Tampereen yliopisto

Emil Calonius

TIETOTURVARISKIT AVOIMEN LÄHDE-KOODIN JAVASCRIPT-KIRJASTOISSA

Informaatioteknologian ja viestinnän tiedekunta
Kandidaattitutkielma
Toukokuu 2022

TIIVISTELMÄ

Emil Calonius: Tietoturvariskit avoimen lähdekoodin JavaScript-kirjastoissa
Kandidaattitutkielma
Tampereen yliopisto
Tietojenkäsittelytieteiden tutkinto-ohjelma
Toukokuu 2022

Avoimen lähdekoodin kirjastojen käyttö web-ohjelmoinnissa on yleinen käytäntö, joka tuo ohjelmoijille paljon etuja esimerkiksi säästämällä aikaa, vaivaa sekä rahaa. Tämä käytäntö vaatii ohjelmoijilta kuitenkin tarkkuutta, sillä kirjastot voivat tuoda hyökkääjille uusia hyökkäysvektoreita tietoturva-aukkojen kautta. Tämän työn tavoite on tutkia minkälaisia riskejä avoimen lähdekoodin JavaScript-kirjastot tuovat web-ohjelmointiin. Lisäksi työssä esitetään konkreettisia toimia joihin kehittäjät voivat ryhtyä minimoidakseen tietoturvariskit.

Tutkielma on toteutettu kirjallisuuskatsauksena. Lähteinä toimivat alan tieteelliset artikkelit sekä konferenssipaperit, verkkojulkaisut ja raportit. Web-ohjelmointi on nopeasti kehittyvä ala, joten lähteiksi on valittu ensisijaisesti julkaisuja viimeisen viiden vuoden ajalta.

Tutkimuksen tulokset osoittavat, että riskejä tuottavat haavoittuvaiset sekä vanhentuneet kirjastot. Haavoittuvaiset kirjastot tarkoittavat kirjastoja, joissa syystä tai toisesta esiintyy haavoittuvaisuus. Usein syynä haavoittuvaisuudelle on kehittäjän tekemä ohjelmointivirhe. Haavoittuvaisen kirjaston tapauksessa olisi tärkeää, että sen ylläpitäjä saisi haavoittuvaisuuden korjattua mahdollisimman pian ja että hän kommunikoisi avoimesti kirjaston käyttäjien kanssa, jotta uusin päivitys tavoittaisi mahdollisimman monet ja että haavoittuvaisen version tuomat uhkat olisivat selvät. Vanhentuneiden kirjastojen tapauksessa haavoittuvaisuus on jo korjattu uusimmassa versiossa, mutta kirjaston käyttäjä ei ole vielä ottanut uusinta versiota käyttöön. Tällaisen tilanteen ehkäisemiseksi myös kirjaston käyttäjän on otettava itselleen vastuuta ja seurattava kirjaston ylläpitäjien viestintää sekä varmistettava ajoittain, että kirjastot ovat ajan tasalla.

Avainsanat: JavaScript, web-ohjelmointi, tietoturva, avoin lähdekoodi

Tämän julkaisun alkuperäisyys on tarkastettu Turnitin OriginalityCheck –ohjelmalla.

Sisällysluettelo

1 Johdanto 1

2 Web-ohjelmointi 3

2.1 Avoimen lähdekoodin JavaScript-kirjastot 3

2.2 Tietoturva 4

3 Tietoturvariskit 6

3.1 Yleisimmät haavoittuvaisuudet 6

3.2 Haavoittuvaiset kirjastot 8

3.3 Vanhentuneet kirjastot 9

3.4 Epäsuorat riippuvuudet 10

4 Parhaat käytännöt 11

4.1 Avoimen lähdekoodin ylläpitäjät 11

4.2 Avoimen lähdekoodin käyttäjät 12

5 Yhteenveto 14

Lähdeluettelo 15

1 Johdanto

Avoimen lähdekoodin kirjastojen käyttö osana web-ohjelmointia on nykypäivänä yleinen käytäntö. Esimerkiksi NPM, suosittu pakettienhallintajärjestelmä web-ohjelmoinnissa, sai vuonna 2019 yli 300 000 uutta pakettia rekisteriinsä, mikä on yli 30 prosentin kasvu vuoteen 2018 verrattuna (Snyk, 2020). Tämä kasvu ei ole yllättävää, sillä avoin lähdekoodi säästää kehittäjiltä aikaa, vaivaa sekä rahaa (Ballhausen et al., 2019). Samana vuonna viiden suurimman avoimen lähdekoodin ekosysteemin joukossa NPM-ekosysteemistä löydettiin toiseksi eniten uusia haavoittuvuuksia (Snyk, 2020). Tietoturvariskit ovat vakava uhka, joihin web-kehittäjien on kiinnitettävä huomiota.

Tietoturva-aukon mahdollistamien hyökkäysten tuomien riskien laajuutta kuvaa erinomaisesti vuonna 2014 löydetty haavoittuvaisuus erittäin suositussa OpenSSL-kirjastossa. OpenSSL on avoimen lähdekoodin kirjasto, joka mahdollistaa internet-sovellusten välisen tietoliikenteen salaamisen. Ohjelmointivirheestä johtuneen haavoittuvaisuuden sisältänyt päivitys julkaistiin vuonna 2011, ja se ehti vuotaa tietoa useamman vuoden ajan ennen kuin se lopulta havaittiin ja korjattiin vuonna 2014. Haavoittuvaisuuden takia esimerkiksi käyttäjien salasanoja on voinut päätyä hyökkääjien käsiin. (Heartbleed, 2022.) Kyseinen haavoittuvaisuus oli äärimmäisen vakava, sillä OpenSSL oli vuonna 2014 käytössä yli 66 prosentissa kaikista aktiivisista verkkosivuista, sillä Apache- ja Nginx-palvelimet molemmat käyttävät sitä (Netcraft, 2014).

Tutkielman tavoitteena on selvittää millä, tasolla JavaScript-kirjastojen tietoturva nykypäivänä on ja minkälaisia tietoturvariskejä niiden käyttäminen osana web-sovelluksia tuo mukanaan. Lisäksi tutkielmassa tullaan esittämään konkreettisia käytäntöjä, joiden avulla web-sovellusten tietoturvaa voidaan parantaa.

