Osaamista
ja oivallusta
tulevaisuuden
tekemiseen

Mikko Jämiä

Tietoturvaongelmien havaitseminen Node.js-ympäristöissä

Metropolia Ammattikorkeakoulu

Insinööri (AMK)

Tieto- ja viestintätekniikka

Insinöörityö

25.4.2019

Tekijä Otsikko Sivumäärä Aika	Mikko Jämiä Tietoturvaongelmien havaitseminen Node.js-ympäristöissä 32 sivua 25.4.2019
Tutkinto	Insinööri (AMK)
Tutkinto-ohjelma	Tieto- ja viestintätekniikka
Ammatillinen pääaine	Mobile Solutions
Ohjaajat	Yliopettaja Jarkko Vuori Senior Developer, Solution Architect Oskari Okko Ojala

Insinöörityön tarkoituksena oli tutkia toimintatapoja haavoittuvuuksien havaitsemiseen ja korjaamiseen Node.js-ympäristöissä. Insinöörityö tehtiin suomalaiselle yritykselle, joka tarjoaa asiakkailleen sovelluskehitystä, suunnittelua ja konsultointia IT-alalta. Asiakas halusi kartoittaa kaikkien aktiivisten Node.js-ympäristöjä käyttävien sovellusprojektiensa haavoittuvuudet. Analyysia varten tuli kaikkien organisaation sovellusprojektien ympäristöä kuvaavat tiedostot kopioida yhden hakemiston alle Github-versionhallintajärjestelmässä. Lopullinen analyysi suoritettaisiin versionhallinnassa käyttäen asianmukaisia työkaluja. Analyysin tarkoituksena oli kartoittaa haavoittuvuuksien nykyinen tilanne ja parantaa sovellusprojektien tietoturvaa tulevaisuudessa.

Ensin työssä vertailtiin neljää haavoittuvuuksien havaitsemiseen ja korjaamiseen tarkoitettua työkalua. Seuraavaksi työssä selvitettiin, kuinka kaikkien projektien Node.js-ympäristöä kuvaavat tiedostot saataisiin kopioitua yhteen hakemistoon. Haluttuja tuloksia saatiin ajamalla komentorivillä komentosarjoja, jotka kopioivat tiedostot paikallisesti uuteen hakemistoon. Ratkaisun heikkoudeksi osoittautui sen paikallisuus, jolloin hakemistojen tuli olla valmiiksi kopioituna käyttäjän työasemalle. Ongelman ratkaisemiseksi päädyttiin kehittämään Node.js-sovellus, joka teki kutsuja Githubin ohjelmointirajapintaan. Rajapinnasta saadun datan pohjalta sovellus loi tiedostot uudestaan uuden hakemiston alle.

Tiedostot kerättiin uuteen hakemistoon, joka siirrettiin versionhallintaan. Versionhallinnassa hakemistoon integroitiin Renovate Bot -työkalu, joka on automaattiseen pakettien päivittämiseen tarkoitettu sovellus. Työkalua voidaan myös käyttää haavoittuvuuksien havaitsemiseen eri ohjelmointiympäristöissä. Renovate Bot avasi analyysin pohjalta Github-hakemistoon uusia vertaisarviointitikettejä, joista kävivät ilmi vanhentuneet ja haavoittuneet paketit.

Valmiin analyysin tulokset ilmoitettiin suoraan asiakkaalle. Projektin hakemisto ja analyysin tulokset jaettiin myös yrityksen kehittäjille. Kehittäjiä kehotettiin korjaamaan ilmi tulleet haavoittuvuudet ja suositeltiin integroimaan Renovate Bot kaikkiin yrityksen Github-organisaation hakemistoihin.

Avainsanat	Node.js, haavoittuvuudet, tietoturva

Author Title Number of Pages Date	Mikko Jämiä Detecting Vulnerabilities in Node.js Environments 32 pages 25 April 2019
Degree	Bachelor of Engineering
Degree Programme	Information and Communication Technology
Professional Major	Mobile Solutions
Instructors	Jarkko Vuori, Principal Lecturer Oskari Okko Ojala, Senior Developer, Solution Architect

The purpose of the thesis was to research methods for detecting and fixing vulnerabilities in Node.js environments. The thesis project was done for a Finnish limited company who offers their customers application development, design and consultation services. The customer wanted to expose known vulnerabilities in all of their active Node.js-based development projects. For the analysis, all Node.js environment files from the projects had to be copied into a single repository in Github version control system. The analysis would then be done in the repository by using appropriate tools. Goal of this project was to get to know the current situation of vulnerabilities in the projects and to aim for more secure software development.

Firstly, four tools for detecting and fixing vulnerabilities were tested and compared. The second task was to examine how to copy all Node.js environment files into the new repository. Results were achieved by using series of commands executed on the command line, which copied all the files into the new repository locally. The issue with this approach was that all the repositories had to be already cloned to the user's workstation. The problem was solved by developing a Node.js application, which made requests directly to Github with the help of their application programming interface. With the data received from the interface, the application was able to reconstruct the files into the new repository.

The repository was then moved to version control. The repository was integrated with Renovate Bot, which is an application used for keeping packages updated. However, the application can also be used for detecting vulnerabilities in software environments. From the analysis Renovate opened several new pull requests to Github which revealed all the outdated and vulnerable packages.

Results from analysis were informed directly to the customer. The results and the repository were also shared with all the company's developers. Developers were urged to fix all the vulnerabilities and recommended to integrate Renovate Bot to all of the organization's repositories.

Keywords	Node.js, vulnerabilities, security

Sisällys

Lyhenteet

Node.js	Javascript-pohjainen runtime-ympäristö.
npm	Node Package Manager, oletuspaketinhallintajärjestelmä Node.js runtime-ympäristössä.
JSON	JavaScript Object Notation, avoimen standardin tiedostomuoto.
NVD	National Vulnerability Database, Yhdysvaltain hallituksen haavoittuvuustietokanta.
CVE	Common Vulnerabilities and Exposures, ohjelma, jonka tarkoitus on kerätä ja jakaa tietoa haavoittuvuuksista.

1 Johdanto

Insinöörityössä tutkittiin toimintatapoja haavoittuvuuksien havaitsemiseen ja niiden korjaamiseen Node.js-ympäristöissä. Työssä perehdyttiin erityisesti npm-paketinhallintajärjestelmään, pakettien riippuvuuksiin, tietoturvaan, DevOps-toimintamalliin ja versionhallintaan. Tämän lisäksi työssä vertailtiin neljää eri työkalua, joita voidaan käyttää haavoittuvuuksien havaitsemiseen ja korjaamiseen Node.js-ympäristöissä. Insinöörityö toteutettiin Frantic Oy:lle. Frantic on vuonna 1997 perustettu suomalainen yritys [1]. Yrityksellä on yli 100 työntekijää ja sillä on toimipisteet Helsingissä ja Tampereella [2].

Avoimen lähdekoodin paketit ovat merkittävä osa nykyaikaista sovelluskehitystä. Paketit voivat auttaa kehittäjiä merkittävästi tuottavuudessa ja ajankäytössä osana sovelluskehitystä. Yksi suosituimmista paketinhallintajärjestelmistä on npm, jonka rekisteristä löytyy yli 600 000 avoimen lähdekoodin Javascript-pakettia. Ulkoisten kehittäjien luomia paketteja on komentorivityökalun avulla helppo asentaa projekteihin tuomaan ominaisuuksia, joiden kehittämiseen omatoimisesti kuluisi tavallisesti paljon aikaa.

Node.js-pakettien kasvattaessaan suosiotaan on niiden turvallisuus noussut suureksi puheenaiheeksi. Paketit voivat sisältää haavoittuvuuksia, jotka voivat olla korjaamatta jätettyinä vakava tietoturvariski. Ongelman ratkaistakseen kehittäjät voivat hyödyntää erilaisia työkaluja, joilla pakettien haavoittuvuuksia voidaan havaita ja korjata.

Osana insinöörityötä tehtiin projekti, jossa suoritettiin analyysi kaikista sellaisista Franticin sovellusprojekteista Github-versionhallintajärjestelmässä, jotka käyttävät Node.js-ympäristöjä. Analyysissa kartoitettiin Franticin Node.js-pohjaisten projektien haavoittuvuuksien nykyinen tilanne ja pyrittiin sen myötä tehostamaan projektien tietoturvaa tulevaisuudessa. Tutkimustyöstä saatua informaatiota ja työkaluja käytettiin apuna tehtävän suorittamisessa.

2 Npm-paketinhallintajärjestelmä

Node package manager, joka tunnetaan paremmin lyhenteenä npm, on paketinhallintajärjestelmä Node.js runtime -ympäristössä. Paketinhallintajärjestelmän tarkoituksena on

helpottaa Javascript-kehittäjien työskentelyä tarjoamalla helppokäyttöinen alusta avoimen lähdekoodin paketeille. Npm-termillä on paketinhallintajärjestelmän lisäksi muitakin merkityksiä: npm-rekisteri, komentorivityökalu ja npm, Inc. Npm-rekisteri on julkinen alusta, jossa avoimen lähdekoodin paketit ovat saatavilla. Npm-komentorivityökalu mahdollistaa pakettien asentamisen ja julkaisemisen kehittäjien toimesta komentoriviä käyttäen. Npm, Inc on yritys, joka ylläpitää npm-työkaluja ja rekisteriä. Yritys perustettiin vuonna 2014, ja alkuperäinen kehittäjä Isaac Z. Sclueter on sen toimitusjohtaja. [3.]

Node.js on suosittu Javascript-pohjainen runtime-ympäristö, jota käytetään pääasiassa palvelinsovellusten ja rajapintojen rakentamiseen web-kehityksessä. Node.js suorittaa prosesseja yksittäisessä säikeessä, toisin kuin monet muut palvelinkielet, jotka suorittavat prosesseja useassa säikeessä. Prosessien ajamista yksittäisessä säikeessä pidetään kuitenkin yleisesti tehokkaampana. Node.js:n yksittäinen säie on tapahtumasilmukka, josta pyyntöjä lähetetään funktioiden käsiteltäviksi. Tapahtumasilmukkaa käyttämällä Node.js-palvelimet eivät tarvitse jatkuvasti auki olevaa yhteyttä käsitelläkseen käyttäjän pyyntöjä. [4.] Node.js:n prosessointikyky onkin tehnyt siitä suositun työkalun palvelinkehityksessä. Node.js-sovellusten kehittäminen vaatii kehittäjiltä pääasiassa vain Javascript-osaamista, joten sen oppii nopeasti.

Vaikka Node.js on monien kehittäjien suosiossa, sillä on kuitenkin myös huonoja puolia. Suurimpana ongelmana voidaan mainita Node.js-sovelluksissa yleisesti käytettyjen avoimen lähdekoodin Javascript-pakettien ylläpitovastuun puuttuminen. Koska paketit ovat pääasiassa ulkopuolisten osapuolien kehittämiä, ne ovat usein huonosti ylläpidettyjä ja dokumentoituja. Ulkopuolisten kehittäjien lähdekoodit eivät aina seuraa hyvän ohjelmoinnin käytäntöjä, joten ne voivat olla laadultaan puutteellisia. Avoimen lähdekoodin paketteja käyttäessään kehittäjät avaavat myös oman sovelluksensa haavoittuvuuksille, jotka voivat altistaa sen erilaisille hyökkäyksille. Suuri haaste Node.js:n kanssa sovelluksia kehittäessä onkin löytää käytettäväksi luotettavia Javascript-paketteja. [5.]

3 DevOps-toimintamalli

DevOps-toimintamalli sai alkunsa vuosituhannen alussa. Toimintamallista on olemassa useita eri tulkintoja, mutta sen keskeisenä periaatteena on yhdistää kehittäminen ja ope-

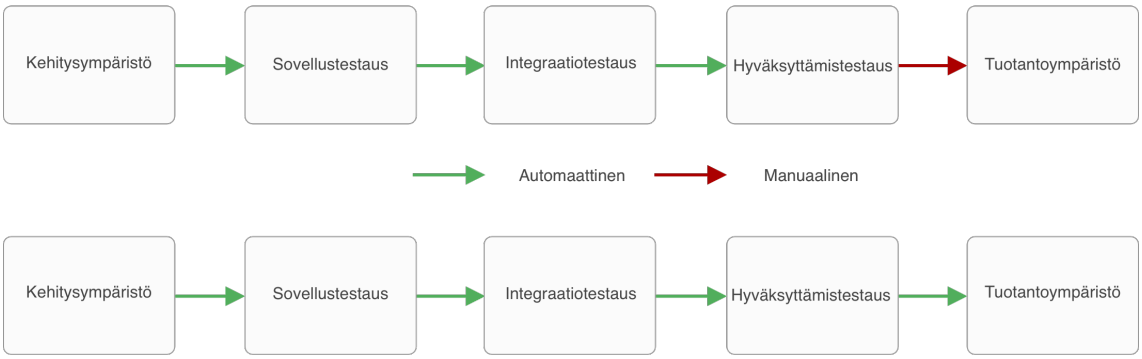
raatiot sovelluskehityksessä. Kehittämisellä tarkoitetaan yleisesti ohjelmointia, testaamista ja laaduntarkkailua. Operaatioilla voidaan tarkoittaa esimerkiksi sovellusten julkaisemista sekä järjestelmä- ja tietokantahallintaa. [6, s. 4.] Kehittämisen ”Development” ja operaatioiden ”Operations” yhdistelmästä toimintamalli on myös saanut nimensä. Toimintamalli ilmaisee tapoja, joilla voidaan tehostaa ohjelmistojen toimittamista kehitysympäristöistä tuotantoympäristöihin. Mallin tarkoituksena on myös korostaa osapuolien henkilökohtaisen osaamisen kehittämistä osana prosessia, jolloin työskentelystä saadaan entistä sujuvampaa ja laadukkaampaa. [6, s. 4.]

3.1 Jatkuvat operaatiot

Yksi DevOps-toimintamalliin vahvasti liittyvistä konsepteista on jatkuvat operaatiot. Toimintamalliin voidaan liittää ainakin seuraavat operaatiot: jatkuva integroiminen, jatkuva toimittaminen, jatkuva julkaiseminen ja jatkuva testaaminen. [7.]

