Sami Frisk

API-RAJAPINTOJEN HALLINTA JA TIETOTURVA

JYVÄSKYLÄN YLIOPISTO
INFORMAATIOTEKNOLOGIAN TIEDEKUNTA
2019

TIIVISTELMÄ

Frisk, Sami
API-RAJAPINTOJEN HALLINTA JA TIETOTURVA
Jyväskylä: Jyväskylän yliopisto, 2019, 28 s.
Tietojärjestelmätiede, kandidaatin tutkielma
Ohjaaja(t): Palonen, Teija
Asiasanat: API, API-rajapintojen hallinta, REST, API haavoittuvuudet, API tietoturva

Nykyaikaiset web-pohjaiset rajapinnat ovat käytössä digitaalisessa maailmassa kaikkialla. Rajapintoja on niin pilvipalveluiden takana, esineiden internetissä, mobiilisovelluksissa, kuin järjestelmien välisissä integraatioissa. Rajapintojen hallinta ja niiden tietoturva ovat nousseet esille viime vuosina julkisessa keskustelussa tietoturvahyökkäysten myötä. Organisaatioissa on myös havaittu ongelmia sen suhteen, miten API-rajapintoja voidaan hallita niiden elinkaaren eri vaiheissa. Tutkielmassa tutustuttiin web pohjaisten rajapintojen taustoihin ja tämän hetkiseen yleisimmin käytettyyn arkkitehtuurimalliin, joka on REST. API-rajapintojen hallinnan eri haasteita esiteltiin yleisellä tasolla ja kuvattiin hallintaohjelmistojen eri ominaisuuksia. Tietoturvan osalta tunnistettiin yleisimpiä haavoittuvuustyyppejä, jotka teknisessä mielessä ovat hyvin yhteneväisiä web-sovellusten haavoittuvuuksien kanssa. Teknisen näkökulman lisäksi API-rajapintojen haavoittuvuuksiin voidaan lukea inhimillisemmät seikat, kuten turvallisuudentunne tai huolimattomuus. Ratkaisut rajapintojen haavoittuvuuksilta suojautumiseksi ovat teknisessä mielessä samoja web sovellusten suojautumisten kanssa. Tutkielmassa esiteltiin myös API-tietoturvakerrokseen liittyvä arkkitehtuuriehdotus, joka koostuu kolmesta eri API-tietoturvatasosta malliratkaisuineen.

ABSTRACT

Frisk, Sami
API MANAGEMENT AND SECURITY
Jyväskylä: University of Jyväskylä, 2019, 28 pp.
Information Systems, Bachelors's Thesis)
Supervisor(s): Palonen, Teija
Keywords: API, API management, API vulnerabilities, API security

API's are used everywhere in our digital world. API's are used in cloud services, internet of things, mobile apps, enterprise application integrations and so forth. API management and API security are topics which are popular in public discussions after successful exploitation of API vulnerabilities. Organizations have also realized challenges on how API's can be managed throughout the whole lifecycle. Web service history and background was introduced and currently dominant architectural model, REST introduced. Challenges regarding API management were identified, also study describes how common API management software functionalities can resolve these API management challenges. API security has a lot of common with web applications vulnerabilities. In addition to technical point of view, there are a lot of other human vulnerabilities like false sense of security or negligence. Technical solutions for securing API's are mostly similar what comes to web applications. Also architectural model for API security layer, with three different protection levels, was described in this study

KUVIOT

TAULUKOT

SISÄLLYS

1 JOHDANTO

Sovellusten ohjelmoitavat rajapinnat (application programming interface) eli API-rajapinnat ovat olleet osa ohjelmisto- ja integraatioarkkitehtuureja yhtä kauan kuin tietojärjestelmiä on ollut olemassa. Hajautettujen järjestelmien ja varsinkin Webin myötä rajapintojen merkittävyys alkoi korostua (Lane, 2012). Rajapintojen käyttö on kasvanut viimeisten vuosikymmenten aikana vuosi vuodelta. Rajapinnat ovat erottamaton osa Webiä, pilvipalveluita ja nykyaikaista ohjelmistokehitystä. (Burns, 2012).

API-rajapinnat eivät ole kuitenkaan pelkästään asia, joka tulisi huomioida ohjelmistokehityksen tai järjestelmäarkkitehtuurin näkökulmasta. API-rajapinnat ovat viime vuosina noussut taas pinnalle alustalouden ja API-rajapintojen ekosysteemiajattelun myötä. API-rajapinnat mahdollistavat lukuisten eri käyttötapausten ja sovellusten välisen kommunikaation toteuttamisen esimerkiksi pilviohjelmistoissa, esineiden internetissä, mobiilisovelluksissa ja niin edelleen. API-rajapinnoista on tulossa yrityksille kilpailutekijä, jolla pysyä mukana kilpailussa ja luoda uutta liiketoimintaa (Moilanen, 2018).

L. Tangin (2013) mukaan käynnissä on API-rajapintojen aikakausi, jossa yritykset tarjoavat omia varantojaan ja palveluitaan muille saavuttaakseen oman paikkansa API-ekosysteemissä. Hyvänä esimerkkinä tästä on useat pilvipohjaiset palvelut, joiden toiminta on vahvasti API-pohjaista. Esimerkiksi vuonna 2016 Salesforcen, pilvipalveluna toimivan CRM-järjestelmän transaktioista 60 % kulkee API-rajapintojen kautta ja liikevaihto, joka kulkee API-rajapintoja hyödyntäen on 5 miljardia dollaria. (Vukovic, 2016.)

Tässä tutkielmassa tarkastellaan web-pohjaisia REST-arkkitehtuuriin pohjautuvia API-rajapintoja. Synonyymina näille rajapinnoille käytän niistä nimitystä API-rajapinta. Proffittin (2013) mukaan API-rajapinnat ovat kokoelma vaatimuksia, jotka määrittelevät miten sovellus kommunikoi muiden sovellusten kanssa. Samaan tapaan Jakobsson (2012) määrittelee API-rajapintojen käytön sopimukseksi siitä, miten kaksi sovellusta kommunikoivat keskenään: dokumentoidun sopimuksen myötä kommunikaatio APIn tarjoajan ja APIn hyödyntäjän välillä on tehokasta, koska säännöt siitä, miten interaktio tapahtuu, ovat selkeät.

API-rajapinnat tuovat esille sovelluksen sisäisiä funktioita tai dataa määritellyn ja selkeän rajapinnan kautta. Tämä mahdollistaa API-rajapintojen tehokkaan käytön ulkopuolisten toimijoiden osalta ilman, että heidän tulisi ymmärtää, miten sovellus itsessään toimii. API-rajapintojen suorina käyttäjinä eivät ole ihmiset, vaan rajapintojen käyttäjinä voidaan pitää ohjelmistoja ja järjestelmiä (machine to machine). (Fremantle ym., 2015.) Esimerkiksi mobiilisovellus voi hyödyntää puhelimen käyttöjärjestelmän tarjoamaa rajapintaa kuvien ottamiseen ilman, että sovelluksen kehittäjän tarvitsee ottaa kantaa siihen, miten kuvan ottaminen tapahtuu käyttöjärjestelmän tai laitteiston osalta.

Pelkästään julkisia rajapintoja on kymmeniä tuhansia tarjolla erilaisten API-katalogien kautta. Tietoturvahaavoittuvuuksien näkökulmasta rajapinnat ja niiden hallinta kattavat monia eri kerroksia ja komponentteja aina API rajapinta-alustojen käyttöjärjestelmätasolta rajapintojen pääsynhallintaratkaisuihin. Huolimatta siitä, että API-rajapinnat ovat kaikkialla käytössä niiden tietoturvaan ei ole kiinnitetty riittävästi huomiota. (Macy, 2018.)

API-rajapintojen suunnittelun oletuksena on, että ne ovat julkisia tai puolijulkisia – nämä rajapinnat eivät siis ole pelkästään sisäisten verkkojen tai VPN-yhteyksien takaa saavutettavissa (Fremantle ym., 2015). Muun muassa API-rajapintojen dokumentaatio ja tietoturvallinen julkaiseminen, julkaisualustojen kapasiteetin hallinta, käyttäjien hallinnointi ja hallinta sekä käytön analytiikka ovat ongelmakohtia organisaatioiden API-käytössä (Alagarasan, 2015). Näitä haasteita API-rajapintojen hallinta (API Management) ja hallintaa varten rakennetut ohjelmistokokonaisuudet omalta osaltaan ratkovat (Fremantle ym., 2015).

Tässä tutkielmassa etsitään vastauksia seuraaviin tutkimuskysymyksiin:

- Mitä tarkoitetaan rajapinnoilla, niiden kategorisoinnilla ja rajapintojen hallinnalla?
- Mitä tarkoitetaan rajapintojen tietoturvalla ja miten suojautua tietoturvaa uhkaavilta haavoittuvuuksilta?