Aiheen valintaan on vaikuttanut oma kiinnostus web-ohjelmointia kohtaan sekä siihen liittyvien tietoturvaongelmien tärkeys. Tietoturvaan on helppo olla kiinnittämättä huomiota web-sovellusta kehittäessä, ja se voi tuntua raskaalta, kun haluaisi keskittyä vain oman vision toteuttamiseen. Tämän takia koen tärkeäksi, että ohjelmoijille tarjottaisiin työkaluja ja konkreettisia toimia, joihin he voivat ryhtyä välittömästi tietoturvan vahvistamiseksi niin, etteivät tietoturva-asiat vie liikaa resursseja web-kehityksen muilta osa-alueilta.

Tämä tutkielma on toteutettu kirjallisuuskatsauksena. Lähteet on kerätty SpringertLink- ja IEE Electronic Library -tietokannoista hakulausekkeella “javascript AND (vulnerability OR security) AND ("open source" OR "third party") AND npm”. IEE Electronic Library -tietokannassa hakutulokseksi saatiin viisi konferenssipaperia. Tulokset on rajattu SpringerLink-tietokannassa tieteellisiin artikkeleihin tietojenkäsittelytieteiden alalta, jolloin hakutuloksia tästä tietokannasta löytyi 29. Tämän jälkeen sopivat lähteet on valittu lukemalla hakutuloksien tiivistelmät. Näiden lisäksi lähteinä on käytetty myös verkkojulkaisuja ja yhtä raportteja.

Tutkielman toisessa luvussa käydään läpi tutkielman aiheeseen liittyviä termejä, joiden tunteminen on tärkeää tutkielman myöhemmissä luvuissa. Kolmannessa luvussa esitellään kirjallisuuskatsauksessa esille nousseet tietoturvariskit ja -uhat. Neljännessä luvussa pohditaan, kuinka riskejä voidaan hallita ja haittavaikutuksia minimoida ottamalla käyttöön alan parhaiksi todettuja käytäntöjä sekä avoimen lähdekoodin ylläpitäjien että käyttäjien näkökulmasta.

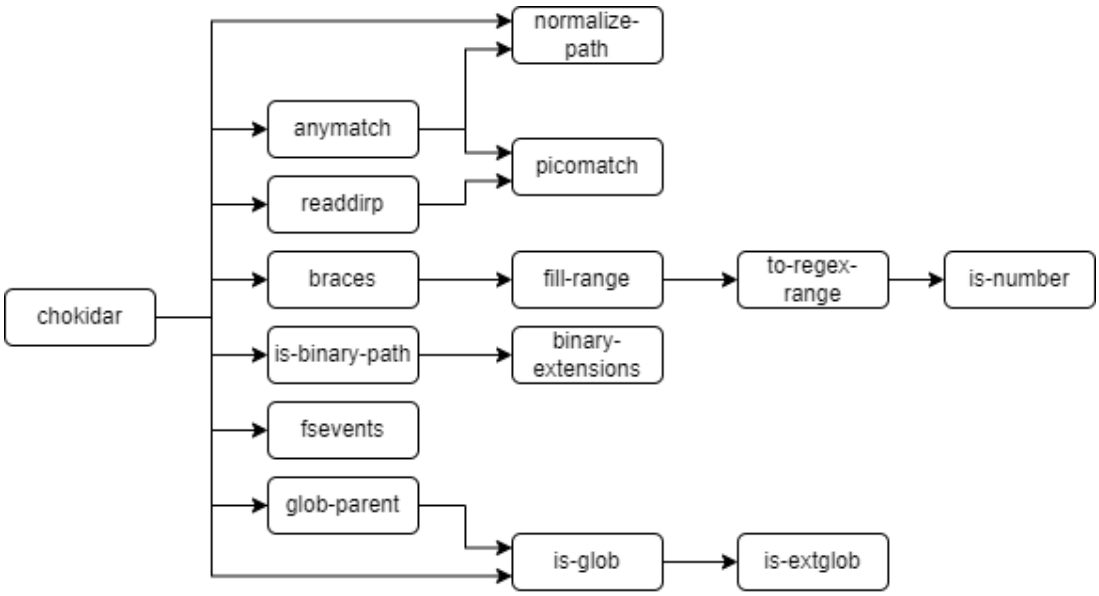
2 Web-ohjelmointi

Tässä luvussa käydään läpi web-ohjelmointiin liittyviä keskeisiä käsitteitä, jotka liittyvät läheisesti tutkielman aiheeseen. Web-ohjelmointi sisältyy laajempaan käsitteeseen, web-kehitys, joka pitää sisällään myös esimerkiksi web-suunnittelun. Web-ohjelmointi itsessään tarkoittaa verkkosivun tai verkkosovelluksen kehittämistä kirjoittamalla koodia eri ohjelmointikielillä, kuten JavaScriptillä.

2.1 Avoimen lähdekoodin JavaScript-kirjastot

JavaScript on ohjelmointikieli, joka on kehitetty varta vasten dynaamisen toiminnallisuuden mahdollistamiseksi verkkosivuilla. Sen avulla voidaan esimerkiksi muokata verkkosivun sisältöä vaikkapa käyttäjän toiminnan perusteella selaimen tarjoaman DOM-rajapinnan avulla. JavaScript on HTML:n ja CSS:n kanssa yksi olennaisista teknologioista verkkosivujen kehittämistä varten. JavaScriptistä tekee joustavan se, että ohjelmoija voi helpottaa omaa työtään ottamalla käyttöön erilaisia valmiiksi luotuja kirjastoja, jotka tuovat mukanaan hyödyllisiä ominaisuuksia ohjelmoijan käytettäväksi.

Kolmansien osapuolten avoimen lähdekoodin JavaScript-kirjastojen hyödyntäminen web-ohjelmoinnissa on nykyään yleinen käytäntö. Se helpottaa ja nopeuttaa kehittäjien työtä tarjoamalla valmiiksi kirjoitettua toimivaa koodia käytettäväksi osana JavaScript-pohjaisia sovelluksia. Kun näitä kirjastoja otetaan osaksi kehitettävää web-sovellusta, niitä kutsutaan ohjelman riippuvuuksiksi. Jokainen kirjasto voi myös sisältää omia riippuvuuksiaan, jolloin ne asennetaan myös osaksi sovellusta.



