Avoimen lähdekoodin kirjastot ja tietoturva – ohjelmistokehittäjän opas

Sonja Fallström

Haaga-Helia ammattikorkeakoulu
Tradenomi (AMK) tietojenkäsittely
Opinnäytetyö
2025

Tiivistelmä

Tekijä(t)
Sonja Fallström

Tutkinto
Tradenomi

Raportin/Opinnäytetyön nimi
Avoimen lähdekoodin kirjastot ja tietoturva – ohjelmistokehittäjän opas

Sivu- ja liitesivumäärä
37 + 13

Tämän opinnäytetyön tavoitteena oli selvittää avoimeen lähdekoodin ohjelmistokirjastoihin liittyviä tietoturvariskejä sekä niiden ehkäisyä kehittäjän oman tarkistuksen ja työkalujen avulla. Opinnäytetyössä tarkasteltiin erityisesti npm-pakettien, jotka ovat tietyntyyppisiä ohjelmistokirjastoja, tietoturvaan liittyviä käytäntöjä. Npm-paketteja hyödynnetään esimerkiksi JavaScriptia käyttävissä ohjelmistoprojekteissa. Opinnäytetyössä ei käsitellä muita ohjelmistoprojekteihin liittyviä tietoturvariskejä, eikä tietoturvallisia käytäntöjä muiden ohjelmointikielten, kuten Javan tai Pythonin, avoimen lähdekoodin kirjastoille.

Opinnäytetyön toimeksiantajana oli HUS Tietohallinto. Taustana oli toimeksiantajan tarve tarkentaa kriteereitä avoimen lähdekoodin ohjelmistokirjastojen käytölle. Opinnäytetyön lopputuotoksena valmistui opas avoimen lähdekoodin ohjelmistokirjastojen riskien tunnistamiseen sekä npm-pakettien käyttöön omassa ohjelmistoprojektissa. Valmis opas on kirjoitettu yleisohjeeksi, joten se soveltuu sekä toimeksiantajan ohjelmistokehityksessä mukana oleville työntekijöille että yleisesti ohjelmistokehittäjien käyttöön.

Opinnäytetyön aikana keväällä 2025 kerättiin tietoa yleisesti avoimen lähdekoodin ja ohjelmistokirjastojen hyödyistä ja riskeistä, avoimen lähdekoodin ohjelmistokirjastoihin liittyvistä tietoturvariskeistä sekä tietoturvallisista käytännöistä npm-pakettien käyttöön pakettia valitessa, käyttöönotossa ja omaa ohjelmistoprojektia ylläpitäessä. Tietoa kerättiin ajankohtaisista tietoturvayritysten raporteista, viranomaislähteistä sekä ohjelmistokirjastoihin liittyvistä blogikirjoituksista. Tämän lisäksi haastateltiin vapaamuotoisesti muutamia toimeksiantajan työntekijöitä ohjelmistokirjastojen käytöstä.

Opinnäytetyössä tutkittiin kahta komentorivityökalua npm-pakettien tunnettujen haavoittuvuuksien tunnistamiseen, jotka olivat npm audit ja NPQ. Työkalujen ominaisuuksia testattiin asentamalla muutamia npm-paketteja ja vertailemalla työkalujen raportteja. Testauksen tehdyn havainnon perusteella npm audit sopii jo käytössä olevien npm-pakettien hallinnointiin ja NPQ uusien käyttöönotettavien pakettien testaukseen. Molempia työkaluja suositeltiin käytettäväksi lopputuotoksena syntyneessä ohjeessa. Valmiit ohjeet tarjoavat perustietoa ohjelmistokirjastoihin liittyvistä tietoturvauhkista ja niiden ehkäisystä eri keinoin. Ohjeista kerätyn vapaamuotoisen palautteen perusteella ohjeet olivat selkeät ja käytännönläheiset.

Kerätyn tiedon perusteella npm-pakettien käyttöönottamisessa on paljon tarkistettavaa, ja paketteja on lähes mahdotonta tarkistaa pelkästään manuaalisesti. Tämän vuoksi on tarpeellista, että ohjelmistokirjastojen käytön seurannaksi otetaan käyttöön työkaluja, jotka reagoivat kirjastojen käytön eri vaiheissa.

Asiasanat
ohjelmistokirjasto, npm, tietoturva, avoin lähdekoodi, ohjelmistokehitys

Sisällys

1 Johdanto

Nykyään on vaikeaa löytää ohjelmistoprojektia, jossa ei olisi käytetty ensimmäistäkään kolmannen osapuolen kehittämää ja avoimena lähdekoodina julkaistua ohjelmistokirjastoa. Arviolta jopa 97 % kaupallisista ohjelmistoprojekteista sisältää vapaata ja avointa lähdekoodia koodikannassaan. (Black Duck Software Inc 2025, 2; 27). Kirjastoilla on eri käyttötarkoituksia, ja niiden käyttäminen säästää huomattavasti ohjelmistokehittäjien aikaa ja vaivaa. Kirjastot tuovat myös laatua ja varmuutta ohjelmistojen toiminnallisuuksiin, kuten lomakkeiden tai päivämäärien käsittelyyn. JavaScript-, TypeScript- ja React-pohjaisissa ohjelmistoprojekteissa ohjelmistokirjastot ovat usein npm-paketteja, joita asennetaan ja ylläpidetään npm-paketinhallintajärjestelmän kautta.

Julkisia npm-paketteja on rekisterissä miljoonittain. Miljoonien mahdollisuuksien joukossa piilee myös riskejä: tunnetut ja tunnistamattomat tietoturvahaavoittuvuudet, paketteihin kohdistuvat toimitusketjuhyökkäykset sekä malware-paketit, joiden nimet muistuttavat tunnettujen pakettien nimiä. Miten ohjelmistokehittäjät voivat pienentää npm-pakettien käyttöön liittyviä tietoturvariskejä valitessaan sopivia paketteja ohjelmistoprojektiin ja ylläpitäessään ohjelmistoprojektia vuosien ajan?

Opinnäytetyön aiheena on avoimen lähdekoodin ohjelmistokirjastojen tietoturva ja siihen liittyvien riskien ehkäisy erilaisilla keinoilla. Päädyin opinnäytetyön aiheeseen, koska minua kiinnostaa ohjelmistokehitykseen liittyvä tietoturva. Opinnäytetyön taustalla on toimeksiantajan tarve tarkentaa avoimen lähdekoodin ohjelmistokirjastojen käyttökriteereitä sekä löytää vaihtoehtoisia työkaluja npm-pakettien tarkistamiseksi tietoturvahaavoittuvuuksien varalta. Opinnäytetyön toimeksiantajana on HUS Tietohallinto.

Opinnäytetyön tavoitteena on kerätä tietoa avoimeen lähdekoodiin ja ohjelmistokirjastoihin liittyvistä tietoturvariskeistä ja sekä npm-pakettien valinnasta ja testaamisesta. Toissijaisena tavoitteena on myös tutkia olemassa olevia työkaluja npm-pakettien tunnettujen haavoittuvuuksien testaamiseen. Toimeksiannon perusteella asetettujen tavoitteiden perusteella kerätystä tiedosta kootaan opas.

Opinnäytetyössä käydään läpi yleisesti avoimen lähdekoodin ja ohjelmistokirjastojen määritelmää sekä luetellaan niiden käytön hyötyjä ja riskejä. Opinnäytetyössä luetellaan npm-pakettien käytön tietoturvariskejä sekä etsitään tietoa niin pakettien valinnan parhaista käytännöistä, kuin pakettien versioiden ylläpidosta omassa ohjelmistoprojektissa. Käytäntöjen painopiste on tietoturvariskien ehkäisyssä. Opinnäytetyössä esitellään kaksi työkalua: npm audit ja NPQ-komentorivityökalu.

Opinnäytetyön toiminnallisessa osuudessa läpikäydään prosessia lopputuotoksena tehdyn oppaan luomiseen, johon kuului myös pienimuotoinen kartoitus npm-pakettien tunnettujen

tietoturvahaavoittuvuuksien testaamiseen tarkoitetuista työkaluista. Valmista opasta voivat hyödyntää toimeksiantajan ohjelmistokehityksessä mukana olevat työntekijät. Valmis opas on yleisluontoinen, joten sitä voi hyödyntää myös muut ohjelmistokehittäjät.

Huomiona on, että mahdollisimman tietoturvallisen sovelluksen kehittäminen vaatii paljon muutakin kuin pelkästään ohjelmistokirjastojen tietoturvallisen käytön kehityksessä. Tässä opinnäytetyössä käsitellään kuitenkin tietoturvan osalta ensisijaisesti ainoastaan npm-pakettien tietoturvallista käyttöä ohjelmistoprojekteissa. Opinnäytetyössä ei käsitellä muiden ohjelmointikielten, kuten esimerkiksi Javan tai Pythonin, avoimen lähdekoodin kirjastoja.

1.1 Keskeiset käsitteet

Tässä osassa esittelen opinnäytetyön keskeisiä käsitteitä aakkosjärjestyksessä.

Avoin lähdekoodi: eng. open source. Ohjelmistot, joiden lähdekoodia kuka tahansa pääsee tutkimaan. Avoimen lähdekoodin ympärille on muodostunut yhteisö, joka kehittää ohjelmistoa edelleen. (von Kügelgen & Laukkonen 2020, 55.) Ohjelmistojen käyttöä ja hyödyntämistä ohjaavat lisenssit. Esimerkiksi joidenkin lisenssien ehtona on, että avoimen lähdekoodin pohjalle kehitetty koodi pitää antaa vapaasti muiden käyttöön.

Haavoittuvuus: eng. vulnerability. Heikkous tai aukko ohjelmistossa, jonka avulla hyökkääjä voi aiheuttaa vahinkoa ohjelmiston omistajalle, sen käyttäjille tai muille ohjelmistoon tukeutuville tahoille. Haavoittuvuuden voi aiheuttaa esimerkiksi suunnittelu- tai toteutusvirhe. (The OWASP Foundation s.a. a.)

Kirjasto tai **ohjelmistokirjasto**: eng. library. Kokoelma valmiiksi kirjoitettuja aliohjelmia, joita voi käyttää oman ohjelmiston toteutuksessa. (von Kügelgen & Laukkonen 2020, 47.)

Npm: 1) Paketinhallintajärjestelmä Node.js:lle. 2) npm-rekisteri: julkinen kokoelma avoimen lähdekoodin paketteja JavaScript-ohjelmointikieltä hyödyntäville projekteille. 3) npm-komentoriviohjelma, jolla ohjelmistokehittäjät voivat asentaa ja julkaista ohjelmistopaketteja. (Npm, Inc. s.a. a.) Viittaan viimeiseen käyttämällä termiä npm-komentorivityökalu.

Npm-paketti: eng. npm package. Yksittäinen npm-paketinhallinnan avulla asennettava ohjelmistokirjasto.

NPQ: Npm-pakettien testaukseen tarkoitettu työkalu, joka tarkistaa esimerkiksi asennettavan npm-paketin tunnetut haavoittuvuudet.

Ohjelmistokehitys: Prosessi, jolla luodaan tietokoneohjelmia. Sisältää usein ohjelmiston suunnittelun, koodauksen, testauksen, dokumentoinnin ja ylläpidon. (Ite Wiki s.a.)

Tietoturva: Tavoitetila, jossa tietojen ja tietojärjestelmien saatavuus, eheys ja luottamuksellisuus on varmistettu. (Digi- ja väestötietovirasto 15.11.2022, 17.)

2 Avoimen lähdekoodin ohjelmistokirjastot

Tässä luvussa käydään läpi avoimen lähdekoodin ohjelmistokirjastojen ja avoimen lähdekoodin määritelmää yleisesti. Tämän jälkeen luetellaan avoimeen lähdekoodin hyötyjä ja riskejä, liittyen niiden käyttöön ohjelmistoprojekteissa.

Avoimen lähdekoodin ohjelmistokirjastot määritellään Stream-nimisen yrityksen sivustolla kokoelmiksi ennaltaluotua koodia, jota kehittäjät saavat käyttää ja uudelleenjakaa vapaasti, riippuen lisenssiehdoista (Stream s.a.). Erilaisia ohjelmistokirjastoja on valtavasti ja niillä on lukemattomia eri käyttötarkoituksia. Valmiit ja ajan tasalla olevat ohjelmistokirjastot tuovat laatua ja varmuutta ohjelmistojen toiminnallisuuksiin. Tällaisia ominaisuuksia ovat esimerkiksi lomakkeiden tai päivämäärien ja ajan käsittely.

Tietoturvayritys Black Duckin helmikuussa 2025 julkaisemassa raportissa ”2025 Open Source Security and Risk Analysis Report” kerrottiin, että jopa 97 prosenttia Black Duckin auditoimista kaupallisten ohjelmistoprojektien koodikannoista sisälsi avointa lähdekoodia. Arvioitavia ohjelmistoprojekteja oli yhteensä 1658, joista keskimäärin yhdessä sovelluksessa oli 911 avoimen lähdekoodin ohjelmistokomponenttia. (Black Duck Software Inc 2025, 3–4.) Avoin lähdekoodi liittyy siis lähes jokaiseen moderniin ohjelmistoprojektiin jollain tavalla ja siksi sen arvioiminen ohjelmistokehityksessä on tärkeää.

Opinnäytetyön luvussa 4 sekä toiminnallisessa osuudessa tarkastellaan npm-paketteja, jotka ovat tietyntyyppisiä avoimen lähdekoodin ohjelmistokirjastoja. Niitä käytetään erityisesti JavaScript-, TypeScript-, React- ja Node.js -projekteissa. Npm-paketinhallinta on järjestelmä yksittäisten npm-pakettien hallitsemiseen ohjelmistoprojektissa. Node.js:n oman dokumentaation mukaan npm-paketinhallinnan rekisterissä oli syyskuussa 2022 yli 2,1 miljoonaa kirjastoa (Node.js 5.12.2024). Määrä on todennäköisesti noussut entisestään tämän jälkeen.

2.1 Yleisesti avoimesta lähdekoodista

Suomen avoimien tietojärjestelmien keskus – COSS ry kertoo avoimen lähdekoodin olevan tapa kehittää ja jakaa tietokoneohjelmistoja. Avoimen lähdekoodin ohjelmaa saa vapaasti käyttää, kopioida, muunnella ja jaella. (COSS ry s.a. a.)

Open Source Initiative (22.3.2007) määrittelee avoimen lähdekoodin kymmenen kohdan listan mukaan. Listan mukaan avoimen lähdekoodin ohjelmistojen jakeluehtojen on täytettävä seuraavat kriteerit:

- Ohjelmiston tulee olla vapaasti uudelleenjaettavissa.
- Ohjelmiston tulee sisältää vapaasti tarkasteltavissa ja muokattavissa oleva lähdekoodi.

- Lisenssin on sallittava muokkaukset ja johdetut teokset.
- Avoimen lähdekoodin lisenssin tulee sallia muokatusta lähdekoodista rakennetun ohjelmiston jakelun.
- Lisenssi ei saa syrjiä ketään henkilöä tai henkilöryhmää.
- Lisenssi se saa rajoittaa käyttämästä ohjelmaa tietyllä alalla tai tiettyyn pyrkimykseen.
- Ohjelmaan liittyvien oikeuksien tulee koskea kaikkia, joille ohjelma on jaettu uudelleen ilman, että näiden osapuolten tarvitsee hankkia lisälisenssiä.
- Avoimen lähdekoodin ohjelmiston lisenssi ei saa olla tuotekohtainen.
- Lisenssi ei saa rajoittaa muita ohjelmistoja esimerkiksi vaatimalla, että kaikkien samalla välineellä jaettavien ohjelmistojen on oltava avoimen lähdekoodin ohjelmistoja.
- Mikään lisenssin ehto ei saa perustua minkään yksittäiseen teknologiaan tai käyttöliittymän tyyliin.

Avoimen lähdekoodin ohjelmistojen vastakohtana voidaan pitää niin sanottuja suljettuja ohjelmistoja. COSS ry:n (s.a. b.) määritelmän mukaan suljettujen ohjelmistojen lähdekoodia ei julkaista, sillä se pidetään yleensä liikesalaisuutena. Suljetun ohjelman käyttäjä ei siis pääse tarkastamaan ohjelman toimintaa, kehittämään tai muuttamaan sitä.

2.2 Avoimen lähdekoodin hyötyjä

Avoimen lähdekoodin käytöllä ohjelmistoprojektissa on useita hyötyjä. Suomen avoimien tietojärjestelmien keskus – COSS ry listaa julkaisussaan ”White paper: Avoimen lähdekoodin osaamisen edistäminen Suomessa” (3.5.2023) avoimen lähdekoodin käytön hyödyiksi esimerkiksi julkishallinnon tehokkuuden lisäämisen ja sekä niin ympäristö- että ihmisresurssien säästämisen, koska samaa asiaa ei tarvitse tehdä moneen kertaan avoimen lähdekoodin avulla. Ilman avointa lähdekoodia ohjelmistokehittäjät joutuisivat jatkuvasti keksimään pyörän uudelleen ohjelmistoratkaisuja kehittäessään.

Stream-yrityksen sivustolla (s.a.) on listattu neljä hyötyä, miten juuri avoimen lähdekoodin ohjelmistokirjastot hyödyttävät sekä yksittäisiä ohjelmistokehittäjiä että suuria tiimejä:

- Ohjelmistokehittämiseen kuluvan ajan ja kustannusten säästö. Avoimen lähdekoodin kirjastojen käyttö on myös yleisesti ottaen ilmaista.
- Läpinäkyvyys, sillä ohjelmistokehittäjien on mahdollista tutkia lähdekoodia itse haavoittuvuuksien varalta.
- Yhteisöllinen kehitystyö, jonka vuoksi kirjastoilla on usein hyvä dokumentaatio ja niissä olevat haavoittuvuudet huomataan.
- Mahdollisuus kehittää taitoja. Esimerkiksi opiskelijat ja ohjelmistouransa alkuvaiheissa olevat henkilöt voivat oppia paljon avoimen lähdekoodin ohjelmistokirjastojen projekteista.

Vaikka edellisessä hyötylistauksessa lueteltiin avoimen lähdekoodin käytön olevan yleisesti ottaen ilmaista, sen vastuulliseen käyttöön voisi katsoa kuuluvan vastavuoroisuus. Esimerkiksi Antti Pohjola neuvoo Cloudamite-yrityksen blogissa, että kehittäjäyhteisöä voi tukea esimerkiksi taloudellisesti tai tarjoamalla omaa osaamistaan avoimen lähdekoodin projekteihin (Pohjola 28.2.2025). Tällainen lähestymistapa voisi hyödyttää avoimen lähdekoodin ratkaisuja hyödyntäviä organisaatioita tarjoamalla kehittäjille mahdollisuuden käyttää osan työajastaan avoimen lähdekoodin projektien parissa, joka edistäisi kehittäjien syvempää ymmärrystä projektista sekä kehittäisi ohjelmointitaitoja.

IT-asiantuntija Jukka Rahkonen mainitsee Maanmittauslaitoksen blogissa julkaisemassa kirjoituksessaan (8.6.2023), että avoimen lähdekoodin projekteista saattaa olla helpompaa löytää vertailukelpoista tietoa verrattuna suljettuihin ohjelmistoihin, sillä avoimen lähdekoodin projekteissa muun muassa virheraportit ovat kaikkien nähtävissä.

2.3 Avoimen lähdekoodin riskejä

Avoimen lähdekoodin hyödyntämiseen ohjelmistoprojektissa liittyy myös riskejä, jotka on hyvä tiedostaa ohjelmistokehityksessä. Tällaisia riskitekijöitä voivat olla esimerkiksi lähdekoodiin liittyvät lisenssit, projektin aikataulutus, avoimen lähdekoodin ratkaisujen integroiminen ohjelmistotuotteeseen sekä avoimen lähdekoodin tietoturva.

