LAUREA
AMMATTIKORKEAKOULU
Yhdessä enemmän

REST-API:en tietoliikenteenvalvonta ja poikkeamien tunnistaminen

Ari Lappalainen

2018 Laurea

Laurea-ammattikorkeakoulu

REST-API:en tietoliikenteenvalvonta ja poikkeamien tunnistaminen

Ari Lappalainen
Tietojenkäsittely
Opinnäytetyö
Huhtikuu, 2018

Laurea-ammattikorkeakoulu **Tiivistelmä**
Tietojenkäsittely
Tietojenkäsittely (AMK)

Ari Lappalainen

REST-API:en tietoliikenteenvalvonta ja poikkeamien tunnistaminen

Vuosi 2018 Sivumäärä 40

REST Api:t ovat nykyaikaina hyvin paljon käytössä oleva osio applikaatioiden ja järjestelmien osina. Tietoturvanosalta REST:ssä on omat huomioon otettavat asiansa ja verkkoliikenteenvalvonta on tärkeässä asemassa nykytilassa.

Opinnäytetyö tehdään toimeksiantaja yritykselle. Ja tarkoituksena on kehittää käyttöön työkalu, jolla voidaan tunnistaa poikkeamia verkkoliikenteessä lokitiedostojen avulla.

Valmista työkalua ei ole, joten menetelmänä käytetään inkrementaalista kehitystä. Kyseinen kehitysmenetelmä soveltuu hyvin ohjelmistoprojekteihin, oli kyseessä vesiputous tai ketterän mallin mukainen kokonaisuus. Opinnäytetyö tehtiin osana muita työtehtäviä, ja kehitystä seurattiin työtuntien mukana kehitysprosessin ajan. Tiivisyhteistyö takasi toimenantajan kanssa sen, että toivottu ratkaisu saatiin aikaiseksi sopivammalla tavalla.

Opinnäytetyöntuloksena luotiin työkalu, jolla saadaan valvottua API tietoliikennettä ja havainnoitua poikkeamia liikenteessä. Työkalu testataan oikeassa asiakasympäristössä ja tulee olemaan osa suurempaa kokonaisuutta.

Asiasanat: Tietoturva, kehitys, koneoppiminen, inkrementaalinen kehitys, REST API

Laurea University of Applied Sciences
Degree Programme in Business Information Technology

Ari Lappalainen

Abstract

REST API security and anomaly detection

Year 2018 Pages 40

Rest API's are part of the applications and systems that are widely used today. In regards to security, REST has it own issues to be taken into consideration. Network traffic monitoring plays an important role.

The bachelor's thesis was commissioned. It's objective is to develop a tool for identifying anomalies or abnormalities in network traffic using log files.

There is no tool to prepare, so incremental development is used. This development method is well suited for software projects, whether it is a waterfall or an agile entity. The thesis was carried out as a part of other job assignments, and development was monitored within working hours during the development process. In close cooperative with the supplier a desired and more appropriate solution was established.

As a result of the thesis, a tool was created to provide controlled API communication and has targeted offset traffic. Tool is tested in the right customer environment and will be part of a large entity.

Keywords: Security, development, machinelearning, REST API, incremental development

Käsitteet

API: Application programming interface, eli ohjelmointirajapinta. Ohjelmointirajapinnan tarkoituksena on määritellä yleisimmät toiminnot. Ohjelmointirajapinnan kautta sovellukset keskustelevat keskenään.

API hallinta: Standardit ja käytännöt tuottavajärjestelmä. Päätarkoituksena API hallinnalla on tehostaa API käyttöä turvallisesti ja hallitusti. API hallinta sisältää API gateway, manager, portaali ja analytiikka komponentit.

API gateway: Osio, jossa ajoaikainen suoritus tapahtuu. Toiminnallisuuksia orkesterointi, sisällön ja sanomien muokkaaminen, ja reititys. Kyseisellä osiolla myös turvataan API:t luvattomalta käytöltä, ja tunnistetaan tietoturvauhkia ja hoidetaan salaus.

API manager: Tietoturvan, käyttöoikeuksia, versiointia ja julkaisua hoitava komponenti.

API portaali: Itsepalveluosio, jonka kautta kehittäjät voivat selata ja testata julkaistuja API julkaisuja. Voidaan sisällyttää myös foorumeita ja blogeja.

API analytiikka: Erillinen osio, jolla hoidetaan API:en käytön määrää ja sanomien kulkua.

Cache: Välimuisti

Client-Server: Asiakas-serveri malli on malli, jota käytetään REST API kutsuissa. Tämä tarkoittaa sitä, että käyttäjä käyttää laitetta, joka lähettää sanoman/kutsun laitteesta erilliselle palvelimelle.

Code-on-demand: mahdollisuus lähettää ohjelmointikielistä sanomaa kutsun mukana eli ladattavakoodi.

CSRF: Cross-Site-Request-Forgery hyökkäys, jossa pakotetaan applikaatio tekemään ei toivottuja toimintoja. Hyökkäystä käytetään kaappaamaan esimerkiksi rahaliikennettä verkkokaupassa tai vaihtamaan käyttäjätunnuksen kirjautumistietoja.

CRUD: Luo, lue, päivitä ja poista toiminallisuudet. Akronyymi neljälle perustoiminnallisuudelle ohjelmoinnissa.

DevSecOps: Ketterä kehitysmenetelmä, jossa automatisoidaan mahdollisimman paljon ja tuodaan turvallisuusajattelu mukaan jokaisen kehitysprosessin vaiheeseen.

Endpoint: Päätepiste. Määritetty päätepiste API kutsulle, eli osoite jonne API ottaa yhteyttä.

Layered system: Kerroksittainen järjestelmä. Eli järjestelmässä on kerroksia, jotka kommunikoivat keskenään ja näiden välillä toimii määritelty hierarkia.

Load balancer: Kuorman tasaaja on pilviarkkitehtuurinen osa, joka jakaa kuormaa virtuaalikoneiden välillä ja tarvittaessa käynnistää uusia virtuaalikoneita.

REST API: API arkkitehtuurityyli. Hyödyntää http:n standardimetodeja, sekä http-protokollaa.

Refress Token: Päivitysavain. Eli avain, joka antaa käyttäjälle luvan käyttää palvelua kirjautumisen jälkeen.

Inkrementaalinen kehitys: Kehitysmenetelmä, jossa luodaan prototyyppiä asiasta jota ei ole vielä kehitetty.

SDLC: Software development life cycle, ohjelmistokehityksen elinkaari. Menetelmä, jolla kehitettään järjestelmää parhaan tavan mukaisessa syklissä.

Security by desing: järjestelmäkehityksen tapa, jossa ajatellaan kehitystä tietoturvan näkökulmasta.

Stateless: Tilattomuus, ohjelmistoarkkitehtuurinen käsite, jossa käytetty ohjelman osa ei tallenna tilaa käytettäessä.

Tiedonlouhinta (Data mining): on menetelmä, missä pyritään löytämään oleellinen tieto suuresta tietomäärästä.

Uniform interface: Yhdenmukainen rajapinta. REST arkkitehtuurissa menetelmä, jonka vuoksi jokainen toiminto voi toimia itsenäisesti.

Man in the middle-hyökkäys: Mies välissä-hyökkäys. Hyökkäys, jossa REST API tapauksessa kolmasosapuoli pääsee asiakaan ja serverin välisen liikenteen väliin ja kykenee manipuloimaan näiden välistä tiedonsiirtoa ja sanomia.

Open source: Avoin lähdekoodi, ohjelmistotuottamis ja kehitysmenetelmä jossa tarjotaan mahdollisuus tutustua ohjelman lähdekoodiin ja muokkaamaan tätä omien tarpeidensa mukaisesti. Tarkoittaa myös vapaita ohjelmistoja.

XSS: Cross site scriptin eli XSS on tietoturva-aukko joka mahdollistaa haitallisen koodin syöttämisen ohjelmaan tai sivustoon ja mahdollistaa tämän kautta tunkeutumisen palveluun.

1 Johdanto

Tämä on opinnäytetyö, jossa ratkotaan toimeksiantajan tietoturvapoikkeamiin liittyvää ongelmaa. Tarkoituksena on luoda työkalu, jolla pystytään havaitsemaan lokitiedostoista poikkeamiksi luokiteltavia tapahtumia.

Yrityksen toimintaan kuuluu jatkuvat palvelut ja näiden valvonta. Tätä suoritetaan erilaisten tuotteiden kautta, ja näihin tulisi lisätä valvontaa liikenteeseen ja havaittuihin tietoturvaongelmiin. Kyseisessä palvelumuodossa valvotaan ulkopuolisten kehittäjien luomia API-ratkaisuja.

Tietoturva-auditoinnissa on tullut ilmi, että API kutsut näyttävät päätepisteitä kutsuissa. Tämä ilmenee tarpeena luoda tietoturvaratkaisuja lokien ja liikenteen valvontaan, joilla voidaan paikantaa poikkeamia liikenteestä.

Materiaalina ratkaisun kehitykseen käytetään olemassa olevia lokitiedostoja, joita tutkimalla voidaan havainnollistaa poikkeamia ja kehittää menetelmiä. Lokitiedoston tutkimiseksi on käytettävä monivaiheista kokonaisuutta.

1.1 Tavoite

Tavoitteena on kehittää työkalu, jolla käydään läpi tietoliikenteen lokitiedostoja ja havaitaan näistä tietoturvaan liittyviä poikkeamia. Tarve kohdentuu toimeksiantajan API-julkaisun valvontaan ja näissä tapahtuvaan tietoliikenteeseen.

Työkalua pitäisi voida hyödyntää erilaisten lähteiden tutkintaan. Ja materiaaliksi testaamiseen käytetään oikeita lokitiedostoja, että voidaan saada varmuus työkalun toimivuudesta.

Tavoite on saavutettu, kun on luotu toimiva työkalu joka tunnistaa poikkeaman verkkoliikennettä seuraavista lokitiedostoista. Tämä tarkoittaa sitä, että työkalu on valmis testattavaksi tuotannossa ja on mahdollisuus esittää tulosta eteenpäin.

Lisätarkoitusta tuo, että onnistuessaan työkalu mahdollistaa tarkoituksensa mukaisen valvonnan toteuttamisen kustannustehokkaasti ja vapaasti valittavassa pilviympäristössä osana muiden API julkaisun työvälineiden toimintaa. Lisätavoitteena ja toiveena on, että työkalulla voitaisiin tutkia myös muita lokitiedostoja.

2 Tutkimusmenetelmät

Työ jakautuu kuuteen eri osioon. Tutkimusmenetelmänä käytetään inkrementaalista kehittämistä.

Inkrementaalisen kehityksen idea on tuottaa ohjelmisto sarjana pienempiä kehitysaskeleita sen sijaan, että ohjelmisto rakennetaan kerralla valmiiksi. (Tuovinen 2013, 15) Tämä menetelmä soveltuu työhön, koska työssä kehitetään uutta osiota isompaan kokonaisuuteen suorittamaan omaa tehtäväänsä. Tämä menetelmä soveltuu tilanteisiin, joissa luodaan uutta ilman olemassa olevaa pohjaa.

