Aatu Martinlaakso

SecDevOpsin hyödyntäminen tietoturvauhkien torjunnassa

Metropolia Ammattikorkeakoulu

Insinööri (AMK)

Ohjelmistotuotanto

Insinöörityö

6.11.2023

Tiivistelmä

Tekijä:	Aatu Martinlaakso
Otsikko:	SecDevOpsin hyödyntäminen tietoturvauhkien torjunnassa
Sivumäärä:	42 sivua
Aika:	6.11.2023

Tutkinto:	Insinööri (AMK)
Tutkinto-ohjelma:	Ohjelmistotuotanto
Ammatillinen pääaine:	Tieto- ja viestintätekniikka
Ohjaajat:	Lehtori Simo Silander

Tämän insinöörityön tarkoituksena oli tehdä syväluotaus tietoturvaan, jossa selvitetään, mitkä ovat ajankohtaisia tietoturvauhkia ohjelmistotuotannossa ja miten tietoturvaa priorisoivaa SecDevOps-kehitystapaa voi hyödyntää niiden torjunnassa. Työ tehtiin tutkimalla aiheeseen liittyvää kirjallisuutta ja verkkolähteitä.

Ensiksi työssä käsiteltiin erilaisia tietoturvauhkia ja DevOps-sovelluskehitykseen liittyviä tietoturvahaasteita. Tämän jälkeen esiteltiin näiden haasteiden ratkaisemiseen tarkoitetut DevSecOps- ja SecDevOps-kehitysmallit.

Insinöörityössä syvennyttiin SecDevOpsiin ja keskityttiin sen käytännön merkitykseen ohjelmiston kehitysvaiheessa. Tässä syventymisessä käytiin läpi SecDevOps-kehitysprosessin eri vaiheissa sovelluksen tietoturvallisuuden parantamiseen käytettyjä käytäntöjä ja työkaluja.

Lopputuloksena ilmeni, että SecDevOpsista voi olla hyötyä tietoturvariskien ehkäisyssä, mutta sillä on myös haittapuolia, joiden takia se ei välttämättä aina ole paras valinta. Tietoturvan laittaminen etusijalle voi hidastaa kehitysprosessia. Sovelluskehityksessä tietoturvallisuuden ja tehokkuuden välillä täytyy usein tehdä kompromissi. Lisäksi SecDevOps vaatii kehittäjiltä hyvää tietoturvaosaamista. SecDevOps voi sopia hyvin projekteihin, joissa tietoturvallisuus on ehdottoman tärkeää.

Avainsanat: devops, secdevops, tietoturva

Abstract

Author:	Aatu Martinlaakso
Title:	SecDevOps in Mitigation of Information Security Threats
Number of Pages:	42 pages
Date:	6 November 2023
Degree:	Bachelor of Engineering
Degree Programme:	Information and Communication Technology
Professional Major:	Software Engineering
Supervisors:	Simo Silander, Senior Lecturer

The purpose of the study was to do a deep dive into information security that aims to identify what are currently relevant information security threats in software production and how SecDevOps development approach can be used to mitigate them. The study is based on an examination of source materials collected from relevant literature and online sources.

Initially, the thesis addressed various security threats and security challenges associated with software development in DevOps. After that, it introduces DevSecOps and SecDevOps development models as solutions to these challenges.

The thesis delves into SecDevOps, focusing on its practical significance in the development phase. This involved going through the methods used to improve security of an application in different steps of the SecDevOps software development life cycle.

The outcome of the study revealed that SecDevOps can be useful in mitigating security risks, but it also has some downsides. Prioritizing security can slow down the development process. In software development it is often necessary to compromize between security and efficiency. SecDevOps also requires developers to have sufficient security expertise. SecDevOps may be well-suited for projects where security is crucial.

Keywords: secdevops, devops, information security

Sisällys

Lyhenteet

CD: *Continuous Deployment*. Jatkuva julkaisu, automatisoitu uusien sovellusversioiden tuotantoon vieminen.

CI: *Continuous integration*. Jatkuva integraatio, lähdekoodiin tehtyjen muutosten automatisoitu yhdistäminen osaksi projektia.

CI/CD: Jatkuvan integraation ja julkaisun yhdistelmä.

DAST: *Dynamic application security testing*. Haavoittuvuuksien etsiminen sovelluksesta ajonaikaisesti työkaluilla.

DDoS: *Distributed denial-of-service*. Hajautettu eli useasta lähteestä tehtävä palvelunestohyökkäys.

IaC: *Infrastructure-as-Code*. Palvelininfrastruktuurin hallinnan automatisointi ja sen määrittely koodina.

IoT: *Internet of things*. Esineiden internet, verkkoyhteyttä käyttävät älykkäät kodinkoneet ja muut laitteet.

OWASP: *Open Web Application Security Project*. Yhteisöprojekti, joka tarjoaa työkaluja ja muita resursseja websovellusten tietoturvallisuuden parantamiseen.

SaC: *Security-as-Code*. Automaattisten työkalujen käyttö koodin turvallisuuden varmistamiseen.

SAST: *Static application security testing*. Haavoittuvuuksien etsiminen sovelluksen lähdekoodia analysoivilla työkaluilla.

SCA: *Software Composition Analysis*. Sovelluksen osien analysointi esimerkiksi haavoittuvuuksien löytämiseksi.

1 Johdanto

Digitalisaation edetessä tietotekniikka koskettaa yhä useampaa osa-aluetta elämässämme. Sen seurauksena tietojärjestelmiin kohdistuvien hyökkäysten määrä ja niiden aiheuttamat vahingot suurenevat. Vuonna 2023 kyberrikollisuuden ennustetaan aiheuttavan maailmanlaajuisesti jopa 8 biljoonan (8 000 000 000 000) dollarin kulut [1]. Tietoturvan merkitys siis kasvaa koko ajan.

Joidenkin arvioiden mukaan jopa 84 prosenttia kyberhyökkäyksistä kohdistuu sovelluskerrokseen [2]. Jotta voitaisiin varmistaa, että ohjelmistot ja sovellukset ovat turvallisia eikä niissä ole haavoittuvuuksia, on ehdottoman tärkeää huolehtia niiden tietoturvallisuudesta jo kehitysvaiheessa. Tietoturvan huomioiminen ohjelmistoa kehittäessä vaatii ohjelmistokehittäjiltä hyvää tietoturvaosaamista. Ohjelmiston tietoturvallisuus ei voi kuitenkaan olla pelkästään yksittäisten kehittäjien vastuulla, vaan lisäksi tarvitaan tietoturvallisuutta edistävien kehitystapojen käyttöä.

Ohjelmistoversioiden nopeaa julkaisua ja tuotantoon viemistä edistävän DevOps-kehitysmallin käytön yleistyessä ohjelmistokehityksessä tulee varmistaa, että ohjelmistojen tietoturvallisuuteen ja kiinnitetään nopeasta tahdista huolimatta tarpeeksi huomiota. DevOps-ympäristöissä käytetään myös paljon uusia tekniikoita, joihin liittyvät mahdolliset tietoturvariskit tulee tiedostaa. DevOps-ympäristössä kehitettävien sovellusten tietoturvallisuuden varmistamiseen varten onkin kehitetty uusia malleja, kuten DevSecOps ja SecDevOps.

Tämän opinnäytetyön tarkoituksena on selvittää, mitkä ovat ajankohtaisia tietoturvauhkia ja miten DevOpsiin turvallisuutta lisäävä SecDevOps-kehitystapa soveltuu niiden torjuntaan. Ensiksi käsitellään tietoturvaympäristöä ja erilaisia tietoturvauhkia. Tämän jälkeen perehdytään DevOpsiin, siihen mahdollisesti liittyviin tietoturvaongelmiin ja niiden ratkaisuihin. Erityisesti työssä syvennytään SecDevOpsiin ohjelmistokehittäjän näkökulmasta ja pohditaan sen etuja tieto-

turvauhkien torjunnassa. Työ tehdään tutkimalla aiheeseen liittyvästä kirjallisuudesta ja verkkolähteistä kerättyä lähdemateriaalia.

2 Tietoturvauhat

Tietojärjestelmät ovat nykyään tärkeä osa yhteiskunnan toimintaa. Ihmisten ja varsinkin organisaatioiden tietojärjestelmissä säilytetään valtavaa määrää arvokasta tietoa. Tämän takia ne ovat myös oivallinen kohde rikollisille ja muille hyökkääjille. Tietojärjestelmiä pitää siis suojata, etteivät hyökkääjät pääse niihin ja niissä säilytettyyn tietoon käsiksi. Tärkeä konsepti tietoturvassa on ”CIA-triad”. Se on malli, joka muodostuu kolmesta tietoturvan peruspilarista, jotka tulee tietoa turvatessa varmistaa.

- Saatavuus (availability). Tiedon pitää olla saatavilla, kun sitä tarvitaan.

- Eheys (integrity). Tietoon saa tehdä vain hyväksyttyjä muutoksia ja väärien muutosten tulee olla havaittavissa.

- Luottamuksellisuus (confidentiality). Tiedon tulee olla vain oikeutettujen henkilöiden käsiteltävissä.

Tietojärjestelmiin hyökkäyksiä tekeviä pahantahtoisia toimijoita on monenlaisia ja heillä on erilaisia motivaatioita. Yleinen motivaatio on raha. Kyberrikolliset tienaavat myymällä organisaatioiden järjestelmistä varastettuja tietoja, kiristämällä organisaatioita ja yksityishenkilöitä. Jotkut hakkerit tekevät elantonsa kehittämällä haittaohjelmia muiden käytettäväksi. Toiset enemmän tai vähemmän pätevät hakkerit taas haluavat todistella taitojaan tai haluavat vain aiheuttaa vahinkoa huvikseen. Haktivistit yrittävät saada toiminnallaan aikaan yhteiskunnallisia muutoksia. Valtiolliset toimijat tekevät kyberhyökkäyksiä toisiin valtioihin häiritäkseen yhteiskunnan toimintaa tai vakoillakseen tärkeitä henkilöitä. Kaikkia näitä toimijoita kuitenkin yhdistää se, että toiminnallaan ne uhkaavat järjestelmien tietoturvaa. [3.]

Tietotekniikan alalla kaikki kehittyy jatkuvasti, eikä tietoturvaympäristö ole poikkeus. Kehitys on seurannut enimmäkseen jo pitkään jatkuneita kehityssuuntia, mutta esimerkiksi Ukrainan sota on vaikuttanut tietoturvaympäristöön mm. lisäämällä valtiollisten toimijoiden ja haktivistien tekemien hyökkäysten määrää huomattavasti. Viime vuosina sekä valtiollisten että rikollisten toimijoiden nollapäivähaavoittuvuuksiin ja tuotantoketjuun kohdistuvat hyökkäykset ovat lisääntyneet. Kiristysohjelmahyökkäykset ovat jatkaneet yleistymistä ja niillä on ollut vakavia seurauksia. Myös palvelunestohyökkäysten yleistyminen on jatkunut ja niistä on tullut suurempia. [4.]

Tässä luvussa käsitellään ajankohtaisia tietoturvauhkia ja tietoturvaympäristön kehitystä. Erilaisia tietoturvauhkia on niin suuri määrä, että kaikkien mahdollisten uhkien käsittely olisi insinöörityön puitteissa mahdotonta, joten käsiteltäväksi on valikoitu uhat, jotka ovat mahdollisimman merkittäviä tai erityisesti ohjelmistokehityksessä huomioon otettavia.

