Joonas Erho

npm-paketinhallintajärjestelmän käytännön ongelmat ja riskit

Tietotekniikan kandidaatintutkielma

28. huhtikuuta 2023

Jyväskylän yliopisto

Informaatioteknologian tiedekunta

Tekijä: Joonas Erho

Yhteystiedot: joonas.a.erho@student.jyu.fi

Ohjaaja: Antti-Jussi Lakanen

Työn nimi: npm-paketinhallintajärjestelmän käytännön ongelmat ja riskit

Title in English: The practical problems and risks of the npm package manager

Työ: Kandidaatintutkielma

Opintosuunta: Kaikki opintosuunnat

Sivumäärä: 24+1

Tiivistelmä: Tässä kandidaatintutkielmassa avataan npm-paketinhallintajärjestelmän toimintaa ja siihen liittyviä riskejä ja ongelmia. Näihin riskeihin kuuluvat muun muassa pakettien ylläpitäjien huolimattomuudesta johtuvat haavoittuvuudet, sekä haavoittuvuuksien leviämistä edesauttavat tekijät, kuten laaja vanhentuneiden tai triviaalien pakettien käyttö. Tutkielmassa pohditaan myös keinoja ennaltaehkäistä ja ratkaista edellämainittuja ongelmia.

Avainsanat: JavaScript, paketinhallintajärjestelmä, npm, Node.js, riippuvuussuhteet, semanttinen versiointi

Abstract: In this bachelor's thesis the general procedures and features of the npm package manager are explained, with focus on the possible risks and other issues pertaining to the usage of npm. These risks include the security vulnerabilites due to human errors made by package administrators, as well as factors that help spread these vulnerabilities, including the usage of deprecated or trivial packages. In this thesis, we also discuss methods to prevent or fix these issues.

Keywords: JavaScript, package manager, npm, Node.js, dependencies, semantic versioning

Sisällys

1 Johdanto

JavaScript on komentokieli (engl. scripting language), jonka avulla web-kehittäjät voivat lisätä nettisivuille toiminnallisuutta (MDN Web Docs 2023a). Se on nykypäivänä yksi tärkeimmistä web-teknologioista, sillä yli 98 % nettisivuista käyttää JavaScriptiä (W3Techs 2023). JavaScriptin suuri suosio näkyy myös verratessa muihin ohjelmointikieliin. JavaScript oli lokakuussa 2017 GitHub-versionhallintajärjestelmässä ylivoimaisesti suosituin ohjelmointikieli. Yli 2,3 miljoonaa avoimen lähdekoodin projektia käytti sitä, mikä oli yli tuplasti enemmän kuin seuraavaksi suosituimmalla ohjelmointikielellä Pythonilla (Decan, Mens ja Constantinou 2018). Vuonna 2022 JavaScript oli edelleen ohjelmointikielistä ykkössijalla (GitHub 2023).

Vuonna 2009 julkaistu Node.js-moottori salli JavaScriptin ajamisen selaimen ulkopuolella. Tämä teki JavaScriptistä sopivan kielen myös muuhunkin kuin nettisivujen toiminnallisuuden toteuttamiseen. Vuosi myöhemmin julkaistiin koodikirjastojen jakamista varten luotu Node.js-ympäristössä toimiva npm-paketinhallintajärjestelmä (engl. package manager) (The Server Side 2018). Paketinhallintajärjestelmien avulla ulkoisen koodin lisääminen omaan projektiin on helppoa ja pitkälti automatisoitua (MDN Web Docs 2023b).

Node.js:n mukana julkaistu npm-paketinhallintajärjestelmä oli aluksi vain yksi monista kilpailevista JavaScript-paketinhallintajärjestelmistä (The Server Side 2018), mutta Noden tuoman suosion kautta siitä kasvoi kaikista paketinhallintajärjestelmistä suurin (Paul Brown 2017). Vuonna 2017 npm-pakettirekisterissä (engl. package registry) oli jo lähes 400,000 koodikirjastoa eli pakettia, ja maaliskuussa vuonna 2023 rekisterissä on jo lähes kolme miljoonaa pakettia (libraries.io 2023).

Noin 80 % ohjelmien koodista tulee ulkoisista kirjastoista (Decan, Mens ja Constantinou 2018), ja on mahdollista, että nykypäivänä, ja erityisesti JavaScript-ohjelmien tapauksessa, tämä luku on vielä suurempi npm-paketinhallintajärjestelmän suosion vuoksi. Vaikka valmiin koodin hyötykäyttäminen onkin houkutteleva ja usein myös järkevä vaihtoehto, ei pakettien käyttäminen julkisista rekistereistä kuten npm-rekisteristä kuitenkaan ole täysin ongelmatonta. npm-rekisterissä olevista paketeista merkittävä osa pitää sisällään haavoittuvuu-

den (Decan, Mens ja Constantinou 2018), ja liiallinen ulkoisten koodikirjastojen käyttö voi johtaa vaikeuksiin projektin ylläpidon ja päivittäminen kanssa (Chen, Abdalkareem ja Mujahid 2021).

Tässä tutkielmassa tutustutaan näihin npm-paketinhallintajärjestelmän käyttöön liittyviin ongelmiin ja riskitilanteisiin. Esimerkiksi Decan, Mens ja Constantinou (2018) kertovat tutkimuksessaan, että noin 25 % kaikista npm-rekisterissä löytyvistä paketeista pitää sisällään tietoturvariskejä, ja että jopa 77 % nettisivuista sisältää ohjelmakoodissaan vähintään yhden tietoturvariskejä sisältävän koodikirjaston tai riippuvuuden. Tietoturvariskien laaja esiintyvyys voi johtua vanhentuneiden (Cogo, Oliva ja Hassan 2022), ja triviaalien eli tyypillisesti hyvin pienien ja helposti toteutettavien pakettien (Chen, Abdalkareem ja Mujahid 2021) käytöstä.