Inkrementaalisessa ohjelmistokehityksessä luodaan prototyyppi, jossa toteutetaan joitakin järjestelmän piirteitä. Siksi tämä on käytännöllisin tähän tutkimustyyppiin. Testauksien perusteella kirjataan havainnot, joita on prototyypin pohjalta tehty. Ja testiversion jälkeen etenemiseen on kaksi vaihtoehtoa.

- Varsinaisen mallinkehitys aloitetaan aivan alusta, huomioiden saadut kokemukset ja palautteet.
- Testattu malli kehitettään valmiiksi tuotteeksi
- Lisäksi näiden kahden mallin välimuotoja käytetään.

Työ jakautuu teoriaan siitä mikä on REST API, mitä uhkia näissä on ja kuinka niitä ratkotaan. Vaatimusmäärittelyllä selvitetään: mitä uudelta osiolta vaaditaan, mitä toiminnallisuuksia kokonaisuudessa täytyy olla? Kehitysympäristö, eli millaisessa ympäristössä kokonaisuutta käytetään. Mitä siinä tulee ottaa huomioon. Toteutukseen tiedon perusteella, eli kuinka ja mitenkä ongelma ratkaistaan. Ja lopulta testaukseen, kuinka työkalu toimii ja tuottaako se toivottuja tuloksia?

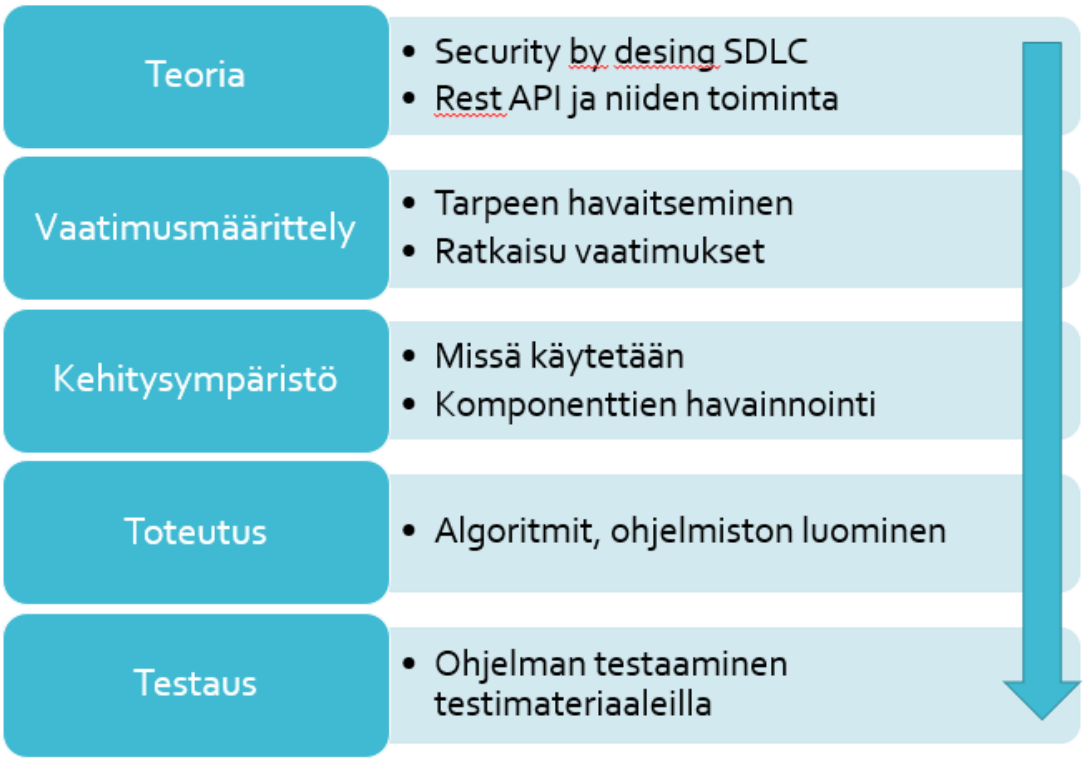
2.1 Tutkimus rakenne

Tutkimus kokonaisuudessaan jakautuu kuuteen erilliseen osioon. Näiden osioiden voidaan todeta olevan seuraavat.

- Teoria
- Vaatimukset
- Kehitysympäristö
- Toteutus
- Testaaminen

Kehitykseen käytetään SDLC mukaista Security by desing-menetelmää ja inkrementaalista kehitystä. Lopputuloksena on tietoturvaa testaava ohjelma, ja tämän vuoksi on hyvä ottaa huomioon tämä myös tuotteen kehityksessä.

Deagun, Johnsson ja Sawano (2018) kertovat kirjassaan Secure by desing, että tietoturvan lähteenä on usein se että tietoturva otetaan osaksi kehitystä erillisenä osiona. Siksi Security by desing:ssa ajatuksena on ottaa tietoturva osaksi kehitysprosessia suunnittelusta lähtien, ja tällä saadaan vähennettyä mahdollisia tietoturvahaasteita projektin loppuvaiheesta.



Kuvio 1 Työnkulku

Tällä jakautumisella ohjataan työtä eri vaiheisiin ja luodaan kokonaiskuvaa kehitettävästä kohteesta ja sen tarpeista. Valvottavan osion ymmärtämiseksi on tärkeää, että ymmärretään mikä on REST API ja mitä tietoturvauhkia näissä otetaan huomioon. Ja millä tavalla niiltä suojaudutaan. Kuvion 2 mukaisen prosessin mukaisesti.

2.2 Käytänteet

Kehityksen viitekehyksenä inkrementaalisessa kehityksessä käytetään Software Development life cycleä. Tämä on Microsoftin kehittämä menetelmä, jossa kehitys on porrastettu erillisiin vaiheisiin. (Microsoft, 2015)

2.3 Software Development life cycle

Ohjelmiston kehityksessä käytetään SDLC(software development life cycleä). Menetelmän tarkoituksena on tuoda turvallisuus osaksi kehitystä, nousseiden tietoturvariskien vuoksi. Ideana on havainnoida aikaisessa kehityksen vaiheessa mahdolliset uhat ja riskit, ja tällä tavoin vähentää mahdollisia kriittisiä tietoturvariskejä kehityksen loppuvaiheessa.

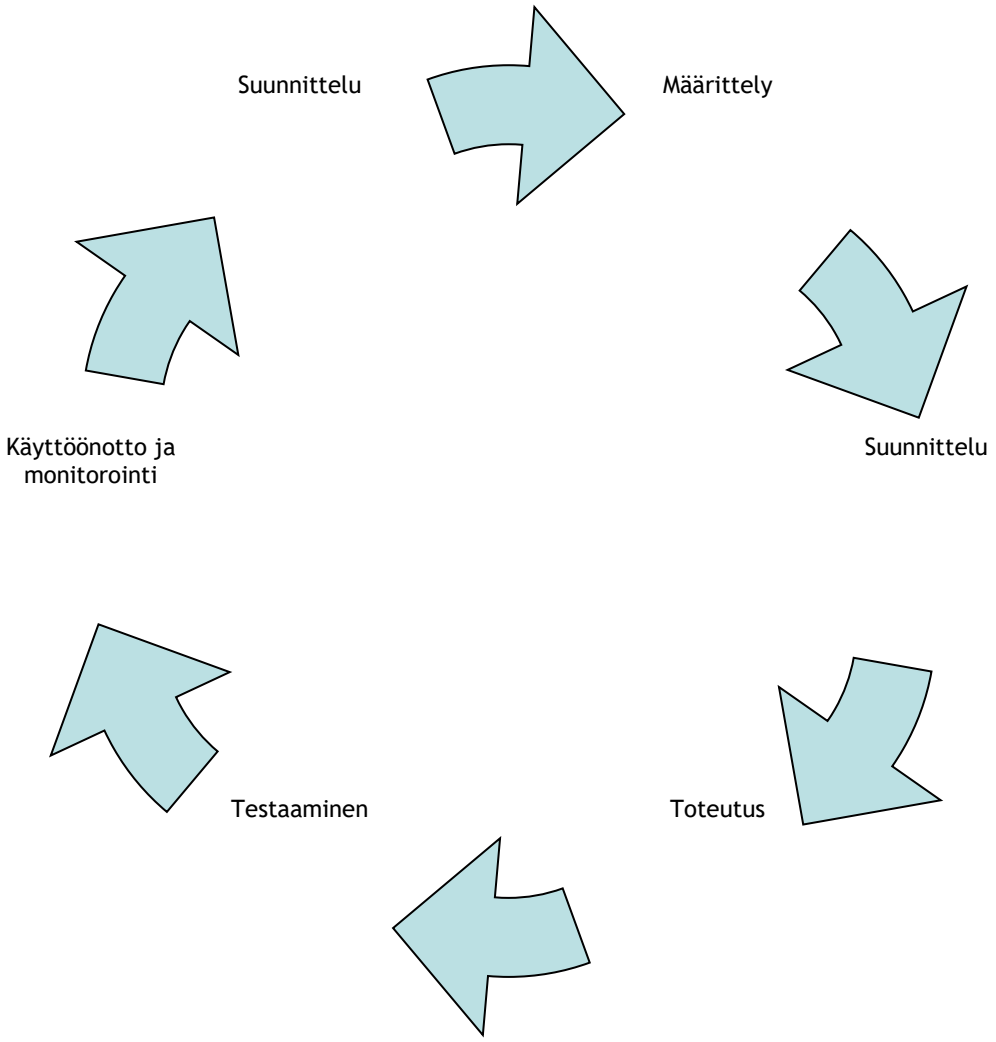
Samalla prosessilla kuvataan tuotetun ohjelmiston elinkaari ja tämän vaiheet, suunnittelusta käyttöönottoon ja valvontaan. Software Development life cycle vaiheita on useita kokonaisuudessa Kay (2002) mukaan ovat:

- Projektin suunnittelu: havainnoidaan projektin päämäärä ja tavoitteet.
- Vaatimusmäärittely: Määritellään vaatimukset järjestelmälle ja määritellään toiminnallisuudet. Analysoidaan loppukäyttäjän tarpeet.
- Järjestelmän suunnittelu: Suunnitellaan ominaisuudet, joita tarvitaan. Rakenteellinen arkkitehtuuri ja tarvittavat dokumentaatiot
- Implementaatio: Luodaan suunniteltu järjestelmä
- Integraatio ja testaaminen: Testataan järjestelmän toimivuus ja todetaan, onko järjestelmä vaatimuksia vastaava.
- Käyttöönotto: Tuodaan järjestelmä käyttöön suunnitellussa ympäristössä.
- Ylläpito: Päivitetään ja ylläpidetään järjestelmää ja suoritetaan monitorointia.

Tässä työssä projektin suunnitteluvaiheessa käsitellään tietoperusta oleellisena osana itse kehitystyötä. Tämän jälkeen suoritetaan vaatimusmäärittely, jonka perusteella havaitaan ratkaisun vaatimukset. Tämän jälkeen tutkitaan, millaisessa ympäristössä työkalua käytetään ja mitä komponentteja on käytössä.

Tämän jälkeen edetään implementaatio vaiheeseen eli toteutukseen. Valitaan tarvittavat algoritmit ja luodaan itse työkalu ohjelmoinnilla. Ohjelmiston ensimmäisen version ollessa

valmis siirrytään testaukseen, jossa työkalu testataan lokitiedostoilla. Koko prosessia menee kuvio 2 mukaisesti noudattaen SDLC prosessia.



