jamk

Web-sovellusten tietoturva

Jon Pekkonen

Opinnäytetyö, AMK
Toukokuu 2022
Tietojenkäsittely ja tietoliikenne
Tietojenkäsittelyn tutkinto-ohjelma

jamk | Jyväskylän ammattikorkeakoulu
University of Applied Sciences

jamk

Kuvailulehti

Pekkonen, Jon

Web-sovellusten tietoturva

Jyväskylä: Jyväskylän ammattikorkeakoulu. Toukokuu 2022, 41 sivua.

Tietojenkäsittely ja tietoliikenne. Tietojenkäsittelyn tutkinto-ohjelma. Opinnäytetyö AMK.

Julkaisun kieli: suomi

Verkkojulkaisulupa myönnetty: kyllä

Tiivistelmä

Web-sovellusten tietoturva on jatkuvassa muutoksessa. Uusia haavoittuvuuksia löydetään jatkuvasti ja olemassa olevia paikataan. Silti yleisimpien tietoturvauhkien lista pysyy melko samana vuodesta toiseen. Iso kysymys onkin, paljonko tavanomainen sovelluskehittäjä pohtii tietoturvaa sen hetkisessä projektissaan? Ei kovin paljoa, jos uhat ja haavoittuvuudet koetaan abstrakteina tekijöinä, joihin ei itse voi juurikaan vaikuttaa. Siksi onkin tärkeää, että jokaisella sovelluskehittäjällä olisi edes yleiskuva yleisistä tietoturvauhista ja haavoittuvuuksista sekä tietotaidot näiden uhkien torjuntaan.

Opinnäytetyön toimeksiantajalla, Jyväskylän ammattikorkeakoulun tietojenkäsittelyn tutkinto-ohjelmalla, oli tarve web-sovellusten tietoturvaa käsittelevälle opintojaksolle. Opinnäytetyön tavoite oli tutkia yleisimpiä web-sovellusten tietoturvauhkia ja tämän pohjalta luoda sisältösuunnitelma toimeksiantajan tarvitsemalle opintojaksolle, joka tarjoaisi tietojenkäsittelyn opiskelijalla yleiskuvan erilaisiin sovellukseen kohdistuviin tietoturvauhkiin.

Työn rajauksessa ja käsiteltävien tietoturvauhkien valinnassa hyödynnettiin OWASP:n kymmenen ajankohtaisimman haavoittuvuus tyypin listaa. Jokaista käsiteltyä haavoittuvuus ja uhka kokonaisuutta tutkittiin tarkemmin hyödyntäen julkisia sovelluskehittäjä- ja tietoturvayhteisöjen julkaisemia materiaaleja. Käsiteltävistä tietoturvauhista tutkittiin niiden taustasyitä sekä keinoja niiden ehkäisyyn tai torjuntaan.

Tutkimusosan tulokset koottiin kokonaisuuksiksi, joissa tarkasteltiin yhtä haavoittuvuustyyppiä kerrallaan. Kokonaisuuksissa tarkasteltiin haavoittuvuuden taustasyitä sekä hyökkäyksiä, joille haavoittuvuus sovelluksen altistaa ja lopuksi keinoja korjata kyseinen haavoittuvuus. Koottujen tietopakettien pohjalta laadittiin toimeksiantajan opintojaksolle sisältösuunnitelma, jossa määriteltiin eri moduulien sisältö sekä esimerkkitehtäviä.

Avainsanat (asiasanat)

Tietoturva, web-sovellus, kyberturvallisuus, ohjelmistokehitys

Muut tiedot (salassa pidettävät liitteet)

jamk

Description

Pekkonen, Jon

Web application security

Jyväskylä: JAMK University of Applied Sciences, May 2022, 41 pages.

Information and communication technologies. Degree Programme in Business Information Technology. Bachelor's thesis.

Permission for web publication: Yes

Language of publication: Finnish

Abstract

Web application security is in constant change. New vulnerabilities are found, and existing ones are patched. Regardless list of most common information security threats for web applications stays the same with little to no changes. This raises the question how much ordinary web application developer thinks about security in their project? If security threats and vulnerabilities are thought of as abstract factors that can hardly be influenced by oneself, probably not much. Therefore, it is important for every web application developer to have even basic understanding of common security threats and vulnerabilities. As well as know-how to combat those threats.

The client of the thesis, Jyväskylä University of Applied Sciences' degree program in business information technology, had a need for a course in web application security. The aim of the thesis was to study the most common security threats which target web applications. And on basis of this study create content plan for the course required by the client. This course should provide the student with an overview of various security threats targeting web applications.

OWASP's list of the ten most significant vulnerabilities was used to limit work and select the security threats to be studied. Each of the vulnerabilities and threats discussed was researched in more detail using public materials published by web application developers and security communities. The root causes of the security threats under consideration and means to prevent or combat them were examined.

The results of the study section were compiled into summaries that presented one type of vulnerability at a time. Summaries explain underlying causes for threat or vulnerability as well as the cyberattacks to which the vulnerability could expose an application. Finally, summaries present means to fix vulnerabilities or to mitigate the effects of a possible cyberattack. Based on these summaries content plan was created for client's upcoming course. The content plan is divided into modules each of which contains references to relevant reading materials and couple of exercise assignments to be used during the course.

Keywords/tags (subjects)

Cyberattack, web application, security, vulnerability

Miscellaneous (Confidential information)

Sisältö

Kuviot

Käsitteet ja lyhenteet

API

Application programming interface, suomeksi ohjelmointirajapinta. Esimerkiksi HTTP:n yli kutsuttavat backend funktiot ovat rajapinnassa josta käyttöliittymä voi niitä hyödyntää.

CORS (Cross-Origin Resource Sharing)

CORS on selaimessa oleva valinnainen ominaisuus, jolla voi hallitusti löysätä saman alkuperän käytäntöä (engl. Same-origin policy, SOP). Saman alkuperän käytännön mukaan www-sivun ladattuun sisältöön ei voi turvallisesti luotta, joten niillä on oikeus käyttää resursseja vain samasta lähteestä.

Full stack sovellus

Sovelluskokonaisuus, joka kattaa kaiken tietokannan ja käyttöliittymän välillä. Usein puhutaan full stack sovelluskehityksestä, etenkin web-sovellusten kehitystyössä. Full stack -sovelluskehittäjä tuntee kaikki sovelluksen osa-alueet ja niiden tekniikat.

Haavoittuvuus

Heikkous sovelluksen rakenteessa, suoritusympäristössä tai verkossa. Kaikkien haavoittuvuuksien paikkaamien, tai edes löytäminen, on lähes mahdotonta vähänkään suuremmassa ohjelmisto kokonaisuudessa. (Alexander 2021.)

HTTP ja HTTPS

Internetissä palvelinten ja selainten välisessä kommunikaatiossa käytettävä tiedonsiirto protokolla. Lyhenne sanoista ”Hypertect Transfer Protocol”. HTTPS on tämän protokollan salattu versio. Sijoittuu OSI-mallissa seitsemännelle kerrokselle.

Käyttöliittymä - frontend

Web-sovelluksen käyttäjälle näkyvä osuus, sovelluksen käyttöliittymä, esimerkiksi web-sivu tai appikaupasta ladattava sovellus. Käyttöliittymä pyytää palvelimelta tietoa ja esittää sen käyttäjäystävällisessä muodossa.

OSI-malli (Open Systems Interconnection Reference Model)

OSI- malli esittää tiedonsiirtoprotokollat jaettuna seitsemään kerrokseen tai tasoon. Jokainen kerros käyttää yhtä alemman kerroksen palvelua ja tarjoaa omia palveluitaan ylemmälle kerrokselle.

Palvelinohjelma - backend

Web-sovelluksen käyttäjälle näkymätön osa, jolta käyttöliittymä pyytää tietoa tarpeen mukaan. Käsittelee ja tarjoilee tietoa pyyntöjen perusteella, tekee tietopyyntöjä tietokannalle tarpeen mukaan. Puhekielessä tähän saatetaan viitata nimellä palvelin/serveri, backend on palvelintietokoneella käynnissä oleva ohjelma.

SQL (Structured Query Language)

Relaatiotietokantojen kyselyjen tekemiseen käytettävä kyselykieli. Kielestä on useita variaatioita, joita eri tietokantojen hallintajärjestelmät käyttävät.

TCP (Transmission Control Protocol)

Yksi internetin ydinprotokollista, mahdollistaa luotettavan tiedonsiirron esimerkiksi selaimen ja palvelimen välillä. Protokollassa on sisäänrakennettuna virheen hallintaa esimerkiksi kadonneiden pakettien varalle. Sijoittuu OSI-mallin neljännelle kerrokselle.

Tietokanta

Kokoelma tallennettua ja järjesteltyä tietoa. Sovelluskehitys kontekstissa tietokannalla viitataan usein koko tietokantaohjelmistoon, joka tallentaa ja hakee pyyntöjen perusteella tietoa.

Uhka, tietoturvauhka

Tietoturvauhka on yleisnimitys toimijoille tai tapahtumille, jotka pyrkivät hyödyntämään sovelluksen haavoittuvuuksia saadakseen pääsyn joko järjestelmään itsessään tai sovelluksen käyttämiin tietoihin. Nämä voivat olla erilaisia haittaohjelmia tai esimerkiksi hakkeri. (Alexander 2021)

Web-sovellus

Tietokoneen selaimessa tai älylaitteessa käytettävä ohjelmisto, joka tarvitsee internet yhteyden toimiakseen. Kehitetty kokonaan tai osin käyttäen web-teknologioita kuten HTML, CSS ja JavaScript. Voidaan jakaa frontendiin ja backendiin.

1 Johdanto

Web-sovellusten tietoturva vaatimukset ja uhat muuttuvat jatkuvasti. Alalla on vanhoja entuudestaan tunnettuja uhkia ja haavoittuvuuksia, joita paikataan ajoittain mahdollisuuksien mukaan. Ohjelmointikieliä, kirjastoja ja sovelluskehyksiä kehitetään koko ajan ja kaikista näistä voi löytyä uusia haavoittuvuuksia tai muita tietoturvauhkia. Myöskään laajassa käytössä olevat työkalut eivät ole täysin turvallisia, niistä saattaa löytyä entuudestaan tuntemattomia haavoittuvuuksia tai päivitykset saattavat tuoda mukanaan jonkin odottamattoman haavoittuvuuden.

Jyväskylä ammattikorkeakoulun tietojenkäsittelyn tutkinto-ohjelmalla ei ole tietoturvalle omistettua opintojaksoa. Aihetta sivutaan eri opintojaksoilla, mutta erilliselle opintojaksolle on selkeä tarve. Tämän opinnäytetyön on tarkoitus olla ensimmäinen askel kyseisen puutteen paikkaamiseksi. Opinnäytetyö oli luonteeltaan kvalitatiivinen tutkimus, jossa tutkittiin erilaisia tietoturvauhkia olemassa olevaa kirjallisuutta hyödyntäen. Toimeksiannon mukaisesti lopullisena tavoitteena oli luoda pohja tietojenkäsittelyn tutkinto-ohjelman tarpeisiin soveltuvalle opintojaksolle. Tätä tavoitetta varten oli tutkittava erilaisia tietoturvauhkia, jotka jokaisen web-sovelluskehittäjän tulisi tuntea ja kuinka näihin uhkiin voi varautua. Tietoturvakilpajuoksun luonteesta johtuen kaiken kattavan ja aina ajankohtaisen uhkakokoelman luominen on mahdotonta, joten opinnäytetyö rajattiin koskemaan yleisimpiä ja ajankohtaisia web-sovelluksiin kohdistuvia uhkia.

Opinnäytetyö, ja sen pohjalta kehitettävä opintojakso, tuo uusille alaa opiskeleville, tuleville sovelluskehittäjille, edes pienen vilauksen tietoturvan maailmaan. Ohjelmointia ja sovelluskehitystä opetellessa tietoturva jää usein sivuosaan, kun keskitytään luomaan toimiva ohjelma. Kaikkein räikeimmät tietoturva virheet usein mainitaan, mutta aiheeseen syventyminen koko sovelluksen ja sen elinkaaren osalta jää kunkin opiskelijan mielenkiinnon varaan vapaa-ajalla opiskeltavaksi. Tämän seurauksena osa valmistuneista sovelluskehittäjistä ei välttämättä omaa merkittävää tietoturva osaamista, tämä ei automaattisesti ole vakava ongelma, etenkään suurempiin ohjelmistotaloihin työllistyville. Sen sijaan jollakin pienellä yrityksellä ei välttämättä ole tietoturvaan erikoistunutta ammattilaista palkkalistalla, jolloin jokaisen yksittäisen sovelluskehittäjän ammattitaito tietoturvan saralla korostuu.

2 Tutkimusasetelma

2.1 Työn tavoitteet ja rajaus

Toimeksiantajalla, Jyväskylän ammattikorkeakoulun tietojenkäsittelyn tutkinto-ohjelmalla (TIKO) ei ole tietoturva-aiheista opintojaksoa. Erityisesti web-sovellusten kehitykseen suuntautuvat opiskelijat tarvitsevat tietoturvaosaamista tulevassa työssään. Opinnäytetyön tavoitteena on tutkia web-sovelluksiin kohdistuvia tietoturvauhkia ja tutkimustulosten pohjalta konstruoida sisältösuunnitelma ja materiaalia tulevalle opintojaksolle.

Opinnäytetyön rajauksessa hyödynnetään toimeksiantajalta saatua karkeaa kuvausta opintojakson sisällöstä ja kohderyhmästä. Työ rajataan koskemaan web-sovelluksiin kohdistuvia uhkia, koska tulevat web-sovelluskehittäjät ovat työnpohjalta kehitettävän kurssin kohderyhmä. Tämän lisäksi rajauksessa otetaan huomioon uhkien vakavuus, yleisyys ja ajankohtaisuus. Toimeksiantajan kehitteillä olevan opintojakson laajuus olisi kaksi tai kolme opintopistettä, tämän otetaan huomioon opintojakson sisältösuunnitelmassa ja tämän työn lopputuloksen laajuudessa tulisi ottaa tämä huomioon.

