Henri Pigg

npm-pakettiekosysteemin haavoittuvuudet

Tietotekniikan kandidaatintutkielma

2. toukokuuta 2023

Jyväskylän yliopisto

Informaatioteknologian tiedekunta

Tekijä: Henri Pigg

Yhteystiedot: henri.m.pigg@student.jyu.fi

Ohjaaja: Antti-Jussi Lakanen

Työn nimi: npm-pakettiekosysteemin haavoittuvuudet

Title in English: Vulnerabilities in the npm package-ecosystem

Työ: Kandidaatintutkielma

Opintosuunta: Kaikki opintosuunnat

Sivumäärä: 19+0

Tiivistelmä: Tässä tutkielmassa käsitellään npm-paketinhallintajärjestelmälle tyypillisiä haavoittuvuuksia, uhkia ja muita pakettiekosysteemeille tyypillisiä ongelmia.Lisäksi tutkielma pyrkii tarjoamaan erilaisia kehitettyjä ratkaisuja ongelmien ehkäisemiseksi.

Tutkimusmenetelmänä on kirjallisuuskatsaus aiheita käsitteleviin tutkimuksiin ja artikkeleihin. Tutkimustulokset selventävät parhaita käytänteitä, joita tulisi hyödyntää ongelmien välttämiseksi.

Avainsanat: npm, haavoittuvuus, ohjelmistoekosysteemi, paketinhallinta

Abstract: This thesis is about vulnerabilities, threats and other common problems in the npm- package ecosystem. Additionally the thesis aims to offer some solutions to possibly prevent these problems handled in the text.

The study method for the thesis is a literary review. The results offer some best practices that should be utilized to avoid the problems presented in the thesis.

Keywords: npm, vulnerability, software-ecosystem, package management

Termiluettelo

npm	JavaScript-ohjelmointikielen yhteydessä käytettävä ilmainen avoimen lähdekoodin pakettienhallintajärjestelmä.
Hyökkäyspinta	Hyökkääjälle avoinna oleva resurssi, jonka kautta hyökkäys voidaan suorittaa.
Hyökkäysvektori	Tapa tai reitti, jolla järjestelmään kohdistuva hyökkäys suoritetaan.

Kuviot

Sisällys

1 Johdanto

Suuri osa nykypäivän ohjelmistokehityksessä käytettävästä koodista tulee eri kirjastoista ja ohjelmistokehyksistä (Decan, Mens ja Constantinou 2018). Yksi suosituimmista pakettiekosysteemeistä on JavaScriptin yhteydessä käytettävä Node Package Manager (npm), joka tarjoaa käyttäjille ilmaisia avoimen lähdekoodin paketteja, jotka sisältävät uudelleenkäytettävää koodia (Zimmermann ym. 2019). Tällä hetkellä npm tarjoaa yli 1,7 miljoonaa ilmaista avoimen lähdekoodin pakettia (Sejfia ja Schäfer 2022). Ei ole kuitenkaan yllättävää, että näin laaja pakettiriippuvuuksiin perustuva ekosysteemi tuo mukanaan myös paljon erilaisia haavoittuvuuksia. Noin neljänneksessä kaikista paketeista on tunnettuja haavoittuvuuksia (Decan, Mens ja Constantinou 2018) ja (Zimmermann ym. 2019).

Yksi pakettiekosysteemien yhteydessä ilmenevä haavoittuvuus on jakeluketjuhyökkäysn, jonka tarkoitus on puskea haitallista koodia ohjelmistoon tai kirjastoon. Haitallisen koodin syöttämisen avulla hyökkääjä voi muuttaa kohteen toimintaa, vaikka ulkoisesti hyökkäyskohteessa ei välttämättä ole havaittavissa muutoksia (Kaplan ja Qian 2021). Vuonna 2021 npm-ekosysteemiin kohdistuvien jakeluketjuhyökkäysten määrä kasvoi 650 % (Zahan ym. 2022).