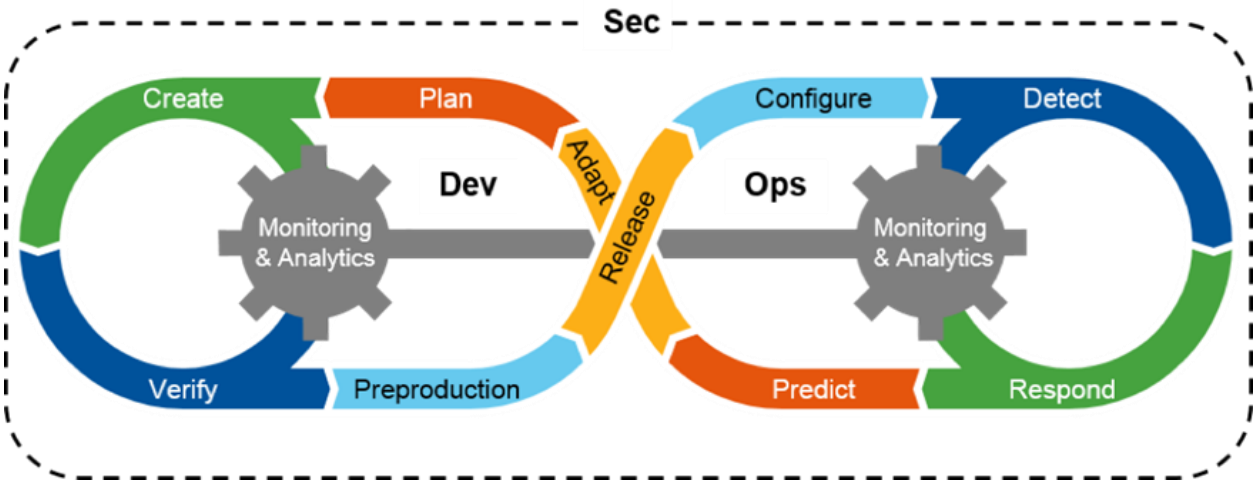
Kuvio 2 SDLC

2.3.1 Valvonta

API hallinta-palveluissa valvotaan API:n julkaisua ja käyttöä, palvelut monitoroivat liikennettä ja liikenteen sisällä tapahtuvia asioita. Palvelut tuottavat valvontalokia ja nämä lähettävät hälytyksiä ongelmatilanteissa. Palvelun tuottamisessa on havaittu tietoturvatestauksessa Nixu:n toimesta. Tietyt palvelut vuotavat API päätepisteitä, ja tämä ei ole suotavaa.

Ongelman ratkaisuksi otetaan palvelun tuottamia lokitiedostoja ja näitä louhitaan etsien poikkeamia ja niiden ilmentymiä. Näiden pohjalta voidaan opettaa järjestelmää koneoppimisen kautta tunnistamaan poikkeamat ja ilmoittamaan näistä hälytyksellä valvovalle tiimille. Näin voidaan raportoida API-kehittäjille kyseisestä ongelmasta ja pyytää korjaamaan ongelman aiheuttaja.

Jatkuva monitorointi on yksi tärkeä DevSecOpsin osa-alueista. Kehitystehtävän mukainen työkalu tukee tätä vaatimusta täydentäen monitoroinnin mahdollisuuksia.



Kuvio 3 DevSecOps (GSA, 2018)

Kuvio 1 mukainen GSA techin (2018) artikkelin mukaan DevSecOpsin kulttuuriin kuuluu jatkuvan tietoturvan huomionottamisen lisäksi jatkuva palveluiden monitorointi ja testaaminen automaatiolla, sekä manuaalisesti.

GSA (2018) kuvasta voidaan havaita, kuinka monitorointi tapahtuu keskeisesti kehitysprosessissa jatkuvana toimenpiteenä.

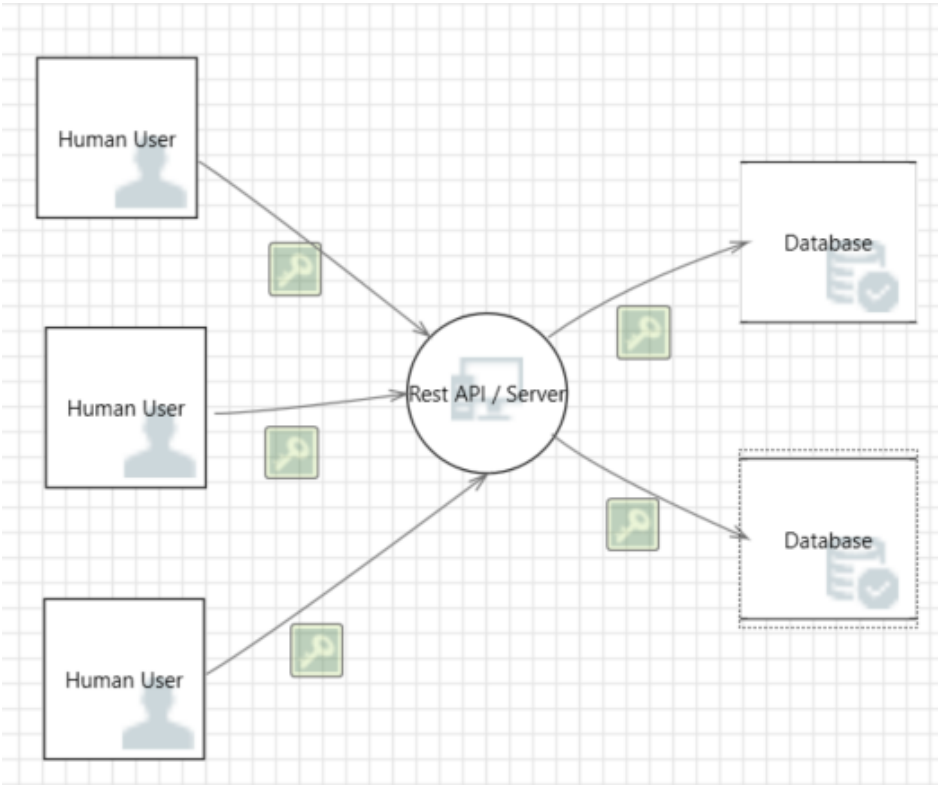
3 Tietoperusta

3.1 Mikä on REST API?

Tässä kappaleessa käydään läpi mikä on REST API. Millaisia ominaisuuksia ja rajoitteita REST rajapinnan käytössä on? On tärkeää hahmottaa rakenne ja toiminnallisuudet, että voidaan ymmärtää kokonaisuudessaan, myös turvallisuudelle välttämättömät piirteet.

API eli Application Programming Interface on pinta, jossa applikaatioiden sisäiset toiminnot tapahtuvat. Näistä nykyaikainen toteutustapa on REST (Reprensentational State Transfer), tämä on Roy Fieldingin kehittämä arkkitehtuurinen tyyli hajautettujen sovellusten väliseen kommunikaatioon. Pääsääntöisesti REST API on client-server-sovellusten väliseen keskusteluun, joka tapahtuu http(s)-protokollaa hyödyntäen. Kuvio 4 kuvaa perinteisen REST API:n toimintaa.

Muita REST rajoitteita ovat tilaton (stateless), välimuisti (cache), yhdenmukainen rajapinta (uniform interface), kerroksittainen järjestelmä (layered system) ja ladattava koodi (code-on-demand). (Fielding 2000, 96-103.)

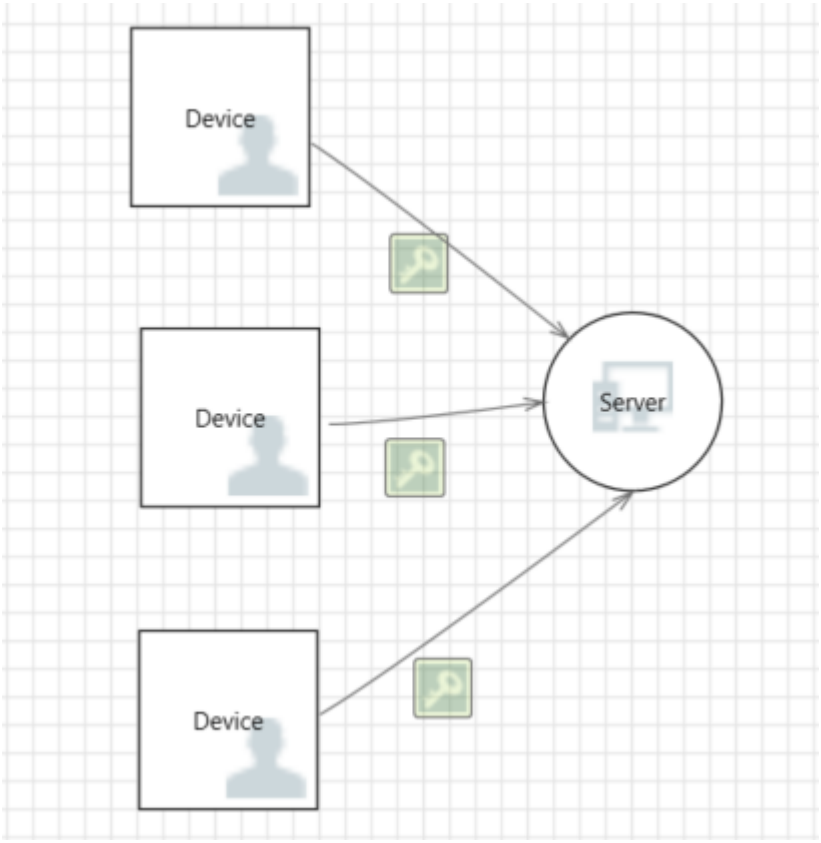


Kuvio 4 REST rajapinta

REST API:n funktioita voidaan verrata perinteisiin CRUD:n (create, read, update, delete) toimintoihin. REST:n käyttämä http-protokolla tarjoaa GET, POST, DELETE, UPDATE ja PATCH toiminnot.

3.1.1 Asiakas-serveri

Sovelluksen toiminta jaetaan kahteen alueeseen. Ensimmäinen osio on asiakkaan ja toinen serverin vastuulla. Asiakas useinten hoitaa käyttöliittymää ja palvelimen alue on hoitaa tiedon tallentaminen (Fielding. 2000. 45). Esimerkiksi asiakkaalla on käyttöliittymä www-selaimen kautta käytettävänä, jonka kautta hän käyttää palvelimen tarjoamia palveluita. Esimerkkinä yleiseen palvelimen tarjoamaan palveluun on tietokannan tiedonhaku ja tallentaminen. Tästä kuvauksena kuvio 5, jossa havainnollistetaan asiakas-serveri mallia.



Kuvio 5 Asiakas-serveri malli

3.1.2 Tilaton (stateless)

Client-server rajoitteen lisäksi REST sisältää tilattomuus-ominaisuuden. Tämä edellyttää sitä, että session tilaa ei tallenneta palvelin puolelle. Jokainen pyyntö, jonka API tekee on itsenäinen kokonaisuus, eli sisältää kaiken tarvittavan tiedon jota palvelin tarvitsee käsittelyyn. Koko sessio tallentuu asiakkaalle kokonaisuudessaan. (Fielding 2000, 47.)