Kuva 1. Riippuvuudet Chokidar-kirjaston versiossa 3.5.3

Kuva 1 kuvaa, kuinka yksittäinen riippuvuus saattaa tuoda lukuisia muita riippuvuuksia mukanaan. Chokidar on JavaScript-kirjasto, joka on luotu seuraamaan tiedostoihin tehtyjä muutoksia. Chokidarilla on omat riippuvuutensa, joilla saattaa olla myös omia riippuvuuksia. Näitä paketteja kutsutaan ohjelman epäsuoriksi riippuvuuksiksi. Kun ohjelmoija lataa Chokidar-paketin osaksi ohjelmaansa, hän lataa siis todellisuudessa 15 eri pakettia. Kun käytössä on useampia tämän kaltaisia kirjastoja, oman JavaScript-ohjelman riippuvuuksia voi olla vaikea hallita.

Avoimen lähdekoodin JavaScript-paketteja on helppo asentaa esimerkiksi paketinhallintajärjestelmän Node Package Managerin (NPM) avulla. Paketit ovat käytännössä vain ladattavaksi tehtyjä kirjastoja, ja molempia termejä käytetään tässä tutkielmassa viittaamaan samaan asiaan. NPM (2022) mainostaa verkkosivuillaan, että sen ohjelmistorekisteri sisältää yli miljoona pakettia ja on täten maailman suurin paketinhallintajärjestelmä. NPM:ää käyttää maailmanlaajuisesti yli 11 miljoonaa kehittäjää. Muitakin paketinhallintajärjestelmiä on olemassa JavaScript-kirjastoja varten, kuten Facebookin kehittämä Yarn. NPM on kuitenkin näistä suurin, ja se on siksi myös vahvasti edustettuna tämänkin tutkielman lähteissä. Paketinhallintajärjestelmien avulla pakettien asennus onnistuu vaivattomasti ilman rekisteröitymistä komentoliittymän kautta.

2.2 Tietoturva

Tietoturvalla tarkoitetaan toimia, joilla voidaan taata tiedon luottamuksellisuus, eheys ja käytettävyys (Kyberturvallisuuskeskus, 2022). Käytännössä tämä tarkoittaa sitä, että tallennettu tieto, kuten vaikkapa käyttäjien henkilötiedot, ovat ainoastaan saatavilla niille, joilla on oikeus tiedon tarkasteluun tai muokkaamiseen. Tietoturvan merkitys on nykypäivänä suuri, sillä erilaiset verkkopalvelut keräävät jatkuvasti käyttäjiltä sensitiivistä tietoa toimintansa mahdollistamiseksi.

Tietoturva-aukolla tai haavoittuvaisuudella tarkoitetaan jotakin sellaista piirrettä tietojärjestelmässä, jota hyökkääjä voi käyttää hyväkseen päästäkseen käsiksi tietoihin, joihin hänellä ei ole oikeutta. Tietoturva-aukot voivat johtua esimerkiksi käyttäjän tekemästä virheestä, kuten heikon salasanan valitsemisesta tai järjestelmän ominaisuuksista, joita hyökkääjä voi käyttää hyväkseen. (NCSC, 2015.) Tässä tutkielmassa tietoturva-aukolla

tarkoitetaan usein ohjelmointivirheestä johtuvaa heikkoutta, jota rikollinen pääsee hyödyntämään hyökkäyksessään. Ohjelmointivirheestä johtuvan tietoturva-aukon sisältävän paketin asentaminen osaksi ohjelmaa tarkoittaa, että ohjelma on haavoittuvainen hyökkäyksille.

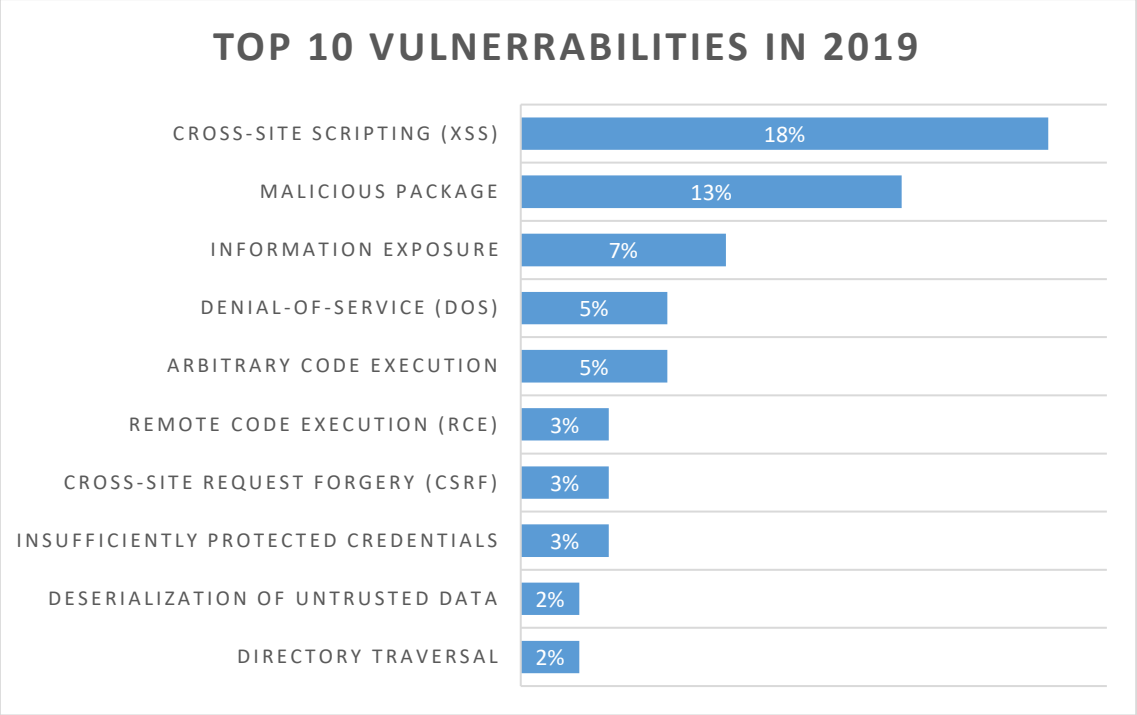
Tietoturvariski ja -uhka sekoittuvat helposti termeinä toisiinsa. Näiden termien ero on helpointa selventää esimerkkien avulla. Tietoturvauhalla tarkoitetaan jotain sellaista tapahtumaa, joka rikkoisi järjestelmän tietoturvan. Tietoturvauhka voisi olla esimerkiksi salatun tiedon joutuminen hyökkääjän käsiin. Tietoturvariskillä tarkoitetaan tietoturvauhan mahdollistavaa tekijää, kuten huonosti koulutettua työntekijää tai haavoittuvaista ohjelmistoa. Tietojärjestelmiin voi kohdistua useita tietoturvauhkia ja suunnitteluvaiheessa niiden huomioiminen on tärkeää, jotta tietoturvariskit voidaan minimoida.