Avoimen lähdekoodin lisenssit määrittelevät, miten ohjelmistoja saa käyttää. Lisenssit saattavat esimerkiksi kieltää koodin käytön kaupalliseen tarkoitukseen. Tämän vuoksi ohjelmistoja valitessa tulee olla tietoinen lisenssien ehdoista. Esimerkiksi the Open Source Initiative listaa sivustollaan kymmeniä erilaisia avoimen lähdekoodin lisenssejä eri tarkoituksiin (Open Source Initiative s.a.).

Nick Vidal on luetellut Open Source Initiativen blogissa joulukuussa 2023 ClearlyDefined-yhteisöprojektin tilastoimia suosituimpia lisenssejä JavaScriptille npm-paketinhallinnan komponenteissa. Tilastoidut suosituimmat lisenssit olivat MIT, Apache 2.0 ja ISC. Muita npm-paketeissa käytössä olevia lisenssejä ovat blogin mukaan esimerkiksi BSD-3-Clause, GPL 3.0 ja LGPL 3.0. (Vidal 7.12.2023.)

Teoksessa ”Ohjelmistoprojektin sudenkuopat ja miten ne vältetään” mainitaan esimerkiksi, että projektissa käytössä olevan ohjelmistokirjaston valmistuminen ei välttämättä noudata sitä hyödyntävän yrityksen aikataulua (Juvonen 2018, 95). Tätä riskiä voinee pienentää pyrkimällä valitsemaan ohjelmistoprojektiin kirjastoja, joiden kehitys on tarpeeksi pitkällä sen käyttöön ottamiseksi. Toisaalta myös sellaisen kirjaston käyttö, jonka kehitys on jo lopetettu, on riski. Ohjelmistoprojektit saattavat olla tuotannossa jopa yli vuosikymmenen ajan, joten on tärkeää tarkastella ajoittain sitä, ovatko projektissa käytössä olevat kirjastot edelleen ylläpidettyjä.

Juvosen teoksessa riskiksi mainitaan myös avoimen lähdekoodin integrointi muuhun projektissa käytettävään koodiin. Esimerkiksi projektiin otettavasta kirjastosta saatetaan hyödyntää vain vähän toiminnallisuuksia, jolloin ohjelmistotuotteeseen tulee mukaan paljon ylimääräistä koodia. (Juvonen 2018, 95.) Ylimääräinen koodi taas saattaa lisätä tietoturvariskejä ja tekee koodikannasta monimutkaisempaa.

Avoimen lähdekoodin ohjelmistot ovat houkuttelevia kyberhyökkäyksille, koska kuka tahansa pääsee tutkimaan lähdekoodia. Siinä missä yksi taho voi etsiä haavoittuvuuksia korjatakseen ne, toinen voi etsiä hyödyntääkseen niitä. Avoimen lähdekoodin ohjelmistokirjastojen tietoturvaa käsitellään tarkemmin seuraavassa luvussa.

3 Avoimen lähdekoodin ohjelmistokirjastojen tietoturvauhat

Pureudun syvemmin avoimen lähdekoodin ohjelmistokirjastojen tietoturvaan. Aluksi avaan yleisesti tietoturvan käsitettä ja sen suhdetta ohjelmistokirjastoihin. Myöhemmin pohdin avoimen lähdekoodin ohjelmistokirjastoja tietoturvan näkökulmasta esittelemällä muutamia kirjastoihin kohdistuvia uhkia.

Usein tietoturva määritellään kolmen pointin – tiedon luottamuksellisuuden, eheyden ja saatavuuden – kautta. Esimerkiksi teoksessa ”Kaikki koodaa” nämä määritellään siten, että luottamuksellisuus tarkoittaa, ettei tieto saa vuotaa ulkopuoliselle. Eheys tarkoittaa, että järjestelmät ovat luotettavia, niissä olevia tieto säilyy muuttumattomana ja se on ajantasaisesti varmuuskopioitu. Saatavuus taas tarkoittaa pääsyä tietoon aina, kun sitä tarvitaan. (von Kügelgen, Laukkonen, & Hyvärinen 2020, 121.)

Oman sovelluksen tai muun ohjelmistotuotteen luottamuksellisuus ja eheys voi vaarantua, mikäli sovelluksen hyödyntämä ohjelmistokirjasto sisältää haavoittuvuuksia tai haitallista koodia. Näiden haavoittuvuuksien avulla on esimerkiksi mahdollista päästä luvattomasti käsiksi tietoihin tai muokata niitä ilman valtuutusta.

Ohjelmistokirjastot ovat kolmannen osapuolen luomaa koodia ja näin ollen kirjastojen käytössä tulee huomioida niiden tietoturvariskit. Esimerkiksi Mark Merkowin teoksessa ”Secure, resilient, and agile software development” mainitaan, että ohjelmistoprojektin riskianalyysien tulisi sisältää muun muassa avoimen lähdekoodin ohjelmistokirjastojen tarkka arviointi. Arvion tekemättä jättäminen saattaa jättää projektiin tietoturvahaavoittuvuuksia, joista kehitystiimi ei ole tietoinen. (Merkow, 81.)

Ohjelmistojen ympäristö muuttuu jatkuvasti. Ohjelmistoprojektin kehittämisen aikana turvalliseksi todettu ohjelmistokirjasto saattaa muuttua aikaa myöten tietoturvariskiksi, mikäli ohjelmistokirjastojen uusia saatavilla olevia versioita ei päivitetä omaan tuotteeseen tai kirjaston ylläpito loppuu. Tämän vuoksi on tärkeää seurata ajoittain projektissa käytettäviä kirjastoja ja niiden elinkaarta, sillä käytön jatkaminen ilman päivityksiä voi vaikuttaa järjestelmän tietoturvaan ja saatavuuteen.

3.1 Ohjelmistokirjastojen tunnetut haavoittuvuudet

Ohjelmistokirjastojen tunnetut haavoittuvuudet tarkoittavat kirjastoista löydettyjä haavoittuvuuksia, jotka ovat tunnistettu. Open Source Web Application Security Project (lyhennettynä OWASP) määrittelee haavoittuvuuden heikkoudeksi tai aukoksi ohjelmistossa, jonka avulla hyökkääjä voi aiheuttaa vahinkoa ohjelmiston omistajalle, sen käyttäjille tai muille ohjelmistoon tukeutuville tahoille. Haavoittuvuuden voi aiheuttaa esimerkiksi suunnittelu- tai toteutusvirhe. (The OWASP Foundation s.a. a.)

Kyberturvallisuuskeskuksen julkaisemassa verkkojutussa haavoittuvuus määritellään laajemmin miksi tahansa heikkoudeksi, ”joka mahdollistaa vahingon toteutumisen tai jota voidaan käyttää vahingon aiheuttamiseksi”. Verkkojutussa todetaan myös, että kaikessa teknologiassa on haavoittuvuuksia. (Kyberturvallisuuskeskus 23.03.2023.)

OWASP listasi vuonna 2021 kymmenen suurinta ohjelmistojen tietoturvauhkaa. Listassa kuudentena oli mainittu haavoittuvat ja vanhentuneet komponentit. Kategoria oli toiseksi suurin OWASPin teettämässä yhteisökyselyssä (The OWASP Foundation s.a. b.; The OWASP Foundation s.a. c.) Uhan päätyminen listalle tuo esiin sen mittavuutta ja merkittävyyttä, vaikkakin OWASPin listassa on huomioitu ohjelmistokirjastojen lisäksi myös esimerkiksi palvelimet, tietokannan hallintajärjestelmät ja APIt eli ohjelmointirajapinnat.

Opinnäytetyön kirjoittamisen aikaan OWASPilla oli työn alla uusi Top 10 -raportti, joka julkaistaan vuoden 2025 aikana. (OWASP Top Ten Project s.a.). Mikäli uusi raportti olisi ehditty julkaista jo keväällä, olisi ollut mielenkiintoista selvittää, onko kyseessä olevan uhan merkittävyys muuttunut viimeisen neljän vuoden aikana suhteessa muihin tietoturvauhkiin.

Linux Foundationin ja Harvardin yhteistutkimus Census III sekä sitä edeltänyt tutkimus Census II vuodelta 2022 nostavat esiin ongelman, että kaupallisissa ja tuotannossa olevissa ohjelmistoratkaisuissa on laajalti avoimen lähdekoodin kirjastoja, joita ei enää ylläpidetä tai tueta. Tällaisista kirjastoista puuttuu usein päivityksiä tunnettujen tietoturvaongelmien korjaamiseksi. (Nagle ym. 2024, 25–26; Perlow 2022.) Oman palvelun tai sovelluksen elinkaari voi olla hyvinkin erimittainen verrattuna kirjaston elinkaareen, jonka vuoksi kirjastoa käyttävän kehittäjätahon tulee varautua siihen, että ohjelmistokirjasto joudutaan korvaamaan jossain vaiheessa toisella.

Census II ja Census III -tutkimusraportit painottavat, että organisaatioiden tulisi tietää, mitä avoimen lähdekoodin kirjastoja ne ovat ottaneet käyttöön. Organisaatioiden tulisi myös tunnistaa, missä näitä koodipohjia ylläpidetään ja päivitetään ajan mittaan. (Nagle ym. 2024, 25–26; Perlow 2022.) Aikaa myöden tukematta jääneet ohjelmistokirjastot muodostavat tietoturvariskejä, sillä niistä saatetaan löytää uusia haavoittuvuuksia tai ne eivät ole enää muuten yhteensopivia ohjelmistoprojektiin.

Ohjelmistokirjastojen tunnettujen haavoittuvuuksien testityökalut hyödyntävät tietokantoja, jotka sisältävät tunnetut haavoittuvuudet kunkin ohjelmistokirjaston version osalta. Esimerkiksi npm-komentorivityökalu käyttää GitHub Advisory -tietokantaa npm-pakettien tunnettujen haavoittuvuuksien listaamiseen (Thomson 7.10.2021). Komentorivityökalua testataan opinnäytetyön toiminnallisessa osuudessa.

3.2 Muita ohjelmistokirjastojen tietoturvauhkia

Tunnettujen haavoittuvuuksien lisäksi ohjelmistokirjastoilla on muitakin tietoturvauhkia, kuten esimerkiksi typosquatting, toimitusketjuhyökkäykset sekä kehittäjäyhteisöön liittyvät uhat.

Typosquatting on hyökkäystapa, jossa hyökkääjätaho onnistuu julkaisemaan ”oikeaa” pakettia muistuttavalla nimellä haitallista koodia sisältävän paketin npm-rekisteriin. Jos kehittäjä kirjoittaa vahingossa haitallisen paketin nimen pakettia asentaessaan, hän asentaa oikean paketin sijasta haitallisen paketin. Esimerkiksi Snykin työntekijä Liran Tal kirjoitti yrityksen blogissa vuonna 2021 tapauksesta, jossa hyökkääjätaho oli julkaissut ”crossenv”-nimisen paketin, jäljitelläkseen pakettia ”cross-env” (Tal 12.1.2021). Myös keväällä 2025 Socket-yrityksen blogissa kerrottiin kuudesta Lazarus-hakkeriryhmän julkaisemasta npm-paketista, joiden nimet muistuttivat laajalti tunnettujen pakettien nimiä ja jotka sisälsivät haitallista koodia (Boychenko 11.3.2025).

HackerOne-yrityksen verkkoartikkelissa ohjelmiston toimitusketjuhyökkäys määritellään hyökkäykseksi, joka kohdistuu organisaation ohjelmistoihin tai palveluntarjoajiin sen digitaalisessa toimitusketjussa. Artikkelissa tarkennetaan avoimen lähdekoodin ohjelmistokirjastojen osalta, että hyökkääjät yrittävät joko hyödyntää paikkaamattomia haavoittuvuuksia tai peukaloida kirjastoja kehityksen tai jakelun aikana. (HackerOne s.a.)

Näiden määritelmien mukaan typosquatting ja toimitusketjuhyökkäykset limittyvät keskenään hieman, sillä typosquattingia voidaan pitää eräänlaisena toimitusketjuhyökkäyksenä. Yksittäisen kehittäjän voi olla vaikeaa suojautua typosquattingilta asentaessaan komentorivin avulla uutta npm-pakettia ilman erillistä työkalua, sillä hyökkäystapa perustuu inhimilliseen näppäinvirheeseen.

Linux Foundationin ja Harvardin Census III -yhteistutkimus nosti esiin esimerkiksi mahdollisen turvallisuusongelman, että on olemassa laajastikin käytettyjä, suosittuja ohjelmistokirjastoja, joita ylläpitää vain muutama ohjelmistokehittäjä (Nagle ym. 2024, 5, 24). Jos vain muutama ylläpitää projektia, mahdollisia tietoturvaongelmia ei välttämättä huomata tai niitä ei paikata nopeasti. Tällaisessa projektissa kehittäjän tunnusten kaappaaminen voisi johtaa siihen, että tunnusten kaappaaja voi helposti ylläpitäjäksi tekeytyneenä lisätä kirjastoon haitallista koodia muiden huomaamatta. Kirjaston käyttäjätahot voivat tietyiltä osin suojautua ongelmalta joko harkitsemalla, käytetäänkö kirjastoa riskit huomioon ottaen. Toinen keino voisi olla kirjastoa hyödyntävän kehittäjän osaamisen tarjoaminen kirjastoprojektin ylläpitoon.

4 Tietoturvalliset käytännöt npm-pakettien käyttöön

Esittelin edellisessä luvussa yleisesti muutamia ohjelmistokirjastoihin liittyviä tietoturvauhkia. Tässä luvussa luettelen parhaita käytäntöjä npm-pakettien valitsemiseen, käyttöönottovaiheeseen ja ylläpitoon tietoturvan näkökulmasta, uhkien torjumiseksi.

Vaiheet on pyritty esittämään kronologisessa järjestyksessä. Todellisuudessa vaiheiden jaottelu oman ohjelmistoprojektin hallinnassa ei ole aivan niin yksinkertaista. Ohjelmistoprojektissa saatetaan esimerkiksi joutua syystä tai toisesta ottamaan jokin jo käytössä oleva kirjasto pois käytöstä ja vaihtamaan se toiseen, jolloin palataan tietyn kirjaston osalta takaisin ohjelmistokirjaston valintaan. Kirjastoversioiden tietoturvan arviointi ja päivityskäytäntöjen toteuttaminen kestää taas käytännössä koko ohjelmistoprojektin elinkaaren ajan.

4.1 Npm-paketin valitseminen

Jotta ohjelmistokirjastojen käyttö olisi onnistunutta ohjelmistoprojektissa, ensimmäiseksi tulee harkita, millaisia kirjastoja ohjelmistotuotteessa tarvitaan tai tarvitaanko sellaista ollenkaan tietyn ominaisuuden kehittämiseksi. Seuraavaksi luettelen parhaita käytäntöjä npm-pakettien valitsemiseen. Käyn myös lyhyesti läpi muutamia keinoja npm-pakettien etsimiseen.

Ennen ohjelmistokirjaston etsimisen aloittamista tiimin tulisi selvittää, tarvitaanko ohjelmistokirjastoa tietyn ominaisuuden toteuttamiseksi. Esimerkiksi freeCodeCampin julkaisemassa artikkelissa kehittäjä David Jaja suosittelee ohjelmistokirjastojen käyttöä muun muassa reititykseen, lomakkeen validointiin, animointiin ja muihin UI-komponentteihin. Jajan mielestä kirjastojen käyttö ei ole välttämätöntä esimerkiksi yksinkertaisiin aputoimintoihin, erikoistuneen tai toimialuekohtaisen logiikan toteuttamiseksi ja suorituskyvyn optimointiin. (Jaja 24.6.2024.) Koska ohjelmistokirjaston käyttäminen sisältää aina jonkintasoisen riskin, houkutteleva päätelmä tietoturvan kannalta voisi olla jättää kirjastot pois kokonaan omasta projektista. Tämä ei kuulosta kuitenkaan realistiselta, sillä omaan koodiin eksyy myös todennäköisesti virheitä.

Gleb Krishin ohjeistaa Dev Communityn blogissa vuonna 2022 ensimmäiseksi luomaan listan potentiaalisista ohjelmistokirjastoista tiettyyn tarkoitukseen. Seuraavaksi Krishin ohjeistaa tutkimaan kirjastojen dokumentaatiot ja vertaamaan niitä ohjelmiston vaatimuksiin. Tämän jälkeen ohjeistetaan tutkimaan projektin eli ohjelmistokirjaston lähdekoodia, tarkistamaan projektin ylläpitäjätahon, tutkimaan projektin kysymykset ja ongelmat (eng. ”issues”) ja niiden käsittelymenetelmät sekä tutkimaan projektin kiteytyssivun (eng. ”insights”). Viimeiseksi Krishin ohjeistaa luomaan taulukon kirjastojen vertailua varten. (Krishin 17.10.2022.)

Adrian Bece lähtee liikkeelle kirjoittamassaan ohjeistuksessa erilaisella näkökulmalla. Bece ohjeistaa ennen npm-paketin asentamista tarkistamaan ensimmäiseksi paketin haavoittuvuudet ja sitten ylläpidon, mahdolliset bugit ja koon. Viimeiseksi Bece kehottaa vertaamaan kirjastoa muihin vastaavanlaisiin. Bece oli myöhemmin lisännyt julkaisuunsa Idris Rampurawalan kommentin, jossa kehotettiin tarkistamaan myös paketin version ja sen riippuvuuksien versioiden yhteensopivuus verrattuna omaan projektiin. (Bece 12.7.2020.)

Krishinin ja Becen ohjeistukset täydentävät toisiaan ja toisaalta limittyvät hieman keskenään. Siinä missä Bece aloittaa suoraan kirjaston tietoturvan arvioinnilla, Krishin kartoittaa sopivat kirjastot yleisellä tasolla. Vastaavasti Bece kehottaa vertaamaan kirjastoa muihin vastaaviin vasta, kun tutkittavan kirjaston tietoturva on arvioitu. Näistä kahdesta strategiasta Becen taktiikka voi olla järkevämpi kirjastojen tietoturvan arvioinnin kannalta. Kirjaston ominaisuuksien arviointiin voi mennä paljon aikaa hukkaan, jos vasta myöhemmin käy ilmi, ettei potentiaalinen kirjasto olekaan sopiva tietoturvan kannalta.

On monia tapoja etsiä ohjelmistokirjastoja. Npm:n virallisen sivuston eli npmjs.comin oma hakutoiminto on melko yksinkertainen. Npm:n dokumentaation mukaan hakutulokset näytetään npm-paketin otsikon, kuvauksen, ReadMe-tiedoston ja avainsanojen perusteella. Dokumentaation mukaan täysin uusien pakettien ilmestymisessä hakutuloksiin saattaa mennä kaksikin viikkoa. Vanhentuneet paketit eivät näy enää hakutuloksissa. (Npm, Inc. 13.2.2025) Vanhentuneiden npm-pakettien piilotus hakutuloksista voi olla hyvä asia tilanteissa, joissa etsitään uutta kirjastoa projektiin. Npm-pakettien piilotus hakutuloksista saattaa kuitenkin aiheuttaa ongelmia potentiaalisissa tilanteissa, jolloin pitäisi tarkistaa vanhentuneen paketin dokumentaatio. Vanhentuneiden pakettien sivut löytyvät ulkoisen hakukoneen, kuten Googlen, avulla.

Npmjs.com-sivuston lisäksi on myös olemassa muitakin sivustoja npm-pakettien etsimiseen ja vertailuun, kuten esimerkiksi npm.io. Paketteja vertaillaan ja esitellään myös useissa verkkoartikkeleissa. Myös sisältöä tuottavan tekoälyn, kuten Microsoft Copilotin, avulla on mahdollista etsiä ohjelmistokirjastoja tiettyä ongelmaa varten. On kuitenkin tärkeää tarkistaa, että npm-paketti todella löytyy npm:n rekisteristä ja arvioida sen turvallisuus. Erityisesti tekoälyn käytön osalta pitää varmistaa, ettei tekoälymalli suosittele vanhentunutta npm-pakettia tai muuten virheellisesti kirjastoa, jota ei tulisi käyttää. Esimerkiksi The Register uutisoi huhtikuussa 2025 ilmiöstä, jossa ohjelmointiin tarkoitettujen tekoälyavustajien ”hallusinoimia” kirjastojen nimiä käytetään hyväksi hyökkäyksissä (Claburn 12.4.2025).