Etuna tästä ominaisuudessa on skaalautuvuuden parantuminen, palvelin ei talleta pyyntöjen välistä tilaa. Tämä johtaa siihen, että palvelimen rakenne yksinkertaistuu ja resurssien vapauttaminen tapahtuu tarvittaessa nopeasti. Etuna voidaan myös nähdä järjestelmän luotettavuuden ja näkyvyyden parantuminen. Eli järjestelmän valvonta helpottuu, API-kutsusta nähdään suoraan pyynnön tyyppi. Luottavuuden parantuminen kohdistuu siihen, että järjestelmä pystyy palautumaan vikatilanteissa nopeammin. (Fielding 2000, 79.)

Tilattomuuden haasteena voidaan pitää verkon kuormittumista. Kaikki oleellinen tieto välitetään pyynnön yhteydessä, eli asiakkaan ja palvelimen välistä kontekstia ei tallenneta palvelimeen. Tämä myös vähentää kontrollia palvelimen päässä sovellukseen, käyttäminen tulee riippuvaiseksi asiakkaan tavasta käyttää sovellusta. (Fielding 2000, 79.)

3.1.3 Välimuisti eli cache

Tilattomuudesta johtuvien suorituskykyongelmien korjaamiseksi REST omaa välimuisti-rajoitteen. Tämä edellyttää palvelimelta sen, että vastauksessa on oltava mukana tieto, voidaanko vastauksen sisältö tallentaa välimuistiin. (Fielding 2000, 79.)

Välimuisti-rajoitteella voidaan jättää jotkin pyynnöt kokonaan tai osittain pois. tämä parantaa suorituskyvyn lisäksi käyttäjä kokemusta ja skaalautuvuutta. Tämän ansiosta viive vähentyy. Haittana voidaan katsoa luotettavuuden heikentyminen. Välimuistiin voi jäädä vanhaa tietoa. (Fielding 2000, 80.)

3.1.4 Yhdenmukainen rajapinta

Rajapinnan yhdenmukaisuus yksinkertaistaa kokonaisarkkitehtuuria sekä parantaa komponenttien kanssakäymisen näkyvyyttä. Tämä mahdollistaa itsenäisen kehityksen komponenteille asiakas-palvelin-järjestelmässä.

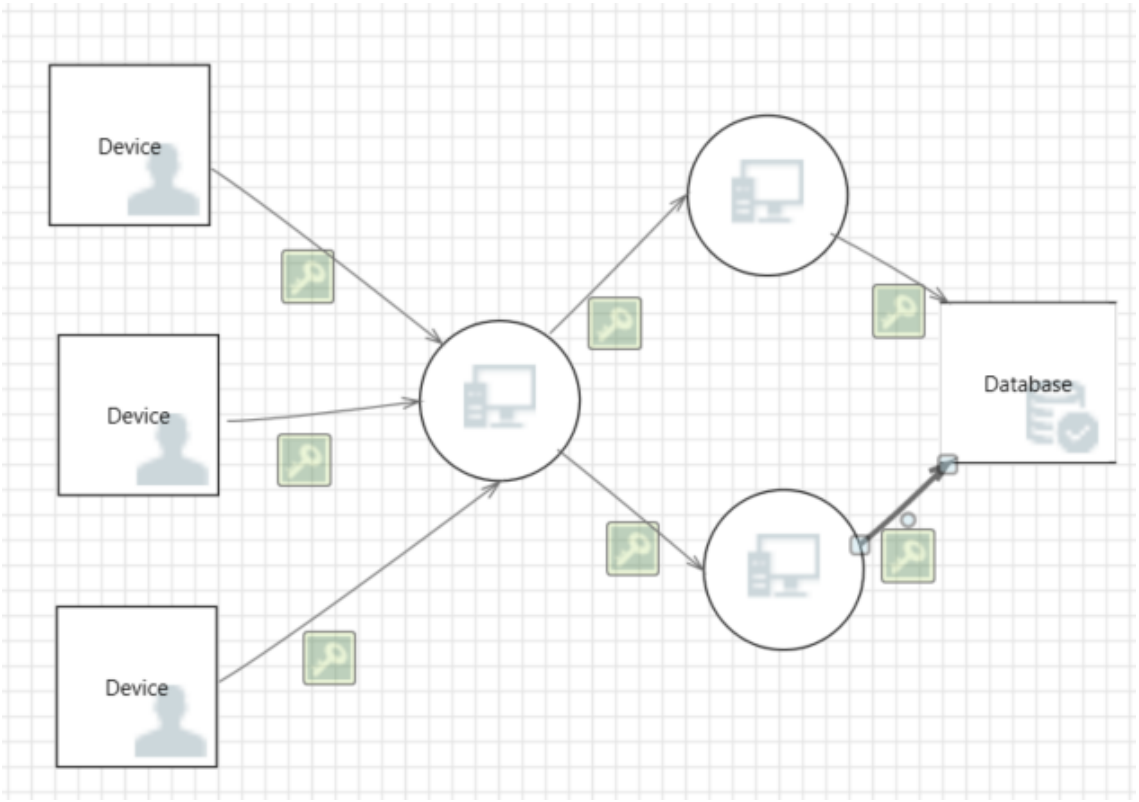
Haittapuolena on suorituskyvyn heikentyminen. Tämä johtuu siitä, että tieto kulkee samassa muodossa riippumatta siitä mikä olisi paras muoto sovelluksen kannalta.

Fielding (2000, 82) tarkentaa yhdenmukaisen rajapinnan rajoitteita neljällä muulla lisärajoitteella: resurssien manipulointi esitysten kautta, itseselitteiset viestit ja hypermedian käyttö tilakoneena sovelluksessa, resurssien tunnistaminen.

REST:ssä resurssilla tarkoitetaan käsitettä, joka voidaan käsitellä yksilöllisenä. Tämä voi olla dokumentti, kuva tai esimerkiksi hetkellinen palvelu esimerkiksi tämän hetkinen sää.

3.1.5 Kerroksittainen järjestelmä

Kerroksittainen järjestelmä on järjestelmä, jossa on kerroksia, jotka kommunikoivat keskenään. Näillä kerroksilla on hierarkia, ja ne voivat kommunikoida vain välittömästi ala- sekä yläpuolelleen kerroksiin. Palvelimen ja asiakkaan välisten kerrosten kautta voidaan kapseloida vanhoja järjestelmän osia, ja näiden vanhojen järjestelmän osien päälle pysytään rakentamaan uudistettuja rajapintoja. Välikerroksien avulla voidaan myös kehittää tietoturvaa. Esimerkiksi palomuuri voidaan asettaa vahtimaan järjestelmän ja muun internetin välillä. (Fielding 2000, 83.) Tätä ominaisuutta esitetään kuviossa 6.



Kuvio 6 Kerroksittainen järjestelmä

3.1.6 Ladattava koodi eli code-on-demand

Palvelimella on mahdollisuus laajentaa asiakassovelluksen toiminnallisuutta lähettämällä vastauksien mukana koodia, ja tämä koodi ajetaan asiakkaan puolella. Tällä yksinkertaistetaan sovelluksen kehittämistä ja vähennetään asiakaspään toteutettavaa toiminnallisuutta. Tämä ominaisuus heikentää järjestelmän näkyvyyttä, ja siksi tämä on vapaaehtoista REST-arkkitehtuurissa. (Fielding 2000, 84.)

3.2 REST turvallisuus

Tässä osiossa käydään läpi REST-rajapinnan turvallisuutta ja sen uhkia. Tarkoituksena on selventää, millaisiin asioihin pitää varautua tämän tyyppisissä API ratkaisuissa ja kuinka niiltä voidaan suojautua.

Fielding (2000) ei ota kantaa REST API:n yksittäisiin turvallisuusuhkiin, mutta on todennut REST-rajapinnan helpottavan turvallista suunnittelua ja toteutusta. Yhdenmukaisuus helpottaa kehityksen turvallisuusuhkien huomioon ottamista ja käsittelyä.

Fielding (2000) ei ole myöskään määritellyt tiettyä tekniikkaa käytettäväksi REST-rajapinnan käyttämiseksi. Sovelluksen voi toteuttaa valitsemallansa kielellä ja tietokantana voidaan käyttää haluttua tietokantaa. Hyökkäyksiä on eri tekniikoita vastaan, esimerkiksi SQL-injektio, näin ollen REST-arkkitehtuuri itsenään ei poista uhkia vaikkakin REST toimii kerroksittain ja rajaa pääsyä kerrosten välillä. Sovellus, joka on toteutettu REST API:lla ei ole suoraa pääsyä tietokantaan, mutta välissä olevaa rajapintaa voidaan käyttää tietokannan manipulaatioon.

Sovelluksen turvallisuutta suunnitellessa valmiiden ohjelmistokehysten ja toteutuksien valitseminen on suositeltavaa. Eli REST pohjaisessakin toteutettaessa kannattaa käyttää mahdollisimman paljon turvallisia, valmiita ja toimivia komponentteja. (OWASP, 2017.) REST:n avulla muun muassa saadaan hyödynnettyä REST:n tukemia autentikaatioita.

Turvallisuuden kannalta on tärkeää, että komponentit testataan yksikkötesteinä. Tämän vuoksi itse ohjelma rakennetaan erillisistä osista eli komponenteista. Tämä mahdollistaa yksittäisen osion vaihtamisen ja muuttamisen tarvittaessa.

REST turvaa funktioita käyttäjähallinnalla. Auktorisoinnilla pystytään turvaamaan eri funktiot käyttäjiltä luotettavasti ja helposti testattavasti. Päätepisteet voidaan määritellä, jolloin voidaan määrittää virheellisille kutsuille statuskoodeja kuten 404 (Not Found) (OWASP 2017). Sallittuja päätepisteitä voidaan määritellä RBAC:n mukaisesti ja sallia pääsy käyttäjälle.

Auktorisointi voidaan toteuttaa lisäämällä tunnisteet käyttäjille. Esimerkiksi isAuthenticate tai isAdmin. Tällöin esimerkissä admin käyttäjän esimerkkikuvassa varmistetaan, että URL sisältää tunnisteen joka täsmää kirjautuneeseen henkilöön. Seuraavan ohjelmointileikkeen mukaisesti.

```
module.exports = function(req, res, next) {
if {
         req.user.role ==== 'Admin' ||
         req.user.id.toString() ===
req.params.personID.toString()
} {
return next();
}
return res.forbidden('You are not permitted to perform this
action');
};
```

3.2.1 XSS

XXS-hyökkäykset ovat myös yksi REST-rajapinnan haavoittuvuuksista, muiden selainpohjaisten verkkopalveluiden mukaisesti (OWASP, 2017). Tämän hyökkäyksen varalta tietokantaan tuleva data ja muille käyttäjille suunnatut syötteet pitää suodattaa tai käsitellä muulla vastaavalla tavalla. Tarkoituksena on estää muiden käyttäjien haitallisen koodin syöttäminen. (OWASP, 2017). Tästä esimerkkinä seuraava.

```
Name

<scipt>alert('hello')</script> Test name

Description

<script>aler('hello')</script> Test description
```

XSS estämiseen on olemassa erilaisia ratkaisuja. Esimerkiksi sanitiza-html avulla voidaan poistaa syötteistä textarea-, script- ja style-tunnisteita. (P'unk Avenue, 2018). Lisäksi rajapintaa voidaan suojata määrittelemällä tarkasti sallitut syötteet. Syötteet voidaan myös tarkistaa palvelinsovelluksen toimesta, ja hylätä jos syötetty data on muuta kuin sallittua.

