Antti Heimonen

Npm-pakettiekosysteemin uhat

Tietotekniikan kandidaatintutkielma

2. toukokuuta 2022

Jyväskylän yliopisto

Informaatioteknologian tiedekunta

Tekijä: Antti Heimonen

Yhteystiedot: antti.j.t.heimonen@student.jyu.fi

Ohjaaja: Antti-Jussi Lakanen

Työn nimi: Npm-pakettiekosysteemin uhat

Title in English: Security threats in npm packet registry

Työ: Kandidaatintutkielma

Opintosuunta: Kaikki opintosuunnat

Sivumäärä: 14+0

Tiivistelmä: Tässä kandidaatintutkielmassa luodaan katsaus npm-pakettivarastoon liittyviin tietoturvauhkiin. Tutkimus käy läpi, minkälaisia uhkia npm-pakettivaraston liittyy ja pohtii myös, mitä sovelluskehittäjä voi tehdä mahdollisten uhkien suhteen. Lopuksi toteamme, ettei npm sinällään ole erityisen vaarallinen, mutta sovelluskehittäjän kannalta on ehkä tärkeintä tiedostaa ulkoisen koodin käyttöön liittyvät riskit.

Avainsanat: npm, pakettivarasto

Abstract: In this bachelor's thesis we take a look at the security issues concerning npm packet ecosystem. We investigate some security issues that npm may have and what a developer can do to protect against these issues. At the end we reason that npm is not very dangerous itself, but it is important for a developer to know of these security concerns.

Keywords: npm, package manager, javascript

Sisältö

1 Johdanto

Node Package Manager (myöhemmin lyhyesti npm) on vuonna 2009 alkunsa saanut Node.js ohjelmalle kehitetty pakettienhallintaohjelma ja pakettivarasto, joka sisältää tuhansia JavaScript-paketteja. Kuka tahansa voi luoda paketteja npm-pakettivarastoon uudelleen käytettäväksi omissa ja mahdollisesti myös kolmansien osapuolien sovelluksissa. Sovelluskehittäjät voivat etsiä pakettivaraston tarjonnasta valmiita paketteja ja ladata näitä käytettäväksi omassa sovelluksessaan.

Verkkosivuillaan npm kertoo, että pakettivarastoa käyttää yli 11 miljoonaa kehittäjää ja sen sisältävän yli miljoona pakettia. Pakettivaraston valikoimasta löytyy paketteja hyvin pienistä ja yksinkertaisista toiminnoista aina suuriin verkkosivujen frontend-kirjastoihin ("npm:n verkkosivut" 2021). Sillä on siis hyvin laaja-alainen Javascript-kehittäjien käyttäjäkunta.

Tässä kirjallisuuskatsauksessa tutkitaan npm-kokonaisuuden tietoturvaa sovelluskehittäjän näkökulmasta: millaisia tietoturvaan liittyviä uhkia npm:n käyttöön sisältyy ja mitä sovelluskehittäjän olisi hyvä tietää näistä asioista npm-pakettivaraston tarjontaa hyödyntäessään.

1.1 Tutkittavan ongelman kuvaus

Sovelluskehittäjät kokevat hyötyvänsä valmiiden ulkopuolisten pakettien käytöstä ohjelmiston kehityksessä (Abdalkareem ym. 2017). Samalla kuitenkin nousee mieleen kysymys, minkälaisia riskejä npm ja ulkopuolisen koodin käyttäminen voi tuoda mukanaan. Voiko sovelluskehittäjä ottaa huoletta käyttöönsä minkä tahansa npm-pakettivarastosta löytyvän paketin vai voiko ulkopuolinen koodi tuoda mukanaan ongelmia?

Yksittäisen npm-paketin käyttöön ottaminen ei ole välttämättä yksinkertainen asia. Esimerkiksi tutkimuksessaan Alfadel ym. (2020) tutkivat, kuinka monta riippuvuutta yksinkertaiset, kehittäjän helposti itse toteutettavissa olevat paketit sisälsivät. Näistä ns. triviaaleista paketeista he huomasivat, että 47,8 % sisältää vähintään yhden ja 11,5 % peräti yli 20 riippuvuutta toisiin paketteihin. Tämän takia yksi projektiin tuotu näennäisesti pienen toiminnon toteuttava paketti saattaakin aiheuttaa kymmeniä riippuvuuksia muihin paketteihin.