2.2 Tutkimusmenetelmät

Opinnäytetyön tutkimusosassa käytetään laadullista eli kvalitatiivista tutkimusotetta. Laadullinen tutkimusote on omiaan tutkimukseen, jossa tutkitaan ilmiötä, sen syitä ja tavoitteena on ymmärtää ilmiö mahdollisimman hyvin. (Kananen 2017, 33.) Tämän työn kohdalla se tarkoittaa tietoturvauhkiin perehtymistä tavoitteena selvittää uhkien taustasyitä, miten niihin voi varautua ja mitä web-sovelluskehittäjän tulisi niistä tietää. Halutaan siis tuntea tutkittava asia mahdollisimman hyvin ja saada aiheesta hyvä kuvaus, jota toimeksiantaja voi hyödyntää opintojakson kehityksessä.

Tutkimusvaiheessa kerättävä aineisto tulee koostumaan erilaisista kirjallisuuslähteistä, muun muassa aiheesta kirjoitettu painettu kirjallisuus sekä erilaiset internetissä julkaistut artikkelit, blogit ja muu aineisto, joka on kyllin ajankohtaisia ja arvioidaan luotettavaksi. Web-sovellusten tietoturva kehittyy alana jatkuvasti ja uusia tietoturvauhkia tulee esille aika ajoin, joten aineisto pyritään pitämään mahdollisimman tuoreena. Aineiston luotettavuuden varmentamisessa hyödynnetään muuta kyseisestä aiheesta julkaistua materiaalia.

Tietoturvauhkia tutkittaessa kohteiden rajauksen pohjana hyödynnetään OWASP-säätiön vuonna 2021 julkaisemaa, Van der Stockin, Glasin, Smithlinen ja Giglerin (2021) laatimaa OWASP Top 10:2021 -listaa. OWASP tai Open Web Application Security Project® on 2001 toimintansa aloittanut, voittoa tavoittelematon säätiö, joka tekee työtä sovellusten tietoturvan parantamiseksi.

Opinnäytetyön loppuvaiheessa toteutetaan tutkimusvaiheessa kerätyn tiedon pohjalta sisältösuunnitelma toimeksiantajan tulevalle opintojaksolle. Tutkituista aihepiireistä koostetaan sisältösuunnitelmaan opintojakson kontaktikerran kokoisia kokonaisuuksia, sekä kullekin kokonaisuudelle esimerkkitehtäviä. Sisällön suunnittelussa hyödynnetään toimeksiantajan opintojaksolle antamia reunaehtoja:

- Laajuus 1–3 opintopistettä
- Toteutus lähiopetuksena
- Edeltävä osaaminen:
    - Perusteet web-ohjelmoinnissa (JavaScript, HTML ja CSS)
    - Perusteet tietokannoissa
    - Perusteet frontend-ohjelmoinnissa
    - Perusteet backend-ohjelmoinnissa

2.3 Tutkimuskysymykset

Opinnäytetyön tutkimusongelma on mitä tietojenkäsittelyn opiskelijan pitäisi tietää web-sovellusten tietoturvasta. Tästä voidaan johtaa tutkimuskysymykset, joihin vastaamalla ongelma ratkeaisi. Ensimmäinen askel on selvittää mitkä ovat ne tietoturvauhat, joista kohderyhmän tulisi olla tietoinen. Tämän jälkeen tutkitaan yksittäisiä uhkia ja pyritään selvittämään niiden toimintaa, sekä löytämään ratkaisuja tai varautumiskeinoja. Tällöin tutkimuskysymykset ovat seuraavat.

- Mitkä ovat yleisimmät web-sovelluksiin kohdistuvat tietoturvauhat?
- Miten kyseiset tietoturvauhat käytännössä toimivat?
- Miten näihin web-sovellusten tietoturvauhkiin voi varautua?

Tämän jälkeen voidaan kootun tiedon pohjalta luoda tietojenkäsittelyn tutkinto-ohjelman tarpeisiin soveltuvan opintojakson sisältösuunnitelma, sekä muuta toimeksiantajan tarpeelliseksi näkemää materiaalia.

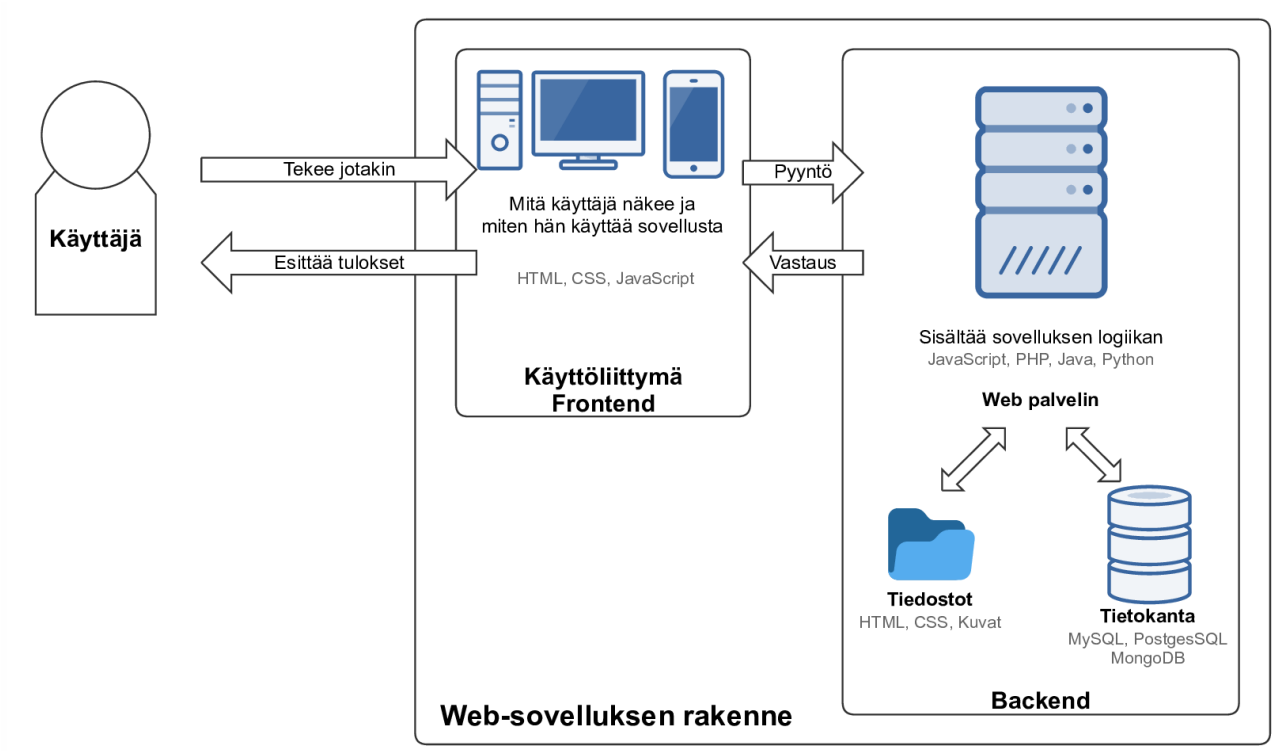
3 Web-sovellus ja sen toiminta

3.1 Web-sovellus kokonaisuutena

Opinnäytetyössä tarkastellaan tietoturvaa fullstack web-sovelluskehittäjän näkökulmasta. Opinnäytetyö on suunnattu web-sovellusten kehittämisestä kiinnostuneille ja jo jonkin verran aihetta opiskelleille, tästä huolimatta pyritään asioita, käsitteitä ja kokonaisuuksia avaamaan kylliksi, jotta myös aiheeseen perehtymätön saa muodostettua yleiskuvan web-sovelluksiin kohdistuvista tietoturvauhista. Näin ollen on tarpeen avata web-sovellusten toimintaa ja rakennetta.

Web-sovelluksen kehitystyössä sovelluskokonaisuus jaetaan usein käyttöliittymään ja palvelimeen. Usein näistä käytetään niiden englanninkielisiä nimiä frontend ja backend. Kuviossa 1. esitetään miten eri sovelluksen palat jakaantuvat tällaisessa jaossa. Sovelluksen eri osat kommunikoivat keskenään lähettämällä HTTP-pyyntöjä ja vastauksia (Choi 2020). Vaikka sovellusten välinen liikenne toimii HTTP-protokollan yli ei ohjelmisto kehittäjä käytännössä koskaan luo näitä pyyntöjä ja vastauksia niin sanotusti käsin, kirjoittamalla niitä sellaisenaan koodiin.

Huomion arvoinen seikka on, että useimmissa tapauksissa eri sovelluksen osien toteutus ohjelmointikielellä tai tavalla ei ole merkitystä (Choi, M. 2020). Tämän seurauksena web-sovelluskehitys maailmassa on lukematon määrä erilaisia ”stäkkejä” (engl. stack), termillä tarkoitetaan koko sovelluksen luontiin käytettyjä sovelluskehyksiä ja merkittävimpiä kirjastoja. Esimerkiksi MEAN-stackin kirjaimet tarkoittavat Mongo Express Angular Node. Tässä tapauksessa tietokantana on MongoDB, backendin web-palvelin on toteutettu Node.Js:llä ja Expressillä, lopuksi frontend on totetettu Angularilla. Tällaisilla nimillä tai lyhenteillä ei ole virallista statusta, vaan ne ovat vakiintuneet yhteisön käytössä.



Kuvio 1. Web-sovelluksen rakenne (Mukaillen: Dabbs 2019).

Palataan vielä sovelluksen osien väliseen liikenteeseen. Sovelluksen frontendin ja backendin välisen kommunikaation toteuttamiseen on kourallinen yleisesti käytettyjä vaihtoehtoja. Yksinkertaisin ja yleisin tapa on käyttää Rest APIa. Restin etuina ovat sen helppokäyttöisyys ja joustavuus, useimmissa tapauksissa Restin käyttö ei vaadi erillistä kirjastoa. Rest-rajapinnat voivat käsitellä monen tyyppisiä pyyntöjä, ja niiden palauttama tieto voi olla juuri kehittäjän tarvitsemassa muodossa. Nykyään yleinen muoto tiedonsiirtoon on JSON/XML-formaatti. Olennaista on, että tietomuoto valitaan käyttötarpeen mukana sopivaksi. Restin rajoitteita on, ettei se sovellu reaaliaikaiseen tiedonsiirtoon tai jatkuvaan tiedonsiirtoon, ns. striimaamiseen. (Kamali 2020.)

Toinen vaihtoehto on Facebookin 2015 julkaisema GraphQL. Sen tavoitteena on minimoida siirrettävän tiedon määrää, mahdollistamalla halutun tiedon kyselyn ilman ylimääräisen tiedon toimittamista. GraphQL on rakennettu Rest APIn päälle, joten Restin rajoitteet vaikuttavat myös tässä tapauksessa. GraphQL sovelluksen backend ei tarvitse kuin yhden rajapinnan, jonka nimi usein on graphql esimerkiksi: esimerkkiappi.com/qraphql. Sovelluksen frontend voi pyytää kyseisestä rajapinnasta kaiken tarvitsemansa tiedon käyttäen GraphQL kyselyä, tässä kyselyssä määritellään mitä

tietoja halutaan ja vastauksena backend palauttaa juuri nämä pyydetyt tiedot. GraphQL:n etu Restiin nähden on sen kyselyjen muokattavuus frontendissä ilman että tarvitsee muokata vastaavaa rajapintaa backendissä. Tällä joustavuudella on hintansa, monimutkaisten kyselyiden käsittely voi olla hyvinkin resursseja vaativaa ja aikaa vievää, joka voi johtaa pidempiin latausaikoihin. (Kamali 2020.)

Mikäli sovellus tarvitsee reaaliaikaista tiedonsiirtoa, WebSocket voi olla sopiva ratkaisu. WebSocket on kommunikaatio protokolla, joka tarjoaa täysin kaksisuuntaisia (engl. full-dublex) kanavia TCP-yhteyden yli. WebSocket-protokolla on OSI-mallin seitsemännellä tasolla muun muassa HTTP:n kanssa. Ne ovat eriset protokollat, mutta WebSocket on suunniteltu toimimaan yhdessä HTTP:n kanssa. WebSocketin yhteyden muodostuksessa käytetäänkin HTTP:tä. (Fette & Melnikov 2011.) WebSocket mahdollistaa käytännössä reaaliaikaisen tiedonsiirron sovelluksen osien välillä. WebSocketin suurimmat heikkoudet ovat luotettavuuden illuusio ja websocket-palvelimien skaalautuvuushaasteet. WebSocket-yhteyksissä viestien katoaminen yhteysongelmien vuoksi on mahdollista ja tämä pitää ottaa huomioon sovelluksen ohjelmoinnissa. Skaalautuvuushaasteet johtuvat asiakassovelluksen tarpeesta avata ja ylläpitää yhteyttä yhteen tiettyyn palvelimeen, tähänkin ongelmaan on kehitetty ratkaisuja. HTTP-protokollan kehittymisen myötä WebSocket alkaa vanhenemaan, eikä sitä enää tueta HTTP/2.0 protokollassa. (Kamali, A. 2020.) Tulevaisuuden tiedonsiirto protokollana tällaiseen tarpeeseen Kamali (2020) esittelee WebTransportin. Tällä hetkellä WebTransport on vielä kehitysvaiheessa, selainten tuki sille vaihtelee ja protokollan tulevaisuus on vielä jokseenkin auki (Konik 2022).

Sovelluksen tiedonsiirtoon on myös muita vaihtoehtoja, mutta opinnäytetyön kohderyhmälle edellä esitellyt ovat tärkeimmät. Niitä käytetään ja opetetaan Tikolla ja ne ovat laajasti käytettyjä kukin omalla vahvuusalueellaan. Seuraavaksi perehdytään hieman syvemmin frontendin ja backendin toimintaan, niiden vastuu alueisiin sekä erilaisiin toteutustapoihin.