4.2 Npm-pakettien käyttöönotto

Kun ohjelmistoprojektiin sopivat npm-paketit ovat kartoitettu, ne voidaan ottaa käyttöön. Kuitenkin mikään ei välttämättä estä ohjelmistokehittäjää asentamasta npm-paketteja suoraan sen enempää arvioimatta niiden tietoturvaa, vaikkapa kokeillakseen uutta kirjastoa tai asentaakseen ulkomuistista aiemmin käyttämänsä kirjaston. Tämän vuoksi on tärkeää pohtia keinoja, miten ohjelmistokirjastojen käyttöönottoon liittyviä riskejä voisi pienentää.

Npm-komentorivityökalun komento ”install” asentaa kaikki package.json-tiedostossa listatut riippuvuudet ohjelmistoprojektiin. Komento ”install <paketin nimi>” asentaa projektiin tietynnimisen npm-paketin. (Npm, Inc 11.9.2024.) Npm-komentorivityökalussa ei tällä hetkellä ole erikseen varmistusta, haluaako kehittäjä asentaa juuri tietyn kirjaston. Se tarkistaa asennettujen kirjastojen haavoittuvuudet vasta sen jälkeen, kun uusi paketti on asennettu. Yksittäisen npm-paketin asentamiseen ei mene muutamaa sekuntia kauemmin aikaa, joten latauksen pysäyttäminen kesken kaiken on hankalaa manuaalisesti ilman erillistä työkalua.

Stacklok-yrityksen blogissa Edward Thomson kirjoittaa, että on vaikeaa olla asentamatta koodia, jota kehittäjä ei tunne, sillä keskimääräisessä JavaScript-projektissa on yli 500 riippuvuutta. Thomson mainitsee, että riippuvuuksien suuren määrän vuoksi on lähes mahdotonta auditoida riippuvuuksien riippuvaisuuksia eli niitä kirjastoja, joita projektiin asennetut kirjastot käyttävät. (Thomson 3.3.2024)

Yksi Thomsonin ehdottama keino npm-pakettien asennukseen olisi käyttää ”--ignore-scripts”-parametriä. Esimerkiksi kirjasto next asennettaisiin seuraavalla komennolla: ”npm install --ignore-scripts next”. Mikäli npm-paketti sisältäisi haitallista koodia pre- tai postinstall -skripteissä, skriptejä ei suoritettaisi tämän komennon avulla. Blogikirjoituksessa huomautetaan kuitenkin, että joissain tapauksissa riippuvuudet tarvitsevat oikeasti pre- tai postinstall -skriptejä. (Thomson 3.3.2024). Parametrin käyttö saattaisi siis lisätä turvallisuutta yhtäältä, mutta luoda uuden ongelman toisaalla.

4.3 Ohjelmistoprojektin ylläpito ja npm-paketit

Luettelen muutamia keinoja ohjelmistoprojektin ylläpitoon liittyvistä käytännöistä ohjelmistokirjastoihin liittyen. Käytännöt ovat sekä tiimin vastuunjakoa, kirjastoihin liittyvää dokumentointia, että teknisiä ohjeita npm-pakettien versioiden päivitykseen. Erityisesti projektin tuotantovaiheeseen mennessä tiimien ja kehittäjien tulisi määritellä vastuut ja käytännöt kirjastoversioiden seuraamiseksi ja päivittämiseksi.

Kyberturvallisuuskeskus ohjeistaa ohjelmistoriippuvuuksien riskienhallintaa käsittelevässä verkkoartikkelissaan (19.2.2025) ensimmäiseksi kartoittamaan, mitä kirjastoja on käytössä sekä

ehdottaa Software Bill of Materials (SBOM) -luettelon hyödyntämistä. SBOMin tarkoituksena on luetteloida käytössä olevat kirjastot ja niiden versiot, jotta niitä voidaan hallinnoida tehokkaasti. Esimerkiksi npm-komentorivityökalulla voi koostaa SBOMin JSON-muodossa komennolla ”npm sbom”. (Npm, Inc. 27.9.2023.) Tässä opinnäytetyössä SBOMin käyttöön toimeksiantajan ympäristössä ei luotu tarkempia käytänteitä, vaikkakin asiaa sivuttiin eräässä toimeksiantajan kanssa käydyssä keskustelussa ja vaihtoehtoja SBOMin hyödyntämiselle tulevaisuudessa pohdittiin lyhyesti.

Kyberturvallisuuskeskuksen ohjeessa pyydetään myös arvioimaan riskejä ja priorisoimaan niitä sillä perusteella, kuinka tärkeä komponentti on ohjelmiston toiminnalle ja kuinka haavoittuvia ne ovat hyökkäyksille. Haavoittuvuuksien hallitsemiseksi ja ajoissa korjaamiseksi tulisi määrittää ylläpitovastuut. (Kyberturvallisuuskeskus 19.2.2025.) Npm-pakettien käytön suhteen tulisi siis esimerkiksi määrittää, kuka tiimistä vastaa pakettien versioiden päivittämisestä uudempaan ja missä tilanteissa päivitys tulisi tehdä. Lisäksii tiimin tulisi pohtia etukäteen mahdollisia toimintaohjeita tilanteisin, joissa kirjasto jouduttaisiin ottamaan kokonaan pois käytöstä.

Halusin selvittää, kuinka usein npm-kirjastojen versioita suositellaan päivitettäväksi omassa projektissa. Julkisia ammattilaisten suosituksia siitä, kuinka usein npm-riippuvuuksia kannattaisi tarkistaa, ei kuitenkaan löytynyt. Joko kehittäjät eivät halua julkisesti kertoa, kuinka usein he päivittävät kirjastoja tai vaihtoehtoisesti tarkistuksen hoitaa jatkuvasti automaattinen työkalu. Säännöllisellä tarkistuksella ehkäistään ongelmaa, että ohjelmistoprojektiin jäisi pitkäksi aikaa roikkumaan haavoittuvuuksia päivittämättömien kirjastojen vuoksi. Erityisesti kriittisten haavoittuvuuksien paikkaaminen vaatii nopeaa toimintaa.

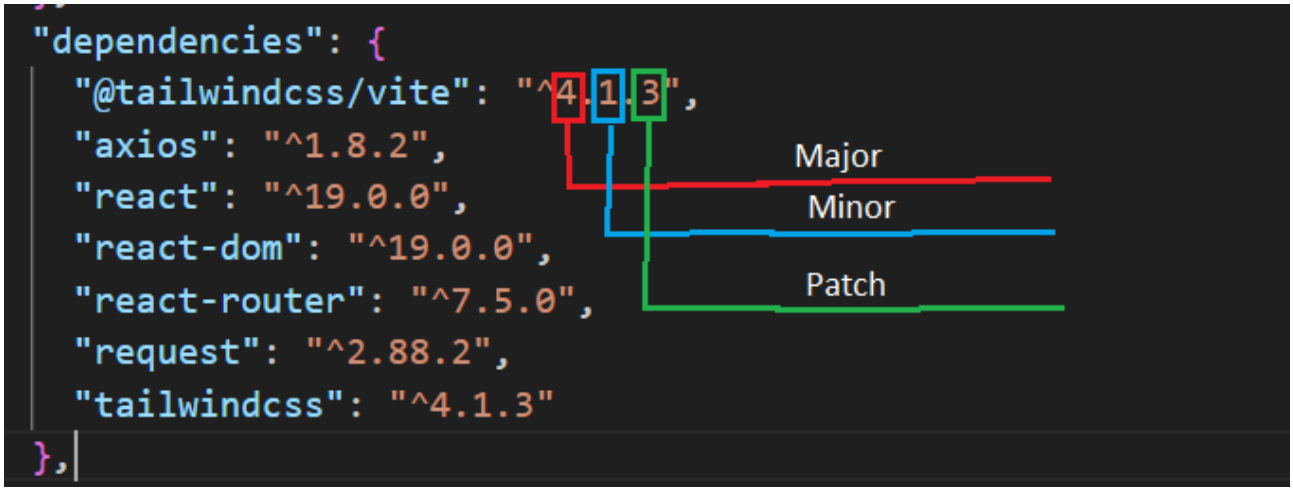
Kyberturvallisuuskeskus ohjeistaa ohjelmistoriippuvuuksien jatkuvaan testaamiseen automaattisten työkalujen avulla. Esimerkiksi Software Composition Analysis (SCA) -työkalut skannaavat kolmannen osapuolen komponentteja ja kirjastojen mahdollisia haavoittuvuuksia. (Kyberturvallisuuskeskus 19.2.2025.) Tässä opinnäytetyössä tutkitaan tarkemmin kahta SCA-työkalua: npm auditia ja NPQ:ta.

Npm hyödyntää semanttista versiointia ohjelmistokirjastojen versioiden hallintaan, jonka vuoksi kehittäjällä on hyvä olla tiedossa semanttisen versioinnin perusteet. Semanttinen versioinnin dokumentaatiosivu (semver s.a.) määrittelee versioinnin olevan kokoelma sääntöjä, ja sen ohjeistamana päivitysversiot koostuvat kolmen numeron sarjasta:

- Ensimmäinen on ”Major”, jota käytetään, kun tehdään yhteensopimattomia muutoksia APIin eli rajapintaan.
- Toinen on ”Minor”, jota käytetään, kun kirjastoon lisätään toiminnallisuutta taaksepäin yhteensopivalla tavalla.

- Kolmas sarjasta on ”Patch”. Patch-versiota käytetään, kun lisätään taaksepäin yhteensopivia vikakorjauksia.
- Kokonainen versiointinumero muodostuu siis osista Major.Minor.Patch.

Esimerkiksi alla olevassa havainnoivassa package.json-tiedostosta otetussa kuvankaappauksessa (kuva 1), kirjaston ”@tailwindcss/vite” versio on 4.1.3. Sen Major-versio on 4, Minor-versio 1 ja Patch-versio 3.



Kuva 1: Havainnoiva kuva npm-pakettien semanttisesta versioinnista

Natalie Pina ohjeistaa vuonna 2022 freeCodeCampin julkaisemassa artikkelissa ensimmäiseksi kartoittamaan, mitkä npm-paketit tarvitsevat päivityksen. Kirjastot voi päivittää hänen mukaansa joko yksitellen tai yhdessä erässä. Artikkelissa suositeltiin päivittämään Major-päivitysversiot erikseen ja testaamaan aina ohjelmiston toimivuus versiopäivitysten jälkeen. (Pina 5.6.2022.)

Pina ehdottaa tarkistamaan komentorivityökalulla komennolla ”npm outdated” tiedon siitä, mihin kaikkiin projektin ohjelmistokirjastoihin on saatavilla päivitys. Komento ”npm update” päivittää kerralla kaikki kirjastot, joihin on saatavilla Minor- tai Patch-päivitys. (Pina 5.6.2022.) Oletuksena package.json-tiedosto on konfiguroitu siten, että versionumerosarjan edessä on ”^”-merkki viittaamassa, että käytössä oleva versio voidaan päivittää isompaan Minor- tai Patch-versioon (node-semver s.a.).

5 Npm-pakettien tunnettujen haavoittuvuuksien testityökalut

Opinnäytetyön yhtenä tavoitteena oli löytää sopivia työkaluja npm-paketinhallintajärjestelmän kautta asennettavien avoimen lähdekoodin ohjelmistokirjastojen tunnettujen tietoturvahaavoittuvuuksien testaamiseen toimeksiantajan kehitysympäristössä. Tässä luvussa esittelen kaksi työkalua, joilla voi hallinnoida npm-paketteja sekä testata, onko niissä tunnettuja haavoittuvuuksia. Työkalut ovat npm audit -komentorivityökalu ja NPQ-komentorivityökalu. Työkaluissa on suurin piirtein sama toimintalogiikka, mutta niiden välillä on joitain eroavaisuuksia. Esiteltyjen työkalujen testauksesta ja tuloksista kerrotaan tarkemmin alaluvuissa 6.5 ja 6.6.

Markkinoilla on paljon työkaluja ohjelmistokirjastojen tunnettujen haavoittuvuuksien tunnistamiseen. Suurin osa opinnäyteprosessin aikana löydetyistä työkaluista on kaupallisia, kuten Snyk, Veracode ja Mend Renovate. Avoimena lähdekoodina julkaistuja ilmaisia työkaluja on myös muutamia, kuten OWASP Dependency Check ja Red Hat Dependency Check -lisäosa Visual Studio Code -tekstieditoriin.

5.1 Npm audit -komentorivityökalu

Npm audit on npm-komentorivityökaluun sisällytetty komento ja työkalu, jolla voi tarkistaa asennettujen npm-pakettien tunnettuja haavoittuvuuksia. (Npm, Inc. 4.4.2024.; rishavanand2k21 25.4.2024.) Npm:n blogin mukaan npm audit on sisältynyt npm-komentorivityökaluun paketinhallinnan versiosta 6 lähtien. Versio 6 julkaistiin vuonna 2018 eli kyseessä on jo melko pitkään olemassa ollut työkalu. (@adam-npm 8.5.2018.)

Työkalu valittiin tarkasteltavaksi opinnäytetyöhön, koska sitä voidaan pitää perustyökaluna npm-paketteja käyttävässä ohjelmistoprojektissa. Npm audit oli esimerkiksi listattu toimeksiantajan eräässä ohjelmistoprojektissa yhdeksi käytettävistä työkaluista. Npm audit toimii tässä työkalujen arvioinnissa vertailupohjana NPQ-työkalulle.

Npm audit -komento lähettää kuvauksen ohjelmistoprojektissa määritetyistä riippuvuuksista ja niiden versioista oletusrekisteriin ja pyytää raporttia tunnetuista haavoittuvuuksista. Mikäli haavoittuvuuksia löytyy, raportti sisältää tietoa kyseessä olevista haavoittuvuuksista, niiden vakavuustasot ja korjaussuositukset niille. (Npm, Inc. 4.4.2024; rishavanand2k21 25.4.2024.)

Npm:n dokumentaation mukaan parametrillä ”--audit-level” voidaan määritellä työkalun kirjastosta löytämän haavoittuvuuden vähimmäistason, joka aiheuttaa komennon epäonnistumisen. Tason määrittely on dokumentaation mukaan hyödyllistä CI-putkessa. (Npm, Inc. 4.4.2024.) Mikäli audit-level olisi määritelty vaikkapa tyypiksi korkea (englanniksi ”high”), komento epäonnistuisi, jos

jossain projektissa olevassa kirjastossa löytyisi tunnettu haavoittuvuus, jonka vaikuttavuustasoksi on määritelty korkea tai kriittinen (englanniksi ”critical”).

5.2 NPQ-komentorivityökalu

NPQ-komentorivityökalun kuvauksen mukaan työkalun on tarkoitus auttaa ohjelmistokehittäjää asentamaan npm-paketteja turvallisemmin verrattuna npm -komentorivityökaluun. NPQ:n dokumentaatio kuitenkin huomauttaa turvallisuuden osalta, että ”ei ole taattua turvallisuutta; haitallinen tai haavoittuva paketti saattaa silti olla olemassa, jossa ei ole julkisesti julkaistuja tietoturva-aukkoja ja joka läpäisee npq:n tarkastukset”. NPQ:n tarkoitus ei myös ole olla kokonaan npm-komentorivityökalun korvaaja, vaan lisätä sen päälle muutamia muita tarkistuksia.

NPQ-komentorivityökalu on julkaistu npm-pakettina. NPQ:n lisenssi on Apache-2.0. Sen kehittäjä ja julkaisija on Liran Tal. (Npm, Inc. 12.9.2024.) Työkalu valittiin tarkasteltavaksi opinnäytetyöhön, koska haluttiin selvittää, olisiko NPQ hyödyllinen täydennys npm auditin lisäksi.

Työkalun kehittäjä Tal työskentelee tietoturvayritys Snykillä kehittäjäsuhteista vastaavana johtajana. (Snyk s.a. a.) NPQ-työkalu käyttääkin Snykin haavoittuvuustietokantaa asennettavan npm-paketin versiossa olevien tunnettujen haavoittuvuuksien tarkistamiseen. NPQ:n dokumentaation usein kysytyt kysymykset -osiossa lukee, että Snykin API-avaimen käyttö ei ole välttämätöntä NPQ-työkalun käyttöön, mutta työkalu ei tällöin tarkista haavoittuvuuksia Snykin tietokannasta. (Npm, Inc. 12.9.2024.)

Testaamatta työkalua dokumentaatiosivulta ei selvinnyt, tarkistaako NPQ haavoittuvuuksia miten-kään, mikäli kehittäjällä ei ole syystä tai toisesta Snykin API-avainta. Opinnäytetyössä tehdyn testauksen perusteella NPQ kertoo Snykin tietokannan sivustolta löytyneen haavoittuvuuksia, mutta kehittäjän pitää erikseen tutkia työkalun tarjoaman URL-osoitteen takana oleva tieto, mikäli hän haluaa kuvauksen tunnetusta haavoittuvuudesta. Testauksessa ei käytetty Snykin API-avainta.

NPQ-työkalun latausmäärät ovat pysyneet melko pieninä, vaikkakin paketin ensimmäinen versio on julkaistu jo seitsemän vuotta sitten. Npm trends -sivuston mukaan NPQ:n viikoittaiset latausmäärät ovat vaihdelleet viimeisen vuoden aikana pääsääntöisesti muutaman sadan ja tuhannen välillä. Työkalun viimeisin versio on julkaistu noin puoli vuotta sitten opinnäytetyön kirjoitushetkellä. (npm trends s.a.)

6 Lopputuotoksen suunnittelu ja toteutus

Tässä luvussa käyn läpi opinnäytetyön aikana tehtyä kehittämisprosessia avoimen lähdekoodin ohjelmistokirjastojen tietoturvariskien tunnistamiseen sekä niiden ehkäisyyn, jonka lopputuotoksena valmistui ohjeet npm-pakettien tietoturvalliseen käyttöön. Määrittelen aluksi opinnäytetyön lähtötilanteen ja kehittämistehtävän. Seuraavaksi kerron lähdeaineiston keruusta ja tutkittavien työkalujen valinnasta, jonka jälkeen etenen ohjeiden suunnitteluvaiheeseen ja niiden luontiin. Tämän jälkeen kerron npm-pakettien tunnettujen haavoittuvuuksien testaamiseen tarkoitettujen työkalujen konfiguroinnista ja testaamisesta. Viimeiseksi esittelen tuloksia.

6.1 Lähtötilanne – tavoite, ongelmat ja kehittämistehtävä

Opinnäytetyöni toimeksiantajana on sosiaali-, terveys- ja pelastusalan julkisyhteisö HUS-yhtymä, tai tarkemmin HUS Tietohallinto. Opinnäytetyön taustana oli toimeksiantajan tarve tarkentaa kriteereitä avoimen lähdekoodin ohjelmistokirjastojen käyttöön ja testaamiseen ohjelmistokehityksessä toimeksiantajan kehitysympäristössä. Tavoitteena oli löytää keinoja, miten ohjelmistokirjastojen käyttöön liittyviä tietoturvariskejä voi pienentää. Ennestään kirjastoja testattiin esimerkiksi satunnaisesti skannaustyökaluilla, mutta niiden valintaan ja testaamiseen ohjelmistokehityksen aikana ei ollut luotu yhteisiä käytäntöjä eri projektien ja tiimien välillä, vaan ne riippuivat tiimissä työskentelevien osaamistasosta ja kiinnostuksesta kirjastoja kohtaan.