Luvussa 2 kerrotaan npm-järjestelmän taustoista. Luvussa 3 tutkitaan npm-järjestelmään liittyviä uhkia. Neljännessä luvussa arvioidaan, kuinka npm-pakettien mukanaan tuomat tietoturvaongelmat voivat vaikuttaa ohjelmiston tietoturvallisuuteen.

2 Npm sovelluskehityksessä

2.1 Mikä on Npm-paketti?

Yksittäinen npm-paketti sisältää Javascript-koodia, joka voidaan suorittaa Node.js Javascript-ajoympäristöllä. Paketit voivat olla luonteeltaan hyvin erilaisia. Osa paketeista sisältää node-moduleita, joita sovelluskehittäjä voi ottaa käyttöönsä omassa sovellusprojektissaan valmiin koodin tavoin. Toiset paketit taas voivat sisältää esimerkiksi kehitystyökaluja, kuten koodin testaamiseen tarkoitettuja ohjelmia. Kaikkia paketteja yhdistää se, että ne suoritetaan Node.js Javascript-ajoympäristöllä.

Pakettivarastossa saatavilla olevan paketin lisääminen projektiin on helppoa. Komennolla *npm install paketin_nimi* saadaan asennettua haluttu paketti ja liitettyä se npm-projektiin.

2.2 Projektin hallinta npm:n avulla

Sovelluskehityksessä npm:n keskeinen tehtävä on hallita kehitysprojektissa käytettäviä paketteja, joita npm lataa keskitetystä pakettivarastosta. Pakettivarastoon julkaiseminen vaattii npm-tunnuksen, mutta julkisten pakettien lataaminen pakettivarastosta onnistuu ilman tunnusta. Pakettivaraston sisältöä voi selata npm:n verkkosivujen kautta.

Npm osaa hallita projektiin kuuluvien riippuvuuspakettien versioita ja päivittää niitä määritettyjen sääntöjen mukaisesti. Pakettien versiointi npm:ssä toteutetaan kolmella, toisistaan pisteellä erotetulla numerolla, esimerkiksi 2.4.3. Ensimmäinen numero on ns. major-versio. Tämän version muuttuessa pakettiin on tullut sellaisia muutoksia, ettei se välttämättä ole enää yhteensopiva aiempaa versiota käyttävän ohjelman kanssa. Keskimmäinen numero on puolestaan minor-versio. Minor-päivitys tuo pakettiin uusia ominaisuuksia, mutta paketin tulisi olla edelleen yhteensopiva edellisten, samaa major-versiota käyttävien ohjelmien kanssa. Viimeinen numero on taas patch-versio. Sen muuttuessa pakettiin ei ole tullut uusia ominaisuuksia, vaan muutokset ovat korjauksia, kuten esimerkiksi haavoittuvuuden paikkauksia.

Npm-paketteja hyödyntävään projektiin kuuluvassa package.json-tiedostossa määritetään pro-

jektissa käytettävät paketit ja niiden versionumerot. Jokaiselle paketille voidaan erikseen määritä tarkasti tietty käytettävä versio, tai sitten tehdä dynaaminen määritys siten, että käyttöön tulee aina esimerkiksi uusin minor-versio tietystä major-julkaisusta. Esimerkiksi paketin päivittäminen versiosta 2.4.5 sallittaisiin versioon 2.5.0, sillä minor-version päivityksen ei pitäisi rikkoa yhteensopivuutta ohjelman kanssa. Dynaamisen versioinnin ansiosta paketin ei tarvitse olla sidottu tiettyyn versioon, vaan npm voi ladata siitä aina uusimman version package.json-tiedostossa määritettävien sääntöjen rajoissa.