3.2 Frontend - käyttöliittymä

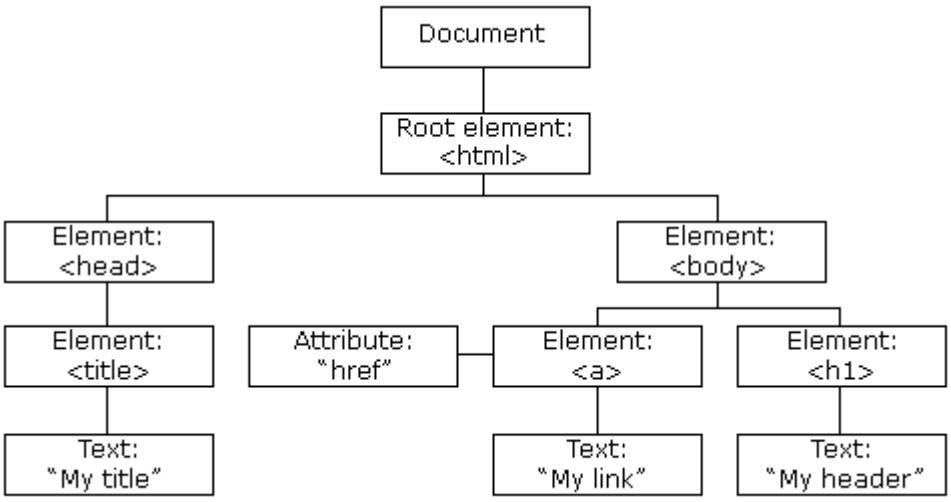
Web-sovelluksen frontend, joskus tästä käytetään myös nimeä client tai käyttöliittymä, vastaa käyttöliittymästä, tiedon pyytämisestä palvelimelta ja saamansa tiedon esittämisestä käyttäjäystävällisessä muodossa. Frontend on usein itsenäinen sovellus ja web-sovelluksista puhuttaessa sen toteuttamiseen käytetään usein jotakin JavaScript-kirjastoa tai -sovelluskehystä (engl. framework).

Yleisiä ja paljon käytettyjä ovat muun muassa React ja Angular. Sekä Facebookin, nykyään Meta, kehittämä React, että Googlen kehittämä Angular ovat erityisesti sovelluksen käyttöliittymän kehittämiseen tarkoitettuja kirjastoja.

Web-sovelluksen käyttöliittymä tarjoillaan käyttäjälle joko selaimen kautta web-sivuna tai erillisenä sovelluksena, jonka käyttäjä lataa esimerkiksi älypuhelimen sovelluskaupasta. Molemmissa tapauksissa käyttöliittymän tietopyynnöt palvelimelle toimivat samoja teknologioita hyödyntäen, suurin ero on kehitystyössä käytetty ohjelmointikieli tai kirjasto. Esimerkiksi Reactia käytetään web-sivun käyttöliittymän tekemiseen. React Nativelle puolestaan voi tehdä älypuhelimen mobiiliapplikaation.

Keskitytään erityisesti tällaiseen web-sivuna toimivaan sovellukseen. Käytännön esimerkkinä web-sovelluksen käyttöliittymän toiminnasta otetaan jollakin JavaScript-kirjastolla toteutettu sovellus, johon käyttäjä voi tallentaa muistiinpanoja ja nähdä aiemmin tallentamansa muistiinpanot. Kun käyttäjä avaa sovelluksen sivun, hän saa palvelimelta käyttöliittymä sovelluksen, jonka selain esittää käyttäjälle. Latauduttuaan käyttöliittymä lähettää palvelimelle pyynnön jo olemassa olevista muistiinpanoista, palvelin vastaa pyyntöön joko pyydetyllä tiedolla tai virheilmoituksella. Saatuaan tiedon käyttöliittymä esittää sen käyttäjälle kehittäjän ohjelmoimalla tavalla. Yksinkertaistettuna frontend reagoi käyttäjän toimiin ja kommunikoi backendin kanssa lähettäen ja vastaanottaen tietoa, sekä esittää tämän tiedon sovelluksen käyttäjälle. Seuraavaksi selvitetäänkin miten selain esittää web-sivun käyttäjälle.

Kun selain on ladannut web-sivun, sivusta luodaan DOM. DOM on lyhenne sanoista Document Object Model, suomeksi dokumenttioliomalli. DOMissa HTML-tiedosto on esitetty puuna, jonka olioita voi hakea ja manipuloida esimerkiksi Javascriptillä. Kuviossa 2. on tällainen puu yksinkertaisesta HTML-sivusta. Kuviossa laatikot kuvaavat edellä mainittuja olioita, joiden ominaisuuksia voi ohjelmallisesti manipuloida, esimerkiksi muuttaa tekstiä, muotoilua tai vaikka piilottaa koko olio. (Swain 2020.)



Kuvio 2. HTML DOM puu (JavaScript HTML DOM n.d.)

Nykyään monet nettisivut ja web-sovellukset on toteutettu yksisivuisina sovelluksina (engl. Single Page Application), usein käytetään lyhennettä SPA. Yksisivuinen sovellus on nimensä mukaisesti yksisivuinen ja sivusta päivitetään vain osia käyttäjän toimien seurauksena. Useimmilla sivuilla on paljon sisältöä, joka pysyy samana koko käytön aikana, esimerkiksi sivun logot, navigointi palkki, alareunan yhteystiedot ynnä muut tällaiset osat. SPA-sovellus hyödyntää tätä toistoa siten, että sivun osat ladataan ja päivitetään vain, kun sille on tarvetta. Käytännössä tämä näkyy sivulle tultaessa tavanomaisena suhteellisen pitkänä ensilatauksena, kun selain lataa HTML-sivun ja kaikki muut tiedostot, mutta tämän jälkeen koko sivua ei enää ladata uudelleen. Sen sijaan sovellus pyytää palvelimelta päivitettyjä tietoja tarpeen mukaan, jotka se sitten esittää päivittämällä sivun elementtejä, tässä kohtaa voidaan hyödyntää edellä mainittua DOMia. Tällainen toteutus tapa tekee käyttäjäkokemuksesta sulavamman ja parantaa sovelluksen suorituskykyä. (Lawson 2018.)

Web-sovelluksen komponentin määritelmä vaihtelee asiayhteydestä ja puhujasta riippuen. Koko sovelluksen komponentit olisivat käyttöliittymä, palvelin ja tietokanta. Frontendissä komponentti on yhdestä tai useammasta HMTL-elementistä koostuva kokonaisuus. Kuviossa 3. on esimerkki kortti-komponentista, sekä samainen komponentti vietynä React ympäristöön. Komponentit muunmuassa selkeyttävät sovelluksen rakennetta sekä helpottavat elementtien muotoilua. Niiden tärkein etu on uudelleenkäytettävyys: komponentti määritellään kerran, jonka jälkeen sitä voi käyttää uudelleen tarpeen mukaan. Komponentteja käytetään paljon esimerkiksi Reactissa. (Dimitrov 2021.)

```
HTML Card-komponentti
<div class="card">
  <h1 class="card-title"></h1>
  <p class="card-description"></p>
</div>
```

```
// React Card-komponentti
export default function Card() {
  return (
    <div class="card">
      <h1 class="card-title"></h1>
      <p class="card-description"></p>
    </div>
  );
}
```

Kuvio 3. Frontend card-komponentti (Muokattu: Dimitrov 2021).

3.3 Backend - palvelinohjelma

Web-sovelluksen backend kokonaisuus sisältää loput sovelluksesta ja toimii kuvainnollisesti sovelluksen aivoina. Lähes kaikilla sovelluksilla on tietokanta tai muu tapa tallentaa tietoa pidempiä ajanjaksoja. Backend ja sovelluksen tietokanta pyörivät palvelintietokoneella ja niitä käytetään APIn eli ohjelmistorajapinnan kautta. Web-sovelluksissa APIt usein noudattelevat REST arkkitehtuuri tyyliä, tosin sovelluksesta ja sen tarpeista riippuen frontendin ja backendin välinen kommunikointi voi olla toteutettu myös muilla tavoin, kuten aiemmin mainituilla GraphQL:llä tai WebSocketilla.

Usein myös sovelluksen tietokanta mielletään osaksi backendiä, tämä tosin voi vaihdella toteutus tavasta ja tilanteesta toiseen. Tietokanta voi olla samalla palvelimella kuin backend tai vaikka toisella palvelimella eri mantereella. Käytännössä backend lähettää HTTP-pyyntöjä tietokannalle samaan tapaan kuin käyttöliittymä backendille. Yksinkertaistettuna backend toimii tiedon välittäjänä ja käsittelijänä käyttöliittymän ja tietokannan välillä.

Käyttöliittymän esimerkkiä jatkaen backendin näkökulmasta, käyttäjän selain pyytää palvelimelta sivulle tultaessa tiedostoja ja koodia, jotta sivu voidaan esittää. Palvelin toimittaa määritellyt tiedostot selaimelle. Kun sivu on latautunut selaimessa, käyttöliittymän koodi lähettää backendille pyynnön saadakseen olemassa olevat muistiinpanot. Backend käsittelee pyynnön ja pyytää puolestaan tietokannalta jo olemassa olevia muistiinpanoja. Vastauksena tietokanta palauttaa pyydetyt tiedot, joita backend käsittelee halutulla tavalla ja toimittaa ne edelleen käyttöliittymälle. Tämän

jälkeen backend jää odottamaan uusia pyyntöjä käsiteltäväksi. Sama backend pystyy tarjoilemaan tietoa useille käyttöliittymä sovelluksille tai sivuille.

Edellä backend on kuvattu monoliittisena palvelinsovelluksena, mutta se ei ole ainoa backendin toteutustapa. Viime vuosikymmenen aikana suosiotaan ovat kasvattaneet erilaiset palvelimettomat (engl. serverless) ja mikropalvelu (engl. microservice) arkkitehtuurit. Palvelimeton arkkitehtuuri on tapa pyörittää sovellusta, mikropalvelu puolestaan on sovelluksen suunnittelutapa. Nämä voidaan yhdistää palvelimettomiksi mikropalveluiksi. Mikropalveluarkkitehtuuri käytännössä vaatii pilvipalvelu (engl. cloud service) ympäristön, jotta siitä saadaan paras hyöty. Palvelimettomat toteutukset edellyttävät pilvipalvelun, esimerkiksi AWS Lambdan, käyttöä. (Tozzi 2021.)

Mikropalveluarkkitehtuurissa sovellus on jaettu pienempiin osiin, joista kullakin on oma tehtävänsä: yksi osa vastaa käyttäjähallinnasta, toinen varsinaisen sivun päivityksestä, kolmas sivulla olevan hakutyökalun toiminnasta ja niin edelleen. Mikropalveluille ei ole mitään kiveen hakattua määritelmää, miten sovellus pitäisi jakaa osiin tai montako osaa pitäisi olla. Mikropalveluarkkitehtuurin etuja on sovelluksen skaalautuvuus, joustavuus ja osin sivutuotteena luotettavuus. Koska backend-sovelluksen eri osat pyörivät nyt itsenäisinä pienempinä palveluina niitä on helpompi lisätä tai vähentää tarpeen mukaan. Lisäksi jos jokin yksittäinen palvelu kaatuu syystä tai toisesta, sen tilalle voidaan nopeasti nostaa uusi vastaava palvelu. Silloinkin vain kyseisen palvelun ominaisuudet ovat epäkunnossa, muu sovellus toimii edelleen normaalisti. (Tozzi 2021.)

Palvelimettomassa arkkitehtuurissa sovelluksen koodia suoritetaan vain tarvittaessa, sovellus voi koostua kokonaan tai osin palvelimettomista funktioista. Tällaiset funktiot käynnistyvät ennalta määritellyn tapahtuman aikana, tekevät toimintonsa ja pysähtyvät kun ovat valmiit. Esimerkiksi käyttäjän sovellukseen lataaman kuvan pienennys tietyn kokoiseksi. Funktio käynnistyy käyttäjän ladatessa kuvan sovellukseen, se pienentää kuvan määriteltyyn kokoon ja lähettää lopputuloksen eteenpäin ja sammuu kun kaikki on valmista. Palvelimettoman mallin suurin etu on sen tehokkuus sellaisen koodin suorituksessa, jota ei tarvitse suorittaa jatkuvasti. Palvelimeton aika alkoi käytännössä, kun AWS (Amazon Web Services) toi Lambda-palvelun markkinoille vuonna 2014. Nykyään vastaavia palveluita on saatavilla kaikilta suurilta pilvipalveluiden tarjoajilta. (Tozzi 2021.)

Web-sovelluksen skaalautuvuudella tarkoitetaan sovelluksen kykyä mukautua aktiivisten käyttäjien määrän muutokseen. Hyvin skaalautuva sovellus toimii yhtä luotettavasti ja tehokkaasti oli käyttäjiä sitten yksi tai kymmeniä tuhansia. Sovellus voi skaalautua joko horisontaalisesti lisäämällä palvelimien määrää tai vertikaalisesti lisäämällä palvelimien suorituskykyä ja resursseja. Skaalautuvuuden toteutus käytännössä liittyy enemmän sovelluksen infrastruktuuriin ja sen suunnitteluun, mutta vaatii asian huomioimisen myös koodin suunnittelussa ja kirjoituksessa. Sovelluksen skaalaamiseen liittyvät olennaisesti kuormituksen tasaaminen (engl. load balancing), prosessi jossa sovelluksen palvelimille tulevaa liikennettä jaetaan ja tasataan automaattisesti eri palvelimien kesken. Pilvipalvelun kuormituksen tasain voi myös automaattisesti käynnistää tai sammuttaa palvelimia käyttäjä liikenteen määrän mukaan, tällöin sovelluksen käytössä on aina sopiva määrä resursseja. (Kryzhanovska & Sharapova 2021.)

4 Web-sovellusten tietoturvauhat

4.1 Tietoturvasta ja tietoturvauhista yleisesti

Web-sovellusten tietoturva on alati liikkeessä oleva kokonaisuus, jossa sovelluskehittäjät ja tietoturva-asiantuntijat pyrkivät suojaamaan sovelluksiaan pahantahtoisilta toimijoilta. Vastapuolella taasen hakkerit pyrkivät löytämään ja hyväksikäyttämään sovellusten haavoittuvuuksia saavuttaakseen jotain hyötyä itselleen tai vain aiheuttaakseen vahinkoa sovellukselle, sen käyttäjille ja omistajille.