Tutkielma on toteutettu kirjallisuuskatsauksena. Aineiston keräämiseen on käytetty IEEE Xplore Digital Library ja Google Scholar -hakutietokantoja. Aihealueesta johtuen lähteinä on käytetty myös muita julkaisuja ja ohjelmistotoimittajien luomia ”white paper”-julkaisuja, Mahdollisuuksien mukaan näiden julkaisujen valinnoissa on painotettu lähteitä, joihin on viitattu tieteellisissä artikkeleissa.

Tiedonkeruussa käytin muun muassa seuraavia hakusanoja: API Management, Web API, API security, API lifecycle ja API vulnerabilities. Kirjallisuudessa termit eivät ole yhteneväisiä, esimerkiksi Web API ja API voivat tarkoittaa samaa asiaa riippuen siitä, onko lähde uudempi vai vanhempi. Uusimmissa lähteissä pelkkä API näyttäisi olevan termi, joka on yleistymässä käyttöön tässä yhteydessä. Ohjelmistotoimittajien ja kaupallisten tutkimus- ja konsultointiyrityksen terminologiassa käytetään poikkeuksetta termiä API. Priorisoin mahdollisimman uusia artikkeleja lähteiden valinnassa.

Luvussa 2 käydään läpi rajapintojen määritelmiä sekä kerrotaan, mitä on rajapintojen hallinta, mistä osa-alueista rajapintojen hallinta koostuu ja mitä yleisimpiä hallintaohjelmistokokonaisuuksia on markkinoilla saatavilla. Kolmannessa luvussa tutkitaan rajapintojen tietoturvaa ja minkälaista haasteita ja uhkia rajapintoja kohtaan kohdistuu. Lisäksi käydään lävitse keinoja, mitä keinoja rajapintojen hallinta tarjoaa tietoturvauhkien suojautumiseksi. Lopuksi luvussa 4 vedetään yhteen vastaukset tutkimuskysymyksiin ja pohditaan mahdollisia lisätutkimusaiheita.

2 API- RAJAPINNAT

Tässä luvussa aluksi selvitetään web-palveluiden historiaa, josta edetään modernien API-rajapintojen määritelmään, toteutustapaan sekä kategorisointiin. Luvussa esitellään yleiskuva API-rajapintojen käytöstä sekä haasteista, joita käyttöön liittyy niin julkaisijan kuin käyttäjän näkökulmasta. Lopuksi esitellään API-rajapintojen hallintaa ja hallintaan tarkoitettuja ohjelmistoja.

2.1 Taustaa

Web suunniteltiin alun perin asiakas-palvelin-pohjaiseksi järjestelmäksi, jota ihmiset käyttävät hypertekstidokumenttipohjaisena (Berners-Lee, 1989). Interaktiivisuus webissä tarkoitti siis, että käyttäjät seurasivat linkkejä ja lukivat sivujen sisältöjä ja dokumentteja. Lomakkeiden mahdollistama tietojen syöttäminen lisättiin myöhemmin HTML-kieleen ja http-protokollaa laajennettiin, jotta tietoja voitiin myös lähettää niiden pyytämisen lisäksi. Samassa yhteydessä W3C-organisaatio standardoi XML-kielen, joka lisäsi kiinnostusta Web-teknologioiden käyttöä kohtaan hajautetuissa järjestelmissä. (Kopecky, 2014.)

2000-luvun alussa SOA (Service-oriented architecture) esiteltiin uudenlaisena arkkitehtuurityylinä, jossa ohjelmistokomponentit ovat pilkottu omiksi itsenäisiksi ja standardoiduiksi palveluiksi, joita voidaan käyttää standardien Webissä käytettävien protokollien välityksellä. Tällä pyrittiin luomaan tietojärjestelmien välille mahdollisimman joustava ja järjestelmäriippumaton vuorovaikutus (Tan, 2016).

SOAP (Simple Object Access Protocol) ja XML-RPC teknologioiden määrittelyn ja julkaisun myötä sai alkunsa liikehdintä, jota johti isot ohjelmistotoimittajat kuten IBM, Microsoft, Sun, Oracle ja BEA. Kyseisen liikehdinnän tulosta standardoinnin ja teknologisen kehitysloikan saavuttamiseksi kutsuttiin nimellä Web-palvelut (Web Services). Tavoitteena oli rakentaa standardi lähestymistapa isojen järjestelmien kytkemiseksi toisiinsa. (Kopecky, 2014.) SOAP-protokollasta tuli siten yksi merkittävä tekninen toteutustapa SOA-arkkitehtuurin mukaiseen palveluiden toteuttamiseen.

SOAP-pohjaisten web-palveluiden kuvauskieli WSDL (Web Services Description) oli yksi tärkeimmistä tekijöistä Web-palveluiden suosioon nousussa.

WSDL mahdollisti helpon tavan ottaa olemassa oleva objekti ja määritellä siitä Web-palvelu. Sama käyttöönoton helppous oli myös asiakkaan määrityksien osalla. (Kopecky, 2014.)

Yksi suurimmista haasteista Web Service-standardien kehityksessä oli isojen ohjelmistotoimittajien väliset poliittiset ristiriidat. Tämä johti tilanteeseen, jossa syntyi useita kilpailevia määritelmiä Web palveluiden eri osa-alueille. Esimerkiksi WSDL:n 1.1-versiossa on kaksi erilaista tapaa tehdä sama asia johtuen toimittajien erilaisista toteuttamisnäkemyksistä. Kiistat ja niistä juontuvat eri tavat tekivät standardista kompleksisemman ja vähemmän yhteensopivan. SOAP-toteutukset eivät myöskään tukeneet modernien mobiilisovellusten käyttämää JSON-formaattia. Kolmas tärkeä tekijä SOAP-toteutusten suosion laskuun on ollut sen kompleksisuus verrattuna http-pohjaisten REST-kutsujen yksinkertaisuuteen. (Kopecky, 2014.)

2.2 REST

SOAP-standardin kilpailija on ollut vuonna 2000 esitelty REST (Representational state transfer)-arkkitehtuurityylin mukainen toteutus rajapinnoille (Fielding, 2000). Kyseessä ei ole siis standardi kuten SOAP on. REST-toteutukset rakentuvat yleensä HTTP-protokollan päälle. Huomattavaa on, että tämän arkkitehtuurityylin kehittäjä on myös ollut määrittämässä http-protokollaa. Yksinkertaistettuna RESTin toiminta perustuu resurssien ja resurssien operointiin http-protokollasta löytyvien metodien avulla. Resurssi ja resurssityypit ovat osa URL-osoitetta. Kaiken kaikkiaan yksittäisen resurssin nimiavaruus koostuu protokollasta, isännän määrityksestä (host), sovelluspolusta, resurssityypistä ja resurssin tunnisteesta. Operaatiot, jotka koskevat resursseja eikä yksittäistä resurssia, kutsuvat osoitetta ilman resurssin tunnistetta. (Battle, 2008.)

REST-tyylisen suunnittelun ytimessä on kokoelma tilan muutosoperaatioista, jotka ovat mille tahansa datan tallentamiseen ja hakemiseen perustuvalle järjestelmälle ominaisia tehtäviä. Käytetyimpiä http-protokollan metodeja resurssien manipulointiin ja hakemiseen ovat GET, POST, PUT ja DELETE (Garcia, 2017) jotka vertautuvat hyvin perinteisiin tietokannan CRUD-operaatioihin (Create, Read, Update ja Delete) taulukon 1 mukaisesti:

TAULUKKO 1 HTTP ja REST-kutsujen vertaus tietokantaoperaatioihin (Battle, 2008)

Tietokantaoperaatio	**http-metodi**	**Syötettävä sisältö (input format)**	**Vastaus (response)**
Create	POST	(HTTP Form Encoded)	Status 201 CREATED
Read	GET	-	Pyynnön otsikkotiedoissa.
Update	PUT	(HTTP Form Encoded)	Status 200 OK
Delete	DELETE	-	Status 200 OK

Käytännön esimerkkinä seuraavanlaisella pyynnöllä voitaisiin kutsua rajapintaa, joka palauttaisi listauksen kaikista tuotteista, joita rajapinnan kautta voi kysyä:

”http get http://api.organisaatio.com/tuotteet”

Vastaavasti poisto-operaatio yksittäiselle resurssille tapahtuisi hyödyntäen DELETE-komentoa samalla resurssille tarvittavan identifioivan tunnisteen lisäämisellä kutsuun:

”http delete http://api.organisaatio.com/tuotteet?id=100 ”

Kumari (2015) listaa RESTin eduiksi seuraavat ominaisuudet:

- Tilattomuus.
- Helppokäyttöinen ja nopea omaksua.
- Käyttää http-protokollan metodeja.
- Tarvittava kaistanleveys viestien siirrossa on pieni, pääasiassa johtuen oletuksena käytetyn JSON-formaatin keveydestä.
- Tietoturvanäkökulmasta liikennöinti tapahtuu standardoitua http-protokollaa käyttäen.
- Asiakassovellukset, jotka hyödyntävät rajapintaa voivat olla hyvinkin heterogeenisiä toteutukseltaan.
- Joustavuus. REST-toteutuksissa voidaan käyttää muitakin tietoformaatteja kuin JSON-formaattia.