Tämän lisäksi selvitetään, kuinka npm-paketinhallintajärjestelmä tai siihen suorassa yhteydessä olevat sovellukset pyrkivät ratkaisemaan edellämainittuja ongelmia ja riskejä. Pohditaan myös, kuinka sekä pakettien julkaisijat että niitä käyttävät ohjelmistokehittäjät voivat minimoida pakettien käyttöön liittyvät riskit.

Toisessa luvussa käydään läpi npm-paketinhallintajärjstelmän toimintaa ja avataan paketinhallintajärjestelmän perusominaisuuksia, kuten riippuvuussuhteita ja pakettien versiointia. Kolmannessa luvussa kerrotaan npm-pakettien haavoittuvuuksien esiintyvyydestä, laadusta ja niiden löytämisen ja korjaamisen tehokkuudesta. Neljännessä luvussa keskitytään haavoittuvuuksien leviämiseen merkittävästi vaikuttaviin vanhentuneihiin paketteihin, ja viidennessä luvussa vastaavasti triviaaleihin paketteihin. Kuudennessa luvussa pohditaan syitä riskien alkuperälle ja keinoja ennaltaehkäistä niiden syntyä ja leviämistä. Pohdintaosiossa kerrotaan tämän tutkielman aiheen kehittymisestä ja mahdollisuuksista aiheeseen liittyvälle lisätutkimukselle. Yhteenvedossa käydään läpi tärkeimmät tutkielman johtopäätökset.

2 npm-paketinhallintajärjestelmä

2.1 npm-pakettirekisteri

Paketinhallintajärjestelmien avulla voidaan ladata muiden osapuolien tekemiä koodikirjastoja osaksi omaa projektia. Näitä koodikirjastoja kutsutaan yleisesti paketeiksi (engl. package). npm on vain yksi paketinhallintajärjestelmistä. Muita paketinhallintajärjestelmiä ovat muun muassa NuGet Microsoftin .NET-ympäristölle, ja pip Python-kielelle. Järjestelmän ominaisuuksiin kuuluu myös pakettien lähettäminen ja päivittäminen jonkinlaiseen rekisteriin (engl. package registry), jossa paketit säilyvät ja josta niitä voidaan ladata. npm-paketinhallintajärjestelmällä rekisteri on samanniminen npm-pakettirekisteri. (MDN Web Docs 2023b). npm-pakettirekisterissä on lähes 3 miljoonaa pakettia (libraries.io 2023), ja sitä hyödyntää yli 17 miljoonaa ohjelmistokehittäjää (npm 2023c).

2.2 package.json-tiedosto

Paketinhallintajärjestelmissä paketteihin liittyvät tiedot määritellään jonkinlaisessa konfiguraatiotiedostossa. npm-paketinhallintajärjestelmää hyödyntävissä projekteissa tiedot määritellään `package.json`-tiedostossa, joka sijoitetaan projektin juureen (MDN Web Docs 2023b). Tiedostossa määritellään muun muassa paketin nimi, versio ja tekijä, joiden perusteella paketti voidaan löytää (npm 2023d).

Konfiguraatiotiedostossa määritellään myös projektissa käytettävät ulkoiset paketit eli riippuvuudet (engl. dependency) ja niiden versiot. Uusien pakettien lisääminen projektiin npm:n komentorivityökalun (engl. command line interface) kautta lisää ne myös `package.json`-tiedostoon. Näin projektin vaatimat paketit on määritetty projektissa ilman, että kaikkia tarvittavia paketteja tarvitsisi esimerkiksi ladata versionhallintaan. Varsinaiset pakettien sisältämät tiedostot ladataan `node_modules`-kansioon, josta projektin tiedostot osaavat niitä käyttää. (npm 2023d)

```
{
  "name": "kandi-esimerkki",
```

```
  "version": "1.0.0",
  "description": "Esimerkki Node-projektista joka hyödyntää
  npm-paketinhallintajärjestelmää.",
  "main": "index.js",
  "scripts": {
    "test": "test"
  },
  "author": "Joonas Erho",
  "license": "MIT",
  "dependencies": {
    "chalk": "^5.2.0"
  },
  "devDependencies": {
    "eslint": "^8.36.0"
  }
}
```

Esimerkillinen `package.json`-tiedosto, joka on syntynyt npm:n komentorivityökalun komennoilla `npm init`, `npm install chalk` ja `npm install --save-dev eslint`. Tiedostosta voidaan nähdä, että `chalk`-niminen paketti on lisätty projektin riippuvuudeksi. Kyseinen paketti helpottaa konsoliin tulostettavan tekstin koristelemista esimerkiksi värein tai muun muotoilun kautta. `chalk`-paketin koodia voidaan siis käyttää kyseisessä projektissa. Myös `eslint`-paketti, joka avustaa kehittäjää pitämään lähdekoodin siistinä ja yhtenäisenä, on projektin riippuvuutena, mutta ainoastaan kehitysympäristössä. Projekteihin voi siis liittää myös paketteja, joita ei kuitenkaan pakata mukaan julkaistavaan tuotteeseen.

2.3 Riippuvuussuhteet

Yllä olevassa esimerkissä projektin riippuvuuksiksi lisättiin kaksi pakettia npm-rekisteristä. Projektin ja pakettien välille on siis luotu riippuvuussuhde, jossa projekti on riippuvainen kyseisistä paketeista (MDN Web Docs 2023b).

Riippuvuussuhteet ovat hyvin keskeinen käsite kaikissa paketinhallintajärjestelmissä. Koko pakettiekosysteemin (engl. package ecosystem) toiminta perustuu siihen, että paketit voivat olla riippuvaisia toisistaan, ja että paketilla, josta jokin paketti riippuu, voi olla myös omia riippuvuuksia. Nämä riippuvuudet määritellään paketinhallintajärjestelmästä riippuen jonkinlaisella konfiguraatiotiedostolla, esimerkiksi npm-järjestelmässä edellä mainitulla `package.json`-nimisellä tiedostolla (MDN Web Docs 2023b).

npm-paketinhallintajärjestelmässä on monia ominaisuuksia, jotka kätevöittävät riippuvuussuhteiden käyttöä projektissa. npm osaa esimerkiksi automaattisesti olla asentamatta useaa kopiota jostain paketista, vaikka se olisi usean paketin omana riippuvuussuhteena. npm osaa kuitenkin sallia saman paketin eri versioiden käytön samassa projektissa (The Server Side 2018). Näitä riippuvuussuhteita voidaan tutkia `npm list –all` -komennolla, jolla voidaan rekursiivisesti tutkia riippuvuussuhteiden ketjuja. Yllä olevan esimerkkiprojektin riippuvuussuhteiden rakenne on esitetty liitteessä A.