Hoffman (2020) tarjoaa kirjassaan kolme eri osa aluetta web sovellusten tietoturvaan. Lyhyesti nämä ovat tiedustelu, hyökkäys ja puolustus. Tiedustelulla tarkoitetaan web-sovelluksen rakenteen tutkimista tavoitteena löytää haavoittuvuuksia sovelluksen rakenteesta ja rajapinnoista. Yleensä sovelluksen tiedustelua tekevät hakkerit ja erilaiset tietoturvatestaajat. Hyökkäyksen tarkoitus on hyväksikäyttää löydettyjä haavoittuvuuksia, joko hakkerin toimesta tavoitteenaan oma hyöty tai muu vahinko, tai tietoturvatestaajan toimesta tarkoituksena tutkia ja raportoida haavoittuvuuksia. Mainitut tietoturvatestaajat voivat olla esimerkiksi yrityksen palkkaamia testaajia tai bugi-palkkionmetsästäjiä. He raportoivat löydöksensä yritykselle, jotta viat voidaan korjata ennen kuin pahantahtoisin tarkoitusperin varustettu hakkeri hyödyntää niitä. (Hoffman 2020, Preface xxiii - xxv.)

Sovelluksen puolustusta voi pitää haasteellisempana kuin sitä vastaan hyökkäystä. Puolustuksen tulisi toimia aina ja kaikkia hyökkäyksiä vastaan, mutta hyökkääjän ei tarvitse onnistua kuin kerran. Hoffman (2020) uskoo, että hakkerien käyttämien keinojen tunteminen ja ymmärtäminen on tärkeä tietotaito sovelluksen tietoturvan parantamisessa. Kun ymmärtää hyökkääjän toimintatapoja, sovelluksen rakenteen ja logiikan piilottaminen on helpompaa ja tehokkaampaa (Hoffman 2020, 187). Tämän voi yleistää ajatukseen, että sovelluksen tehokas turvaaminen hyökkäyksiä vastaan edellyttää ymmärrystä hyökkääjän toimintatavoista.

Hoffmanin (2022) mukaan valtaosa sovellusten tietoturvatyöstä on varmistaa tiedon turvallinen kulku sovelluksenosien välillä sekä pitää huolta että tieto on turvassa myös lähtö- ja päätepisteissä. Tämä tulee ottaa huomioon jo sovelluksen arkkitehtuurin suunnittelu vaiheessa, koska sovelluksen arkkitehtuurista kumpuavat haavoittuvuuden ovat erittäin haasteellisia ja työläitä paikata jälkeenpäin. (Hoffman 2022, 188)

Erilaisia listauksia yleisimmistä web-sovellusten tietoturvauhista löytyy runsaasti ja sisällöltään ne ovat hyvin samanlaisia. Van der Stockin, Glasin, Smithlinen ja Giglerin (2021) laatima OWASP Top 10:2021 -raportti antaa hyvän kuvan yleisimmistä haavoittuvuuksista vuonna 2021. Selenius (2021) puolestaan ottaa käytännönläheisemmän otteen web-sovelluksen turvallisuuden tarkistuslistassaan. Van der Stock ja muut avaavat tietoturvauhkien taustatekijöitä tilastoimansa tiedon valossa ja antavat lukuja haavoittuvuuksien yleisyydestä. Selenius hyödyntää tätä raporttia omassa blogissaan tarjoten kattavan käytännöllisen listan konkreettisista toimenpiteistä, jotka sovelluskehittäjän tulisi tehdä turvatakseen sovelluksensa. (Van der Stock ym. 2021; Selenius 2021.)

Tässä esiteltyjen listojen pohjalta tutkitaan syvällisemmin tietoturvauhkia. Uhkien tutkimisessa hyödynnetään erityisesti edellä mainittua OWASP Top 10 -listaa, jossa eri tietoturvauhat on jo jaettu loogisiin kategorioihin ja josta myös selviää näiden uhkatyyppien yleisyys kyseisen tutkimuksen tutkimusmateriaalissa. Seuraavissa luvuissa koostetaan tietoturvauhista löydettyä tietoa helpommin käsiteltäviksi kokonaisuuksiksi, joita voidaan hyödyntää kehitettävän opintojakson oppimateriaalina. Jokaisen kokonaisuuden osalta tutkitaan uhkaa yleisesti, sen taustasyitä sekä miten kyseiseen uhkaan tulisi varautua.

4.2 Tiedon salauksen puutteet

Van der Stockin ja muiden (2021) mukaan erilaiset salaukseen liittyvät virheet tai puutteet olivat toiseksi yleisen tietoturva haavoittuvuuksien lähde vuonna 2021. Salauksen puutteet altistavat tiedon erilaisille hyökkäyksille sekä tietoa siirrettäessä että levossa, esimerkiksi tietokannassa. Salaukseen tulee panostaa erityisesti, kun käsitellään arkaluonteisia tietoja ja tällaisia tietoja ei pitäisi tallentaa tarpeettomasti. Paras tapa suojella arkaluontoista tietoa onkin olla keräämättä ja tallentamatta sitä, mikäli se ei ole aivan välttämätöntä (Freeman 2022). Seuraavaksi perehdytään erilaisiin salaukseen liittyviin uhkiin syvällisemmin.

Sekä Selenius (2021) että Van der Stock ja muut (2021) ovat yhtä mieltä siitä, että kaikki käyttöliittymän ja palvelimen välinen liikenne täytyy salata. Salaamaton liikenne on altis erilaisille ”mies välissä-hyökkäyksille” (engl. man-in-the-middle attack), joissa hyökkääjä tarkkailee tietoliikennettä esimerkiksi julkisessa langattomassa verkossa etsien hyödynnettäviä tietoja (Chivers 2020). Ei riitä, että vain arkaluonteisten osien liikenne on salattu. Hyökkääjälle voi riittää yksi salaamaton HTTP-pyyntö, jota hyödyntää hyökkäyksessään. Salauksen tärkeydestä kertoo Googlen Chrome-selainta koskeva päätös merkitä sivusto turvattomaksi, jos se ladataan salaamatonta HTTP-protokollaa käyttäen (Schechter 2018). Liikenteen salaamiseen suositellaan käytettäväksi HTTPS-protokollaa. Omalle sovellukselle tai nettisivulle liikenteen salauksen ja HTTPS-sertifikaatin saa helposti ja ilmaiseksi esimerkiksi LetsEncryptillä (Selenius 2021).

Salaukseen käytettäviä avaimia, salaisuuksia, salasanoja ynnä muita ei koskaan tule tallentaa koodiin eikä julkiseen versionhallintaan. Yksi hyvä tapa näiden erittäin arkaluonteisten muuttujien arvojen säilyttämiseen on käyttää ympäristömuuttujia, tällöin niitä ei voi helposti selvittää tutkimalla sovelluksen koodia. (Security 2022.)

Sovelluksen tietojen salaamisessa on tärkeää, että salaukseen käytettävä työkalu on kyseisen tiedon salaukseen soveltuva ja riittävä. Työkalun tulee siis olla ajantasainen ja salausalgoritmin kyllin vahva, jottei sitä voi murtaa helposti. Van der Stock ja muut (2021) mainitsevat vanhentuneista salaustavoista esimerkkeinä MD5:n ja SHA1:n. Salaustyökalua ei myöskään kannata ruveta tekemään itse, pienten virheiden riski on olemassa ja niillä voi olla valtavat vaikutukset. On parempi käyttää jo kehitettyjä ja testattuja työkaluja. Mitä laajemmassa käytössä työkalu on, ja mitä vähemmän sitä tarvitsee konfiguroida, sen helpompi se on saada toimimaan oikein ja luotettavasti.

(Freeman 2022.) Van der Stockin ja muiden (2021) mukaan, esimerkiksi salasanan salaamiseen sopivia työkaluja ovat muun muassa Argon2, PBKDF2 ja bcrypt. Sopivin salauskirjasto tai -funktio riippuu sovelluksenkehitykseen käytetyistä teknologioista ja tilannekohtaisesta salaustarpeesta. Sovellusta suunniteltaessa onkin tärkeää selvittää salaustarpeet ja etsiä sovelluksen tarpeisiin sopiva salaustyökalu.

4.3 Käyttöoikeuksien hallinta

OWASPin top 10 -listan kärjessä vuonna 2021 olivat erilaiset kulunhallintaan ja käyttöoikeuksiin liittyvät haavoittuvuudet, edellisessä vastaavassa listauksessa nämä haavoittuvuudet olivat viidennellä sijalla. Tämä hieman kankeahko termi kattaa haavoittuvuudet, jotka mahdollistavat käyttäjän saavan haltuunsa jotain tietoa, johon hänellä ei pitäisi olla pääsyä. Myös haavoittuvuudet, jotka mahdollistavat käyttöoikeuksien kiertämisen tai manipuloinnin kuuluvat tähän aihepiiriin. (Van der Stock ym. 2021.)

Käyttöoikeuksien hallinnan puutteet voivat altistaa sovelluksen erilaisille haitoille. Hyökkääjä voi saada haltuunsa arkaluontoista tietoa, muokata tai jopa hävittää sitä, aiheuttaen näin vakavaa haittaa sovellukselle, sen omistajalle ja käyttäjille. Usein osana tällaista hyökkäystä hyökkääjä pyrkii hankkimaan mahdollisimman paljon käyttäjätilejä ja oikeuksia, joilla edistää hyökkäystä, varastaa käyttäjätietoja tai muutoin vahingoittaa sovellusta. (Sengupta 2021.)

Sengupta (2021) jakaa aiheen haavoittuvuudet pysty- ja vaakasuuntaan vaikuttaviin haavoittuvuuksiin. Pystysuunnan (engl. vertical) haavoittuvuus voi mahdollistaa pääsyn esimerkiksi erilaisiin järjestelmänvalvojien työkaluihin, joihin tavallisella käyttäjällä ei pitäisi olla pääsyä. Vaakasuunnan (engl. horizontal) haavoittuvuus puolestaan voi mahdollistaa pääsyn muiden käyttäjien tietoihin. Usein hyökkääjä yhdistää erilaisia haavoittuvuuksia ja hyökkäyssuuntia tavoitteensa saavuttamiseksi. Esimerkiksi vaakasuunnan haavoittuvuus mahdollistaa hyökkääjän saavan haltuunsa järjestelmänvalvojan tilin, joka puolestaan antaa huomattavasti laajemmat työkalut hyökkäyksen toteutukseen. (Sengupta 2021.)

Ensimmäinen askel käyttöoikeuksien hallinnan uhkien torjuntaan on kieltää oletuksena (engl. Deny by Default) pääsy sovelluksen resursseihin. Tätä voidaan pitää nyrkkisääntönä käyttöoikeuksien

hallinnassa. Tietenkään kaikkea ei voi estää tai kieltää, julkiseksi tarkoitettu sisältö on edelleen pidettävä sellaisena. Käytännössä tämä tarkoittaa, että käyttäjille tulisi antaa vain sen verran oikeuksia kuin on pakko. (Sengupta 2021.) Tähän liittyen myös CORS:n käyttö kannattaa pitää mahdollisimman vähänä, mutta jos sovelluksen toiminta edellyttää CORS:n käyttöä tulee sen kanssa olla varovainen (Selenius 2021).

Mahdollisten hyökkäysten havaitsemisessa voi hyödyntää käyttöoikeuksien virheilmoituksia ja niiden lokiin kirjaamista. Toistuvista rike yrityksistä tulisi varoittaa järjestelmänvalvojaa. Esimerkiksi jos käyttäjä toistuvasti yrittää tehdä jotain, johon hänellä ei ole oikeuksia, tai käyttäjä yrittää tehdä jotakin mitä sovelluksen käyttöliittymästä ei edes voi tehdä, voi kyseessä olla hyökkäys sovellusta vastaan. Järjestelmänvalvojan tulisi olla tietoinen tilanteesta. Tietyissä tilanteissa APIn kutsutahdinrajoittaminen (engl. rate limit) voi olla hyvä työkalu erilaisten automaattisten hyökkäystyökalujen hidastamiseksi tai pysäyttämiseksi, yhden käyttäjän ei tarvitse pystyä yrittää kirjautumista tuhansia kertoja minuutissa. (Van der Stock ym. 2021.)

Käyttöoikeuksien hallintaan liittyy olennaisesti istunnot (engl. session) ja niiden hallinta. Istunnon toteutukseen on sovelluspalvelimella kaksi tapaa, tilallinen (engl. stateful) tai tilaton (engl. stateless). Tilallisessa arkkitehtuurissa palvelin tietää käyttöliittymän tilan ja ylläpitää istunnon tilaa pyyntöjen yhteydessä. Tilaton palvelin ei tiedä käyttöliittymän tilaa, joten jokaisen pyynnön yhteydessä on välitettävä olennaiset palvelimen tarvitsemat istunnon tiedot. Sovelluksen kannalta paras tila-arkkitehtuuri riippuu kulloisesta tilanteesta, tosin tilattomien palvelimien suosio kasvanut johtuen ainakin osin arkkitehtuurin helposta skaalautuvuudesta. (Shah 2019.)

Tilallinen palvelin luo istunnon alussa pienen tiedoston, jossa on muun muassa uniikki istunnontunniste, kirjautumis- ja vanhenemisaika, sekä muuta tietoa. Tämä tiedosto voidaan säilyttää esimerkiksi selaimen evästeessä (engl. cookie), josta se lähetetään palvelimelle osana pyyntöä. Palvelin vertaa pyynnön istuntotunnistetta muistissaan olevaan tunnisteeseen ja vastaa pyyntöön. Ulos kirjautumisen yhteydessä istuntotunniste tuhotaan sekä käyttöliittymän evästeistä että palvelimen muistista. (Session vs Token Authentication 2021.)

Tilaton palvelin tunnistaa käyttäjän kirjautumisen yhteydessä luodulla tokenilla (engl. token). Tokeni on kryptografisesti allekirjoitettu ja toimii väliaikaisena käyttäjän tunnistusvälineenä. Web-

sovelluksissa tähän tarkoitukseen käytetään paljon esimerkiksi JSON Web Tokenia, lyhennettynä JWT. Allekirjoitettu JWT on kolmesta JSON olioista koostuva merkkijono. JWTn otsikko (engl. header) oliossa määritellään väitteitä (engl. claim), esimerkiksi käytettävä algoritmi sekä tokenin tyyppi. Niin sanotussa hyötykuorma (engl. payload) oliossa on varsinainen kuljetettava tieto määriteltynä väitteinä kuten vanhenemisaika, myöntämisaika, myöntäjä, kohde. Tokenin kolmas olio sisältää allekirjoituksen. (Peyrott 2018.)