3.2.2 Cross site request forgery

Cross site request forgeryltä suojaudutaan palvelimen päässä regeneroitavilla suojatunnuksilla. Jokainen API-kutsu tulee suojata CSRF-hyökkäykseltä niin, että API-kutsu hyväksytään jos se sisältää hyväksyttävän avaimen. CSRF-hyökkäyksestä saadaan tehoton, jos käytetään JWT-avaimia tai OpenID Connectin tyylisiä ratkaisuja, joissa ei käytetä todentamiseen evästeitä. (OWASP, 2018). Näissä tapauksissa on määritelty autentikointia varten tunnus, joka on voimassa määritellyn ajan. Eli määritetyn ajan jälkeen kutsut eivät enää toimi.

3.2.3 Arkaluontoinen materiaali ja kryptaaminen

Arkaluontoista materiaalin suhteen REST API täytyy suunnitella niin, ettei materiaali paljasteta osapuolille, jotka voisivat käyttää sitä väärin. Asiakkaan ja serverin välinen liikenne tulee salata SSL/TLS yhteyksien avulla. Tietokannan puolella on hyvä suojata nämä tiedot salaamalla tai hajauttamalla tiedon rakennetta.

Tiedon salaamiseen on monenlaisia ratkaisuja. Esimerkiksi brypt-nodejs-kirjastolla voidaan salata merkkijonot Blowfish-algoritmillä. Ja lisänä tietokanta voidaan konfiguroida niin, ettei siihen saa yhteyttä muualta, kuin samalta palvelimelta.

3.2.4 Injektiot

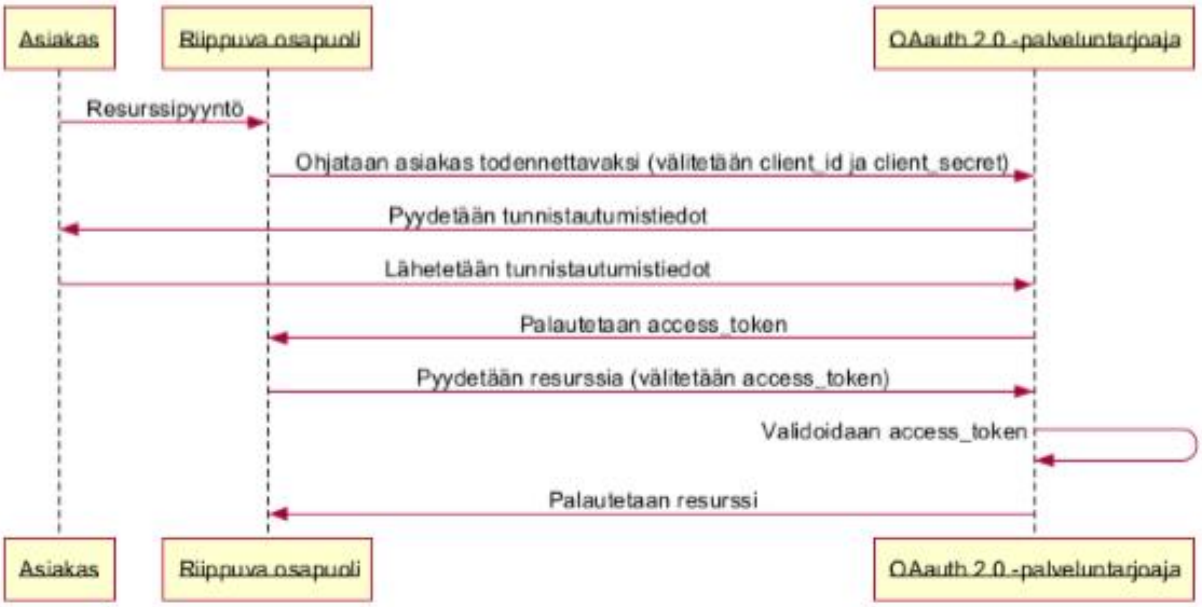
Injektio on hyökkäys, jossa pyritään manipuloimaan tietokantaa ja sen sisältöä. Ideana on syöttää tietokantaan ohjaavia komentoja. Esimerkiksi haku-, poisto-, lisäys- ja muokkauskäskyjä (OWAPS, 2017).

```
var username = req.body.username;

var password = req.body.password;

loginSQLCommand = "SELECT * FROM Users WHERE username = '" + username
+ "' AND password = " + password + "'";
```

SQL-injektioiden estämiseksi kannattaa välttää suoraa arvojen sijoittamista SQL-lauseisiin. Lisäksi on olemassa suojaamiseen useita erilaisia kirjastoja. Kirjastojen funktionaalisuus on usein, että ne tunnistavat syötteitä ja kriittisiä erikoismerkkejä ja merkitsevät ne niin etteivät ne toimi. Erikoismerkkien eteen usein lisätään \-merkki. (OWASP, 2017).

3.2.5 OAuth 2.0

OAuth 2.0 on avaimen auktorisointi standardi. Käyttäjä tunnuksen ja salasanan sijaan tunnistautuminen perustuu generoituihin avaimiin. Tarkoituksena on tarkistaa, että käyttäjällä on lupa funktioon, jota hän kutsuu.

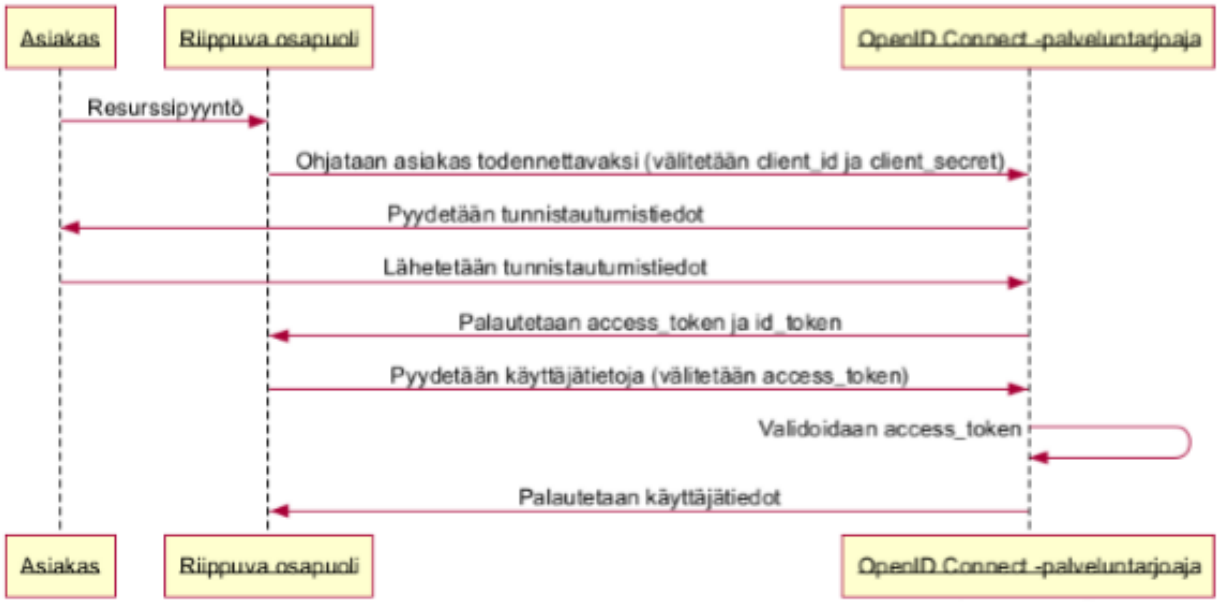


Kuvio 7 OAuth 2.0-sekvenssikaavio

Kun käyttäjä haluaa käyttää tarjoajan resursseja, hänet ohjataan asiakastunnuksen ja -salaisuuden kanssa tunnistautumaan sivulle, ja autentikoitumaan palveluntarjoajan sivulle. Todentamisen jälkeen käyttäjä ohjautuu uudelleenohjaussivulle. Tällöin käyttäjälle välitetään pääsyavain eli access tokeni. Samalla generoituu päivitysavain eli refresh tokeni, jolla voidaan uusia pääavain (Chapman and Chapman, 2012).

3.2.6 OpenID connect

OpenID Conenct on OAuth 2.0:n päälle vuonna 2014 julkistettu avoin autentikointistandardi. Tällä ratkaistaan OpenID:hen liittyviä turvallisuus- ja käyttöongelmia.



Kuvio 8 OpenID connect

Toiminnaltaan OpenID connect on samankaltainen kuin OAuth 2.0. Lisänä tässä kumminkin on tunnistusavaimen eli id tokeni pääsyavaimen lisäksi, jolla voidaan varmistaa käyttäjän identiteetti. Turvallisuuden kannalta on tärkeää, että liikenne turvataan SLL/TLS-salauksien kautta. Muuten OpenID connect ei takaa turvallisuutta.

Ll ja Mitchell (2015) kirjoittivat, että väärin asetetusti OpenID connect altistaa palvelun CSRF, man in the middle ja istunnon kaappaus hyökkäyksille. Tätä riskiä voidaan minimoida käyttämällä palveluntarjoajan eli Googlen oppaita ja varmistamalla SLL/TLS-salauksien käytön.

3.3 Monitorointi ja valvonta

Osiossa käydään läpi, millaisilla työkaluosilla valvotaan ja monitoroidaan API liikennettä, julkaisua ja toimintaa. Nämä osiot ovat välttämättömiä yrityksen API julkaisun kannalta ja erilaisia ratkaisuja on olemassa useita, mutta pääsääntöisesti ne sisältävät seuraavia osioita.

3.3.1 API:en hallinta

Hallinnan kannalta oleellista on käytännöt ja standardit. Tarkoituksena on API puolen käytön tehostaminen ja turvallisuus. Usein nämä järjestelmät, ovatkin palomuurituotteen päälle rakennettuja teknologiaratkaisuja.

Järjestelmä sisältää neljä eri komponenttia.

3.3.2 API gateway

Komponentti, jossa tapahtuu API:n ajoaikainen suoritus. Tämän kautta API:t julkaistaan turvallisesti, hallitusti ja tehokkaasti. Toiminnallisuuksiin kuuluu esimerkiksi reititys, orkesterointi ja sisällön ja rakenteen muokkaaminen. Gateway myös toimii turvallisuuden osana valvoen muun muassa API:en käyttöä luvattomalta käytöltä, uhkien tunnistaminen sekä torjuminen. Lisäksi myös torjunta ja salaus sekä allekirjoittaminen.

3.3.3 API manager

Managerin alueisiin kuuluu versiointi, julkaiseminen, tietoturvassa käyttäjät ja käyttöoikeudet ja poistaminen. Tämän osuuden puolella myös määritellään API:en käyttämät luvat. Esimerkiksi tietoturvan, reitityksen sekä sanomien muunnoksissa. Tämä osio voi sisältää myös analytiikkaa ja monitorointia.

3.3.4 API portaali

Itsepalveluportaali, tätä kautta kehittäjät voivat selata julkaistuja API-julkaisuja ja testata näitä käyttäjäoikeuksista riippuen. Portaalissa on myös API:en dokumentaatio, ja toiminnallisuuden ideana on, että kehittäjä voi valikoida tarvitsemansa API:t sovelluksiensa käyttöön. Portaaleihin voidaan sisältää foorumeita ja blogeja tiedon jakamiseen. Portaali voidaan tehdä aina asiakkaan tarpeiden mukaiseksi ja näköiseksi.