Jatkuvalla integroimisella tarkoitetaan, että sovelluksen kehitysversioista muodostetaan uusi julkaisuvalmis versio mahdollisimman usein [7]. Käytännössä jatkuva integroiminen voi tarkoittaa esimerkiksi sitä, että versionhallinnan avulla kehittäjät tuovat sovellukseen uusia ominaisuuksia tai korjauksia jatkuvasti. Integroiminen ei kuitenkaan tarkoita sovelluksen julkaisemista tuotantoon säännöllisesti, vaan periaatteena on jatkuvasti rakentaa sovellusta kokonaisuutena ennen julkaisua. Osana jatkuvaa integrointia on myös järkevää ajaa automatisoituja testaus- ja analysointiprosesseja. Testien ja analysoinnin avulla voidaan varmistaa sovelluksen laadukkuus ja toimivuus. Testeillä ja analysoinnilla voidaan tarkoittaa esimerkiksi haavoittuneiden Node.js-pakettien etsimistä ja niiden korjaamista.

Jatkuvasta toimittamisesta ja julkaisemisesta puhutaan usein kahtena eri prosessina, vaikka niillä ei ole juurikaan eroa. Molemmat prosessit alkavat kehitysympäristöissä tehdyistä muutoksista testaamisen kautta tuotantoon. Erona näillä kahdella prosessilla on kuitenkin se, että jatkuva julkaiseminen on täysin automatisoitu prosessi, kun taas jatkuvassa toimittamisessa julkaisuvalmis versio julkaistaan manuaalisesti testauksen jälkeen tuotantoon. [7.] Kuvassa 1 on kuvattu jatkuvan toimittamisen ja julkaisemisen ero.



Kuva 1. Jatkuvan toimittamisen ja julkaisemisen eroavaisuus.

Hyötynä jatkuvassa toimittamisessa on, että kehittäjillä on mahdollisuus perusteellisesti tarkistaa sovellus kokonaisuudessaan manuaalisesti ennen julkaisua. Manuaalisesti tarkistamalla voidaan minimoida virheitä ja tietoturvariskejä, jotka eivät ole testausvaiheessa tulleet ilmi. Esimerkiksi sovelluksen käyttöliittymän käytettävyyttä on vaikeaa testata automaattisilla testeillä, jolloin testaaminen kehittäjien omasta toimesta on paljon luotettavampaa. [7.] Tämä kuitenkin lisää kehittäjien työmäärää entisestään, jolloin jatkuva julkaiseminen voi olla parempi vaihtoehto.

Testaamisen konsepti esiintyy tavalla tai toisella kaikissa edellä mainituissa operaatioissa. Testaaminen onkin välttämätön osa DevOps-toimintamallia. Jatkuva testaaminen antaa kehittäjille jatkuvasti tietoa sovelluksesta kokonaisuutena [7]. Kun testaamista suoritetaan jo sovelluksen kehittämisen alussa ja sen aikana, voidaan myöhemmin välttyä kustannuksia kasvattavilta virheiltä. Testaamisen ei tarvitse tarkoittaa pelkästään ohjelmointiin liittyviä yksikkötestejä, vaan se voidaan myös tulkita käyttäjäkokemukseen liittyviksi käyttäjätesteiksi.

3.2 Versionhallinta

Versionhallinta on välttämätön osa nykyaikaista sovelluskehitystä. Versionhallinnalla sovelluskehityksessä tarkoitetaan järjestelmää, joka seuraa yhden tai useamman tiedoston muutoksia tietyllä aikavälillä. Näistä muutoksista järjestelmä luo versioita, joihin kehittäjät voivat tarvittaessa palata. [8.]

Aluksi versionhallintaa suoritettiin kehittäjien omilla työasemilla paikallisesti. Kehittäjillä oli yksi tai useampia kopioita koko sovelluksesta työasemillaan. Versiot saatettiin nimetä päivämäärän mukaan, jotta eri versiot löytyisivät helposti. Tämä työskentelytapa kuitenkin osoittautui hyvin virhealttiiksi ja epäkäytännölliseksi. Ratkaisuna kehittäjät hyödynsivät paikallisia tietokantoja, jonne pystyttiin tallentamaan eri versioita sovelluksista. Tämä ratkaisu auttoi helpottamaan yksittäisen kehittäjän versionhallintaa. Suurin ongelma toimintamallissa oli kuitenkin sen kykenemättömyys jakaa versioita sovelluksesta helposti muiden kehittäjien kanssa. [8.]

Seuraavana askeleena parempaan versionhallintaan oli siirtyminen keskitettyihin versionhallintajärjestelmiin, joissa sovelluksen koko versionhallinta tapahtui yksittäisellä palvelimella. Tässä ratkaisumallissa kehittäjät pystyivät helpommin jakamaan versioita keskenään ja seuraamaan muiden kehittäjien työskentelyä lähdekoodissa. Suurimmaksi ongelmaksi keskitetyssä versionhallinnassa osoittautui sen yksittäinen palvelin. Jos palvelimella sattui toimintahäiriö, kehittäjät eivät pystyneet tallentamaan tai lataamaan uutta versiota sovelluksesta. Jos data tietokannassa korruptoitui, sen palauttaminen oli vaikeaa ilman kunnollisia varmuuskopioita. [8.]

Viimeisimpänä ratkaisuna versionhallintaan on otettu käyttöön hajautetut versionhallintajärjestelmät. Tällaiset versionhallintajärjestelmät, kuten Github ja Mercurial, ovat nykyään monien kehittäjien suosiossa. Hajautetut versionhallintajärjestelmät korjasivat keskitettyjen järjestelmien puutteita hajauttamalla koko versiohistorian kehittäjien kesken. Jos palvelimella sattuu toimintahäiriö tai dataa katoaa, kehittäjät voivat helposti palauttaa viimeisimmän version sovelluksesta palvelimelle. [8.]

Hajautetut versionhallintajärjestelmät ovat tehokas työkalu osana DevOps-toimintamallia. Github on nykyaikaisista versionhallintajärjestelmistä suosituin. Siinä kehittäjien on mahdollista luoda sovelluksesta oma haaransa. Haaralla versionhallinnassa tarkoitetaan kopiota sovelluksen viimeisimmästä julkaisuvalmiista versiosta, jossa kehittäjät voivat vapaasti työskennellä uusien ominaisuuksien parissa ja lopulta yhdistää ne takaisin julkaisuvalmiiseen versioon. Kehittäjät voivat luoda omasta haarastaan ennen master-versioon yhdistämistä vertaisarvioinnin, jossa muiden kehittäjien on mahdollista arvioida muutoksia. Vertaisarvioinnit edistävät sovellusten laaduntarkkailua ja ovat myös tehokas apuväline kehittäjien oman ja yhteisen osaamisen kehittämisessä. [9.]

Github tukee myös useita työkaluja tukemaan jatkuvaa integrointia, kuten Travis CI. Travis CI mahdollistaa yksikkötestien rakentamisen suoraan versionhallintaan. Yksikkötestien lisäksi Travis CI mahdollistaa myös julkaisuvalmiin version julkaisemisen palvelimelle, joten sitä voidaan käyttää myös jatkuvaan julkaisuun. [10.]

4 Haavoittuvuudet Node.js-ympäristöissä

Haavoittuvuudet määritetään sovelluskehityksessä ohjelmiston tai laitteiston koodin heikkouksina, jotka vaikuttavat negatiivisesti sen tietosuojaan, eheyteen tai saatavuuteen [11]. Tietoturvahaavoittuvuudet ovat suuri ongelma avoimen lähdekoodin Node.js-paketeissa. Haavoittuneet paketit voivat pahimmassa tapauksessa avata sovelluksen erilaisille hyökkäyksille. Pakettien haavoittuvuudet voivat myös vaikuttaa niistä riippuvaisten pakettien turvallisuuteen. Jopa neljännes Node.js-ympäristöissä käytetyistä paketeista voi sisältää tunnettuja haavoittuvuuksia. [12, s. 181.]

Jokainen Node.js-paketti sisältää JSON-tiedoston, josta käyvät ilmi paketin nimi, versio ja sen riippuvuudet muihin paketteihin. Node.js-paketit käyttävät semanttista versionhallintaa, joka on yksinkertainen ohjesääntö paketin version ilmaisuun. Node.js-pakettien versiot ilmaistaan kolmen numeron formaatissa: Major.Minor.Patch. Ensimmäinen numero ilmaisee suuria (Major) muutoksia paketin lähdekoodissa. Tällaiset päivitykset eivät usein ole taaksepäin yhteensopivia, jolloin paketin käyttäjät saattavat joutua tekemään muutoksia omassa lähdekoodissaan päivityksen yhteydessä. Formaatin toinen numero ilmaisee pienempiä (Minor) muutoksia, jotka ovat taaksepäin yhteensopivia. Tällaisia muutoksia voivat olla esimerkiksi sellaisten ominaisuuksien lisääminen pakettiin, jotka eivät vaikuta paketin normaaliin käyttäytymiseen. Viimeinen numero ilmaisee pieniä (Patch) muutoksia, jotka eivät muuta paketin normaalia käyttäytymistä eivätkä myöskään tuo pakettiin uusia ominaisuuksia. Tällaisia muutoksia voivat olla esimerkiksi ohjelmointivirheiden korjaukset. [12, s. 182.]

Kehittäjien on mahdollista rajoittaa, mitä versioita paketista he haluavat sovelluksessaan käyttää. Tyypillinen tapa ilmaista rajoituksia on asettaa paketille minimi- tai maksimiversionumerot, joita suurempia tai pienempiä versioita paketista ei ole luvallista käyttää. Rajoitus voidaan ilmaista esimerkiksi muodossa >= 1.2.3 tai < 1.3.0. Rajoituksia voidaan myös määrittää yksityiskohtaisemmin estämään tietynlaisia päivityksiä. Kehittäjät voivat

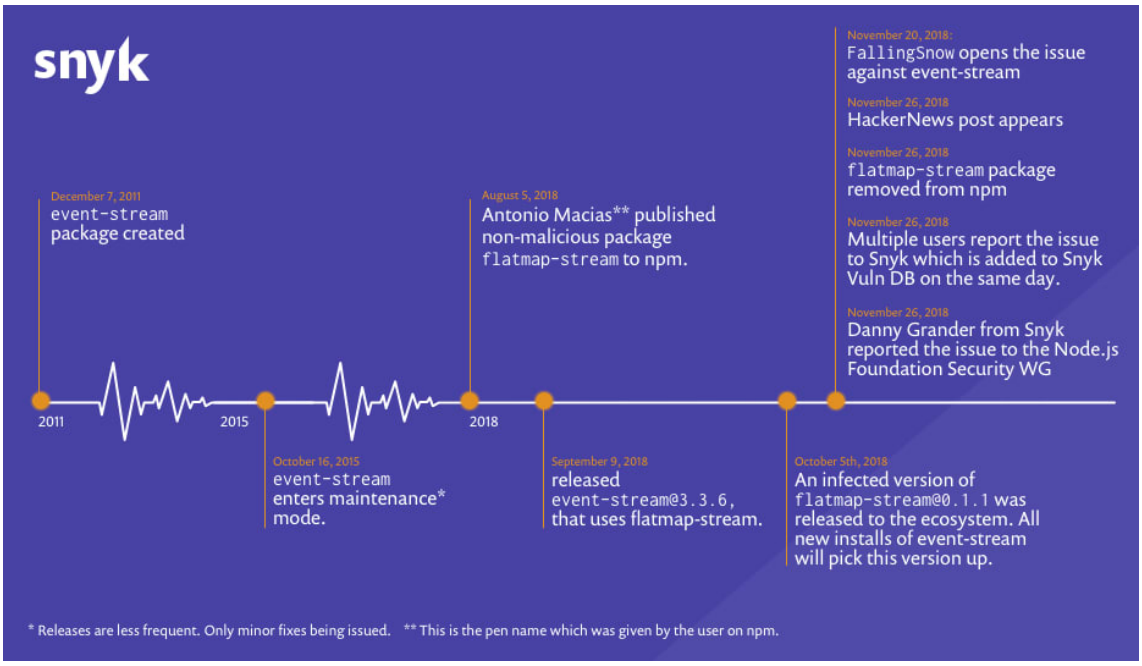
esimerkiksi asettaa paketin versionumerolle rajoituksen, jossa taaksepäin yhteensopimattomia päivityksiä ei sallita. [12, s. 182.]

Niin kauan, kuin paketit ovat haavoittuneita, ne voivat aiheuttaa tietoturvaongelmia sovelluksessa. Haavoittuvuuksien poistaminen paketeista voi kestää pahimmassa tapauksessa jopa vuosia, riippuen haavoittuvuuksien vakavuudesta ja suuruudesta. Tämän lisäksi paketit ovat saattaneet olla haavoittuneita jo kauan ennen, kuin haavoittuvuudet ovat tulleet ilmi. Tuona aikana paketeista on myös todennäköisesti julkaistu useita eri versioita. Tämä tarkoittaa, että pakettien käyttäjien ei aina ole mahdollista palata pakettien vanhempiin versioihin. Pakettien ensimmäisestä julkaisusta voi kulua kaksikin vuotta, ennen kuin puolet niiden haavoittuvuuksista on käynyt ilmi. Vakavuudeltaan korkean ja keskitason suuruiset haavoittuvuudet tulevat yleensä ilmi ennen matalan riskitason haavoittuvuuksia. Suurin osa haavoittuvuuksista saadaan myös korjattua, ennen kuin ne ovat julkisesti käyneet ilmi. Tämä johtuu usein siitä, että pakettien kehittäjät eivät halua tuoda haavoittuvuuksia julkisuuteen. [12, s. 184–186.]

Jotta haavoittuvuuksien aiheuttamia riskejä voidaan minimoida, on äärimmäisen tärkeää korjata haavoittuvuudet mahdollisimman nopeasti ongelman havaitsemisesta. Nopean korjaamisen lisäksi on myös tärkeää analysoida paketteja säännöllisesti, jotta haavoittuvuudet havaittaisiin mahdollisimman nopeasti. [12, s. 187.]