Muita hyökkäystyyppejä ovat erilaiset kirjoitusvirheharhautukset (engl. *typosquatting*, *combosquatting*). Näissä hyödynnetään kirjoitusvirheitä nimeämällä haitallinen paketti mahdollisimman samalla tavalla, kuin jokin suosittu paketti. Tällöin kehittäjä saattaa pakettia asentaessaan tehdä kirjoitusvirheen ja asentaa vahingossa haitallisen paketin (Kaplan ja Qian 2021). Näiden ongelmien lisäksi käsitellään myös pakettiriippuvuuksia ja ylläpidon puutteesta aiheutuvia haavoittuvuuksia.

Tutkielma käsittelee edellä mainittuja haavoittuvuuksia, sekä muita npm-ekosysteemille ominaisia ongelmia. Luku 2 käsittelee npm-paketinhallintaa yleisellä tasolla. Luvussa 3 tarkastellaan erilaisia haavoittuvuuksia ja riskejä tarkemmin. Luku 4 puolestaan käsittelee mahdollisia ratkaisuja ja parhaita käytäntöjä näihin ongelmiin.

Aihetta pitää tutkia, sillä npm-paketinahallintajärjestelmä on haavoittuvassa tilassa, kuten tilastot ja tutkimukset todistavat. On siis tärkeää selventää ja ymmärtää ongelmakohtia, jotta tulevaisuudessa niitä voidaan ehkäistä paremmin. Npm-ekosysteemiin kohdistuvaa tutki-

musta voidaan myös hyödyntää muiden paketinhallintajärjestelmien kohdalla, jotka saattavat kärsiä samankaltaisista ongelmista.

2 npm-paketinhallinta

2.1 Ohjelmistoekosysteemit

Koodin uudelleenkäyttö on yleisesti hyväksytty käytänne ohjelmistokehityksessä (Gkortzis, Feitosa ja Spinellis 2019). Ohjelmistoekosysteemien, tarkoitus on helpottaa ja nopeuttaa ohjelmistokehitystä tarjoamalla avoimen lähdekoodin ratkaisuja. Ohjelmistoekosysteemi on kokoelma itsenäisiä ohjelmistopaketteja, jotka on koottu yhteiselle tekniselle alustalle (Mujahid, Abdalkareem ja Shihab 2023). JavaScript-ohjelmointikielen yhteydessä käytettävä Node Package Manager (npm) lukeutuu näihin ekosysteemeihin. Muita samankaltaisia ohjelmistoekosysteemejä ovat esimerkiksi Python-ohjelmointikielen yhteydessä käytettävä PyPi (Taylor ym. 2020). Npm koostuu tietokannasta, josta voidaan hakea tehtävään sopivia paketteja ja paketinhallitsijasta, jonka avulla paketti, sekä kaikki sen riippuvuudet, voidaan asentaa kehitysympäristöön (Zimmermann ym. 2019). Npm-paketteja hallinnoidaan projektikohtaisesti JSON-tiedostosta, josta voidaan nähdä esimerkiksi tiettyyn projektiin liittyvät pakettien versionumerot, sekä riippuvuussuhteet (Kaplan ja Qian 2021).

2.2 npm-paketti

Npm-paketti sisältää JavaScript-koodia, jota kehittäjä voi hyödyntää projektissaan lataamalla paketin npm-tietokannasta. Yksittäinen paketti voi tarjota esimerkiksi testausvälineitä tai valmiita funktioita jonkin ongelman käsittelyyn. Paketit ovat siis ohjelmistojen riippuvuuksia ja niitä hyödynnetään kehityksessä. Ne eivät yleensä ole siis itsenäisiä ohjelmia, vaan työkalua ohjelmistojen rakentamiseen (Alfadel ym. 2020).