3 Tietoturvariskit

Tietoturvaan liittyvien ongelmien välttämiseksi kehittäjien on äärimmäisen tärkeää tuntea hyökkääjien käyttämät hyökkäysvektorit sekä olla tietoisia siitä, mitkä seikat voivat helpoiten johtaa haavoittuvaisuuksiin omassa projektissa. Tässä luvussa tarkastellaan avoimen lähdekoodin kirjastojen käyttöön liittyviä tietoturvariskejä ja asioita, joilla on vaikutusta ohjelmien tietoturvan tasoon.

3.1 Yleisimmät haavoittuvaisuudet

State Of Open Source Security -raportissa (Snyk, 2020) esitellään selkeästi vuoden 2019 suurimmat avoimen lähdekoodin haavoittuvuudet. Kuvien 1 ja 2 taulukoidensisältö on saatu suoraan Snykin (2020) raportin tutkimustuloksista. Taulukot ovat englanninkieliset, sillä monet englanninkieliset termit eri haavoittuvuuksille ovat vakiintuneet käyttöön myös Suomessa.

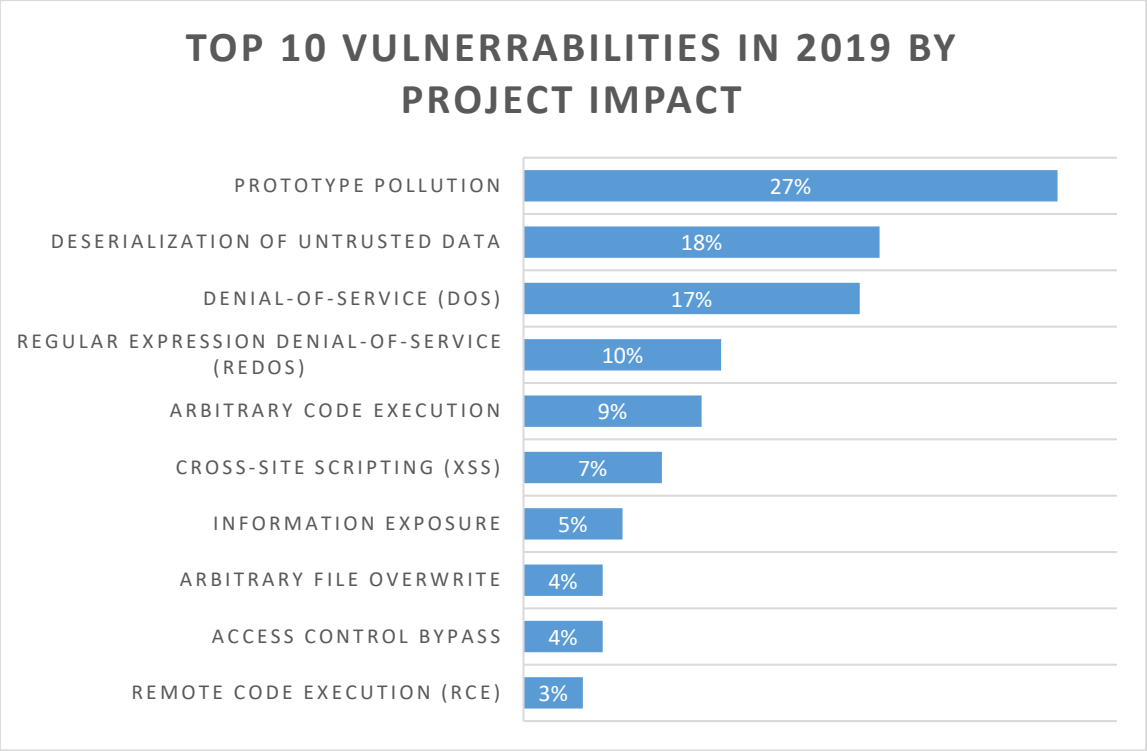


Kuva 2. Kymmenen suurinta haavoittuvuutta vuonna 2019 (Snyk, 2020).

Kuvasta 2 nähdään, että kaksi suurinta haavoittuvuutta muodostavat yli 30 prosenttia kaikista haavoittuvaisuuksista. XSS eli cross-site scripting on ollut yksi yleisimmistä haavoittuvuuksista jo reilusti yli vuosikymmenen ajan ja olikin Snykin raportissa vuonna

2019 kaikista yleisin (Snyk, 2020). XSS-hyökkäys tapahtuu, kun hyökkääjä lähettää haitallista koodia haavoittuvaisen verkkosivun kautta uhrin selaimeen. Hyökkäys voidaan toteuttaa käytännössä kaikilla verkkosivuilla, jotka ottavat vastaan käyttäjältä syötteitä tarkistamatta niitä. Hyökkäyksellä voidaan muun muassa kaapata uhrin evästeet tai muita selaimen tallentamia arkaluontoista tietoja. (Owasp, 2022)

Toiseksi suurin haavoittuvaisuus vuonna 2019 oli haitalliset paketit (malicious package). Haitallisilla paketeilla tarkoitetaan paketteja, jotka ovat varta vasten suunniteltu siten, että ne sisältävät haitallista koodia. Haitalliset paketit voivat myös olla luotettavia paketteja, jotka hyökkääjä on kaapannut ja saanut sisältämään omaa haitallista koodiaan. (Snyk, 2022) Etenkin jälkimmäisessä tilanteessa haavoittuvaisuudella on mahdollisuus levitä laajalle ja vaikuttaa moniin eri projekteihin. Haitallisten pakettien avulla hyökkääjät käyttävät hyväksi paketinhallintajärjestelmien ja avoimen lähdekoodin kirjastojen suurta suosiota ja jatkuvaa kasvua.



Kuva 3. Kymmenen suurinta haavoittuvuutta niiden vaikutuksen perusteella vuonna 2019 (Snyk, 2020).