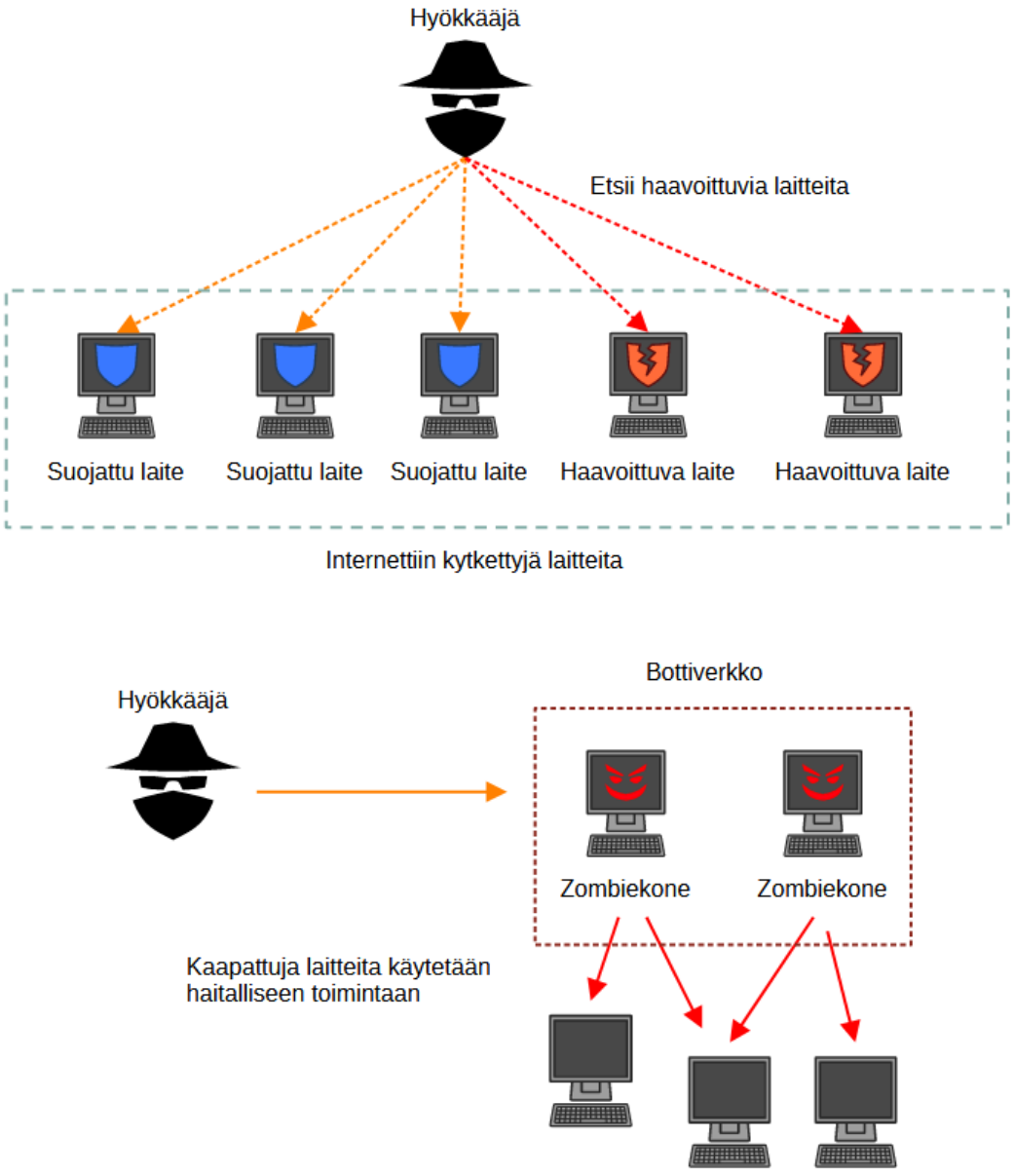
2.1 Haittaohjelmat

Haittaohjelmiksi luokitellaan kaikki ohjelmat, joiden tarkoitus on tehdä tietojärjestelmän eheyttä, luottamuksellisuutta tai saatavuutta vaarantavia luvattomia toimia. Haittaohjelmia on monenlaisia ja niiden luokitteluun on useita tapoja. Ne voivat esimerkiksi varastaa laitteelta tietoja kuten salasanoja tai pankkitunnuksia, ladata muita haittaohjelmia, näyttää käyttäjille mainoksia, käyttää laitetta kryptovaluuttojen louhimiseen tai ottaa laitteen kokonaan hyökkääjän hallintaan.

Haittaohjelmien torjunta on jatkuvaa kilpajuoksua. Voidakseen kiertää jatkuvasti kehittyviä torjuntakeinoja täytyy haittaohjelmien kehittäjien parannella haittaohjelmiaan jatkuvasti. Uusia haittaohjelmia tulee siis koko ajan ja vanhatkin kehittyvät. Monet yleiset haittaohjelmat ovat vuosia vanhoja, mutta huomattavasti muuttuneet alkuperäisestä versiosta. [4.]

Rikolliset käyttävät laitteen hallintaansa ottavia haittaohjelmia tehdäkseen useista hallitsemistaan laitteista muodostuvia bottiverkkoja. Osana bottiverkkoja

näitä ”zombiekoneita” käytetään muuhun haitalliseen toimintaan, kuten haittaohjelmien levittämiseen edelleen tai palvelunestohyökkäyksiin. Kuva 1 havainnollistaa, kuinka hyökkääjä tekee internetiin kytketyistä haavoittuvista laitteista osan bottiverkkoaan. Rikolliset myös myyvät bottiverkkojaan muiden rikollisten käyttöön. [5.]



Kuva 1: Bottiverkon muodostaminen

2.1.1 Kiristysohjelmat

Ransomware eli kiristysohjelma on haittaohjelma, joka salaa kryptografisesti kaikki laitteella säilytetyt tiedostot niin, ettei uhri pääse niihin enää käsiksi ilman avainta. Salauksen purkamiseen tarvittavasta avaimesta hyökkääjä yleensä vaatii uhria maksamaan lunnaat, mutta tiedostoja ei välttämättä saa takaisin, vaikka maksaisi hyökkääjälle. Kiinnijäämisen välttämiseksi maksu vaaditaan useimmiten vaikeasti jäljitettävillä kryptovaluutoilla. [6.]

Kiristysohjelmahyökkäykset ovat vuosikausien ajan olleet kasvava uhka ja uusia hyökkäyksiä aletaan tehdä nopeasti lähes jokaisen julkistetun yleisen haavoittuvuuden kautta. Hyökkääjien käyttämät kiristystekniikat ovat myös kehittyneet. Pelkän tiedostojen salaamisen tai tuhoamisen lisäksi jotkut kyberrikollisryhmät ovat alkaneet käyttää kiristysohjelmia, jotka siirtävät uhrin tiedot kiristäjien palvelimelle ja vuotavat ne nettiin, jos lunnaita ei makseta. Organisaatioiden järjestelmissä on paljon arkaluontoisia ja salaisia tietoja, joten vuodettujen tietojen joukossa voi olla tärkeitä liikesalaisuuksia tai luottamuksellisia asiakastietoja. [4.]

Viranomaisiin tai muuhun tärkeään kohteeseen kohdistuvalla kiristysohjelmahyökkäyksellä voi olla vakavia seurauksia. Tällainen hyökkäys voi jopa johtaa kuolemaan. Ensimmäinen uutisoitu kiristysohjelmasta johtuva kuolemantapaus kävi syyskuussa 2020, kun Düsseldorfin yliopistollisessa sairaalassa kuoli potilas sairaalaan kohdistuneen kiristysohjelmahyökkäyksen takia [7].

2.1.2 Älypuhelinten haittaohjelmat

Nykyään lähes jokaisen taskusta löytyy älypuhelin, ja käytämme niitä päivittäin netissä selailuun, kommunikointiin ja verkkopankkiasiointiin ja moniin muihin asioihin. Älypuhelimet ovat tärkeä osa elämäämme. Siksi niistä löytyy paljon arvokasta tietoa, joten myös hyökkääjät ovat kiinnostuneita niistä. Päästäkseen käsiksi älypuhelimiin ja näiden tietoihin rikolliset käyttävät haittaohjelmia, joten älypuhelinten yleistyessä ovat yleistyneet myös niille tehdyt haittaohjelmat. Ku-

ten tietokoneille kohdistetut haittaohjelmat, myös älypuhelinten haittaohjelmien tyypillisesti varastavat käyttäjätunnuksia, näyttävät käyttäjille mainoksia tai salaavat laitteen tiedot ja kiristävät käyttäjiltä rahaa. Yleensä näitä haittaohjelmia levitetään tekstiviestien välityksellä. [8.]

Vuoden 2022 kesäkuussa Europol otti hallintaansa Flubot-haittaohjelman hallintainfrastruktuurin yhdentoista valtion yhteisessä poliisioperaatiossa lopettaen ohjelman leviämisen. Suomessakin laajalle levinnyt Flubot oli Android-laitteille kohdistettu haittaohjelma, jota levitettiin tekstiviesteihin laitettujen linkkien välityksellä. Laitteelle päästyään Flubot varasti käyttäjiltä tunnuksia pankkitileille ja kryprovaluuttalompakkoihin. Lisäksi se lähetti kaikkiin puhelimeen tallennettuihin numeroihin Flubottia edelleen levittäviä huijausviestejä. [9.]

Suurinta osaa älypuhelinten haittaohjelmista ei käytetä johonkin tiettyyn henkilöön tai organisaatioon kohdistettuihin hyökkäyksiin, vaan niiden tarkoituksena on yksinkertaisesti levitä mahdollisimman moneen laitteeseen. Poikkeuksiakin kuitenkin löyty. Yksi viime vuosina huomiota herättänyt älypuhelinhaittaohjelma on israelilaisen NSO Groupin kehittämä rikollisuuden ja terrorismin vastaiseen toimintaan tarkoitettu Pegasus. Pegasus on aiheuttanut kohua, sillä sitä on myyty useille valtioille, jotka ovat käyttäneet sitä kyberaseena aktivistien, toimittajien ja poliitikkojen vakoiluun. Pegasus on vakoiluohjelma, joka kykenee tarttumaan sekä Android- että iOS-laitteisiin. Tartunnan jälkeen vakoiluohjelma tarkkailee kaikkea käyttäjän toimintaa laitteella, se pystyy varastamaan salasanoja, lukemaan viestejä, seuraamaan laitteen sijaintia ja käyttämään laitteen kameraa ja mikrofonia. Alun perin sitä levitettiin haitallisen linkin sisältävien viestien välityksellä, mutta myöhemmät versiot pystyvät tartuttamaan laitteen tuntemattoman nollapäivähaavoittuvuuden avulla ilman, että käyttäjän tarvitsee tehdä mitään. [10.]

2.1.3 IoT-haittaohjelmat

Internetissä ei ole vain perinteisiä tietokoneita ja älypuhelimia vaan yhä enemmän muitakin laitteita. Kaikenlaiset laitteet, esimerkiksi termostaatit, tulostimet,

valvontakamerat ja monenlaiset älykkäät kodinkoneet voivat nykyään käyttää verkkoyhteyttä ollakseen yhteydessä toisiinsa ja ovat usein yhteydessä julkiseen internetiin. Tällaisia laitteita kutsutaan ”esineiden internetiksi”, englanniksi internet of things, josta käytetään lyhennettä IoT. [11.] Nämä IoT-laitteet kuitenkin sisältävät mikroprosessorin ja niissä on riisuttu versio jostain tavanomaisesta, yleensä Unix-pohjaisesta käyttöjärjestelmästä. Usein tällaiset laitteet ovat myös joko kehittäjien tai käyttäjien huolimattomuuden vuoksi heikosti suojattu. [12.]

IoT-laitteiden heikko suojaus tekee niistä helpon kohteen hyökkääjille, jotka haittaohjelmien avulla tekevät niistä osan bottiverkkojaan. Laitteiden suuri määrä mahdollistaa massiivisten bottiverkkojen rakentamisen. Hyökkääjät skannaavat jatkuvasti internetiä etsien haavoittuvia laitteita. Usein laitteen suojauksen murtamiseen riittää, että tietää laitteen valmistajan asettaman vaihtamatta jääneen heikon oletussalasanan. Haittaohjelmatartunta jää helposti huomaamatta, koska laitetta hallitseva haittaohjelma ei aiheuta laitteen omistajalle havaittavaa muutosta laitteen toimintaan. [12.]

2.2 Käyttäjän manipulointi

Käyttäjän manipulointi, englanniksi social engineering on sellaista toimintaa, jossa hyökkääjä manipuloi uhria päästäkseen käsiksi tietoon tai järjestelmään, johon hyökkääjällä ei ole käyttöoikeutta. Tähän lukeutuu moninaista toimintaa pankkitunnusten kalastelusta roskapostilla viranomaisen esittämiseen puhelimessa. Käyttäjän manipulointi on yleinen tapa saada järjestelmään alustava pääsy, joka mahdollistaa muun haitallisen toiminnan, kuten haittaohjelmien asentamisen tai tietojen varastamisen. Jopa 60 prosentissa tietomurroista hyökkääjä on käyttänyt käyttäjän manipulointia. [4.]

Tietojenkalastelu eli englanniksi phishing on yleinen käyttäjän manipuloinnin muoto, jossa kohteelle lähetetään huijausviestejä sähköpostilla, tekstiviestillä tai muulla tavalla. Viestien tavoitteena on joko saada käyttäjä syöttämään käyttäjätunnuksensa ja salasanansa joltain muulta sivustolta näyttävälle huijaussivustol-

le tai saada käyttäjä avaamaan haittaohjelman laitteelle tartuttava linkki tai tiedosto. [13.] Johonkin tiettyyn henkilöön kohdistettua tietojenkalastelua kutsutaan englanniksi spear-phishingiksi ja sen kohdistuessa johonkin korkeassa asemassa olevaan henkilöön sitä kutsutaan whalingiksi [4].

Käyttäjän manipuloinnissa hyväksikäytetään ihmisten heikkouksia ja yritetään saada ihminen tekemään virhe. Hyökkääjät vetoavat tunteisiin kuten empatiaan tai pelkoon, yrittävät saada uhrin toiminaan kiireessä tai he saattavat käyttää hyväksi vaikka uhrin laiskuutta. Käyttäjän manipuloinnin torjuminen on vaikeaa, koska manipuloimalla järjestelmään käsiksi pääsevää ihmistä ohitetaan kaikki ohjelmalliset estot. Inhimillistä elementtiä ei voi täysin poistaa kaikista tietojärjestelmistä ja pahantahtoiset toimijat keksivät aina uusia temppuja manipulointiin. Toimiva keino ehkäistä käyttäjän manipulointia on kouluttaa ihmiset noudattamaan hyviä tietoturvakäytäntöjä, sekä onnistuneen hyökkäyksen aiheuttamien haittojen vähentämiseksi varmistaa, että pääsyoikeuksia hallitaan asiallisesti. [13.]