3.3.5 API analytiikka

Analytiikka voi olla API managerissa tai portaalissa. Tai vaihtoehtoisesti olla ulkoinen komponentti API hallinta kokonaisuudessa. Tämän avulla voidaan arvioida API:n liiketoiminnan hyötyä ja käyttöä. Ja hyödyntää tätä dataa kapasiteetti tarpeen suunnittelussa.

4 Vaatimukset

Tarpeena on luoda osa järjestelmään, joka kykenee tunnistamaan turvallisuuspoikkeamia lokidatan perusteella. Lokidata esimerkkitapauksessa menee pilvipalvelutarjoajan lokienhallintaan. Turvallisuus-auditoinneissa on havaittu, joidenkin valvottavien API:en vuotavan päätepisteitä, ja näiden tunnistamiseen ja valvontaan halutaan toimiva valvonta kokonaisuus. Tämän avulla voidaan parantaa yleisturvallisuutta, ja virheitä tuottavien API:en kehittäjille voidaan ilmoittaa havainnoista.

Tarkoituksena on, että tapahtumista pystytään tunnistamaan tarvittavilla aikaväleillä poikkeavuuksia. Poikkeavuustunniste nimittäin voi muuttua, riippuen aikavälistä jolta lokitiedos-

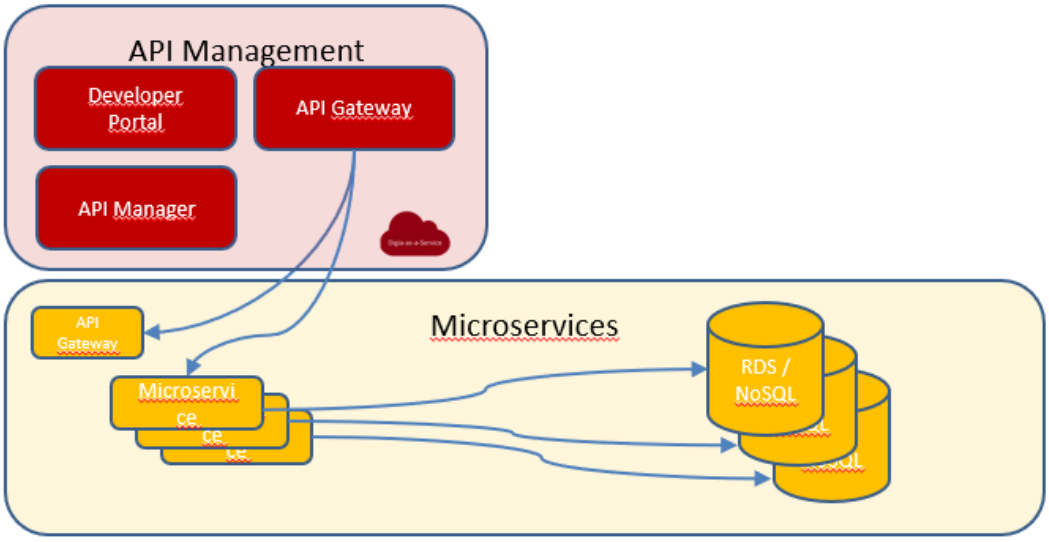
toja tutkitaan. Nimittäin useimmat työkalut arvioivat tapahtuman toistuvuutta muihin tapahtumiin.

Työkalun pitäisi olla helposti toimiva ja hyödyntää koneoppimista. Eli työkalun pitäisi osata tutkia tiedostoja itsenäisesti ilman sen erikoisempaa ohjausta. Tarkoituksena on, että sovellus tuottaisi löydöksistään raporttia jonka kautta voidaan asettaa hälytyksiä poikkeavuuksista. Vastaavia kokonaisuuksia on käytössä vikatiloihin, mutta ei turvallisuuteen painottuen.

4.1 Kehittämisympäristö

Verkkoympäristöjen digitaalisen vallankumouksen jälkeen palvelut ovat siirtyneet verkkoon. Tämän vuoksi tietoturvassa poikkeamientunnistus on tullut huomattavan tärkeäksi osaksi tietoturvasuunnittelua ja kehitystä. Poikkeavuuksien tunnistuksella pystytään tunnistamaan palveluun kohdennettavia palvelunestohyökkäyksiä ja yrityksiä käyttää palvelua vääriin manipuloimalla komentoja. (Adams, 2014)

Verkkoliikenteen valvonta ja poikkeamien tunnistaminen on tärkeä osa pilviympäristöön rakennettavia kokonaisuuksia.



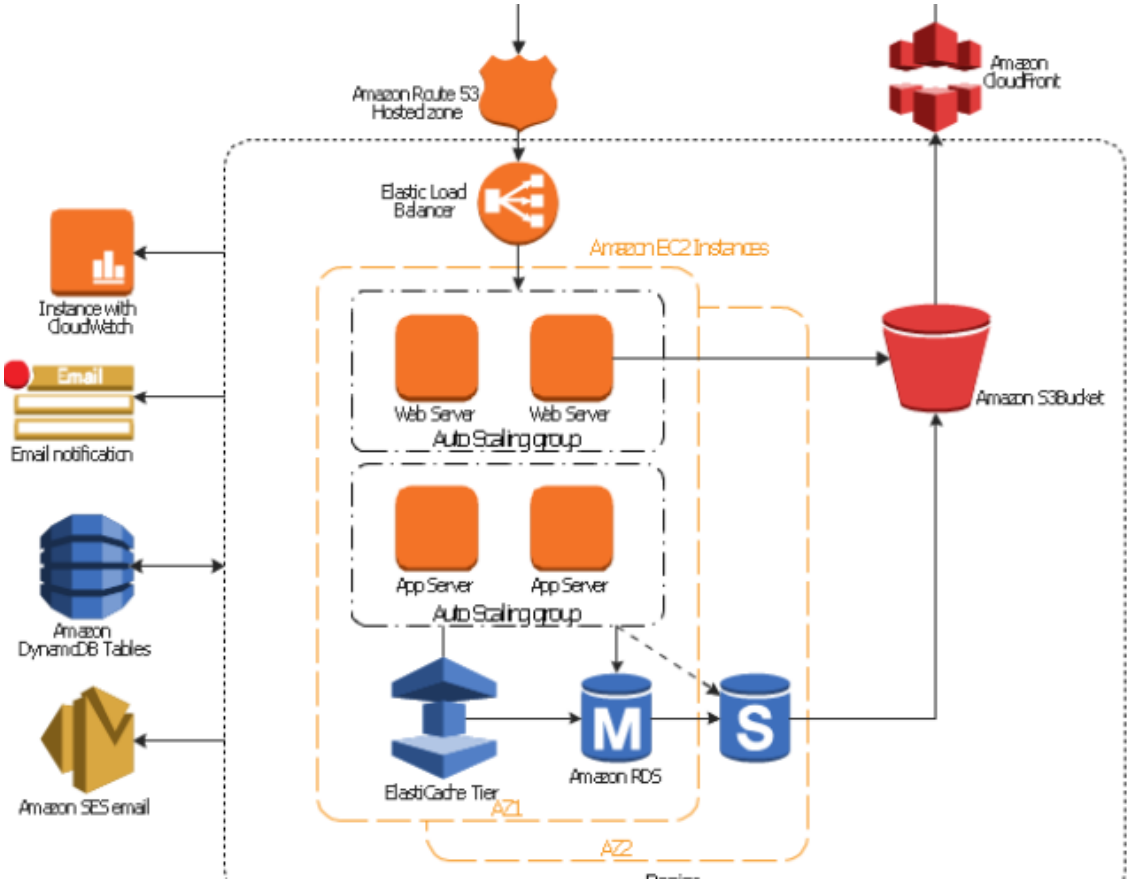
Kuvio 9 API management ja mikropalvelut

Kehityskohteen ympäristö sijaitsee palveluntarjoajan pilviympäristössä ja pääsyä hallitaan palomuurien ja kerroksittaisen toiminnan kautta. Gateway valvoo pääsyä ja orkesteroi liikennettä eteenpäin, samalla valvoen APi:en käyttöoikeuksia. Yksinkertaistettu arkkitehtuurikuvaus kuviossa 10.

API-hallinta toimii täysin omana palkkinaan kokonaisuudessa pilven sisällä, ja tämän toiminnasta muodostuu lokitiedostoa, joka siirtyy lokienhallintaa suorittavaan osioon ja kantaan. Lokien hallinta tunnistaa virheitä ja hälyttää virheistä eteenpäin.

Näitä lokeja voidaan hyödyntää kehittämään oma ohjelmansa, jolla tunnistetaan poikkeavia tapahtumia eli poikkeavuuksia tapahtumista. Tämä uusi osio lähettää ilmoitukset ja hälytykset eteenpäin havaitessaan turvallisuus poikkeamia tallennettavissa tapahtumissa.

Instansseina pilvipalvelussa toimivat Linux-pohjaiset järjestelmät, joiden sisällä pyöritetään vuorostaan käytettäviä palveluita. Näiden ympärille rakennetaan eri roolituksen sisältäviä palomuureja ja käyttäjähallintoja. Instanssit on vuorostaan pystytetty erillisiin ryhmiin, ja näihin on rakennettu kuormanjakajat (load balancerit) määrittämään tarvetta ja työkuorma instanssien välille. Tästä esimerkkinä AWS (Amazon web servicen) suosituksen mukainen arkkitehtuuri esimerkki kuviossa 11.



Kuvio 10 Esimerkki Amazonin verkkoarkkitehtuuri suosituksen mukaisesta toteutuksesta (Amazon web services 2018)

Erilaisilla rajauksilla tehdään tietoverkkoarkkitehtuurin skannaamisesta vaikeampaa. Ja toiminta kerrokset ovat toisistaan erillisissä osioissa, joka vaatii erillisiin kerroksiin pääsyä että

vahinkoa itse verkkorakennetta voitaisiin hyödyntää haitalliseen toimintaan. Kuvion 11 kuvassa kuvataan Amazon Web Servicen-virtuaaliympäristön esimerkki arkkitehtuuria, jossa on rakennettu pääsy kerroksittaisesti eri ympäristöihin.

5 Toteutus

Tämä osio käsittelee itse ongelmanratkaisua ja työkalun rakentamista. Osiossa kuvataan, kuinka ratkaisu toimii ja perustellaan valittuja työvälineitä. Lopputuloksena on ratkaisun beta-versio, jota voidaan hyödyntää havaintojen tekemiseen ja ratkaisun toimivuuden mittaamiseen testaamisella.

Ohjelmointikielinä yleisesti ison tietomäärän käsittelyyn käytetään Pythonia ja R-kieltä. Työkalu luodaan R-kielellä, koska tämä perustuu C-kieleen ja ei vaadi niin paljoa prosessointitehoa toimintaympäristöltään.

Ongelmanratkaisua varten käytetään lokidataa, jota tutkimalla voidaan havainnoida poikkeamia tietoliikenteessä ja kehittää näiden valvontamenetelmiä. Suuren datamäärän vuoksi lokitiedoston läpikäyminen käsin on vaivalloista ellei mahdotonta. Tämä vuoksi lokitiedoston läpikäymiseksi tarvitaan työkaluja tai algoritmejä, joilla lokitiedosto voidaan muuntaa lukukelpoiseksi.