Kuvassa 3 on kuvattu kymmenen suurinta haavoittuvuutta sen perusteella, kuinka laajalle ne ovat levinneet eli kuinka moneen projektiin ne ovat vaikuttaneet. Kirjastojen sisältämät

haavoittuvaisuudet voivat levitä nopeasti, mikäli niitä otetaan osaksi monia projekteja. Tällaisessa tilanteessa yhdenkin haavoittuvaisen kirjaston aiheuttamat vahingot voivat kasvaa valtavan suuriksi. Tästä erinomainen esimerkki on äärimmäisen suositusta Lodash-kirjastosta löydetty Prototype Pollution -haavoittuvaisuus. Haavoittuvaisuus oli läsnä kirjaston jokaisessa versiossa, mikä vaikutti haavoittuvaisuuden laajaan vaikutukseen. (Snyk, 2020.) Lodash on JavaScript-kirjasto, joka tuo ominaisuuksia helpottamaan esimerkiksi taulukoiden, merkkijonojen, numeroiden ja olioiden kanssa työskentelyä. Sen tarjoamia apuvälineitä voi hyödyntää lähes missä tahansa projekteissa, minkä takia sitä ladataan kuukausittain kymmeniä miljoonia kertoja.

Prototype Pollution -hyökkäys on ajonaikainen hyökkäys, jossa hyökkääjä pääsee muokkaamaan olioiden attribuutteja. Tämä voi johtaa monenlaisiin vaikutuksiin. Hyökkääjä voi esimerkiksi kaataa ohjelman tai ajaa omaa haitallista koodia etäältä.

3.2 Haavoittuvaiset kirjastot

Antal et al. (2020) toteavat artikkelissaan JavaScript-kehittäjien olevan hitaita korjaamaan haavoittuvaisuuksia koodissaan. Heidän suorittamassa tutkimuksessa kerättiin commit-tietoja useista eri julkisista projekteista. Näiden tietojen perusteella pystyttiin tutkimaan, kuinka paljon aikaa haavoittuvaisuuden julkaisemisen ja sen korjaamisen välillä kuluu. Tiedot ovat vuosilta 2010–2018. Tuloksista selviää, että keskimäärin haavoittuvuuden paikkaamiseen kestää noin 100 päivää. Tämä tarkoittaa, että haavoittuvuuksien korjaaminen on usein hyvinkin hidasta, mikä johtaa siihen, että haavoittuvaiset kirjastot ovat kierrossa pitkään. Tänä aikana haavoittuvaisuudet voivat levitä muihin projekteihin, joissa ne pysyvät tietoturva-aukkoina, kunnes haavoittuvaisuus on korjattu ja riippuvuus päivitetty.

Decan et al. (2018a) tutkivat artikkelissaan, kuinka monien pakettien tiedetään sisältävän haavoittuvaisuuksia, kuinka kauan ne pysyvät haavoittuvaisina ja kuinka nopeasti haavoittuvaisuudet korjataan sekä paketissa itsessään että riippuvaisissa paketeissa. Tutkimuksessa kävi ilmi, että haavoittuvaisuuksien korjaamisessa kestää usein kauan. Yksi syy tähän on vanhentuneiden pakettien käyttö. Pakettien sallittuja versioita voidaan rajoittaa semanttisen versionumeroinnin avulla. Jos nämä rajoitukset ovat liian tiukkoja, paketit saattavat olla riippuvaisia haavoittuvaisista vanhentuneista versioista vielä senkin jäl-

keen, kun haavoittuvaisuus on korjattu uudemmassa versiossa. (Decan et al., 2018a; Decan et al., 2018b.) Tämän takia on tärkeä tarkkailla omia riippuvaisuuksiaan jatkuvasti, jotta mahdollisen haavoittuvaisuuden ilmetessä voidaan toimia mahdollisimman nopeasti vahinkojen minimoimiseksi tai uuden haavoittuvaisuuden korjaavan version ilmestyessä se voidaan ottaa käyttöön mahdollisimman nopeasti.

Puolella kaikista Decanin ja muiden (2018b) tutkimista 2786:sta haavoittuvaisuuksista kestää yli kaksi vuotta tulla julkiseksi ja yli neljä vuotta tulla korjatuksi. Haavoittuvuudet on kerätty tutkittavaksi Snykin haavoittuvaisuuksia sisältävästä tietokannasta. Avoimen lähdekoodin ylläpitäjien tulisi panostaa enemmän resursseja tietoturva-aukkojen etsimiseen etenkin silloin, kun kirjastolla on paljon riippuvaisia kirjastoja. Avoimen lähdekoodin ylläpitäjille tulisi myös kehittää parempia työkaluja sekä haavoittuvuuksien löytämistä että korjaamista varten. (Decan et al., 2022.)

3.3 Vanhentuneet kirjastot

Kirjastojen käyttöönotto on paketinhallintajärjestelmien avulla äärimmäisen helppoa, mikä voi johtaa siihen, että kehittäjät eivät aina toimi tietoturvan kannalta parhaalla tavalla. Riippuvuuslistan kasvaessa tietoturva-aukkojen riski kasvaa varsinkin silloin, jos mukana on vähemmän tunnettuja kirjastoja, eikä niihin ole tutustuttu tarkasti ennen niiden ottamista osaksi kehitettävää sovellusta. Abdalkareem et al. (2020) tutkivat artikkelissaan yksinkertaisten JavaScript-pakettien (trivial packages) käyttöä osana web-sovellusten kehitystä. Osana tutkimusta 1055:lle JavaScript kehittäjälle lähetettiin linkki kyselyyn, johon saatiin vastauksia 125 kappaletta. Kyselyyn vastanneiden kesken yksinkertaisten pakettien huonoiksi puoliksi annettiin muun muassa riippuvuuksien pitäminen ajan tasalla (55,7 % vastaajista) ja tietoturvaan liittyvät huolet (8,0 % vastaajista). Riippuvuuksien ajan tasalla pitämiseen liittyen eräs vastaajista sanoi: ”On raskasta ylläpitää monia pieniä paketteja” (“Hard to maintain a lot of tiny packages”). Toinen vastaaja taas huomautti: ”Jos riippuvuuksien versioita ei hallitse aktiivisesti voi kohdata vakavia ongelmia” (“[...] people who don’t actively manage their dependency versions could [be] exposed to serious problems [...]”). Näistä vastauksista huomataan, että riippuvuuksien hallinnointia pidetään raskaana taakkana kehittäjille. Tämä on suuri riski, sillä vanhentuneet paketit voivat sisältää haavoittuvaisuuksia, jotka uudemmat versiot olisivat jo korjanneet. Yksinkertaisten pakettien käyttöä on harkittava tarkkaan ja otettava käyttöön vain paketteja, jotka ovat välttämättömiä, jotta niiden pitäminen ajan tasalla pysyy hallittavissa.