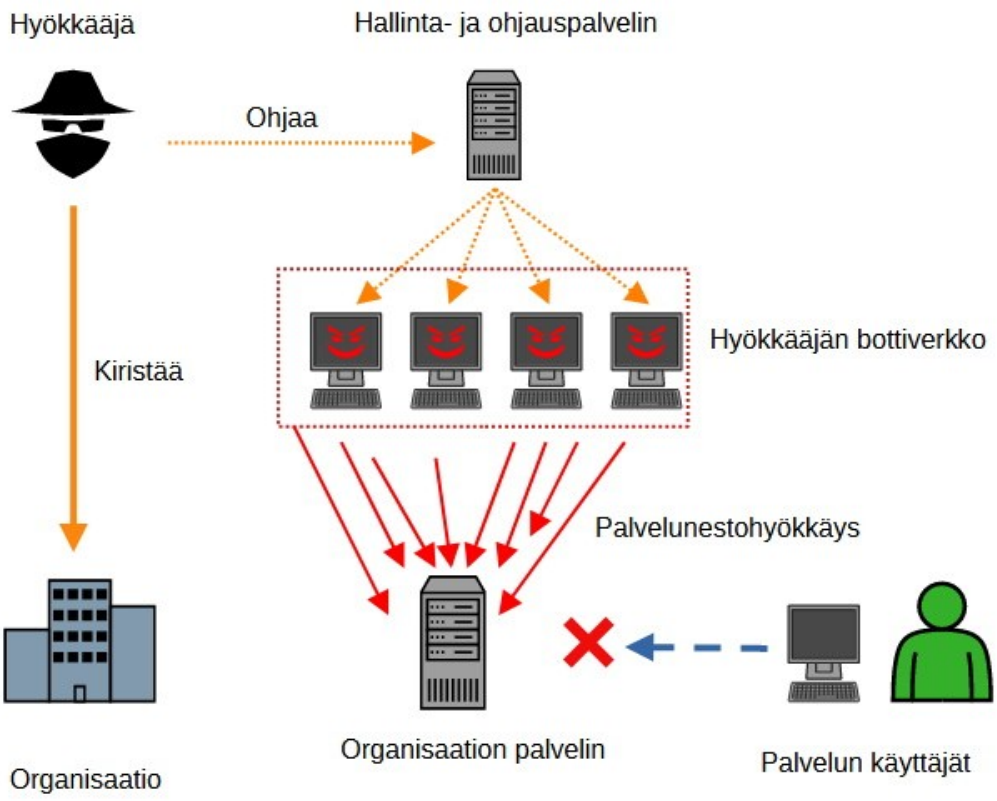
2.3 Palvelunestohyökkäys

Palvelunestohyökkäys, englanniksi denial of service attack on hyökkäys, jossa hyökkääjä estää tai hankaloittaa palvelun käyttöä. Kyseessä on siis tiedon saatavuuteen kohdistuva hyökkäys. Yleensä palvelunestohyökkäys tehdään kuormittamalla kohde suurella määrällä verkkoliikennettä tai lähettämällä kohteelle suuri määrä sellaisia pyyntöjä, joiden käsittely vaatii paljon resursseja, kuten muistia tai prosessorin käyttöä. Suurimmassa osassa palvelunestohyökkäyksistä haitallinen liikenne kohteeseen tulee useammasta lähteestä, silloin kyseessä on hajautettu palvelunestohyökkäys (englanniksi distributed denial of service attack, DdoS attack). Hajautettujen palvelunestohyökkäysten tekemiseen käytetään yleensä bottiverkkoja. Rikolliset myyvät palvelunestohyökkäyksiä palveluna, joten niiden tekemiseen ei välttämättä tarvitse paljoa osaamista. [14.]

Palvelunestohyökkäykset vaikuttavat haitallisesti tietojärjestelmien toimintaan ja palveluiden saatavuuteen. Ne ovatkin yksi merkittävimmistä tietojärjestelmiin

kohdistuvista uhista. Suurin osa hyökkäyksistä on melko pieniä, mutta heikosti suojatuista IoT-laitteista ja reitittimistä muodostuvat suuret, jopa satoja tuhansia laitteita käsittävät bottiverkot mahdollistavat massiivisten hyökkäysten tekemisen. DdoS-suojausta tarjoava Cloudflare on raportoinut hyökkäyksiä, joissa kohteeseen on lähetetty liikennettä jopa lähes 2 terabittiä sekunnissa. [4.]

Rikolliset käyttävät palvelunestohyökkäyksiä organisaatioiden kiristämiseen. Organisaatiota joko uhataan hyökkäyksellä, jos he eivät maksa rikollisille, tai organisaation tietojärjestelmiä tai verkkoa vastaan aloitetaan palvelunestohyökkäys, jonka lopettamisesta vaaditaan maksua. Kuva 2 havainnollistaa palvelunestohyökkäyksen käyttöä kiristämiseen. Jos hyökkääjät ovat onnistuneet murtautumaan organisaation järjestelmiin, saatetaan kiristyksissä lisäksi käyttää kiristyshaittaohjelmia tai organisaatiota voidaan uhkailla järjestelmistä saatujen tietojen vuotamisella. [4.]



Kuva 2: Kiristäminen palvelunestohyökkäyksellä

Palvelunestohyökkäyksiä käytetään myös osana kybersodankäyntiä. Osana hyökkäystä Ukrainaan vuonna 2022 Venäjältä tehtiin Ukrainan hallinnollisiin ja taloudellisiin organisaatioihin kohdistettuja voimakkaita palvelunestohyökkäyksiä. Myöhemmin sodan molempiin osapuoliin on kohdistunut useita suuria hyökkäyksiä. DdoS-hyökkäyksistä on tullut osa molempien puolien kybersodankäyntiä. [4.]

2.4 Toimitusketjuhyökkäys

Supply-chain attack eli toimitusketjuhyökkäys on sellainen hyökkäys, jossa hyökkääjä ottaa kohteekseen organisaation toimitusketjusta jonkin heikommin suojatun luotetun kolmannen osapuolen päästäkseen käsiksi itse organisaation

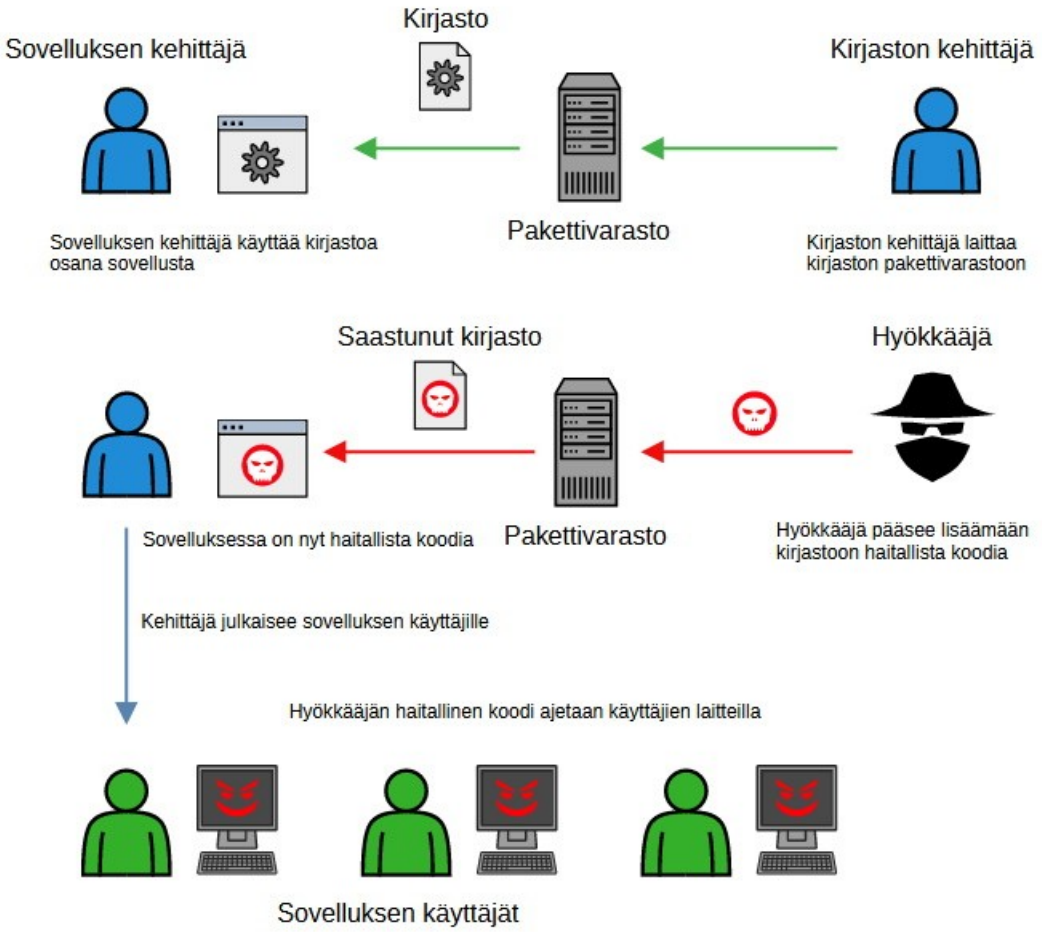
tietojärjestelmiin. Ohjelmistotuotannossa tällainen toimitusketjuhyökkäys voi kohdistua esimerkiksi johonkin ohjelmiston riippuvuuksista, kuten ohjelman käyttämään kirjastoon. [15.]

Haavoittuvuus yleisesti käytetyssä riippuvuudessa voi altistaa hyökkäyksille suuren määrän järjestelmiä, joilla käytetään riippuvuutta hyödyntäviä ohjelmistoja. Näin kävi vuoden 2021 lopulla, kun yleisestä lokitiedostojen pitoon käytetystä Log4j-kirjastosta löytyi vakava haavoittuvuus, jonka avulla hyökkääjä pystyi suorittamaan laitteella omaa koodiaan. Haavoittuvuus koski valtavaa määrää järjestelmiä ja sitä alettiin hyväksikäyttää heti. Koska haavoittuvuus oli kirjastossa, joka oli osa monia ohjelmia, näiden ohjelmien käyttäjät eivät voineet helposti tietää, oliko jokin heidän käyttämistään ohjelmista haavoittuvainen vai ei. Log4j-haavoittuvuus oli yksi historian vakavimmista. [16.]

Erilaisia tapoja saastuttaa osa ohjelmiston toimitusketjusta on paljon. Ohjelmistokehityksessä on yleistä, että riippuvuuksienhallintaa varten on käytössä jokin paketinhallintajärjestelmä, kuten JavaScript-kehityksessä käytetty NPM tai Python-kehityksessä käytetty pip. Paketinhallintajärjestelmien käytön yleisyyden takia ne ovat myös hyökkääjien kohteena. Hyökkääjä voi esimerkiksi laittaa paketinhallintajärjestelmän käyttämään julkiseen pakettivarastoon haitallista koodia sisältävän kirjaston, joka on nimetty samalla tavalla kuin jokin ohjelmaa kehittävän organisaation sisäisesti kehittämä ja käyttämä kirjasto. Huonosti konfiguroitu paketinhallintajärjestelmä lataa hyökkääjän paketin ja käyttää sitä oikean kirjaston sijaan. Toinen yleinen tapa on ”typosquatting”, jossa hyökkääjä laittaa pakettivarastoon paketin, joka on melkein saman niminen suositun kirjaston kanssa. Kun kehittäjä kirjoittaa vahingossa paketin nimen väärin, hän lataakin vahingossa hyökkääjän paketin. [17.]

Saastuneen kirjaston avulla hyökkääjä voi sisällyttää ohjelmistoon mitä tahansa haitallista koodia. Tällainen saastunutta kirjastoa käyttävä ohjelma voi levittää haittaohjelman kaikille sen käyttäjille, suositun ohjelman tapauksessa uhreja voi olla valtava määrä. Kuva 3 havainnollistaa saastuneen kirjaston avulla tehtyä toimitusketjuhyökkäystä. Nykyään kehitettävät sovellukset ovat yleensä moni-

mutkaisia ja koodia käytetään uudelleen paljon, joten ohjelmistoilla on usein paljon riippuvuuksia. Riippuvuuksien suuri määrä tekee toimitusketjuhyökkäysten ehkäisemisestä vaikeaa.



Kuva 3: Toimitusketjuhyökkäys

2.5 Tietoturva ohjelmistotuotannossa

Jotta rikolliset, haittaohjelmat tai muut hyökkääjät pääsevät järjestelmään sisälle tekemään haitallisia toimia, täytyy niiden hyväksikäyttää joko käyttäjän tekemää virhettä tai jotakin järjestelmästä löytyvää haavoittuvuutta. Ohjelmistokehittäjillä on vastuu kehittää ohjelmistoistaan turvallisia, sillä ohjelmistojen haavoittuvuu-

det vaarantavat käyttäjien tietoturvan. Pahantahtoiset toimijat käyttävät kaapattuja laitteita palvelunestohyökkäyksiin ja muuhun haitalliseen toimintaan, joten haavoittuvat järjestelmät aiheuttavat harmia myös ulkopuolisille.

Vaikka haavoittuvuus korjattaisiin nopeasti, niin jälkikäteen korjattu haavoittuvuus muodostaa edelleen uhan ohjelmiston vanhempaa versiota syystä tai toisesta käyttäville käyttäjille. Haavoittuvuuden paljastuessa jossain yleisessä ohjelmassa hyökkääjät alkavat välittömästi skannata internetistä haavoittuvia järjestelmiä automatisoitujen työkalujen avulla, eivätkä kaikki käyttäjät millään ehdi päivittää ohjelmistoaan turvalliseen versioon. Pahimmassa tapauksessa kyseessä on nollapäivähaavoittuvuus, eikä turvallista versiota ole vielä edes saatavilla.

Ohjelmiston tekeminen täysin tietoturvalliseksi on mahdotonta, mutta asianmukaiset kehitystavat ehkäisevät vakavia haavoittuvuuksia tehokkaasti.