2.3 npm-pakettien hyödyntäminen ohjelmistokehityksessä

Koodin jakamisen ja uudelleenkäytön helppous on yksi keskeisistä syistä npm-ekosysteemin suureen suosioon (Zimmermann ym. 2019). Uudelleenkäytön tuomia hyötyjä ovat esimerkiksi tehostettu tuottavuus, sekä parempi ohjelmiston laatu (Abdalkareem ym. 2017). Pakettipohjainen uudelleenkäyetty koodi voi siis jossain määrin parantaa ohjelman laatua, sekä

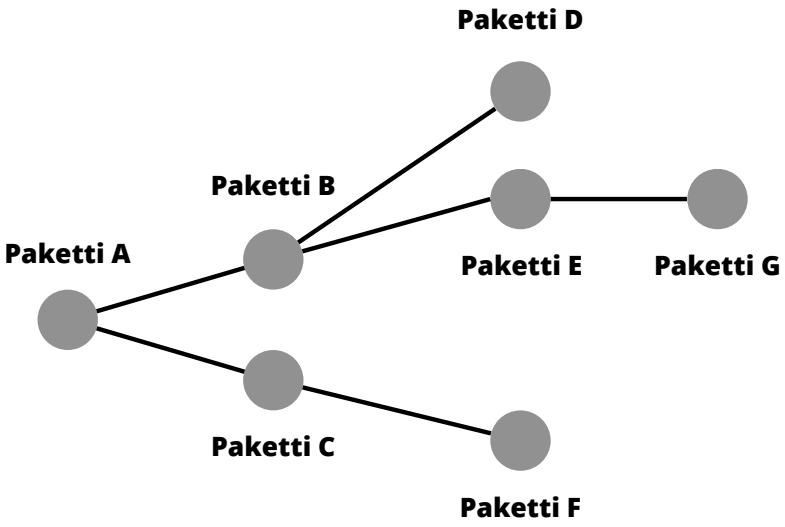
turvallisuutta pitkäaikaisen testauksen ja kypsyyden johdosta (Gkortzis, Feitosa ja Spinellis 2019). Avoin lähdekoodi on usein myös hyvin joustava ratkaisu muokattavuuden ansiosta (Mockus 2007).

Toisaalta pakettiekosysteemi voi kutienkin tuoda mukanaan myös erilaisia haavoittuvuuksia, sekä näiden lisäksi muita negatiivisia vaikutuksia, kuten projektin kulujen kasvaminen (Abdalkareem ym. 2017). Näiden lisäksi myös tarkoitukselliset suorat hyökkäykset, sekä esimerkiksi ylläpidon puutteesta tai vahingoista johtuvat seuraukset ovat tavallisia. Suuri hyökkäyspinta ja epäturvalliset ohjelmointikäytänteet ovat yksi keskeisistä syistä avoimen lähdekoodin pakettien haavoittuvuuksiin (Gkortzis, Feitosa ja Spinellis 2019). Pakettien hyödyntäminen voi siis tuottaa ongelmia monessa eri kohtaa ohjelmiston elinkaarta. Yksi pieni pakettipäivitys tai inhimillinen virhe voi hyvinkin olla syy suurien tietovuotojen takana.

3 Pakettien haasteet ja riskit

3.1 Pakettiriippuvuudet

Pakettiriippuvuus tarkoittaa yksinkertaisimmillaan sitä, että paketti A riippuu jostakin toisesta paketista B. Paketti A siis hyödyntää jollain tavalla paketin B funktioita, komponentteja tai muita koodin osia (Kaplan ja Qian 2021). Tämän lisäksi esimerkiksi paketin B riippuessa paketista C myös paketti A riippuu Paketista C. Tätä kutsutaan transitiivisuudeksi. Pakettien transitiivisen luonteen vuoksi suosittuja paketteja ladataan hyvin usein, sillä ne ovat usean muun paketin riippuvuuksia (Sejfia ja Schäfer 2022). Tämänkaltaisia riippuvuusketjuja voidaan kuvata riippuvuuspuiden avulla 1.



Kuvio 1. Mahdollinen npm-paketin riippuvuuketju. Tässä esimerkissä A riippuu paketista B ja C. Koska B riippuu paketeista E, D ja G, niin paketti A on myös epäsuorasti riippuvainen niistä. Sama pätee paketin C riippuvuuksien kohdalla.