Tavoitteenani oli etsiä tietoa avoimeen lähdekoodiin ja ohjelmistokirjastoihin liittyvistä tietoturvariskeistä ja määrittää näiden pohjalta kriteerit kirjastojen valintaan sekä testaamiseen. Tutkin myös pienimuotoisesti olemassa olevia työkaluja ohjelmistokirjastojen testaamiseen. Näistä ratkaisuista pyrin poimimaan sopivia vaihtoehtoja toimeksiantajan kehitysympäristöön. Myöhemmin keskityin tarkentamaan ohjeistuksia npm-paketteihin, joita käytetään esimerkiksi JavaScript-, TypeScript- ja React-projekteissa. Rajaus tehtiin siksi, että npm-paketit olivat minulle kehittäjänä ennestään kaikista tutuimpia kolmannen osapuolen kehittämistä avoimen lähdekoodin komponenteista. Npm-paketteja käytetään myös jonkin verran toimeksiantajan ohjelmistoprojekteissa. Useampaan ohjelmointikielen ohjelmistokirjastoihin painottuminen olisi tehnyt opinnäytetyöstä liian laajan.

Toimeksiantajan toiveena oli myös löytää työkaluja, joilla voidaan tarkistaa, onko ohjelmistoprojektissa olevilla kirjastoilla tunnettuja tietoturvahaavoittuvuuksia. Jos työkalu havaitsee, että tietoturvaan liittyvät kriteerit eivät täyty, se lähettää varoituksen ohjelmistokehittäjälle. Toiveena oli myös, että työkalu hyödyntäisi automaatiotestausta.

Tavoitteiden perusteella opinnäytetyön lopputuotoksena luotiin ohjeet. Oppaan toivottiin selkeyttävän avoimen lähdekoodin ohjelmistokirjastojen valintaan liittyviä kriteereitä sekä säästävän

organisaation ohjelmistokehittäjien aikaa ja vaivaa. Opinnäytetyön aikana tehdyn ohjeistuksen toivotaan lisäävän ohjelmistokehittäjien tai muuten ohjelmistokehityksessä mukana olevien työntekijöiden ymmärrystä avoimen lähdekoodin kirjastojen käytöstä erityisesti tietoturvan näkökulmasta.

Opinnäytetyön kohderyhmänä ovat toimeksiantajan työntekijät, joiden työtehtäviin liittyy ohjelmistokehittämistä tai ohjelmistokehitykseen liittyvien työkalujen hallintaa. Tuotoksen yleisluonteisuuden vuoksi opinnäytetyön lopputuotosta voi hyödyntää kuitenkin myös yleisesti ohjelmistokehittäjät, jotka käyttävät tai suunnittelevat käyttävänsä avoimen lähdekoodin komponentteja ja erityisesti npm-paketteja projekteissaan.

6.2 Tiedonkeruu ja tutkittavien työkalujen valinta

Etsin tietoa npm-pakettien tietoturvauhista ja keinoja, miten niitä voisi välttää. Keskustelin myös toimeksiantajan työntekijöiden kanssa selvittääkseni, millaisia asioita uutta pakettia käyttöönottaessa tulisi huomioida. Keskusteluissa nousi esiin esimerkiksi lisenssit, paketin taustalla olevat kehittäjätahot, paketin tunnettuvuus ja suosio sekä paketin elinkaari. Jälkimmäisellä tarkoitetaan sitä, onko npm-paketti edelleen aktiivisesti tuettu kehittäjäyhteisön toimesta. Lisenssit eivät varsinaisesti ole juuri tietoturvaan liittyvä ominaisuus, mutta lisenssin tarkistamisella pienennetään riskiä, jotta käytettäisiin organisaatiolle sopimattomia komponentteja. Näihin asioihin on pyritty ottamaan kantaa tietoperustan luvuissa 3 ja 4.

Tietoperustassa kävin läpi asioita, jotka tulisi ottaa huomioon ohjelmistokirjastoja valitessa ja käyttöönotossa. Kävin läpi muutamia asioita ohjelmistokirjastojen ylläpidosta omassa ohjelmistoprojektissa. Tutkin tietoa erinäisistä blogilähteistä ohjelmistokirjastojen käytön parhaille käytännöille.

Npm-pakettien tunnettujen haavoittuvuuksien tunnistamiseen tarkoitettuja työkaluja on paljon, niin kaupallisia että avoimena lähdekoodina julkaistuja vaihtoehtoja. Ainakin yhdestä kaupallisesta työkaluvaihtoehdosta oli jo aiemmin pidetty esittely toimeksiantajalle. Osa työkaluista on ilmaisia yksityiskäyttöön, mutta ne maksavat organisaatiokäytössä, kuten esimerkiksi GitHub Dependabot. Yhdeksi haasteeksi kaupallisten työkalujen tutkimiselle oli hinta. Toimeksiantaja saatetaan tulkita enterprise-tason organisaatioksi sen koon vuoksi, vaikkakin organisaation oma ohjelmistokehittäminen olisikin pienimuotoista. Tämä tulkinta johtaa siihen, että kaupalliset työkalut saattavat muodostua hintaviksi.

Koska opinnäytetyöllä oli rajallinen aikataulu ja resurssit, kehittämistyössä pyrittiin keskittymään ensisijaisesti avoimen lähdekoodin kirjastojen, erityisesti npm-pakettien, käyttöohjeiden luomiseen tietoturvan näkökulmasta. Työkalujen suhteen pyrittiin tutkimaan npm-komentorivityökalua sekä NPQ-komentorivityökalua. Npm-komentorivityökalu valittiin siksi, että se oli ennestään käytössä

toimeksiantajan ohjelmistoprojekteissa. NPQ valittiin taas sen vuoksi, että eräs haastatelluista työntekijöistä oli kuullut kyseisestä työkalusta, mutta hän ei ollut ehtinyt itse perehtyä siihen.

6.3 Oppaan suunnittelu

Aloitin oppaan kirjoittamisen avoimen lähdekoodin ohjelmistokirjastojen valintaan ja päivittämiseen tietoperustan keräämisen aikana. Ohjeiden painopisteeksi tuli opinnäytetyön aiheen mukaan npm-pakettien tietoturva, vaikkakin ohjeiden tarkoituksena oli käydä läpi avoimen lähdekoodin ohjelmistokirjastoja ohjeen alkupäässä myös yleisesti.

Sain oppaan suunnitteluun melko vapaat kädet aiheen tarkentumisen jälkeen. Listasin muutamia kriteereitä ohjeiden sisällölle ja muodolle:

- Ohjeiden tulisi sisältää tarkistuslista niille asioille, joita ohjelmistokehittäjän tulisi tarkistaa ennen kuin npm-paketti otetaan käyttöön ohjelmistoprojektissa.
- Ohjeiden tulisi ottaa huomioon myös ohjelmistokirjaston testaaminen tunnettujen haavoittuvuuksien osalta sen jälkeen, kun se on otettu käyttöön.
- Tarkistuslistaa edeltäisi myös lisätietoa sisältävät ohjeet kuhunkin tarkistuslistan kohtaan, kuitenkin niin, että ohjeet ovat olisivat kokonaisuudessaan selkeät ja napakat.
- Ohjeet tulisivat olemaan kirjallisessa muodossa, jotta niitä voisi tarvittaessa jatkojalostaa opinnäytetyön valmistumisen jälkeen.

Aloitin ohjeiden rungon suunnittelun luomalla ajatuskartan alustavista ohjeiden luvuista ja niiden järjestyksestä (kuva 2). Alustava sisällysluettelo sisälsi otsikot ”johdanto”, ”kirjastojen tietoturva yleisesti”, ”kirjastojen valinta”, ”kirjastojen asennusvaihe”, ”testaus / työkalut”, ”ohjelmistoprojektin ylläpito” sekä ”yhteenveto / checklist”. Pyysin palautetta toimeksiantajan puolelta ajatuskartan luomisen jälkeen, jolloin pohdimme ohjeiden pituutta ja painopisteitä tarkemmin.

Kuva 2: Ajatuskartta oppaan alustavasta sisällysluettelosta

6.4 Oppaan luonti

Aloin luomaan ohjeita avoimen lähdekoodin ohjelmistokirjastojen käyttöön sen jälkeen, kun ohjeiden alustava runko oli hyväksytty toimeksiantajan puolelta. Kirjoitin ohjeita Word-tiedostoon, otsikoiden ohjeet alustavan rungon mukaan. Ohjeiden luonnissa painopiste oli selkeässä asiasisällössä, joten esimerkiksi visuaalisuus jäi toissijaiseksi. Sisällössä painopiste sovittiin olevan kirjastojen valintaan liittyvissä käytännöissä, sillä siitä löytyi helpoiten sekä kattavaa yleistietoa että konkreettisia teknisiä ohjeita, verrattuna projektin ylläpitovaiheeseen.

Tutkin toimeksiantajan intranetiä löytääkseni jo olemassa olevia käytäntöjä npm-pakettien käyttöön. Tällä hakutavalla löytyi suppeat ohjeet npm auditin käytöstä ja muista testauskäytännöistä eräässä toimeksiantajan tämänhetkisessä ohjelmistoprojektissa. Tapasin myös muutamia toimeksiantajan ohjelmistoprojekteissa mukana olevia ihmisiä, jotka kertoivat minulle kokemuksiaan npm-pakettien valitsemis- ja päivittämiskäytännöistä. Käytäntöihin ei ollut valmiita dokumentteja, mutta esimerkiksi valitessa npm-paketteja tehtiin vapaamuotoisia vertailuja, tarkistettiin paketin suosio ja tarvittaessa kysyttiin asiasta enemmän kirjastoihin perehtyneiltä kollegoilta. Näiden lisäksi hyödynsin ohjeissa opinnäytetyön tietoperustaan kerättyä tietoa.

Valmiiden ohjeiden lukujaottelussa ja sisällössä on hienoisia eroja. ”Asennusvaiheen” sijaan päätettiin kuvata vaihetta ”kirjaston käyttöönottona”, sillä termi ”asennus” viittasi ainoastaan tekniseen osaan käyttöönotosta. Ohjeiden loppuun suunniteltu kokoava tarkistuslista pilkottiin lukujen

sisältöön selkeyden säilyttämiseksi ja turhan toiston välttämiseksi. Viimeisen luvun tarkistuslistan sijaan kerättiin linkkejä ohjeissa hyödyllisiksi mainituille verkkosivustoille.

6.5 Työkalujen konfigurointi ja testaus

Tutkittavia työkaluja olivat npm audit ja NPQ-komentorivityökalu. Testausta varten asensin uusimman Node.js:n LTS-version Windowsille sekä uusimman npm-version. Node.js:n uusin versio oli testauksen aikana huhtikuussa 2025 22.14.0 ja npm:n 11.2.0 (Node.js s.a.; npm, Inc. s.a. b). Näiden kahden asennuksen jälkeen pystyin asentamaan npm-paketteja sekä testaamaan niiden tunnettuja haavoittuvuuksia npm audit -työkalun avulla. NPQ vaatii myös Node.js:n ja npm:n toimiakseen. NPQ asennettiin komentorivin avulla ilman Snykin API-avainta.

Sovimme toimeksiantajan kanssa, että työkalujen testaamista varten testaan ReactJS-projekteissa käytettäviä npm-paketteja. Suurin osa valituista npm-paketeista oli minulle ennestään tuttuja muiden opintojen tai projektien kautta. Tätä testausta varten ei luotu kokonaista ohjelmistoprojektia, vaan testauksessa painotettiin pakettien asentamista ja auditointia. Testipaketit pyrittiin kuitenkin valitsemaan siten, että ne muodostaisivat yhdessä mahdollisimman realistisen kokonaisuuden, kuten oikeassa projektissa.

Varmistin valittujen pakettien osalta, että testaushetkellä niiden tietyissä versioissa on vahvistettu olevan tunnettu haavoittuvuus Snykin avoimessa tietokannassa, Snyk Vulnerability Databasessa (Snyk s.a. b). Vastaavasti varmistin, että tietyissä paketeissa ei ole haavoittuvuuksia. Haavoittuvuustietokantoja olisi ollut useampikin vaihtoehto, mutta Snykin tietokanta vaikutti esittävän kirjastojen haavoittuvuuksiin liittyvät tiedot selkeimmin.

Testauksen onnistumisen varmistamiseksi määrittelin asennuskomentojen yhteydessä haluavani asentaa tietyn vanhemman version kirjastosta. Asensin myös vertailun vuoksi npm-paketteja, joissa ei ollut kyseisen tietokannan mukaan tunnettuja haavoittuvuuksia. Valitsin myös request-kirjaston, joka oli vahvistettu olevan vanhentunut npm:n sivuilla (npm, Inc. s.a. c). Jätin tietoisesta syystä testaamatta malware-kirjaston asentamisen kokeilun.

Testattavat npm-paketit ja niiden haavoittuvuudet on listattu alla olevaan taulukkoon (taulukko 1). Testattavat paketit olivat Vite, axios, react-router, Tailwindcss, @tailwindcss/vite ja request. Taulukko sisältää kunkin haavoittuvuuden osalta tiedon kirjaston nimestä, asennetusta versiosta, tunnetun haavoittuvuuden englanninkielisen termin sekä CVE (Common Vulnerabilities and Exposures) -tunnisteen, siitä koskeeko haavoittuvuus asennettua versiota sekä mahdollisen tarkentavan lisätiedon.

Taulukko 1: Testattavat npm-paketit

ID	Npm-paketti	Asennettu versio	Tunnettu haavoittuvuus	Haavoittuvuus koskee asennettua versiota	Lisätietoa
1	vite	^6.2.0	Incorrect Authorization, CVE-2025-31486	kyllä	
2	vite	^6.2.0	Access Control Bypass, CVE-2025-31125	kyllä	
3	vite	^6.2.0	Incorrect Authorization, CVE-2025-30208	kyllä	
4	axios	^1.8.2	Server-side Request Forgery (SSRF), CVE NOT AVAILABLE	kyllä	
5	axios	^1.8.2	Server-side Request Forgery (SSRF), CVE-2025-27152	ei	
6	react-router	^7.5.0	ei		
7	tailwindcss	^4.1.3	ei		
8	@tailwindcss/vite	^4.1.3	ei		
9	request	^2.88.2	Server-side Request Forgery (SSRF), CVE-2023-28155	kyllä	Kehitys lopetettu.

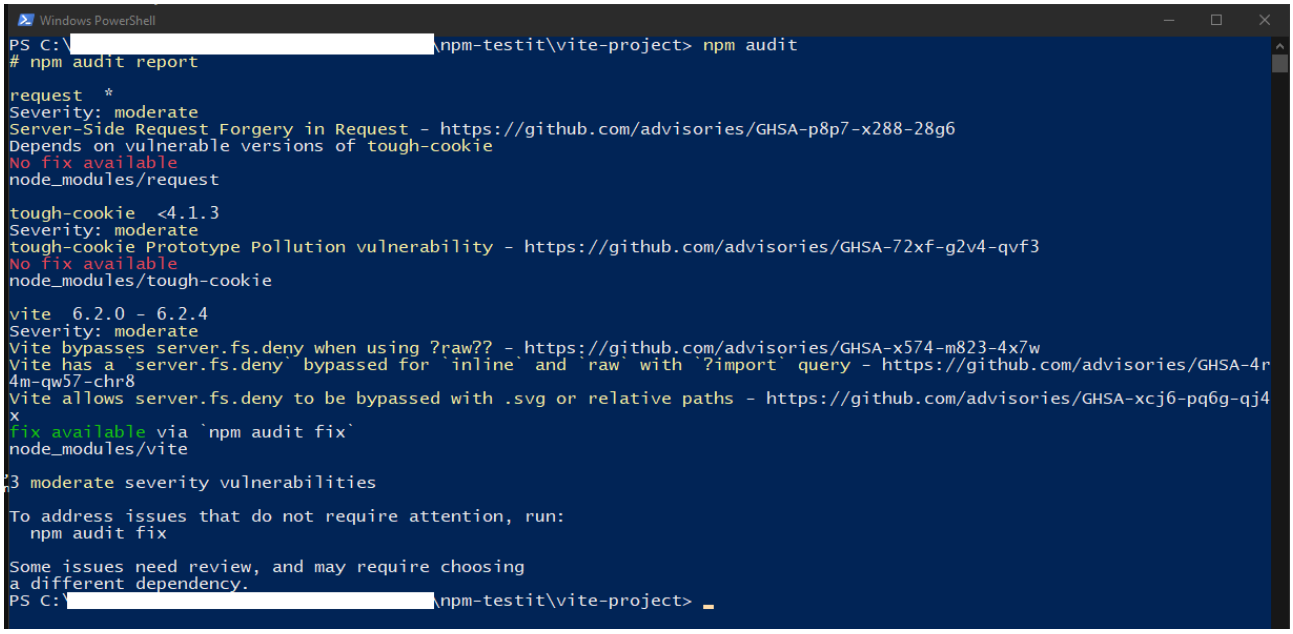
Ennen työkalujen testauksen aloittamista tiedossa oli seuraavat tunnetut haavoittuvuudet:

- Viten versiolla 6.2.0 oli yhteensä kolme haavoittuvuutta.
- Axioksella oli kaksi haavoittuvuutta, joista vain toinen koski asennettua versiota 1.8.2.
- Vanhentuneen request-kirjaston haavoittuvuus koski kaikkia asennettuja versioita.
- React-routerilla ja Tailwindcss:llä ei ollut tunnettuja haavoittuvuuksia Snykin tietokannassa.

Npm auditin testausta varten loin hakemiston tietokoneelle. Asensin testattavat kirjastot npm-komentorivityökalulla yksi kerrallaan, ellei kirjaston dokumentaatio pyytänyt muuta. Pääsääntöisesti käytin asennukseen ”npm install” -komentoa, mutta Viten asennukseen käytin komentoa ”npm create”, joka luo projektin. Viten dokumentaatio pyytää tekemään asennuksen näin. Viimeiseksi ajoin npm audit -komennon avulla raportin haavoittuvuuksista. NPQ-työkalun testausta varten loin vastaavasti toisen hakemiston, johon asensin sanat npm-paketit työkalun avulla yksitellen.

6.6 Työkalujen tulokset

Npm audit palautti testattavien kirjastojen raportissa kolme löydettyä haavoittuvuutta (kuva 3). Yksi niistä liittyi requestiin, toinen requestin aliriippuvuutena olevaan tough-cookie -kirjastoon ja kolmas Viteen. Ensimmäiset ovat haavoittuvuuksia, joihin ei ole saatavilla korjausta npm auditin mukaan, ja auditointiraportin mukaan riippuvuus olisi mahdollisesti tarpeen vaihtaa toiseen. Viten haavoittuvuus on korjattavissa päivittämällä kirjasto komennolla ”npm audit fix”.