Listauksesta voi myös huomata, että `chalk`-paketilla ei ole omia riippuvuussuhteita. Sen asentaminen ei siis lisää projektiin muuta kuin kyseisen kirjaston. Tähdellisempää tämän tutkimuksen kannalta on kuitenkin `eslint`-paketin riippuvuuksien määrä. Osia listauksesta on leikattu pois, sillä `eslint`-paketilla on yhteensä 40 suoraa riippuvuutta (npm 2023b), jonka lisäksi useilla sen riippuvuuksilla on vielä omia riippuvuuksia.

Vaikka komentoriviltä asentaisi vain yhden paketin, voi suuri riippuvuussuhteiden määrä johtaa siihen, että projektiin tuleekin huomattavasti suurempi määrä paketteja. `eslint` ei ole riippuvuussuhteidensa määrältä poikkeuksellinen paketti. Keskimäärin npm-paketilla on lähes 80 riippuvuussuhdetta, kun otetaan rekursiivisesti kaikkien riippuvuuksien omat riippuvuussuhteet huomioon. Lisäksi npm-rekisterissä noin 80 % kaikista paketeista riippuu vähintään yhdestä toisesta paketista, ja noin 25 % riippuu yli kuudesta eri paketista (Wittern, Suter ja Rajagopalan 2016), joten ohjelmistokehittäjien on vaikea välttää riippuvuussuhteiden kautta useiden pakettien lataamista. Decan, Mens ja Constantinou (2018) mukaan noin 80 % kaikesta ohjelmakoodista on peräisin ulkoisesti ladatuista kirjastoista.

2.4 Semanttinen versiointi

npm-paketit versioidaan tavalla, josta käytetään nimeä semanttinen versiointi (engl. semantic versioning) (npm 2023a). Semanttisessa versioinnissa versiointi rakentuu kolmesta numerosta. Ensimmäinen kertoo suurista muutoksista (engl. major release), jotka voivat potentiaalisesti rikkoa vanhoja versioita kyseisestä ohjelmistosta, esimerkiksi muuttamalla jotain ohjelmiston perustoimintoa. Toinen numero kertoo pienemmistä muutoksista (engl. minor release), joissa on usein uusia ominaisuuksia, mutta jotka eivät vaikuta tuhoisasti olemassa oleviin ohjelmiin. Kolmas numero kertoo pienistä korjauksista (engl. patch), jotka eivät välttämättä vaikuta juuri ollenkaan ohjelmiston toimintaan. (semver 2023)

Semanttisen versioinnin avulla pakettien kehittäjät voivat määrittää `package.json` -tiedostossa, mitä versiota tai versioita jostain riippuvuudesta kuuluisi käyttää. Kehittäjä voi esimerkiksi sallia patch- tai minor-tasoiset päivitykset, mutta estää riippuvuuden päivittymiseen uuteen major-versioon. Näin rikkovia muutoksia ei pääse itsestään tulemaan riippuvuuksiin. Tämä voi kuitenkin johtaa myös tietoturvaongelmiin, sillä semanttisen versioinnin käyttö voi johtaa hyvinkin vanhojen versioiden käyttöön, joiden tietoturva ei ole välttämättä ajan tasalla (Decan ja Mens 2021).

3 Pakettien haavoittuvuudet

3.1 Haavoittuvuuksien esiintyvyys

Decan, Mens ja Constantinou (2018) tekemässä tutkimuksessa, jossa tutkittavana oli noin 600,000 npm-pakettirekisterissä olevaa pakettia kävi ilmi, että vaikka näistä vain 269 pakettia piti sisällään haavoittuvuuden, riippuvuussuhteiden myötä nämä haavoittuvuudet vaikuttivat jopa yli 133,000 pakettiin. Tämä vahvistaa käsitystä siitä, että jo yksittäisessäkin paketissa oleva tietoturvaongelma voi johtaa valtavaan määrään tietoturvariskejä. Esimerkiksi yli 80 miljoonaa kertaa kuukaudessa riippuvuutena ladatussa `lodash`-kirjastossa havaittiin vakava tietoturvariski, joka johti yli 4,35 miljoonan GitHub-projektin haavoittuvuuteen (Liu, Chen ja Fan 2022).

Koska suuret projektit saattavat käyttää suuria määriä ulkoisia paketteja, haavoittuvuuden esiintyminen projektissa kasvaa hyvinkin todennäköiseksi. Lisätessään yhden paketin npm-rekisteristä käyttäjä tuo projektiinsa koodia keskimäärin 79 paketista ja 39 eri ylläpitäjältä (Zimmermann ym. 2019). Decan, Mens ja Constantinou (2018) mukaan jopa lähes 80 % nettisivuista käyttää jotain front-end -kirjastoa, jossa on haavoittuvuus.

Haavoittuvuudet esiintyvät myös usein useassa paketin julkaistussa versiossa. 269:n tietoturvattomien pakettien haavoittuvuuksia sisältäneiden julkaistuja versioita oli yhteensä lähes 15,000 (Decan, Mens ja Constantinou 2018). Keskimäärin haavoittuneita versioita on noin 55 haavoittunutta pakettia kohden. Haavoittuvuudet esiintyvät siis todennäköisesti paketin useissa eri versioissa. Yli 90 % julkaistuista versioista pitää sisällään tietoturvariskin jopa kolmessa neljästä paketista, joista on löydetty haavoittuvuus (Decan, Mens ja Constantinou 2018). Jos jokin projekti tai toinen paketti käyttää haavoittunutta pakettia, on todennäköistä, että käytetty versio on tietoturvariski riippumatta versiosta.