Riippuvuussuhteet voivat siis hyvinkin helposti laajentua useihin eri paketteihin. Projektin suurentuessa ja hyödynnettyjen npm-pakettien määrän kasvaessa riippuvuussuhteet monimutkaistuvat. Tämänkaltainen riippuvuuksien laajentuminen puolestaan kasvattaa hyökkäyspintaa (Kaplan ja Qian 2021). Tätä vahvistaa Zimmermann ym. (2019) esittämät tilastot, joissa todetaan, että keskiverto npm-paketti sisältää 79 suoraa tai epäsuoraa riippuvuutta, sekä 39 ylläpitäjää. Riippuuvuussuhteiden määrä kasvaa siis hyvin helposti suureksi. Toisaalta suuri määrä riippuvuuksia ei myöskään korreloi suoraan suureen määrän haavoittuvuuksia, mutta hyökkäykselle altistuminen on todennäköisempää jos mahdollisia haavoittuvuuskohtia on useita.

Eräs toinen syy riippuvuusketjujen kasvuun ovat niin kutsutut mikropaketit. Ne ovat paketteja, jotka koostuvat vain muutamasta rivistä koodia, mutta ovat keskeinen riippuvuus monelle suuremmalle paketille (Zimmermann ym. 2019). Kula ym. (2017) mukaan mikropakettien määrän kasvu riippuvuusketjuissa johtaa heikentyneeseen ekosysteemiin, jossa yksi kriittisen paketin muutos voi johtaa koko riippuvuusketjun hajoamiseen. Esimerkki tällaisesta tapahtumasta on vuoden 2016 maaliskuussa tapahtunut *left-pad* -nimisen paketin poistuminen, joka johti lukemattomien verkkosovellusten hajoamiseen. (Kula ym. 2017). Paketin tehtävänä oli vain lisätä vasemman puolen pehmustusta (engl. *padding*) html-sivuun (Kula ym. 2017). Ongelma olisi siis selkeästi ollut helposti vältettävissä, mutta mikropaketin tarjoama valmis ratkaisu arkipäiväiseen tehtävään oli mieluisampaa kehittäjille. Mikropakettien käyttöön voi olla muitakin syitä , helppouden lisäksi esimerkiksi ajan ja budjetin rajoitteet voivat johtaa niiden käyttöön. Kuitenkin niiden kohdalla tulisi miettiä mahdollisia haavoittuvuuksia, joita triviaalien pakettien lisäys ketjuun mahdollistaa.

3.2 Hyökkäyspinta

Hyökkäyspinnalla tarkoitetaan hyökkääjälle avoinna olevaa resurssia, jota kautta hyökkäys voidaan suorittaa (Zhang ym. 2015). Vastaavasti hyökkäysvektorilla tarkoitetaan tapaa tai reittiä, jolla hyökkäys suoritetaan (Zahan ym. 2022). Pakettiekosysteemin mahdollistama laaja hyökkäyspinta tarjoaa monta eri tapaa hyökätä yksittäiseen haavoittuvuuteen riippuvuussuhteista koostuvan ekosysteemin sisällä, sekä myös erillisten hyökkäystapojen yhdistelmiä (Zimmermann ym. 2019) ja (Zhang ym. 2015). Tämän tutkielman kontekstissa hyök-

käyspinta tarkoittaa siis paketteja, niiden riippuvuuksia, mahdollisia konfiguraatiovirheitä sekä käyttäjien paketteihin kohdistamia toimia.

Pakettien luoma laaja-alainen riippuvuusketju vaikuttaa huolestuttavan hyvältä kohteelta potentiaalisille hyökkääjille. Toisaalta, vaikka yksi ketjun paketti saastutettaisiin, koko ketju ei välttämättä automaattisesti hajoa. Esimerkiksi, jos paketin tiettyä funktiota tai toimintoa ei hyödynnetä, niin hyökkääjä ei pääse käsiksi mihinkään järjestelmän kriittiiseen tietoon tai toimintoon. Tämä ei toki päde kaikkiin hyökkäystyyppeihin.