Event-stream

Syksyllä 2018 Node.js-paketti event-stream aiheutti kohua Copay-nimistä kryptovaluutta-alustaa kohtaan tapahtuneiden hyökkäysten takia. Event-stream on todella suosittu Node.js-paketti, jota ladataan viikoittain keskimäärin kaksi miljoonaa kertaa. [13.] Node.js-paketin huomattiin sisältävän siitä riippuvaisen paketin flatmap-streamin, jonka pakattu lähdekoodi varasti Copay-käyttäjien henkilökohtaisia tietoja. Haitallinen paketti ehti olla käytössä yli kaksi kuukautta, ennen kuin se huomattiin ja raportoitiin npm-tietoturvaryhmälle. Pian raportoinnin jälkeen npm poisti flatmap-stream-paketin rekisteristään ja otti event-stream-paketin omistukseensa ehkäistäkseen lisävahinkoja. [14.] Kuva 2 havainnollistaa tapahtumien kulkua.



Kuva 2. Event-stream-tapauksen tapahtumien kulku [14].

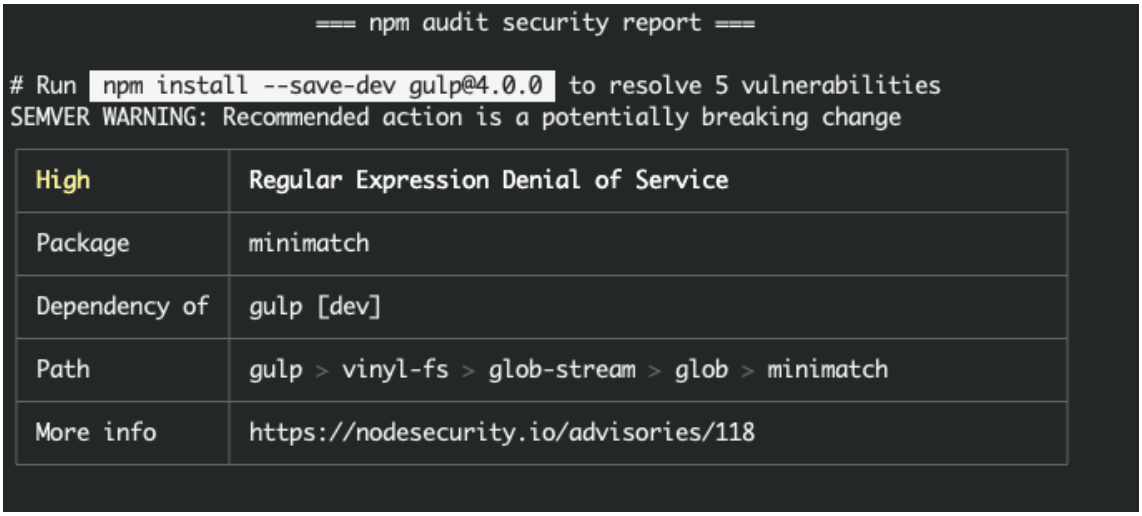
Vaikka event-stream on hyvin suosittu paketti, sen alkuperäinen kehittäjä ei ollut ylläpitänyt sitä aktiivisesti. Hyökkääjä oli ottanut alkuperäiseen kehittäjään yhteyttä sähköpostitse ja tarjoutunut ylläpitämään pakettia tämän puolesta. Omistajuuden vaihtaminen on hyvin tavanomaista avoimen lähdekoodin kehityksessä, kun alkuperäinen kehittäjä ei enää itse tahdo ylläpitää pakettia. Omistajuuden vaihduttua hyökkääjä lisäsi haitallisen flatmap-stream-paketin riippuvuudeksi event-streamille. Flatmap-streamin lähdekoodia oli tarkoituksellisesti sotkettu, jotta haitallista osuutta olisi mahdollisimman vaikea huomata. [13.] Hyökkäys kohdistui Copay-sovelluksen käyttäjiin, joiden kryptovaluutan saldo oli riittävällä tasolla. Haitallinen koodi varasti käyttäjän henkilökohtaiset tiedot ja lähetti datan ulkoiselle palvelimelle. [14.]

Event-streamin tapaus on todella äärimmäinen esimerkki siitä, millaisia pakettien kautta tapahtuvat hyökkäykset voivat olla. Tapaus osoittaa, kuinka haavoittuvainen avoimen lähdekoodin toimintamalli voi olla, jos sen sääntöjä ei noudateta. Suurin syy tapaukseen oli sen alkuperäisen kehittäjän vastuuttomuus, kun hän antoi vieraalle henkilölle täydet oikeudet paketin ylläpitoon. Event-streamin kaltaisilta tapauksilta voitaisiin välttyä, jos suosittujen pakettien ylläpidosta vastaisi osittain myös niiden käyttäjäkunta. [15.]

5 Haavoittuvuuksien havaitseminen ja korjaaminen

Npm audit

Npm esitteli komentorivityökalunsa kuudennessa versiossa npm audit -komennon. Komentoa käytetään analysoimaan projektin ohjelmointiympäristön haavoittuvuuksia tutkien sen package.json- ja package-lock.json-tiedostoja. Package.json-tiedosto listaa sovelluksessa käytetyt Node.js-paketit ja niiden käytössä olevat versiot. Package-lock.json on package.json-tiedostosta generoitu tiedosto, joka ilmaisee tarkan kuvauksen projektin koko Node.js-ympäristöstä. Npm audit analysoi package.json-tiedostossa riippuvuudeksi listatut paketit ja vertaa niitä haavoittuvuustietokantaa vasten. Tämän jälkeen komentorivityökalu tulostaa raportin mahdollisista haavoittuneista Node.js-paketeista. Kuvassa 3 on esimerkki työkalun tulostamasta raportista.



Kuva 3. Npm audit -raportin esimerkki yksittäisestä Node.js-paketista [16].

Työkalu liittää raporttiin komennon, jolla haavoittunut paketti voidaan päivittää uuteen versioon. Käyttäjä voi halutessaan päivittää kaikki haavoittuneet paketit kerralla ajamalla komentorivillä npm audit fix -komennon. Raportti saattaa ilmoittaa päivityksen sisältävän taaksepäin yhteensopimattomia muutoksia, jolloin fix-komennon ajaminen ei ole mahdollista. Tämä johtuu siitä, että päivittäminen todennäköisesti muuttaa pakettien normaalia käyttäytymistä, mikä saattaa rikkoa sovelluksen toiminnallisuutta. Käyttäjät voivat pakottaa automaattisen päivityksen lisäämällä liitteen ”--force” komennon perään, mikä ei ole suositeltavaa.

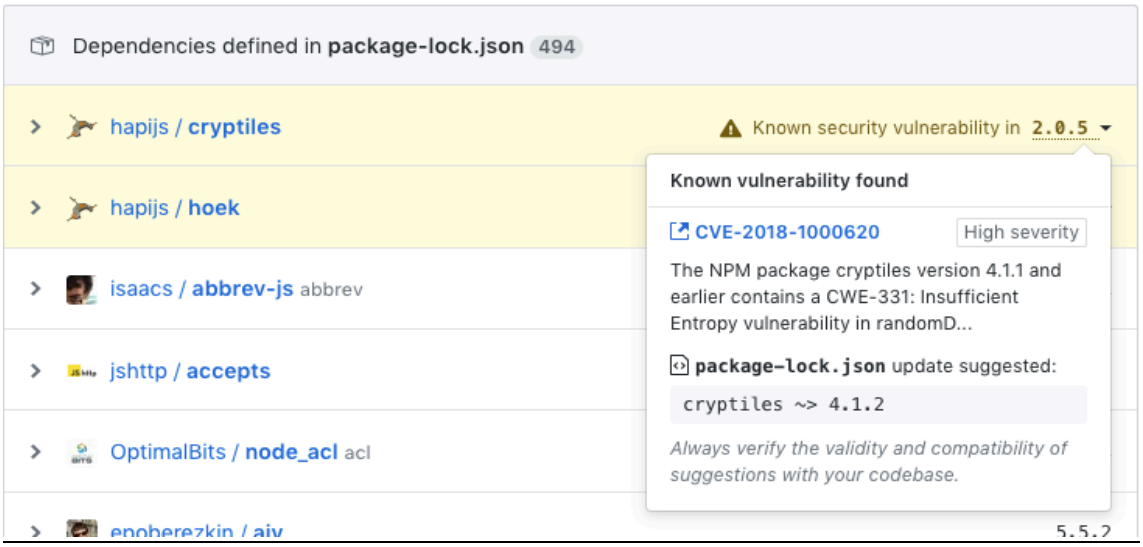
Keväällä 2018 ^Lift Security ja koko sen kehittämä Node Security Platorm liitettiin osaksi npm, Inc. -organisaatiota, npm-komentorivityökalua ja rekisteriä. Node Security -alustaan kuuluu myös komentorivityökalu nsp, jolla on hyvin samanlaisia ominaisuuksia kuin npm auditilla. Nsp päihittää npm auditin kuitenkin kattavammalla haavoittuvuustietokannalla. Tämän muutoksen myötä myös npm audit -komento vertaa paketteja Node Security -alustan haavoittuvuustietokantaan. [17.] Muutoksen myötä nsp-komentorivityökalua pidetään nykyisin vanhentuneena, koska npm audit on virallisesti korvannut sen.

Githubin tietoturvatyökalut

Vuonna 2017 Github ilmoitti, että sen järjestelmässä voidaan jatkossa havaita haavoittuvuuksia riippuvuuskaaviota ja tietoturvahälytyksiä käyttäen. Työkalut vertaavat sovellusprojekteissa käytettyjä paketteja National Vulnerability -haavoittuvuustietokantaa vasten. [18.] National Vulnerability -tietokanta on Yhdysvaltojen hallituksen kehittämä hakemisto, joka koostuu Common Vulnerabilities and Exposures -ohjelman toimittamasta informaatiosta. National Vulnerability -tietokannasta ja Common Vulnerabilities and Exposures -ohjelmasta puhutaan yleensä lyhenteinä NVD ja CVE. CVE-ohjelma sai alkunsa vuonna 1999, ja se on yhdysvaltalaisen voittoa tavoittelemattoman yrityksen MITRE:n kehittämä. MITRE operoi tutkimus- ja kehittämiskeskuksia, joiden tarkoituksena on tunnistaa ja kerätä haavoittuvuuksia ohjelmistoista. CVE ei varsinaisesti ole haavoittuvuustietokanta, vaan sen tarkoituksena on jakaa tietoa haavoittuvuuksista eteenpäin muille osapuolille. CVE määrittää haavoittuvuuksille oman tunnistenumeron, statuksen, lyhyen kuvauksen ja informaation lähdeviitteet. NVD tarjoaa CVE:n datalle tietokannan ja lisätietoja haavoittuvuuksista. [19.]

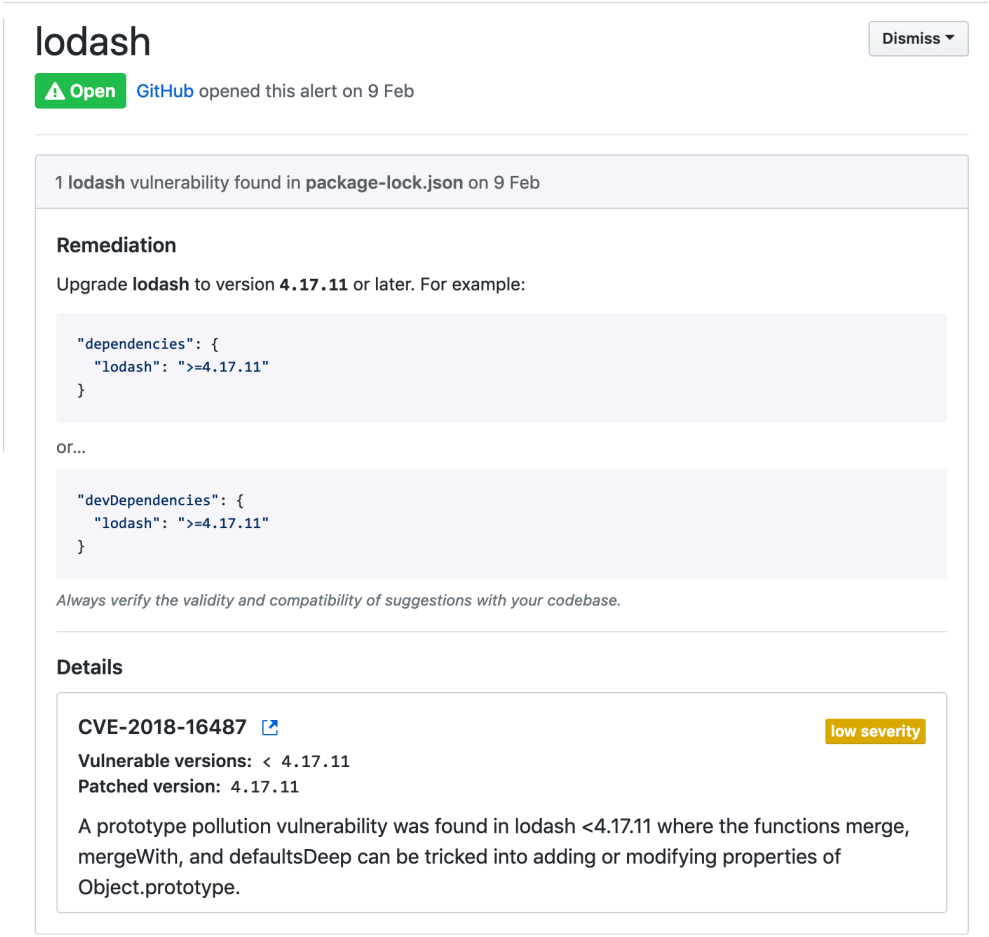
Julkisissa hakemistoissa Githubin tietoturvatyökalut ovat oletuksena päällä. Yksityisissä hakemistoissa käyttäjien täytyy erikseen asettaa työkalut päälle hakemiston asetuksista. Asettamalla tietoturvatyökalut päälle käyttäjä antaa myös Githubille luvan yksityisen hakemiston lukemiseen. Työkaluista voidaan kytkeä molemmat työkalut päälle erikseen tai samanaikaisesti. Riippuvuuskaavio etsii haavoittuvuuksia kaikista hakemiston eri riippuvuuksista, mukaan lukien hakemiston alakansioista ja muista linkitetyistä hakemistoista. Tietoturvahälytykset reagoivat vain hakemiston juuressa määritettyihin riippuvuuksiin. Parhaan hyödyn käyttäjä saakin, kun molemmat työkalut ovat kytkettyinä päälle samanaikaisesti.