3.2 Haavoittuvuuksien laatu

Decan, Mens ja Constantinou (2018) tutkimuksessa löydetyillä haavoittuvuuksilla on CVSS-luokittelun mukaan määritelty vakavuusluokka, joka on matala (low), keskitaso (medium) tai

korkea (high). Luokittelussa otettiin huomioon, kuinka helppoa haavoittuvutta on käyttää hyväksi, ja kuinka vakavia seurauksia haavoittuvuudella voi olla (Decan, Mens ja Constantinou 2018).

Valtaosa Decan, Mens ja Constantinou (2018) löytämistä haavoittuvuuksista on määritelty keskitason tai korkean vakavuuden haavoittuvuuksiksi. Kirjoittajat pohtivat, että tämä voi johtua siitä, ettei matalamman luokitustason haavoittuvuuksista välttämättä edes tehdä viralisia raportteja, koska niitä ei pidetä niin vakavina.

3.3 Haavoittuvuuksien löytö ja korjaus

Vaikka lähes kaikki Decan, Mens ja Constantinou (2018) tutkimuksessa esiintyneet haavoittuvuudet on lopulta korjattu, pakettirekisterissä olevien haavoittuvuuksien löytämiseen menee keskimäärin pitkä aika. Vain 25 % haavoittuvuuksista löydetään alle 13 kuukaudessa, ja vastaavasti 25 % löytyy vasta yli 46 kuukauden jälkeen. Decan, Mens ja Constantinou 2018 perustelevat tätä myös sillä, että mahdollisesti uudemmat paketit eivät ole vielä niin laajassa käytössä, minkä vuoksi haavoittuvuuksiakaan ei löydy niin helposti.

Noin 30 % haavoittuvuuksista on korjattu jo ennen kuin haavoittuvuus on virallisesti löydetty. Tämä johtuu mahdollisesti Decan, Mens ja Constantinou 2018 mukaan siitä, että ylläpitäjät eivät halua antaa hyökkääjille mitään tietoa haavoittuvuudesta ennen kuin se on korjattu, erityisesti jos haavoittuvuus on luokiteltu vakavaksi. Yli 80 % kaikista haavoittuvuuksista korjataan ennen kuin niistä ilmoitetaan julkisesti. (Decan, Mens ja Constantinou 2018)

Vaikka noin puolet pakettien haavoittuvuuksista korjataan kuukauden sisällä niiden löytymisestä ja noin 83 % vuoden sisällä, kaikkien haavoittuneesta paketista riippuvaisten pakettien korjaaminen kestää huomattavasti pidempään. 14 kuukauden jälkeen vasta 50 % paketeista, joilla on riippuvuus haavoittuneesta paketista, ovat korjanneet ongelman päivittämällä haavoittuneen paketin riippuvuussuhteen uudempaan versioon. Tämä tuo ilmi merkittävän ongelman paketinhallintajärjestelmien käytössä. Projektien ylläpitäjien pitäisi jatkuvasti tarkkailla kaikkia projektin riippuvuussuhteita haavoittuvuuksien varalta, jotta haavoittuvuudet voidaan korjata nopeasti (Decan, Mens ja Constantinou 2018). On myös tärkeää huomioida, että jopa 90 % pakettien haavoittuvuuksista esiintyvät yli kolmen riippuvuussuhteen päässä

paketista (Liu, Chen ja Fan 2022). Tämä tarkoittaa sitä, että jotta paketin haavoittuvuus saadaan korjattua, täytyy kaikkien pakettien tämän paketin ja haavoittuneen riippuvuussuhteen välillä korjata, mikä voi viedä merkittävästi aikaa.

3.4 Muut hyökkäykset

Merkittävä turvallisuusriski npm-paketinhallintajärjestelmää käyttäessä voivat olla haavoittuvuuksien hyväksikäyttämisen lisäksi myös suoremmat, tahalliset hyökkäykset. Esimerkiksi vuonna 2018 suositun `eslint-scope`-paketin ylläpitäjän käyttäjätunnusten paljastuminen johti hyökkäykseen, jossa hyökkäjä pääsi ylläpitäjän nimissä lisäämään pakettiin haitallista koodia (Zimmermann ym. 2019). Ylläpitäjät eivät tätä hyökkäystä havainneet, ja kun käyttäjät viimein huomasivat sen, hyökkääjä oli ehtinyt varastaa jo yli 4500 käyttäjän tiedot (Scalco ym. 2022).

Zahan ym. (2022) löysivät tutkimuksessaan, että jopa 2,818 npm-käyttäjällä oli vanhentunutta verkkotunnusta käyttävä sähköpostiosoite. Mahdollinen hyökkääjä olisi voinut ostaa käyttäjän aikaisemman verkkotunnuksen, ja päästä siten käsiksi tämän tiliin. Nämä ylläpitäjät hallitsivat npm-rekisterissä yli 8 tuhattaa pakettia, joista kustakin riippui keskimäärin 2.43 pakettia. Tutkimuksen mukaan kuitenkin tämä lukumäärä saattaisi olla suurempikin. Saman tutkimuksen mukaan paketin voi tehdä hyökkäysalttiiksi myös liian suuri ylläpitäjien tai ylipäätään kehittäjien määrä, pakettien kehityksen pitkäaikainen epäaktiivisuus tai liian työllistynyt ylläpitäjä, jolla ei ole aikaa ylläpitää huolellisesti kaikkia pakettejaan.