3.3 Vanhentuneet ja hylätyt paketit

Jokaisella paketilla on yksi tai useampi kehittäjä, jotka ovat vastuussa paketin päivittämisestä (Zimmermann ym. 2019). Tästä johtuen paketti saattaa ajan kuluessa tulla hylätyksi esimkeriksi ylläpitoa vaativien resurssien puutten vuoksi. Vanhentunut paketti voi aiheuttaa ongelmia esimerkiksi siitä riippuvien pakettien kohdalla, jolloin jokin laajempi npm-pakettikokonaisuus mahdollisesti hajoaa.

Npm-paketteja voidaan hylätä syystä tai toisesta. Ei ole siis yllättävää, että vanhentuneet paketit, ja niistä aiheutuvat ongelmat, ovat niin yleisiä. Noin 3,2 % kaikista npm-paketeista on vanhentuneita, mutta noin puolet paketeista perivät transitiivisesti ainakin yhden vanhentuneen paketin. (Cogo, Oliva ja Hassan 2022). Järjestelmänä npm varoittaa vanhentuneista riippuvuuksista viestin avulla, jonka paketin ylläpitäjä voi lisätä vanhentuneeseen pakettiin. Kuitenkin tästä huolimatta paketteja jää päivittämättä (Cogo, Oliva ja Hassan 2022).

Vanhentuneen paketin käyttäminen ei välttämättä itsessään ole haavoittuvuus, etenkin jos hyödynnetään vain yhtä tiettyä paketin ominaisuutta. Turvallisuusvaatimukset ja hyökkäyskeinot kuitenkin muuttuvat ajan edetessä, joten pitkällä aikavälillä ei ole suositeltavaa käyttää vanhentunutta pakettia, sillä se saatta osoittautua haavoittuvuudeksi päivitysten puutteen takia. Varsinkin lukittujen riippuvuuksien kohdalla tämä saattaa koitua ongelmaksi. Tietyt projektit lukitsevat paketin version esimerkiksi vakauden takia, jolloin uusimat turvallisuuspäivitykset jäävät tekemättä.

3.4 Jakeluketjuhyökkäys

Jakeluketjuhyökkäysten tarkoitus on syöttää haitallista koodia johonkin ohjelmaan tai ohjelmakirjastoon (Kaplan ja Qian 2021). Hyökkäysten tarkoituksena on muuttaa jollakin tavalla kohdeohjelman toimintaa. Zimmermann ym. (2019) ja Kaplan ja Qian (2021) mukaan tämä voi esimerkiksi johtua siitä, että paketin ylläpitäjien kirjautumistietoja tai omistajuuksia on joutunut vääriin käsiin. Zahan ym. (2022) esittävät myös muita keinoja, jolla jakelukteju hyökkäys voidaan toteuttaa. Ylläpitäjän manipuloinnin (engl. *social engineering*) tai suoran kaappauksen lisäksi hyökkääjät voivat myös itse julkaista haitallisen paketin ekosysteemiin ja päästä tätä kautta käsiksi arkaluontoiseen tietoon jakeluketjussa (Zahan ym. 2022).

Myös pakettien hyödyntymät kolmannen osapuolen palvelut voivat joutua hyökkäyksen kohteeksi, joka puolestaan taas vaikuttaa jakeluketjuun (Zahan ym. 2022). Kolmannen osapuolen palveulla tarkoitetaan esimerkiksi jonkinlaista ohjelmointirajapintaa. Tämänkaltaisten paketitiekosysteemien kohdalla jokainen riippuvuusketjun saastunut solmukohta on suuri ongelma, sillä pakettien transitiiviset ominaisuudet mahdollistavat laajoja hyökkäyksiä (Kaplan ja Qian 2021).