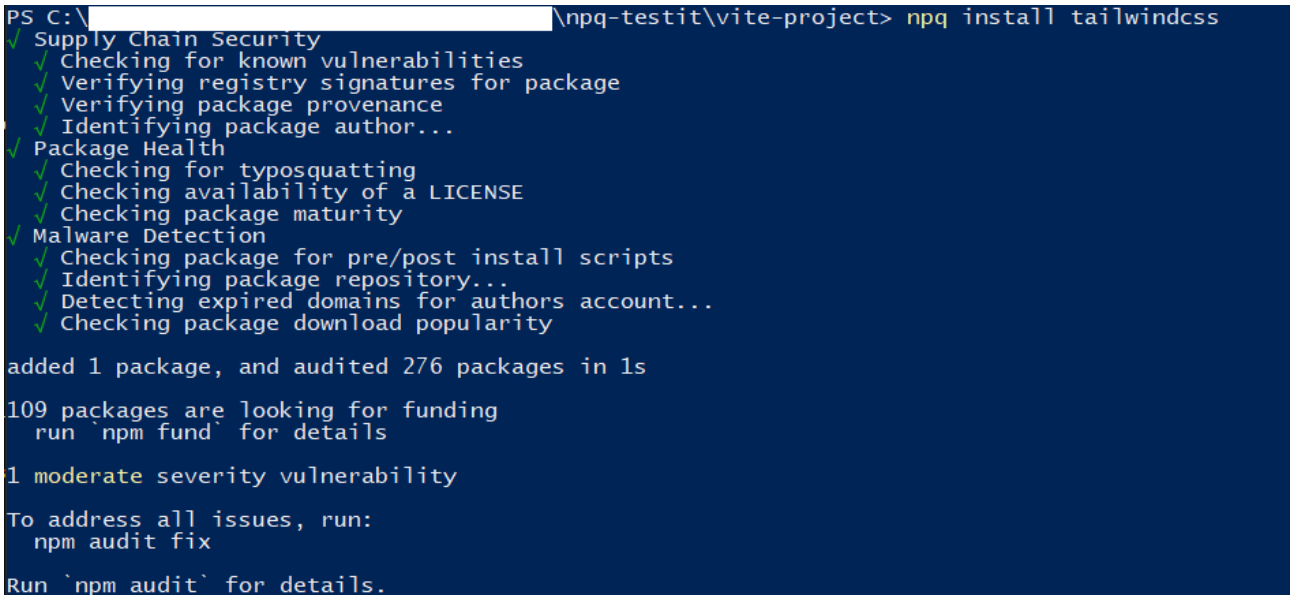


Kuva 3: Npm audit -raportti PowerShellissa

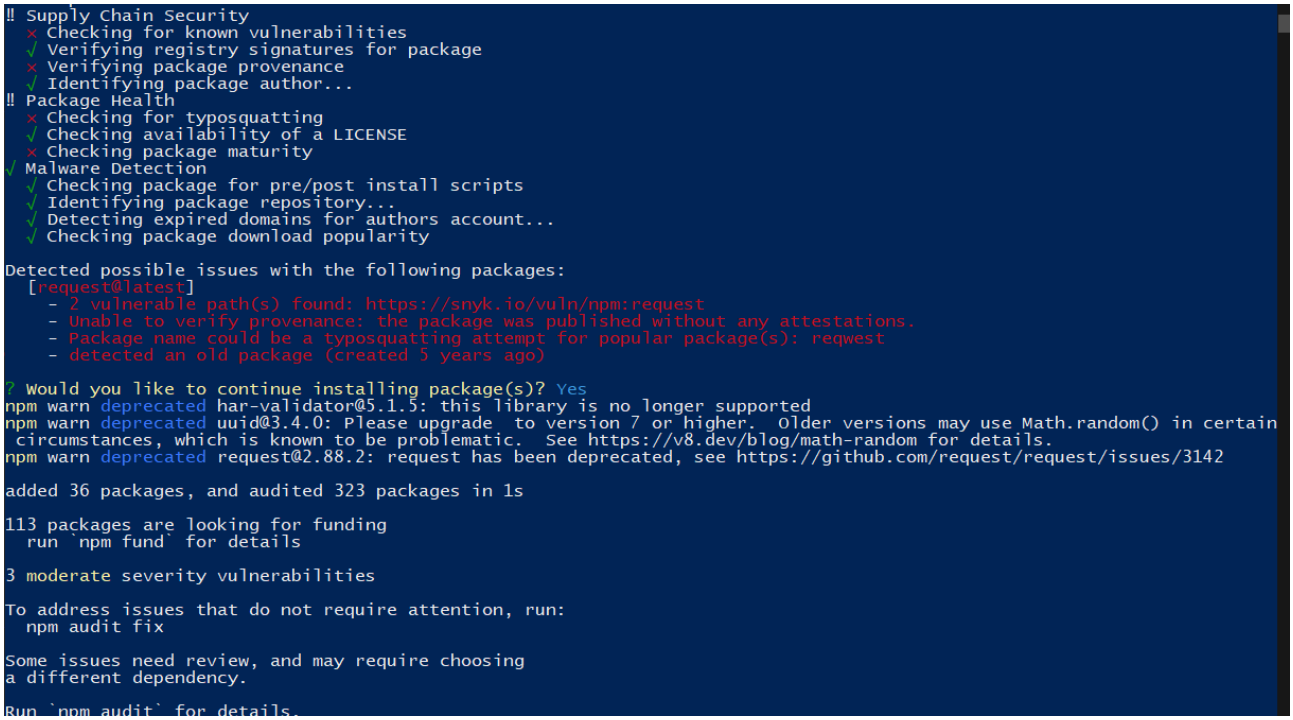
NPQ-työkalu tulostaa raportin joka kerta, kun uutta npm-pakettia asennetaan. Työkalu tarkisti esimerkiksi toimitusketjun osalta tunnetut haavoittuvuudet, paketin kunnon osalta mahdollisen typosquattingin ja lisenssin sekä arvioi, onko npm-paketti mahdollisesti haitallinen. Jos kaikki työkalun tarkistukset menivät läpi, asennus onnistui ilman kehittäjän väliintuloa (kuva 4).

Kun NPQ:n joku tarkistuksista ei mennyt läpi, raporttiin tulostui virheilmoituksia. Raportti näytti, mihin tarkistukseen virhe liittyi ja siinä lueteltiin, millaisia ongelmia työkalu havaitsi. Esimerkiksi requestin asennuksessa (kuva 5) työkalu huomasi kaksi haavoittuvuutta, se ei löytänyt todistuksia kirjaston alkuperästä ja se ilmoitti kirjaston olevan viisi vuotta vanha. Yllättäen työkalu epäili myös requestin olevan typosquatting-hyökkäysyritys “reqwest”-kirjastolle. Koska NPQ:n kaikki tarkistukset eivät menneet läpi, kehittäjällä on mahdollisuus peruuttaa asennus tai jatkaa asennusta valitsemalla komentorivillä ei tai kyllä.

Kuten kuvista 4 ja 5 huomataan, NPQ:lla npm-paketteja asentaessa myös npm:n auditointiraportti tulostuu NPQ:n raportin perään.



Kuva 4: NPQ-työkalun raportti Tailwindcss-kirjastoa asentaessa



Kuva 5: NPQ-työkalun raportti vanhentunutta request-kirjastoa asentaessa

Alla olevassa taulukossa 2 on vertailtu npm auditia ja NPQ-työkalua eri ominaisuuksien mukaan. NPQ:n etuna oli monipuolinen tarkistusprosessi ennen asennusta sekä kehittäjän mahdollisuus keskeyttää asennus, mikäli jokin työkalun asennuksista ei mene läpi. NPQ myös löysi kaikki Snykin tietokannasta poimitut haavoittuvuudet, toisin kuin npm audit, joka löysi vain muutaman. Toisaalta

NPQ myös väitti löytävänsä tunnetun haavoittuvuuden “@typescript/vite”-kirjastosta, johon ei löytynyt selvää vastinetta Snykin tietokannasta.

Tunnettujen haavoittuvuuksien osalta npm audit tarjosi kattavamman raportin komentorivillä, sillä raporttiin sisältyi haavoittuvuuksiin vaikuttavat kirjastot, haavoittuvuuden vaikuttavuusluokka sekä lyhyen kuvauksen haavoittuvuudesta. NPQ:lla tunnettujen haavoittuvuuksien tiedot tulee tarkistaa erikseen Snykin verkkosivuilta.

Taulukko 2: Npm auditin ja NPQ:n vertailu.

Ominaisuus	**Npm audit**	**NPQ**
Käyttötarkoitus	Jo asennettujen npm-pakettien auditointi.	Npm-pakettien auditointi asennusvaiheessa.
Tiedon esittäminen auditointiraportissa	Erittelee projektin kaikki löydetyt tunnetut haavoittuvuudet siihen vaikuttavan kirjaston mukaan. Kertoo kunkin haavoittuvuuden vaikuttavuusluokan sekä lyhyen kuvauksen. Määrittelee, onko haavoittuvuus korjattavissa vai ei ilman kirjaston vaihtamista.	Tarkistaa tunnettujen haavoittuvuuksien lisäksi myös muita pakettien tietoturvaan liittyviä asioita. Ei näytä suoraan tietoja tunnetuista haavoittuvuuksista, vaan ne tulisi tarkistaa erikseen Snykin verkkosivuilta.
Erillinen varmistus asennuksen keskeyttämiseksi	Ei.	Kyllä, jos jokin työkalun tarkistuksista ei mene läpi.
Löydetyt haavoittuvuudet	2 requestin haavoittuvuutta ja 1 Viten haavoittuvuus. Ei löytänyt kaikkia Snykin tietokannasta poimittuja haavoittuvuuksia.	3 Viten haavoittuvuutta, 1 Axioksen haavoittuvuus, 1 @tailwindcss/viten haavoittuvuus, 2 requestin haavoittuvuutta.
Herjaa kirjaston haavoittuvuudesta, joka ei vaikuta asennettavaan versioon	Ei.	Antoi tiedon tunnetusta haavoittuvuudesta @tailwindcss/vite -kirjaston kohdalla. Työkalun antamasta linkistä Snykin tietokannan verkkosivulle ei kuitenkaan löytynyt vastinetta.
Muuta huomioitavaa		Näyttää auditointiraportissaan sekä oman koosteen, että npm auditin suppeat tiedot.

Työkalujen luomia raportteja käytettiin lopputuotoksena tehtyjen ohjeiden luvussa 5: “Testaus / työkalut”. Molempia työkaluja suositeltiin käytettäväksi: NPQ-työkalua npm-pakettien käyttöönottovaiheessa ja npm auditia yleisesti npm-pakettien tunnettujen haavoittuvuuksien tarkistukseen.

6.7 Valmis opas

Valmis opas on tämän opinnäytetyön liitteenä (liite 1) ja se julkaistiin myös toimeksiantajan käyttöön tulevia ohjelmistoprojekteja varten. Ohjeiden kokonaispituus on 13 sivua sisällysluettelo mukaan laskettuna. Valmiiden ohjeiden sisällys on tiivistetysti seuraavanlainen:

- **Johdanto**. Kerrotaan lyhyesti ohjeiden sisällön koskevan avoimen lähdekoodin ohjelmistokirjastojen, tarkemmin npm-pakettien, tietoturvaa. Ohjeiden kerrotaan olevan suunnattu ensisijaisesti ohjelmistokehittäjille.
- **Kirjastojen tietoturva yleisesti**, joka sisältää alaluvut ”tunnetut haavoittuvuudet”, ”toimitusketjuhyökkäykset”, ”typosquatting” ja ”legacykirjastot”.
- **Kirjastojen valinta**, joka sisältää alaluvut ”vinkkejä kirjastojen etsintään”, ”tarkistus kirjastoja valitessa” sekä ”lisensseistä”.
- **Kirjaston käyttöönotto**. Muutamia suosituksia npm-pakettien käyttöönottoon, kuten NPQ-työkalun käyttö pakettia asentaessa.
- **Testaus / työkalut**, joka sisältää alaluvut ”npm audit” ja ”NPQ-komentorivityökalu”. Sisältää komentoja npm auditin käyttöön sekä havainnoivia kuvia kummankin työkalun antamista raporteista.
- **Ohjelmistoprojektin ylläpito**. Luvussa käydään läpi asioita, joista tiimien tulisi sopia ohjelmistokirjastojen käytön suhteen, kuten kuka vastaa oman projektin riippuvuuksien päivittämisestä, kuinka usein kirjastoja päivitetään ja millaisilla käytännöillä kirjastoja tulisi vaihtaa.
- **Hyödyllisiä resursseja**. Loppuun on koottu ulkoisia linkkejä ohjeissa mainituille työkaluille, npm-pakettien hakukoneille ja tunnettujen haavoittuvuuksien tietokannoille.

Oppaasta pyydettiin vapaamuotoista palautetta muutamilta toimeksiantajan työntekijöiltä, joista osaa oli haastateltu aiemmin opinnäytetyön aikana. Palautteita saatiin yhteensä neljältä henkilöltä, jotka toimivat erilaisissa ammatillisissa rooleissa.

Ensimmäinen palautteenantaja oli tietoturva-asiantuntija, jolla ei ollut ohjelmointitaustaa. Palautteenantajan palaute oli seuraavanlainen:

> "Ohje on kattava ja hyvin jäsennelty. Se on jaettu loogisiin osiin, jotka käsittelevät aiheen keskeisiä näkökohtia riittävällä laajuudella. Ohje on huolellisesti laadittu ja sisältää konkreettisia esimerkkejä sekä suosituksia. Kokonaisuudessaan dokumentti on mielestäni siis hyvin laadittu, selkeä, informatiivinen ja käytännönläheinen."

Toinen palautteenantaja oli perehtynyt kattavasti kirjastojen tietoturvaan. Palautteessa mainittiin ohjeiden olevan hyvät ”kehittäjälle, joka ei vielä tiedä, mitä riskejä avoimen lähdekoodin pakettien käyttämiseen liittyy”. Palautteessa mainittiin ohjeiden taustoittavan aihetta selkeästi, sopivan käytännön työhön ja nostavan esiin ajankohtaisia haasteita, kuten tekoälyn ”hallusinoimia” paketteja. Toinen palautteenantaja toivoi lisää tietoa joihinkin ohjeiden kohtiin, esimerkiksi tunnettujen

haavoittuvuuksien osalta eron suorien ja transitiivisten eli ”periytyvien” haavoittuvuuksien välillä sekä kevyen esimerkin npm auditin ja NPQ:n automatisoinnista CI/CD-putkessa. Ylläpidon osioon toivottiin mainintaa haavoittuvuuksien yleisestä monitoroinnista ja raportoinnista:

> ”Isommissa kokonaisuuksissa on hyödyllistä tietää esim. kuinka vakavia haavoittuvuuksia on ratkaisematta, mitä projekteja ne koskee, kuinka pitkään ne ovat olleet hoitamatta ja millaisen riskin ne muodostavat.”

Kolmas palautteenantaja toimi palvelupäällikkönä. Hän piti sisältöä pääosin selkeänä ja kattavana, sekä kuvankaappauksia ja käytettyjä esimerkkejä tekstin ymmärrettävyyttä tukevana. Ohjeiden sisältämiä tarkistuslistoja ja käytännön vinkkejä pidettiin hyödyllisenä. Ohjeisiin toivottiin kuitenkin enemmän konkreettisia esimerkkejä tietoturvaongelmista ja niiden ratkaisuista. Kolmas palautteenantaja huomautti myös ohjeiden lisensseistä kertovasta osiosta seuraavasti:

> ”GPL lisenssi ei automaattisesti pakota julkaisemaan koodia avoimena lähdekoodina. GPL lisenssi **voi** pakottaa julkaisemaan oman koodin, jos GPL lisenssi mukainen ohjelmisto on suoraan linkitetty omaan ohjelmistoon. Lisensoinnin asiantuntijoilla on lisenssien tulkinnasta tietyissä tilanteissa eriäviä näkemyksiä, Tässä voisi kuvata myös LGPL:n lisenssi ja suositella sen käyttöä.”

Neljäs palautteenantaja työskenteli projektipäällikkönä toimeksiantajan ohjelmistoprojektissa. Ohjeita kehuttiin selkeiksi. Palautteenantaja nosti hänen ammatillisen roolinsa näkökulmastaan kehityskohteen, että ohjeet olisi hyvä integroida osaksi toimeksiantajan ympäristössä jonkin yleisen sovelluskehitystä koskevan ohjeistuksen sisään, sillä useita erillisiä ja irrallisia ohjeita voi olla hankalaa löytää.

7 Pohdinta

Viimeiseksi pohdin opinnäytetyössä syntyneen tuotoksen tuloksia. Esitän niiden pohjalta johtopäätöksiä ja kehittämisehdotuksia. Arvioin myös opinnäytetyöprosessia ja omaa oppimistani opinnäytetyön aikana.

7.1 Tulosten tarkastelu

Opinnäytetyön tavoitteena oli etsiä tietoa avoimeen lähdekoodiin ja ohjelmistokirjastoihin liittyvistä tietoturvariskeistä ja määrittää näiden pohjalta kriteerit npm-pakettien valintaan sekä testaamiseen. Opinnäytetyön aikana luodut ohjeet toimivat suuntaa antavana manuaalina ohjelmistokirjastojen käytölle niin nykyisille kuin tuleville ohjelmistoprojekteille.

Oppaasta saadun palautteen perusteella ohjeiden sisältö oli pääosin selkeä ja käytännönläheinen. Palaute oli pääosin positiivista. Saadun palautteen osalta olisi kuitenkin mahdollisesti tarpeen lisätä muutamia konkreettisia esimerkkejä lisää ohjeisiin, jotta ohjeet tarjoaisivat entistä kattavammin tietoa. Ohjeiden mahdollinen jatkokehitys toimeksiantajan ympäristössä ajoittuu opinnäytetyön valmistumisen jälkeiseen aikaan.

Ajoittain oli haasteellista löytää konkreettisia vinkkejä npm-pakettien käyttöön, sillä suurin osa löytämistäni kirjalähteistä käsitteli ohjelmistokirjastoja hyvin pienimuotoisesti ja yleisellä tasolla. Siksi osa parhaiden käytäntöjen lähteistä on blogeista tai npm-pakettien testaukseen tarkoitettujen yritysten ja muiden kehittäjätahojen verkkosivuilta, joiden tarjoaman tiedon suhteen tulee olla tietyiltä osin kriittinen.

Tavoitteena oli myös tutkia pienimuotoisesti olemassa olevia työkaluja ohjelmistokirjastojen tunnettujen haavoittuvuuksien testaamiseen. Opinnäytetyön aikana ehdittiin tutkia tarkemmin kahta työkalua: npm auditia sekä NPQ-työkalua. Opinnäytetyössä tehdyn pienimuotoisen testauksen perusteella npm audit ja NPQ täydentävät toisiaan npm-pakettien käytön eri vaiheissa. NPQ:n raportti on kattavampi kuin npm audit ja sen tuottama tieto on hyödyllistä erityisesti siinä vaiheessa, kun uutta kirjastoa ollaan ottamassa käyttöön ohjelmistoprojektiin.

Toimeksiantajan yhtenä toiveena oli, että tutkittavat työkalut hyödyntäisivät automaatiotestausta. Tätä asiaa ei ehditty tutkimaan opinnäytetyön aikana. Npm auditin voi kuitenkin integroida osaksi CI/CD-putkea, joten se hyödyntää omilta osin automaatiotestausta. NPQ-työkalun käytöstä CI/CD-putkessa ei löytynyt valmista dokumentaatiota, ja asiaa ei tutkittu opinnäytetyön aikana sen tarkemmin.

7.2 Johtopäätökset ja kehittämisehdotukset

Npm-pakettien ympäristö muuttuu jatkuvasti. Uusia kirjastoja julkaistaan jatkuvasti lisää ja osa niistä syrjäyttää toisen kirjaston suosiollaan, vanhoja kirjastoja ei enää ylläpidetä, kirjastoista löydetään haavoittuvuuksia ja niitä korjataan. Uudet työkalut, kuten tekoäly, voivat helpottaa kirjastojen vertailua, mutta toisaalta ne voivat myös luoda uusia tietoturvariskejä luodessaan koodia.

Arvioin npm-pakettien käyttöön luotujen ohjeiden olevan ajankohtaisia, vaikkakaan npm-paketit eivät ole uusi asia. Tällä hetkellä ei vaikuta siltä, että jokin toinen ratkaisu syrjäyttäisi npm-pakettien tai avoimen lähdekoodin käytön pian täysin. Luotujen ohjeiden pituus pysyi napakkana. Ohjeet pyrkivät tarjoamaan perustietoa kirjastojen hyödyntämisestä. Opinnäytetyön aikana kerätyn palautteen perusteella ohjeita on kuitenkin mahdollisuus kehittää entisestä paremmaksi opinnäytetyön jälkeen.