Kumarin (2015) mukaan REST-toteutusten haasteiksi määritellään:

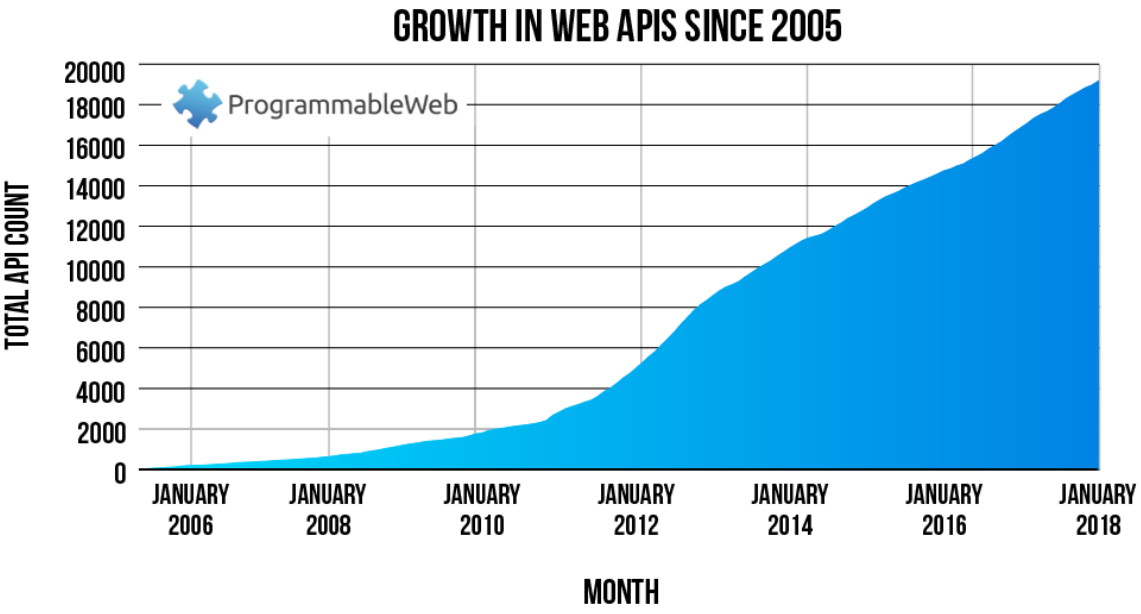
- Ei sovellu isojen tietomäärien käsittelyyn.
- SOAP-protokollaan verrattuna ei yhtä kattavia ominaisuuksia tietoturvan tai operaatioiden käsittelyille.
- Luotettavuus.

Tänä päivänä web-pohjaisissa sovelluksissa käytettävät rajapinnat voivat olla eri teknologioilla tai standardeilla toteutettuja, mutta REST-arkkitehtuuriin perustuvat toteutukset ovat suosituimpia kuten voidaan todeta taulukosta 2. Haasteistaan huolimatta SOAP-pohjaiset toteutukset ovat olleet aiemmin laajasti käytössä organisaatioissa, mutta jo vuonna 2010 REST-arkkitehtuuriin perustuvat toteutukset olivat huomattavasti suositumpia kuin SOAP-standardin mukaiset. (Lensmar, 2013.) Myös Tanin (2016) mukaan REST-toteutukset ovat oletusarvoinen tapa toteuttaa rajapintoja Webissä, mobiilisovelluksissa, pilvi-infrastruktuurissa ja esineiden internetissä. Isoista ohjelmistotalousta esimerkiksi vuonna 2009 Google hylkäsi hakurajapinnoissaan SOAP-toteutusten käytön ja alkoi toteuttaa vain REST-pohjaisia rajapintoja (Tan, 2016). Tässä tutkielmassa API-rajapinnalla tarkoitetaan siis http-protokollaa hyödyntävää REST-toteutusta, ellei toisin mainita.

TAULUKKO 2 Programmableweb-kirjastossa julkaistujen julkisten rajapintojen tekninen toteutustapa (Santos, 2017)

Toteutustapa	**Lukumäärä**
REST	14903
RPC / SOAP	1711
Null	1259
Native/Browser	170
Push/Streaming	152
Indirect	76
GraphQL	8

API-rajapinnat ovat siis kaikkialla käytössä: mobiilisovelluksissa, esineiden internetissä, pilvipalvelutoteutuksissa, web-sovelluksissa, osana sovellusten välisiä integraatioita, pääasiallisena kanavana transaktioiden toteuttamiseen liiketoimintasovelluksissa ja niin edelleen. Standardoiduista tai vähemmän standardoiduista rajapinnoista on tullut liima, joka pitää digitaalista maailmaa koossa. (Macy, 2018.) Julkisten ja vapaasti saatavilla olevien rajapintojen määrä on vuosi vuodelta noussut tähän päivään asti kuten voidaan kuvasta 1 todeta.



Kuva 1 Julkisten API-rajapintatoteutusten määrä Programmableweb-katalogissa (Wendell, 2018).

2.3 Kategorisointi

API-rajapintojen kategorisointiin löytyy useita eri tapoja ja mielipiteitä (Brodsky ja Oakes 2017; De 2017; Jacobson ym., 2011; Morgan ym., 2016; Moilanen ym., 2018). Moni lähteistä jakaa toteutukset kahteen osaan, jota tässäkin tutkielmassa käytetään: julkisiin ja yksityisiin. Yksityiset API-rajapinnat taasen voidaan edelleen jaotella sisäisiin ja kumppanirajapintoihin. Taulukossa 3 on koottu yhteen eri kategorioiden ominaisuuksia.

Sisäiset API-rajapinnat ovat nimensä mukaisesti rajoitettu organisaation sisäiseen käyttöön. Sisäinen API voi olla käytössä osana sisäisesti käytettyä tuotetta, kuten yksittäinen liiketoimintasovellus tai HR-järjestelmä. Rajapintojen kautta tehdyt transaktiot tai saatu data on yleensä vain sisäiseen käyttöön. Oletuksena sisäinen rajapinta on saavutettavissa vain organisaation sisäisen verkon kautta. Tämän tyyppinen API voi myös olla teknisesti saavutettavissa julkisen internet-yhteyden kautta esimerkiksi sovelluksen mobiilikäytön mahdollistamiseksi, mutta tällöin rajapinnan käyttö on rajattu vain organisaation käyttäjille. APIa ei voi rekisteröidä tai ottaa käyttöön ulkopuolisten toimesta. (Moilanen, 2018.)

Kumppaneille (Partner) tarkoitetut API-rajapinnat ovat yleensä räätälöityjä toteutuksia, joiden kohdeyleisönä on tietyt valitut kumppanit, asiakkaat tai muut sidosryhmät. Toteutus on myös yleensä hyvin rajattu tiettyyn yksittäiseen liiketoimintaprosessiin tai transaktioon. Kumppani-APIn julkaisija ja hyödyntäjä ovat yleensä jo ennestään keskenään yhteistyössä ja rajapinnan julkaisu ja hyö-

dyntäminen kummankin osapuolen välillä voidaan nähdä sitoutumista ja yhteistyötä lisäävänä tekijänä. Rajapinta voi olla maksuton lisäpalvelu, kuten johonkin transaktioon tehokkuutta tuova ilmainen toteutus, joka hyödyttää molempia osapuolia, tai rajapinnan käyttö voi olla maksullista kumppanille. Kumppaneiden käyttöön tarkoitettu rajapinta on suojattu ulkopuoliselta käytöltä ja se vaatii rekisteröinnin ja käyttäjän tunnistautumisen. (Moilanen, 2018; Morgan ym., 2016.)

Julkisten API-rajapintojen käyttäjäkuntaa ei ole oletuksena rajoitettu ja ne ovat kaikkien saatavilla. Yleensä API-rajapinnan hyödyntäjän ei tarvitse rekisteröityä saadakseen rajapinnan käyttöönsä eikä minkäänlaista tunnistautumista tarvita. Julkisilla rajapinnoilla voidaan saavuttaa lisää näkyvyyttä omille tuotteille, uusia markkinoita ja sovellutuksia ulkopuolisten toimijoiden hyödyntäessä rajapintoja. Julkisten rajapintojen rooli ei ole niin suuri kuin voisi luulla, jos asiaa ajatellaan taloudellisesta tai määrällisestä näkökulmasta. Yksityiset API-rajapinnat ovat varsinainen ajava voima API-taloudessa. Monet julkiset API-rajapinnat ovat alkujaan olleet sisäisiä toteutuksia, jotka on elinkaaren myöhemmässä vaiheessa vasta avattu julkisiksi. Julkinen API voi olla ilmainen palvelu, tai tietyn käyttövolyymin ylittyessä käytöstä voidaan ryhtyä veloittamaan maksua. Julkiset organisaatiot ovat avanneet viime vuosina lukuisia täysin ilmaisia palveluita kuten esimerkiksi Suomessa Ilmatieteenlaitos, joka tarjoaa tuottamaansa tietoa vapaasti rajapintojen kautta. (Moilanen, 2018; De, 2017.)