3 DevOps ja tietoturva

3.1 Mikä DevOps

DevOps on viimeisen vuosikymmenen aikana yleistynyt lähestymistapa tietoteknisten palvelujen kehittämiseen. Sen keskeisin periaate on kehitys- ja operointitiimien välisen yhteistyön ja kommunikaation lisääminen ja tehostaminen. Sana ”DevOps” onkin yhdistelmä sanoista ”development” ja ”operations”. DevOpsissa kehittäjät usein osallistuvat ohjelmiston ylläpitoon tuotannossa ja operaatiotiimin palaute pyritään ottamaan huomioon kehityksessä. Tiimien välisen yhteistyön parantamisen lisäksi DevOpsiin liittyy yleensä kehitysprosesseja tehostavien teknologioiden ja työkalujen käyttö.

Agile-kehitystavoista inspiroituneessa DevOpsissa ohjelmistoa kehitetään jatkuvassa syklissä, jossa tehdään tiheään tahtiin paljon pieniä muutoksia, jotka viedään tuotantoon mahdollisimman nopeasti. Jatkuvaa pienten muutosten tekemistä koodiin, niiden testaamista ja yhdistämistä kutsutaan jatkuvaksi integraatioksi (englanniksi continuous integration, CI) ja näiden viemistä tuotantoon kutsutaan jatkuvaksi julkaisuksi (englanniksi continuous delivery tai continuous deployment, CD). [18.] DevOpsissa sovelluksen elinkaari on siis sykli, ja se koostuu useista toistuvista vaiheista. Kuvan 4 esittämässä tyypillisessä DevOps-syklissä on kahdeksan vaihetta:

- Plan. Suunnitteluvaihe, jossa sovellusta suunnitellaan.

- Code. Koodausvaihe, jossa kehittäjät kirjoittavat koodia suunnitteluvaiheen perusteella.

- Build. Koontivaihe, jossa koodiin tehdyt muutokset lisätään osaksi projektia ja sovellus kootaan.

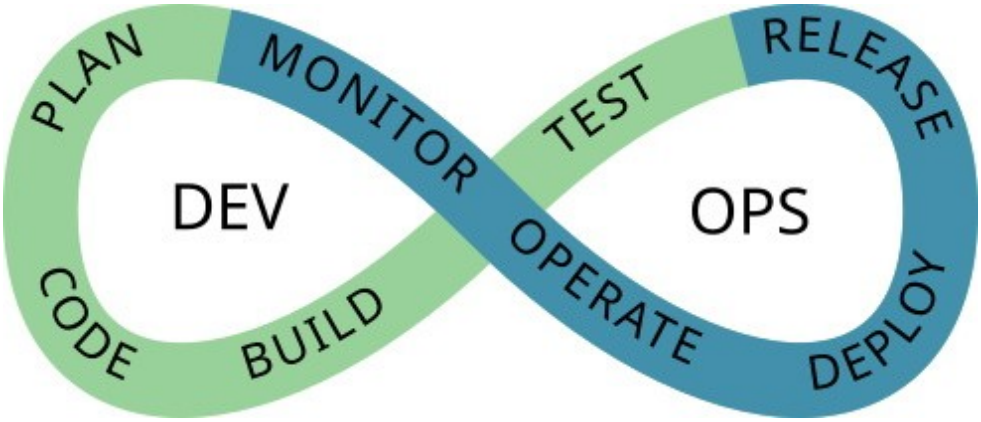
- Test. Testausvaihe, jossa sovellukselle tehdään testejä, joilla varmistetaan sen toimivan oikein.

- Release. Julkaisuvaihe, jossa testit läpäissyt sovellus odottaa tuotantoon viemistä.

- Deploy. Käyttöönottovaihe, jossa sovellus viedään tuotantoympäristöön.

- Operate. Operointivaihe, jossa sovellus on käytössä tuotannossa. Sovelluksen toiminnasta kerätään jatkuvasti dataa.

- Monitor. Valvontavaihe, jossa sovelluksen käytöstä saatu data ja palaute kerätään, jotta sitä voidaan hyödyntää seuraavassa suunnitteluvaiheessa. [19.]



Kuva 4: DevOps-sykli.

Jatkuvan integraation ja julkaisun (CI/CD) tehostamiseen käytetään monia työkaluja, jotka automatisoivat jatkuvaan integraatioon ja julkaisuun liittyviä tehtäviä. Näistä CI/CD-työkaluista pyritään luomaan tehokkaasti automatisoitu CI/CD-putki (englanniksi CI/CD-pipeline). CI/CD-putken avulla uudet ohjelmaver-

siot etenevät kehityksestä tuotantoon mahdollisimman vähällä vaivalla. Tehokkuuden lisäksi jatkuvan integraation ja julkaisun automatisoinnissa on muitakin etuja. Kun ihmisen osuutta tässä prosessissa vähennetään, tapahtuu myös vähemmän inhimillisiä virheitä. [20.]

3.2 DevOpsin ongelmat

DevOpsin käyttäminen tehostaa ohjelmistokehitystä, mutta sillä voi olla myös haittapuolia. Yksi näistä haittapuolista on mahdolliset tietoturvariskit, sillä alun perin DevOpsissa ei otettu tietoturvaa huomioon tarpeeksi. Perinteisen vesiputousmallin keinot huolehtia ohjelmistojen tietoturvallisuudesta, joissa ohjelmiston turvallisuus varmistetaan kehityksen loppuvaiheessa soveltuvat DevOpsin nopeaan tahtiin, jatkuviin muutoksiin ja tehokkuuskeskeisyyteen huonosti ja uusien teknologioiden käyttö tuo myös uusia riskejä.

3.2.1 Puutteelliset turvallisuuskäytännöt

Tiimien välisen yhteistyön lisääminen ja jaetut vastuut ovat olennainen osa DevOpsia. DevOps-ympäristöissä kehittäjien tehtävät ovat aikaisempaa monipuolisempia, jonka takia heillä pitää olla oikeudet päästä käsiksi useampiin järjestelmiin kuin ennen. Myös monet ohjelmat, kuten prosessien automatisoinnissa käytetyt työkalut ja sovelluksen muodostavat mikropalvelut tarvitsevat monia pääsy- ja käyttöoikeuksia. Näiden oikeuksien hallinnasta voi nopeasti tulla vaikeaa ja sekavaa. Tämä on ongelma, sillä ylimääräiset käyttöoikeudet muodostavat aina tietoturvariskin. Oikeuksien hallinnasta määräävät turvallisuuskäytännöt ovat tarpeellisia siihen liittyvien epäselvyyksien välttämiseksi.

DevOps-ympäristöissä kehitetyt modernit sovellukset perustuvat usein mikropalveluarkkitehtuuriin. Mikropalveluarkkitehtuurissa sovellus on jaettu useisiin pieniin palveluihin, joista jokainen vastaa vain pienestä määrästä toiminnallisuutta. Tällaisissa sovelluksissa tapahtuu paljon kommunikaatiota eri tietokantojen, rajapintojen ja palveluiden välillä. Sovelluksen osien välisen kommunikaa-

tion varmentamiseen käytetään salasanoja, tokeneita ja muita todennustietoja. Virheet näiden todennustietojen käsittelyssä ovat yleinen tietoturvariski, esimerkiksi kovakoodattu salasana tai API-token päätyy helposti vääriin käsiin.

Virheet käyttöoikeuksien hallinnassa ja tunnistetietojen käsittelyssä voivat aiheuttaa tietoturvariskejä. Ilman tarvittavia käyttöoikeuksia ja todennusta koskevia turvallisuuskäytäntöjä ei voida varmistua tunnistetietojen turvallisesta käsittelystä ja siitä, ettei henkilöillä tai prosesseilla ole tarpeettomia oikeuksia. Tietoturvariskien välttämiseksi sovelluskehityksessä tulisi ottaa huomioon, että projektille on määrätty kunnolliset turvallisuuskäytännöt. [18.]

3.2.2 Teknologia

DevOpsissa yleisesti käytettyjen kehitys- ja julkaisutahtia nopeuttavien teknologioiden, kuten säiliöinnin (käyttöjärjestelmätason virtualisointi) ja pilvipalveluiden käyttö on hyödyllistä, mutta ne voivat myös aiheuttaa uusia turvallisuusriskejä, varsinkin ilman tarvittavaa osaamista. Väärin konfiguroitu virtuaalikone tai säiliö voi vuotaa tietoja, joita ei pitäisi tai jopa sallia taitavalle hyökkääjälle pääsyn isäntäkoneelle. Oikein käytettynä säiliöinti yhdistettynä suoritusympäristöjen automaattiseen luomiseen, konfigurointiin ja hallintaan Ansiblen, Cheffin tai Puppetin kaltaisilla IaC-työkaluilla (Infrastructure-as-Code) voi kuitenkin parantaa sovelluksen tietoturvallisuutta. [21.]

Sovellusten lisääntyvä monimutkaisuus voi tehdä tietoturvallisuuden varmistamisesta vaikeaa. Varsinkin mikropalveluarkkitehtuurilla tehdyt sovellukset saattavat olla toiminnaltaan äärimmäisen monimutkaisia ja vaikeasti ymmärrettäviä. Mahdollisten haavoittuvuuksien löytäminen vaikeutuu, jos ymmärrys sovelluksen toiminnasta on puutteellista. Lisäksi eri kielillä ja kehyksillä tehtyihin mikropalveluihin liittyy erilaisia tietoturvariskejä ja tietoturvallisuuden varmistaminen vaatii näiden kaikkien ymmärtämistä.

Nykyään omien palvelinten ylläpitämisen sijaan monet ohjelmistoyritykset käyttävät pilvipalveluita, kuten Amazonin AWS:ää ja Microsoftin Azurea. Pilvipalve-

luiden ja IaaS:n (Infrastructure-as-a-Service) tai PaaS:in (Platform-as-a-Service) käyttö poistaa tarpeen huolehtia palvelininfrastruktuurista ja verkoista itse. Suuret pilvipalvelualustat tarjoavat paljon käytännöllisiä ominaisuuksia ja mahdollistavat helpon skaalautumisen. Kuten muidenkin teknologioiden kohdalla, väärin konfiguroituna myös pilvipalvelu on turvallisuusriski. Siksi pilvipalveluita käyttäessä on tärkeää tietää, mitkä tehtävät ovat asiakkaan ja mitkä ovat palvelun-tarjoajan vastuulla. [18.]

3.2.3 Nopea tahti

Viikon tai parin Agile-sprinttien sijaan DevOpsissa uusia versioita viedään tuotantoon jatkuvalla tahdilla, usein kymmeniä versioita päivässä. Prosesseista poistetaan pullonkauloja ja automaatiota käytetään tehokkuuden maksimoimiseksi. Turvallisuuden varmistaminen vie aikaa, ja se saatetaan nähdä esteenä tehokkuudelle.

Lean-periaatteet ovat vaikuttaneet DevOpsiin paljon. Projekteissa on usein tapana tehdä aluksi mahdollisimman yksinkertainen versio tuotteesta (Minimum viable product, MVP), joka yritetään tehdä ja saada asiakkaiden käyttöön mahdollisimman nopeasti. Ominaisuuksia suunnitellaan, lisätään ja karsitaan käytön aikana saadun palautteen ja metriikan perusteella jatkuvasti iteroiden. Suunnitelmat ja vaatimukset siis muuttuvat jatkuvasti, turvallisuustiimeille voi olla vaikeaa tietää missä vaiheessa tällaiselle jatkuvasti muuttuvalle suunnitelmalle voi tehdä turvallisuusarvioita ja uhkamallinnuksia. [18.]

3.3 Ratkaisut

Jotta kehitettävän ohjelman tietoturvallisuus toteutuisi paremmin, tarvitaan uusia kehitystapoja, joissa turvallisuutta on siirretty vasemmalle. DevOpsissa vasemmalle siirtäminen (shift left) tarkoittaa prosessin siirtämistä tehtäväksi aikaisemmassa vaiheessa ohjelmiston elinkaarta [22]. Eli sen sijaan, että ohjelmiston turvallisuudesta huolehditaan vasta elinkaaren lopussa, turvallisuuden vasemmalle

siirtävässä kehitysmallissa tulee turvallisuus integroida ohjelmiston elinkaareen aikaisempiin vaiheisiin.

Ratkaisuksi DevOpsiin liittyviin turvallisuusongelmiin on kehitetty useita eri kehittämistapoja, käytetyimmät näistä ovat DevSecOps ja hieman uudempi SecDevOps. Molemmat ovat tapoja siirtää turvallisuus vasemmalle ja integroida se paremmin ohjelmiston elinkaareen. DevSecOpsin ja SecDevOpsin ero on melko epäselvä, sillä sanoja käytetään usein synonyymeinä ja DevSecOps terminä on huomattavasi SecDevOpsia tunnetumpi ja sitä usein käytetään tarkoittamaan kaikkia tapoja lisätä DevOpsin turvallisuutta. Kyse on kuitenkin eri asioista. [23.]

Tässä insinöörityössä keskitytään lähinnä SecDevOpsiin. Termien epäselvän käytön takia täytyy tarkentaa, että tässä työssä SecDevOpsilla tarkoitetaan sellaista DevOps:in kaltaista lähestymistapaa ohjelmistotuotantoon, jossa tärkeimpänä prioriteettina on ohjelmiston tietoturvallisuuden varmistaminen.

3.3.1 DevSecOps

DevSecOps on ohjelmistokehitystapa, jossa turvallisuus on integroitu osaksi ohjelmiston elinkaarta, pyrkien silti säilyttämään tehokkuus mahdollisimman hyvin. Ohjelmiston tietoturvallisuutta parantavia toimenpiteitä on lisätty osaksi elinkaaren vaiheita, esimerkiksi kehittäjien tekemien testien aikana voidaan testata myös turvallisuutta ja turvallisuustiimi työskentelee koko ajan ohjelman turvallisuuden varmistamiseksi yhdessä kehitystiimin kanssa.