4 Vanhentuneet paketit

Vanhentuneiden (engl. deprecated) pakettien tai pakettien vanhentuneiden versioiden käyttö voi altistaa haavoittuvuuksille npm-ympäristössä. npm-rekisterissä 54 % paketeista on riippuvaisia jostain vanhentuneesta paketista tai versiosta, vaikka kyseiset paketit on määritelty vanhentuneiksi nimenomaan viestimään sitä, että niiden käyttöä tulisi välttää. npm-pakettien tapauksessa 49 % paketeista on merkitty vanhentuneiksi, koska niitä ei enää ylläpidetä, ja 45 %, koska niiden käyttö on tarkoitus korvata jollain toisella paketilla. Yksittäisen npm-pakettien aikaisemmat versiot taas on yleisimmin – 63 % ajasta – vanhennettu vian tai tietoturvariskin takia. (Cogo, Oliva ja Hassan 2022)

Vaikka npm tarjoaakin tapoja ilmoittaa kehittäjille, että heidän käyttämissään riippuvaisuuksissa on vanhentuneita paketteja tai versioita, npm-pakettirekisterissä 27 % paketeista on suoraan riippuvaisia ja 54 % epäsuoraan riippuvaisia vähintään yhdestä vanhennetusta paketista (Cogo, Oliva ja Hassan 2022). Vaikka Cogo, Oliva ja Hassan (2022) kertovat tutkimuksessaan, ettei vanhentuneiden versioiden tietoturvariskejä voida helposti suoraan määrittää, Decan, Mens ja Constantinou (2018) suosittelevat omassa artikkelissaan aina välttämään vanhentuneita paketteja ja versiota, sillä riski altistua haavoittuvuudelle on korkeampi kuin ajan tasalla olevien versioiden kanssa.

Semanttisen versioinnin oikeaoppinen käyttö ilmenee myös vanhentuneita versioita tutkiessa. Kuten Cogo, Oliva ja Hassan (2022) tutkimuksessa selvisi, 63 % pakettien versioista vanhennetaan vian takia, joka on sittemmin korjattu uudessa versiossa. Decan ja Mens (2021) huomasivat omassa tutkimuksessaan, että 40 % paketeista, joissa on haavoittuvuus, voitaisiin korjata päivittämällä semanttinen versiointi vähemmän tarkaksi. Huonon semanttisen versioinnin takia saattaa siis altistaa oman projektinsa haavoittuvuuksille, kun npm ei päivitäkään vanhentunutta versiota uudeksi. Tämä ei kuitenkaan ole välttämättä aina pakettia riippuvuutena käyttävän kehittäjän vika. Chinthanet ym. (2021) kertovat tutkimuksessaan, että npm-pakettien haavoittuvuuksia korjaavissa päivityksissä yli 85 % on muita päivityksiä kuin itse haavoittuvuuden korjaava koodi. Voi siis olla mahdollista, että osasyy puutteelliselle vanhentuneista paketeista päivittämiselle johtuu epäselvistä päivityksistä itse paketteihin. Chinthanet ym. (2021) päättelevät kuitenkin tutkimuksessaan myös, että vaikka paketin haa-

voittuvuus korjataan patch-tyyppisellä päivityksellä, vain noin 18 % paketeista, jotka riippuvat tästä, tuo vastaavan päivityksen omalle paketilleen. Tämä osoittaa, että riippuvussuhteiden välillä tapahtuvien haavoittuvuuksien korjausten välillä on merkittävästi viivettä.

5 Triviaalit paketit

Chen, Abdalkareem ja Mujahid (2021) määrittelevät triviaaleiksi paketeiksi (engl. trivial packages) sellaiset paketit, joissa on alle 35 riviä koodia, ja joiden monimutkaisuus on McCaben (1976) menetelmällä laskettuna alle 10. Käytännössä triviaaleilla paketeilla viitataan paketteihin, joiden koodi olisi suhteellisen nopeaa ja vaivatonta kirjoittaa itse, jolloin tarvetta riippuvuussuhteelle ei olisi. Tällaisia paketteja on Chen, Abdalkareem ja Mujahid (2021) mukaan noin 15 % npm-pakettirekisterin paketeista. Abdalkareem ym. (2017) määrittivät vastaavasti, että niistä tuhannessa paketista, joista muut paketit ovat eniten riippuvaisia, 113 ovat triviaaleja.

Triviaalien pakettien laaja käyttö voi myös johtaa haavoittuvuuksille altistumisen riippuvuussuhteiden tarpeettoman suuren määrän vuoksi (Chen, Abdalkareem ja Mujahid 2021).

Chen, Abdalkareem ja Mujahid (2021) tekemän kyselyn mukaan ohjelmoijat haluavat käyttää triviaaleja paketteja niiden uudelleenkäytettävyyden, testauksen ja dokumentaation helppouden ja yksinkertaiden vuoksi. Myös optimointi ja yleinen halu auttaa muita ohjelmoijia yhteisössä olivat syitä, joiden perusteella ohjelmoijat julkaisevat triviaaleja paketteja.

Triviaalien pakettien käytöstä voi muutenkin olla haittaa. Suurimmiksi ongelmiksi Chen, Abdalkareem ja Mujahid (2021) tekemän kyselyn mukaan nousevat useiden pakettien ylläpito sekä suuresta määrästä riippuvuussuhteita johtuvat ongelmat, eli ns. “dependency hell”. Myös vaikeudet löytää sopiva paketti ja useat lähes identtiset paketit ovat nousseet ongelmiksi. Jälkimmäinen voi johtaa myös projektin tarpeettomaan suurenemiseen, jos kaksi eri pakettia ovat riippuvaisia kahdesta eri paketista, jotka ovat kuitenkin käytännössä identtisiä. (Chen, Abdalkareem ja Mujahid 2021)

Valtaosa paketeista – noin 75 % – eivät ole minkään muun paketin riippuvuutena, ja vain noin 5 % paketeista on yli kuuden muun paketin riippuvuutena (Wittern, Suter ja Rajagopalan 2016). Tämä viittaa siihen, että tietyt paketit ovat hyvin laajassa käytössä muiden pakettien riippuvuussuhteina. Kuten edellisessä luvussa todettiin, jo yksittäinen suosittu paketti, jossa on haavoittuvuus, voi johtaa siihen, että haavoittuvuus vaikuttaakiin jopa miljooniin projekteihin. Kuten aiemmin mainitusta Decan, Mens ja Constantinou (2018) kävi ilmi, pie-

nikin määrä haavoittuneita paketteja voi vaikuttaa suuren määrään muita paketteja. Lisäämällä projektiin suuren määrän triviaaleja paketteja kehittäjä lisää riippuvuussuhteiden määrää ja siten altistaa oman työnsä todennäköisemmin haavoittuvuuksille (Chen, Abdalkareem ja Mujahid 2021).