Opinnäytetyössä kerättiin tietoa yleisesti ohjelmistokirjastoihin liittyvistä tietoturvauhista sekä kerättiin vinkkejä npm-pakettien valintaan, käyttöönottoon ja hallintaan. Npm-pakettien käyttöönottamisessa on paljon tarkistettavaa ja paketteja on lähes mahdotonta tarkistaa pelkästään manuaalisesti. Kerätyn tiedon perusteella on tarpeellista, että ohjelmistokirjastojen käytön seurannaksi otetaan käyttöön työkaluja, jotka reagoivat kirjastojen käytön eri vaiheissa.

Opinnäytetyön aikana muodostui muutamia jatkokehittämisideoita. Tässä opinnäytetyössä keskityttiin kirjastojen valinnan ja testaamisen osalta npm-paketteihin. Tästä syystä yksi konkreettinen jatkokehittämisidea olisi etsiä työkaluja ja parhaita käytänteitä jonkin toisen ohjelmointikielen, kuten Javan tai Pythonin, avoimen lähdekoodin ohjelmistokirjastojen testaamiseen.

Toinen kehitysidea on kirjastojen tarkistamisen käytäntöjen kehittäminen entisestään jonkin viitekehyksen avulla, kuten vaikkapa Microsoftin kehittämällä Secure Supply Chain Consuption Frameworkillä (S2C2F), joka sisältää yhdistelmän prosesseja ja työkaluja avoimen lähdekoodin käyttöönottoon ja hallinnointiin. (Microsoft s.a.). S2C2F-viitekehys löydettiin opinnäytetyöprosesin aikana, mutta sitä ei ehditty tutkimaan erityisen tarkasti.

Kolmas kehitysidea on ohjelmistokirjastojen hallintaan tarkoitettujen, kaupallisten työkalujen tarkempi kartoitus. Vaikka kaupalliset työkalut maksavat rahaa, myös tietoturvariskit voivat toteutuessaan olla hintavia ja aiheuttaa muunlaisia ongelmia, kuten henkilötietojen loukkaus, luottamuksen menetys organisaatioon tai toimintakyvyn heikentyminen estyneiden palveluiden saatavuuden vuoksi.

7.3 Opinnäytetyöprosessin ja oman oppimisen arviointi

Viimeiseksi tarkastelen opinnäytetyöprosessia sekä omaa oppimistani. Arvioin prosessiin liittyviä onnistumisia ja haasteita.

Ennen opinnäytetyön aloittamista en tiennyt avoimen lähdekoodin ohjelmistokirjastoista paljoakaan. Olin toki käyttänyt niitä ohjelmistoprojekteissa ja tutustunut suppeasti kirjastoihin liittyviin dokumentaatiosivustoihin. En kuitenkaan osannut etsiä tietoa, miksi minun pitäisi käyttää jotain tiettyä kirjastoa toisen sijasta tietoturvan näkökulmasta, enkä kiinnittänyt huomiota vaikkkapa ohjelmistokirjastojen lisenssiehtoihin.

Opinnäytetyön aikana minulle muodostui kattava kuva npm-pakettien käytöstä ja niiden arvioimisesta tietoturvan kannalta. Kiinnostuin myös opinnäytetyön aikana yleisesti avoimen lähdekoodin konseptista. Koen, että onnistuin keräämään oleellista tietoa ja jakamaan oppimani opinnäytetyön aikana kirjoitetun oppaan muodossa. Toisaalta npm-pakettien ympäristö on valtava ja monimutkainen, ja aiheeseen olisi varmasti voinut perehtyä loputtomiin.

Haasteeksi muodostui tarkasteltavien työkalujen valinta. Sopivien työkalujen etsimiseen kului aikaa tai sille annettiin opinnäytetyöprosessin alussa mahdollisesti liikaa painoarvoa verrattuna avoimen lähdekoodin ohjelmistokirjastojen käytön parhaiden käytäntöjen etsimiseen. Toisaalta lopputuotoksena syntyviin ohjeisiin haluttiin myös säilyttää työkaluista kertova osio, joten siksi niitä ei rajattu kesken prosessin kokonaankaan pois. Parhaisiin käytänteisiin kuuluu myös joiltain osin työkalujen käyttö ohjelmistokirjastojen hallinnan apuna, joten työkalujen pois jättö ei olisi ollut järkevää.

Opinnäytetyön aiheen rajaaminen aiheutti välillä hankaluuksia, sillä aihe haki vielä lopullista muotoaan opinnäytetyöprosessin ensimmäisinä viikkoina. Npm-pakettien tutkiminen aiheena vaikutti välillä hyvinkin pienimuotoiselta ja koin tarvetta kasvattaa aiheen laajuutta. Toisaalta aihe kytkeytyi moneen isompaan kokonaisuuteen, kuten kaikkien ohjelmistoprojektiin liittyvien tietoturvariskien ehkäisyyn tai avoimen lähdekoodin käyttöön yleisesti toimeksiantajan ympäristössä. Näiden isompien kokonaisuuksien tarkastelu kattavasti olisi kasvattanut opinnäytetyöstä liian suuren.

Opinnäytetyöprosessin aikana keskustelin lähes viikoittain toimeksiantajan yhteyshenkilön kanssa. Pidin käytäntöä onnistuneena, sillä se motivoi minua edistämään opinnäytetyötä viikoittain. Tällä käytännöllä pyrittiin myös varmistamaan se, että aiheen ja tavoitteiden muotoilu pysyy hallinnassa, vaikka se tuottikin haasteita.

Opinnäytetyön aiheen muotoiluun ja lopputuotoksen valmistumiseen oli varattu aikaa tammikuusta toukokuun puoleenväliin. Arvioin, että kokonaisuudessaan huomioon ottaen AMK-opinnäytetyön laajuus, opinnäytetyölle oli varattu tarpeeksi aikaa ja liikkumatilaa määräaikoihin, vaikkakin raportin välitavoitteiksi asetetut päivämäärät siirtyivät hieman tulevaisuuteen. Lopputuotos valmistui ajoissa ja sitä voidaan hyödyntää tulevissa ohjelmistoprojekteissa.

Lähteet

@adam-npm 8.5.2022. `npm audit`: identify and fix insecure dependencies. Npm, Incin blogi. Luettavissa: https://blog.npmjs.org/post/173719309445/npm-audit-identify-and-fix-insecure. Luettu: 3.4.2025.

Bece, A. 12.7.2020. Checklist for choosing an optimal npm package. Dev Community. Luettavissa: https://dev.to/adrianbdesigns/checklist-for-choosing-an-optimal-npm-package-4dpm. Luettu: 2.4.2025.

Boychenko, K. 11.3.2025. Lazarus Strikes npm Again with New Wave of Malicious Packages. Luettavissa: https://socket.dev/blog/lazarus-strikes-npm-again-with-a-new-wave-of-malicious-packages. Luettu: 18.3.2025.

Black Duck Software Inc 2.2025. 2025 Open Source Security and Risk Analysis. Luettavissa: https://www.blackduck.com/content/dam/black-duck/en-us/reports/rep-ossra.pdf. Luettu: 9.4.2025.

Claburn, T. 12.4.2025. LLMs can't stop making up software dependencies and sabotaging everything The Register. Luettavissa: https://www.theregister.com/2025/04/12/ai_code_suggestions_sabotage_supply_chain/. Luettu: 21.4.2025.

COSS ry s.a. a. Avoin lähdekoodi. Luettavissa: https://coss.fi/palvelut/avoimuus/avoin-lahdekoodi/. Luettu: 2.5.2025.

COSS ry s.a. b. Yleisiä käsitteitä. Luettavissa: https://coss.fi/palvelut/avoimuus/yleisia-kasitteita/. Luettu: 2.5.2025.

COSS ry 3.5.2023. White paper: Avoimen lähdekoodin osaamisen edistäminen Suomessa. COSS ry – Suomen avoimien tietojärjestelmien keskus. Luettavissa: https://coss.fi/wp-content/uploads/2023/05/White-paper_-Avoimen-lahdekoodin-osaamisen-edistaminen-Suomessa.pdf. Luettu: 2.5.2025.

Digi- ja väestötietovirasto 15.11.2022. VAHTI-riskienhallintasanasto digitaaliseen toimintaympäristöön – esittely ja johdatusta riskiviestintään. Luettavissa: https://dvv.fi/documents/16079645/110183105/VAHTI-riskienhallintasanasto+digitaaliseen+toimintaympäristöön.pdf/6d71d86f-c7bc-6683-9b36-c55d16d4c1f0/VAHTI-riskienhallintasanasto+digitaaliseen+toimintaympäristöön.pdf?t=1674484177085. Luettu: 10.1.2025.

HackerOne s.a. Supply Chain Attacks: Impact, Examples, and 6 Preventive Measures. Luettavissa: https://www.hackerone.com/knowledge-center/supply-chain-attacks-impact-examples-and-6-preventive-measures. Luettu: 25.3.2025.

Ite Wiki s.a. Mitä on ohjelmistokehitys? Luettavissa: https://www.itewiki.fi/opas/mita-on-ohjelmistokehitys/. Luettu: 10.1.2025.

Jaja, D. 24.6.2024. When to Use NPM Packages – A Guide for Developers. freeCodeCamp. Luettavissa: https://www.freecodecamp.org/news/when-to-use-npm-packages/. Luettu: 17.4.2025.

Juvonen, R. 2018. Ohjelmistoprojektin sudenkuopat ja miten ne vältetään. BoD - Books on Demand. Helsinki.

Krishin, G. 17.10.2022. How to choose the right NPM package in 4 steps 📦. Dev Community. Luettavissa: https://dev.to/dealwith/how-to-choose-the-right-npm-package-in-a-4-steps-42bo. Luettu: 2.4.2025.

von Kügelgen, M., Laukkonen, V. & Hyvärinen, A-M. 2020. Kaikki koodaa: päivitä itsesi – käytännön opas ajankohtaisiin digitaitoihin. Into. Helsinki.

Kyberturvallisuuskeskus 23.3.2023. Haavoittuvuudet - miten niistä ilmoitetaan oikein. Luettavissa: https://www.kyberturvallisuuskeskus.fi/fi/ajankohtaista/haavoittuvuudet-miten-niista-ilmoitetaan-oikein. Luettu: 20.3.2025.

Kyberturvallisuuskeskus 19.2.2025. Ohjelmistoriippuvuuksien riskienhallinta – miksi se kannattaa ottaa osaksi jokapäiväistä tekemistä? Luettavissa: https://kyberturvallisuuskeskus.fi/fi/toimintamme/hankkeet-ja-projektit/ohjelmistoturvallisuus/ohjelmistoriippuvuuksien-riskienhallinta. Luettu: 6.5.2025.

Microsoft s.a. Secure Supply Chain Consumption Framework (S2C2F). Luettavissa: https://www.microsoft.com/en-us/securityengineering/opensource. Luettu: 5.5.2025.

Nagle, F., Powell, K., Zitomer, R. & Wheeler, D. A. 2024. Census III of Free and Open Source Software. The Linux Foundation. Luettavissa: https://www.linuxfoundation.org/hubfs/LF%20Research/lfr_censusiii_120424a.pdf?hsLang=en. Luettu: 30.1.2025.

Node.js s.a. Run JavaScript Everywhere. Luettavissa: https://nodejs.org/en. Luettu: 7.4.2025.

Node.js 5.12.2024. An introduction to the npm package manager. Luettavissa: https://nodejs.org/en/learn/getting-started/an-introduction-to-the-npm-package-manager. Luettu: 6.3.2025.

Node-semver s.a. semver(1) -- The semantic versioner for npm. GitHub. Luettavissa: https://github.com/npm/node-semver. Luettu: 15.4.2025.

Npm, Inc. s.a. a. About npm. Luettavissa: https://www.npmjs.com/about. Luettu: 9.4.2025.

Npm, Inc. s.a. b. npm. Luettavissa: https://www.npmjs.com/package/npm?activeTab=versions. Luettu: 7.4.2025.

Npm, Inc. s.a. c. request. Luettavissa: https://www.npmjs.com/package/request. Luettu: 15.4.2025.

Npm, Inc. 12.9.2024. npq. Luettavissa: https://www.npmjs.com/package/npq. Luettu: 21.4.2025.

Npm, Inc. 4.4.2024. npm-audit. Luettavissa: https://docs.npmjs.com/cli/v11/commands/npm-audit. Luettu: 6.5.2025.

Npm, Inc. 27.9.2023. npm-sbom. Luettavissa: https://docs.npmjs.com/cli/v11/commands/npm-sbom. Luettu: 30.4.2025.

Npm, Inc. 13.2.2025. Searching for and choosing packages to download. Luettavissa: https://docs.npmjs.com/searching-for-and-choosing-packages-to-download. Luettu: 2.4.2025.

Npm trends s.a. npq. Luettavissa: https://npmtrends.com/npq. Luettu: 17.4.2025.

The Open Source Initiative 22.3.2007. The Open Source Definition. Luettavissa: https://opensource.org/osd. Luettu: 30.4.2025.

The Open Source Initiative s.a. OSI Approved Licenses. Luettavissa: https://opensource.org/licenses. Luettu: 10.2.2025.

The OWASP Foundation s.a. a. Vulnerabilities. Luettavissa: https://owasp.org/www-community/vulnerabilities/. Luettu: 20.3.2025.

The OWASP Foundation s.a. b. OWASP Top Ten. Luettavissa: https://owasp.org/www-project-top-ten/. Luettu: 30.4.2025.

The OWASP Foundation s.a. c. A06:2021 – Vulnerable and Outdated Components. Luettavissa: https://owasp.org/Top10/A06_2021-Vulnerable_and_Outdated_Components/. Luettu: 20.3.2025.

OWASP Top Ten Project s.a. OWASP Top Ten 2025. Luettavissa: https://www.owasptopten.org. Luettu: 20.3.2025.

Perlow, J. 7.3.2022. A Summary of Census II: Open Source Software Application Libraries the World Depends On. Linux Foundationin blogi. Luettavissa: https://www.linuxfoundation.org/blog/blog/a-summary-of-census-ii-open-source-software-application-libraries-the-world-depends-on. Luettu: 30.1.2025.

Pina, N. 5.6.2022. How to Update NPM Dependencies. freeCodeCamp. Luettavissa: https://www.freecodecamp.org/news/how-to-update-npm-dependencies/. Luettu: 15.4.2025.

Pohjola, A. 28.2.2025. Your Company Needs an Open Source Strategy. Luettavissa: https://cloudamite.com/your-company-needs-an-open-source-strategy/. Luettu: 21.4.2025.

Rahkonen, J. 8.6.2023. Avointa lähdekoodia on kaikkialla, joten sitä kannattaa yrittää ymmärtää. Maanmittauslaitoksen blogi. Luettavissa: https://www.maanmittauslaitos.fi/ajankohtaista/avointa-lahdekoodia-kaikkialla-joten-sita-kannattaa-yrittaa-ymmartaa. Luettu: 11.2.2025.

Rishavanand2k21 25.4.2024. What is npm audit? Geeks for geeks. Luettavissa: https://www.geeksforgeeks.org/what-is-npm-audit/. Luettu: 9.4.2025.

Semver s.a. Semantic Versioning 2.0.0. Luettavissa: https://semver.org. Luettu: 15.4.2025.

Snyk s.a. a. Liran Tal. Luettavissa: https://snyk.io/contributors/liran-tal/. Luettu: 7.5.2025.

Snyk s.a. b. Snyk Vulnerability Database. Luettavissa: https://security.snyk.io. Luettu: 15.4.2025.

Stream s.a. Open-Source Libraries. Luettavissa: https://getstream.io/glossary/open-source-libraries/. Luettu: 6.2.2025.

Tal, L. 12.1.2021. What is typosquatting and how typosquatting attacks are responsible for malicious modules in npm. Snykin blogi. Luettavissa: https://snyk.io/blog/typosquatting-attacks/. Luettu: 18.3.2025.

Thomson, E. 7.10.2021. GitHub Advisory Database now powers npm audit. GitHubin blogi. Luettavissa: https://github.blog/security/supply-chain-security/github-advisory-database-now-powers-npm-audit/. Luettu: 17.3.2025.

Thomson, E. 3.3.2024. How npm install scripts can be weaponized: A real-world example of a harmful npm package. Stacklog-yrityksen blogi. Luettavissa: https://stacklok.com/blog/how-npm-install-scripts-can-be-weaponized-a-real-life-example-of-a-harmful-npm-package. Luettu: 14.4.2025.

Vidal, N. 7.12.2023. The most popular licenses for each language in 2023. Open Source Initiativen blogi. Luettavissa: https://opensource.org/blog/the-most-popular-licenses-for-each-language-2023. Luettu: 10.2.2025.

Liitteet

Liite 1. Ohjeet avoimen lähdekoodin ohjelmistokirjastojen (npm-paketit) käyttöön

Ohjeet avoimen lähdekoodin ohjelmistokirjastojen (npm-paketit) käyttöön

Sonja Fallström

2025

Sisällys

1. Johdanto

Avoimen lähdekoodin ohjelmistokirjastojen käytössä on monia mahdollisuuksia: parhaimmillaan ne parantavat ohjelmiston suorituskykyä, nopeuttavat kehittämiseen kuluvaa aikaa ja säästävät vaivaa. Maailmanlaajuisen kehittäjäyhteisön luoma koodi on monen silmäparin tarkkailtavana haavoittuvuuksien varalta ja avoimen lähdekoodin projektit ovat usein hyvin dokumentoituja. Koska avoimen lähdekoodin ohjelmistokirjastot ovat kolmannen osapuolen luomaa koodia, sen käyttöön liittyy myös riskejä tietoturvan kannalta.

Ohjeissa keskitytään Node.js:n npm-paketteihin, joita on käytössä esimerkiksi JavaScript-, TypeScript- ja ReactJS-projekteissa. Osaa ohjeista voi soveltaa myös muunlaisissa ohjelmistoprojekteissa, joissa käytetään avoimen lähdekoodin ohjelmistokirjastoja. Ohjeet ovat tarkoitettu ensisijaisesti ohjelmistokehittäjille. Ohjeet sisältävät perustietoa kirjastojen tietoturvan testaamiseen tarkoitetuista työkaluista. Ne eivät sisällä tietoa kirjastojen testaamisesta eristetyssä ympäristössä.

Oletuksena on, että kehittäjällä on vähintään perustietämys ohjelmistokehityksestä sekä Node.js ja npm-paketinhallinta on jo asennettu. Ajantasaiset asennusohjeet näiden asentamiseksi löytyvät Node.js:n verkkosivustolta.

Ohjeet pyrkivät vastaamaan seuraaviin kysymyksiin:

- Millaisia uhkia ohjelmistokirjastojen käyttöön liittyy?
- Miten ohjelmistokirjastoja käyttäessä voidaan ehkäistä tietoturvahaavoittuvuuksia ja -loukkauksia?
- Miten kirjastoja kannattaa valita ja tarkistaa?
- Mitä tulisi huomioida ohjelmistoprojektin ylläpidossa kirjastojen kannalta?

Dokumentin lopussa on tarkistuslista ohjelmistokirjastojen käyttöönottamiseksi npm-paketteja hyödyntävässä projektissa.

Luvut ”Kirjastojen valinta”, ”Kirjaston käyttöönotto” ja ”Ohjelmistoprojektin ylläpito” sisältävät tarkistuslistat kirjastojen käyttöön. Dokumentin loppuun on koottu lista hyödyllisistä resursseista, kuten dokumentaatiosivut työkalujen käyttöön ja esimerkkejä tunnettujen haavoittuvuuksien avoimista tietokannoista.

Ohjeet ovat tehty osana tietojenkäsittelyn tradenomin opinnäytetyötä keväällä 2025.