Kun riippuvuuskaavio on kytketty päälle projektissa, saavat hakemiston ylläpitäjät sähköpostitse ilmoituksia mahdollisista haavoittuvuuksista. Työkalu pystyy myös ehdottamaan päivityksiä haavoittuneiden versioiden tilalle perustuen Githubin omasta kehittäjäyhteisöstä löytyviin ratkaisuihin. [18.] Kuvasta 4 käy ilmi, kuinka riippuvuuskaavio ilmaisee tunnettuja haavoittuvuuksia hakemistossa.



Kuva 4. Haavoittuvuus riippuvuuskaavion ilmaisemana [20].

Tietoturvahälytysten ollessa kytkettynä päälle kaikki haavoittuvuudet listataan hakemiston hälytykset-osioon. Hälytykset ovat pakettikohtaisia ja niistä käyvät ilmi haavoittuvuuden vakavuus, mahdollinen korjaava päivitys sekä haavoittuvuuden alkuperäinen CVE-tunnistenumero. Hälytyksestä on myös suora linkki NVD-tietokantaan, josta käyttäjät voivat lukea koko CVE:n toimittaman raportin haavoittuvuudesta. Kuva 5 havainnollistaa, miten tietoturvahälytykset ilmaisevat haavoittuvuuksia.



Kuva 5. Tietoturvahälytys haavoittuneesta Node.js-paketista [20].

Github on ottanut työkaluillaan pitkän askeleen kohti turvallisempaa sovelluskehitystä. Ohjelmistoympäristöjen haavoittuvuuksien analysointi versionhallinnassa on järkevää, ja erityisesti tietoturvahälytykset antavat käyttäjälle kattavasti informaatiota haavoittuvuuksista. Suurin ongelma Githubin tietoturvatyökaluissa käytettävyyden kannalta on, että haavoittuvuuksia ei voida korjata suoraan versionhallinnassa. Kehittäjien täytyy ensin korjata ja testata paketit paikallisesti, minkä jälkeen muutokset voidaan lopulta siirtää versionhallintaan. Tietoturvatyökalujen aktivointi on kuitenkin suositeltavaa aina uudessa hakemistossa, sillä ne tarjoavat helpon ratkaisun haavoittuvuuksien monitorointiin.

Snyk.io

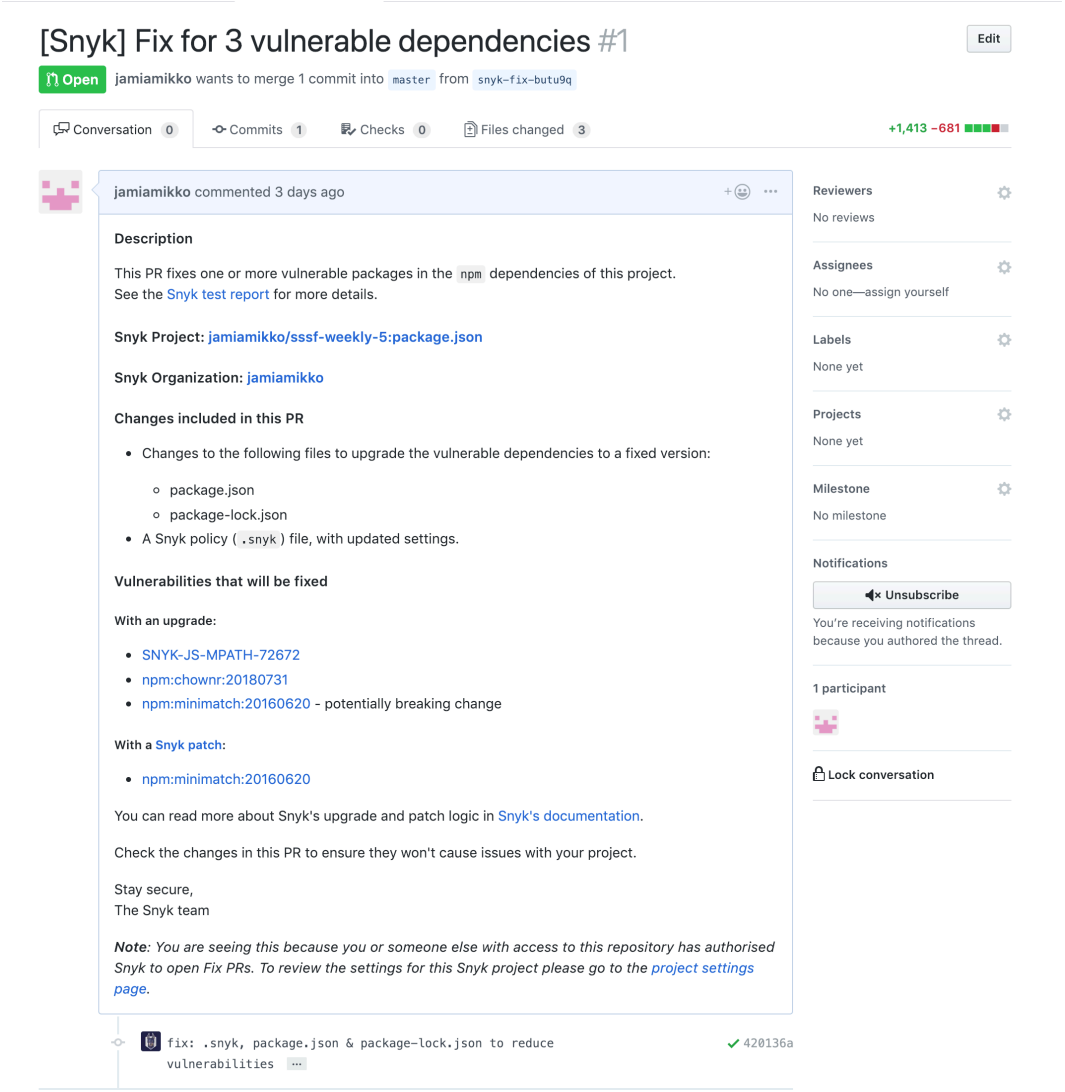
Snyk.io on avoimen lähdekoodin tietoturva-alusta. Npm auditin ja Githubin tietoturvatyökalujen tavoin Snyk analysoi ja vertaa sovelluksen paketteja haavoittuvuustietokantaa vasten. Snykin omaa haavoittuvuustietokantaa ylläpitää Israelissa toimiva tietoturvaryhmä. Tietokanta on rakennettu tietoturvaryhmän tekemän tutkimustyön pohjalta, ja sen lähteitä ovat verkosta ja muista haavoittuvuustietokannoista löytyvät raportit. Lisäksi tietoturvaryhmä analysoi Github-käyttäjien ongelmaraportteja, vertaisarviointeja ja yksittäisiä koodipäivityksiä. Työryhmä käyttää myös omia automatisoituja työkalujaan haavoittuvuuksien etsimiseen yksittäisistä Node.js-paketeista. [21.]

Haavoittuvuustietokanta on vapaasti käytettävissä Snykin verkkosivuilla. Tietokannasta voi helposti etsiä raportteja tunnetuista haavoittuvuuksista. Raportit ilmaisevat yksityiskohtaisesti, mistä haavoittuvuudesta on kyse ja miten ongelma voidaan korjata. Raportista löytyvät myös informaation alkuperäiset lähdeviitteet. Haavoittuvuustietokannan selainversio on kätevä työkalu, jos kehittäjät tarvitsevat nopeasti informaatiota yksittäisten pakettien turvallisuudesta.

Snyk on helposti integroitavissa moderneihin git-pohjaisiin versionhallintajärjestelmiin. Tällä hetkellä alusta on integroitavissa Githubin, Gitlabin ja Bitbucketin kanssa. Snyk tukee useita eri ohjelmointikieliä, mukaan lukien Javascript, Java, Python ja PHP. [22.] Snykin pääasiallinen työkalu on web-sovellus, jonka avulla käyttäjät voivat hallinnoida omia projektejaan. Web-sovellus ei vaadi käyttäjältä erikseen rekisteröitymistä, vaan sovellukseen voi kirjautua esimerkiksi käyttäjän omilla Github- tai Google-tunnuksilla. Snyk tarvitsee erikseen käyttäjän luvan voidakseen integroitua käyttäjän hakemistoihin. Työkalu listaa käyttäjän hakemistot listaan, josta käyttäjä voi helposti valita, minkä hakemistojen kanssa haluaa alustan integroituvan.

Kun käyttäjä on integroinut hakemistonsa, Snyk analysoi sen ympäristön välittömästi. Web-sovellus listaa kaikki ympäristön haavoittuvuudet järjestykseen niiden vakavuuden mukaisesti. Githubissa Snyk korjaa haavoittuvuuksia vertaisarviointien avulla. Käyttäjät voivat luoda korjaavan vertaisarviointitiketin joko kaikista haavoittuneista paketeista kerralla tai valita paketit yksitellen. Tietoturva-alusta luo vertaisarvioinnin Githubiin ja päivit-

tää haavoittuneet paketit korjaaviin versioihin. Päivityksiä ei kuitenkaan yhdistetä hakemiston master-versioon, ennen kuin käyttäjä on hyväksynyt vertaisarvioinnin. Kuvassa 6 on esimerkki Snykin luomasta vertaisarvioinnista.

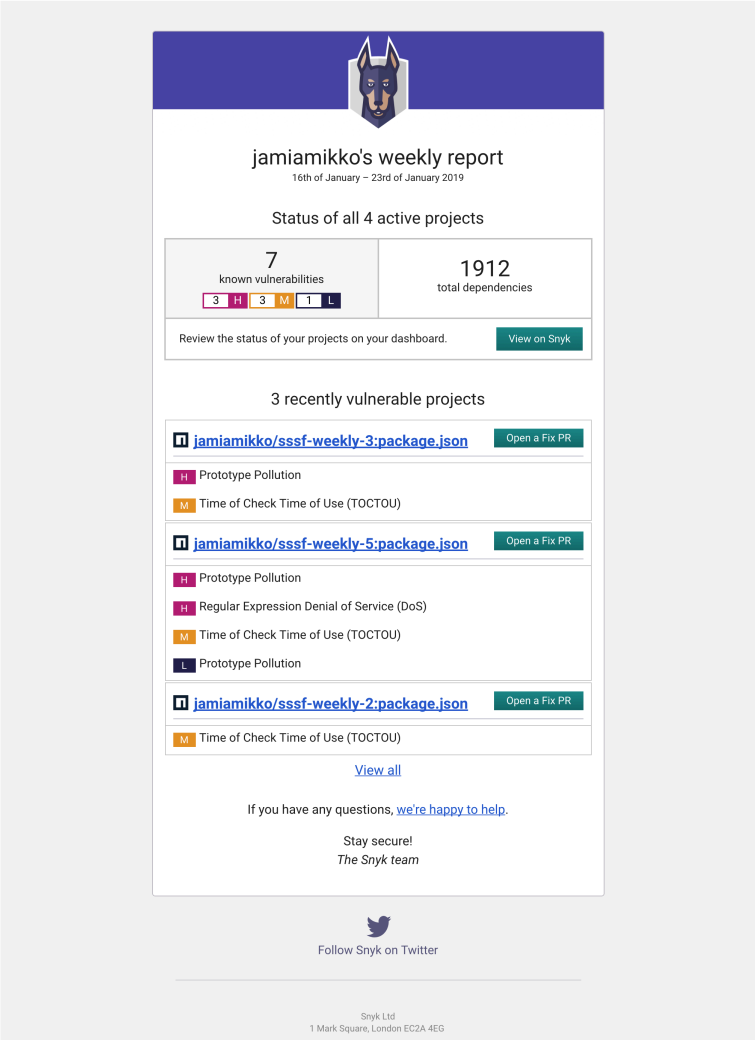


Kuva 6. Snykin luoma vertaisarviointitiketti [23].

Snyk korjaa haavoittuvuuksia kahdella tapaa: päivittämällä ja paikkaamalla. Päivittämällä Snyk päivittää haavoittuneiden pakettien versiot korjaaviin versioihin normaalisti. Pakettien päivitetyt versiot eivät aina ole taaksepäin yhteensopivia, jolloin kehittäjien on järkevää testata muutokset ennen niiden hyväksymistä. Snyk voi myös paikata haavoittuneita paketteja, jos paketille ei sillä hetkellä löydy korjaavaa versiota. Paikkaukset tehdään takaisinsovitusmenetelmällä, jossa paketin uusimpaan versioon tehdään ongelman

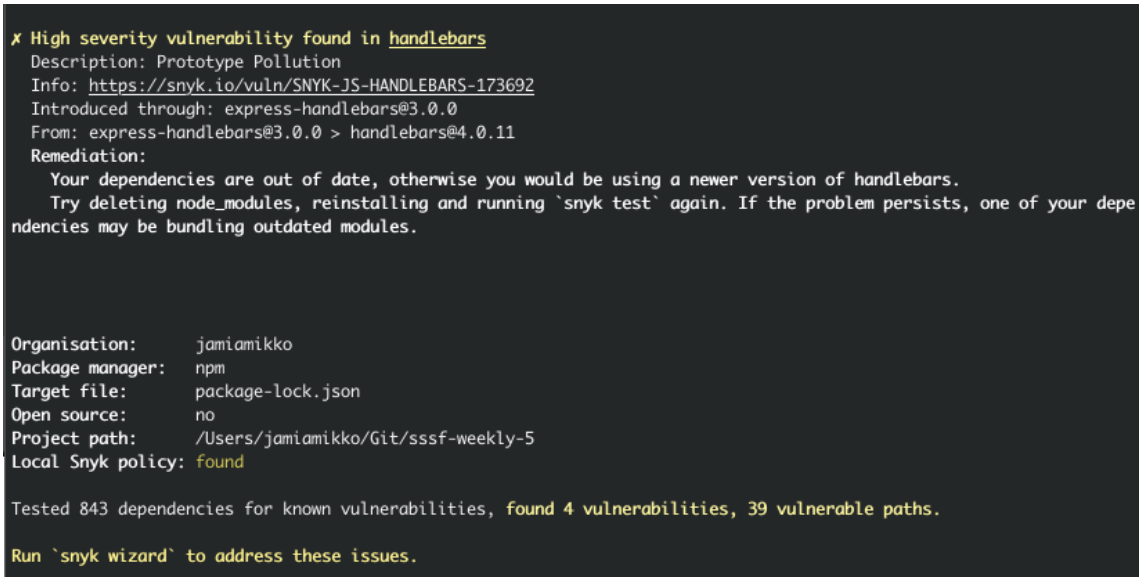
korjaava muutos, joka sen jälkeen asetetaan paketin vanhempaan versioon. Paikkauksien luomisesta ja testaamisesta vastaavat Snykin tietoturvainsinöörit, ja niitä luodaan vain tarpeen mukaan korkean riskitason haavoittuvuuksia varten. [24.]