DevSecOps ei kuitenkaan varmista ohjelmiston turvallisuutta aina niin hyvin kuin voisi toivoa. Kriittiset haavoittuvuudet tulevat DevSecOpsissa korjatuksi, mutta kehittäjillä on kuitenkin paine saada ohjelman seuraava versio julkaistua mahdollisimman nopeasti, joten lievempien haavoittuvuuksien korjaus saatetaan laiminlyödä. [24.] Turvallisuustiimin resurssit eivät välttämättä riitä kaikkeen, mitä heiltä vaaditaan, jos muiden tiimien tietoturvaosaaminen on puutteellista [25].

3.3.2 SecDevOps

SecDevOpsissa turvallisuus on otettu kehitysprosessissa tärkeimmäksi prioriteetiksi. Se on siis siirretty vielä enemmän vasemmalle kuin DevSecOpsissa. Tietoturvallisuuden varmistaminen menee kaiken muun edelle jokaisessa ohjelmiston elinkaaren jokaisessa vaiheessa. Ohjelmasta suunnitellaan alusta asti turvallinen määrittelemällä turvallisuuskäytännöt heti kehityksen alussa ja ottamalla uhkamallinnus osaksi suunnittelua. Kun ohjelma on suunniteltu turvalliseksi, virheitä ja haavoittuvuuksia tarvitsee korjata vähemmän, koska niitä tehdään vähemmän. Samalla tietoturvallisuuden lisäksi myös ohjelman luotettavuus paranee. [24.]

Automatisointi kuuluu oleellisesti DevOpsiin, joten SecDevOpsissa myös turvallisuuden varmistaminen pyritään automatisoimaan mahdollisimman hyvin. Ohjelman turvallisuuden analysointiin käytetään työkaluja, jotka voi integroida osaksi jo käytössä olevia DevOps-työkaluja, esimerkiksi CI/CD-putkea, kehitysympäristöä ja versionhallintaa. Näin ohjelman turvallisuuden analysointi tulee osaksi jo olemassa olevia elinkaaren vaiheita.

Kun ohjelmiston tietoturvallisuus asetetaan kehitysprosessissa etusijalle, tulee jokaisen tiimin työskennellä turvallisuuden edistämiseksi ja jokaisen kehittäjän täytyy ottaa siitä vastuuta. Tämä voi aluksi herättää vastustusta, mikä voi aiheuttaa ongelmia. Esimerkiksi tehokkuudesta välittävälle kehittäjälle voi olla vaikeaa joustaa siitä, vaikka ohjelmiston tietoturvallisuus sitä vaatisi. SecDevOpsin käyttöönotto vaatii siis muutosta kulttuurillista muutosta tietoturvallisempaan suuntaan. Kehitettäessä ohjelmistoja tietoturvallisuus muun edellä kehittäjiltä vaaditaan myös tavallista enemmän tietoturvaosaamista. Kehittäjät tarvitsevat tietoturvaosaamisensa parantamiseksi koulutusta. SecDevOps varmistaa ohjelmiston tietoturvallisuuden tehokkaasti, mutta varsinkin käyttöönottovaiheessa se voi vaatia paljon rahaa ja vaivaa. [25.]

4 SecDevOps käytännössä

Sovelluksen kehittäminen tietoturvalliseksi on monimutkainen prosessi. Tietoturvariskien minimointiin keskittyvään SecDevOpsiin liittyykin paljon käytännönasioita ja konsepteja. Jokainen SecDevOps-ympäristö on rakennettu vastaamaan erilaisiin tarpeisiin, mutta SecDevOps-lähestymistapaan liittyy tiettyjä ydinkäytäntöjä, joita tässä luvussa käydään läpi.

4.1 Organisaation turvallisuus

Kokonaisvaltainen tietoturvallisuuden varmistaminen edellyttää, että tietoturvallisuuteen panostetaan koko organisaation tasolla. SecDevOpsissa vastuu tietoturvasta ei ole vain tietyillä tietoturva-asiantuntijoilla, vaan kaikilla kehittäjillä. Jotta SecDevOps-kehitysprosessi saadaan toimimaan tehokkaasti ja turvallisesti, täytyy organisaatiossa ruokkia kulttuuria, jossa kaikki tiimit osallistuvat sovelluksen turvallisuuden varmistamiseen yhdessä. Tämän mahdollistamiseksi täytyy kehittäjillä olla riittävästi tietoturvaosaamista. Tietoturvaosaamista pitää tarvittaessa lisätä tarjoamalla koulutusta. [26.]

Selkeät kehitysprosessia koskevat turvallisuuskäytännöt vähentävät epäselvyyksiä ja auttavat kehittäjiä sovelluksen turvallisuuden parantamisessa [26]. Myös sovellusta kehittävän organisaation yleiset turvallisuuskäytännöt lisäävät sovelluksen tietoturvallisuutta, vaikka ne eivät koskisi kehitysprosessia suoraan. Jos organisaatiolla on heikko tietoturva, mahdollinen hyökkääjä voi päästä käsiksi salaisiin tietoihin tai pahimmillaan tehdä sovelluksen käyttäjiin kohdistuvan toimitusketjuhyökkäyksen.

Käyttäjätilien ja käyttöoikeuksien huolimaton hallinnointi aiheuttaa tietoturvariskejä. Tietoturvariskien välttämiseksi on tärkeää, että organisaatiossa on määritelty hyvät käytännöt käyttäjätilien ja oikeuksien hallinnointiin. Käyttäjillä tulisi olla vain ehdottomasti tarpeelliset oikeudet, varsinkin täysiä järjestelmänvalvojan

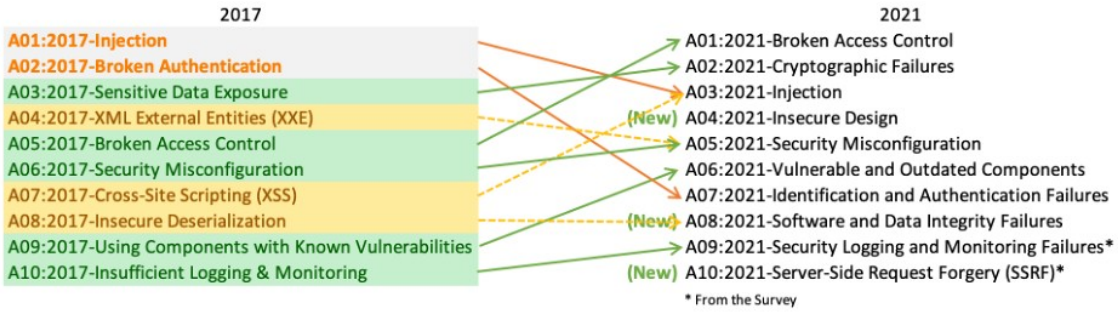
oikeuksia tarvitsevat vain harvat. Ylimääräiset oikeudet pahantahtoisella henkilöllä mahdollistavat järjestelmien väärinkäytön ja osaamattomissa käsissä niillä voi aiheuttaa vahingossakin paljon haittaa. Kaikkien tulisi käyttää päivittäisessä käytössä tavallista käyttäjätiliä silloin, kun korkeammille oikeuksille ei ole tarvetta. Korotetuilla oikeuksilla varustetun tilin käyttö päivittäisessä käytössä on tietoturvariski. Esimerkiksi sähköposteja lukiessa huijausviestin liitteen kautta tarttunut haittaohjelma aiheuttaa suurempaa tuhoa, jos sen avannut käyttäjä on kirjautunut järjestelmänvalvojan tilille. [3.]

Käyttäjätilien hallinnoinnin lisäksi organisaation tietojärjestelmien turvalliseen konfigurointiin ja organisaatiossa käytettyjen ohjelmistojen turvallisuuden varmistamiseen tulisi olla kunnolliset turvallisuuskäytännöt [3]. Organisaation tietojärjestelmissä tapahtuvasta toiminnasta tulee kerätä lokeja, joiden avulla haitallista toimintaa on helpompi havaita ja myöhemmin analysoida. Lokien keräämiseen, säilyttämiseen ja analysointiin tulee olla asianmukaiset käytännöt. [21.]

4.2 Sovelluksen suunnittelu turvalliseksi

SecDevOpsissa tietoturva on tärkein prioriteetti, joten SecDevOps-ympäristöissä kehitetyt sovellukset tulee suunnitella alusta asti turvallisiksi. Tietoturvariskien minimointi aloitetaan heti projektin alussa määrittelemällä käytännöt muun muassa turvallisen koodin kirjoittamiseen, sovelluksen turvallisuuden testaamiseen ja työkalujen käyttöön [24].

Turvallisten kirjastojen ja sovelluskehysten käyttö on perusta sovelluksen suunnittelulle turvalliseksi [18]. Kuvassa 5 on OWASP:in Top 10 lista kriittisimmistä websovellusten haavoittuvuuksista [27]. Listalla on monta haavoittuvuutta, jotka olisivat vältettävissä käyttämällä turvallisia kirjastoja.



Kuva 5: OWASP Top Ten [27]

Sovelluksen turvatoimintoihin, kuten kirjautumiseen tai syötteiden validaatioon voi olla hyvä käyttää valmiita toteutuksia. Monille ohjelmointikielille on olemassa valmiita sovelluskehyksiä, joista löytyy toteutukset esimerkiksi todentamiseen, kryptografiaan ja istunnonhallintaan. Esimerkiksi Java-sovellusta kehitettäessä OWASP:in listalla kuudentena olevat tunnistautumisen ja todentamisen virheistä johtuvat haavoittuvuudet voi välttää käyttämällä Spring Security -sovelluskehystä. Tunnetun turvallisen sovelluskehyksen käyttäminen on yleensä turvallisempaa kuin omien toteutusten, ja se tekee turvallisen sovelluksen kehittämisestä helpompaa kehittäjille. [21.]

Toinen samalla tavalla helposti korjattava haavoittuvuus OWASP:in listalla on kolmannelta sijalta löytyvät injektioon perustuvat haavoittuvuudet [27]. Injektiohaavoittuvuudet, kuten XSS (cross-site scripting) johtuvat syötteiden ja tulosteiden puutteellisesta sanitaatiosta ja validaatiosta. Injektiohaavoittuvuudet voi websovellusten kehityksessä torjua tehokkaasti käyttämällä sovelluskehystä, kuten Angular ja React, jotka varmistavat syötteiden ja tulosteiden turvallisuuden automaattisesti. [28.]

Kolmannen osapuolen kirjastojen ja kehysten hyödyntäminen sovelluksen kehittämisessä on myös tehokasta. Jos sovelluksen tarvitsemasta toiminnosta on olemassa jo valmis toteutus, ei sen kehittämiseen itse kannata kuluttaa aikaa. Kolmannen osapuolen riippuvuuksien käyttö voi kuitenkin aiheuttaa tietoturva-

riskejä, sillä riippuvuuksien haavoittuvuudet siirtyvät itse sovellukseen. Toimitusketjuhyökkäysten välttämiseksi sovelluksen riippuvuuksien turvallisuus täytyy varmistaa. Riippuvuuksien turvallisuuden varmistamisen voi tehdä automaattisesti työkaluilla. [18.]

Yksi SecDevOpsin ydinperiaatteista on turvallisuus koodina, englanniksi security as code, josta käytetään lyhennettä SaC. SaC tarkoittaa sovelluksen tietoturvallisuuden parantamista käyttämällä sovelluksen elinkaaren eri vaiheisiin integroituja automatisoituja työkaluja haavoittuvuuksien löytämiseen. [26.] Testaamista varten sovellukselle täytyy määritellä turvallisuusvaatimukset ja testisuunnitelma. Projektien alussa tulisi tehdä riskiarvio, jonka avulla turvallisuusvaatimukset määritellään. Kun turvallisuusvaatimukset on määritelty, voidaan tehdä testisuunnitelma. Testisuunnitelma kertoo millä tavoilla ja missä elinkaaren vaiheissa testataan, täyttääkö sovellus sille määritellyt turvallisuusvaatimukset. [3.]