JSON Web Tokeneita hyödyntävissä hyökkäyksissä tokenia muokataan, esimerkiksi antamalla käyttäjälle tokeniin järjestelmänvalvojan oikeudet. Seuraavaksi koitetaan saada palvelin hyväksymään manipuloitu tokeni. Hyökkääjä voi myös koettaa käyttää tokenia sellaisenaan saadakseen pääsyn saman organisaation toiseen palveluun, joka käyttää samaa tunnistuspalvelinta. Hyökkääjän tokeni voi väittää käyttävänsä eri salausalgoritmia kuin mitä sovellus alun perin käytti, tai tokeni väittää käyttäjällä olevan laajemmat käyttöoikeudet kuin käyttäjällä on palvelimella myönnetty. Tokeneita käyttäessä onkin tärkeää vahvistaa niiden oikeellisuus palvelimella ennen kuin niillä tehdään mitään. Tämä koskee tokenin kaikkia väitteitä ja myös tokeneita tokeneiden sisällä. Palvelimen ei koskaan tule sokeasti luottaa käyttöliittymän väitteisiin. (Peyrott 2018.)

Tietoturvan kannalta on olennaista, että käyttäjän istunnon tunnistamiseen käytettävä väline on vain väliaikainen ja vain istunnon mittainen, oli se sitten istunnon tunniste, JWT tai muu tokeni (Van der Stock ym. 2021). Voi tuntua käytännölliseltä sovellusta tehdessä, että käyttäjänistunto on voimassa esimerkiksi kuukauden, mutta tämä tarkoittaa, että tunnisteen vääriin käsiin päätyessä hyökkääjällä on samainen aikaikkuna hyödynnettävänään. Tässä luvussa jo sivuttiin erilaisia käyttäjätyyppejä ja muitakin käyttäjienhallinnointiin liittyviä asioita, joten jatketaan seuraavassa luvussa käyttäjän kirjautumiseen, tunnistamiseen ja todentamiseen liittyvien uhkien tutkimista.

4.4 Käyttäjän tunnistus ja todennus

Käyttäjän tunnistaminen (engl. identification) ja todentaminen (engl. authentication) ovat olennainen osa mitä tahansa sovellusta, jossa käyttäjä voi luoda oman tilin. Van der Stockin ja muiden (2021) suurin osa tunnistamisen ja todentamisen uhista liittyy käyttäjän kirjautumisprosessiin ja ennen kaikkea salasanoihin. Muita mahdollisia aihepiirin haavoittuvuuksia ovat salasanan tai tilin palautuksen heikkoudet, monivaiheisen todennuksen heikkous tai kokonaan puuttuminen, sekä istunnon toteutuksen haavoittuvuudet.

Kirjautumisprosessia voidaan pitää heikkona, jos se mahdollistaa erilaiset automaattiset hyökkäykset, joilla hyökkääjä pyrkii saamaan pääsyn käyttäjän tilille. Yksi yleinen kirjautumiseen kohdistuva hyökkäys on kirjautumistietojen täyttöhyökkäys (engl. credential stuffing). Tällaisessa hyökkäyksessä hyökkääjä on saanut käsiinsä listan käyttäjänimiä ja salasanoja esimerkiksi jonkin tietomurron seurauksena. Hyökkääjä käyttää automaattista työkalua kirjautuakseen kohde sovellukseen koettaen jokaista listalla olevaa käyttäjätunnus ja salasana -paria, jos kirjautuminen onnistuu hyökkääjällä, on kyseisen käyttäjän kirjautumistiedot kohde sovellukseen ja mahdollisesti myös muihin palveluihin. Tämän jälkeen hyökkääjä voi varastaa käyttäjän tiedot, käyttää tiliä omiin tarkoituksiinsä ja kokeilla samoja kirjautumistietoja muihin palveluihin. (Mueller n.d.)

Toinen kirjautumisprosessiin itseensä kohdistuva hyökkäys on raakavoimainen kirjautumishyökkäys (engl. brute force login attack). Tämän perinteisessä versiossa hyökkääjä tietää vain käyttäjän käyttäjätunnuksen ja käytännössä arvailee salasanaa automaattisella työkalulla. Hyökkäyksessä usein käytetään listaa yleisistä salasanoista. Myös edellä mainittu täyttöhyökkäys on versio tästä hyökkäyksestä. Kolmas versio on password spraying -hyökkäys, siinä hyökkääjä kokeilee muutamia yleisiä salasanoja kaikille tietämilleen käyttäjätunnuksille. (Johnson 2021.)

Kirjautumisprosessin hyökkäyksiä vastaan käytetyimmät keinot ovat erilaiset lukitukset tietyn kirjautumisyritysmäärän jälkeen. Myös kirjautumisyritysten tahdin rajoittaminen esimerkiksi CAPTCHA-lomakkeella, joka vaatii ihmisen väliintuloa hidastaen hyökkäystä merkittävästi. Lukitus voi kohdistua IP-osoitteeseen, käyttäjätunnukseen tai molempiin eri yritysmäärien jälkeen. Yleensä lukitus kannattaa tehdä määräaikaiseksi esimerkiksi tunniksi, jotta tilin oikealle käyttäjälle ei aiheudu liikaa vaivaa. (Johnson 2021.)

Onnistuneen kirjautumishyökkäyksen vaikutuksien minimointiin on myös hyviä työkaluja, joista ehkä tärkein on toimiva kaksivaiheinen tunnistautuminen, joka vähentää tietomurrossa vuodettujen kirjautumistietojen käytettävyyttä merkittävästi. Muita huomion arvoisia toimenpiteitä on vahvan salasanan vaatiminen, käyttäjätunnusten turvassa pitäminen ja kirjautumisliikenteen valvonta mahdollisten hyökkäysten varalta. (Johnson 2021.)

Salasanojen osalta kaikkein tärkeimmät asiat ovat vahvan salasanan vaatiminen käyttäjältä sekä salasanan säilytys oikein sovelluksen tietovarastossa. Tavanomaisten salasanan pituus ja merkkivaatimusten lisäksi salasanan testaaminen huonojen salasanojen listaa vastaan on hyvä idea. Salasanojen säilytystä käsiteltiin jo luvussa 4.2., mutta mainittakoon vielä, että käyttäjien salasanat tulee säilyttää tiivistemuodossa (engl. hashed password). Tällöin niitä ei voi nähdä eikä myöskään suoraan purkaa. Tiivistäminen (engl. hashing) on yksisuuntainen toiminto, sen tulos voidaan toistaa samoilla lähtötiedoilla, mutta tuloksesta ei voi päätellä tai purkaa lähtötietoja. Salasana tulee myös ns. suolata (engl. salting) ennen sen tiivistämistä. Useimmat modernit algoritmit tekevät suolauksen automaattisesti, esimerkkinä näistä Argon2, brycpt ja PBKDF2, joita OWASP suosittelee käyttämään. (Van der Stock ym. 2021.)

Tilin tai unohtuneen salasanan palautusominaisuuden toteutuksen yleisin haavoittuvuus on, erilaiset palvelimen palauttamat vastaukset riippuen onko käyttäjätunnus olemassa vai ei. Tämän avulla hyökkääjä voi selvittää olemassa olevia käyttäjätunnuksia. Suositeltavaa onkin, että palvelin palauttaa saman vastauksen kaikissa tilanteissa, oli käyttäjätunnus olemassa tai ei. Toinen heikkous on mahdolliset turvakysymykset, joihin on mahdollista joko arvata oikea vastaus tai muutoin selvittää se. Tämän vuoksi OWASP suositteleekin, ettei turvakysymyksiä käytettäisi ainoana tunnistamisvälineenä salasanan palautuksessa. (Van der Stock ym. 2021.)

Mikäli sovellus käyttää tilallista palvelinta tulee onnistuneen kirjautumisen jälkeen luotavan istuntotunnisteen olla uusi ja satunnainen. Istuntotunnisteen paikka ei ole URL:ssä, se tulee säilyttää turvallisesti ja ennen kaikkea tunniste pitää mitätöidä uloskirjautumisen yhteydessä, tai käyttäjän ollessa toimeton tietyn aikaa. (Van der Stock ym. 2021.)

4.5 Injektiohyökkäykset

Injektiohyökkäykset ovat yleisnimitys hyökkäyksille, joissa hyökkääjä syöttää sovellukselle JavaScript-koodia, tietokantakyselyitä, käyttöjärjestelmän komentoja tai muuta sellaista, tavoitteena saada sovellus tekemään asioita, joita sen ei pitäisi tehdä. Yhteistä erilaisilla injektiohyökkäyksille on, että niissä käyttäjän syöttämä teksti käytetään sovelluksessa ilman että sitä tarkistetaan, suodatetaan tai muuten puhdistetaan. Onnistuneen hyökkäyksen seuraukset ovat yleensä data varkauksia tai menetyksiä, palvelunestoja tai pahimmillaan jopa koko palvelimen menetys hyökkääjän käsiin. (Van der Stock ym. 2021.)

Useimmat web-sovellukset käyttävät tietokantaa ja monesti tietokanta on SQL-pohjainen. Tällöin sovellus voi olla joutua SQL-injektiohyökkäyksen kohteeksi. SQL-tietokannan käyttö itsessään ei tee sovelluksesta haavoittuvaksi injektioille, se vain mahdollistaa tämän tietyn hyökkäystyypin yrittämisen. Yksinkertaisimmillaan SQL-injektiohyökkäyksessä hyökkääjä kirjoittaa sovelluksen tekstikenttään kokonaisen tai osan SQL-kyselyä ja lähettää sen eteenpäin. Mikäli sovelluksessa on tällaisen hyökkäyksen mahdollistava haavoittuvuus, lähetetty kysely välitetään sellaisenaan tietokannalle, jossa se injektoidusta kyselystä riippuen, varastaa, muuttaa tai poistaa tietokantaan tallennettua tietoa. (Banach 2020.)

NoSQL-injektio on samantyylinen hyökkäys kuin SQL-injektio, erona sovelluksen käyttämä tietokantajärjestelmä. Nimensä mukaisesti noSQL-injektiot kohdistuvat noSQL-tietokantoihin. NoSQL-tietokannoilla ei ole yhtä yhteistä kyselykieltä ja hyökkäyksessä käytettävän kielen onkin oltava sama kuin kohdesovelluksen käyttämän tietokannan kyselykieli. Esimerkkeinä noSQL-tietokannoista mainittakoon MongoDB, Redis ja Google Cloud Datastore, näistä jokaisella on oma kyselykieli. Yhteistä monilla noSQL-tietokannoilla on kyselyjen pohjautuminen JSONiin ja että niissä voi olla käyttäjän syöttämää tekstiä. Jos käyttäjän syötettä ei tarkasteta ja puhdisteta, sovellus on haavoittuvainen injektio hyökkäyksille. (Nidecki 2020a.)

Käyttöjärjestelmäkomentoinjektiohyökkäyksessä (engl. OS command injection) hyökkääjä pyrkii sovelluksen haavoittuvuuden avulla antamaan komentoja isäntäkäyttöjärjestelmälle (engl. host operating system). Komentoinjektiot ovat mahdollisia, jos sovellus antaa turvatonta käyttäjän syöttämää tietoa järjestelmän komentotulkille (engl. system shell). (Zhong 2021.)

XSS- tai Cross-site Scripting-hyökkäys hyödyntää samannimistä haavoittuvuutta suorittaakseen haitallista koodia uhrin selaimessa. Edellisistä injektio hyökkäyksistä poiketen XSS-hyökkäyksen kohde on nimenomaan toinen käyttäjä, hyökkääjä käyttää sovellusta haitallisen koodin levitykseen varsinaisille kohteille. XSS-haavoittuvuus mahdollistaa hyökkääjän antaman syötteen esittämisen toisille käyttäjille sellaisenaan puhdistamatta sitä. XSS-hyökkäyksissä käytetään usein JavaScript-kieltä sen yleisyyden ja laajan selaintuen vuoksi, muita käytettyjä kieliä ovat VBScript, ActiveX ja CCS. Onnistunut JavaScript XSS-hyökkäys voi kohteen selaimessa muun muassa varastaa evästeitä, muokata DOMia, lähettää HTTP-pyyntöjä valitsemiinsa kohteisiin ja käyttää HTML5 APIa. Näistä viimeisin voi mahdollistaa pääsyn käyttäjän sijaintiin, webkameraan, mikrofoniin ja jopa joihinkin

käyttäjän tiedostoihin. Tosin suurin osa näistä vaatii käyttäjän erillisen hyväksynnän. (Cross-site Scripting (XSS) n.d.)

Tyypillinen XSS-hyökkäys on kaksivaiheinen. Ensin hyökkääjän täytyy injektoida koodi sovellukseen, jonka jälkeen uhrin täytyy vierailla sivulla, jolle haitallinen koodi latautuu. Esimerkiksi jos sovellus esittää viimeisimmän sovellukseen kirjoitetun tekstin kaikille käyttäjille ja esitettävä teksti vain asetetaan HTML-tunnisteiden sisään sivua luotaessa, voi hyökkääjä muotoilla tekstinsä toimimaan koodina kyseisten tunnisteiden sisällä kun sivu ladataan. Teksti voi olla esimerkiksi JavaScript-koodin pätkä, joka kerää sivun ladanneen käyttäjän tietoja ja lähettää ne hyökkääjälle. (Cross-site Scripting (XSS) n.d.)

Injektiohyökkäysten torjunnassa on muutamia tehokkaita toimia, joista ehkä tärkein on käyttäjän syötteen tarkastus ja puhdistus palvelimella, käyttäjiltä tulevan tekstin turvallisuuteen ei pidä koskaan luottaa. Käytännössä tämä voi tarkoittaa tiettyjen, tietokannan käyttämien erikoismerkkien kieltämisen käyttäjiltä tulevassa tekstissä. Tämä ei aina ole toivottava ratkaisu koska osa tietokannan erikoismerkeistä voi olla tavanomaisia kirjoitetussa kielessä. (Li 2021.)