Kula et al. (2018) toteavat artikkelissaan, että kehittäjät ovat melko huonoja pitämään käyttämiään kirjastoja ajan tasalla. Tämä tukee Abdalkareemin ja muiden (2020) teettämän tutkimuksen tuloksia, joiden mukaan riippuvuuksien ajan tasalla pitäminen koetaan raskaaksi. Kirjastojen pitäminen ajan tasalla on tietoturvan kannalta äärimmäisen tärkeää, sillä se on yksi keino haavoittuvaisuuksien paikkaamiseksi. Kula et al. (2018) kertovat artikkelissaan, että yli 4 600:sta tutkitusta julkisesta GitHub-projektista jopa 81,5 % ei ole päivittänyt riippuvuuksia vaan käyttää niiden vanhentuneita versioita. Samassa artikkelissa selviää myös, että haastatelluista kehittäjistä 69 % ei tiennyt käytettyjen kirjastojen sisältävän haavoittuvaisuuksia. Tulokset viittaavat siihen, että riippuvuuksien pitäminen ajan tasalla ei ole monille kehittäjille etusijalla. Tämä voi usein johtaa harmillisesti tietoturvan heikkenemiseen.

3.4 Epäsuorat riippuvuudet

Organisaatioille on tärkeää tuntea heidän käyttämänsä riippuvuudet, jotta he voivat hallita niitä tehokkaasti ja välttää tietoturva-aukkoja. Se on kuitenkin hankalaa, sillä suorien riippuvuuksien lisäksi tulisi tuntea myös epäsuorat riippuvuudet. Suurin osa ohjelmien haavoittuvaisuuksista johtuvat nimenomaan epäsuorista riippuvuuksista. NPM-ekosysteemissä tämä määrä on peräti 86 prosenttia kaikista löydetyistä haavoittuvaisuuksista. (Snyk, 2020.) Ongelmasta tekee vakavamman se, että yli 60 prosenttia Snykin (2020) haastattelemista organisaatioista ei tunne tarkasti heidän kaikkia epäsuoria riippuvuuksiaan. Tämä johtaa siihen, että organisaation on hankalaa lähteä selvittämään vaikuttaako jokin hiljattain tunnistettu haavoittuvaisuus avoimen lähdekoodin kirjastossa juuri heidän ohjelmaansa.

Yhdenkin epäsuoran riippuvuuden hajoaminen voi vaikuttaa moniin muihin siitä riippuvaisiin paketteihin. Ongelmia voi aiheuttaa esimerkiksi paketin poistuminen kokonaan paketinhallintajärjestelmästä tai uuden päivityksen mukana tulevat yhteensopivuusongelmat muiden pakettien kanssa. Vuonna 2016 NPM-ekosysteemistä poistettiin paketti nimeltä left-pad. Tämä johti tuhansien siitä riippuvaisten pakettien hajoamiseen. (Decan et al., 2017.) Riippuvuuksien päivittäminen saattaa siis aiheuttaa ylimääräistä ylläpitoa edellä mainituista syistä. Tämän takia kehittäjille on helpompaa olla päivittämättä riippuvuuksiaan, jolloin tärkeät tietoturvapäivitykset jäävät saamatta.

4 Parhaat käytännöt

Tässä luvussa käydään läpi mihin konkreettisiin toimiin sekä avoimen lähdekoodin ylläpitäjät että käyttäjät voivat ryhtyä tietoturvan parantamiseksi. Avoimen lähdekoodin ylläpitäjät ovat kehittäjiä, yrityksiä tai organisaatioita, jotka ovat luoneet ja julkaisseet kirjaston tai kirjastoja avoimeen lähdekoodiin. Avoimen lähdekoodin käyttäjillä taas tarkoitetaan kaikkia, jotka käyttävät avoimen lähdekoodin kirjastoja osana omia projektejaan. Avoimen lähdekoodin ylläpitäjät kantavat toki vastuuta oman koodin tietoturvallisuudesta, mutta vahinkojen välttämiseksi tarvitaan vahvaa ylläpitäjien ja käyttäjien välistä kommunikaatiota ja yhteistyötä.

4.1 Avoimen lähdekoodin ylläpitäjät

Avoimen lähdekoodin ylläpitäjille on ensisijaisen tärkeää kommunikoida mahdollisimman avoimesti ja tehokkaasti, jotta tieto havaituista haavoittuvuuksista ja uusista päivityksistä välittyisi avoimen lähdekoodin käyttäjille nopeasti ja vaivattomasti. Kommunikaatiota varten ylläpitäjät voivat kehittää omat käytäntönsä tai ottaa käyttöön jo olemassa olevia käytäntöjä. Hyvä käytäntö on esimerkiksi lisätä repositorioon SECURITY.md tiedosto, jossa lukee ylläpitäjän tietoturvakäytännöt. Repositorioon voi myös asettaa näkyville projektin tietoturvan tasoa kuvaavan merkin. (Snyk, 2019.) Chinthanet et al. (2021) ehdottavat artikkelissaan, että haavoittuvaisuuksia korjaavia muutoksia ei tulisi yhdistää muiden muutosten kanssa samaan julkaisuun. Jos korjaus yhdistetään osaksi suurempaa päivitystä se voi jäädä kehittäjiltä huomaamatta, jolloin kirjaston päivittäminen ei välttämättä vaikuta yhtä tarpeelliselta. Haavoittuvuuksien korjaukset tulisi julkaista esimerkiksi kiireellisissä päivityksissä, jotka ovat erillään suuremmista päivityksistä. Tällöin myös taaksepäin yhteensopivuus säilyisi. (Chinthanet et al., 2021.)