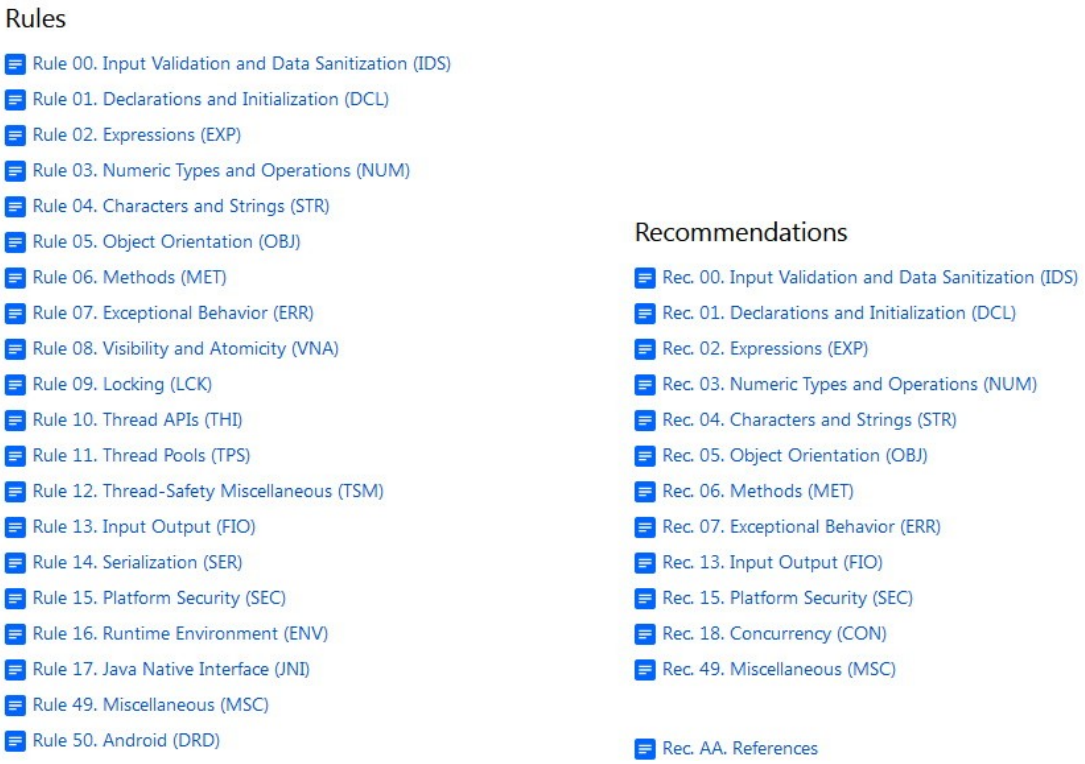
4.2.1 Turvallinen koodi

Osana sovelluksen suunnittelua turvalliseksi SecDevOpsissa tulee ottaa kirjoitetun koodin turvallisuus huomioon [26]. Hyvien käytäntöjen mukaisesti kirjoitettu koodi on tietoturvallisen sovelluskehityksen ensimmäinen askel. Sovelluksesta on mahdotonta tehdä turvallinen, jos kehittäjät eivät kirjoita turvallista koodia. Turvallisen koodin kirjoittaminen edellyttää riittävää ymmärrystä käytetyistä ohjelmointikielistä ja niihin liittyvistä parhaista käytännöistä. Jokaisella kielellä on omat erityispiirteensä ja turvallisuusriskinsä. [21.] Projektin suunnitteluvaiheessa tulisi selvittää hyvät käytännöt ja yleiset haasteet käytettyjen ohjelmointikielten osalta ja varmistaa, että kehittäjät osaavat ne. Riittävän osaamisen varmistamiseksi kehittäjiä tulee tarvittaessa kouluttaa. [18.]

Yksi esimerkki tietoturvariskejä aiheuttavasta huonosta käytännöstä on monesta ohjelmointikielestä löytyvän eval-funktion käyttö. Muun muassa PHP- ja JavaScript-kielistä löytyvä eval on funktio, joka suorittaa sille argumenttina annetun merkkijonon ohjelmakoodina. Yhdistettynä käyttäjän syötteen puutteelliseen

validointiin eval-funktion käyttö voi aiheuttaa haavoittuvuuden, joka mahdollistaa haitallisen koodin suorittamisen. [29.]

Turvallisiin ohjelmointikäytäntöihin on olemassa standardeja. Esimerkiksi CERT Coding Standards sisältää turvalliset standardit C-, C++-, Java-, Perl- ja Android-ohjelmointiin. [21.] Standardit ovat hyvin kattavia. Kuvassa 6 on Java-ohjelmoinnin CERT-standardin sisällysluettelo. Standardissa määritellään oikeat käytännöt muun muassa muuttujien esittelyyn, metodien kirjoittamiseen, säikeiden käyttöön ja merkkijonojen käsittelyyn Java-ohjelmoinnissa. [30.]



Kuva 6: SEI CERT Oracle Coding Standard for Java sisällysluettelo [30]

Monet näistä säännöistä on tarkoitettu estämään sovellusta toimimasta odottamattomasti tietyissä tilanteissa. Yksi tällainen sääntö on lausekkeita koskeva sääntö ”EXP00-J. Do not ignore values returned by methods”, joka määrää, että

metodien paluuarvot tulee aina huomioida. Jotkin metodit ilmoittavat onnistumisesta tai epäonnistumisesta paluuarvossaan. Jos metodin paluuarvoa ei huomioida, ei voida tietää, onnistuiko se. Kun kutsutun metodin epäonnistumista ei huomioida, suorituksen jatkuessa sovellus voi toimia väärällä tavalla. [31.] Sääntöihin on merkitty, kuinka vakavaa niiden rikkominen on. Korkean riskin sääntöihin kuuluu esimerkiksi syötteitä ja tulosteita koskevien sääntöjen joukosta löytyvä sääntö ”FIO08-J. Distinguish between characters or bytes read from a stream and -1”, joka määrää, miten tietovirrasta lukiessa täytyy erottaa luettu tavu tai merkki tietovirran päättymisestä kertovasta palautusarvosta -1. Esimerkkikoodissa 1 tiedostovirrasta luetulle tavulle tehdään tyyppimuunnos ennen virran päättymisen tarkistamista. Jos tiedostosta luetaan arvo 0xFF, tyyppimuunnoksen takia se on mahdotonta erottaa virran päättymistä merkitsevästä arvosta -1 ja silmukka loppuu ennenaikaisesti. [32.]

```
FileInputStream in;
// Initialize stream
byte data;
while ((data = (byte) in.read()) != -1) {
  // ...
}
```

Esimerkkikoodi 1: Tavujen lukeminen tietovirrasta virheellisesti

Esimerkkikoodissa 2 virhe korjataan tekemällä tyyppimuunnos vasta, kun virran päättyminen on tarkistettu. Sääntö on tarpeellinen, koska virheet palautusarvon tulkinnassa tietovirrasta lukiessa voivat saada ohjelman toimimaan ennakoimattomalla tavalla ja ovat joissain tapauksissa aiheuttaneet haavoittuvuuksia. [32.]

```
FileInputStream in;
// Initialize stream
int inbuff;
byte data;
while ((inbuff = in.read()) != -1) {
  data = (byte) inbuff;
  // ...
}
```

Esimerkkikoodi 2: Tavujen lukeminen tietovirrasta oikein

Koodiin tehtyjen muutosten vertaisarviointi (code review) ennen niiden hyväksymistä on yleinen tapa sovelluskehityksessä. Kun kehittäjät osaavat tunnistaa tietoturvariskejä aiheuttavat huonot käytännöt, voi koodin vertaisarviointeja käyttää myös koodin turvallisuuden varmistamiseen. Näin ainakin selkeimmät haavoittuvuudet voidaan havaita ja korjata varhaisessa vaiheessa jo ennen koodin viemistä CI/CD-putkeen. [18.]

Ohjelman lähdekoodin laadun varmistamiseksi on olemassa työkaluja, jotka skannaavat koodia ja havaitsevat siitä mahdolliset virheet. Monet tällaiset työkalut voi integroida suoraan kehitysympäristöön, jolloin kehittäjät saavat reaaliaikaista palautetta kirjoittamastaan koodista. Koska jokaisella ohjelmointikielellä on omat parhaat käytäntönsä, kehittäjien täytyy muistaa suuri määrä sääntöjä. Kehitysympäristöön integroidut työkalut tekevät hyvien käytäntöjen seuraamisesta helpompaa kehittäjille. Projekteissa tulisi etsiä sopivat työkalut kieli- ja tapauskohtaisesti. [18.]

Sovelluksen kehityksessä käytetyillä ohjelmointikielillä on vaikutusta sovelluksen turvallisuuteen. Tämä tulisi huomioida SecDevOpsissa kieliä valitessa. Esimerkiksi muistinhallinnassa tehtyjen virheiden aiheuttamien haavoittuvuuksien välttämiseksi voi olla hyvä harkita jonkin muistinhallinnan turvallisuuden varmistavan ohjelmointikielen, kuten Javan tai Rustin käyttöä [28]. Virheet muistinhallinnassa aiheuttavat C:n kaltaisilla matalamman tason kielillä ohjelmoidessa vaikeasti löydettäviä bugeja ja haavoittuvuuksia. Rust voi olla hyvä valinta tapauksissa, joissa esimerkiksi Java ei ole tarpeeksi nopea. Rust on viime aikoina paljon huomiota saanut kieli, jonka tarkoituksena on olla yhtä suorituskykyinen kuin C++, mutta automaattisesti turvallisella muistinhallinnalla. C- ja C++-kielillä ohjelmoidessa muistinhallinnassa on helppo tehdä virheitä, eivätkä niiltä välty aina edes kokeneet ohjelmoijat. Rust-ohjelmoinnissa on mahdotonta tehdä samanlaisia virheitä, sillä Rust ei salli virheellistä muistin käyttöä. [33.]

4.2.2 Uhkamallinnus

Uhkamallinnus on prosessi jonkin suojeltavaan asiaan kohdistuvien uhkien tunnistamiseksi. Sitä voidaan soveltaa monella alalla. Ohjelmistokehityksessä se on tärkeä osa tietoturvallisen sovelluksen kehitysprosessia. Sen tarkoituksena on parantaa kehitettävän sovelluksen tietoturvallisuutta tunnistamalla mahdolliset tietoturvauhat, arvioimalla niiden aiheuttamat riskit ja määrittelemällä keinot niiden torjumiseksi. Prosessin tuloksena on uhkamalli, esitys sovelluksen tietoturvaan liittyvästä tiedosta. [34.]

Uhkamallinnusta voi tehdä eri laajuuksilla ja sen tekemiseen on monia tapoja. Tapauksesta riippuen uhkamallinnusta voi tehdä esimerkiksi korkean tason arkkitehtuurille tai pelkästään yhdelle sovelluksen osalle. Uhkamallinnuksen tekemiseen on monia tapoja dokumentaatiota vaativista formaaleista prosesseista kevyeen vapaaseen brainstormaukseen ja sen eri vaiheisiin on olemassa monia työkaluja ja apuvälineitä. [21.]

OWASP tarjoaa ohjatun lähestymistavan sovellusten uhkamallinnukseen. OWASP:in uhkamallinnusprosessi sisältää kolme vaihetta. Ensimmäisessä vaiheessa sovellus puretaan osiin, jotta sen toimintaa voi analysoida. Sovelluksen analysointi sisältää seuraavat asiat:

- Käydään läpi sovelluksen tavalliset käyttötapaukset.
- Tunnistetaan tulokohdat, joita hyökkääjät voivat käyttää.
- Tunnistetaan kohteet, joista hyökkääjä olisi kiinnostunut.
- Määritellään, mitä oikeuksia vaaditaan sovelluksen eri osien käyttöön.

Sovelluksen osista muodostetaan sen tiedonkulkua havainnollistava tietovuokaavio (data-flow-diagram, DFD), joka on hyödyllinen seuraavassa vaiheessa. Seuraavassa vaiheessa tunnistetaan sovellukseen kohdistuvia uhkia. Uhkien

tunnistamiseen on olemassa valmiita malleja, kuten STRIDE. Tunnistettujen uhkien aiheuttamat riskit arvioidaan, myös riskien arviointiin kannattaa käyttää jotain valmista mallia. Viimeisessä vaiheessa määritellään, mitä torjuntakeinoja uhkien torjuntaan käytetään. Uhkien torjuntaa voi priorisoida niiden aiheuttaman riskin perusteella. Kaikissa vaiheissa kerätyt tiedot otetaan talteen, ja tuloksena on kattava dokumentti. [35.]

Kattavan uhkamallinnuksen tekemiseen tarvitaan suunnitteludokumentteja, joita DevOps-ympäristöissä ei minimaalisen suunnittelun vuoksi usein ole. DevOpsissa projekteja ei yleensä suunnitella kovin yksityiskohtaisesti etukäteen ja suunnittelua tehdään iteratiivisesti, joten suunnitelma muuttuu jatkuvasti. Kattavan uhkamallinnuksen tekeminen projektin alussa ei ole DevOps-ympäristöissä yleensä siis mahdollista. [18.] Uhkamallinnus tulisikin siis ottaa mukaan suunnitteluprosessia. Projektin alussa tulisi tehdä karkeiden suunnitelmien perusteella uhkamalli. Tätä mallia tulee parannella jatkuvasti projektin edetessä. Näin uhkamalli kehittyy torjumaan uusia sovelluksen kehittyessä syntyviä uhkia. [34.]

4.3 Haavoittuvuuksien löytäminen

Mahdolliset haavoittuvuudet täytyy löytää, jotta ne voidaan korjata ja poistaa niiden aiheuttama riski. Sovelluksesta voi etsiä haavoittuvuuksia analysoimalla sen rakennetta tai lähdekoodia ja penetraatiotestaamalla sitä. Sovellukset ovat kuitenkin usein hyvin monimutkaisia ja kaikkia haavoittuvuuksia ei ole mahdollista löytää manuaalisesti etsimällä. Haavoittuvuuksien etsimiseen automaattisesti on olemassa monenlaisia työkaluja. Automatisoituja työkaluja hyödynnetään SecDevOpsissa manuaalisen testaamisen ohella, jotta mahdollisimman suuri osa sovelluksen haavoittuvuuksista löytyisi.

4.3.1 SAST

SAST (Static application security testing) on ohjelman lähdekoodin staattista analyysiä, jossa etsitään haavoittuvuuksia ja virheitä. SAST-työkalut helpottavat

virheiden huomaamista varhaisessa vaiheessa. Koska SAST-työkalut analysoivat sovelluksen lähdekoodia, ne pystyvät näyttämään hyvin tarkasti, missä kohdassa koodia löydetyt virheet ovat. [36.] SAST-työkalut voivat tuottaa melko paljon virheellisiä positiivisia havaintoja, joiden tarkistaminen vie aikaa. SAST-työkalujen tehokas käyttö voi vaatia niiden asetusten hienosäätämistä virheellisten havaintojen vähentämiseksi. [18.]