Toinen tehokas injektiotorjuntakeino on APIn parametrisointi (engl. parameterization). Sen sijaan, että käyttäjän syöte ketjutetaan (engl. concatenating) tietokantakyselyyn, kysely rakennetaan etukäteen ja käyttäjän syöttämät tekstit lisätään kyselyyn parametreinä. Tällöin kyselyn logiikka ja data osat pysyvät erillään. Kolmantena keinona on käyttäjältä tulevien erikoismerkkien käsittely koodinvaihtomerkillä (engl. escape character), jotta niitä käsitellään datana eikä erikoismerkkeinä. Kaikissa mainituissa torjuntakeinoissa on omat heikkoutensa ja haasteensa sovellusta toteutettaessa. Tehokkaimman injektiotorjunnan toteutus riippuukin hyvin paljon käytetystä ohjelmointikielestä ja tietokannasta, sekä siitä millaista tekstiä käyttäjän tarvitsee syöttää. (Li 2021.)

4.6 SSRF – Server-side request forgery

SSRF-haavoittuvuus antaa hyökkääjän lähettää itse tekemiään pyyntöjä sovelluksen backend-palvelimelta. SSRF-hyökkäys kohdistuu usein kohteen sisäisiin järjestelmiin, jotka on suojattu palomuurilla ulkopuoliselta liikenteeltä. SSRF-haavoittuvuus johtuu yleensä ominaisuudesta, jossa

käyttäjä antaa URL-osoitteen tai osan siitä ja sovelluspalvelin tekee pyynnön kyseiseen osoitteeseen. Tällöin hyökkääjä voi tehdä kutsuja käytännössä mihin tahansa osoitteeseen, mahdollisesti myös palvelimen sisäisiin resursseihin, joihin ei ulkopuolelta pääse käsiksi. (Muscat 2022.)

SSRF-haavoittuvuuden paikkaamiseen ei ole yhtä oikeaa tapaa ja paras ratkaisu riippuukin paljon sovelluksen toiminnasta ja tarpeista. Joissain tapauksissa mustalista lähestymistapa voi sopia sovelluksen tarpeisiin, jolloin tiettyihin osoitteisiin ei voi lähettää pyyntöjä mutta tällainen lista on melko helppo kiertää. Huomattavasti tehokkaampi keino on tehdä lista sallituista osoitteista, niin sanottu valkoinen lista (engl. whitelist), joihin sovellus voi pyyntöjä lähettää. Ja mitä tarkemmin sallitut osoitteet määritellään, sen tehokkaampi ratkaisu on. (Muscat 2022.)

Useimmat sovellukset käyttävät http:tä tai https:ää kyselyjen tekoon, joten on suositeltavaa, että vain kyseisen malliset URL:t hyväksytään. Tällöin hyökkääjä ei voi käyttää muita URL-malleja kuten file:// tai dict://. Tärkeää on myös poistaa käytöstä HTTP-uudelleenohjaus, jota hyökkääjä voisi käyttää mustan listan kiertämiseen. Tärkeä asia on lähetetyn kyselyn vastauksen käsittely. Käyttäjälle voi oikeanlaisen vastuksen tuloksista lähettää tarvittavat tiedot. Missään tapauksessa saatua vastausta ei saa välittää sellaisenaan käyttäjälle, etenkään jos vastaus ei ole odotetun lainen. (Muscat 2022.)

4.7 Palvelunestohyökkäykset

Palvelunestohyökkäys (engl. Denial of Service, DOS) pyrkii pysäyttämään palvelun toiminnan, tai ainakin merkittävästi hidastamaan sitä, ylikuormittamalla joko palvelimen fyysiset resurssit tai verkkoyhteydet. Käytännössä palvelimelle tulee niin paljon liikennettä tai käsiteltävää dataa, että peruskäyttäjä ei pysty käyttämään palvelua. Usein hyökkäyksen liikenne tulee ulkopuolelta mutta joissakin tapauksissa jokin sovelluksen haavoittuvuus voi mahdollistaa sovelluksen ylikuormittamisen sisältäpäin. Jotkut haittaohjelmat mahdollistavat tartutettujen laitteiden käytön palvelunestohyökkäyksissä. Kun useaa laitetta tai bottiverkkoa käytetään hyökkäykseen, kutsutaan sitä hajautetuksi palvelunestohyökkäykseksi (engl. Distributed Denial of Service, DDoS). (Denial of service (DOS) n.d.)

Palvelunestohyökkäysten torjuminen täysin on käytännössä mahdotonta, mutta se ei tarkoita, ettei hyökkäyksen tekoa kannata tehdä mahdollisimman vaikeaksi. Sovelluksen suunnittelulla ja

toteutetulla koodilla on merkittävä vaikutus siihen, kuinka helppoa palvelunestohyökkäys on toteuttaa ja kuinka suuri vaikutus mahdollisella hyökkäyksellä on. Jos palvelimella olevien isojen tiedostojen latausta ei ole rajoitettu tai kuka tahansa voi tehdä sovelluspalvelimelle laskennallisesti raskaita pyyntöjä, ovat palvelunestot paljon yleisempiä ja helpompia toteuttaa. Sovelluspalvelimen suorituskyvyn optimoinnilla sekä paljon haettujen resurssien välimuistiin tallentamisella pääsee jo pitkälle, jotta tavallisesta poikkeava käyttäjäpiikki ei aiheuta vahingossa syntynyttä palvelunesto tilannetta. (Van der Stock ym. 2021.)

Varsinaisia palvelunestonhyökkäyksen torjuntakeinoja on hyökkäyspintojen minimointi, sovelluksen ja ympäristön haavoittuvuuksien paikkaus sekä erilaisten palvelunestohyökkäysten torjuntapalveluiden käyttö. Hyökkäyspintojen minimointi tarkoittaa turhien palveluiden ja rajapintojen sisäiseen verkkoon siirtämistä tai kokonaan poistamista. Mitä vähemmän potentiaalisia kohteita palvelimella on näkyvissä internettiin, sen vaikeampaa hyökkäys on toteuttaa. Kaikki tunnetut haavoittuvuudet kannattaa paikata mahdollisimman pian ja pitää sovellus ja sen suoritusympäristö ajan tasalla, jotta myös tulevaisuudessa löydettävät haavoittuvuudet tulee paikattua mahdollisimman pian. Markkinoilla on myös erilaisia palvelunestohyökkäysten torjuntapalveluita. Näitä todennäköisesti tarjoaa käytetty pilvipalvelu tai internet palveluntarjoaja. Nämäkään palvelut eivät täysin torju hyökkäystä, vaikka ne varmasti pienentävätkin hyökkäyksen vaikutuksia. Mitä tahansa ennaltaehkäisykeinoja palvelun turvaamiseen käytetäänkään, on tärkeää tehdä myös suunnitelma siitä mitä tehdään, kun bottiverkko on jo ovella. (Beaver 2017)

4.8 Vanhentuneet kirjastot ja päivitysten eheys

Web-sovelluksilla on yleensä lista riippuvuuksia (engl. dependency) jotka helpottavat ominaisuuksien toteuttamista sovelluksessa. Nämä ovat helppoja lisätä ja usein ominaisuutta ei kannata edes yrittää tehdä itse alusta, jos on olemassa oleva kirjasto sen toteuttamiseen. Mutta nämä kirjastot ja komponentit, jotka tekevät kehityksestä helppoa, voivat altistaa sovelluksen haavoittuvuuksille. Etenkin jos niitä ei päivitetä lainkaan tai päivitysväli on pitkä. Vanhentuneet tai haavoittuvat komponentit ja kirjastot eivät suoraan altista jollekin tietylle hyökkäystyypille, mutta niissä olevat haavoittuvuudet voivat altistaa sovelluksen hyökkäyksille, joita vastaan tavanomaiset toimet eivät auta. (Van der Stock ym. 2021.)

Tärkein toimenpide haavoittuvuuksien välttämiseksi tai ainakin paikkaamiseksi mahdollisimman pian, on pitää sovelluksen omat riippuvuudet sekä sen suoritusympäristö ajan tasalla. On suositeltavaa poistaa sovelluksesta sellaiset riippuvuudet, kirjastot, komponentit ja tiedostot, joita ei käytetä lainkaan, tämä siistii sovellusta, sen luettavuutta ja samalla vähentää haavoittuvuuksien riskiä. Suositeltavaa olisi myös seurata aktiivisesti sovelluksessa käytettyjen kirjastojen ja komponenttien turvallisuustiedotteita, jos käytetystä kirjastosta löytyy haavoittuvuus toimenpiteisiin kannattaa ryhtyä mahdollisimman pian. Yksi työkalu sovelluksen nykyisten riippuvuuksien tarkistamiseen tunnettujen haavoittuvuuksien varalta on OWASP Dependency Check. (Van der Stock ym. 2021.)

Edellä käsiteltiin ajantasaisten kirjastojen ja komponenttien tärkeyttä. Sovelluksen käyttämien kirjastojen hankinnassa ja päivittämisessä on tärkeää, että lähde on luotettava. Käytännössä on varmistettava, että käytettävä paketinhallinta työkalu, esimerkiksi npm tai Maven, käyttää luotettuja repositorioita. Näin varmistetaan sovelluksen käyttämien kirjastojen eheys. (Van der Stock ym. 2021.)

4.9 Konfiguraatiovirheet

Erilaiset sovelluksen osien konfiguroinnin virheet ovat yleistymään päin oleva haavoittuvuustyyppi. Tällaisia haavoittuvuuksia ovat tarpeettomat asennetut ja päälle jätetyt ominaisuudet, kuten ylimääräiset avoimet portit, oletuskäyttäjät ja -oikeudet. Oletuskäyttäjissä lisäriskin tuo mahdollisesti oletusarvoihin jätetyt salasanat. Todelliset sovelluksessa mahdollisesti olevat haavoittuvuudet riippuvat hyvin paljon siitä millainen komponentti sovellukseen on lisätty, mitä se käytännössä tekee ja kuinka hyvin tai huonosti sen konfigurointi on tehty. Konfiguraatiovirheenä voidaan myös pitää sovelluksen virheenhallinnan käyttäjälle palauttamia tarpeettoman laajoja virheilmoituksia, käyttäjän ei tarvitse tietää mikä koodirivi kyseisen virheen aiheutti. (Van der Stock ym. 2021.)

Olennainen osa hyvää konfiguraatiota on sovelluksen ja sen suoritusympäristön turvallisuuskarkaisu (engl. security hardening). Tämän prosessin tavoitteena on minimoida hyökkäyspinnat ja vähentää turvallisuus uhkien määrää. Suoritusympäristön osalta tehtäviä toimia on turhien sovellusten poistaminen, turhien käyttäjätilien poistaminen ja jäljelle jäävien oikeuksien vähentämien vain tarpeellisiin sekä vahvojen salasanojen vaatiminen. Verkkoyhteyksistä sammutetaan ja poistetaan kaikki mitä sovellus ei tarvitse sekä toimeenpannaan vahvat palomuurisäännöt, vain tarpeelliset yhteydet sallitaan. Web-palvelimesta (engl. web server, esim. nginx ja apache) poistetaan kaikki

tarpeettomat osat, vaihdetaan oletusasetukset ja asennetaan tilannekohtaisesti tarvittavat sovellus palomuurit ynnä muut sellaiset. Lisäksi koko suoritusympäristöön kannattaa ottaa automaattiset päivitykset käyttöön. (Nidecki 2020b.)

Varsinaisen web-sovelluksen tietoturva karkaisu sisältää haavoittuvuus skannauksen sekä mahdollisimman kattavan tunkeutumistestauksen (engl. penetration testing). Myös itse sovelluksesta kannattaa siivota kaikki käyttämättömät palaset pois, seurauksena koodin luettavuus ja turvallisuus paranee. Edellä kuvattu karkaisuprosessi tulisi olla jatkuva, järjestelmälle tulisi tehdä säännöllisesti tarkistuksia pitäen huolta ajantasaisuudesta ja sekä muutoin tarkistaa järjestelmän kunto. Samaan aikaan tulee seurata muutoksia sovelluksen turvallisuus ympäristössä ja reagoida niihin tarpeen mukaa. (Nidecki 2020b.)

4.10 Arkkitehtuurin haavoittuvuudet

Sovelluksen arkkitehtuurista tai niin sanotusta turvattomasta designista (engl. insecure design) johtuvat haavoittuvuudet voivat altistaa sovelluksen monille eri hyökkäys tyypeille. OWASP erottaa designin ja toteutuksen (engl. implementation) virheet toisistaan, koska ne johtuvat eri juurisyistä, jolloin myös niiden korjaustoimenpiteet poikkeavat toisistaan. Ne ovat toisistaan riippumattomia, eikä toisen korjaaminen automaattisesti korjaa toista. Sovelluksen osien täydellinen toteutus ei korjaa arkkitehtuurin tai designin puutteita. Sovelluksen suunnitteluvaiheessa onkin hyvä pysähtyä tekemään riskiarvio sovelluksen toiminnoista ja siihen mahdollisesti kohdistuvista uhista, sekä päivittää ja tarkistaa riskiarvioita kehityksen edetessä. (Van der Stock ym. 2021.)

Esimerkkinä designin haavoittuvuudesta voidaan ottaa sovellukseen kirjautumisen yhteydessä käyttäjälle esitettävät virheviestit. Jos sovellus ilmoittaa, ettei kyseistä käyttäjätiliä ole olemassa tai että salasana ei täsmää annettuun tiliin, voi hyökkääjä etsiä olemassa olevia käyttäjätilejä ja hyödyntää näitä tietoja käyttäjän kalastelussa (engl. phishing) tai suoremmassa kirjautumishyökkäyksessä. Parempi käytäntö kirjautumisen virheisiin on antaa käyttäjälle sama ilmoitus, riippumatta siitä oliko käyttäjän nimi vai salasana väärin. Myös ennen niin yleiset turvakysymykset tilin tai salasanan palauttamiseksi ovat nykyään heikko tunnistus vaihtoehto, kyseinen tapahan ei vahvista muuta kuin tietääkö vastaaja kyseisen kysymyksen oikean vastauksen. Nykypäivän sosiaalisessa mediassa ihmiset jakavat paljon informaatiota itsestään, jolloin hyökkääjä voi selvittää turvakysymysten vastauksia vain tutkimalla käyttäjän julkaisuja. (Boothe 2021.)