Vaihtoehtoja on monia ja näistä valikoitui tehtävään neljä menetelmää. Nämä ovat:

- Ikkunointi
- Parsinta
- TFIDF algoritmi
- PCA & poikkeavuuksien tunnistaminen

5.1.1 Ikkunointi

Apte, Skilliforn, Liu & Parthasarathy (2007) toivat esille, että tietoturvan valvonnassa tiedonlouhinnassa on tärkeää käyttää menetelmää, jota kutsutaan ikkunoinniksi. Menetelmän tarkoituksena on selkeyttää lokien luettavuutta ja rakennetta, että näitä olisi helpompi lukea ja tulkita.

```
#pat1 <- ""
pat1 <- "gtid\\([^()]+\\)"
pat2 <- "trans\\([^()]+\\)"
#pca variance included
pcathresh <- 0.95
#spe confidence level
```

```
alpha_spe <- 0.005
#window size (used only if pat1 == "")
window <- 10
```

Ideana on, että lokiriveistä poistetaan kaikki ylimääräiset merkit pois. Esimerkiksi \([^()\)-merkit poistetaan ja asetellaan saman loki id:n omaavat osiot samaan alueeseen. Tarkoituksena on luokittaa kaikki osiot yhtenäisten elementtien mukaisesti toisiinsa. Alla ohjelmointikatkelmassa on sovelluksen toteutuksesta, jossa määritellään poistettavia elementtejä ja luokitellaan ikkunoinnin asetukset tarvittavaan malliin.

```
totalrows <- nrow(data)

#if pattern given, make id using it. otherwise use windowing
if(pat1 != ""){
  if(pat2==""){pat2<-"xxxyyyzzz####"} # dummy pattern
  #extract gtid and transid
  data[,":="(
    gtid=str_extract_all(V1, pat1,simplify=T),
    trans=str_extract_all(V1, pat2,simplify=T)
  )]
  #make id column
  data[,":="(
    id=ifelse((is.na(gtid) | gtid ==""),trans,gtid)
  )]
}else {
  chunk <- ceiling(1:nrow(data)/window)
  data <- cbind(data,id=as.character(chunk))
}
```

5.1.2 Parsinta

Parsinnassa on kyseessä ikkunoinnin lisäksi tarvittava osuus lokin saamiseksi ymmärrettäväksi. Ideana on, että aakkosissa olevat merkit sisältävät osuudet kerätään sanoiksi vektorin sisältä. Ja käytetään Englannin sanankirjaa muodostaakseen näistä ymmärrettäviä ja hyväksyttäviä sanoja. Seuraavan ohjelmointikatkelman mukaisesti.

```
data_sample <- data_sample[!""]

#remove punctuation marks
data_sample[,":="(
  text=gsub("[[:punct:]]"," ",text)
)]
#keep alphabetic chars and spaces; split to vector by space
data_sample[,":="(
  word-
split=strsplit(gsub("[^[:alpha:][:space:]]","",tolower(text)),"\\s+")
```

```
)]

#remove too short/long words, keep only proper english words, no names
(stopwords not removed!)
#create list of allowed words
dictwords <- tolower(GradyAugmented)
dict_dt <- data.table(dictwords,key =
names(dictwords));setnames(dict_dt,"word")
names_fin <- read.csv(file="etunimitilasto_vrk.csv", head-
er=TRUE,stringsAsFactors = F)
allnames <- c(tolower(NAMES$name),tolower(names_fin$Etunimi))
allnames_dt <- data.table(allnames,key =
names(allnames));setnames(allnames_dt,"word")
#remove names and too short/long words
OKwords <- fsetdiff(dict_dt,allnames_dt)
OKwords <- subset(OKwords,nchar(word)>3 & nchar(word)<15)
rm(allnames,allnames_dt,dict_dt,dictwords,names_fin)
gc()

uniquewords_dt <- da-
ta.table(unique(unlist(data_sample$wordsplit)));setnames(uniquewords_d
t,"word")
uniquewords_dt <- subset(uniquewords_dt,word %in% OKwords$word)
```

Menetelmällä saadaan riveistä selkeitä ja ymmärrettäviä ihmiselle ja helpotetaan sekä nopeutetaan analysointiprosessia. Tulos on esimerkiksi: `[xx.xx.x.xxx]: could not establish SSL for incoming connection.`

5.1.3 TFIDF algoritmi

Beel, Gipp, Langer ja Breitinger (2015, 305-3018) esittävät, että tfidf menetelmällä saadaan louhittua lokitiedostoista sanoja ja arvotettua näiden tärkeyttä. Menetelmä tutkii lokitiedoston rivejä ja laskelmoi eri sanojen toistuvuutta ja näiden poikkeavuuksia.

$$\mathrm{tf}(t,d) = 0.5 + 0.5 \cdot \frac{f_{t,d}}{\max\{f_{t',d} : t' \in d\}}$$

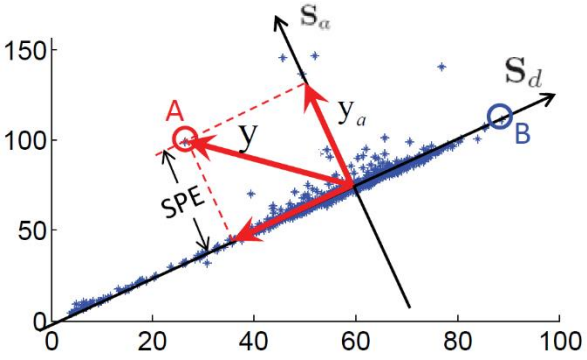
Kuvio 11 TFIDF algoritmi

Algoritmi laskee sanojen toistuvuuksia ja tiputtaa turhaksi luokiteltavia sanoja sananjonosta pois TFIDF matriksista. Kyseisellä menetelmällä lasketaan sanojen arvoa dokumentin suhteen, menetelmää muun muassa käytetään hakukoneissa hakutulosten arvottamiseen. Menetelmässä tunnistetaan siis tärkeimmät sanat. Esimerkiksi voidaan määrittää sanoiksi ”error” ”SSL” ”not”. Algoritmi kerää tuloksena vain ne dokumentaation osat, joissa esiintyy kyseiset kolme sanaa. Ja laskelmoi sanojen toistuvuuden riveiltä ja dokumenteista, ja muodostaa näiden mukaan dokumentin arvon. Seuraavalla sivulla olevan ohjelmointikatkelman mukaisesti.

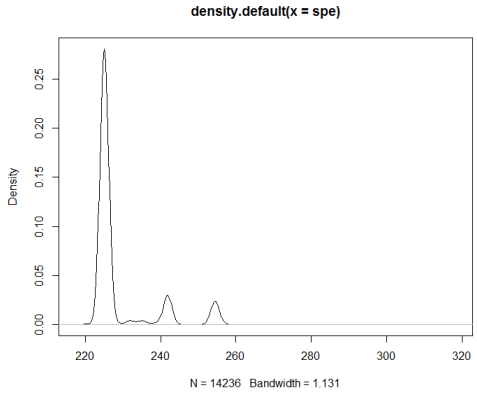
```
#calculate word frequency
system.time({
  cl<-makeCluster(cl_cnt)
  clusterEvalQ(cl,library(stringr))
  clusterExport(cl, "uniquewords_dt")
  wordfreq <- clusterApply(cl, data_sample, function(y) {
    t <- table(y)
    v <- vector(mode="integer",
length=length(uniquewords_dt$word));names(v) <- uniquewords_dt$word
    v[names(t)]<-t
    as.vector(v)
  })
  stopCluster(cl)
})

rm(data_sample,cl)
gc()

#finalize
wordfreq <- matrix(unlist(wordfreq,use.names = F),
nrow=length(wordfreq), byrow=T)
colnames(wordfreq) <- uniquewords_dt$word
```

5.1.4 PCA & poikkeaman tunnistaminen

Zwietasch (2014, 7) esitteli menetelmää PCA (Principal Component Analysis) opinnäytetyössään. Menetelmää käytetään käsitellessä suurta tietomäärää, joissa on monta ulottuvuutta. PCA:lla tunnistetaan toistuvia kuvioita lokitiedostoista ja verrataan näiden toistuvuutta muihin tapahtumiin.



Kuvio 13 Poikkeaman tunnistaminen



Kuvio 12 Poikkeama graafi

PCA:lla määritellään tapahtuman eroavaisuuden suuruus. Esimerkissä käytetään 95% poikkeavuutta eroksi poikkeaman ja normaalin välille. TFIDF käsittelemät rivit siirretään poikkeaman tunnistus jonoon, ja näiden avulla lasketaan tapahtuman SPE arvo. Korkea SPE arvo ilmaisee seuraavan ohjelmointikatkelman mukaisesti eroavaisuuden normaalista liikenteestä ja merkitsee tapahtuman poikkeamaksi.

```
spe <- col.norm(y_a)^2
#calculate spe control limit
#according to: Fault Detection and Diagnosis Based on PCA and a New
Contribution Plots
spe_cr <- spe^(1/3)
mu <- mean(spe_cr)
sd <- sd(spe_cr)
alpha <- alpha_spe #confidence limit
z <- qnorm(1-alpha, 0, 1)
Q_thres <- (mu + sd*z)^3

anom_ind <- which(spe > Q_thres)
anomalies <- data.frame(spe =
spe[anom_ind],id=ids[anom_ind],stringsAsFactors = F)
anomalies <- anomalies[with(anomalies, order(-spe)), ]

#add rownumbers if windowing done based on row
if(pat1 == ""){
  startrow <- as.character((as.numeric(anomalies$id)-1)*window+1)
  endrow <-
as.character(pmin((as.numeric(anomalies$id))*window,totalrows))
  anomalies$rows <- paste(startrow,"-",endrow,sep = "")
```

Tämän prosessin jälkeen työkalu muodostaan tiedon poikkeamista, jotka järjestäytyvät algoritmin määrittämän arvon mukaiseen järjestykseen. Ajoaika kokonaisuudelle on testidatan osalta noin 12 sekuntia.

6 Testaaminen

Tässä osiossa käydään läpi työkalun testaamista ja sen tuottamia havaintoja. Testausta suoritetaan testidatan avulla ja havainnoidaan, millaisia tuloksia saadaan sekä mitä näistä tuloksista voidaan päätellä. Käytännössä työkaluun ajetaan lokitiedostoja ja varmistetaan, että nämä algoritmit toimivat niin kuin pitääkin ja vertaillaan tuloksia ja rivejä lokitiedoston riveihin.

Testaamisen tarkoitus on laadun varmentaminen tuotteessa. Testauksella selvitetään saavuttaako työkalu sille asetetut vaatimukset, että ohjelmisto toimii oletetusti ja että ohjelma kyetään ottamaan käyttöön halutuilla ominaisuuksilla. (Kaner 2006)

Testeissä käytettävä lokitiedosto on pilviympäristön lokien hallintaohjelman käyttämää .log-formaattista tiedostotyyppiä.