SAST-työkaluja voi käyttää kehitysympäristöön integroituna helpottamaan kehittäjiä välttämään yleisiä virheitä ja kirjoittamaan parempaa koodia. Kehitysympäristöön integroitujen työkalujen lisäksi tulisi käyttää CI/CD-putkeen integroitavia SAST-työkaluja jokaisen muutoksen automaattiseen analysointiin. [18.]

SonarQube on yli kolmeakymmentä ohjelmointikieltä tukeva SAST-työkalu. Haavoittuvuuksien lisäksi se tunnistaa myös muita koodin laatua heikentäviä virheitä. SonarQube on integroitavissa toimimaan automaattisesti useiden CI/CD-työkalujen kanssa ja monelle kehitysympäristölle on saatavilla lisäosa, jolla SonarQubea voi käyttää koodin tarkistukseen reaaliajassa. [37.]

4.3.2 DAST

DAST (Dynamic application security testing) on sovelluksen turvallisuusanalyysiä, jossa etsitään haavoittuvuuksia sovelluksesta sen ollessa käynnissä. DAST-työkalut simuloivat sovellukseen kohdistuvia hyökkäyksiä antamalla sovellukselle haitallisia syötteitä ja tarkkailevat, kuinka sovellus reagoi niihin. Niiden avulla on mahdollista löytää ajonaikaisia haavoittuvuuksia, jotka eivät löytyneet lähdekoodia analysoidessa. Dynaaminen analyysi soveltuu erityisen hyvin websovellusten ja rajapintojen turvallisuuden testaamiseen, ja se tuottaa muita tapoja vähemmän virheellisiä positiivisia havaintoja. [36.]

Käytetyimpien DAST-työkalujen joukkoon kuuluu websovellusten haavoittuvuuksien löytämiseen tarkoitettu OWASP ZAP. ZAP on ilmainen avoimen lähdekoodin ohjelma muiden OWASP:in kehittämien työkalujen tapaan. Se soveltuu hy-

vin haavoittuvuuksien löytämiseen SecDevOps-ympäristöissä, koska sen käytön voi automatisoida ja integroida osaksi CI/CD-putkea. [38.]

4.3.3 Salaisuudet versionhallinnassa

Salaiset tiedot, kuten salausavaimet tai salasanat tulisi pitää poissa versionhallinnasta, etteivät ne joutuisi vääriin käsiin. Työkaluilla voi varmistaa, että tällaiset tiedot eivät vahingossa päätyisi versionhallintaan. Git-versionhallintaa tarjoavissa GitHubissa ja GitLabissa on sisäänrakennetut työkalut, jotka havaitsevat versionhallintaan päätyneet salaiset tiedot. GitLab käyttää tähän avoimen lähdekoodin Gitleaks-työkalua, jonka voi integroida myös CI/CD-putkeen. [39.]

4.3.4 Toimitusketju

Sovelluksen riippuvuuksissa voi olla haavoittuvuuksia, jotka tekevät koko sovelluksesta haavoittuvan. Kehitysprosessissa täytyy siis pitää huoli siitä, että sovelluksen riippuvuudet ovat turvallisia. Haavoittuvien riippuvuuksien löytämiseen voi käyttää SCA-työkaluja (software composition analysis). SCA-työkalut skannaavat automaattisesti sovelluksen riippuvuudet ja ilmoittavat, jos niiden joukossa on tunnettuja haavoittuvuuksia sisältäviä tai vanhentuneita riippuvuuksia. [18.]

OWASP:in ilmaisten avoimen lähdekoodin työkalujen joukosta löytyy myös SCA-työkalu, OWASP Dependency-Check. Dependency-Check käy läpi projektin riippuvuudet ja kokoaa niistä ja niistä mahdollisesti löytyvistä haavoittuvuuksista raportin. [40.]

4.3.5 Yksikkö- ja integraatiotestaus

Kehittäjien kirjoittamia yksikkö- ja integraatiotestejä käytetään yleensä varmistamaan, että sovellus toimii oikein, mutta niitä voidaan käyttää myös tietoturvallisuuden parantamiseen. Testejä varten sovellusta suunnitellessa tavallisen käyt-

tötapausten luomisen lisäksi mietitään miten pahantahtoinen käyttäjä voisi toimia ja luodaan sen perusteella ”väärinkäyttötapauksia”. Näille väärinkäyttötapauksille kehitetään haitallista toimintaa simuloivia testejä, joissa yritetään esimerkiksi päästä käsiksi toisen käyttäjän tietoihin tai käyttää toimintoja, joihin käyttäjällä ei ole oikeuksia. Jos haitallinen toiminta onnistuu, on jotain ilmiselvästi pielessä. [18.]

4.3.6 Penetraatiotestaus

Automaattisten testien lisäksi sovelluksen turvallisuutta tulisi testata myös CI/CD-putken ulkopuolella ja muillakin keinoilla kuin automaattisilla työkaluilla. Penetraatiotestaus on keino etsiä haavoittuvuuksia, jossa tietojärjestelmän omistajalta luvan saanut penetraatiotestaaja hyökkää järjestelmään käyttäen samoja metodeja kuin mahdolliset pahantahtoiset hyökkääjät [3]. Hyvän penetraatiotestaajan tekemässä testauksessa voi löytyä bugeja ja haavoittuvuuksia, jotka ovat päässeet livahtamaan testausprosessin läpi. Näiden haavoittuvuuksien löytäminen voi auttaa sovelluksen testaus- ja kehitysprosessien parantamisessa paljastamalla niiden puutteita. [18.]

4.4 Turvallisuuden automatisointi CI/CD-putkessa

Kun tietoturva on etusijalla sovelluksen kehitysprosessissa, vaaditaan tietoturvallisuuden varmistamiseen paljon työtä. Tehokkuuden parantaminen toistettavien tehtävien automaatiolla on DevOpsin ytimessä. SecDevOpsissa automaatio valjastetaan sovelluksen tietoturvallisuuden parantamiseen. Automatisoimalla osan turvallisuustoimista vähennetään tiimien taakkaa tinkimättä sovelluksen turvallisuudesta. [26.] Tekemällä tietoturvaprosesseista osan CI/CD-putkea tietoturvallisuuden varmistamisesta tulee tehokkaasti osa kehitysprosessia.

Automatisoidun CI/CD-putken tekemiseen on monia työkaluja. Perinteisten ohjelmistojen lisäksi tarjolla on myös valmiiksi integroidut työkalut tarjoavia pilvipalveluita. Yksi yleinen CI/CD-työkalu on Jenkins, jota tässä luvussa käytetään

esimerkkinä. [20.] Suurin osa edellisessä luvussa mainituista työkaluista on integroitavissa Jenkinsiin. Seuraavaksi käydään läpi esimerkki mahdollisista sovelluksen turvallisuutta parantavista toimista CI/CD-putken eri vaiheissa.

Kun sovellukseen tehdyt muutokset viedään versionhallintaan, Jenkins hakee ne automaattisesti. Ennen sovelluksen kokoamista sen riippuvuuksien turvallisuus tarkistetaan SCA-työkalujen avulla ja sen tulee läpäistä automaattinen testaaminen SAST-työkaluilla. Jenkinsiä käyttäessä SAST-työkaluna voi käyttää SonarQubea ja riippuvuuksien tarkistamiseen OWASP Dependency-Checkiä. [21.] Käyttämällä useampaa kuin yhtä työkalua on mahdollista löytää haavoittuvuuksia, joita toinen työkalu ei havaitse. [18.]

Kun lähdekoodi on läpäissyt vaaditut testit, se on aika koota valmiiksi sovellukseksi. Sovelluksen kokoamisessa käytettyjen työkalujen, kuten kääntäjän kanssa täytyy käyttää turvallisia asetuksia. Esimerkiksi C- ja C++-kielten kääntäjissä on muistivirheiltä suojaavia asetuksia, joita tulee käyttää, jotta vältyttäisiin puskurin ylivuotoihin ja muihin muistinkäsittelyn virheisiin liittyviltä haavoittuvuuksilta. Automatisoitu kokoaminen CI/CD-putkessa varmistaa, että kokoamisessa käytetään aina samoja turvallisia asetuksia. [21.]

Kun valmis sovellus on koottu onnistuneesti, se viedään automaattisesti testiympäristöön, jossa sille voidaan tehdä ajonaikaista testausta. Tässä vaiheessa DAST-työkaluilla tehtävällä automatisoidulla turvallisuusanalyysillä ja kehittäjien kirjoittamilla integraatiotesteillä pyritään löytämään virheitä ja haavoittuvuuksia, joita ei havaittu aikaisemmissa vaiheissa. Tämän vaiheen läpäistyään sovellus on valmis vietäväksi tuotantoon. [18.]

Edellinen oli vain yksi esimerkki, CI/CD-putket ovat erilaisia ja ne tulee rakentaa kehitettävän sovelluksen tarpeisiin. Sovelluksen turvallisuuden automatisointiin pitää valita oikeat työkalut tapauskohtaisesti ja harkitusti.

4.5 Tuotantovaiheen ja infrastruktuurin turvallisuus

Valmis turvalliseksi varmistettu sovellus viedään tuotantoympäristöön automatisoidusti. Sovelluksen tietoturva täytyy varmistaa myös sen ollessa tuotannossa. [18.] Tätä vaihetta ei tässä sovelluksen kehitysvaiheeseen keskittyvässä insinöörityössä käsitellä kovin syvällisesti, mutta tässä luvussa käydään muutamia siihen liittyviä asioita läpi.

Sovelluksen suojaamiseksi tuotannossa täytyy sen käyttämä palvelininfrastruktuuri suojata. Hyvät turvallisuuskäytännöt, joita aikaisemmin käsiteltiin, ovat tässä tärkeitä. Infrastruktuurin turvallisuuden varmistamiseen käytettäviä ohjelmia ovat esimerkiksi tunkeutumisen havaitsemisjärjestelmät (IDS, intrusion detection system) ja palomuurit.

DevOpsiin kuuluu sovelluksen toiminnan jatkuva valvonta tuotannossa ja siitä saadun palautteen hyödyntäminen sovelluksen kehityksessä [20]. Myös sovelluksen turvallisuudesta ja siihen kohdistuvista hyökkäyksistä saa sovelluksen valvontaan käytetyillä työkaluilla tietoa, jota voi hyödyntää sovelluksen kehittämisessä tietoturvallisemmaksi [18].

Infrastruktuuri koodina (IaC, Infrastructure as code) ja sen käyttö tietoturvallisuuden parantamiseen on SaC:n ohella toinen SecDevOpsin ydinperiaate [26]. Käyttämällä konfiguraationhallintaohjelmia, joissa palvelininfrastruktuurin hallinta ja konfiguraatio pohjautuu koodiin, kuten Ansible, Chef tai Puppet on mahdollista pystyttää nopeasti suuri määrä yhdenmukaisia palvelinympäristöjä. Skaalautuvien palvelinympäristöjen rakentamisen helpottamisen lisäksi IaC mahdollistaa kehitys-, testi ja tuotantoympäristöjen erojen minimoinnin. Automatisoimalla infrastruktuurin hallinta vältytään palvelinten manuaalisessa konfiguraatiossa tapahtuvilta huolimattomuusvirheiltä, jotka saattaisivat tehdä järjestelmästä haavoittuvan hyökkäyksille.

Infrastruktuurin konfiguraatioiden pohjautuessa koodiin niitä on myös mahdollista kehittää samalla tavalla. Konfiguraatiotiedostot ovat versionhallinnassa, joten

niiden muutosten seuraaminen on helppoa. Muutokset menevät CI/CD-putken läpi tuotantoon ja niille tehdään automaattisia testejä, kuten muullekin koodille. IaC:in avulla turvallisuuden voi kirjoittaa osaksi konfiguraatiota. Haavoittuvuuksien korjaaminen ja korjausten vieminen tuotantoon onnistuu nopeasti ja tehokkaasti. [18.]

Infrastruktuurin konfiguraation turvallisuuden varmistaminen onnistuu parhaiten käyttämällä kehystä, joka sisältää turvalliset lähtökohdat konfiguraatiolle [21]. DevSec Hardening Framework on alun perin kehitetty Deutsche Telekomin infrastruktuurin turvallisuuden automatisointiin. Nykyään se on avoimen lähdekoodin kehys Ansiblen, Cheffin ja Puppetin turvalliseen konfigurointiin automaattisesti. [41.]

4.6 SecDevOps-elinkaari

Tässä insinöörityössä on nyt käyty läpi keskeisimmät asiat sovelluksen kehittämisestä SecDevOps-ympäristössä. Sovelluksen turvallisuuden parantamiseen käytetään SecDevOpsissa monia keinoja, jotka on integroitu osaksi sovelluksen elinkaarta. Aikaisemmin esiteltiin kahdeksasta vaiheesta koostuva tyypillinen sovelluksen elinkaari DevOpsissa, jonka vaiheet ovat plan, code, build, test, release, deploy, operate ja monitor [19]. Taulukossa 1 on tiivistettynä SecDevOpsissa tehtävät toimenpiteet sovelluksen turvallisuuden parantamiseksi jokaisen vaiheen osalta.

