com/en-us/security/business/security-101/what-is-vulnerability-management>
- Mitre. (1.2.2016). *Common Vulnerabilities and Exposures — CVE®*  
*The Standard for Information Security Vulnerability Names*. <https://cve.mitre.org/docs/cve-intro-handout.pdf>
- Murray, A. (16.5.2023). *Dependency Management: A Guide and 3 Tips to Keep You Sane*. <https://www.mend.io/blog/dependency-management-a-guide-and-3-tips-to-keep-you-sane/>
- Nelaupé, L. (30.3.2021). *Keeping 170 Libraries Up to Date on a Large Scale Android App*. <https://medium.com/grab/keeping-170-libraries-up-to-date-on-a-large-scale-android-app-6a6ea054c175>

- Owasp. (n.d.). OWASP Dependency-Track. <https://owasp.org/www-project-dependency-track/>
- Palla, U. (11.11.2022). *The top three differences between an open source audit and an open source scan*. <https://www.synopsys.com/blogs/software-security/differences-between-open-source-audit-vs-scans.html>
- Paskoski, N. (21.6.2022). *4 Stages of the Vulnerability Management Process*. <https://rhisac.org/vulnerability-management/stages-process/>
- Perälä, J. (2023). *Ohjelmistoriippuvuuksien automaattinen ylläpito* [opinnäytetyö, Vaasan ammattikorkeakoulu]. [https://www.theseus.fi/bitstream/handle/10024/798413/Perala\\_Jussi.pdf?sequence=2&isAllowed=y](https://www.theseus.fi/bitstream/handle/10024/798413/Perala_Jussi.pdf?sequence=2&isAllowed=y)
- Rapid7. (n.d.). *Vulnerability Management Guide*. <https://www.rapid7.com/fundamentals/vulnerability-management-and-scanning/>
- Snyk. (n.d.). *Software dependencies: How to manage dependencies at scale*. <https://snyk.io/series/open-source-security/software-dependencies/>
- Snyk. (n.d.). *4 steps of the Vulnerability Remediation Process*. <https://snyk.io/learn/vulnerability-remediation-process/>
- Soare, B. (13.9.2023). *What Are the Main Attack Vectors in Cybersecurity?* <https://heimdalsecurity.com/blog/attack-vectors/>
- Sonatype. (n.d.). *What is a software dependency?* <https://www.sonatype.com/launchpad/what-are-software-dependencies>
- Tremplin-Numérique. (22.1.2022). *Mikä on semanttinen versiointi?* Haettu 21.1.2024 osoitteesta <https://www.tremplin-numerique.org/fi/mik%C3%A4-on-semanttinen-versiointi>
- Van Dijk, M. (27.9.2022). *Keeping dependencies up to date with Maven*. <https://medium.com/@mlvandijk/keeping-dependencies-up-to-date-with-maven-be8f7fb6441e>
- Williams, L. Imtiaz, N. Thorn, S. (11.10.2021). *A comparative study of vulnerability reporting by software composition analysis tools*. <https://ar5iv.labs.arxiv.org/html/2108.12078>
- Wisenhoff, D. (2.1.2021). *Vulnerabilities in Dependencies, Third Party Components and Open Source: What you need to know*. <https://debricked.com/blog/vulnerabilities-dependencies/>