Snyk lähettää oletuksena käyttäjälle ilmoituksia hakemistojen haavoittuvuuksista sähköpostitse. Raportti listaa kaikista projekteista löytyneet haavoittuvuudet ja niiden vakavuuden. Sähköpostista on myös suora linkki web-sovellukseen, josta käyttäjät voivat avata uuden korjaavan vertaisarvioinnin Githubiin. Kuvassa 7 on esimerkki Snykin lähettämästä sähköpostista.



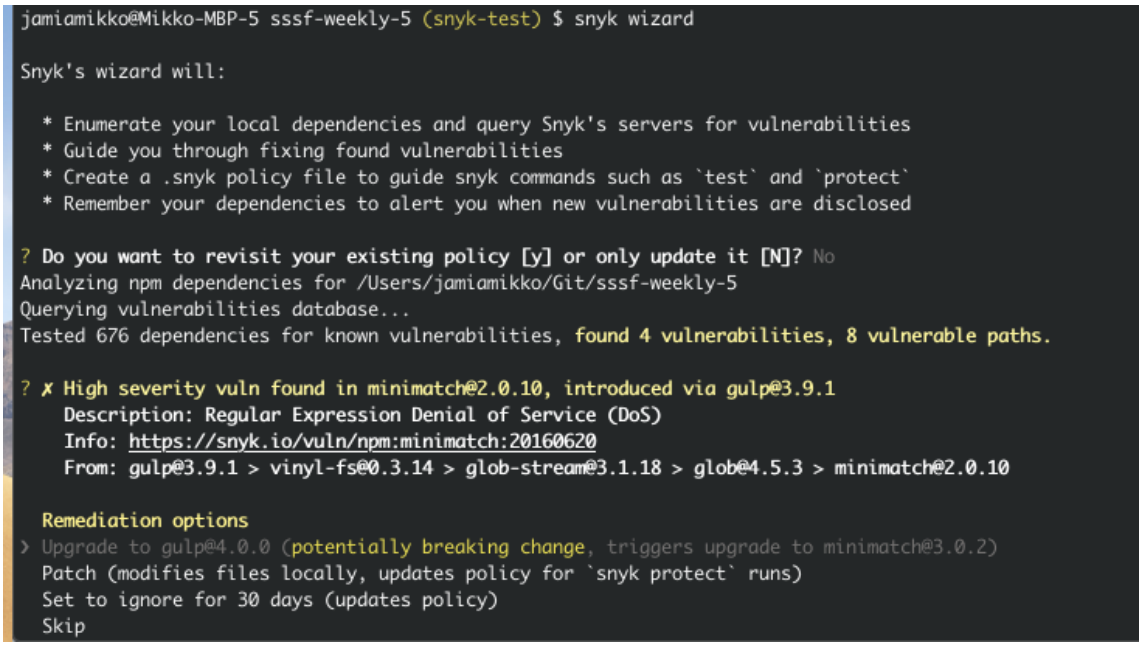
Kuva 7. Snyk-projektien sähköpostiraportti [25].

Käyttäjät voivat myös käyttää Snykin omaa komentorivityökalua analysoidakseen ympäristöjä paikallisesti. Npm auditin tavoin komentorivityökalu vertaa sovelluksen ympäristöä haavoittuvuustietokantaa vasten, tulostaa raportin komentoriville ja ohjeistaa haavoittuvuuksien korjaamisessa. Ajamalla komennon snyk test komentorivityökalu analysoi hakemiston ohjelmointiympäristön ja tulostaa raportin komentoriville. Npm auditiin verrattuna Snykin tulostama raportti on kuitenkin paljon selkeämpi ja yksityiskohtaisempi. Snyk esimerkiksi ilmaisee pakettien riippuvuudet haavoittuneisiin paketteihin erikseen. Käytännössä tämä tarkoittaa sitä, että jos kaksi Node.js-pakettia ovat riippuvaisia samasta haavoittuneesta paketista, Snyk ilmaisee sen raportissa yhtenä haavoittuneena pakettina, jolla on kaksi haavoittunutta polkua. Npm audit ilmaisee polut omina haavoittuvuuksinaan, joka ilmaistaisiin tässä tapauksessa kahtena haavoittuvuutena. [22.] Kuva 8 ilmaisee esimerkin Snyk-komentorivityökalun tulostamasta raportista.



Kuva 8. Esimerkki Snyk-komentorivityökalun tulostamasta raportista [26].

Ajamalla komentorivillä komennon snyk wizard työkalu pyrkii korjaamaan haavoittuvuudet yksitellen. Käyttäjän voi valita, korjataanko haavoittuvuus päivittämällä vai paikkaamalla vai jätetäänkö ongelma toistaiseksi korjaamatta. Komentorivityökalu osaa myös varoittaa käyttäjää, jos paketin päivitys ei ole taaksepäin yhteensopiva tai korjaavaa versiota ei ole saatavilla. Kuva 9 ilmaisee, miten Snyk-komentorivityökalun korjausprosessi tapahtuu.



Kuva 9. Esimerkki Snyk-komentorivityökalun korjausprosessista [26].

Snyk.io on työkaluna monipuolisempi kuin npm audit ja Githubin tietoturvatyökalut. Npm auditiin verrattuna Snykin komentorivityökalu on käyttäjäystävällisempi ja antaa paremman kontrollin haavoittuvuuksien korjaamisessa. Npm audit ja Github ovat toimiva ratkaisu ympäristöjen analysointiin satunnaisesti kehityksen aikana. Snyk.io on toimivampi vaihtoehto, kun ympäristön haavoittuvuuksia halutaan monitoroida ja korjata jatkuvasti.

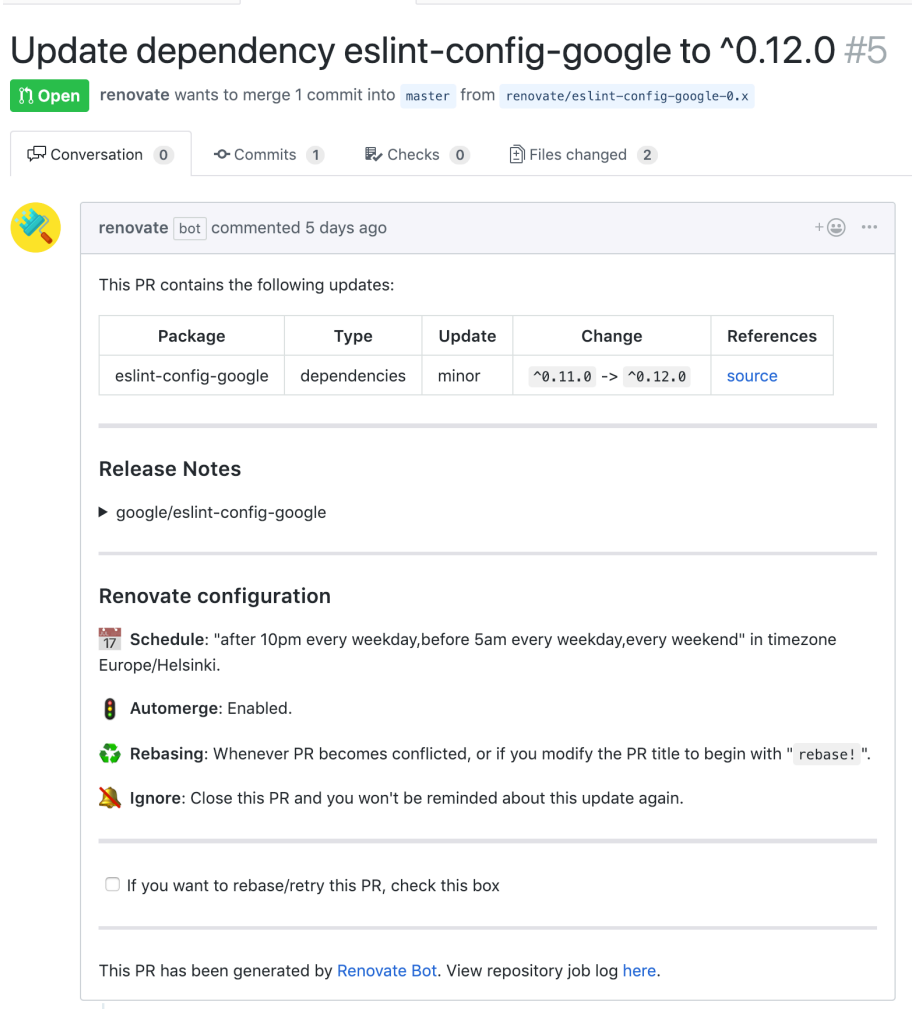
Renovate Bot

Renovate Bot on avoimen lähdekoodin sovellus, joka mahdollistaa pakettien päivittämisen automaattisesti osana versionhallintaa. Renovaten päätarkoitus ei ole etsiä ohjelmistoympäristöistä haavoittuvuuksia, vaan se pyrkii pitämään ympäristöjen paketit päivitettyinä kehittäjien haluamalla tavalla. Pitämällä ohjelmiston paketit päivitettyinä kehittäjät voivat kuitenkin edesauttaa projektien tietoturvaa. Sovellus kykenee myös havaitsemaan haavoittuvuuksia hakemistoista. Snykin tavoin sovellus tukee useita ohjelmointikieliä, kuten Node.js, Javascript ja Python. Toisin kuin Snyk.io, Renovate Bot ei ole selaimen kautta hallinnoitava web-sovellus. Renovate on sovellus, joka integroidaan suoraan hakemistoon ja määritetään toimimaan käyttäjän tarpeiden mukaisesti. Sovellus on tällä hetkellä saatavilla Githubille ja Gitlabille sekä Bitbucketille. [27.]

Renovate Botin asentaminen Github-hakemistoon onnistuu helposti Github Marketplacen kautta. Sovelluksesta on saatavilla erihintaisia tilauspaketteja henkilökohtaiseen ja organisaatioiden käyttöön. Henkilökohtainen tilaus maksaa yhden dollarin kuussa, ja sen käyttö rajoittuu käyttäjän omiin julkisiin ja yksityisiin hakemistoihin. Organisaatioille on saatavilla kolme erilaista tilausta. Organisaatiotilausten hinnat alkavat 20 dollarista päättyen 100 dollariin kuussa, riippuen organisaation koosta. Sovelluksesta on saatavilla myös ilmaisversio, jonka käyttö rajoittuu käyttäjän julkisiin hakemistoihin. [28.]

Osana asennusta käyttäjä valitsee hakemistot, joihin haluaa sovelluksen integroituvan ja myöntää samalla Renovatelle oikeudet hakemistojen lukemiseen. Asennuksen yhteydessä Renovate avaa ensimmäisen vertaisarvioinnin asetusten määrittämistä varten. Asetuksia ei määritetä selaimessa käyttöliittymän avulla kuten Snykissä, vaan ne määritetään hakemiston juureen lisättyyn renotave.json-tiedostoon. Tiedosto luodaan automaattisesti asennuksen yhteydessä, ja se sisältää valmiiksi Renovaten perusasetukset. Perusasetuksilla Renovate Bot analysoi hakemiston ympäristön kerran tunnissa etsien päivitystä tarvitsevia paketteja. Renovate suorittaa pakettien päivittämistä Snykin tavoin vertaisarviointitikettien avulla. Jos paketista löytyy uusi versio tai haavoittuvuuksia havaitaan, avataan automaattisesti uusi vertaisarviointitiketti.

Oletuksena Renovate avaa taaksepäin yhteensopivat päivitykset omiin vertaisarviointeihin ja taaksepäin yhteensopimattomat päivitykset omiin vertaisarviointeihin. Vertaisarviointeja on järkevää luoda erillään, sillä taaksepäin yhteensopimattomat päivitykset voivat rikkoa sovelluksen toiminnallisuutta ja usein vaativat muutoksia lähdekoodiin. Kuvassa 10 on esimerkki sovelluksen luomasta vertaisarvioinnista.



Kuva 10. Esimerkki Renovate Botin luomasta vertaisarvioinnista [23].

Npm-paketinhallintajärjestelmä rajoittaa usein pakettien versionumeroita semanttisesti. Nämä ehdot perustuvat pakettien suositeltuihin versioihin, joita pakettien kehittäjät ovat niille asettaneet. Paketinhallintajärjestelmä voi esimerkiksi asettaa käytetyksi versionumeroksi ^1.1.0. Tämä tarkoittaa, että mikä tahansa paketin versio, joka on suurempi kuin 1.1.0, mutta pienempi kuin 2.0.0, on sallittua käyttää. Renovate voi luoda semanttisesti rajatuista paketeista vertaisarvioinnin, joissa ehdotetaan pakettien versioiden kiinnittämistä sen nykyiseen käytettyyn versioon. Kiinnittämisen jälkeen paketin versionumero pysyy aina samana, vaikka käyttäjä ajaisi päivityksiä ympäristöön. Esimerkiksi aiemmin mainittu versionumero 1.1.0 säilyisi päivityksistä huolimatta aina samana. Kiinnittämisestä voi olla hyötyä, jos paketeilta vaaditaan aina samanlaista käyttäytymistä. [29.]