Jo olemassa olevan npm-projektin riippuvuuspakettien asentaminen on hyvin suoraviivaista. Projektihakemistossa suoritettava komento npm init lataa ja asentaa projektiin liitetyt kirjastot pakettivarastosta.

2.3 Riippuvuuspaketit

Yksikin projektiin lisätty npm-paketti saattaa tuoda mukanaan suuren määrän riippuvuuksia toisiin paketteihin. Myös näissä riippuvuuspaketeissa on usein omat riippuvuutensa, joten yksittäisen paketin lisääminen projektiin saattaa epäsuorien riippuvuksien kautta tuoda mukanaan suuren määrän ulkoisia paketteja. Abdalkareem ym. (2017) havaitsivat triviaaleja paketteja tutkiessaan, että 43,7 % näistä paketeista sisälsi vähintään yhden riippuvuuden, kun taas 11,5 % triviaaleista paketeista sisälsi jopa yli 20 riippuvuutta. Määrä voi tuntua todella suurelta, sillä triviaaleilla pakteilla on yleensä hyvin rajattu ja tarkka käyttötarkotus.

Epäsuorat riippuvuudet voivat siis tuoda projektiin suuren määrän ulkopuolisia paketteja. Riippuvuuksien suuri määrä ei vielä sinällään ole vaarallista, mutta jokainen paketti voi kuitenkin sisältää haavoittuvuuksia ja aiheuttaa ongelmia. Sovelluskehittäjät kokevat myös suuren riippuvuuspakettien määrän ongelmaksi (Abdalkareem ym. 2017).

2.4 Hylätyt paketit

Npm-pakettivarasto sisältää myös paketteja, joiden kehitys on kokonaan lopetettu. Jos tällaisesta hylätystä paketista löytyy haavoittuvuus, niin on hyvinkin mahdollista, ettei havaittuun ongelmaan ole tulossa korjausta. Hylätystä paketista löytyneen haavoittuvuuden vakavuutta

lisää se, että usein haavoittuvuudesta on julkaistu tietot avoimiin haavoittuvuustietokantoihin (Zimmermann ym. 2019). Hylätyn paketin haavoittuvuus voi siis olla käytännössä helpommin hyödynnettävissä oleva kohde.

2.5 Vanhentuneiden pakettien käyttäminen

Sen jälkeen, kun npm-pakettiin julkaistaan päivitys, täytyy kehittäjän tuoda päivitetty paketti ohjelmistoonsa. Alfadel ym. (2020) havaitsivat, että usein ohjelmistoissa käytettiin vanhentunutta ja haavoittuvaa versiota paketista, vaikka korjaus oli jo olemassa. Päivitetyn paketin tuominen ohjelmistoon on siis kehittäjän vastuulla. Toisaalta kehittäjä on myös voinut lukita käytössä olevan paketin version, jolloin uusi ja korjattu versio ei päivity automaattisesti käyttöön.

2.6 Väärin kirjoitettu paketin nimi

Eräs tietoturvauhka on vahingolliset paketit, joiden nimet muistuttavat tarkoituksella toista pakettia. Esimerkiksi jos oikea paketti olisi nimeltään "paketti", niin vahingollinen paketti olisi voitu nimetä "paketi". Haitallinen paketti voi päätyä sovelluksen käyttöön, mikäli kehittäjä tekee kirjoitusvirheen tai kopioi verkkosivuilta paketin asennuskomennon, joka on tarkoituksella kirjoitettu väärin. Koska npm tallentaa tarvittavien pakettien nimet package.json-tiedostoon, niin sovelluksen kehittäjistä vain yhden tarvitsee tehdä tämä virhe. Toisille kehittäjille haitallinen paketti asentuu ilman, että heidän tarvitsee välittää onko nimi kirjoitettu oikein vai ei. Myös npm-pakettivarastossa on yrityksiä hyödyntää typosquatting-hyökkäyksiä (Zimmermann ym. 2019).