TAULUKKO 3 API-rajapintakategorioiden ominaisuuksia (mukaillen Osaango, 2018)

	Rekisteröitymi-nen avointa	Maksullisuus	Käyttäjän tun-nistaminen
Julkinen API	Kyllä	Mahdollista	Mahdollista
Kumppani API	Ei	Mahdollista	Kyllä
Sisäinen API	Ei	Ei	Kyllä / Ei

2.4 Haasteet

Alagarasan (2015) nostaa esille kahdeksan eri aihealuetta, joissa rajapintoja julkaisevat tai hyödyntävät organisaatiot ovat havainneet ongelmia API-rajapintojen käytön osalta:

1. Sopimusten joustamattomuus, jossa kaikkien rajapintoja hyödyntävien tulee käyttää samaa sopimusta.
2. Tietoturvamekanismien joustamattomuus. API-rajapinta pitäisi olla käytettävissä monille käyttäjäryhmille erilaisilla teknisillä tunnistautumistavoilla.

3. Datamuotojen joustamattomuus tulee esille tapauksissa, joissa yksittäinen rajapinta käyttää vain yhtä formaattia datalle. Tämä voi aiheuttaa ongelmia rajapintojen hyödyntämisessä muissa järjestelmissä tai sovelluksissa, jotka eivät välttämättä tue julkaisijan valittua dataformaattia.
4. Dokumentaation puutteet. Jotta API-rajapinnan käyttöönotto ja ylläpito olisi tehokasta ja laadukasta, tulee API-rajapinnan julkaisijan tarjota riittävän kattava ja ajantasainen dokumentaatio.
5. Kapasiteetti. API-rajapintojen käytölle voi tulla yksittäisiä kuormitushuippuja yllättäen. Järjestelmäkapasiteetin kohdentaminen hyödyntäjien tarpeiden mukaisesti on haasteellista, jos taustalla oleva alustapalvelu ei ole skaalautuva.
6. Käyttäjien eli hyödyntäjien hallinta. Organisaatioilla ei ole saatavilla tietoa siitä, mitä rajapintoja kukin hyödyntäjä käyttää ja mitkä ovat käytön volyymit. ”Ovatko kaikki käyttäjät tunnettuja?” ja ”miten hallinnoidaan tuntemattomat käyttäjät?” ja niin edelleen ovat kysymyksiä, joihin tulisi saada vastauksia vaivattomasti.
7. Analytiikka ja käytön hallinta. Kuinka tunnistaa ja estää käyttö joka on ongelmia aiheuttava julkaisijalle. Esimerkiksi palvelunestohyökkäykset tai muu radikaali rajapintojen kuormitus tahallisesti tai tahattomasti.
8. Versionhallinta rajapinnan elinkaaren ajan. API-rajapinnan julkaisijan päivitys rajapintaan voi pakottaa hyödyntäjän päivittämään omaa toteutustaan, ellei API-rajapinnan tarjoaja tuo saataville eri versioita rajapinnasta.

Yksittäisen APIn laadun näkökulmasta löytyy myös ongelmia. Rajapinnoista moni on huonosti suunniteltu ja toteutettu. On helppoa nykyisillä työkaluilla ja ohjelmointipakeilla luoda REST-pohjaisia API-rajapintoja, mutta paljon vaikeampaa on luoda hyvä toteutus. Pienistä virheistä ja puutteista muodostuu suurempia ongelmia sitä mukaa, mitä enemmän API-rajapintaa käytetään ja mitä useampi eri taho sitä hyödyntää. (Mosqueira-Reya, 2018.)

REST-pohjaisten API-rajapintojen kuvauskieleksi on vasta viime vuosina muotoutunut yleisesti käytetty standardi. Tämän hetkinen nimitys standardille on ”OpenAPI Specification”, jota aiemmin on kutsuttu nimellä ”Swagger”. Samalla tavalla kuin 2000-luvun alkupuolella isot ohjelmistotalot olivat web-palveluiden takana kehittämässä tarvittavia standardeja, niin IBM, Microsoft ja muut ovat OpenAPI-kuvauskielen tukijoina. Kuvauskielen puute tai kuvauskielien kirjo on ollut yksi REST pohjaisten rajapintojen yleistymisen ja laadullisen kehittymisen este. API-rajapinnan metadatan määrittely kuvauskielen avulla saa niin ihmiset kuin sovellukset löytämään ja ymmärtämään tehokkaammin julkaistujen rajapintojen toiminnallisuuksia ilman, että joudutaan käymään lävitse lähdekoodia tai dokumentaatioita.

2.5 Hallinta ja hallintaohjelmistot

Kopecky ym. (2015) mukaan akateemisessa tutkimuksessa ei ole määritelty yksiselitteisesti ja universaalisti sitä, mitä rajapintojen hallinta pitää sisällään ja sitä kautta mitä hyötyjä rajapintojen hallinnalla tavoitellaan. Kopecky edelleen esittää, että rajapintojen hallinta (Web API Management) pitää sisällään ainakin seuraavat osa-alueet:

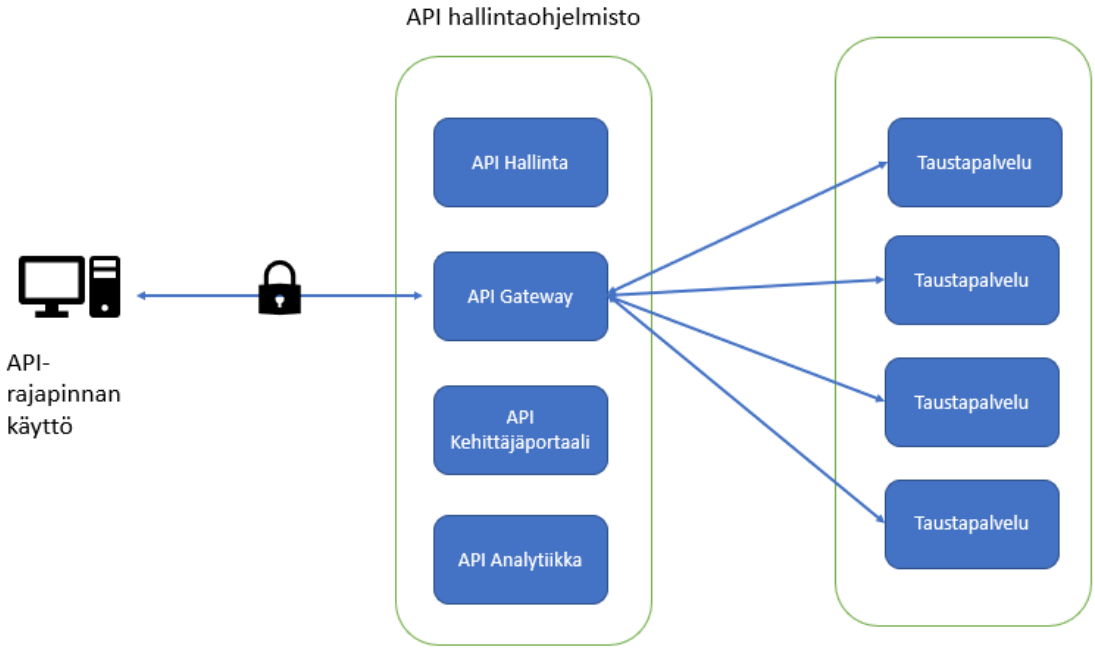
- Kehittäjäportaali, jonka kautta kehittäjät ja muut API-rajapintojen hyödyntäjät voivat tilata, testata ja ottaa käyttöön rajapintoja.
- API-rajapintojen julkaisu ja rajapintoihin liittyvän materiaaliin tarvittava hallinta, joka pitää sisällään ainakin rajapintojen dokumentaation, ohjelmistokehityspakit ja muut ihmisten tai koneiden hyödyntämät ohjeistus- ja tukimateriaalit.
- API-rajapintoja käyttävien asiakassovellusten autentikoinnin ja pääsynhallinnan toteuttaminen esimerkiksi API-avaimia (API keys) ja tunnisteita (security token) jakamalla.
- Käytönhallinta ja mahdollisuus rajoittaa asiakasohjelmien käyttöä pohjautuen SLA-sopimuksiin tai muihin vaikuttaviin tekijöihin
- Käyttöanalytiikka, jonka perusteella voidaan muun muassa toteuttaa edellä mainittua API-rajapintojen käytönhallintaa tai analytiikan perusteella voidaan muodostaa rajapintojen käytön laskutustietoja.

Yksi tärkeimmistä rajapintojen hallinnan tavoitteista on ottaa huomioon edellä listattuja asioita jo suoraan rajapintojen suunnittelusta ja toteutuksesta lähtien aina rajapintojen elinkaaren loppuun asti. Tämä tuo kaivattua standardinomaisuutta rajapintojen kehittämiseen ja päästään eroon sovelluskohtaisista räätälöinneistä rajapinnoille. (Kopecky, 2015.) API-rajapintojen hallinta on prosessi, joka pitää sisällään rajapintojen suunnittelun, julkaisun, dokumentoinnin ja analytiikan tietoturvallisessa ympäristössä. Hallintaohjelmistojen käyttö takaa organisaatiolle sisäisten ja julkisten API-rajapintojen hallittavuuden ja tietoturvan (Mulesoft, 2018).