Renovate antaa käyttäjälle paljon vapauksia sovelluksen asetusten määrittämisessä. Tyypillisinä asetuksina käyttäjä voi esimerkiksi asettaa taaksepäin yhteensopivien pakettien päivitykset automaattisiksi, jolloin vertaisarvioinnit eivät vaadi käyttäjän hyväksyntää. Vastaavasti käyttäjä voi asettaa Renovaten jättämään tiettyjä paketteja kokonaan päivittämättä. Käyttäjä voi myös esimerkiksi määrittää Renovaten ajamaan pakettien analysoinnin tapahtumaan työaikojen ulkopuolella. Esimerkkikoodi 1 havainnollistaa, miten edellä mainitut esimerkit voidaan määritellä asetuksissa.

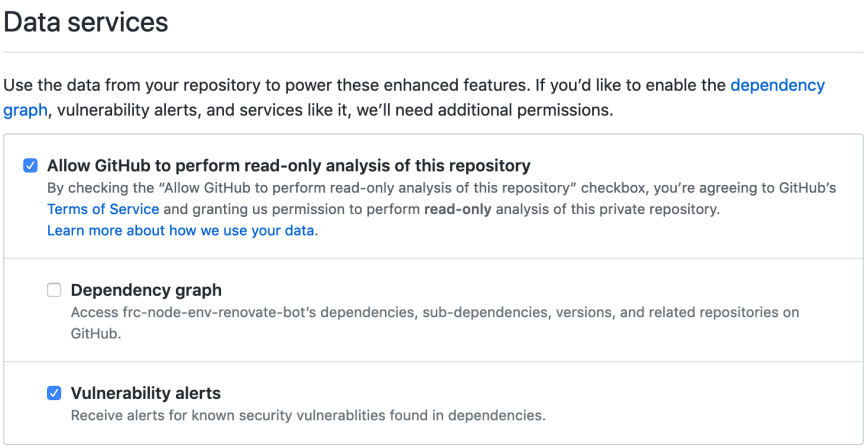
```
{
  "extends": ["config:base"],
  "assignees": ["jamiamikko"],
  "automerge": true,
  "major": {
    "automerge": false
  },
  "ignoreDeps": ["nodemon"],
  "schedule": [
    "after 10pm every weekday",
    "before 5am every weekday",
    "every weekend"
  ],
  "timezone": "Europe/Helsinki",
  "vulnerabilityAlerts": {
    "labels": ["security"],
    "assignees": ["jamiamikko"]
  }
}
```

Esimerkkikoodi 1. Esimerkki tyypillisistä asetuksista renovate.json-tiedostossa.

Käyttämällä assignees-asetusta voidaan määrittää sovellus ilmoittamaan uusista päivityksistä valituille Github-käyttäjille sähköpostitse. Renovate bot suorittaa myös validointia kaikkiin käyttäjän tekemiin muutoksiin asetuksissa. Jos asetuksissa ilmenee virheellistä JSON-dataa, avaa Renovate uuden Github-virhetiketin. Virhetiketit opastavat käyttäjää asetusten korjaamisessa.

Haavoittuvuuksien etsimiseen sovellus hyödyntää Githubin tietoturvahälytyksiä. Käyttäjä voi muuttaa sovelluksen käyttäytymistä haavoittuvuuksien ilmetessä käyttämällä Renovaten vulnerabilityAlerts-asetusta. Asetuksella voidaan lisätä haavoittuvuuksista johtuneille vertaisarvioinneille leimoja tai suffikseja, joilla tiketit ovat helposti löydettävissä hakemistosta. [30.] Renovate Bot pystyy hyödyntämään tietoturvahälytyksiä esteettömästi julkisissa Github-hakemistoissa. Jotta sovellus kykenee havaitsemaan haavoittuvuuksia

yksityisessä hakemistossa, tulee käyttäjän ottaa ensin tietoturvahälytykset käyttöön hakemiston asetuksista. Kuvassa 11 on tietoturvahälytykset otettu käyttöön hakemiston asetuksissa.



Kuva 11. Tietoturvahälytykset otettuna käyttöön Github-hakemiston asetuksissa [20].

Kun Renovate on asennettu yksittäiseen hakemistoon, se analysoi vain kyseisen hakemiston ympäristöä. Tämä tarkoittaa käytännössä sitä, että jos käyttäjä haluaa käyttää sovellusta useammassa hakemistossa, tulee jokaiseen hakemistoon määrittää asetukset erikseen. Usean hakemiston yksitellen hallinnoiminen voi olla epäkäytännöllistä organisaatiossa, jossa kehittäjät työskentelevät usean hakemiston kanssa samanaikaisesti. Tähän ongelmaan Renovate on luonut valmiiksi määritettyjä asetuksia, jotka voivat helpottaa usean hakemiston hallinnoimista vähentämällä asetusten määrittämiseen käytettyä aikaa. [31.]

Käyttäjät voivat myös luoda omia oletusasetuksiaan käytettäväksi hakemistojen välillä. Asetuksille voidaan luoda Node.js-paketti, jonka package.json-tiedostossa asetukset ovat määritettynä renovate-config-avaimen taakse. Tämän jälkeen paketti julkaistaan npm-rekisteriin, mistä sitä voidaan käyttää laajentamaan eri hakemistojen Renovate-asetuksia. [31.] Esimerkkikoodi 2 havainnollistaa, miten asetukset voidaan määritellä Node.js-paketissa.

```
{
  "name": "renovate-config-jamiamikko",
  "version": "0.0.1",
  "description": "My default configuration for Renovate projects.",
  "renovate-config": {
    "default": {
      "extends": ["config:base"],
      "assignees": ["jamiamikko"],
      "automerge": true,
      "major": {
        "automerge": false
      },
      "ignoreDeps": ["nodemon"],
      "schedule": [
        "after 10pm every weekday",
        "before 5am every weekday",
        "every weekend"
      ],
      "timezone": "Europe/Helsinki",
      "vulnerabilityAlerts": {
        "labels": ["security"],
        "assignees": ["jamiamikko"]
      }
    }
  }
}
```

Esimerkkikoodi 2. Renovate-projektin oletusasetukset Node.js-pakettina.

Käyttäjä voi myös vaihtoehtoisesti luoda omille Renovate-asetuksilleen oman Github-hakemiston, jonka juureen asetetaan renovate.json-tiedosto yleisille asetuksille [31]. Käyttäjät voivat esimerkiksi luoda omaan käyttöönsä uuden hakemiston nimeltä my-renovate-config, joka sisältää vain renovate.json-tiedoston esimerkkikoodin 1 asetusten mukaisesti. Tämän jälkeen asetuksia voidaan käyttää muissa hakemistoissa laajentamalla esimerkkikoodin 3 mukaisesti.

```
{
  "extends": ["github>jamiamikko/my-renovate-config"]
}
```

Esimerkkikoodi 3. Asetuksia laajennettuna Github-hakemistosta.

Renovate Bot on toimiva työkalu yksittäisille kehittäjille ja organisaatioille, jotka haluavat pitää projektiensa paketit päivitettyinä ja turvallisina. Renovate Bot antaa käyttäjille paljon toiminallisuutta pienessä paketissa. Sovelluksella ei ole varsinaisesti käyttöliittymää tai omaa komentorivityökalua kuten Snykillä. Renovate on kuitenkin onnistunut täysin eristämään sovelluksen toimimaan yhteistyössä versionhallinnan kanssa, jolloin ulkopuoliset komponentit jäävät tarpeettomiksi. Sovelluksen käyttö vaatii kuitenkin laajempaa ymmärrystä ohjelmistoympäristöistä, versionhallinnasta ja JSON-tiedostomuodosta.

Sovellus onkin selvästi suunnattu kehittäjille, kun taas Snyk.io helpolla käyttöliittymällään soveltuu laajemmalle käyttäjäkunnalle.

6 Sovellusprojektien haavoittuvuuksien kartoitus

6.1 Asiakastyön suunnittelu ja työkalut

Merkittävä osa asiakkaan Github-organisaation hakemistoista käyttää Node.js-ympäristöjä tuomaan sovelluksiin ominaisuuksia ja tehostamaan projektien kehitys- ja julkaisuprosesseja. Koska Node.js-ympäristöjen haavoittuvuudet ovat jatkuva puheenaihe, asiakas oli kiinnostunut organisaation sovellusprojektien tietoturvasta. Insinöörityön tarkoituksena oli kartoittaa projektien Github-hakemistoista tunnetut haavoittuvuudet ja tehostaa niiden tietoturvaa tulevaisuudessa.

Ensimmäinen hahmotelma projektista oli kehittää sovellus, joka integroitaisiin yhteen tai useampaan hakemistoon versionhallinnassa. Sovellus olisi säännöllisesti analysoinut hakemiston julkaisuvalmiin version Node.js-ympäristön etsien tunnettuja haavoittuvuuksia. Jos haavoittuvuuksia havaitaan, sovellus pysäyttäisi hakemiston jatkuvan julkaisuprosessin väliaikaisesti. Jotta jatkuvaa julkaisua voitaisiin jatkaa, tulisi kehittäjien ensin korjata hakemiston tunnetut haavoittuvuudet. Tieto löytyneistä haavoittuvuuksista saataisiin kehittäjille sähköpostin tai Slackin kautta.

Tehdessäni tutkimustyötä Node.js-ympäristöjen haavoittuvuuksista ja vertaillessani saatavilla olevia työkaluja oivalsin alun perin suunnitellun sovelluksen kehittämisen olevan ajan hukkaa. Asiakkaan kanssa käydyn keskustelun jälkeen päätettiin muuttaa asiakastyön toteutusta vaihtamatta kuitenkaan työn alkuperäisiä tavoitteita. Projektissa päätettiin hyödyntää olemassa olevia työkaluja analyysin suorittamiseen. Uudessa suunnitelmassa Github-organisaation Node.js-ympäristöä käyttävien hakemistojen package.json-tiedostot kerättäisiin yhteen hakemistoon. Tiedostojen keräämisen jälkeen kaikkien hakemistojen ympäristöt voitaisiin analysoida asianmukaisilla työkaluilla. Valmiin analyysin tulokset toimitettaisiin lopulta asiakkaalle.

Jotta projektia voitiin jatkaa, tuli ensin päättää työkalu, jolla hakemistojen analysointi voitaisiin parhaiten toteuttaa. Snyk.io ja Renovate Bot tuntuivat parhailta vaihtoehdoilta tehtävän suorittamiseen niiden monipuolisuuden ansiosta. Npm audit ja Githubin tietoturvatyökalut eivät itsessään pystyisi ilmaisemaan haavoittuvuuksia tavalla, joka tekisi analyysista tarpeeksi sujuvaa. Valinta osui lopulta Renovate Bot -työkaluun. Renovate Bot valittiin sen yksinkertaisuuden, joustavuuden ja luotettavuuden ansiosta. Työkalun hallinnoiminen suoraan Github-hakemistossa vaikutti paljon luontevammalta kuin Snykin erillisessä web-sovelluksessa. Lisäksi valintaan vaikutti sovelluksen joustavuus. Renovaten monipuolisilla asetuksilla sovellus voitaisiin määrittää suorittamaan analyysi juuri halutulla tavalla. Työkalua oli myös aiemmin käytetty Franticin suurimman asiakkaan Github-hakemistoissa, joissa se oli todettu toimivaksi työkaluksi pitämään projektien paketit päivitettyinä ja turvallisina.

6.2 Tiedostojen kopiointi ja analyysi

Ensimmäinen haaste projektin toteutuksessa oli ratkaista, kuinka kaikkien Franticin Github-hakemistojen Node.js-ympäristöjä kuvaavat tiedostot voitaisiin helposti kopioida yhteen hakemistoon. Franticin Github-organisaatioissa on yli 450 hakemistoa, joten hakemistojen käsin kopioiminen ei olisi kannattanut. Potentiaalinen ratkaisu ongelmaan oli ajaa komentorivillä komentosarjoja, jotka etsivät paikallisista hakemistoista tarvittavat tiedostot ja kopioisivat ne lopulta uuteen hakemistoon.

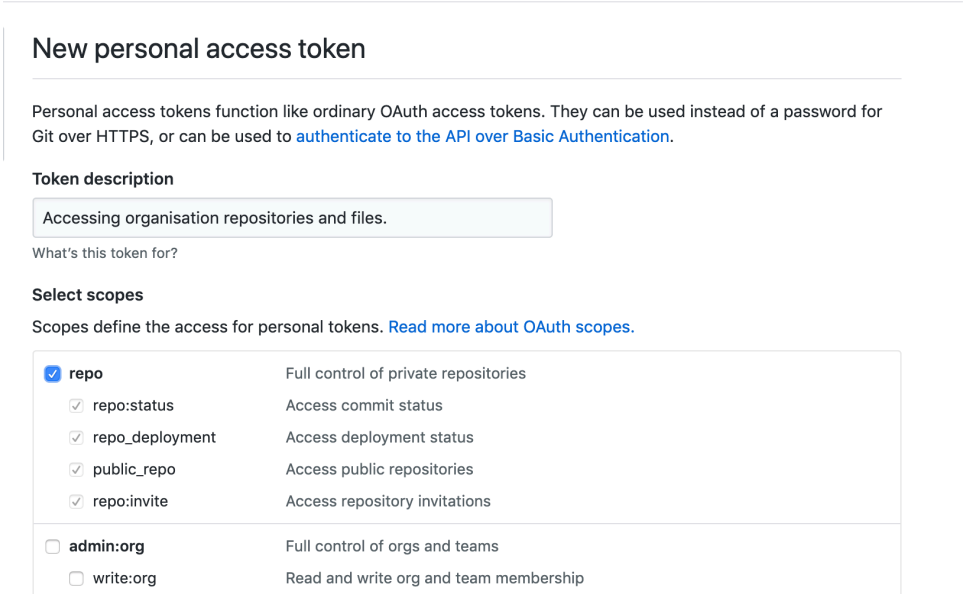
Jotta Renovate Bot voisi analysoida projektien paketit, oli välttämätöntä kopioida vähintään jokaisen hakemiston package.json-tiedosto. Renovate Bot ei tarvitse pakettien analysointiin package-lock.json-tiedostoa. Tiedosto kuitenkin antaa tarkan kuvauksen hakemiston Node.js-ympäristöstä ja pakettien riippuvuuksista, joten totesin sen kopioimisen kannattavan, jos tiedosto vain olisi saatavilla. Komentosarjan tuli myös luoda jokaiselle tiedostolle oma kansio hakemiston nimellä, jotta se olisi myöhemmin helposti tunnistettavissa. Komentosarjat ajettiin macOS-käyttöjärjestelmän Unix-pohjaisella komentorivityökalulla. Useiden kokeilujen jälkeen päädyin esimerkkikoodin 4 mukaiseen komentosarjaan.

```
find repositories -maxdepth 2 -name "package*.json" -exec cp --parents {}
testCopy \;
```

Esimerkkikoodi 4. Komentosarja tiedostojen kopioimiseen paikallisesti.