2.7 Sovelluksen hajottava pakettipäivitys

Toisinaan paketin päivittäminen saattaa hajottaa sitä käyttävän ohjelman toiminnallisuuden. Paketti saattaa sisältää muutoksia, jotka aiheuttavat muutoksen tarpeen myös pakettia hyödyntäviin ohjelmiin. Luonteenomaista npm-ympäristössä näyttäisi olevan kehittäjien arvostus helppoa ja luotettavaa päivityssykliä kohtaan luotettavuuden kustannuksella (Bogart

ym. 2016).

2.8 Asiallisen paketin tahallinen saastuttaminen

Eräs npm-ekosysteemin piirre on pienen joukon vaikutusvalta hyvin suureen määrään paketteja epäsuorien riippuvuuksien kautta. Vuonna 2018 järjestelmässä oli 391 kehittäjää, jotka pystyivät omalla toiminnallaan vaikuttamaan yli 10 000 pakettiin, kun vuonna 2015 vastaava kehittäjien määrä oli 59, joten kasvutahti on ollut nopea (Zimmermann ym. 2019). Lisäksi vaikka suurin osa npm-paketeista sisältää riippuvuuksia muihin paketteihin, niin riippuvuutena toimivien pakettien joukko on suhteellisen suppea (Wittern, Suter ja Rajagopalan 2016). Yksittäisen kehittäjän laaja vaikutusvalta voidaan tulkita riskitekijäksi. Halutessaan kehittäjä voisi julkaista haitallisen päivityksen, joka saastuttaisi useita ohjelmia. Toisaalta taas, vaikka itse vaikutusvaltainen kehittäjä olisi hyväntahtoinen, voi esimerkiksi heidän tunnuksensa olla mielenkiintoinen kohde verkkorikollisille. Esimerkiksi heinäkuussa 2018 suositun eslint-scope paketin kehittäjän tunnukset päätyivät vääriin käsiin, jonka seuraksena rikolliset julkaisivat paketista haitallisen version, joka yritti lähettää tietokoneen paikallisia tiedostoja ulkoiselle palvelimelle (Zimmermann ym. 2019). Vastaava hyökkäys on mahdollista toteuttaa edelleen, sillä npm ei tarkasta julkaistujen pakettien koodia millään tavalla.

3 Uhkien vaikutuksen arviointi

Alfadel ym. (2020) tutkimus keskittyy havainnollistamaan npm-paketteja käyttävien todellisten sovelluksien haavoittuvuuksia niiden versiohallinnan historian avulla ja luokittelemaan tunnetut haavoittuvuudet kolmeen luokkaan vaarallisuuden mukaan: vähäisen, keskitason sekä suuren vaaran luokkiin.

Vähäisen vaaran luokkaan kuuluu haavoittuvuudet, jotka ovat olleet ohjelmassa, mutta on jo korjattu ennen haavoittuvuuden löytymistä. Keskitason vaaraan kuuluu haavoittuvuudet, jotka ovat ohjelmassa haavoittuvuuden löytymisen jälkeen, mutta ennen kuin haavoittuvuus on tuotu julki. Suurimman vaaran luokkaan kuuluu ne ohjelmissa esiintyvät haavoittuvuudet, jotka ovat vielä hyödynnettävissä haavoittuvuuden julki tulemisen jälkeen. He havaitsivat, että peräti 68% tutkituista ohjelmista käytti haavoittuvaa pakettia, mutta 95% näistä haavoittuvuuksista kuuluivat vähäisen vaaran luokkaan. Suurin osa oikeisiin ohjelmiin päätyneistä haavoittuvuuksista korjattiin siis ennen haavoittuvuuden julki tulemista. Vaikuttaisi siis, että npm-pakettien kehittäjät julkaiseevat hyvin päivityksiä. Myös haavoittuvuuksiin, jotka kuuluivat suurimman vaaran luokkaan, oli julkaistu päivitys 91% tapauksista. Haavoittuvaa pakettia ei kuitenkaan oltu päivitetty ohjelman tekijän toimesta. Sovelluskehittäjät vaikuttaisivat olevan laiskoja päivittämään käytettyjä riippuvuuspaketteja.