2. Kirjastojen tietoturva yleisesti

Näissä ohjeissa avoimen lähdekoodin ohjelmistokirjastolla tarkoitetaan kokoelmaa ennalta luotua koodia, jota kehittäjät saavat lisenssiehtojen mukaan käyttää ja uudelleen jakaa vapaasti. Ohjeissa keskitytään Node.js:n julkisesti saatavilla oleviin npm-paketteihin, joita on käytössä esimerkiksi JavaScript-, TypeScript- ja ReactJS-projekteissa eli lähinnä frontend-projekteissa. Npm-paketteja käytetään myös Node.js-projekteissa. Ohjeisiin on koottu erilaisia vinkkejä ja toimenpiteitä, miten ohjelmistokirjastoja voi käyttää mahdollisimman tietoturvallisesti.

Lähes kaikki modernit ohjelmistoprojektit sisältävät avointa lähdekoodia koodikannassaan, ja ohjelmistotuotteen luominen ilman avointa lähdekoodia on äärimmäisen vaikeaa. Parhaimmillaan avoimen lähdekoodin ohjelmistokirjastot säästävät huomattavasti kehittämiseen kuluvaa aikaa sekä parantavat ohjelmistotuotteen suorituskykyä ja laatua.

Ohjelmistokirjastot ovat kolmannen osapuolen luomaa koodia, jonka vuoksi niitä valitessa ja käyttäessä tulee ottaa huomioon erilaisia asioita tietoturvan varmistamiseksi. Uusia ohjelmistokirjastoja luodaan koko ajan, olemassa olevista kirjastoista julkaistaan uusia versioita ja vanhoja kirjastoja lakataan tukemasta.

2020-luvulla avoimen lähdekoodin ohjelmistokirjastojen kehittäjätahot huolehtivat pääosin tietoturvasta kirjastoa kehittäessään ja ylläpitäessään. Kirjastojen käyttäjätahoilla on kuitenkin oma vastuunsa kirjastojen tietoturvallisuuden arvioimisella. Uusina haasteina ovat esimerkiksi tekoälyn ”hallusinoivat” väärät pakettien nimet.

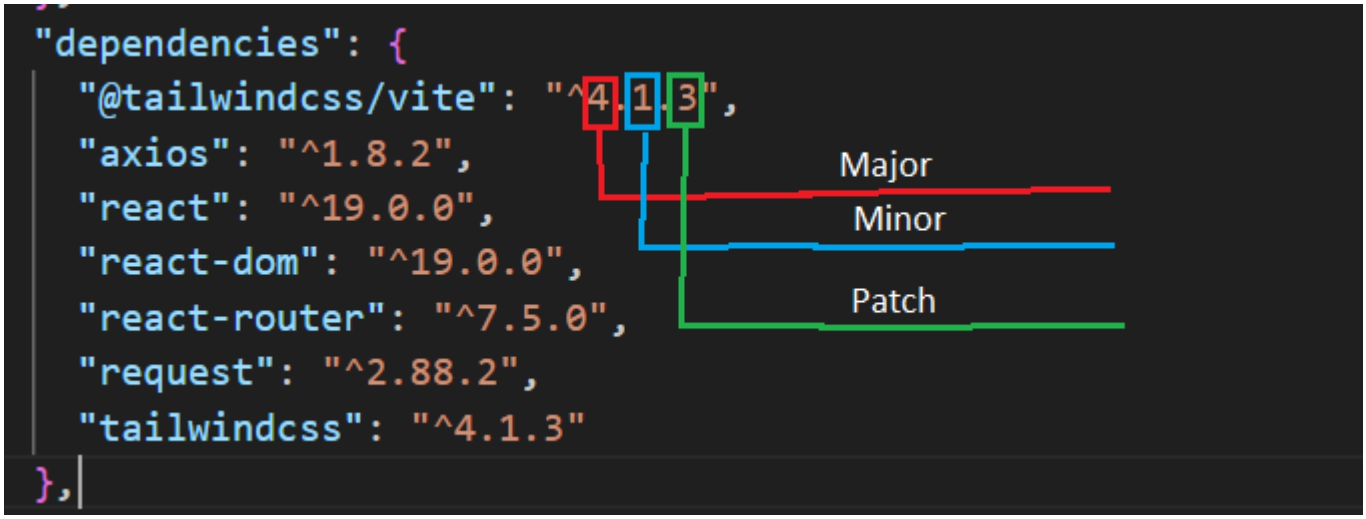
Aliluvuissa on kuvattu lyhyesti muutamia ohjelmistokirjastojen tietoturvaan liittyviä riskejä ja miten niitä voi ehkäistä: tunnetut haavoittuvuudet, toimitusketjuhyökkäykset, typosquatting-hyökkäykset sekä legacykirjastot.

2.1. Tunnetut haavoittuvuudet

Ohjelmistokirjastojen **tunnetut haavoittuvuudet** (eng. known vulnerabilities) ovat kirjastoista löydettyjä, tunnistettuja haavoittuvuuksia. Haavoittuvuudet tarkoittavat heikkouksia tai aukkoja ohjelmistossa, jonka avulla hyökkääjä voi aiheuttaa vahinkoa.

Npm hyödyntää semanttista versiointia ohjelmistokirjastojen hallintaan. Semanttinen versiointi on kokoelma sääntöjä, ja sen ohjeistamana päivitysversiot koostuvat kolmen numeron sarjasta: ensimmäinen on ”Major”, toinen ”Minor” ja kolmas ”Patch” eli Major.Minor.Patch. Minor-versiota käytetään, kun kirjastoon lisätään toiminnallisuutta taaksepäin yhteensopivalla tavalla. Patch-versiota käytetään, kun lisätään taaksepäin yhteensopivia vikakorjauksia.

Esimerkiksi alla olevassa havainnoivassa package.json-tiedostosta otetussa kuvankaappauksessa (kuva 1), kirjaston ”@tailwindcss/vite” versio on 4.1.3. Sen Major-versio on 4, Minor-versio 1 ja Patch-versio 3.



Kuva 1: Semanttinen versiointi

Miten riskiä voi ehkäistä:

- Valitse kirjastoja, joita ylläpidetään aktiivisesti
- Päivitä säännöllisesti käyttämäsi kirjastoversiot
- Vaihda kirjasto, mikäli sitä ei enää ylläpidetä kehittäjäyhteisön toimesta
- Tarkista haavoittuvuuksia sisältävästä tietokannasta, onko kirjastolla tunnettuja haavoittuvuuksia. Useat toimijat ylläpitävät tietokantoja, esimerkiksi GitHub Advisory, Open Source Vulnerabilities (OSV) ja Snykin avoin haavoittuvuustietokanta.

2.2. Toimitusketjuhyökkäykset

Toimitusketjuhyökkäyksiin (eng. supply chain attack) kuuluvat esimerkiksi ohjelmistokirjaston ylläpitäjien tilien kaappaaminen ja/tai haitallisen koodin injektointi ohjelmistokirjastoon.

Avoimen lähdekoodin ohjelmistokirjastot muodostavat riippuvuuksia ohjelmistoprojektiin ja usein npm-paketit ovat riippuvaisia toisista paketeista. Esimerkiksi kirjasto ”axios”, jota käytetään http-pyyntöjen käsittelyyn ja on suosittu kirjasto ohjeiden kirjoitusvaiheessa vuonna 2025, on itse riippuvainen vain kolmesta kirjastosta. Axios-kirjastosta riippuvaisia on noin 150 000 npm-pakettia. Mikäli axios-kirjastossa olisi haavoittuvuus, se vaikuttaisi huomattavaan määrään kirjastoja.

Toimitusketjuhyökkäyksiin liittyviä riskejä voi olla vaikea ehkäistä. Yleisohjeena tulisi käyttää tunnettuja ja luotettavia ohjelmistokirjastoja, joita ylläpidetään aktiivisesti.

2.3. Typosquatting

Typosquatting-hyökkäyksessä hyökkääjätaho luo npm-paketin, jonka nimi muistuttaa ”oikeaa” pakettia. Jos kehittäjä kirjoittaa vahingossa haitallisen paketin nimen pakettia asentaessaan, hän asentaa oikean paketin sijasta haitallisen paketin. Usein tällaiset kirjastot poistuvat npm-rekisteristä nopeasti, mutta niiden suhteen on hyvä olla valppaana.

Typosquatting-hyökkäystä voi olla vaikea ehkäistä, sillä hyökkäys perustuu kehittäjän tekemään puhtaaseen vahinkoon. Riskiä voi joiltain osin ehkäistä erillisellä työkalulla, kuten NPQ-komentorivityökalulla.

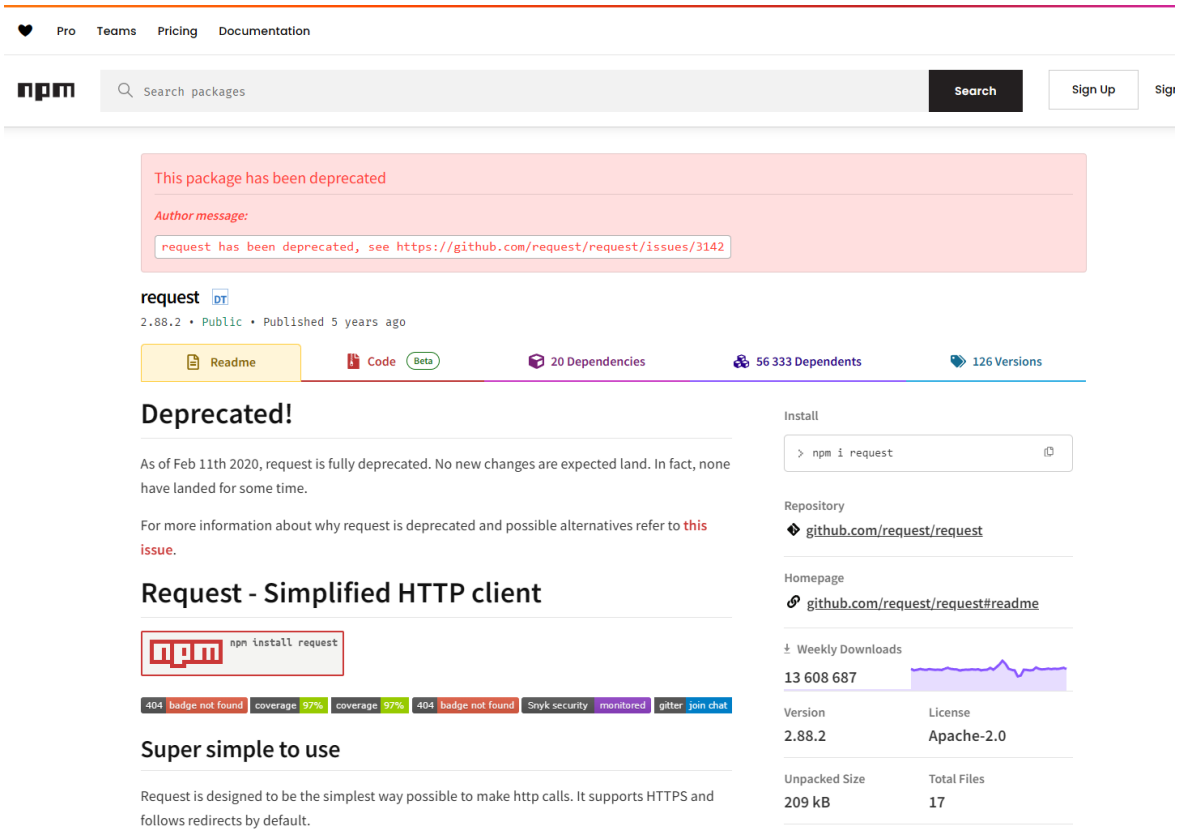
Erityishuomio tekoälyn käyttöön: Tekoäly saattaa ”hallusinoida” pakettien nimiä luodessaan koodia. Mikäli käytät tekoälyn luomaa koodia ja sen ehdottamaa kirjastoa projektissasi, tarkista kirjasto luvussa 3.2. olevan listan avulla.

2.4. Legacykirjastot

Tässä **legacykirjastoilla** tarkoitetaan npm-paketteja, jotka eivät ole enää aktiivisessa kehityksessä. Tällaisten kirjastojen käyttö muodostaa ennen pitkää tietoturvariskin, sillä paketeista saattaa ennen pitkää löytyä haavoittuvuuksia eikä niitä korjata yhteisön toimesta. Legacykirjastot eivät ole pitkän ajan kuluttua enää muutenkaan yhteensopivia omaan ohjelmistoprojektiin, mikäli muita komponentteja päivitetään.

Alla on lueteltu muutamia keinoja tunnistamaan kirjasto, jonka kehitys on lopetettu:

- Npm audit -raportti tai komento ”npm view <paketin nimi>” kertoo kirjaston kehityksen loppuneen
- Npmjs.com-sivustolla lukee ilmoitus: ”This package has been deprecated” (kuva 2)
    - Npmjs.com-sivuston hakutoiminto ei näytä vanhentuneita paketteja omissa hakutuloksissaan. Paketin sivu tulee tällöin etsiä esim. Googlen hakukoneen avulla.
- GitHubin ReadMe-tiedostossa maininta kehityksen lopettamisesta
- Kirjaston omilla dokumentaatiosivuilla lukee jotain, mikä viittaa kirjaston kehityksen lopettamiseen.



Kuva 2: Vanhentuneen request-kirjaston sivu npmjs.com-sivustolla

3. Kirjastojen valinta

Tässä osassa käydään läpi keinoja npm-kirjastojen etsimiseksi ja valitsemiseksi. Sopivuutta arvioidaan ensisijaisesti tietoturvan kannalta. Alaluvussa 3.3 käydään läpi lyhyesti myös kirjastoihin liittyviä lisenssejä.

Uudessa ohjelmistoprojektissa ohjelmistokirjastojen riskejä tulisi käydä läpi sovelluksen turvallisuussuunnittelun yhteydessä, ennen kehityksen aloittamista. Ennen uuden ominaisuuden luomista kannattaa pohtia, onko kirjaston käyttäminen välttämätöntä.

Kirjastoa kannattaa käyttää esimerkiksi reititykseen, lomakkeen validointiin, animointiin ja muihin UI-komponentteihin. Kirjastoa ei lähtökohtaisesti kannata käyttää yksinkertaisiin aputoimintoihin ja erikoistuneen tai toimialuekohtaisen logiikan toteuttamiseksi.

Potentiaalisia kirjastoja voi vertailla esimerkiksi taulukon avulla.

3.1. Vinkkejä kirjastojen etsintään

Npm-kirjastojen etsintään voi käyttää esimerkiksi seuraavia resursseja:

- Npmjs.com – virallinen sivusto
- Npm.io – toinen sivusto, joka listaa npm-paketteja

- Tekoäly, kuten Microsoft Copilot [1]
- Artikkelit ja blogit, joissa vertaillaan kirjastoja [2]
- Npm-komentorivityökalu → komento ”npm search <hakusana>”

[1] **HUOM! Tekoäly, kuten Microsoft Copilot** → Tekoäly saattaa ehdottaa vanhaa versiota kirjastosta tai kirjastoa, jota ei enää tueta. Tekoäly voi myös hallusinoida vääriä kirjastojen nimiä. Tämä altistaa tietynlaiselle typosquatting-hyökkäykselle (luku 2.3.). Tekoäly saattaa olla oiva apu olemassa olevien kirjastojen vertailuun, mutta promptissa kannattaa pyytää lähteitä tekoälyn tuottamalle vertailulle.

[2] **HUOM! Artikkelit ja blogit** → Koska npm-ympäristö muuttuu ajoittain nopeasti, artikkelien ja blogien tarjoama tieto ei ole välttämättä ajantasaista. Yleisohjeena kirjastojen etsinnässä kannattaa olla lähdekriittinen.

3.2. Tarkistus kirjastoja valitessa

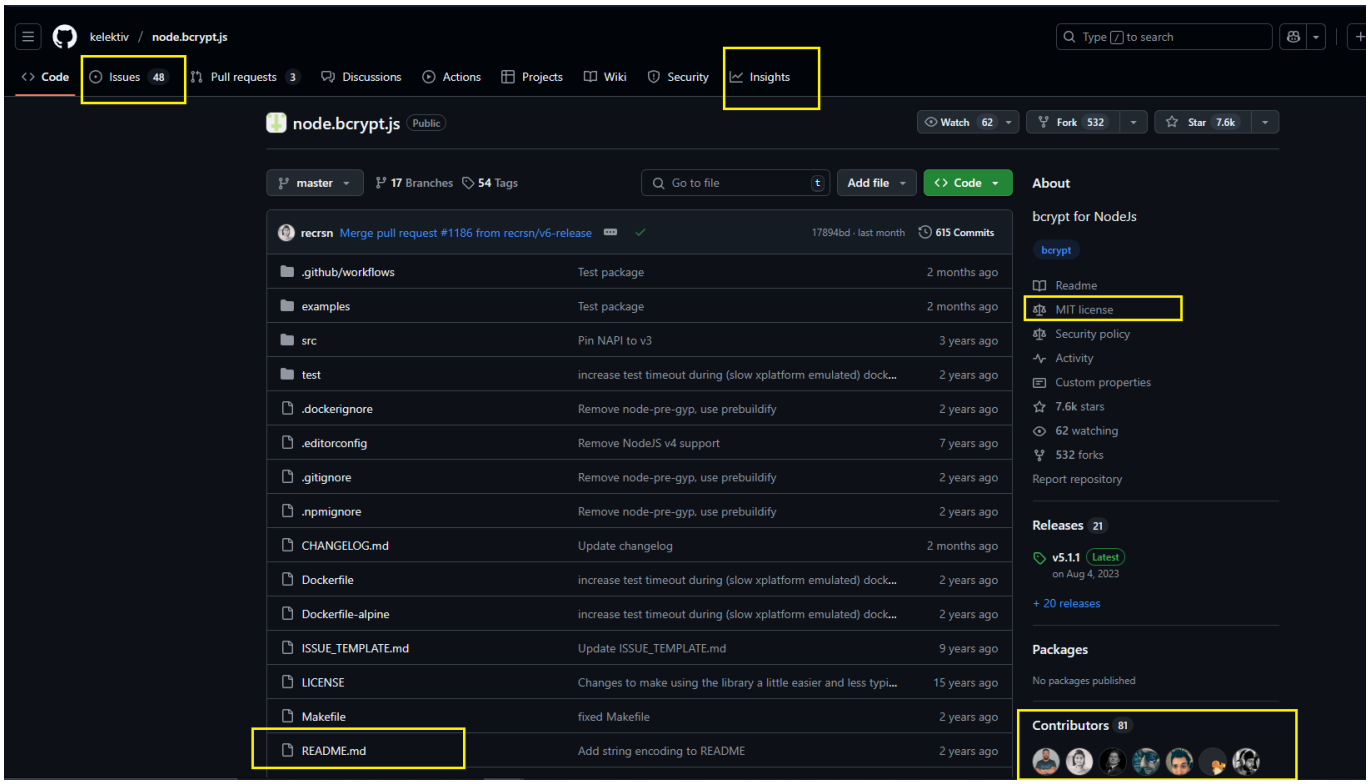
Kartoituksen tarkoituksena on varmistaa kirjaston sopivuus projektiin erityisesti tietoturvan kannalta. Ohjelmistoprojektiin sopivia kirjastoja valitessa tulisi tarkistaa seuraavat asiat:

1. Onko kirjastolla tunnettuja haavoittuvuuksia? Tarkista haavoittuvuuksia listaava julkinen tietokanta, esimerkiksi GitHub Advisory.
2. Npmjs.com, GitHub-repositorio sekä mahdollinen erillinen projektisivu
    a. Dokumentaatio (esim. ReadMe-tiedosto tai kirjaston erillinen projektisivu) – asennatko ylimääräisiä tai raskaita kirjastoja turhaan?
        i. Jos kirjastolla on paljon ylimääräisiä toiminnallisuuksia, voiko kirjaston korvata toisella?
    b. Kirjaston kehitysvaihe
    c. Kirjaston ylläpitäjätahot
    d. Kirjaston yhteensopivuus oman ohjelmistoprojektin ja muiden käytössä olevien kirjastojen kanssa
    e. Lisenssin sopivuus omaan projektiin. Lisensseistä hieman enemmän seuraavassa alaluvussa.