Komentosarja suoritti paikallisten tiedostojen kopioinnin sujuvasti. Ongelmaksi muodostui kuitenkin se, että kaikkien yli 450 hakemiston tulisi olla paikallisesti kopioituna käyttäjän työasemalle, jotta tiedostot saataisiin kopioitua. Hakemistojen yksitellen lataaminen Githubista käsin veisi niin paljon aikaa, ettei se olisi kannattanut. Ratkaisuna ongelmaan päätin hyödyntää Githubin omaa ohjelmointirajapintaa. Se mahdollistaa informaation hakemisen suoraan Githubista http-kutsujen avulla. Kutsuja rajapintaan voitaisiin myös kirjoittaa komentosarjoina komentoriville. Prosessia helpottaakseni päätin kuitenkin kehittää projektia varten Node.js-sovelluksen tiedostojen hakemiseen. Sovellus ajettaisiin index.js-tiedostossa, joka sijaitsisi uuden hakemiston juuressa.

Tietoa haettaessa rajapinnasta Github vaatii käyttäjän tunnistuksen. Githubin asetuksissa käyttäjät voivat luoda itselleen henkilökohtaisia tunnisteavaimia, jotka toimivat tunnisteena http-kutsuissa. Tunnisteavain on numerosarja, joka liitetään parametrina osaksi kutsua. Tunnisteavainta voidaan myös käyttää tunnistukseen tietoa haettaessa Github-organisaatioista, edellyttäen että käyttäjä on organisaation jäsen. Tunnisteavaimelle voidaan määrittää Githubin asetuksissa oikeuksia sen käyttötarkoituksen mukaisesti. Hakemistojen ja niiden tiedostojen hakemiseen tunnisteavaimelle riitti kaikki hakemistoihin liittyvät oikeudet. Kuva 12 havainnollistaa, miten uusi henkilökohtainen tunnisteavain luodaan ja kuinka tarvittavia oikeuksia voidaan määrittää.



Kuva 12. Githubin tunnisteavain, jolla on täydet oikeudet hakemistoihin [32].

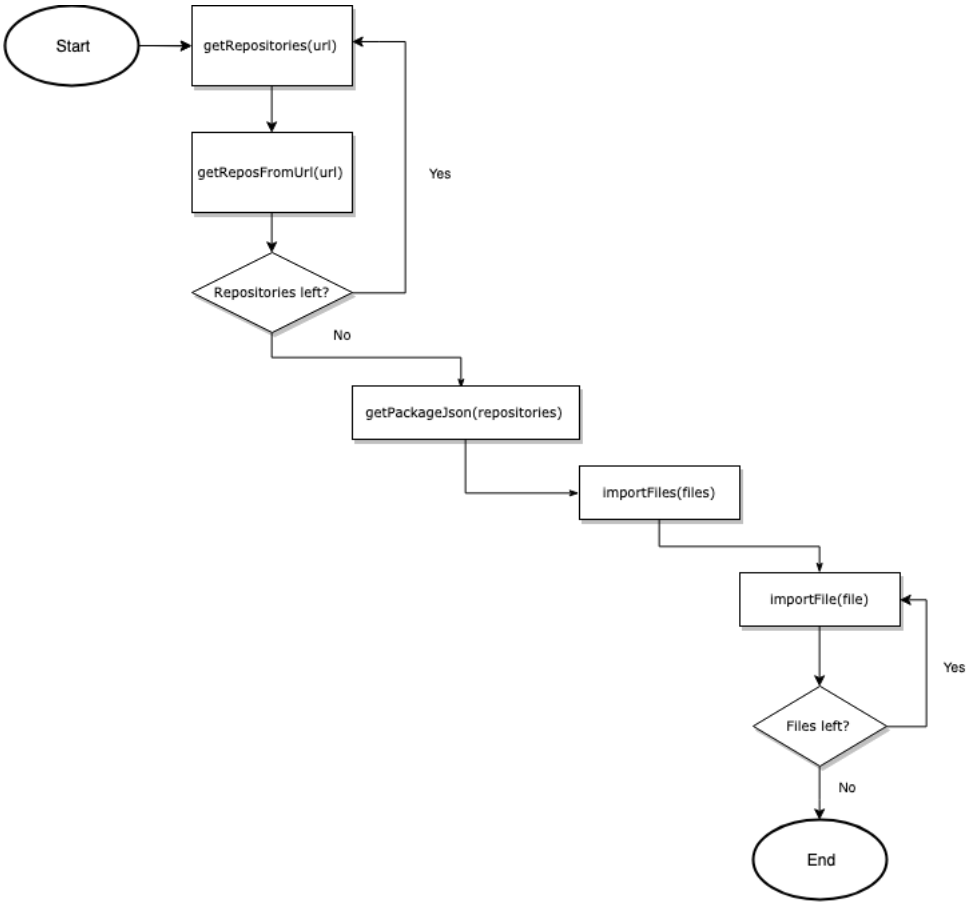
Node.js-sovellus käytti kolmea npm-rekisteristä asennettua Node.js-pakettia apunaan tiedostojen hakemiseen: dotenv, SuperAgent ja chalk. Dotenv on riippumaton Node.js-paketti, jonka avulla käyttäjät voivat helposti luoda ympäristömuuttujia käytettäväksi Node.js:n tapahtumasilmukkaan. Yleisesti ympäristömuuttujiin asetetaan informaatiota, jota ei haluta paljastaa muun maailman nähtäväksi. Tämänkaltaista informaatiota ovat esimerkiksi rajapintakutsuissa käytetyt tunnisteavaimet. SuperAgent on http-kutsujen tekemiseen tarkoitettu Node.js-kirjasto. Paketin käyttö sovelluksessa ei ollut välttämätöntä, sillä Node.js sisältää oletuksena oman ohjelmointirajapinnan http-metodeille. Totesin kuitenkin kutsujen tekemisen SuperAgentilla sujuvaksi, joten päätin käyttää sitä sovelluksessa. Chalk on Node.js-paketti, joka mahdollistaa värikkäiden viestien tulostamisen komentoriville. Chalk-paketti ei ole tiedon hakemisen kannalta välttämätön, mutta se parantaa sovelluksen käyttökokemusta ilmaisemalla sovelluksen kulkua komentorivillä värikkäillä viesteillä.

Dotenv-paketti lukee käyttäjän määrittämiä ympäristömuuttujia hakemiston juureen lisätystä .env-tiedostosta. Tähän tiedostoon oli sovelluksessa määritetty muuttuja GITHUB_AT, jonka arvona oli Githubissa luotu tunnisteavain rajapintakutsuja varten. Sovelluksen käynnistyessä tunnisteavain oli välittömästi saatavilla globaalin Process-objektin kautta. Sen avulla tunnisteavain voitiin asettaa rajapintakutsuun tunnistusta varten muodossa process.env.GITHUB_AT.

Npm-rekisteristä asennettujen pakettien lisäksi sovellus käytti Node.js-ympäristöön sisäänrakennettua File System -moduulia. Moduuli sisältää ohjelmointirajanpinnan kansioiden ja tiedostojen hallinnoimiseen Node.js-sovelluksissa. Moduulin tarkoitus sovelluksessa oli luoda kopioiduille hakemistoille uudet kansiot ja luoda uudestaan package.json- ja package-lock-tiedostot kansioiden alle.

Ensimmäiseksi sovellus haki Githubin rajapinnasta kaikki Franticin organisaation alla olevat hakemistot. Näistä yli 450 hakemistosta rajapinta kykeni kuitenkin kerralla hakemaan vain 100. Tämä johtui siitä, että Github pyrkii edistämään palvelintensa toimintaa rajoittamalla rajapinnan palauttaman datan määrää. Tästä syystä rajapintaan jouduttiin tekemään useita kutsuja, jotta kaikki hakemistot saatiin varmasti listattua. Rajapinnasta saadun datan pohjalta luotiin sovelluksen muistiin lista, joka sisälsi kaikkien hakemistojen nimet.

Jokaisesta listatusta hakemiston nimestä tehtiin kaksi uutta kutsua rajapintaan. Ensimmäinen kutsu haki hakemiston juuresta package-json-tiedoston, ja toinen kutsu yritti hakea hakemiston package-lock.json-tiedoston, jos tiedosto oli saatavilla. Rajapinnan palauttamasta datasta muodostettiin uusi objekti, johon tallennettiin hakemiston nimi sekä tiedoston nimi ja sisältö. Kun kaikki saatavilla olevat tiedostot oli haettu rajapinnasta, niille luotiin File System -moduulilla uudet kansiot sovelluksen juureen frc-repositories-kansioon. Näihin kansioihin luotiin File System -moduulilla uudestaan ympäristöä kuvaavat tiedostot rajapinnasta saadun datan pohjalta. Kuva 13 visualisoi sovelluksen kulkua ja sovelluksessa käytetyt funktiot.



Kuva 13. Node.js-sovelluksen kulkukaavio.

Kun hakemistojen tiedostot oli saatu kopioitua onnistuneesti, voitiin tiedostot siirtää uuteen Github-hakemistoon. Sovellukselle luotiin uusi hakemisto nimeltä frc-node-js-renovate-bot, jonne sovellus ja kopioidut tiedostot siirrettiin versionhallintaan. Hakemiston asetuksista kytkettiin päälle Githubin tietoturvatyökalut, jotta Renovate Bot voisi myöhemmin etsiä ympäristöistä haavoittuvuuksia. Koska ympäristöjä kuvaavat tiedostot olivat alakansion frc-repositories alla, tuli tietoturvatyökaluista kytkeä päälle tietoturvahälytysten lisäksi myös riippuvuuskaavio.

Kun tietoturvatyökalut oli kytketty päälle, voitiin aloittaa Renovate Botin integroiminen hakemistoon. Sovellukselle annettiin ensin normaalit lukuoikeudet hakemistoon. Asennuksen yhteydessä luotuun renovate.json-tiedostoon laajennettiin Renovaten perusasetukset. Perusasetuksista yliajettiin asetukset prConcurrentLimit ja prHourlyLimit. Perusasetuksilla Renovate voi avata vain kaksi vertaisarviointia tunnissa ja pitää saman aikaisesti auki 20 vertaisarviointia. Jotta analysointi voitaisiin tehdä mahdollisimman nopeasti,

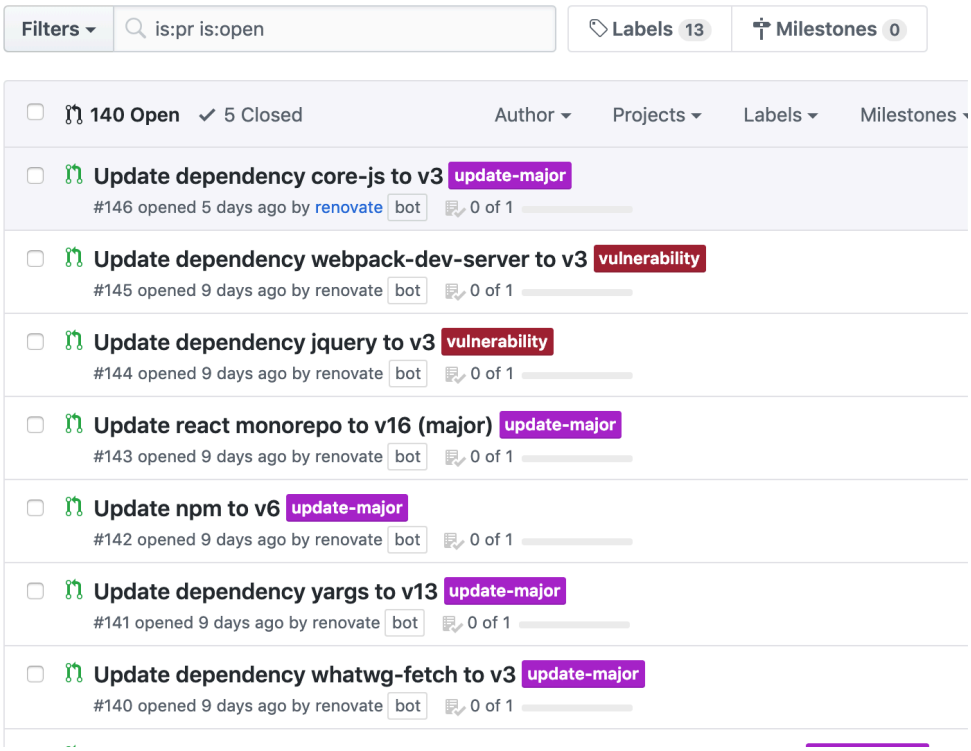
annettiin Renovatelle lupa avata vertaisarviointeja ilman rajoituksia asettamalla molempien asetusten arvoksi nolla. Lopuksi asetuksiin lisättiin eritasoisille päivityksille kuvaavat leimat, jotka helpottaisivat vertaisarviointien kategorisointia hakemistossa. Kun asetukset oli määritelty, voitiin analyysi aloittaa yhdistämällä renovate.json-tiedosto hakemiston master-versioon. Esimerkkikoodi 5 havainnollistaa projektissa käytetyt Renovate-asetukset.

```
{
  "extends": ["config:base"],
  "assignees": ["mjamia"],
  "vulnerabilityAlerts": {
    "labels": ["vulnerability"]
  },
  "prConcurrentLimit": 0,
  "prHourlyLimit": 0,
  "packageRules": [
    {
      "updateTypes": ["major"],
      "labels": ["update-major"]
    },
    {
      "updateTypes": ["minor"],
      "labels": ["update-minor"]
    },
    {
      "updateTypes": ["pin"],
      "labels": ["update-pin"]
    },
    {
      "updateTypes": ["patch"],
      "labels": ["update-patch"]
    }
  ]
}
```

Esimerkkikoodi 5. Frc-node-js-renovate-bot-hakemiston Renovate Bot -asetukset.