Vaikka npm-paketteihin julkaistaan siis julkaistaan hyvin tietoturvapäivityksiä, niin usein korjaus saattaa olla tehty paketin toiseen major-versioon (Alfadel ym. 2020). Major-versionumeron kasvaessa päivitetty paketti sisältää muutoksia, jonka takia se ei välttämättä ole enää yhteensopiva kehitettävän ohjelman kanssa. Päivitys voi siis aiheuttaa ylimääräistä työtä sovelluskehittäjälle. Alfadelin ym. (2020) tutkimus osoitti, että jopa 43 % npm-pakettien haavoittuvuuksien korjauksista oli tehty paketin toiseen major-versioon. Tämä saattaa olla yksi syy sovelluskehittäjien haluttomuuteen päivittää paketteja (Alfadel ym. 2020).

Erityisen heikko tilanne voi olla, jos ohjelmistoprojektissa käytetään hylättyjä paketteja, joista paljastuu haavoittuvuus. Myös tällaisia paketteja voi olla käytössä epäsuorien riippuvuuksien kautta (Zimmermann ym. 2019). Hylättyyn pakettiin ei todennäköisesti julkaista korjausta, joten kehittäjän on päätettävä, kuinka tilanteessa toimii. Eräs vaihtoehto voi olla haa-

voittuvan paketin vaihtaminen toiseen. Tämä aiheuttaa jonkin verran työtä riippuen siitä, kuinka helppo korvaava paketti on löytää ja minkälaisia eroavaisuuksia uudella ja vanhalla paketilla on keskenään. Toinen vaihtoehto on hylätä käytössä oleva haavoittuva paketti, ja toteuttaa itse sen toiminnallisuus. Myös tämän vaihtoehdon työmäärä riippuu hylättävästä paketista. Pieni yksinkertainen toiminnallisuus on nopeampi toteuttaa kuin suuri. Kolmas vaihtoehto on jättää haavoittuva paketti projektiin. Mikäli haavoittuvuus ei aiheutta suurta riskiä ohjelman tietoturvan ja toiminnallisuuden kannalta, niin haavoittuvan paketin käytöstä ei välttämättä koidu haittaa.

4 Pohdinta

Vaikka hyvin moni ohjelma käyttää haavoittuvia npm-kirjastoja, niin ongelma ei välttämättä ole niin paha, kun miltä se pelkästään näitä lukuja katsomalla vaikuttaa. Pelkästään se, että ohjelma käyttää haavoittuvaa pakettia, ei vielä tarkoita, että haavoittuvuutta voisi hyödyntää ohjelmaa vastaan (Alfadel ym. 2020). Mikäli ohjelma ei käytä paketin ongelmallista funktiota, kyseinen haavoittuvuus ei periydy ohjelmaan.

Valmis npm-paketti voi olla helppo ja nopea ratkaisu halutun toiminnallisuuden saavuttamiseksi ja siksi todella houkutteleva vaihtoehto. Yksinkertaisen toiminnon suhteen kehittäjän olisi hyvä harkita, että etsiikö toiminnon toteuttamiseen npm-paketin vai olisiko haluttu toiminto toteutettavissa omin resurssein. Toisaalta toteuttamalla yksinkertaisen toiminnon itse, tulee ohjelmistoprojektiin vähemmän riskialttiita ulkoisia riippuvuuksia. Molemmissa lähestysmistavoissa on siis omat hyötynsä ja haittansa.

Jos kyseessä on paketti, joka on todella yksinkertainen, niin projektin voisi määrittää käyttämään paketista vain tiettyä versiota. Tällä tavoin paketti ei päivity automaattisesti, joten, jos paketista julkaistaan myöhemmin haitallinen versio, niin haavoittuvuudet eivät pääse kehitettävään ohjelmaan asti. Tämän ratkaisun haittapuolena on selvästi se, ettei paketti päivity myöskään asiallisilla päivityksillä, jolloin myös tärkeä turvapäivitys voi jäädä saamatta. Jos yksinkertaisen paketin lisenssi sallii koodin vapaan käyttämisen, niin kehittäjä voisi kopioida koodin luodakseen oman itsenäisen version kyseisestä paketista. Kopioitavasta koodista voi samalla jättää pois myös ominaisuuksia, joita kehitettävässä ohjelmassa ei tarvita.