Suuret ja pienemmätkin ohjelmistotalot ja pilvipalveluiden tarjoajat ovat julkistaneet omia hallintaohjelmistopakettejaan API-rajapintojen hallintaan. Muutamia yleisimpiä ja tunnetuimpia tuotteita ovat esimerkiksi Azure API Management, Apigee API Management, Mulesoft Anypoint, WSO2 API Manager, CA API Management, IBM API Connect. Tuotekuvausten perusteella API hallintaohjelmistot toteuttavat lähes poikkeuksetta kaikki luvussa 2.6 mainitut osa-alueet. Toteutustavat ja arkkitehtuurivalinnat vaihtelevat eri ratkaisujen välillä. Myös ratkaisuissa käytetty terminologia ei ole aihealuetta kuvaavasti kovinkaan yhtenäistä, vaan eri tahojen käyttämät nimitykset hallintaohjelmistojen kyvykkyyksistä eroavat jonkin verran toisistaan. (Alagarasan,2015.)

Konseptuaalisella tasolla kuvattuna hallintaohjelmistot koostuvat neljästä eri kyvykkyydestä, jotka myös ovat kuvattuna kuvassa 2:

- API julkaisukanava (Gateway)
- API kehittäjäportaali (Developer Portal)
- API analytiikka (Analytics Portal)
- API hallinta- ja palvelunkehityskerros (Service Development UI)



Kuva 2 API-rajapintojen hallintaohjelmiston koostuminen (mukaillen Alagarasan ,2015)

API Gateway on tekninen julkaisukanava, jonka kautta varsinainen API-rajapinnan toteutus tarjotaan teknisellä tasolla käytettäväksi hyödyntäjille tietoturvallisella tavalla. API Gateway eristää taustalla toimivat järjestelmät taakseen, jolloin voidaan tarjota taustajärjestelmien dataa tai transaktioita paljastamatta suoraan taustalla toimivan järjestelmän toimintaa tai palveluita. (Apigee, 2015.) Mulesoft (2018) kuvaa oman tuotteensa ratkaisussaan tätä roolia kirjaimellisesti portinvartijan (gatekeeper) tehtäväksi. Tässä kerroksessa voidaan kuvata säännöstöillä, miten käyttöä rajoitetaan tai miten käyttäjät tunnistetaan ja millä oikeuksilla käyttäjä operoi rajapintaa. Suurin osa ratkaisuista tarjoaa myös ominaisuuksia datan transformaatioihin tai uudelleenreitityksiin sekä ratkaisuja myös suorituskyvyn näkökulmasta, kuten esimerkiksi välimuistin käytön. (Alagarasan,2015.)

API kehittäjäportaali toimii oletuksena itsepalveluperiaatteella. Kaikki organisaation jaeltavat toteutukset ovat julkaistuna kehittäjäportaaliin. Kehittäjät voivat rekisteröityä portaalin kautta haluamiensa API-rajapintojen käyttäjiksi. Kehittäjäportaalin kautta kehittäjille on tarjolla dokumentaatiota, jossa hallintaohjelmistot tukevat OpenAPI-standardin mukaista kuvauskieltä. API-avainten ja tunnisteiden jakelu voidaan toteuttaa myös tämän portaalin kautta. (Alagarasan,2015.)

API analytiikka on kytköksissä API gateway-komponenttiin. Analytiikka kerää dataa rajapintojen sisäänpäin ja ulospäin tapahtuvat liikenteestä. Analytiikan avulla saadaan läpinäkyvyyttä siihen, miten käyttäjät hyödyntävät API-rajapintoja valmiiksi muodostettujen raporttipohjien avulla. Analytiikan avulla voidaan havaita poikkeamia käytössä tai sen avulla voidaan kohdentaa kapasiteettia tai resursseja sinne missä niitä tarvitaan. (Alagarasan,2015). Analytiikka tarjoaa myös työkalut, joiden avulla toteuttaa laskutusta sen mukaan, miten rajapintaa on käytetty.

API hallinnassa eli palvelunkehityskerroksessa on työkaluja, jossa varsinainen API-rajapintojen kehitys tapahtuu. Kehittämisen yhteydessä dokumentaation luonti tapahtuu hallintatyökalun kautta hyödyntäen yleensä OpenAPI-standardia. Hallintaohjelmistot tarjoavat valmiita säännöstöjä ja toiminnallisuuksia, jotka voidaan konfiguroiden ottaa käyttöön rajapintojen toteutuksessa (Mulesoft, 2018). Kehittäjät voivat tarpeen mukaan myös lisätä omia säännöstöjään toteutukseen. Kehitystyökalussa myös määritellään ja toteutetaan API-rajapintojen julkaiseminen, käytöstä poisto tai mahdollinen migraatio. (Alagarasan,2015.)

Vuonna 2018 toteutetun kyselyn mukaan kyselyn kohteena olevista yrityksistä 23 % toteuttaa API-rajapintansa hallintaohjelmistojen kautta. 10 % yrityksistä toteuttaa rajapintansa integraatioalustaratkaisujaan hyödyntäen. API hallintaan tarvittavat komponentit voivat siis olla osa laajempaa integraatioalustaratkaisua. Kuitenkin 60% rajapintojen luonnista ja hallinnasta on yhä kehittäjien vastuulla. (Jitterbit, 2018.)

3 API-RAJAPINTOJEN TIETOTURVA

Tässä luvussa käydään lävitse API-rajapintojen tietoturvan taustoja sekä mitä osa-alueita API-rajapintojen tietoturvaan sisältyy. Luvussa esitellään myös yleisimpiä haavoittuvuuksia, mitä käyttöön ja julkaisuun liittyy, sekä miten näiltä haasteilta voidaan suojautua teknisesti ja arkkitehtuurillisesti.

3.1 Taustaa

Samaan aikaan kun sovelluksia tulee pystyä käyttämään kaikkialta erilaisilla päätelaitteilla, tulee sovellusten käyttävien API-rajapintojen käytön olla kontrolloitua. API-rajapintojen julkaisu organisaatioiden sisäisten verkkojen ulkopuolelle tietoturvallisella tavalla onkin muodostumassa erittäin haastavaksi tehtäväksi IT-osastoille. (Tang, 2018.) Leaden (2018) kuvaa, että rajapintojen tietoturvakenttä on turvallisesti ja johdonmukaisesti toimivan kaupungin sijaan villi länsi, jossa eri standardit ja tavat toimia ovat ristiriidassa keskenään tai tarvittavat säännöt voivat olla puutteellisia. Tämä on johtanut monelta eri sektorilta tietoturvahyökkäysten tulvaan rajapintoja kohtaan. API-rajapintojen tietoturva on isommassakin mittakaavassa merkittävää huomioida, sillä esimerkiksi Aldossary (2016) listaa seitsemän pilvipalveluiden käyttöön liittyvää kriittisintä tietoturvauhkaa, joista yhtenä on API-rajapintojen tietoturvariskit.

API-rajapintojen tietoturva ja haavoittuvuudet ovat yhä suhteellisen tuntematon alue useimmille organisaatioille ja jopa monille tietoturva-alan ammattilaisillekin. Valitettavasti suurin osa hyökkäyksistä ja haavoittuvuuksista tulee tietoisuuteen ja havaitaan vasta siinä vaiheessa, kun niistä tiedotetaan ja uutisoidaan laajasti ja julkisesti. Vaikka haavoittuvuus ei ole hyökkääjien tai rajapintojen kehittäjien tiedossa, ei se tarkoita sitä, etteikö haavoittuvuutta olisi olemassa (unknown unknown). Huonoimmassa tapauksessa haavoittuvuudesta ei tiedä kukaan muu kuin se taho, joka on jo hyödyntänyt haavoittuvuutta ja murtautunut järjestelmään tai sovellukseen sisään (known unknown). (Macy, 2018.)

3.2 Haavoittuvuudet

Haavoittuvuus on vika tai puutteellisuus järjestelmässä, jonka avulla hyökkääjä voi ohittaa tietoturvasuojaukset. (Alhazmi, Malaiya, Ray, 2007) Tangin (2018) mukaan internet ja sen taustalla toimiva http-protokolla ei ole suunniteltu ottamaan huomioon nykyajan vaatimaa tietoturvanäkökulmaa. Tarvittavia ominaisuuksia ei siis ole olemassa ja oletuksena tämä aiheuttaa ongelman varsinkin REST-pohjaisissa rajapintojen toteutuksissa, jotka usein sisältävät haavoittuvuuksia, kuten injektointihyökkäykset, arkaluontoisen informaation vuotaminen, autentikoinnin ja pääsynhallinnan puutteet tai Cross-site Scripting (XSS)-haavoittuvuus. REST-pohjaisiin rajapintoihin kohdistuneita hyökkäyksiä on tapahtunut ja tapahtuu jatkuvasti (Tang, 2018). API-rajapintojen haavoittuvuudet ovat hyvin samanlaisia kuin web-sovelluksissa esiintyvät yleisimmät haavoittuvuudet (Ibrahim ym., 2014).