6.3 Analyysin tulokset

Analyysista saatiin luotua 140 uutta vertaisarviointitikettiä. Näistä vertaisarvioinneista 62 sisälsi major-tason päivityksiä paketteihin. Minor- ja patch-tason päivityksiä sisälsivät 89 vertaisarviointitikettiä. Vertaisarviointeja, jotka korjasivat suoranaisia haavoittuvuuksia paketeissa, avattiin kaksi. Haavoittuneiksi paljastuivat Node.js-paketit jQuery ja webpack-dev-server. Haavoittunut versio jQuery-paketista oli käytössä yhdeksässä hakemistossa ja haavoittunut webpack-dev-server-paketin versio neljässä hakemistossa. Kuva 14 havainnollistaa, kuinka vertaisarvioinnit listautuivat Githubiin.



Kuva 14. Analyysista avattuja vertaisarviointikettejä Githubissa [23].

JQuery on yleisesti käytetty avoimen lähdekoodin Javascript-kirjasto. NVD-tietokannan raportti paljasti 3.0.0-versiota vanhempien versioiden paketista olevan alttiita Cross site scripting -hyökkäyksille. Cross site scripting on tietoturvahyökkäys, jossa hyökkääjä syöttää haitallista koodia verkkosivulle. Haitallinen koodi on usein Javascript-koodia, jonka hyökkäyksen uhri ajaa tietämättään selaimessa. Tyypillisessä Cross site scripting -hyökkäyksessä käyttäjän identiteetti voidaan varastaa esimerkiksi selaimen evästeistä. [33.] NVD-tietokannan raportin mukaan vanhat jQueryn versiot sisältävät hyökkäyksille altistavan tietoturva-aukon kirjaston http-kutsuissa, jos kutsuun ei määritetä minkä tyyppistä dataa palvelimelta halutaan vastaanottaa.

Webpack on työkalu, jota käytetään sovelluskehityksessä Javascript-moduulien yhdistämiseen. Webpack-dev-server-paketti on osa Webpackia, joka tarjoaa kehittäjille lähdekoodin muutoksiin reagoivan palvelimen kehitysympäristöön. NVD-tietokannan raportti paljasti haavoittuvuuden paketin 3.1.6-versiota vanhemmissa versioissa. Haavoittuvuus havaittiin pakettiin kuuluvassa Server.js-tiedostossa, ja se mahdollistaa hyökkääjien varastaa kehittäjien lähdekoodia sovelluksista.

Kun analyysi oli suoritettu, sen tulokset ilmoitettiin asiakkaalle. Projektin frc-node-js-renovate-bot-hakemisto ja sen vertaisarvioinnit jaettiin Franticin teknologiaan keskittyvälle Slack-kanavalle asiakkaan toiveesta. Slack-kanavalta tulokset olisivat helposti kaikkien yrityksen kehittäjien nähtävillä. Kehittäjiä myös kehotettiin korjaamaan kaikki analyysista paljastuneet haavoittuvuudet ja samalla integroimaan omat hakemistonsa Renovate Botin kanssa.

Asiakas oli projektin tuloksiin tyytyväinen. Projektissa onnistuttiin täyttämään sen alussa asetetut tavoitteet, vaikka alkuperäisessä suunnitelmassa ei pysytty. Analyysin tulokset saivat myös aikaan keskustelua kehittäjien keskuudessa, mikä oli palkitsevaa. On toivottavaa, että keskustelun lisäksi tulokset inspiroisivat kehittäjiä tulevaisuudessa kiinnittämään huomiota käyttämiinsä paketteihin hyödyntäen asianmukaisia työkaluja, esimerkiksi Renovate Botia.

7 Yhteenveto

Insinöörityön tarkoituksena oli perehtyä Node.js-ympäristöihin ja tutkia toimintatapoja ympäristöjen haavoittuvuuksien havaitsemiseen ja korjaamiseen. Työssä perehdyttiin myös DevOps-toimintamalliin ja versionhallintaan sekä esiteltiin neljä eri työkalua, joilla haavoittuvuuksia voidaan havaita ja korjata. Tutkimustyöstä saadun informaation avulla tehtiin projekti, jossa asiakkaan suuren Github-organisaation hakemistoista kartoitettiin ja analysointiin tietoturvahaavoittuvuudet asianmukaisia työkaluja käyttäen.

Projektin alussa tehdyt suunnitelmat eivät lopulta säilyneet samoina. Hyvin pian projektin alettua hylättiin ajatus oman sovelluksen rakentamisesta, sillä Renovate Bot todettiin toimivaksi ratkaisuksi tehtävän suorittamiseen. Projektin tavoitteet kuitenkin säilyivät samana koko sen etenemisen ajan. Suurimpana haasteena projektissa oli hakemistojen ja tiedostojen kopioiminen uuteen hakemistoon sujuvasti ja luotettavasti. Oman Node.js-sovelluksen rakentaminen oli toimiva ratkaisu ongelmaan, ja sen kehittäminen teki työstä entistä mielenkiintoisempaa. Lisäksi päätös sovelluksen rakentamisesta kyseisellä teknologialla tuntui luonnolliselta, sillä insinöörityössä keskityttiin juuri Node.js-ympäristöihin.

Alussa analyysin suorittaminen suunnitellussa skaalassa tuntui hyvin kunnianhimoiselta. Renovate Bot kuitenkin kykeni suorittamaan kaikkien hakemistojen Node.js-ympäristöjen analysoimisen vaivatta, kun tarkoitukseen parhaiten sopivat asetukset löydettiin. Analyysista paljastuneiden haavoittuvuuksien lukumäärä oli myös huojentavan vähäinen. Kaikkien hakemistojen analysoiminen kerralla oli toimiva ratkaisu kartoittamaan haavoittuvuuksien nykytilanteen, mutta jatkuvana prosessina osana versionhallintaa sitä ei voida suositella. Parempi toimintatapa olisi integroida Renovate Bot yksitellen jokaiseen hakemistoon, jossa haavoittuneet ja vanhentuneet paketit olisivat suoraan korjattavissa. On myös todennäköistä, että integroimalla sovelluksen suoraan yksittäiseen hakemistoon saadaan analyysista entistä tarkempia tuloksia. Tämä johtuu siitä, että suoraan hakemistoon integroituna sovelluksella on suora pääsy kaikkiin hakemiston kansioihin ja tiedostoihin, joista voi ilmetä enemmän informaatiota hakemiston Node.js-ympäristöstä kokonaisuutena. Projektissa tehty analyysi onnistui kuitenkin täyttämään projektin alussa asetetut tavoitteet kiitettävästi.

Analyysissa onnistuttiin kartoittamaan asiakkaan Node.js-pohjaisten sovellusprojektien haavoittuvuudet. Kaikki Franticin hakemistot eivät kuitenkaan käytä Node.js-ympäristöjä, joten muita ohjelmointiympäristöjä hyödyntävien projektien tilannetta ei kartoitettu. Työtä oli rajattava, jotta työmäärä säilyisi järkevissä mitoissa yhtä analyysia varten. Projektissa käytettyä ideaa usean hakemiston analysoimisesta kerralla voitaisiin hyödyntää muissakin ohjelmointiympäristöissä.

Kehittäjät ovat vasta viime vuosina heränneet haavoittuvuuksien tuomien tietoturvariskien vakavuuteen. Pelkästään kiinnittämällä huomiota ympäristöissä käytettävien Node.js-pakettien versioihin voidaan helposti suojata sovelluksia haavoittuvuuksilta. Pakettien haavoittuvuuksien riskit eivät koske vain pakettien käyttäjiä, vaan myös niiden kehittäjiä. Pakettien kehittäjien tulisi käyttäjien tavoin kiinnittää huomiota pakettiensa turvallisuuteen ja niiden riippuvuuksiin muista paketeista. Saatavilla on työkaluja, joilla projektien Node.js-pakettien haavoittuvuuksia voidaan kuitenkin sujuvasti havaita ja korjata. Näitä työkaluja voidaan vaivatta ottaa osaksi DevOps-toimintamallin mukaisia jatkuvia operaatioita ja versionhallintaa, jolloin kehittäminen säilyy tehokkaana ja turvallisena.

Lähteet

1 Mediaosakeyhtiö Frantic. Verkkoaineisto. Finder. <https://www.finder.fi/Uusmediapalvelut/Mediaosakeyhti%C3%B6+Frantic/Helsinki/yhteystiedot/133843>. Luettu 26.11.2018.

2 Ota yhteyttä. Verkkoaineisto. Frantic. <https://www.frantic.com/fi/yhteystiedot>. Luettu 26.11.2018.

3 About npm. Verkkoaineisto. Npm. <https://www.npmjs.com/about>. Luettu 8.10.2018.

4 Pierotti, Luke. 2017. Node.js: What is it and how does it work? Verkkoaineisto. Medium. <https://medium.com/@lukepierotti/node-js-what-is-it-and-how-does-it-work-e3e83d054662>. 29.8.2017. Luettu 10.10.2018.

5 Chrzanowksa, Natalia. 2017. Why to Use Node.js: Pros and Cons of Choosing Node.js for Back-end Development. Verkkoaineisto. Netguru. <https://www.netguru.co/blog/pros-cons-use-node.js-backend>. 23.3.2017. Luettu 10.10.2018.

6 Hüttermann, Michael. 2012. DevOps for Developers. E-kirja. Springer.

7 Hering, Mirco. 2018. Continuous everything in DevOps…what is the difference between CI, CD, CD…? Verkkoaineisto. Accenture. <https://www.accenture.com/us-en/blogs/blogs-continuous-everything-devops>. 23.9.2018. Luettu 14.10.2018.

8 Chacon, Scott & Straub, Ben. 2014. Pro Git. E-kirja. Springer.

9 Goff-Dupont, Sarah. Super-powered continuous-delivery with Git. Verkkoaineisto. Atlassian. <https://www.atlassian.com/continuous-delivery/why-git-and-continuous-delivery-are-super-powered>. Luettu 16.10.2018.

10 Core Concepts for Beginners. Verkkoaineisto. Travis CI. <https://docs.travis-ci.com/user/for-beginners>. Luettu 10.1.2019.

11 Vulnerabilities. Verkkoaineisto. National Vulnerability Database. <https://nvd.nist.gov/vuln>. Luettu 10.3.2019.

12 Decan, Alexandre; Mens, Tom & Constantinou, Eleni. 2018. On the impact of security vulnerabilities in the npm package dependency network. Verkkoaineisto. The ACM Digital Library. <https://dl.acm.org/citation.cfm?id=3196401>. 28.5.2018. Luettu 18.10.2018.

13 Grander, Danny. 2018. Malicious code found in npm package event-stream downloaded 8 million times in the past 2.5 months. Verkkoaineisto. Snyk Blog. <https://snyk.io/blog/malicious-code-found-in-npm-package-event-stream>. 26.11.2018. Luettu 7.2.2019.

14 Details about the event-stream incident. 2018. Verkkoaineisto. The npm Blog. <https://blog.npmjs.org/post/180565383195/details-about-the-event-stream-incident>. 27.11.2018. Luettu 7.2.2019.

15 Grander, Danny & Tal, Liran. 2018. A Post-Mortem of the Malicious event-stream backdoor. Verkkoaineisto. Snyk Blog. <https://snyk.io/blog/a-post-mortem-of-the-malicious-event-stream-backdoor>. 6.12.2018. Luettu 7.2.2019.

16 Npm Command Line Interface. 2010. Npm, Inc.

17 Npm Acquires ^Lift Security and the Node Security Platform. 2018. Verkkoaineisto. Npm, Inc. Medium. <https://medium.com/npm-inc/npm-acquires-lift-security-258e257ef639>. 10.4.2018. Luettu 21.10.2018.

18 Han, Miju. 2017. Introducing security alerts on GitHub. Verkkoaineisto. The Github Blog. <https://blog.github.com/2017-11-16-introducing-security-alerts-on-github>. 16.11.2017. Luettu 30.9.2018.

19 Armerding, Taylor. 2017. What is CVE, its definition and purpose? Verkkoaineisto. CSO. <https://www.csoonline.com/article/3204884/what-is-cve-its-definition-and-purpose.html>. 10.7.2017. Luettu 10.3.2019.

20 Github data services. 2017. Github, Inc.

21 Security. Verkkoaineisto. Snyk. <https://snyk.io/docs/security>. Luettu 4.2.2019.

22 Shmukler, Igor. 2018. Comparing npm audit with Snyk. Verkkoaineisto. NearForm. <https://www.nearform.com/blog/comparing-npm-audit-with-snyk>. 17.8.2018. Luettu 4.2.2019.

23 Github pull requests. 2010. Github, Inc.

24 Fixing vulnerabilities. Verkkoaineisto. Snyk. <https://snyk.io/docs/fixing-vulnerabilities>. Luettu 4.2.2019.

25 Snyk Open Source Security Platform. 2015. Snyk.

26 Snyk Command Line Interface. 2015. Snyk.

27 Renovate. Verkkoaineisto. Renovate Docs. <https://renovatebot.com/docs>. Luettu 4.2.2019.

28 Renovate. Verkkoaineisto. Github. <https://github.com/marketplace/renovate>. Luettu 5.2.2019.

29 Should you Pin your Javascript Dependencies? Verkkoaineisto. Renovate Docs. <https://renovatebot.com/docs/dependency-pinning>. Luettu 5.2.2019.

30 Arkins, Rhys. New feature – Github vulnerability alerts. Verkkoaineisto. Renovate Blog. <https://renovatebot.com/blog/github-vulnerability-alerts>. Luettu 26.3.2019.

31 Shareable Config Presets. Verkkoaineisto. Renovate Docs. <https://renovatebot.com/docs/config-presets>. Luettu 5.2.2019.

32 Github personal access tokens. 2013. Github, Inc.

33 Rouse, Margaret & Rosencreance, Linda. 2018. How to prepare for the emerging threats to your systems and data. Verkkoaineisto. TechTarget. <https://searchsecurity.techtarget.com/definition/cross-site-scripting>. Päivitetty 26.2.2018. Luettu 26.3.2019.