On olemassa myös työkaluja, joilla sovelluskehittäjä voi saada tietoa npm-pakettien tietoturvauhista. Esimerkiksi npm-ohjelmiston mukana tuleva npm audit käy läpi npm-projektiin liitetyt riippuvuuspaketit ja niiden versiot, ja ilmoittaa niistä löytyvistä haavoittuvuuksista (Zimmermann ym. 2019). Tämän tarkastuksen tiedot perustuvat jo löydettyihin haavoittuvuuksiin. Myös esimerkiksi Githubista löytyy toiminnallisuus, jonka avulla Github-käyttäjä saa ilmoituksen, mikäli Githubissa säilöttävästä projektista löytyy tunnettu haavoittuvuus. Sovelluskehittäjä voi saadun tiedon perusteella päätää minkälaisia toimia löydetty tietoturvauhka vaatii.

Kuitenkin ongelmien ilmaantymisen myötä myös npm:n toiminnassa on tapahtunut kehitystä. Esimerkiksi npm-paketin kehittäjä ei voi enää poistaa pakettiaan järjestelmästä ja aiheuttaa sitä kautta ongelmia toisille, poistetuista paketista riippuvaisille paketeille (Abdalkareem ym. 2017).

Ehkä kaikkein tärkeintä npm-pakettivaraston käytön kannalta on tiedostaa npm:n ominaispiirteet ja tietoturvaan liittyvät riskit. Pakettivaraston tarjontaa selatessa kannattaa kiinnittää huomiota paketin tarjoaman toiminnalisuuden lisäksi myös sen tietoihin, kuten viimeisimmän julkaisun ajankohtaan. Kehittäjät, jotka julkaisevat paketteja npm-pakettivarastoon, arvostavat nopeaa ja helppoa julkaisuprosessia, jonka npm heille mahdollistaa (Bogart ym. 2016). Pakettivaraston tarjontaa hyödyntävät käyttäjät taas osaavat odottaa, että ulkopuolisen paketin päivitys voi rikkoa kehitettävän ohjelman toiminnan ja että tuolloin on määritettävä käyttöön ongelmallisen paketin vanhempio versio (Bogart ym. 2016). Ymmärrys npm-pakettivaraston toimintatavasta ja luonteesta auttaa todennäköisesti sovelluskehittäjiä tekemään parempia päätöksiä valmiita koodipaketteja hyödyntäessään.

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. doi: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.* arXiv: 2009.09019 [cs.SE].

Bogart, Christopher, Christian Kästner, James Herbsleb ja Ferdian Thung. 2016. "How to Break an API: Cost Negotiation and Community Values in Three Software Ecosystems". Teoksessa *Proceedings of the 2016 24th ACM SIGSOFT International Symposium on Foundations of Software Engineering,* 109–120. FSE 2016. Seattle, WA, USA: Association for Computing Machinery. ISBN: 9781450342186. doi:10.1145/2950290.2950325.

"npm:n verkkosivut". 2021. Viitattu 23. maaliskuuta 2021. https://www.npmjs.com/.

Wittern, Erik, Philippe Suter ja Shriram Rajagopalan. 2016. "A Look at the Dynamics of the JavaScript Package Ecosystem". Teoksessa *Proceedings of the 13th International Conference on Mining Software Repositories,* 351–361. MSR '16. Austin, Texas: Association for Computing Machinery. ISBN: 9781450341868. doi:10.1145/2901739.2901743.

Zimmermann, Markus, Cristian-Alexandru Staicu, Cam Tenny ja Michael Pradel. 2019. "Small World with High Risks: A Study of Security Threats in the npm Ecosystem". Teoksessa *28th USENIX Security Symposium (USENIX Security 19),* 995–1010. Santa Clara, CA: USENIX Association, elokuu. ISBN: 978-1-939133-06-9. https://www.usenix.org/conference/usenixsecurity19/presentation/zimmerman.