Havaintoja API-tietoturvan haavoittuvuuksista tai toteutusten heikkouksista tietoturvan näkökulmasta, ja näihin toteutuksiin kohdistuneista onnistuneista hyökkäyksistä, on lukuisia. Esimerkiksi vuonna 2016 tutkijat havaitsivat, että Nissanin sähköauto Leafin rajapintoihin pystyttiin tekemään kyselyjä käyttäjää millään tavalla tunnistamatta, eli ilman autentikointia. Rajapintojen avulla pystyi hakemaan tietoa auton analytiikasta ja auton omistajasta, sekä ohjaamaan auton ilmastointilaitteen toimintaa. API-rajapinnat olivat tehty auton käyttäjän mobiilisovellusta varten, mutta syystä tai toisesta jätetty suojaamatta käyttäjän tunnistamisen näkökulmasta. API-rajapinnat eivät siis vaatineet minkäänlaista autentikointia, vaan toiminta perustui pelkästään auton tunnisteen lähettämiseen REST-kutsun parametrina, esimerkki kutsusta kuvassa 3. (Hunt, 2016.) Kyseinen tapaus on hyvä esimerkki siitä, kuinka rajapinnan toteutuksessa on sekoitettu avoimen ja yksityisen APIn ominaispiirteitä – käyttäjän tunnistus on tehty kuten avoimessa rajapinnassa, vaikka rajapinnan oikea kategorisointi on sisäinen rajapinta.

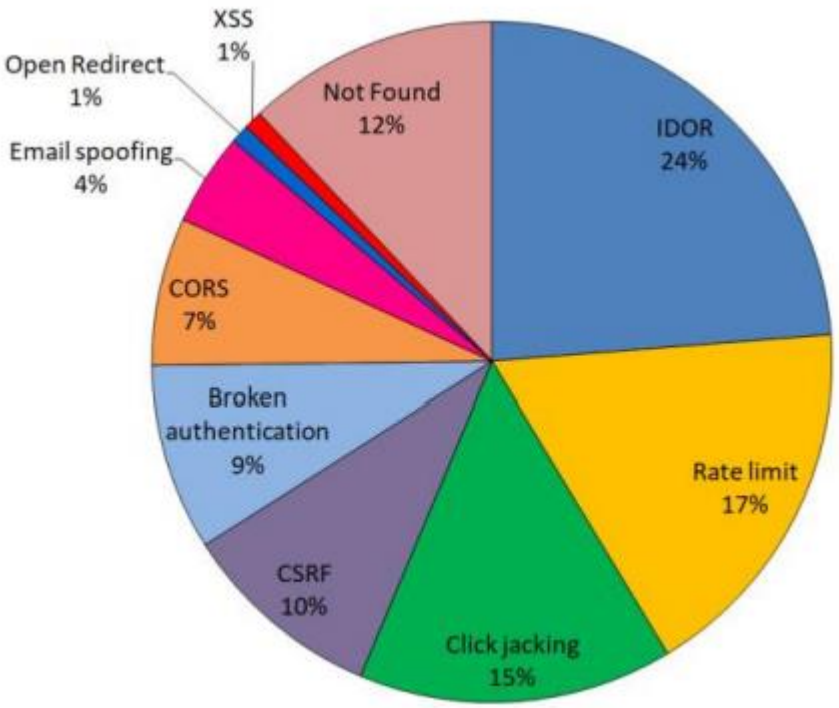
```
GET https://[redacted].com/orchestration_1111/gdc/BatteryStatusRecordsRequest.php?
RegionCode=NE&lg=no-NO&DCMID=&VIN=SJNFAAZE0U60XXXXX&tz=Europe/Paris&TimeFrom=2014-09-
27T09:15:21
```

Kuva 3 Esimerkkikutsu Nissan LEAF-sähköauton API-rajapintaan (Hunt, 2016)

Haavoittuvuudet voivat kohdistua useaan eri komponenttiin, kun katsotaan järjestelmäkokonaisuutta, josta yksittäinen API-palvelu voi muodostua. Jotkin haavoittuvuudet voivat antaa pääsyn ja muokkausoikeudet API-rajapintojen takana (backend) toimivaan tietokantaan. Toiset haavoittuvuudet antavat mahdollisuuden injektoida tai muokata taustalla toimivaa web-sovellusta. Jotkut haavoittuvuudet taasen perustuvat tiedon salaukseen liittyvän kerroksen peukalointiin, jonka myötä siirrettävään dataan voidaan päästä käsiksi tai sitä voidaan muokata. Kohteita (Bhuiyan ym.,2018.)

The Open Web Application Project on järjestö, jonka toiminta tähtää edesauttamaan tietoturvallisten sovellusten elinkaaren hallinnassa aina lähtien sovellusten luomisestä niiden ylläpitoon. Järjestön tunnetuin projekti on ”OWASP (Open Web Application Security Project) Top 10” johon päivitetään säännöllisesti kymmentä vakavinta haavoittuvuutta web sovelluksiin liittyen. (OWASP, 2017) Näistä haavoittuvuuksista yhdeksän kymmenestä ovat API-toteutuksiin tavalla tai toisella liitettäviä (Macy, 2018). Järjestö julkisti 2018 joulukuussa uutisen, jossa kerrottiin uuden pelkästään API-rajapintojen tietoturvaan keskittyvän projektin alkamisesta. Projektin nimi on *OWASP API Security Projecta* ja yhtenä tuotoksena vuoden 2019 aikana julkaistaan samanlainen top 10-lista API-haavoittuvuuksista ja niiden torjuntakeinoista kuin web-sovelluksista. (OWASP, 2019.)

Bhuiyan ym. (2018) testasivat 60 julkista API-toteutusta tavoitteenaan kartoittaa rajapinnoista löytyviä haavoittuvuuksia. Haavoittuvuudet jaettiin kolmeen eri luokkaan (low-medium-critical). API-rajapinnat olivat sisältötyypiltään mahdollisimman heterogeeninen otos sisältäen toteutuksia esimerkiksi eläimiin, kirjoihin, musiikkiin, säähän tai ruokaan liittyen. 88 % tutkituista, julkisista rajapinnoista piti sisällään haavoittuvuuden. Yleisin haavoittuvuus (24%) oli ”IDOR”-tyylinen, jossa muuttamalla REST-kutsuun liitettyä parametria voidaan saada haettua tietoja esimerkiksi muista käyttäjistä, organisaatioista tai aiemmin kuvatussa Nissanin sähköautoesimerkissä saadaan esille tietoja toisten omistamista autoista. Muut havaitut haavoittuvuustyypit löytyvät kuvasta 4.



Kuva 4 Haavoittuvuustyypit (Bhuiyan ym. 2018)

Kuten todettua, olemassa olevat ja kehitettävät API-rajapinnat eivät aina noudata parhaita tietoturvakäytänteitä. Kehittäjät voivat vaipua turvallisuudentunteeseen, ettei heidän toteutukseensa voi kohdistua hyökkäystä. (Leaden, 2018.) Samoin sisäisten API-rajapintojen suunnittelussa ja toteutuksessa voidaan luottaa liikaa pelkän sisäverkon turvallisuuteen ja jätetään käyttäjän tunnistautuminen ja oikeuksien tarkistus pois rajapinnan toteutuksesta (Moilanen, 2018). Luottamus muihin tekijöihin ja tämän vuoksi tarvittavien suojausten käyttämättä jättäminen aiheuttaa siis yhden haavoittuvuuden tyypin. Myös Macyn (2018) mukaan yksi haavoittuvuuksista, joka yleisimmin jätetään huomioimatta, on API-rajapinnan käyttäjän identiteetin ja pääsynhallinnan toteutus (Identity and Access Management).

Macy (2018) esittää myös toisen yleisen haavoittuvuustyypin, jota ei yleensä löydy 'top 10'-tyylisistä listauksista. Haavoittuvuus on API-rajapinta-arkkitehtuuri itsessään tapauksissa, jossa on käytössä hallintaohjelmiston kautta tai ilman API Gateway-ratkaisu. API Gateway muodostuu kriittiseksi, ja sitä kautta hyökkäysten kohteeksi, koska organisaation kaikki API-rajapinnat suojataan ja julkaistaan yhden pisteen kautta.

3.3 Suojautuminen

Koska web-sovellukset ovat samankaltaisia haavoittuvuuksiltaan kuin API-rajapinnat, samat tekniset suojautumiskeinot kuin web-sovellusten suhteen ovat päteviä pääpiirteissään myös API-rajapinnoille. Ohjeistuksia ja parhaita käytänteitä yleisimpien teknisten haavoittuvuuksien välttämiseksi API-rajapintojen kehittämisen yhteydessä löytyy lukuisia. Aiemmin haavoittuvuuksien yhteydessä mainittu OWASP tarjoaa julkaistussa web-sovellusten top 10-haavoittuvuuslistauksessa jokaiselle haavoittuvuudelle myös suojautumiskeinot. (OWASP,2017.)