3.5 Kirjoitusvirheharhautus

Toinen yleinen hyökkäystapa suoran paketin kaappauksen tai saastuttamisen lisäksi on kirjoutsvirheharhautusten (*typosquatting*, *combosquatting*) yrittäminen. Kuten luvussa 1 mainittiin hyökkäyksen tarkoituksena on saada kehittäjä asentamaan haitallinen paketti tukeutumalla paketin asentajan tekemään mahdolliseen kirjoitusvirheeseen (*typosquatting*). Esimerkkinä tästä on suositun *lodash* -paketin kohdalla tapahtunut harhautusyritys, jossa haitallinen harhautuspaketti oli nimetty nimellä *loadsh* (Kaplan ja Qian 2021). Tämän tapauksen kaltainen kirjoutsvirhe on hyvin huomaamaton ja tavallinen asia, josta voi kuitenkin olla paljon haitallisia seurauksia.

Toinen tapa tehdä kirjoutsvirheharhautuksia on moniosaisten nimien kohdalla, missä esimerkiksi sanojen järjestyksellä on väliä (*combosquatting*). Tässä tapauksessa tukeudutuaan siihen mahdollisuuteen, etttä kehittäjä unohtaa pakettia asentaessaan sanojen oikean järjestyksen ja asentaa sitä kautta vahingossa väärän paketin (Kaplan ja Qian 2021).

Molemmat tavat tukeutuvat siis inhimillisten virheiden varaan, mutta ovat vaikutukseltaan todistetusti yllättävän laaja-alaisia. Kaplan ja Qian (2021) toteavat, että eräässä tutkimuksessa jaettuja kirjoitusvirhepaketteja oli ladattu yli 45 000 kertaa useiden kuukausien ajalta. Ratkaisu tähän ongelmaan on kuitenkin teoriassa yksinkertainen, kehittäjien tulee kiinnittää enemmän huomiota oikeinkirjoitukseen pakettien asennuksen kohdalla. Tällaiset paketit eivät saisi edes päästä npm-ekosysteemiin sisälle, mutta kehittäjällä on mahdollisuus ehkäistä kirjoitusvirhepaketeista aiheutuvat ongelmat olemalla tarkempia asennusvaiheessa. Luonnollisesti virheitä tulee tapahtumaan, mutta kehittäjien tulisi ainakin olla tietoisia tällaisista havoittuvuuksista ja niiden seurauksista.

4 Ratkaisuja haavoittuvuuksiin

Suurien pakettiekosysteemien, kuten npm, kohdalla on selkeästi monia mahdollisia haavoittuvuuskohtia. Järjestelmän transitiivisuus ja riippuvuudet, sekä jossain määrin sen helppokäyttöisyys ja avoimuus, ovat myös osittain sen heikkouksia. Mahdollisia korjauksia edellä käsiteltyihin ongelmiin on esitetty monien tahojen toimesta. Zimmermann ym. (2019) ehdottavat muun muassa yleisen tietoisuuden lisäämistä paketteja koskevista haavoittuvuuksista kehittäjien keskuudessa. On selkeää, että suuri osa kehittäjistä ei ole tietoisia haavoittuvuuksista omassa koodissaan ja käytänteissään (Gkortzis, Feitosa ja Spinellis 2019). Myös pakettien ennakkotarkistusta npm-alustan toimesta samaan tapaan, kuten esimerkiksi mobiilikaupoissa voisi auttaa ehkäisemään haitallisten pakettien pääsyä alustalle (Zimmermann ym. 2019).

Ennakkotarkistus voisikin olla hyvä tapa ehkäistä haavoittuvan tai pahantahtoisen koodin pääsemistä ekosysteemiin. Manuaalinen koodin tarkistaminen vaatisi kuitenkin hyvin paljon resursseja. Jonkinlainen automaattinen ratkaisu jo olemassa olevan npm-audit -työkalun rinnalle voisi olla realistisempi vaihtoehto. Npm-audit vertailee kaikkia riippuvuusketjun paketteja tietokantaan, jossa on tunnettuja haavoittuvuuksia ja varoittaa kehittäjää, kun käytössä on haavoittuvainen versio (Zimmermann ym. 2019). Se on hyvä reaktiivinen työkalu, mutta proaktiivisuus olisi mahdollisesti tehokkaampi tapa kitkeä haavoittuvuuksia pois ekosysteemistä. Rajoittamalla haavoittuvan koodin pääsyä järjestelmään automaattisella tai manuaalisella tarkastuksela voitaisiin ongelmaa pienentää merkittävästi.