Myös pakettien poistuminen voi aiheuttaa ongelmia. Vuonna 2016 tapahtunut `left-pad`-paketin poisto npm-rekisteristä johti siihen, että suuri osa paketeista ei enää toiminut, sillä ne riippuivat tästä erittäin suositusta paketista (Zimmermann ym. 2019). Tämä herätti keskustelua triviaalien pakettien käytön järkevyydestä (Chowdhury ym. 2022), ja johti myös väliaikaiseen pakettien välisten riippuvuussuhteiden määrän kasvun hidastumiseen (Zimmermann ym. 2019).

6 Riskien ja ongelmien ennaltaehkäisy

6.1 Paketinhallintajärjestelmän ominaisuudet

npm-paketinhallintajärjestelmä tarjoaa suoraan tapoja havaita ongelmia ja ilmoittaa niistä käyttäjilleen. Esimerkiksi npm-komentorivityökalun `npm audit` -komennolla voi milloin tahansa tarkistaa, onko missään projektin riippuvuussuhteessa haavoittuvuuksia. Komento kertoo haavoittuneiden pakettien nimet, haavoittuvuuden tyypin ja sen vakavuusasteen. Myös paketit asentava `npm install` -komento varoittaa samalla tavalla käyttäjää asennettujen pakettien mahdollisista haavoittuvuuksista. (npm 2021b)

npm ilmoittaa myös käyttäjälle, mikäli jokin asennetuista paketeista on vanhennettu. Komennolla `npm outdated` voi myös saada listan kaikista riippuvuussuhteista, jotka käyttävät vanhennettua versiota. (npm 2021c)

npm-paketinhallintajärjestelmä tarjoaa siis suoraan keinon ilmoittaa haavoittuvuuksista ja vanhentuneista paketeista, jos sellaisia löytyy. Käyttäjät voivat kuitenkin edelleen käyttää myös haavoittuneita paketteja, joten on loppujen lopuksi käyttäjän omalla vastuulla varmistaa, että haavoittuneet paketit joko otetaan väliaikaisesti pois käytöstä, tai että ne päivitetään uuteen haavoittumattomaan versioon, jos sellainen on.

6.2 Haavoittuvuuksien estäminen ja tunnistaminen

GitHub tarjoaa oman järjestelmän, jotka luotaa pakettien lähdekoodia ja pyrkii löytämään mahdolliset haavoittuvuudet (GitHub 2021). Tällaiset järjestelmät ovat Scalco ym. (2022) mukaan tarpeellisia, sillä manuaalinen lähdekoodin läpikäynti on hidas ja työläs prosessi. Lisäksi Scalco ym. (2022) argumentoivat, että on yhtä lailla tärkeää estää viime hetkellä tapahtuvat injektiohyökkäykset, jossa hyökkääjät lisäävät haitallista koodia esimerkiksi siinä vaiheessa, kun automatisoitu järjestelmä rakentaa lähdekoodista käyttövalmiin paketin. npm tarjoaa tämän ehkäisemiseksi keinoja allekirjoittaa paketit niin, etteivät ulkoiset hyökkääjät pääse muuttamaan niiden sisältöä (npm 2021a).

6.3 Parhaat käytänteet

Edellä mainittujen järjestelmien säännöllisen käytön lisäksi on tärkeää, että pakettien lähdekoodin kehittämiseen osallistuvat henkilöt ja erityisesti pakettien ylläpitäjät varmistavat omat käyttäjätietonsa. Kuten (Zahan ym. 2022) tutkimus osoitti, huolimattomuus käyttäjätietojen kanssa, ylityöllistäminen ja vanhentuneiden verkkotunnusten käyttö altistivat paketteja tietoturvariskeille.

Sen lisäksi, että pakettien ylläpitäjien on syytä varmistua heidän ylläpitämiensä pakettien laadusta, on järkevää myös harkita, mitä triviaaleja paketteja käyttää, sillä suuri riippuvuussuhteiden lukumäärä altistaa todennäköisemmin haavoittuvuuksille, mutta myös muille ongelmille, kuten projektin koon kasvuun.

7 Pohdinta

Alun perin tämän tutkimuksen aiheen oli tarkoitus olla huomattavasti laajempi, sisältäen haavoittuvuuksiin liittyvien riskien lisäksi myös pohdintaa paketinhallintajärjestelmien käytön tehokkuudesta ja tilankäytöstä levyllä. Kävi kuitenkin ilmi, ettei näistä aiheista ole juurikaan tehty tieteellistä tutkimusta. Joitain npm-paketinhallintajärjestelmää ja siitä johdettuja uudempia versioita vertailevia tutkimuksia löytyi, mutta oli hyvin selvää, että haavoittuvuuksiin liittyvää tutkimustietoa oli huomattavasti enemmän.

On toisaalta hyvin ymmärrettävää, että haavoittuvudet ovat priorisoitu aihe paketinhallintajärjestelmien tutkimuksessa. Vaikka esimerkiksi järjestelmän tehokkuus voi olla vaikuttava asia suurella skaalalla, ovat haavoittuvuudet selvästi vakavampi aihe, josta tehty tutkimus voi auttaa ihmisiä suojaamaan itsensä tai omat projektinsa paremmin tietoturvariskeiltä. Tästä syystä päädyin myös itse keskittymään tässä kandidaatintutkielmassa lähinnä paketinhallintajärjestelmän haavoittuvuuksiin, ja pysymään vain npm-paketinhallintajärjestelmässä sen sijaan, että etsisin tutkimustietoa myös muista paketinhallintajärjestelmistä.