6.1 Ensimmäinen testi

Ensimmäisessä vaiheessa varmistetaan työkalun toimintaa. Lokitiedostoon manipuloidaan satunaiseen kohtaan, kaksi varmasti muusta virrasta poikkeavaa merkintää. Näin voimme todentaa, että kirjoitettu työkalu toimii ja havaitsee poikkeamat. Näinä merkintöinä käytetään nyt merkintää gtid(testing2). Ajettavat algoritmit ovat:

- Ikkunointi
- Parsinta
- TFIDF algoritmi
- PCA & poikkeavuuksien tunnistaminen

Lokitiedostona käytetään asiakasympäristöstä kerättyä esimerkki lokitiedostoa, jotta saadaan todettua työkalun soveltuvuus itse työkalun tulevaan käyttöympäristöön. On tässä vaiheessa tärkeää, että käsitelty tieto on oikean mallista ja sisältää oikean rakenteen.

Testin osuudessa ajetaan kaikki työkalun algoritmit lokitiedostolle.

Lokitiedosto ajetaan tiedoston läpi. Analyysin kesto 12 sekuntia.

```
317.568540397868,gtid(testing2)
274.568901471778,gtid(testing1)
257.861987155026,trans(10384448)
257.861987155026,trans(10385808)
257.861987155026,trans(10386912)
```

Merkityt rivit nousevat analyysissä ylimmiksi poikkeavuuden mukaisesti. Eli työkalu havaitsee selkeästi poikkeamat lokitiedostossa. Eli työkalun toiminnallisuus on tarkoituksen mukainen.

6.2 Toinen testi

Ensimmäisessä testissä totesimme, että työkalu toimii ja ohjelma suoriutuu tehtävästään. Löysimme itse asettamme poikkeukset lokitiedostoista, ja ne arvottuivat oikein tuloksessa. Toisessa osiossa testidatana käytetään puhdasta lokitiedostoa, minne ei olla lisätty erikseen ylimääräisiä rivejä. Testissä ajetaan työkalun kaikki valitut algoritmit.

- Ikkunointi
- Parsinta

- TFIDF algoritmi
- PCA & poikkeavuuksien tunnistaminen

Itse testattava lokitiedosto on asiakasympäristöstä otettua ja sisältää oikeita tapahtumia verkkoliikenteestä. Tämän testin avulla todetaan, voiko työkalua käyttää toivotulla tavalla kohdeympäristössään.

Tällä testauksella voimme todeta, että työkalu löytää poikkeamat ja osaa määritellä niiden eroavaisuuden oikeissa tilanteissa ja on valmis testaukseen asiakasympäristössä. Materiaalina käytetään yhdenpäivän ajalta kerättyä lokitiedostoa.

Yhden päivän lokiin menee aikaa 12.3 sekuntia ja tuloksessa löytyy muutama kymmentä eri tapahtumaa, jotka luokittuvat jollain tasolla poikkeamiksi.

```
676.851432805962,gtid(6955520)
664.797799395936,trans(17191393)
663.954702107969,trans(10781267)
663.954702107969,trans(1953331)
663.954702107969,trans(2659314)
663.954702107969,trans(274689)
663.954702107969,trans(4114610)
663.954702107969,trans(5003424)
663.954702107969,trans(5736001)
663.954702107969,trans(628577)
660.313279533538,trans(3563971)
660.313279533538,trans(8151234)
```

Seuraavaksi tarkistetaan manuaalisesti poikkeaman tyyppejä.

```
could not establish SSL for incoming connection

HTTP/1.1 401 Unauthorized: Client id in wrong location.

could not establish SSL for incoming connection
```

Työkalu löytää lokitiedostosta poikkeamia ja kykenee muodostamaan toivottua poikkeama tietoa. Virheilmoitukset tulevat poikkeavuus määreen mukaisesti ja lasketulla kaavalla arvotetusti listattuna raporttiin. Projekti voidaan tässä vaiheessa esitellä tuloksineen toimeksiantajalle, josta voidaan lähteä jatkokehittämään toivottuun suuntaan.

7 Johtopäätelmät ja kehitysideat

Työkalun voidaan testien mukaan todistetusti tunnistavan poikkeamia verkkoliikenteestä. Aktiivisella testauksella huomattiin, että käytetyt algoritmit suoriutuvat prosessista toivotulla tavalla ja lokitiedoston poikkeavat tapahtumat saadaan käsiteltyä. Ja Sofware development life cyclen mukainen ohjelmistotuotanto prosessi toimi hyvin työntoteutukseen. Porrastettu luominen onnistui toivotusti. Määritellyt vaiheet auttoivat ymmärtämään tarvetta ja ympäristöä, jotka työkalulle oli määritelty.

Ikkunointi ja parsinta rajaavat muuntavat ja rajaavat lokin toivotusti ihmisen ymmärrettäväksi. Tämän avulla lokitiedostoa on analyysin jälkeen helpompi analysoida sekä tulkita.

TFIDF saadaan todennettua sanojen toistuvuutta. Algoritmi samalla arvottaa sanoja toivotusti ja tiputtaa turhia elementtejä pois, joita ei laskelmoi ihmiskieliseksi. Toiminnallisuuden avulla saatiin todennetusti luokitettua poikkeavuuksia toistuvuuksien perusteella.

PCA (Principal Component Analysis) ja poikkeamientunnistamisen avulla saatiin laskelmoitua rivien toistuvuutta lokista ja luokiteltua toivotusti poikkeavuudet TFIDF algoritmin jälkeen. Tämän osion avulla saadaan jatkossa osoitettua graafeilla poikkeavuuksia lokianalyysi käyttöliittymiin.

Myös työkalun lokitiedoston tutkimiseen käyttämä aika oli nopea, joka osoittaa että työkalun toiminallisuudet ovat rakennettu tarpeen mukaisiksi ja nopeiksi. R-kielen valinta mahdollisti, sen että jokainen vaihe on rakennettu tarkkaan askel kerrallaan.

Testivaiheen tulokset kertovat, että työkalu havaitsee poikkeavuudet lokitiedostoista valittujen menetelmien kautta. Kokonaisuudessaan poikkeaman tunnistus tuo lisäarvoa DevSecOpsin kannalta palvelun monitorointiin, eli tietoturvan valvontaan jatkuvia palveluita käyttävien osapuolien ja tuottavien hyödyksi.

DevSecOpsin periaatteisiin kuuluu, että palvelun käyttöä monitoroidaan ja havainnoidaan riskien havaitsemiseksi. Ja DevSecOpsin mukaan, palvelua tulee skannata jatkuvasti. Tämän haasteen ratkaisemiseksi työkalusta voidaan havaita olevan hyötyä osana turvallisen kehityksen elinkaarta. Tuotettaessa jatkuvaa palvelua, jossa julkaistaan muiden tekemiä tuotoksia, on tärkeää pystyä havaitsemaan poikkeavuuksia aikaisessa vaiheessa tuotteen elinkaarta.

Palveluiden monitoroiminen julkaisun aikana ja jälkeen, mahdollistaa riskien vähentämisen ja myös havaitsemisen haitallisten kokonaisuuksien suhteen. Lisäksi omavarainen työkalu toimii kustannustehokkaana tuotteena asiakastuote tarjonnassa.

8 Kehitysideat

Kehitystyön ja testaamisen jälkeen työkalu esiteltiin toimeksiantajalle esittelemällä toimintaa. Tilaisuudessa käytiin läpi käytetyt menetelmät, ja esiteltiin työkalua toiminnassa ajaen lokidataa scriptien läpi. Tällä saatiin esitettyä tuloksia, mitä työkalulla saadaan aikaiseksi. Toimeksiantaja totesi työkalun olevan hyödyllinen ja kiinnostava. Ja työkalua tullaan testauttamaan asiakkaan ympäristössä ja esittelemään tuloksia asiakkaalle.

Opinnäytetyön tekoprosessin aikana kehitetty työkalu mahdollistaa tarpeen mukaisen havainnoinnin tietoliikenteen lokeista. Ja työkalua ja ideaa voidaan kehittää haluttuihin suuntiin. Valmisversio havaitsee yksinkertaisuudessaan tehokkaasti tärkeitä huomioita tuovia osuuksia ja jatko vaihto ehtoja on muutamia.

- Työkalu voidaan asettaa omalle pilvi-instanssille louhimaan lokidataa ja sen ilmoitukset voidaan integroida osaksi analytiikkajärjestelmää.
- Työkalusta voidaan jalostaa monipuolisempaa analytiikkaa tekevä työkalu.
- PCA:ta voidaan käyttää laskemaan poikkeamien oletusarvo ja tämän tulos voidaan tallentaa. Uutta dataa voidaan laskelmoida TFIDF avulla, mutta nämä voidaan analysoida tallennettujen PCA tulosten avulla. Tämä on edullinen ratkaisu.
- Tai työkalu voidaan optimoida analysoimaan kaikkea liikkuvaa dataa.
- Työkalu voidaan kääntää Python kielelle, ja siitä voidaan tehdä osa avoimen lähdekoodin lokianalyysi järjestelmää.

Lähteet

Painetut

Adams, N & Heard, N. 2014. Data analysis for network cyber-security. London. Imperial College Press.

Niiniluoto, Ilkka. 2002. Tutkijan eettiset valinnat. Tampere. Gaudeamus

Fielding, Roy Thomas. 2000. Architectural Styles and the Design of Network-based Software Architectures. Kalifornian yliopisto, Irvine.

Chapman, N & Chapman J. 2012. Authentication and Authorization on the Web. MacAvon Media.

Sähköiset

Kay, Russel. 2002. How to System Development life cycle. Viitattu 22.05.2018 https://www.computerworld.com/article/2576450/app-development/app-development-system-development-life-cycle.html

GSA. Understanding the differences between agile & DevSecOps – from a business perspective. 2018. Tech at GSA.

Wanpeng, L & Mitchell, C. 2015. Analysing the Security of Google's implementation of OpenID Connect. University of London, Infomartion Security Group.

Tuovinen, Antti-Pekka. 2013. Testaus osana ohjelmistojen elinkaarta II. Helsinki. Helsingin Yliopisto. https://www.cs.helsinki.fi/u/aptuovin/testaus/Ohj_testaus_2013_4.pdf

OWASP Security Champion handbook. 2017. OWASP

OWASP. 2017. REST security cheat sheet. Viitattu 01.04.2018. https://www.owasp.org/index.php/REST_Security_Cheat_Sheet

Deagun, D, Johnsson Dan & Sawano, Daniel. 2018. Secure by Desing. Manning Publications

Security audit report. 2017. Nixu

The security development life cycle. 2015. Viitattu 20.5.2018 https://social.technet.microsoft.com/wiki/contents/articles/7100.the-security-development-lifecycle.aspx

P'unk Avenue. 2018. sanitize-html. Viitattu 05.04.2018. https://github.com/punkave/sanitize-html

Breitinger, T,, Gipp, B., Langer, S. 2015. Research-paper recommender systems: a literature survey. 17. painos. International Journal on Digital Libraries.

Ding, C., He, X. 2004. Proceeding of the Twenty-first international conference on machine learning. ACM.

Kaner, C. 2006. Exploratory Testing. Orlando. Florida Institute of Technology.

Exploratory Testing, Cem Kaner, Florida Institute of Technology, Quality Assurance Institute Worldwide Annual Software Testing Conference, Orlando, FL, November 2006

Julkaisemattomat

Niinioja, M. 2018. Henkilökohtainen tiedoksianto

Kuviot

Liite 1: Ensimmäinen liite