Myös pakettien ylläpitäjille tulisi mahdollisesti asettaa enemmän vastuuta tai tiettyjä kohtuullisia rajoitteita pakettien julkaisun suhteen (Kaplan ja Qian 2021).Toisaalta liialliset rajoitteet ovat haitaksi avoimeen lähdekoodiin pohjautuville ekosysteemeille. Rajoitteet saattavat esimerkiksi vähentää kehittäjien halua edistää paketteja vaadittavien resurssien puutteen takia. On siis tärkeää ekosysteemin elinkaaren kannalta, että järjestelmän turvallisuus ja helppokäyttöisyys, sekä ylläpidettävyys säilyvät jonkinlaisessa tasapainossa. Kaplan ja Qian (2021) esittävät myös monivaiheisen todennuksen (engl. Multi Factor Authentication) lisäämistä kriittisten tilien turvaamiseksi. Sen avulla voitaisiin vähentää paketteihin ja ylläpitotileihin kohdistuvia suoria kaappauksia.

Kehittäjällä ja muulla ohjelmistoprojektin osallisilla on myös tietty vastuu paketeista johtuvien ongelmien ehkäisemisessä. Tunnettuja haavoittuvuuksia sisältäviä paketteja, sekä vanhentuneita versioita tulisi välttää. Myös mahdolliset yksityiset paketit tulisi tarkastaa haavoittuvuuksien varalta, sekä pakettien ylläpitotilit voitaisiin varmistaa tässä luvussa aiemmin mainittujen menetelmien mukaan.

5 Yhteenveto

Kaiken edellä mainitun asian ja tilastojen pohjalta voidaan todeta, että npm-ekosysteemi on hyvin altis haavoittuvuuksille ja niiden määrää tulisi pyrkiä vähentämään ehdotettujen keinojen avulla. Vastuu näistä parannuksista ja turvallisuuden ylläpitämisestä on pakettien ylläpitäjillä ja kehittäjillä, mutta myös itse npm-organisaation tulisi puuttua ongelmaan ja asettaa tiettyjä kriteerejä pakettien julkaisemiseen ja ylläpitoon. Haavoittuvuudet ovat kuitenkin monimutkainen ongelma, jota ei pysty yksi sidosryhmä itsenäisesti ratkaisemaan.

Proaktiiviset ratkaisut kuten laaja koulutus ja mahdollisista ongelmista tiedottaminen, sekä tekniset menetelmät pakettien tarkastukseen ja tilien suojaukseen ovat hyvä ensiaskel hyökkäyksien vähentämiseksi. Osa mahdollisista haavoittuvuuksista on epätodennäköisempiä, kuin toiset. Minkään osa-alueen vaikutuksia ei tule kuitenkaan vähätellä sillä, Jos kyseessä on esimerkiksi kriittisen datan käsittelystä tai kalliista tietojärjestelmistä voi pienikin haavoittuvuus aiheuttaa suurta haittaa organisaatiolle tai yksilölle.

Lähteet

Abdalkareem, Rabe, Olivier Nourry, Sultan Wehaibi, Suhaib Mujahid ja Emad Shihab. 2017. "Why Do Developers Use Trivial Packages? An Empirical Case Study on Npm". Teoksessa *Proceedings of the 2017 11th Joint Meeting on Foundations of Software Engineering,* 385–395. ESEC/FSE 2017. Paderborn, Germany: Association for Computing Machinery. ISBN: 9781450351058. https://doi.org/10.1145/3106237.3106267. https://doi.org/10.1145/3106237.3106267.

Alfadel, Mahmoud, Diego Elias Costa, Mouafak Mokhallalati, Emad Shihab ja Bram Adams. 2020. *On the Threat of npm Vulnerable Dependencies in Node.js Applications.* https://doi.org/10.48550/ARXIV.2009.09019. https://arxiv.org/abs/2009.09019.