Hallintaohjelmistoista yleensä aina löytyvä komponentti API Gateway itsessään kokoaa yhteen teknologioita, käytänteitä ja poliitikoita, joilla suojautua haavoittuvuuksilta. API gateway siten tulee suojata erityisellä huolella. Macy (2018) määrittelee, että tietoturvallisesti toteutettu API Gatewayn tulisi pitää sisällään useissa eri järjestelmän arkkitehtuurikerroksissa huomioon otettavat asiat, kuten:

- Käyttöjärjestelmä: API Gateway-arkkitehtuurin näkökulmasta käyttöjärjestelmätaso on alin kerros arkkitehtuuripinossa. Käyttöjärjestelmätasolla tulee toteuttaa tietoturvakovennukset ja estot, kuten ylläpitäjän oikeuksilla järjestelmän käytön estäminen, kolmansien osapuolten sovellusten esto tai järjestelmän eheyden tarkistaminen käynnistyksen yhteydessä.
- Ratkaisuarkkitehtuuri: Arkkitehtuurin tulee olla tietoturvanäkökulmasta suunniteltu, jotta ylläpito, poliitikoiden säilöminen tai muut herkkäluonteiset sisällöt, kuten salasanat ovat kryptattu kaikissa niiden käytön eri vaiheissa.

- Eheys- ja vakaus: API Gatewayn tulee suoriutua suurista volyymeista suorituskykyisesti, mutta kuitenkin siten että tunkeutumisyritykset ja muut hyökkäykset eivät saa vaikuttaa toimintakykyyn. Tämä voi tarkoittaa käytettyjen protokollien kovennuksia, kutsujen osoitepolkujen kovennuksia tai käytetyn tietomallin ja sen parserin toteutuksen siten, että viestipohjaisesti ei voida suorittaa hyökkäystä rajapintaa kohden.

Tunnistautuminen ja siihen liittyvät taustajärjestelmät sekä identiteetinhallinta nousevat esiin lukuisissa artikkeleissa yhdeksi tärkeimmistä osa-alueista API-rajapintojen tietoturvassa. (Aldossary, 2016; Tang,2015; Macy, 2018.) Tunnistautumisen tekninen toteutus, tai jopa sen puute inhimillisestä tai muusta syystä, on osa-alue, joka voidaan säännöstöillä tai poliitikoilla suojata helposti ja yhtenäisesti. Hallintaohjelmistot tuovat valmiita ratkaisuja tähän, mutta yhtä hyvin organisaatio voisi suojautua aiheeseen liittyviltä haavoittuvuuksilta noudattamalla API-rajapintojen kehityksessä määriteltyjä säännöstöjä ja poliitikoita.

3.4 Arkkitehtuurimalli

Tang (2015) määrittelee kolme eri vahvuustasoa API-kerroksen tietoturvalle julkaisussaan ”Multi-factor web API security for securing Mobile Cloud”. Vahvuustasot ovat heikko, keskitaso ja korkea. Artikkelissa esitetty malli ei ole millään tavoin mobiiliympäristöön lukittu. Esitetyt tekniset ratkaisut eivät ole mobiilipäätelaitteisiin sidottuja tai vain niissä käytössä olevia teknologioita. Yksi REST-pohjaisten rajapintatoteutusten eduista on riippumattomuus käytettävistä asiakassovelluksista, joten esitelty jaottelu on käytettävissä missä tahansa ratkaisussa, jossa on REST-pohjaisia rajapintoja toteutettuna. Mallissa jokaiselle tasolle määritellään erikseen resurssit, teknologiset ratkaisut tai palvelut, joilla kyseinen API-tietoturvataso saavutetaan.

Rajapinnat ja resurssit on suojattu heikoimmalla suojaustasolla vain perinteisellä yhdistelmällä, jossa käytetään SSL (Secure Socket Layer)-salausta sisällön suojaamiseen ja autentikoinnissa HTTP /1.0 määrityksen mukaista basic-autentikointia. Heikon tason API-tietoturvaratkaisussa käytetään yhden tekijän (one-factor) autentikointia ja tämä tapa on yleisesti käytössä julkisissa API-julkaisuissa, joissa ei ole tiukempia vaatimuksia tiedon turvaamiseksi. Esimerkiksi yleinen versionhallintapalvelu GitHub on oletuksena suojattu tällä heikolla tasolla. Heikko API-tietoturvataso suojaa ainoastaan kuljetuskerroksessa APIn julkaisijan ja hyödyntäjän välistä liikennettä, jolloin itse data jää suojaamatta. Tämänkaltaista ratkaisua ei voida suositella käytettäväksi käyttötapauksissa, joissa on esimerkiksi taustalla liiketoiminnallisia prosesseja tai jos siirrettävä data on sisällöllisesti sensitiivistä.

Keskitason tietoturvataso koostuu kahdesta komponentista, jotka edelleen jakautuvat tarkempiin osioihin:

1. SSL-salaus samalla tavalla kuin tasolla heikko.
2. Modernit API-spesifi tietoturvamekanismi, joka koostuu edelleen useasta eri osa-alueesta:
    - Tunnistautuminen
    - Pääsynhallinta
    - Datan kryptaaminen
    - Tietoturvamonitorointi ja hyökkäysten estäminen

Vahvin tietoturvaratkaisu pitää sisällään sekä heikoimman tason kryptauksen ja käyttäjän autentikoinnin että keskitason modernit tietoturvamekanismit ja lisää vielä yhden tekijän: API-rajapinnan ja taustalla olevan palvelun välisen yhteyden suojauksen. Lisäksi API-toteutusten tunnistautuminen ja pääsynhallinta on kytketty organisaation laajuisiin identiteetin ja pääsynhallintaratkaisuihin.

4 YHTEENVETO

Tässä tutkielmassa tutkittiin API-rajapintoja ja niiden hallintaan ja tietoturvaan liittyviä aiheita. Tutkielmassa selvitettiin mitä rajapinnoilla tarkoitetaan ja miten niitä voidaan kategorisoida, sekä mitä rajapintojen hallinta tarkoittaa. Toisena näkökulmana rajapintoihin selvitettiin mitä on rajapintojen tietoturva ja erityisesti mitä haavoittuvuuksia rajapintoihin kohdistuu, sekä yleisiä suojautumiskeinoja haavoittuvuuksiin.

Tutkielmassa API-rajapinnoilla tarkoitetaan REST-arkkitehtuurityylin mukaisia toteutuksia, jotka toimivat HTTP-protokollaa hyödyntäen. (Fleming, 2000) Modernit REST API-rajapinnat ovat web palveluiden (web services) evoluution tulos. Aiemmin suosittu tapa toteuttaa web-pohjaisia rajapintoja on ollut SOAP-standardi, jonka käyttö on pienentynyt vuosi vuodelta. SOAP-toteutuksilla on omat vahvuutensa tietoturvan ja luotettavuuden näkökulmasta, mutta varsinkin julkisissa rajapintojen toteutuksissa REST on viime vuosina ollut lähes ainoa toteutustapa. REST on helposti omaksuttava ja yksinkertainen toteutustapa, jonka haittapuolena nähdään luotettavuus ja tietoturva.

API-rajapintoja voidaan kategorisoida eri tavoin, joista yleisin malli on jakaa rajapinnat kahteen osaan: julkisiin ja yksityisiin. Yksityiset rajapinnat jaotellaan vielä kahteen eri alikategoriaan; sisäinen- ja kumppanitoteutus. Eri kategorioiden rajapinnoilla on tiettyjä ominaispiirteitä. Julkiset rajapinnat ovat esimerkiksi usein anonyymejä, eikä niissä tunnisteta käyttäjää millään tavoin.

Rajapintojen käyttöön ja elinkaaren hallintaan liittyy lukuisia eri haasteita: kapasiteetin hallinta, käyttäjien hallinta, käytön hallinta, analytiikka, tietoturva, versionhallinta ja niin edelleen. Omalta osaltaan näihin haasteisiin vastaa API-rajapintojen hallinta- ja tehtävään tarkoitetut ohjelmistot, joista löytyy työkalut, jotka toteuttavat ratkaisuja edellä mainittuihin osa-alueisiin. Lisäksi hallintaohjelmistoista löytyy valmiita konfigurointeja ja poliitikoita, joita käyttäen voidaan varmistaa, että API-rajapintojen kehityksessä ja julkaisussa käytetään yhtenäisiä käytänteitä ja ratkaisuja.

API-rajapintojen tietoturva on haasteellinen aihe monille organisaatioille. Teknisesti REST-pohjaisten API-rajapintojen tietoturva on lähtötasoltaan haastava käytettyjen protokollien vuoksi. Rajapintojen haavoittuvuuksilta suojautuminen on teknisiltä yksityiskohdiltaan hyvin samanlaista kuin web-sovelluksien haavoittuvuuksilta suojautuminen. Tutkielmassa ei tarkemmin teknisellä tasolla

esitelty haavoittuvuuksia ja niiden suojautumiskeinoja, vaan käytiin yleisellä tasolla lävitse havainnot kirjallisuuden pohjalta.