Osan tiedoista saa kiteytetysti myös komentorivillä komennolla ”npm view <kirjaston nimi>”.

Esimerkkinä on kuvankaappaus bcrypt-kirjaston GitHub-repositoriosta, jossa on korostettu keltaisella värillä muutamia huomion arvoisia kohtia projektissa (kuva 3):

- ReadMe-tiedosto (LueMinut)
- Issues (ongelmat) → miten projektissa olevia ongelmia käsitellään
- Insights (kiteyttävät tiedot projektista)
- License (lisenssi)
- Contributors (projektiin osallistuvat) → ylläpitäjät ovat usein listalla ensimmäisenä, sillä he osallistuvat projektiin eniten.



Kuva 3: bcrypt-kirjaston GitHub-repositorio

3.3. Lisensseistä

Avoimen lähdekoodin lisenssejä on yhteensä kymmeniä erilaisia. Niistä sopivia sekä kaupallisiin että ei-kaupallisiin projekteihin ovat esimerkiksi **MIT, Apache 2.0** ja **BSD-3-Clause**.

Osa lisensseistä pakottaa julkaisemaan myös oman tuotoksen avoimena lähdekoodina, jos käyttää projektissaan tietyn lisenssin alaista kirjastoa. Tällainen lisenssi on esimerkiksi **GPL**. Tätä lisenssiä käyttäviä kirjastoja ei suositella omaan ohjelmistoprojektiin, ellei projektia ole ehdottomasti tarkoitus julkaista avoimena lähdekoodina.

Joissain npm-paketeissa ei ole poikkeuksellisesti ollenkaan määritelty lisenssiä. Mikäli tällainen kirjasto tulee vastaan, sitä ei suositella käyttämään oikeudellisen epäselvyyden vuoksi.

Yksittäisen ohjelmistokirjaston lisenssin voi tarkistaa ainakin seuraavin keinoin:

- Kirjaston sivu npmjs.com-sivustolla
- Kirjaston GitHub-repositorio
- Package-lock.json-tiedosto (jos kirjasto on jo asennettu)
- npm-komentorivityökalulla komennolla ”npm view <kirjaston nimi>

4. Kirjaston käyttöönotto

Tässä luvussa on muutamia suosituksia kirjaston käyttöönottoon. On ehdottoman tärkeää, ettei kirjastoja asennettaisi sokkona tai ulkomuistin varassa, vaan ne tarkistettaisiin etukäteen luvussa 3 ehdotettujen keinojen avulla.

Ohjelmistoprojektin luonteesta riippuen voi olla tarpeellista testata kirjastot eristetyssä ympäristössä. Näissä ohjeissa annetaan ohjeet kirjaston käyttöönottamiseksi kuitenkin vain yleisellä tasolla.

- Asennukseen suositeltu: NPQ-komentorivityökalu, joka tarkistaa mm. tunnetut haavoittuvuudet ennen asennusta.
- Harkinnan varainen vaihtoehto: --ignore-scripts -parametri npm install -komennolle, joka estää pre- ja postinstall -skriptien suorituksen. Skripteissä voi olla haitallista koodia.
    - Esimerkki next-kirjaston asennuksesta: npm install --ignore-scripts next
    - Mahdollinen ongelma: Joissain tapauksissa kirjastot tarvitsevat oikeasti skriptejä toimiakseen.
- Npm install -komennon aikana taustalla tarkistetaan myös projektissa käytössä olevien kirjastojen haavoittuvuudet.

5. Testaus / työkalut

Markkinoilla on valtavasti avoimen lähdekoodin ohjelmistokirjastojen tarkistamiseen ja riippuvuuksien hallintaan tarkoitettuja työkaluja. Työkalut ovat joko kaupallisia, ilmaisia tai jotain siltä väliltä. Useat työkalut saattavat olla esimerkiksi ilmaisia yksittäisille kehittäjille, mutta maksullisia tiimeille.

Ohjeissa käydään läpi vain npm auditin ja NPQ-komentorivityökalun käyttö. Molemmat työkalut tarkistavat kirjastoissa olevia tunnettuja haavoittuvuuksia. Npm auditin voi suorittaa milloin vain ja NPQ-työkalua suositellaan kirjaston käyttöönottovaiheeseen.

5.1. Npm audit

Npm audit tarkistaa ohjelmistoprojektissa käytössä olevien kirjastojen tunnetut haavoittuvuudet ja koostaa niistä raportin. Mikäli haavoittuvuuksia löytyy, raportti sisältää tietoa kyseessä olevista haavoittuvuuksista, niiden vakavuustasot ja korjaussuositukset niille.

Kattava lista npm-komentorivin komennoista löytyy npm:n dokumentaatiosivuilta. Tässä muutamia oleellisimpia tärppejä:

Komento *npm audit fix*

→ päivittää automaattisesti yhteensopivat paketit.

Komento *npm audit fix --force*

→ päivittää myös yhteensopimattomat paketit. Tämä voi rikkoa projektin toimivuuden! Suositellaan mieluummin päivittämään kirjastot tarvittaessa yksitellen ja testaamaan projektin toimivuus sen jälkeen.

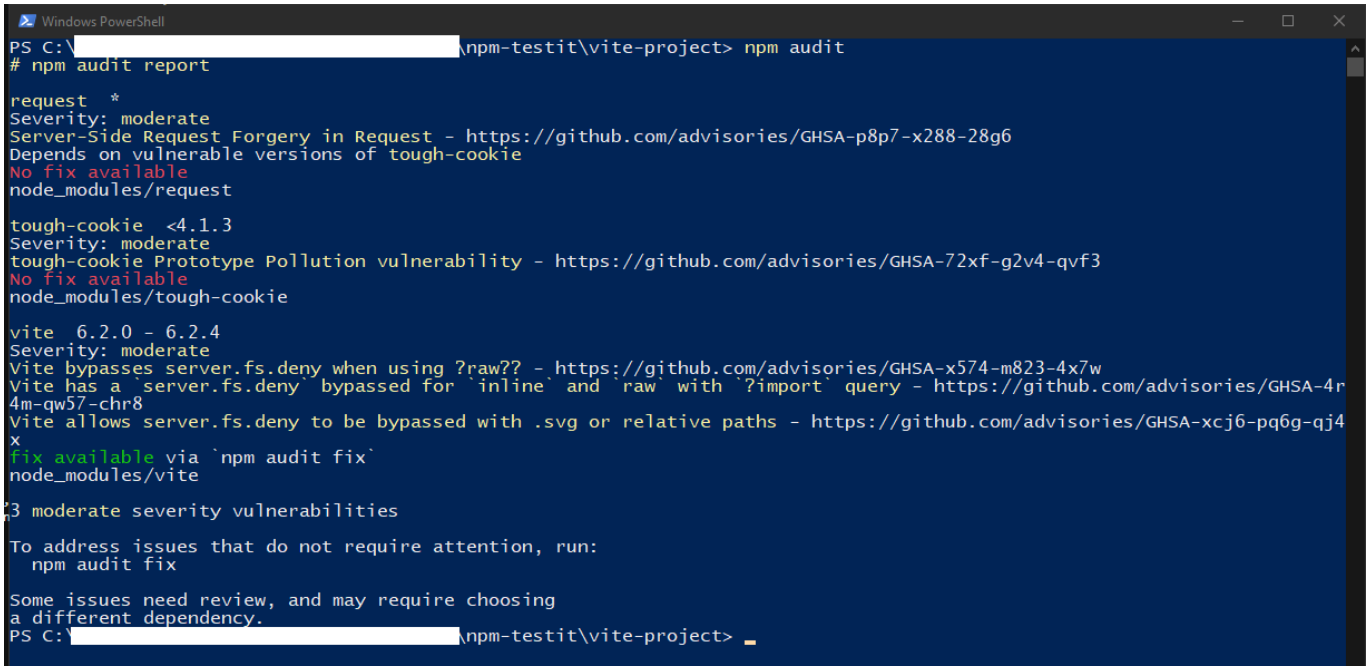
Komento *npm audit --audit-level=null | info | low | moderate | high | critical | none*

→ määrittää pienimmän haavoittuvuuden tason, joka aiheuttaa komennon epäonnistumisen. CI-ympäristössä voi olla hyödyllistä käyttää parametriä.

→ esimerkkinä komennolla *npm audit --audit-level=high* build pysähtyy, jos löytyy suuri tai kriittinen haavoittuvuus.

Kuvassa 4 on esimerkki npm audit -raportista. Muutamia raportista poimittavia asioita ovat:

- request-kirjastolla on yksi haavoittuvuus ja tough-cookie-kirjastolla toinen (requestin aliriippuvuus). Näitä haavoittuvuuksia ei voi korjata, sillä requestin kehitys on lopetettu.
- vite-kirjastolla on kolmas haavoittuvuus. Tämä haavoittuvuus on korjattavissa komennolla audit fix.
- Kaikkien kolmen kirjaston haavoittuvuustaso on ”moderate”.



Kuva 4: Esimerkki npm audit -raportista

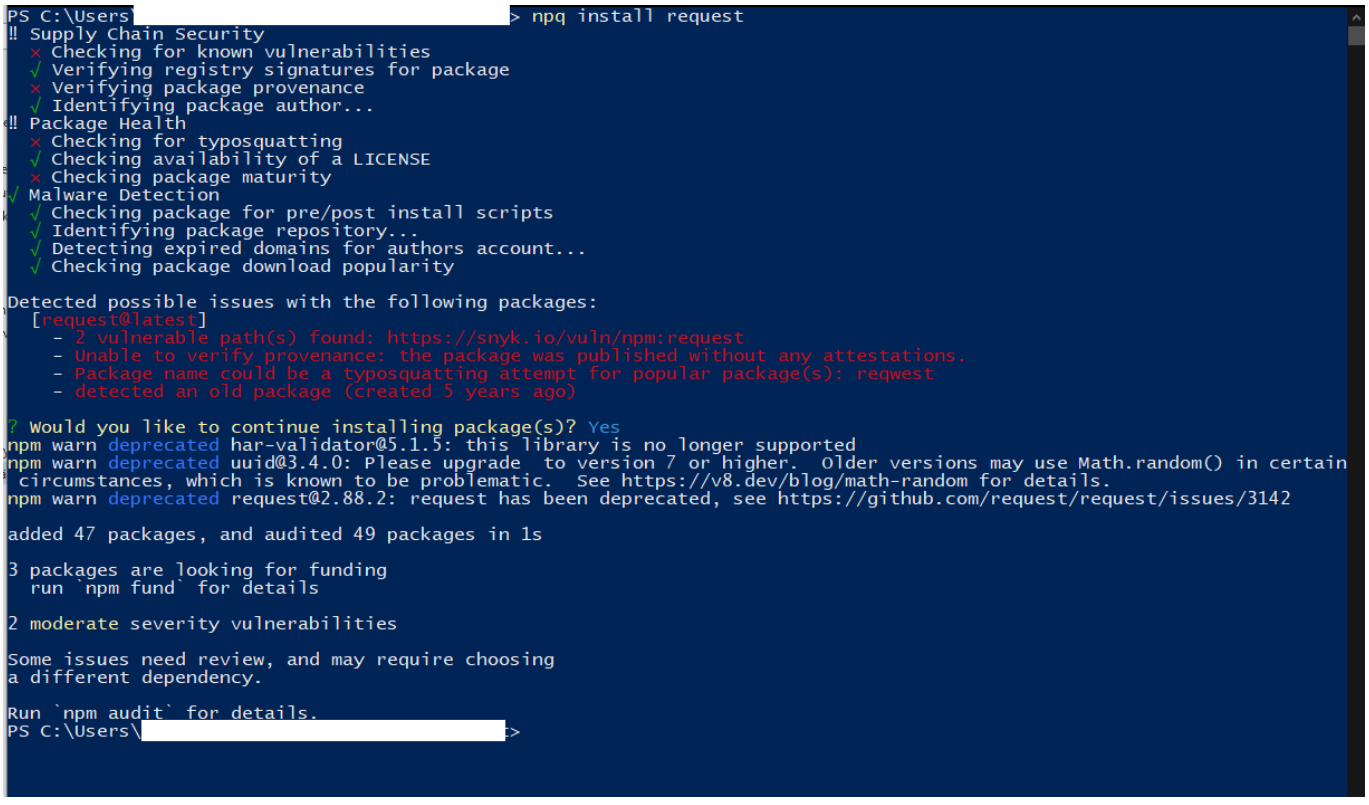
5.2. NPQ-komentorivityökalu

NPQ-komentorivityökalu on npm-pakettien tietoturvahaavoittuvuuksien tarkistukseen käytetty työkalu, jota käytetään komentorivillä. Sen tarkoituksena on täydentää npm:n komentorivityökalua kirjaston käyttöönottovaiheessa.

NPQ on julkaistu npm-pakettina, ja se asennetaan komentorivin avulla. Tarkista ajantasaisin dokumentaatio työkalun sivuilta npmjs.com-sivustolla. Dokumentaatiossa mainittu Snykin API-avaimen käyttö (vaatii Snykin tunnusten omistaminen) työkalun käyttämiseksi ei ole välttämätöntä ohjeiden kirjoitushetkellä.

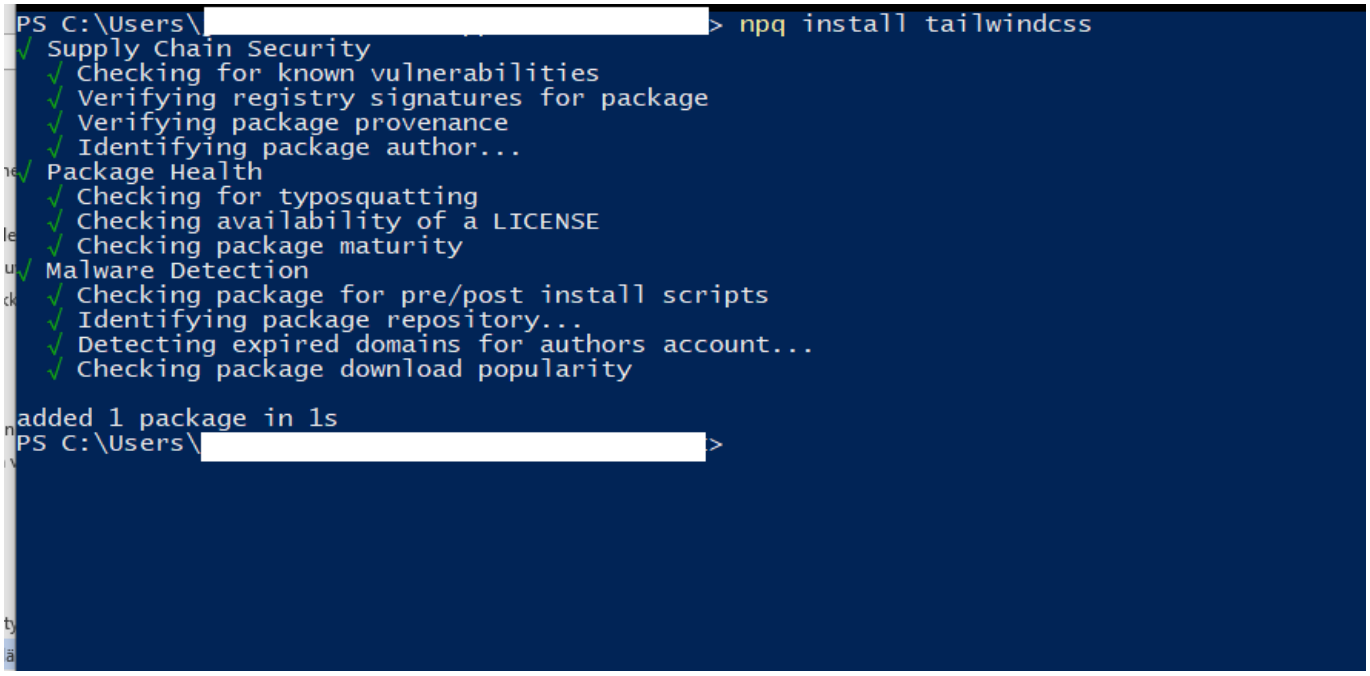
Alla olevissa kuvissa on kaksi esimerkkiä NPQ:n raporteista. Kuvan 5 raportissa työkalun tarkistukset eivät mene läpi. Huomiona on, että NPQ:n raportin perään tulostuu automaattisesti myös npm-raportti.

Koska NPQ pyytää erillisen varmistuksen asennukseen sen löytäessä jotain huomautettavaa, sen avulla on hieman turvallisempaa asentaa npm-paketteja ns. sokkona, vaikka se ei ole suositeltavaa.



Kuva 5: NPQ-raportti, kun työkalu löytää asennettavasta npm-paketista jotain huomautettavaa.

Kuvassa 6 on esimerkki raportista, kun työkalu ei löydä asennettavasta npm-paketista moitittavaa.



Kuva 6: NPQ:n raportti, kun työkalu ei löydä asennettavasta kirjastosta moitittavaa.

6. Ohjelmistoprojektin ylläpito

Ohjelmistokirjastojen ylläpito on tasapainottelua liian vanhojen ja liian uusien versioiden välillä. Joissain tapauksissa liian nopea päivitys voi altistaa sille, että uuden version mukana tulee haitallista koodia tai uusi versio rikkoo jotain.

Tiimien tulee huomioida ja sopia seuraavat asiat:

- Kuinka usein kirjastojen päivitystarpeita kartoitetaan ja kirjastoja päivitetään uuteen versioon? Onko riippuvuuksien päivittäminen sidottu ohjelmistotuotteen versiopäivittämiseen?
    - Kartoitukseen voi kuulua myös käyttämättä jääneiden kirjastojen karsiminen.
- Kuka tai ketkä päivittävät projektissa olevat riippuvuudet (eli kirjastot)?
- Ohjelmistoprojektin toimivuuden testaus kirjastojen versiopäivitysten jälkeen
- Toimintaohjeet tilanteisiin, jolloin kirjasto tulisi päivittää välittömästi tai ottaa kokonaan pois käytöstä
    - Mahdollisesti esim. kriittinen haavoittuvuus, johon ei ole saatavilla korjausta tai vanhentunut kirjasto.
- Käytännöt kirjaston vaihtamiseen tai käytöstä pois ottamiseen: kuka tai ketkä päättävät?

Tiimien on mahdollista vaihtaa ohjelmistokirjastojen päivitykseen liittyviä käytäntöjä projektin aikana. Tärkeintä on kuitenkin se, että käytännöt ovat olemassa ja kaikki ovat tietoisia niistä.

Kirjastojen versioiden ja tunnettujen haavoittuvuuksien monitorointiin suositellaan käytettäväksi automaattista seurantaa, sillä erityisesti kirjaston aliriippuvuuksien seuraaminen manuaalisesti on hankalaa sen suuren määrän vuoksi.

Suositeltu strategia kirjastojen päivittämiseen:

1. Komento ”npm outdated” → tarkistaa, onko kirjastoihin saatavilla päivityksiä
2. Uudempien Minor- ja Patch -päivitysten asennus könttänä
3. Major-versiopäivitykset yksitellen
4. Ohjelmiston toimivuuden testaus jokaisen Major-päivityksen jälkeen.

7. Hyödyllisiä resursseja

Työkaluja

Node.js (esim. asennusohjeet): https://nodejs.org/en.

Koottu dokumentaatio npm:n komennoista: https://docs.npmjs.com/cli/v11/commands.

NPQ-työkalu: https://www.npmjs.com/package/npq.

Hakukoneita npm-paketeille

Npm:n virallinen sivu ja hakukone npm-paketeille: https://www.npmjs.com.

Epävirallinen hakukone npm-paketeille: https://npm.io.

Tietokantoja tunnetuille haavoittuvuuksille

OSV Database: https://osv.dev.

GitHub Advisory Database: https://github.com/advisories.

Snyk Vulnerability Database: https://security.snyk.io.