Van der Stock ja muut (2021) esittävät hyvänä käytänteenä turvallisen designin (engl. secure design) kulttuurin. Tämän sovelluskehitys kulttuurin ydin on sitoa tietoturva olennaiseksi osaksi sovelluksen suunnittelua ja kehitystä. Käytännössä tämä tarkoittaa tietoturva sanaston ja käytänteiden ottamisen osaksi käyttäjä tarinoita (engl. user stories) ja erillisten hyökkääjä tarinoiden luontia, uhkien mallintamista sovelluksen osissa sekä näiden mallien integrointi sovelluksen testeihin. Turvallisen designin kulttuuriin kuuluu myös olennaisena osana virheistä oppiminen ja positiiviset kannustimet tietoturvan parantamiseksi. (Van der Stock ym. 2021.)

4.11 Valvonta ja lokitus

Sovelluksen toiminnan valvonta ja lokitus ovat tärkeitä työkaluja sovelluksen tietoturvan kannalta. Ne eivät varsinaisesti suojaa sovellusta haavoittuvuuksilta tai hyökkäyksiltä, sen sijaan ne mahdollistavat hyökkäysten ja muun epäilyttävän toiminnan havaitsemisen. Valvonnan käytännön toteutuksesta riippuen mahdollisista hyökkäyksistä jää ainakin jälki sovelluksen lokeihin ja mahdollinen hälytys lähetetään eteenpäin, jotta järjestelmänvalvoja saa tiedon mahdollisimman pian. Huonoin mahdollinen tilanne on, jos minkäänlaista valvontaa tai tietoturvalokitusta ei ole tehty sovellukseen. Tällöin hyökkäyksiä tai tietomurtoja voi olla käytännössä mahdotonta havaita tai jälkeenpäin tutkia. (Van der Stock ym. 2021.)

Valvonnan merkittävimmät heikkoudet painottuvat puutteelliseen lokitukseen tapahtumissa, joissa käyttäjän antamaa tietoa validoidaan, kuten kirjautumistilanteessa ja erityisesti epäonnistuneessa kirjautumisessa. Toinen lokituksen heikkous voi olla puutteelliset lokit sovelluksen virheistä ja varoituksista. Sovelluksen lokien muoto ja luotettavuus vaikuttavat siihen, kuinka hyödyllistä ja käytettävää niiden sisältämä tieto on. Lokien hallintaan on erilaisia ratkaisuja ja kirjotetun lokin on oltava yhteensopiva käytetyn hallintatyökalun kanssa. Lokit itsessään voivat myös olla hyökkäyksen kohteena. Hyökkääjä voi pyrkiä peittelemään jälkiään tai etsiä lokeista sovelluksen mahdollisia haavoittuvuuksia. (Van der Stock ym. 2021.)

Lokitietojen keräämisessä pitää myös ottaa huomioon joitakin lainsäädäntöön liittyviä seikkoja. Esimerkiksi jos lokiin kirjataan henkilötietoja, tulee lokista henkilörekisteri ja siihen pätevät samaiset lait ja asetukset kuin muihinkin henkilörekistereihin (Näin keräät ja käytät lokitietoja 2020).

Sovelluksen valvonnan ja lokituksen hyvinä käytänteinä voidaankin pitää seuraavia asioita. Kaikkien käyttäjätileihin liittyvien validointivirheiden lokitus riittävien tunnistustietojen kanssa, jotta epäilyttävät tai pahantahtoiset käyttäjätilit voidaan tunnistaa. Sovelluksen lokit ovat oikeassa muodossa ja ne on riittävästi suojattu injektioita tai muita hyökkäyksiä vastaan. Sovelluksella on sopivat valvonta- ja hälytystyökalut, joilla epäilyttävät toimet havaitaan ajoissa ja niihin voidaan reagoida. Ja ennen kaikkea on laadittu suunnitelma toimenpiteistä, joihin ryhdytään, kun jotain tapahtuu. (Van der Stock ym. 2021.)

5 Opintojakson sisältösuunnitelma

5.1 Opintojakson sisällön yleiskuvaus

Web-sovellusten tietoturvaopintojakso pyrkii laajentamaan opiskelijan yleistietämystä erilaisista tietoturvauhista ja -haavoittuvuuksista. Opintojakson tavoitteena on, että opiskelija tuntee merkittävimmät uhat, pystyy arvioimaan uhkien vakavuutta ja vaikutuksia omissa tulevissa projekteissa, opiskelija tuntee yleistasolla tietoturvauhkien ehkäisyn ja torjunnan keinoja. Opintojakson esivaatimuksina opiskelijan tulee tuntea perusteet web-ohjelmointiin, tietokantoihin, frontendiin ja backendiin. Esivaatimukset voi täyttää suorittamalla niitä vastaavat opintojaksot tai muutoin omatoimisesti hankkia tarvittavat tietotaidot vaatimusten täyttämiseksi.

Opintojakson sisältö on jaettu neljään osaan tai moduuliin, ja jokaisen osan sisällä käsitellään saman aihepiirin tietoturvauhkia. Kunkin moduulin osalta tavoite on, että opiskelija tuntee aihepiirin merkittävimmät haavoittuvuudet sekä parhaita käytänteitä näiden haavoittuvuuksien ehkäisemiseksi. Ensimmäisessä osassa tarkastellaan web-sovelluksen toimintaa ja tietoturvaa yleistasolla. Osan tarkoituksena on luoda yleiskuva web-sovelluksen rakenteesta, yleisesti käytetyistä arkkitehtuuriratkaisuista sekä katsoa tietoturvauhkia yleisesti koko sovelluksen näkökulmasta. Toisessa osassa perehdytään erityisesti sovelluksen käyttäjienhallintaan ja tiedonsalaamiseen. Kolmannessa kokonaisuudessa käsitellään erilaisia teknisiä uhkia ja hyökkäyksiä, joiden kohteeksi sovellus voi joutua. Opintojakson neljännen osan aiheena on sovelluksen designin ja suoritusympäristön haavoittuvuudet.

Edellisen kappaleen kokonaisuuksia voi käyttää opintojaksojen kontaktikertojen aiheina sellaisenaan tai opintojakson lopullisesta laajuudesta riippuen kokonaisuuksia voi myös jakaa useammalle

kontaktikerralla samalla syventyen aiheisiin tarkemmin. Myös osallistujien tarpeet, mielenkiinnonkohteet ja ajankohtaiset muutokset tietoturvaympäristössä kannattaa ottaa huomioon opintojakson kulussa. Ajankohtaisten, ja miksei myös vanhempien, hyökkäysten taustasyiden, vaikutusten ja ratkaisujen tutkiminen tarjoaa opiskelijoille konkreettisen esimerkin tietoturvan tärkeydestä sekä hyökkäysten vaikutuksista reaalimaailmassa.

Opintojakson aikana tehtävissä harjoitustehtävissä ja erilaisien hyökkäysten demonstroinnissa käytännössä voi hyödyntää OWASPin Juice Shop -demosovellusta, jota mainostetaan moderneimpana ja hienostuneimpana turvattomana sovelluksena. Sovelluksesta löytyy valtava määrä erilaisia haavoittuvuuksia mukaan lukien kaikki OWASPin Top 10 -listan haavoittuvuudet. (OWASP Juice Shop n.d.)

5.2 Web-sovellus ja tietoturvasta yleisesti

Opintojakson ensimmäisen moduulin aiheena on web-sovellusten rakenne ja toiminta sekä yleiskatsaus erilaisiin tietoturvauhkiin. Osan lukumateriaalina toimii opinnäytetyön luku 3, jossa käsitellään web-sovelluksen rakennetta ja toimintaa, sekä luku 4.1, joka on johdanto tietoturvauhkiin. Tietoturvauhkien osalta toinen hyvä lukumateriaali on Van der Stockin ja muiden (2021) laatiman OWASP Top 10 2021 -raportin johdanto ja sisällysluettelo.

Harjoitustehtävänä tässä moduulissa opiskelija etsii yhden itseään kiinnostavan jo tapahtuneen web-sovellukseen kohdistuneen kyberhyökkäyksen ja tutkii hyökkäystä löytämiensä lähteiden avulla. Opiskelija selvittää hyökkäyksestä ainakin kohteen, ajankohdan sekä tekijöitä, jotka mahdollistivat hyökkäyksen. Jos hyökkäys aiheutti muutoksia kohdesovelluksessa tai -organisaatiossa, mitä tehtiin tai muutettiin? Opiskelija esittelee lyhyesti löydöksensä muille seuraavalla kontaktikerralla.

5.3 Salaus ja käyttäjähallinta

Toisen moduulin aiheina on tiedon salaus sekä käyttäjienhallinta. Tiedonsalauksen lukumateriaalina on opinnäytetyön luku 4.2, sekä OWASPin Top 10 -listan kohta A02 salausvirheet (engl. cryptographic failures). Käyttäjähallinnan yhteydessä käsitellään sekä käyttäjän tunnistusta ja varmennusta sekä käyttöoikeuksienhallintaa. Materiaalina opinnäytetyön luvut 4.3 ja 4.4, sekä OWASP

listan kohdat A01 ja A07, jotka käsittelevät niin sanottua rikkinäistä kulunvalvontaa sekä tunnistuksen- ja todennuksenvirheitä.

Salauksen harjoitustehtävänä opiskelija selvittää miten seuraavia tietoja tulisi säilyttää sovelluksessa: käyttäjän salasana, käyttäjänimi ja sähköpostiosoite. Opiskelija etsii esimerkkejä sopivista salaustyökaluista ja perustelee, miksi kyseinen työkalu on sopiva esimerkiksi salasanan salaukseen. Opiskelija perehtyy hyviin salauskäytänteisiin ja laatii itselleen listan parhaista käytänteistä.

Käyttäjäntunnistuksen tehtävänä opiskelija laatii kuvitteelliselle sovellukselle salasana- ja kirjautumiskäytännöt ja -säännöt. Sääntöjen tulee noudattaa nykyisiä parhaita käytänteitä ja ne tulee perustella. Mallivastauksessa olisi määriteltynä ainakin salasanan merkki- ja pituusvaatimukset, heikkojen salasanojen tunnistus ja kielto, sekä joitakin keinoja automaattisten kirjautumishyökkäysten ehkäisyyn.

5.4 Tekniset uhat

Kolmannen moduulin aiheena on erilaiset web-sovellukseen kohdistuvat tekniset uhat kuten injektiot, SSRF-hyökkäykset ja erilaiset palvelunestot. Lukumateriaalina toimii opinnäytetyön luvut 4.5–4.7, sekä OWASP listan kohdat A03 ja A10, jotka käsittelevät injektioita ja SSRF-hyökkäyksiä. Palvelunestohyökkäysten torjunnasta luettavaa tarjoaa muun muassa Beaver (2017) laatimassaan vinkki artikkelissa.

Harjoitustehtävänä opiskelija selvittää miten injektio hyökkäyksiä voi torjua jossakin tietyssä sovellusympäristössä, esimerkiksi Mongo Express React Node -kirjastoista koostuvassa sovelluksessa. Suositeltavaa on, että opiskelija valitsee jonkin aiemmin käyttämänsä ympäristön. Tehtävän tavoitteena on, että opiskelija ymmärtää miten käyttäjän syötteen validointi ja injektiohyökkäysten torjunta tulisi käytännössä tehdä valitussa ympäristössä.

Toisena harjoituksena opiskelijat, joko pienissä ryhmissä tai yksin, laativat suunnitelman (engl. incident response plan) moduulissa käsiteltyjen teknisten uhkien varalle. Sovellus, jolle suunnitelma tehdään voi olla opiskelijoiden keksimä sovellus tai jokin jo olemassa oleva julkinen sovellus. Sovel-

luksen toiminnallisuudet tulee määritellä kyllin tarkasti. Käytännössä laaditaan toimintasuunnitelma hyökkäyksen jälkeisiin toimenpiteisiin, verkosta löytyvien mallien, esimerkkien ja muiden ohjeiden hyödyntäminen on erittäin suositeltavaa.

5.5 Sovelluksen design ja suoritusympäristö

Opintojakson neljännen moduulin aiheena ovat erilaiset sovelluksen designin, rakenteen ja suoritusympäristön haavoittuvuudet ja uhat. Lukumateriaalina opinnäytetyön luvut 4.8–4.11 sekä OWASP listalta osat: A04-A06, A08 ja A09, joissa käsitellään designin heikkouksia, konfiguraatiovirheitä, komponenttien ja kirjastojen vanhenemista, sovelluksen eheyttä sekä lokituksen ja valvonnan vikoja. Sovellus päivitysten eheyteen liittyen lisälukemisena kannattaa tutustua 2020 tapahtuneeseen SolarWinds -kyberhyökkäykseen.

Harjoitustehtävänä opiskelija tutustuu erilaisiin kirjastojen tai riippuvuuksien skannaustyökaluihin (engl. dependency scan) ja kokeilee ainakin yhtä jossakin omassa projektissa, esimerkiksi toisen opintojakson aikana tehty sovellus. Onko sovelluksen kirjastoissa tunnettuja haavoittuvuuksia?

Laajempana harjoituksena opiskelija tutustuu OWASPin Juice shop -sovellukseen, joka on täynnä haavoittuvuuksia. Sovelluksessa on hakkerointi haasteita, joilla voi kokeilla erilaisten hyökkäysten toteutusta. Opiskelija tekee ainakin kolme valitsemaansa haastetta ja miettii mitä sovelluksessa pitäisi muuttaa tai korjata, jotta hyökkäyksesi ei onnistuisi. (OWASP Juice Shop n.d.)