Snykin (2019) kyselyyn vastanneista ylläpitäjistä 48 prosenttia saa tietää heidän koodistaan löytyneestä haavoittuvuudesta julkisten kanavien kautta. Tämä korostaa sitä, kuinka tärkeää koodin tarkistaminen itse ajoittain on. Avoimen lähdekoodin ylläpitäjien tulisi ottaa tietoturva-asiat oleelliseksi osaksi koodin tarkastamista. Myös työntekijöiden tietoturvakoulutuksen parantaminen on tärkeää, jotta koodia tarkastaessaan he tietävät mitä etsiä. Manuaalisen tarkistamisen lisäksi on suositeltavaa käyttää automatisoituja työkaluja osana koodin tarkistusta. (Snyk, 2019.)

Niin kutsuttu shift-left -strategia auttaa vahvistamaan projektien tietoturvaa (Snyk, 2019). Strategialla tarkoitetaan tietoturva-asioiden huomioimista kehityksen alusta lähtien. Perinteisesti tietoturvan testaaminen on jätetty vasta kehityksen loppuvaiheeseen, jolloin ajan tai rahan puutteesta johtuen tietoturvan tavoitteet eivät täyty. (Firesmith, 2015.)

4.2 Avoimen lähdekoodin käyttäjät

Avoimen lähdekoodin käyttäjillä on monia keinoja, joilla arvioida avoimen lähdekoodin kirjastojen turvallisuutta ja luotettavuutta. Snykin (2020) teettämän kyselyn perusteella kehittäjät selvittävät muun muassa seuraavia asioita arvioidessaan kirjastojen luotettavuutta:

- Kuinka aktiivinen yhteisö kirjaston ympärillä on?
- Kuinka usein kirjasto saa uusia päivityksiä?
- Kuinka paljon kirjastoa ladataan kuukausittain?
- Sisältääkö kirjasto tunnistettuja haavoittuvaisuuksia?
- Millainen on kirjaston ylläpitäjän tiedonantopolitiikka?

Kaikki edellä mainitut asiat voivat auttaa tunnistamaan luotettavia kirjastoja, mutta mikään niistä ei yksinään riitä tekemään tarkkaa arviota. Usein julkaistuja päivityksiä pidetään myönteisenä asiana, mutta niilläkin voi olla kielteisiä vaikutuksia kirjaston tietoturvan kannalta. Etenkin tietoturvaan liittyvät usein tapahtuvat päivitykset voivat vaikeuttaa huomattavasti kirjastosta riippuvaisten ohjelmien ylläpitoa. (Snyk, 2020.) On siis tärkeää tarkastella kirjastoja monesta eri näkökulmasta ennen lopullista päätöstä niiden luotettavuudesta.

Avoimen lähdekoodin ylläpitäjien on ilmoitettava korjatuista haavoittuvuuksista, jotta käyttäjät tietäisivät päivittää kirjastonsa. Nämä ilmoitukset kiinnostavat myös rikollisia, sillä ne voivat avata uusia mahdollisuuksia hyökkäyksille. Hyökkääjät ottavat kohteekseen kirjastoja, jotka ovat juuri korjanneet haavoittuvaisuuksia, mikäli hyökkääjillä on syytä epäillä, etteivät käyttäjät ole päivittäneet kirjastoja vielä uusimpaan versioon. Haavoittuvaisuuden korjaamisen jälkeen hyökkäysten määrän voidaan olettaa kasvavan hetkellisesti, kun hyökkääjät pyrkivät hyödyntämään haavoittuvaisuutta ennen kuin käyttäjät ehtivät päivittää kirjaston turvalliseen versioon. (Arora et al., 2006.) Tästä syystä, käyttäjien on äärimmäisen tärkeää seurata käyttämiensä kirjastojen päivityksiä ja päivittää kirjastot uusimpaan versioon välittömästi uuden version tullessa saataville.

Riippuvuuksien ajan tasalla pitämisen vaikeutta on alettu huomioimaan entistä enemmän. Kehittäjille tarjotaan nykyään monia työkaluja, joiden avulla riippuvuuksien pitäminen ajan tasalla on entistä helpompaa. Yksi tällainen työkalu on GitHubin Dependabot. Sen avulla kehittäjä voi automatisoida riippuvuuksien päivittämisen. Dependabotin voi konfiguroida tarkastamaan riippuvuudet niin usein kuin kehittäjä haluaa, ja haavoittuvaisuuden löytäessään se ilmoittaa siitä käyttäjälle ja mikäli mahdollista luo GitHub repositoriossa pull requestin haavoittuvaisen paketin päivittämiseksi korjattuun versioon (Mullans, 2020). Kehittäjät ovat ottaneet tämän ominaisuuden hyvin vastaan, sillä Dependabot on varoittanut käyttäjiä yli 425 miljoonasta mahdollisesta haavoittuvuudesta (Havens, 2022).

NPM:n komentoliittymä tarjoaa komennon *npm audit.* Tämä komennon avulla käyttäjä voi tarkistaa onko asennetuissa paketeissa tunnettuja haavoittuvaisuuksia. Löydetyt haavoittuvaisuudet on jaettu kolmeen kategoriaan sen perusteella, kuinka vakavia ne ovat Kategoriat ovat matala (low), keskisuuri (moderate) tai korkea (high). NPM kertoo löydetyt haavoittuvaisuudet myös pakettien asentamisen yhteydessä, joten siihen on myös syytä kiinnittää huomiota. Antamalla komennon *npm audit fix* NPM pyrkii korjaamaan kaikki löydetyt haavoittuvaisuudet automaattisesti päivittämällä paketteja. On hyvä huomioida, että automaattinen korjaaminen ei kuitenkaan aina ole mahdollista. (NPM Docs, 2022.)

5 Yhteenveto

Avoimen lähdekoodin kirjastojen käyttö on yleinen käytäntö web-ohjelmoinnissa, mutta kirjastojen vahvasta tietoturvasta ei voi aina olla varma. Tutkielmassa selvisi, että kirjastojen tietoturva-aukkojen syynä on usein haavoittuvaisuuksien hidas korjaaminen ja riippuvuuksien hidas päivittäminen. Riippuvuuksien uusien versioiden seuraaminen ja päivittäminen on raskasta ja vaatii kehittäjiltä paljon resursseja. Riippuvuuksien päivittämiseen ja haavoittuvaisuuksien nopeaan korjaamiseen on kuitenkin panostettava, sillä yhden kirjaston haavoittuvaisuudet voivat vaikuttaa moniin muihinkin kirjastoihin, jotka ovat siitä riippuvaisia. Lisäksi haavoittuvaisuudet johtuvat usein epäsuorista riippuvuuksista, mikä tekee niiden havaitsemisesta ja korjaamisesta haastavampaa.