Cogo, Filipe R., Gustavo A. Oliva ja Ahmed E. Hassan. 2022. "Deprecation of Packages and Releases in Software Ecosystems: A Case Study on NPM". *IEEE Transactions on Software Engineering* 48 (7): 2208–2223. https://doi.org/10.1109/TSE.2021.3055123.

Decan, Alexandre, Tom Mens ja Eleni Constantinou. 2018. "On the Impact of Security Vulnerabilities in the Npm Package Dependency Network", 181–191. MSR '18. Gothenburg, Sweden: Association for Computing Machinery. ISBN: 9781450357166. https://doi.org/10.1145/3196398.3196401. https://doi.org/10.1145/3196398.3196401.

Gkortzis, Antonios, Daniel Feitosa ja Diomidis Spinellis. 2019. "A Double-Edged Sword? Software Reuse and Potential Security Vulnerabilities". Teoksessa *Reuse in the Big Data Era,* toimittanut Xin Peng, Apostolos Ampatzoglou ja Tanmay Bhowmik, 187–203. Cham: Springer International Publishing. ISBN: 978-3-030-22888-0.

Kaplan, Berkay, ja Jingyu Qian. 2021. "A Survey on Common Threats in npm and PyPi Registries". Teoksessa *Deployable Machine Learning for Security Defense,* toimittanut Gang Wang, Arridhana Ciptadi ja Ali Ahmadzadeh, 132–156. Cham: Springer International Publishing. ISBN: 978-3-030-87839-9.

Kula, Raula, Ali Ouni, Daniel German ja Katsuro Inoue. 2017. "On the Impact of Micro-Packages: An Empirical Study of the npm JavaScript Ecosystem" (syyskuu).

Mockus, Audris. 2007. “Large-Scale Code Reuse in Open Source Software”. Teoksessa *First International Workshop on Emerging Trends in FLOSS Research and Development (FLOSS’07: ICSE Workshops 2007),* 7–7. https://doi.org/10.1109/FLOSS.2007.10.

Mujahid, Suhaib, Rabe Abdalkareem ja Emad Shihab. 2023. “What are the characteristics of highly-selected packages? A case study on the npm ecosystem”. *Journal of Systems and Software* 198:111588. ISSN: 0164-1212. https://doi.org/https://doi.org/10.1016/j.jss.2022.111588. https://www.sciencedirect.com/science/article/pii/S0164121222002643.

Sejfia, Adriana, ja Max Schäfer. 2022. “Practical automated detection of malicious npm packages”. Teoksessa *Proceedings of the 44th International Conference on Software Engineering*. ACM, toukokuu. https://doi.org/10.1145/3510003.3510104. https://doi.org/10.1145%2F3510003.3510104.

Taylor, Matthew, Ruturaj K. Vaidya, Drew Davidson, Lorenzo De Carli ja Vaibhav Rastogi. 2020. *SpellBound: Defending Against Package Typosquatting*. arXiv: 2003.03471 [cs.SE].

Zahan, Nusrat, Thomas Zimmermann, Patrice Godefroid, Brendan Murphy, Chandra Maddila ja Laurie Williams. 2022. “What are Weak Links in the npm Supply Chain?” Teoksessa *2022 IEEE/ACM 44th International Conference on Software Engineering: Software Engineering in Practice (ICSE-SEIP),* 331–340. https://doi.org/10.1145/3510457.3513044.

Zhang, Su, Xinwen Zhang, Xinming Ou, Liqun Chen, Nigel Edwards ja Jing Jin. 2015. “Assessing Attack Surface with Component-Based Package Dependency”. Teoksessa *Network and System Security,* toimittanut Meikang Qiu, Shouhuai Xu, Moti Yung ja Haibo Zhang, 405–417. Cham: Springer International Publishing. ISBN: 978-3-319-25645-0.

Zimmermann, Markus, Cristian-Alexandru Staicu, Cam Tenny ja Michael Pradel. 2019. “Smallworld with High Risks: A Study of Security Threats in the Npm Ecosystem”, 995–1010. SEC’19. Santa Clara, CA, USA: USENIX Association. ISBN: 9781939133069.