6 Pohdinta

Opinnäytetyön päätavoite oli luoda sisältösuunnitelma ja sisältöä toimeksiantajan kehitteillä olevalle web-sovellustentietoturvaa käsittelevälle opintojaksolle. Tämän tavoitteen saavuttamiseksi täytyi ensin tutkia web-sovellusten tietoturvaympäristöä ja erilaisia tietoturvauhkia, jotka kohdistuvat web-sovelluksiin. Tutkittavien uhkien ja niistä koottavan tiedon rajaamiseksi toimeksiantajan kanssa muodostettiin tutkimuskysymykset, joiden avulla tutkimuskohteita ja niistä kerättävää tietoa rajattiin tarpeeseen sopivaksi. Tutkimus toteutettiin perehtymällä olemassa olevaan aiheesta tehtyyn kirjallisuuteen ja muuhun sovelluskehittäjä- ja tietoturvayhteisöjen julkaisemaan materi-

aaliin. Tähän materiaaliin viitaten ja sitä referoiden koottiin tutkittavista tietoturvauhista tietopaketteja, joita voidaan hyödyntää kehitettävän opintojakson materiaalina ja joihin laaditussa sisältösuunnitelmassa viitataan.

Opinnäytetyön alusta asti oli tavoitteena koota ja esittää työn tulokset siten, että niitä voi hyödyntää toimeksiantajan myöhemmin kehittämän opintojakson oppimateriaalina sellaisenaan tai vain pienin muutoksin. Sekä päätavoite että tämä tulosten esityksen tavoite saavutettiin. Myös tutkimuskysymyksiin löydettiin vastaukset kaikille tutkituille tietoturvauhille. Parissa tapauksessa löydettyjen vastausten kattavuus tosin jätti toivomisen varaa. SSRF-hyökkäyksien osalta aineiston löytymisessä oli haasteita, ja palvelunestohyökkäyksien torjuntaan ei ole kovin tehokkaita torjuntakeinoja, joten päädyttiin esittämään keinoja negatiivisten vaikutusten minimointiin. Koko opinnäytetyön kirjoituksen ajan haasteena oli lähteissä käytetyn termistön kääntäminen suomeksi, erityisesti tapauksissa, joissa termeille ei ole virallisia suomenkielisiä termejä. Tämän vuoksi työssä käytettiin paikoin englanninkielisiä termejä tai lyhenteitä. Jos käytetylle termille löytyi järkevä suomennos, sen yhteydessä mainittiin myös termi englanniksi. Näin termien väärinymmärrys riski saatiin mahdollisimman pieneksi.

Tutkimustyön tulosten ja koottujen tietopakettien luotettavuus oli olennainen osa koko työn luotettavuutta. Työn luonteen vuoksi tuloksia voidaan pitää yhtä luotettavina kuin käytettyjä lähteitä. Merkittävin työssä käytetty lähde, Van der Stockin ja muiden (2021) OWASPille kokoama top 10 -listamainen raportti, jossa esitettyihin tuloksiin opinnäytetyön tietoturvauhkien rajaus perustuu, on yleisesti luotettavana pidetty ja siihen on myös viitattu laajalti. Sen on julkaissut tunnettu, tietoturva standardeja kehittävä voittoa tavoittelematon järjestö OWASP.

Muiden työssä käytettyjen internetlähteiden, kuten blogien ja artikkelien luotettavuutta on arvioitu tapauskohtaisesti niitä valittaessa, pääasiassa tarkastelemalla saman asiasisällön toistuvuutta muissa samaa aihetta käsittelevissä lähteissä. Tämä vahvistaa valitun lähteen luotettavuutta. Monet käytetyistä lähteistä ovat jonkin tietoturvapalveluita tarjoavan yrityksen julkaisemia, jolloin niiden puolueettomuuteen ei voi luottaa, joissakin suorastaan mainostettiin yrityksen palveluita aiheena olevan haavoittuvuuden paikkaamiseksi. Tämä ei suoranaisesti heikennä lähteen tarjoaman tiedon luotettavuutta tietoturvauhan tai haavoittuvuuden perustietojen osalta, mutta voi vaikuttaa lähteessä esiteltäviin ratkaisuvaihtoehtoihin. Tämän vuoksi tietoturvauhkien ehkäisykeinoja

tarkasteltaessa painotettiin OWASPin esittämiä hyviä käytänteitä, joita tarpeen mukaan tutkittiin tarkemmin muita lähteitä hyödyntäen.

Opinnäytetyössä käytettiin vain julkisesti saatavilla olevaa kirjallista materiaalia, jota voi kutsua myös työn tutkimusaineistoksi. Koska tutkittu aineisto on julkista, sen käsittelyssä ei ollut tarpeen huomioida muuta kuin tekijänoikeudet ja aineiston luotettavuus. Merkittävin eettinen kysymys liittyykin työn aiheeseen ja tuloksiin. Voiko työn tuloksia käyttää väärin ja hyödyntää esimerkiksi kyberhyökkäyksen toteuttamisessa? Osa opinnäytetyön tavoitteesta oli koota ohjeita ja vinkkejä, kuinka web-sovellusta voi suojata erilaisia tietoturvauhkia vastaan. Jotta sovellusta voi suojata tehokkaasti on ensin tiedettävä minkälaisia uhkia ja mahdollisia hyökkäyksiä sovellukseen voi kohdistua. Konkreettisien hyökkäystyyppien toimintaa avattaessa tuloksissa niitä käsiteltiin melko yleisellä tasolla. Sen sijaan että olisi lukijalle kerrottu yksityiskohtaisesti miten kyseinen hyökkäys voidaan toteuttaa, keskityttiin niihin hyökkäyksen vaiheisiin, jotka ovat olennaisimmat sen torjunnan kannalta. Tosin sanoen, opinnäytetyössä kyllä käsitellään kyberhyökkäysten toimintaa, mutta ei anneta ohjeita niiden toteuttamiseen.

Ilmeinen jatkokehitys suunta on tuloksia ja työssä laadittua sisältösuunnitelmaa hyödyntäen tietoturva opintojakson kehitys. Tässä tapauksessa jatkokehitys edellyttää ainakin sovitusta varsinaisen opintojakson raameihin, opintojakson sisältöjen tarkempaa suunnittelua sisältäen oppimateriaalin laatimista tai täydennystä sekä harjoitusten, tehtävien ja arviointikriteerien luontia.

Lähteet

Alexander, J. 2021. Risk, Threat, or Vulnerability? How to Tell the Difference. Kenna Security blogi. Julkaistu 28.1.2021. Viitattu 12.11.2021. https://www.kennasecurity.com/blog/risk-vs-threat-vs-vulnerability/.

Banach, Z. 2020. Top 5 Most Dangerous Injection Attacks. Invicti.com web turvallisuus blogi. Viitattu 18.4.2022. https://www.invicti.com/blog/web-security/top-dangerous-injection-attacks/

Beaver, K. 2017. Preventing DoS attacks: The best ways to defend the enterprise. Viitattu 18.4.2022. https://www.techtarget.com/searchsecurity/tip/Preventing-DoS-attacks-The-best-ways-to-defend-the-enterprise

Boothe, H. 2021. Bridges fall down due to insecure design – make sure your web applications don't. Viitattu 1.5.2022. https://www.securityjourney.com/post/bridges-fall-down-due-to-insecure-design-make-sure-your-web-applications-dont

Chivers, K. 2020. What is a man-in-the-middle attack? Norton tietoturva artikkeli. Julkaistu 26.3.2020. Viitattu 20.1.2022. https://us.norton.com/internetsecurity-wifi-what-is-a-man-in-the-middle-attack.html

Choi, M. 2020. HTTP request and response and how web applications work. Artikkeli. Julkaistu 28.4.2020. Viitattu 6.3.2022. https://codesensei.medium.com/http-request-and-response-and-how-web-applications-work-76780d4cb14c

Cross-site Scripting (XSS). N.d. Acunetix yrityksen artikkeli. Viitattu 18.4.2022. https://www.acunetix.com/websitesecurity/cross-site-scripting/

Dabbs, M. 2019. The Fundamentals of Web Application Architecture. Reinvently blogi. Julkaistu 29.7.2019. Viitattu 6.3.2022. https://reinvently.com/blog/fundamentals-web-application-architecture/

Denial of service (DOS). N.d. F-Secure artikkeli. Viitattu 18.4.2022. https://www.f-secure.com/v-descs/articles/denial-of-service.shtml

Dimitrov, A. 2021. What are Components in the front-end and why do we need them? Viitattu 17.4.2022. https://dev.to/xavortm/what-are-components-in-the-front-end-and-why-do-we-need-them-2o2p

Fette, I. & Melnikov, A. 2011. The WebSocket Protocol. IETF standardin ehdotus. Viitattu 13.3.2022. https://datatracker.ietf.org/doc/html/rfc6455

Freeman, C. 2022. Five Cryptography best practices for developers. Synopsys blogi. Viitattu 14.3.2022. https://www.synopsys.com/blogs/software-security/cryptography-best-practices/

Hoffman, A. 2020. Web Application Security. E-kirja. Kolmas painos. USA, CA, Sebastopol: O'Reilly Media, Inc. Viitattu 8.12.2021. https://www.nginx.com/resources/library/web-application-security/

JavaScript HTML DOM. N.d. W3Schools JS HTML DOM tutoriaali. Viitattu 13.3.2022 https://www.w3schools.com/js/js_htmldom.asp

Johnson, J. 2021. Protectiong Your Web Application From Brute-Force Login Attacks. Viitattu 17.4.2022. https://predatech.co.uk/protecting-your-web-app-brute-force-login-attacks/

Kamali, A. 2020. gRPC, Restful API, GraphQL, Web Socket, TCP Sockets and UDP, WebTransport — Beyond modern client server communications. Artikkeli. Viitattu 8.3.2022. https://tech-lead.medium.com/grpc-vs-restful-api-vs-graphql-web-socket-tcp-sockets-and-udp-beyond-client-server-43338eb02e37

Kananen, J. 2017. Laadullinen tutkimus pro graduna ja opinnäytetyönä. Jyväskylä: Jyväskylän ammattikorkeakoulu

Konik, J. 2022. What is WebTransport and can it replace WebSockets? Ably blogi. Viitattu 13.3.2022. https://ably.com/blog/can-webtransport-replace-websockets

Kryzhanovska, A. & Sharapova, M. 2021. Building scalable web application for your project: best principles and practices. Gearhead yrityksen artikkeli. Julkaistu 17.11.2021. Viitattu 17.4.2022. https://gearheart.io/articles/how-build-scalable-web-applications/

Lawson, K. 2018. What are Single Page Applications and Why Do People Like Them So Much? Bloomreach blogi. Julkaistu 19.7.2018. Viitattu 13.3.2022. https://www.bloomreach.com/en/blog/2018/what-is-a-single-page-application

Li, V. 2021. API Security 101: Injection. Blogi. Viitattu 18.4.2022. https://blog.shiftleft.io/api-security-101-injection-a7feea1d4fd

Mueller, N. N.d. Credential stuffing. Viitattu 17.4.2022. https://owasp.org/www-community/attacks/Credential_stuffing

Muscat, I. 2022. What is server-side request forgery (SSRF)? Acunetix artikkeli. Viitattu 18.4.2022. https://www.acunetix.com/blog/articles/server-side-request-forgery-vulnerability/

Nidecki, T. 2020a. NoSQL Injections and How to Avoid Them. Acunetix blogi. Viitattu 18.4.2022. https://www.acunetix.com/blog/web-security-zone/nosql-injections/

Nidecki, T. 2020b. Web System Hardening in 5 Easy Steps. Acunetix blogi. Viitattu 18.4.2022. https://www.acunetix.com/blog/web-security-zone/web-system-hardening-5-easy-steps/

Näin keräät ja käytät lokitietoja. 2020. Kyberturvallisuuskeskuksen opas. Viitattu 1.5.2022. https://www.kyberturvallisuuskeskus.fi/fi/ajankohtaista/ohjeet-ja-oppaat/nain-keraat-ja-kaytat-lokitietoja

OWASP Juice Shop. N.d. OWASPin demo-sovelluksen etusivu. Viitattu 8.5.2022. https://owasp.org/www-project-juice-shop/

Peyrott, S. 2018. A Look at The Draft for JWT Best Current Practices. Auth0 blogi. Julkaistu 11.4.2018. Viitattu 10.4.2022. https://auth0.com/blog/a-look-at-the-latest-draft-for-jwt-bcp/

Schechter, E. 2018. A milestone for Chrome security: marking HTTP as “not secure”. Googlen blogi. Julkaistu 24.7.2018. Viitattu 20.1.2022. https://blog.google/products/chrome/milestone-chrome-security-marking-http-not-secure/

Security. 2022. React Native dokumentaation opas. Viitattu 20.1.2022. https://reactnative.dev/docs/security

Selenius, T. 2021. Web Application Security Checklist. AppSec Monkey blogi. Julkaistu 13.2.2021. Viitattu 4.1.2022. https://www.appsecmonkey.com/blog/web-application-security-checklist

Sengupta, S. 2021. Broken access control and how to prevent it. Crashtest security blog. Julkaistu 20.9.2021. Viitattu 7.4.2022. https://crashtest-security.com/broken-access-control-prevention

Session vs Token Authentication. 2021. Viitattu 17.4.2021. https://www.authgear.com/post/session-vs-token-authentication

Shah, H. 2019. Stateless vs Statefull – Which direction should you take? Julkaistu 17.10.2019. Viitattu 10.4.2022. https://dev.to/hsshah_/stateless-vs-stateful-which-direction-should-you-take-2c7o

Swain, A. 2020. HTML DOM in Depth. Artikkeli. Julkaistu 24.8.2020. Viitattu 13.3.2022. https://medium.com/the-ui-girl/html-dom-in-depth-ae24965d1920

Tozzi, C. 2021. Microservices vs. serverless architecture. Sumo logic blogi. Julkaistu 18.3.2021. Viitattu 13.3.2022. https://www.sumologic.com/blog/microservices-vs-serverless-architecture/

Van der Stock, A., Glas, B., Smithline, N. & Gigler, T. 2021. OWASP Top 10:2021. OWASP-tietoturvasäätiön artikkelikokonaisuus. Viitattu 1.5.2021. https://owasp.org/Top10/

Zhong, W. 2021. Command Injection. OWASP yhteisö artikkeli. Viitattu 18.4.2022. https://owasp.org/www-community/attacks/Command_Injection