Osa määrällisistä tutkimuksista, joihin tässä kandidaatintutkielmassa viitataan, ovat jo melko ikääntyneitä. Vaikka yksi päälähteistäni, Decan, Mens ja Constantinou (2018), on vain noin viisi vuotta vanha, on tämä npm:n kaltaisen ekosysteemin elinkaaressa pitkä aika. Esimerkiksi npm-pakettirekisterissä olevien pakettien määrä on lähes viisinkertainen siihen määrään, mitä kyseisessä tutkimuksessa vuonna 2018 tutkittiin. Ei ole myöskään täysin selvää, millä tavoin npm:n oma tietoturva tai npm-paketteja ylläpitävien tai lataavien kehittäjien käytänteet ovat kehittyneet vuosien saatossa. npm-paketinhallintajärjestelmän haavoittuvuuksista ja turvallisuudesta voisi siis olla syytä tehdä nykypäivänä lisää tutkimusta, sillä sen jatkuvasti kasvava suosio voisi altistaa suuren määrän ohjelmistokehittäjiä ja heidän kehittämiensä web-sovellusten käyttäjiä tietoturvariskeille.

Kuitenkin jo nykyisen tutkimustiedon perusteella pystyttiin havaitsemaan riskien lähteitä ja syitä niiden leviämiselle, ja niiden perusteella määrittelemään käytänteitä, joiden avulla näitä riskejä voidaan ennaltaehkäistä ja välttää.

8 Yhteenveto

Tutkielmassa käsiteltiin laajassa käytössä olevan npm-paketinhallintajärjestelmän toimintaa ja siihen liittyviä ongelmia ja riskejä. Pääasialliseksi riskiksi npm-paketinhallintajärjestelmän käytössä nousi pakettien haavoittuvuudet, jotka voivat lukuisien riippuvuussuhteiden myötä levitä hyvinkin moneen pakettiin.

Vaikka vain pieni osa npm-rekisterissä olevista paketeista ovat haavoittuneita, paketin laaja käyttö muiden pakettien riippuvuutena johtaa siihen, että haavoittuvuus esiintyy myös näissä paketista riippuvaisissa paketeissa. Yleisiä syitä siihen, että jokin paketti tai projekti päätyy olemaan riippuvainen haavoittuneesta paketista, ovat vanhentuneiden pakettien käyttö ja liiallinen triviaalien pakettien käyttö.

Tyypillisesti vanhentunut versio paketista jää paketin riippuvuussuhteeksi, jos semanttisen versioinnin avulla ollaan pyritty välttämään mahdollisesti rikkovat muutokset, joita paketin päivittyessä voisi ilmetä. Riippuvuussuhde vanhentuneeseen versioon voi ilmetä myös epäselvien uusien versioiden takia, tai jos paketti riippuu paketista, joka ei ole itse päivittänyt omaa riippuvuuttaan haavoittuneeseen pakettiin.

Liiallisessa triviaalien pakettien käytössä taas todennäköisyys haavoittuvaisen riippuvuussuhteen sisällyttämiseen projektiin kasvaa. Vaikka triviaalien pakettien käyttö voikin helpottaa ohjelmistokehitystä, niiden yleinen käyttö useiden eri pakettien välillä voi johtaa siihen, että yhdenkin triviaalin paketin altistuminen haavoittumiselle saa suuren määrän npm-rekisterissä olevien pakettien alistumaan samalle tietoturvaongelmalle.

Päällimmäisenä ongelmana voidaan siis havaita lukuisien riippuvuussuhteiden tuomat ongelmat. npm-paketinhallintajärjestelmää käyttävä ohjelmistokehittäjä tai yritys voivat ennaltaehkäistä näitä ongelmia pysymällä ajan tasalla käyttämiensä pakettien versioista, ja mahdollisesti välttämällä triviaaleja paketteja.

Merkittävä tietoturvariskejä npm-paketinhallintajärjestelmän kanssa ovat kuitenkin myös paketin ylläpitäjien käyttäjätietojen päätyminen hyökkääjien käsiin ja hyökkääjien tekemät injektiohyökkäykset paketteihin. npm ja GitHub tarjoavat monia keinoja ennaltaehkäistä myös

tällaisia tietoturvariskejä.

Tutkielman perusteella voidaan nostaa myös esiin npm-paketinhallintajärjestelmän käytön filosofisia kysymyksiä. Esimerkiksi triviaalien pakettien käyttö ei ole objektiivisesti huono asia, vaan niiden avulla voidaan myös parantaa ohjelmistokehityskokemusta. Myös semanttinen versiointi voi tuoda sekä riskejä että hyötyjä. Tahallisestikin vanhaan versioon jääminen voi mahdollisesti olla tietoturvariski, mutta uuteen versioon päivittäminen voisi tarkoittaa valtavaa määrää työtä koodin uudelleenkirjoituksen parissa.

Tutkielma tuo esille npm-paketinhallintajärjestelmään liittyvät riskit ja käy läpi myös keinoja välttää kyseisiä riskejä. Tutkielma tuo myös esille kysymyksiä siitä, minkälaiset toimintaperiaatteet olisivat parhaita npm-paketinhallintajärjestelmän kanssa työskentelyssä. Etenkin npm-paketinhallintajärjestelmän suuren suosion myötä on tärkeää, että sekä paketinhallintajärjestelmän omat työkalut että sen käyttäjien toimet seuraavat parhaita käytänteitä, jotta haavoittuvuuksien syntyä ja laajaa leviämistä voidaan välttää.

Lähteet

Abdalkareem, R., O. Nourry, S. Wehaibi, S. Mujahid ja E. Shihab. 2017. “Why do developers use trivial packages? an empirical case study on npm”. *Proceedings of the 2017 11th Joint Meeting on Foundations of Software Engineering,* 385–395.

Chen, X., R. Abdalkareem ja S. et al. Mujahid. 2021. “Helping or not helping? Why and how trivial packages impact the npm ecosystem”. *Empirical Software Engineering* 25:1168–1204.

Chinthanet, B., R. Kula, S. McInthsh, T. Ishio, A. Ihara ja K. Matsumoto. 2021. “Lags in the release, adoption, and propagation of npm vulnerability fixes”. *Empirical Software Engineering (2021)* 26.