Kirjastoista löytyy yhä samoja haavoittuvaisuuksia, kuin aikaisempinakin vuosina ja jopa vuosikymmenen takaa. Lisäksi yleisimpien haavoittuvaisuuksien taulukosta huomataan, että yksikin suosittu haavoittuvainen kirjasto voi aiheuttaa laajalle levinneen tietoturvariskin, kuten Lodash-kirjastosta löydetty Prototype Pollution -haavoittuvaisuus todistaa. Lodash-kirjaston takia Prototype Pollution oli vuoden 2019 suurin haavoittuvaisuus sen vaikutuksen perusteella, vaikka Cross Site Scripting -haavoittuvuudesta löydettiinkin enemmän erillisiä tapauksia.

Riippuvuuksien päivittämisestä johtuvaa taakkaa voi lievittää käyttämällä erilaisia työkaluja, joilla uusien versioiden tarkistamisen ja päivittämisen voi automatisoida. Yksi näistä työkaluista on esimerkiksi GitHubin tarjoama Dependabot. Myös NPM:n ilmoittamia haavoittuvaisuuksia kannattaa tarkkailla ja hyödyntää *npm audit* -komentoa.

Avoimen lähdekoodin ylläpitäjien parhaisiin käytäntöihin kuuluu avoin ja selkeä viestintä, hyvät tietoturvakäytännöt sekä shift-left -strategian omaksuminen kehitysprosessiin. Avoimen lähdekoodin käyttäjien kannattaa tutustua käyttämiinsä kirjastoihin ja tutkia niitä monesta eri näkökulmasta saadakseen mahdollisimman luotettavan kuvan niiden tietoturvan tasosta.

Lähdeluettelo

Abdalkareem, R. Oda, V., Mujahid, S. & Shihab, E. (2020) On the impact of using trivial packages: an empirical case study on npm and PyPI. Empirical software engineering : an international journal. [Online] 25 (2), 1168–1204.

Antal, G., Keleti, M. & Hegedűs, P. (2020). Exploring the Security Awareness of the Python and JavaScript Open Source Communities. Proceedings of the 17th International Conference on Mining Software Repositories, 16–20.

Arora, A., Nandkumar, A. & Telang, R. (2006). Does information security attack frequency increase with vulnerability disclosure? An empirical analysis. Inf Syst Front 8, 350–362.

Ballhausen, M. (2019) Free and Open Source Software Licenses Explained. *Computer*, vol. 52, no. 06, 82–86.

Chinthanet, B. Kula, R. G., McIntosh, S., Ishio, T., Ihara, A. & Matsumoto, K. (2021) Lags in the release, adoption, and propagation of npm vulnerability fixes. Empirical software engineering : an international journal. [Online] 26 (3).

Decan, Mens, T., & Claes, M. (2017). An empirical comparison of dependency issues in OSS packaging ecosystems. 2017 IEEE 24th International Conference on Software Analysis, Evolution and Reengineering (SANER), 2–12.

Decan, A. Constantiou, E. & Mens, T. (2018a) On the impact of security vulnerabilities in the npm package dependency network. Proceedings of the 15th International Conference on mining software repositories. [Online]. 2018 ACM. 181–191.

Decan, A., Constantiou, E. & Mens, T. (2018b). On the evolution of technical lag in the npm package dependency network. Proceedings of the 15th International Conference on Mining Software Repositories (MSR), 181–191.

Decan, A., De Roover, C., Mens, T. & Zerouali, A. (2022). On the Impact of Security Vulnerabilities in the npm and RubyGems Dependency Networks. arXiv pre-print arXiv:2106.06747.

Firesmith, D., (2015) Four Types of Shift Left Testing. *https://in-sights.sei.cmu.edu/blog/four-types-of-shift-left-testing/.* (Haettu 26.4.20222).

Gustavsson, T. (2020) Managing the Open Source Dependency. Computer, vol. 53, no. 02, 83–87.

Havens, E. (2022). Improving the developer experience for Dependabot alerts. *https://github.blog/2022-02-08-improving-developer-experience-dependabot-alerts/*.

Heartbleed. (2022). The Heartbleed Bug. *https://heartbleed.com/*. (Haettu 19.3.2022).

Kula, R. G., German, D. M., Ouni, A., Ishio, T. & Inoue, K. (2017) Do developers update their library dependencies?: An empirical study on the impact of security advisories on library migration. Empirical software engineering : an international journal. [Online] 23 (1), 384–417.

Kyberturvallisuuskeskus. (2022) Tietoturva. *https://www.kyberturvallisuusk-eskus.fi/fi/toimintamme/saantely-ja-valvonta/tietoturva*. (Haettu 19.3.2022).

Mullans, A. (2020). Keep all your packages up to date with Dependabot. *https://github.blog/2020-06-01-keep-all-your-packages-up-to-date-with-*

dependabot/.

NCSC. (2015). National Cyber Security Centre. Understanding vulnerabilities. https://www.ncsc.gov.uk/information/understanding-vulnerabilities. (Haettu 21.3.2022).

Netcraft. (2014). April 2014 Web Server Survey. *https://news.netcraft.com/archives/2014/04/02/april-2014-web-server-survey.html*. (Haettu 19.3.2022).

NPM. (2022). NPM Website. *https://www.npmjs.com/*. (Haettu 21.3.2022).

NPM Docs. (2022). npm-audit. *https://docs.npmjs.com/cli/v8/commands/npm-audit.* (Haettu 24.4.2022).

Owasp. (2022). Cross Site Scripting (XSS). *https://owasp.org/www-community/attacks/xss/.* (Haettu 15.4.2022).

Snyk. (2022). Prototype pollution. *https://learn.snyk.io/lessons/prototype-pollution/javascript/*. (Haettu 22.4.2022).

Snyk. (2019). The State of Open Source Security Report. *https://snyk.io/wp-content/uploads/sooss_report_v2.pdf.* (Haettu 24.4.2022).

Snyk. (2020). The State of Open Source Security Report. https://go.snyk.io/SoOSS-Report-2020.html. (Haettu 15.4.2022).