Taulukko 1: Tietoturvan varmistaminen elinkaaren vaiheissa

Vaihe	**Tietoturvatoimet**
Plan	Uhkamallinnus, turvalliseksi suunniteltu arkkitehtuuri, turvallisten kehysten ja kirjastojen käyttö.
Code	Hyvät ohjelmointikäytännöt, koodin laadun varmistaminen kehitysympäristön lisäosilla, turvallisuuden huomiointi koodin vertaisarvioinnissa.
Build	Staattinen analyysi, riippuvuuksien tarkistaminen, sovelluksen kokoaminen turvallisilla asetuksilla.
Test	Dynaaminen analyysi, turvallisuutta testaavat yksikkö- ja integraatiotestit.
Release	-
Deploy	Automatisoitu tuotantoon vieminen ehkäisee inhimillisiä virheitä.
Operate	Turvalliseksi konfiguroitu IaC, sovelluksen toiminnan valvonta työkaluilla.
Monitor	Jatkuvasta valvonnasta saadun tiedon käyttö tietoturvan parantamiseen.

5 Yhteenveto

Tämän insinöörityön tarkoituksena oli selvittää, mitkä ovat ajankohtaisia tietoturvauhkia ja miten SecDevOps-kehitystapa soveltuu niiden torjuntaan. Työn alussa käsiteltiin tietoturvauhkia, jotka perustuvat enimmäkseen Euroopan unionin kyberturvallisuusviraston Threat Landscape 2022-raporttiin [4]. Erilaisia uhkia on valtava määrä, joten käsiteltäväksi täytyi valikoida erityisesti ohjelmistotuotannossa huomioitavia uhkia. Tietoturvauhkien jälkeen käsiteltiin DevOpsiin liittyviä tietoturvaongelmiin ja esiteltiin niiden ratkaisuksi luotuja kehistystapoja. Sen jälkeen työssä syvennyttiin SecDevOpsiin ohjelmistokehittäjän näkökulmasta ja käytiin läpi siinä käytettyjä keinoja sovelluksen tietoturvallisuuden parantamiseksi.

Työtä varten tehdyn selvityksen perusteella on selkeää, että SecDevOpsista voi olla hyötyä monien tietoturvariskien ehkäisyssä. Valitsemalla sovelluksen kehityksessä käytetyt teknologiat ja kielet turvallisuuden perusteella tietoturvallisen koodin kirjoittamisesta tulee kehittäjille helpompaa. Toimitusketjuhyökkäysten yleistyessä SCA-työkalujen käyttö haavoittuvien riippuvuuksien tunnistamiseen on yhä tärkeämpää. Uhkamallinnus varmistaa, että sovellukseen kohdistavat uhat huomioidaan kehityksessä. Lisäämällä kehittäjien vastuuta, parantamalla tietoturvaosaamista ja käyttämällä SAST-työkaluja vähennetään kirjoitettujen haavoittuvuuksien määrää ja parannetaan mahdollisuutta havaita haavoittuvuudet varhain. Havaitsemalla haavoittuvuudet varhain ne on helpompi korjata. Haavoittuvuuksien paljastuessa myöhään suuri määrä koodia saatetaan joutua kirjoittamaan uudelleen, jolloin sen kirjoittamiseen kulunut aika menee hukkaan.

Yhteenvetona todettakoon, että sovelluskehitys tavalla, jossa tietoturva on etusijalla parantaa sovelluksen turvallisuutta huomattavasti. Projekteissa, joissa tietoturvallisuus ei ole ehdottoman tärkeää, SecDevOps ei kuitenkaan välttämättä ole paras valinta. Tietoturvan priorisointi laskee aina jonkin verran sovelluskehityksen nopeutta ja tehokkuutta, joten se ei ole aina järkevää. Lisäksi SecDe-

vOps vaatii paljon osaamista ja työtä. Sovelluskehityksessä joutuu yleensä tasapainoilemaan, jotta tietoturvallisuuden varmistamiseen käytetään riittävästi resursseja ilman, että se haittaa kehitysprosessin muita osia. Vaikka SecDevOps on jäänyt esimerkiksi DevSecOpsiin verrattuna melko vähälle huomiolle, on SecDevOpsissa kuitenkin selkeitä etuja ja sille on varmasti paikkansa kehitystapojen joukossa. DevOpsissa tarvitaan tietoturvaa, ja SecDevOpsissa turvallisuuden parantamiseen käytetyt menetelmät ovatkin viime vuosina saaneet suosiota DevOpsissa yleisesti.

Lähteet

1 Morgan, Steve. 2022. Cybercrime To Cost The World 8 Trillion Annually In 2023. Verkkoaineisto. Cybercrime Magazine. <https://cybersecurityventures.com/cybercrime-to-cost-the-world-8-trillion-annually-in-2023/>. 17.10.2022. Luettu 31.8.2023.

2 Clark, Tim. 2015. Most Cyber Attacks Occur From This Common Vulnerability. Verkkoaineisto. Forbes. <https://www.forbes.com/sites/sap/2015/03/10/most-cyber-attacks-occur-from-this-common-vulnerability/?sh=75884ec37454>. 10.3.2015. Luettu 1.9.2023.

3 Death, Darren. 2017. Information Security Handbook. Packt Publishing.

4 ENISA Threat Landscape 2022. 2022. European Union Agency for Cybersecurity.

5 Zieniūtė, Ugnė. 2021. Mikä on botnet eli bottiverkko?. Verkkoaineisto. NordVPN. <https://nordvpn.com/fi/blog/bottiverkko/>. 27.7.2021. Luettu 20.9.2023.

6 What is a ransomware attack?. Verkkoaineisto. F-Secure. <https://www.f-secure.com/en/articles/what-is-a-ransomware-attack>. Luettu 12.9.2023.

7 Cimpanu, Catalin. 2020. First death reported following a ransomware attack on a German hospital. Verkkoaineisto. ZDNET. <https://www.zdnet.com/article/first-death-reported-following-a-ransomware-attack-on-a-german-hospital/>. 17.9.2022. Luettu 8.9.2023.

8 Mobile Malware Threatens Smartphones & Tablets. Verkkoaineisto. Kaspersky. <https://usa.kaspersky.com/resource-center/threats/mobile-malware>. Luettu 8.9.2023.

9 Takedown of SMS-based FluBot spyware infecting Android phones. 2022. Verkkoaineisto. Europol. <https://www.europol.europa.eu/media-press/newsroom/news/takedown-of-sms-based-flubot-spyware-infecting-android-phones>. 1.6.2022. Luettu 15.9.2023.

10 Farrier, Ellie. 2022. What Is Pegasus Spyware and Is Your Phone Infected with Pegasus?. Verkkoaineisto. Avast. <https://www.avast.com/c-pegasus-spyware>. 30.9.2022. Luettu 16.9.2023.

11 Ranger, Steve. 2020. What is the IoT? Everything you need to know about the Internet of Things right now. Verkkoaineisto. ZDNET. <https://www.zdnet.com/article/what-is-the-internet-of-things-everything-you-need-to-know-about-the-iot-right-now/>. 3.2.2020. Luettu 30.9.2023.

12 Perry, J. Steven. 2019. Anatomy of an IoT malware attack. Verkkoaineisto. IBM. <https://developer.ibm.com/articles/iot-anatomy-iot-malware-attack/>. 30.10.2019. Luettu 22.9.2023.

13 10 Types of Social Engineering Attacks. 2021. Verkkoaineisto. Crowdstrike. <https://www.crowdstrike.com/cybersecurity-101/types-of-social-engineering-attacks/>. 28.12.2021. Luettu 10.9.2023.

14 Toimintaohje – palvelunestohyökkäys. 2022. Verkkoaineisto. Kyberturvallisuuskeskus. <https://www.kyberturvallisuuskeskus.fi/fi/julkaisut/toimintaohje-palvelunestohyokkays>. 20.09.2022. Luettu 15.9.2023.

15 Toimintaohje – toimitusketjuhyökkäys. 2022. Verkkoaineisto. Kyberturvallisuuskeskus. <https://www.kyberturvallisuuskeskus.fi/fi/julkaisut/toimintaohje-toimitusketjuhyokkays>. 20.09.2022. Luettu 8.9.2023.

16 Grustniy, Leonid. 2022. Log4Shell a year on. Verkkoaineisto. Kaspersky. <https://www.kaspersky.com/blog/log4shell-still-active-2022/46545/>. 8.12.2022. Luettu 8.9.2023.

17 What is Supply Chain Attack. 2021. Verkkoaineisto. Crowdstrike. <https://www.crowdstrike.com/cybersecurity-101/cyberattacks/supply-chain-attacks/>. 8.12.2021. Luettu 9.9.2023.

18 Bird, Jim. 2016. DevOpsSec. O'Reilly.

19 Ozanich, Athena. 2021. What Is DevOps: A Complete Guide. Verkkoaineisto. HubSpot. <https://blog.hubspot.com/website/what-is-devops>. 14.12.2021. Luettu 15.10.2023.

20 Courtemanche, Meredith; Mell, Emily; Gillis, Alexander. What is DevOps? The ultimate guide. Verkkoaineisto. TechTarget. <https://www.techtarget.com/searchitoperations/definition/DevOps>. Luettu 1.9.2023.

21 Hsu, Tony. 2018. Hands-On Security in DevOps. Packt Publishing.

22 Shift Left DevOps. Verkkoaineisto. Aqua Security. <https://www.aquasec.com/cloud-native-academy/devsecops/shift-left-devops/>. Luettu 1.9.2023.

23 SecDevOps vs DevSecOps: What's the Difference?. Verkkoaineisto. CodeSigningStore. <https://codesigningstore.com/secdevops-vs-devsecops>. Luettu 2.9.2023.

24 Reynolds, Justin. 2022. SecDevOps: A Practical Guide to the What and the Why. Verkkoaineisto. Plutora. <https://www.plutora.com/blog/secdevops-a-practical-guide-to-the-what-and-the-why>. 4.5.2022. Luettu 2.9.2023.

25 Chan, Zen. 2022. AppSec: SecDevOps or DevSecOps? Do We Need to Choose? Guide to the What and the Why. Verkkoaineisto. Hackernoon. <https://hackernoon.com/appsec-secdevops-or-devsecops-do-we-need-to-choose-guide-to-the-what-and-the-why>. 28.10.2022. Luettu 3.9.2023.

26 SecDevOps: What it is and why it matters. 2023. Verkkoaineisto. Pluralsight. <https://www.pluralsight.com/blog/software-development/secdevops>. 5.5.2023. Luettu 5.9.2023.

27 OWASP Top Ten. 2021. Verkkoaineisto. OWASP. <https://owasp.org/www-project-top-ten/>. Luettu 1.9.2023.

28 Girnus, Peter. 2023. Cybersecurity - Secure by Default vs. Secure by Design. Verkkoaineisto. <https://www.petergirnus.com/blog/cybersecurity-secure-by-default-vs-secure-by-design>. 10.10.2023. Luettu 20.10.2023.

29 CWE-95: Improper Neutralization of Directives in Dynamically Evaluated Code ('Eval Injection'). Verkkoaineisto. MITRE. <https://cwe.mitre.org/data/definitions/95.html>. Luettu 19.10.2023.

30 SEI CERT Oracle Coding Standard for Java. Verkkoaineisto. Software Engineering Institute CERT Coordination Center. <https://wiki.sei.cmu.edu/confluence/display/java/SEI+CERT+Oracle+Coding+Standard+for+Java>. Luettu 19.10.2023.

31 EXP00-J. Do not ignore values returned by methods. Verkkoaineisto. CERT Coordination Center. <https://wiki.sei.cmu.edu/confluence/display/java/EXP00-J.+Do+not+ignore+values+returned+by+methods>. Luettu 25.10.2023.

32 FIO08-J. Distinguish between characters or bytes read from a stream and -1. Verkkoaineisto. CERT Coordination Center. <https://wiki.sei.cmu.edu/confluence/display/java/FIO08-J.

+Distinguish+between+characters+or+bytes+read+from+a+stream+and+-1>. Luettu 25.10.2023.

33 Newman, Lily. 2022. The ‘Viral’ Secure Programming Language That’s Taking Over Tech. Verkkoaineisto. WIRED. <https://www.wired.com/story/rust-secure-programming-language-memory-safe/> 2.11.2022. Luettu 18.10.2023.

34 Drake, Victoria. Threat Modeling. Verkkoaineisto. OWASP. <https://owasp.org/www-community/Threat_Modeling>. Luettu 2.10.2023.

35 Conklin, Larry; Drake, Victoria; Strittmatter, Sven. Threat Modeling Process. Verkkoaineisto. OWASP. <https://owasp.org/www-community/Threat_Modeling_Process>. Luettu 2.10.2023.

36 Sengupta, Sudip. 2022. SAST vs. DAST for Security Testing – How do they differ?. Verkkoaineisto. Crashtest Security. <https://crashtest-security.com/sast-dast/>. 22.11.2022. Luettu 5.10.2023.

37 SonarQube. Verkkoaineisto. SonarSource. <https://www.sonarsource.com/products/sonarqube/>. Luettu 5.10.2023.

38 Rosenbaum, Sasha. 2020. Maintainer spotlight: How to secure your project with one of the world’s top open source tools. Verkkoaineisto. The GitHub Blog. <https://github.blog/2020-07-30-maintainer-spotlight-how-to-secure-your-project-with-one-of-the-worlds-top-open-source-tools/>. 30.7.2020. Luettu 8.10.2023.

39 Kuppens. Jamie. 2022. A Guide to Keeping Secrets out of Git Repositories. Verkkoaineisto. This Dot Labs. <https://www.thisdot.co/blog/a-guide-to-keeping-secrets-out-of-git-repositories/>. 31.5.2022. Luettu 10.10.2023.

40 OWASP Dependency-Check. Verkkoaineisto. OWASP. <https://owasp.org/www-project-dependency-check/>. Luettu 10.10.2023.

41 DevSec Project. Verkkoaineisto. DevSec Hardening Framework Project. <https://dev-sec.io/project/>. Luettu 18.10.2023.