Chowdhury, M. A. R., R. Abdalkareem, E. Shihab ja B. Adams. 2022. “On the Untriviality of Trivial Packages: An Empirical Study of npm JavaScript Packages”. *IEEE transactions on software engineering* 48 (8): 2695–2708.

Cogo, F., G. Oliva ja A. Hassan. 2022. “Deprecation of Packages and Releases in Software Ecosystems: A Case Study on NPM”. *IEEE Transactions on Software Engineering* 48:2208–2223.

Decan, A., ja T. Mens. 2021. “What Do Package Dependencies Tell Us About Semantic Versioning?” *IEEE Transactions on Software Engineering* 47:1226–1240.

Decan, A., T. Mens ja E. Constantinou. 2018. “On the impact of security vulnerabilities in the npm package dependency network”. *ACM/IEEE 15th International Conference on Mining Software Repositories,* 181–191.

GitHub. 2021. *About code scanning.* WWW-sivu, https://docs.github.com/en/code-security/code-scanning/automatically-scanning-your-code-for-vulnerabilities-and-errors/about-code-scanning, viitattu 11.4.2023.

———. 2023. *The top programming languages | The State of Octoverse.* WWW-sivu, https://octoverse.github.com/2022/top-programming-languages, viitattu 13.3.2023.

libraries.io. 2023. *npm.* WWW-sivu, https://libraries.io/npm, viitattu 13.3.2023.

Liu, C., S. Chen ja L. et. al. Fan. 2022. “Demystifying the Vulnerability Propagation and Its Evolution via Dependency Trees in the NPM Ecosystem”. *2022 IEEE/ACM 44th International Conference on Software Engineering (ICSE),* 672–684.

MDN Web Docs. 2023a. *JavaScript.* WWW-sivu, https://developer.mozilla.org/en-US/docs/Web/JavaScript, viitattu 22.2.2023.

———. 2023b. *Package management basics.* WWW-sivu, https://developer.mozilla.org/en-US/docs/Learn/Tools_and_testing/Understanding_client-side_tools/Package_management, viitattu 22.2.2023.

npm. 2021a. *About ECDSA registry signatures.* WWW-sivu, https://docs.npmjs.com/about-registry-signatures, viitattu 11.4.2023.

———. 2021b. *Auditing package dependencies for security vulnerabilities.* WWW-sivu, https://docs.npmjs.com/auditing-package-dependencies-for-security-vulnerabilities, viitattu 11.4.2023.

———. 2021c. *npm-outdated.* WWW-sivu, https://docs.npmjs.com/cli/v7/commands/npm-outdated, viitattu 11.4.2023.

———. 2023a. *About semantic versioning.* WWW-sivu, https://docs.npmjs.com/about-semantic-versioning, viitattu 8.4.2023.

———. 2023b. *eslint.* WWW-sivu, https://www.npmjs.com/package/eslint?activeTab=dependencies, viitattu 14.3.2023.

———. 2023c. *npm-rekisterin etusivu.* WWW-sivu, https://www.npmjs.com/, viitattu 11.4.2023.

———. 2023d. *package.json.* WWW-sivu, https://docs.npmjs.com/cli/v9/configuring-npm/package-json, viitattu 13.3.2023.

Paul Brown. 2017. *State of the Union: npm.* Blogi, https://www.linux.com/news/state-union-npm/, viitattu 13.3.2023.

Scalco, S., R. Paramitha, D. Vu ja F. Massacci. 2022. “On the feasibility of detecting injections in malicious npm packages”. *Proceedings of the 17th International Conference on Availability, Reliability and Security,* 1–8.

semver. 2023. *Semanttisen versionnin määritelmä.* WWW-sivu, https://semver.org/, viitattu 8.4.2023.

The Server Side. 2018. *The secret history behind the success of npm and Node.* Blogi, https://www.theserverside.com/blog/Coffee-Talk-Java-News-Stories-and-Opinions/The-secret-history-behind-the-success-of-npm-and-Node, viitattu 13.3.2023.

W3Techs. 2023. *Usage statistics of JavaScript as client-side programming language on websites.* WWW-sivu, https://w3techs.com/technologies/details/cp-javascript, viitattu 22.2.2023.

Wittern, E., P. Suter ja S. Rajagopalan. 2016. “A Look at the Dynamics of the JavaScript Package Ecosystem”. *2016 IEEE/ACM 13th Working Conference on Mining Software Repositories,* 351–361.

Zahan, N., T. Zimmermann, P. Godefroid, B. Murphy, C. Maddila ja L. Williams. 2022. “What are weak links in the npm supply chain?” *ICSE-SEIP ’22: Proceedings of the 44th International Conference on Software Engineering: Software Engineering in Practice,* 331–340.

Zimmermann, M., C.A. Staicu, C. Tenny ja M. Pradel. 2019. “Small World with High Risks: A Study of Security Threats in the npm Ecosystem”. *USENIX security symposium* 17.

Liitteet

A Riippuvuussuhteiden esimerkki

`npm list –all` -komennolla saatu lista 2. luvussa esitetyn `package.json`-tiedoston mukaisen npm-paketinhallintajärjestelmää hyödyntävän Node-projektin riippuvuussuhteista. `deduped`-sanalla merkatut paketit ovat jo jonkin muun paketin riippuvuutena. `[...]`-merkein merkatut kohdat on leikattu pois luettavuuden parantamiseksi.

```
chalk@5.2.0
eslint@8.36.0
  @eslint-community/eslint-utils@4.2.0
   eslint-visitor-keys@3.3.0 deduped
   eslint@8.36.0 deduped
  @eslint-community/regexpp@4.4.0
  @eslint/eslintrc@2.0.1
   ajv@6.12.6 deduped
   debug@4.3.4 deduped
   [ ... ]
  [ ... ]
  file-entry-cache@6.0.1
   flat-cache@3.0.4
     rimraf@3.0.2
       glob@7.2.3
         fs.realpath@1.0.0
         inflight@1.0.6
          once@1.4.0 deduped
          wrappy@1.0.2
         [ ... ]
  [ ... ]
```