API-rajapintojen tietoturvan haasteisiin on vaikuttanut standardien ja parhaiden käytänteiden puuttuminen. Organisaatioissa API-rajapintojen tietoturvan toteutus on laajasti ollut kehittäjäkohtaista eikä usein minkäänlaisia hallintaohjelmistoja ole käytetty kehittämiseen tai julkaisuun. API-hallintaohjelmistot tuovat omalta osaltaan suojautumiskeinoja ja teknologioita helposti ja tuotteenomaisesti kehittäjille käyttöön.

Suojautumiskeinot vaikuttavat olevan harvinaisempi aihealue tieteellisessä kirjallisuudessa kuin uhkien tunnistaminen. Kirjallisuudessa nostetaan teknisten toteutusten ja hallintaohjelmistojen käytön lisäksi esille kehittäjänäkökulmaa ja oikeastaan koko organisaation näkökulmaa ja suhtautumista API-rajapintojen tietoturvaan. API-rajapintoja ei tulisi tarkastella pelkkänä yksittäisenä ja erillisenä teknisenä toteutuksena, jonka yksittäinen kehittäjä luo, vaan tietoturvamielessäkin tuotemainen lähestymistapa toisi lisähyötyjä. Tässä työssä on rajatusti käsitelty muita, laajempia näkökulmia API-rajapintojen tietoturvaan vaikuttaviin tekijöihin, kuten esimerkiksi organisaatioiden tietoturvan hallinta ja siihen liittyvät tietoturvapolitiikat, niiden noudattaminen tai noudattama jättäminen.

Tutkielma oli kirjoittajan ensimmäinen akateeminen tutkimus. Tarkoituksena on ollut oppia akateemisen tutkimuksen tekoa ja saada ylipäätään tuotetuksi valmis tutkielma. Tämä varmasti osaltaan näkyy tutkielman yleisluontoisena katsauksena aiheeseen sekä tutkielman argumentaatiossa tai sen puutteena.

Mielenkiintoinen jatkotutkimuksen kohde on API-rajapintojen tietoturvan toteuttaminen kehittäjän näkökulmasta; miten ja miten hyvin olemassa olevat työkalut ja ohjelmistot tukevat tietoturvallisten rajapintaratkaisujen luomista. Myös kysymys siitä, mitä työkaluja ja ohjeistuksia eri organisaatioissa on käytössä API-rajapintojen näkökulmasta, olisi kiinnostava tutkimuksen aihe.

LÄHTEET

Alhazmi, O. H., Malaiya, Y. K., Ray, I. 2007. Measuring, analyzing and predicting security vulnerabilities in software systems. Computers & Security. Volume 26, Issue 3, May 2007, Sivut 219-228

Aldossary, Sultan. (2016). Data Security, Privacy, Availability and Integrity in Cloud Computing: Issues and Current Solutions. International Journal of Advanced Computer Science and Applications, Vol. 7, No. 4

Alagarasan, Vijay. (2015). API Management Introduction and Principles. Service Technology Magazine.

Apigee. 2015. The Definitive Guide to API Management. Saatavilla osoitteessa https://pages.apigee.com/rs/351-WXY-166/images/apigee-ebook-api-mgmt-2015-07.pdf

Battle, R., Benson, Edward. (2008). Bridging the semantic Web and Web 2.0 with Representational State Transfer (REST). Journal of Web Semantics, Volume 6, Issue 1, sivut 61-69

Berners-Lee, Tim. (1989). Information Management: A Proposal. CERN, W3C. Saatavilla osoitteesta http://www.w3.org/History/1989/proposal.html

Bhuiyan, Touhid & Begum, Afsana & Rahman, s & Hadid, I. (2018). API vulnerabilities: Current status and dependencies. International Journal of Engineering and Technology (UAE). 7. Sivut 9-13.

Brodsky, L., Oakes, L. (2017). Data sharing and open banking. McKinsey & Com

Burns, C., Ferreira, J., Hellmann, T. D. ja Maurer J., (2012). Usable results from the field of API usability: A systematic mapping and further analysis. 2012 IEEE Symposium on Visual Languages and Human-Centric Computings.

Fielding, Roy. (2000). Architectural Styles and the Design of Network-based Software Architectures

García De Prado, A., G. Ortiz and J. Boubeta-Puig. (2017). CARED-SOA: A Context-Aware Event-Driven Service-Oriented Architecture. IEEE Access, vol. 5, pp. 4646-4663

Jacobson, D., Brail, G., Woods, D. (2011). Apis: A Strategy Guide. O'Reilly Media, Inc.

Jitterbit. 2018. The 2018 State of API Integration. Saatavilla osoitteesta http://info.jitterbit.com/rs/017-VOG-951/images/State_of_API_Integration_2018.pdf Haettu 12.4.2019

Hunt, Troy. 2016. Controlling vehicle features of Nissan LEAFs across the globe via vulnerable APIs. Saatavilla osoitteesta https://www.troyhunt.com/controlling-vehicle-features-of-nissan/ V

Kopecký J., Fremantle P., Aziz B. (2015) Web API Management Meets the Internet of Things. In: Gandon F., Guéret C., Villata S., Breslin J., Faron-Zucker C., Zimmermann A. (eds) The Semantic Web: ESWC 2015 Satellite Events. ESWC 2015. Lecture Notes in Computer Science, vol 9341. Springer, Cham

Kopecky, J., Fremantle, P., Boakes, R. (2014) A history and future of Web APIs. Inf.

Kumari, V. 2015. Web Services Protocol: SOAP vs REST. International Journal of Advanced Research in Computer Engineering & Technology. Volume 4 Issue 4. May 2015. Technology 56, 90–97

Lane, K., (2019) History of APIs. apievangelist.com., Haettu osoitteesta http://history.apievangelist.com

Leaden, G. , Zimmermann, M. , DeCusatis C. and A. G. Labouseur. 2017. An API honeypot for DDoS and XSS analysis. IEEE MIT Undergraduate Research Technology Conference (URTC).

Lensmar, O., (2013). Is REST losing its flair – REST API Alternatives. ProgrammableWeb blog. (2013) Saatavilla osoitteesta https://www.programmableweb.com/news/rest-losing-its-flair-rest-api-alternatives/analysis/2013/12/19

Macy, Jason. (2018) How to build a secure API gateway. Network Security, Volume 2018, Issue 6, 12-14

Maleshkova, M.,Pedrinaci,C.,Domingue,C. (2010). Investigating Web APIs on the World Wide Web. 2010 Eighth IEEE European Conference on Web Services, Ayia Napa, 2010, pp. 107-114.

Mohamed Ibrahim, B., & Mohamed Shanavas, A. R. (2014). Applying Security for RESTful Web Services–Limitations and Delimitations. International Journal of Emerging Technology and Advanced Engineering, 4(9).

Moilanen, J., Niinioja, M., Seppänen, M. & Honkanen, M. (2018). API talous 101. BALTO print, Liettua 2018.

Mulesoft. What is API management?. 2018. https://www.mulesoft.com/resources/api/what-is-api-management

Osaango. Why should you categorize your APIs?. Saatavilla osoitteesta https://www.osaango.com/blog/why-should-you-categorize-your-apis#references

OWASP. 2017. OWASP Top 10 – 2017. The Ten Most Critical Web Application Security Risks. Saatavilla osoitteesta https://www.owasp.org/images/7/72/OWASP_Top_10-2017_%28en%29.pdf.pdf

OWASP. 2019. OWASP API Security Project. Saatavilla osoitteesta https://www.owasp.org/index.php/OWASP_API_Security_Project

Eduardo Mosqueira-Reya, David Alonso-Ríos , Vicente Moret-Bonillo, Isaac Fernández-Varelaa, Diego Álvarez-Estévez. 2018. A systematic approach to API usability: Taxonomy-derived criteria and a case study. Information and Software Technology. Numero 97, sivut 46-63.

Santos, Wendell. (2018). Research Shows Interest in Providing APIs Still High. Saatavilla https://www.programmableweb.com/news/research-shows-interest-providing-apis-still-high/research/2018/02/23 7.4.2019

Santos, Wendell. (2017). Which API Types and Architectural Styles are Most Used? Saatavilla https://www.programmableweb.com/news/which-api-types-and-architectural-styles-are-most-used/research/2017/11/26

Tan, W., Fan,Y., Ghoneim,A., Hossain, M. A. , Dustdar, S. (2016). From the Service-Oriented Architecture to the Web API Economy. IEEE Internet Computing, vol. 20, no. 4, pp. 64-68, July-Aug. 2016

Tang,L., Liubo Ouyang and W. Tsai, (2015) Multi-factor web API security for securing Mobile Cloud, 2015 12th International Conference on Fuzzy Systems and Knowledge Discovery (FSKD), Zhangjiajie, 2015, pp. 2163-2168.

Tang,L. M. Little, (2014). API Governance and Management. Service Technology Magazine, no. LXXXVI, 2013.

Tiago Espinha, Andy Zaidman, Hans-Gerhard Gross. (2014). Web API growing pains: Stories from client developers and their code, Software Maintenance Reengineering and Reverse Engineering (CSMR-WCRE) 2014 Software Evolution Week - IEEE Conference on, pp. 84-93

Vukovic, Maja. (2016). Magazine Communications of the ACM Volume 59 Issue 3, March 2016 Pages 35-37