Juho Salminen

Verkkosovelluksen haavoittuvuustestaus

Kyberturvallisuuden pro gradu -tutkielma

12. toukokuuta 2020

Jyväskylän yliopisto

Informaatioteknologian tiedekunta

Tekijä: Juho Salminen

Yhteystiedot: juho.v.j.salminen@student.jyu.fi

Ohjaaja: Timo Hämäläinen

Työn nimi: Verkkosovelluksen haavoittuvuustestaus

Title in English: Web Application Vulnerability Testing

Työ: Pro gradu -tutkielma

Opintosuunta: Kyberturvallisuus

Sivumäärä: 74+0

Tiivistelmä: Tutkimuksen aiheena oli verkkosovellusten haavoittuvuuksien testaaminen ja tietoturvallisuuden parantaminen. Aihe on ajankohtainen, koska verkkosovelluksia käytetään yhä enemmän päivittäisten asioiden hoitamiseen. Lisäksi myös yhteiskunnan tärkeät toiminnot digitalisoituvat, mikä voi lisätä niiden haavoittuvuutta. Tutkimuskysymyksenä on, mitä haavoittuvuuksia eräästä verkkosovelluksesta löytyy ja miten niitä voidaan havaita. Yleisesti verkkosovellusten haavoittuvuustestausta ja haavoittuvuuksien hyödyntämistä on tutkittu runsaasti. Tässä tutkimuksessa tarkoituksena oli kuvata tarkasti itse testausprosessin toteutus ja suorittaa verkkosovellukseen kattava tietoturvatestaus.

Tietoturvallisuudella tarkoitetaan tiedon suojaamista luottamuksellisuuden, eheyden ja saatavuuden näkökulmasta. Verkkosovellus on verkossa toimiva palvelu, kuten verkkopankki tai sosiaalisen median palvelu. Haavoittuvuudet ovat sovellusten teknisiä ominaisuuksia, jotka voivat ilmetä tietoturvallisuutta vaarantavina tekijöinä. Esimerkiksi ohjelmointivirhe voi mahdollistaa luottamuksellisten tietojen paljastumisen.

Tutkimus toteutettiin konstruktiivista tutkimusotetta käyttäen. Tarkoituksena oli koostaa raportti, josta ilmenee verkkosovelluksen haavoittuvuudet ja kehitysehdotukset. Teoreettiseksi viitekehykseksi kerättiin aineistoa kirjallisuuskatsauksella. Tutkimuksessa toteutettiin haavoittuvuustestaus eri työkaluilla määriteltyä testausprosessia käyttäen, joka käsitti tiedonhankinnan, haavoittuvuuksien skannaamisen automaattisilla työkaluilla, palvelunestohyök-

käyksen ja sovelluksen lähdekoodin analysoinnin. Testauksen perusteella löydettiin muutamia haavoittuvuuksia, joiden poistamiseksi annettiin suosituksia kehittäjille. Tuloksista voidaan päätellä, että sovellus on tietyiltä osin haavoittuva, mutta haavoittuvuudet ovat helposti korjattavissa.

Avainsanat: verkkosovellus, haavoittuvuus, tietoturva, testaus, OWASP

Abstract: The subject of the thesis was to test vulnerabilities in web applications and improve their information security. Web applications are used more and more in everyday life and digitalization can increase the number of vulnerabilities. The research question is what vulnerabilities there are in a web application and how they can be improved. In general, the topic of web application vulnerability testing and exploitation of vulnerabilities has been researched abundantly. The objective of this research was to describe in detail the testing process and perform testing to a web application.

Information security means protecting the confidentiality, integrity, and availability of information. A web application is a service provided on the Internet, such as an online bank or social media platform. Vulnerabilities are technical features of those applications that can danger the security of information. For example, a programming error can cause sensitive data exposure.

The research was conducted using a constructive research method. The aim was to construct a report with known vulnerabilities in the application and provide means on how to improve them. A literature review was used to build a theoretical framework. Vulnerability testing was done according to the testing process using different tools. The process included information gathering, vulnerability scanning with automated tools, denial of service, and source code analysis. A few vulnerabilities were found in the testing, and instructions were given to developers in order to fix them. The findings indicate that the application was vulnerable but the vulnerabilities are easily fixed.

Keywords: web application, vulnerability, information security, testing, OWASP

Kuviot

Taulukot

Sisältö

1 Johdanto

Verkkosovellukset ovat pitkään olleet osa päivittäistä elämäämme ja niitä käytetään lähes kaikissa arjen toiminnoissa niin työelämässä kuin vapaa-aikana. Verkkosovelluksella tarkoitetaan verkossa toimivaa palvelua, yleisesti erilaisia verkkosivustoja. Oli kyseessä sitten ystäville viestittely, ostosten teko tai elokuvan katselu, voidaan kaikki nämä tehdä erilaisten verkkosovellusten avulla.

Sovellusten tietoturvallisuus on keskeinen elementti tietojärjestelmissä käsiteltävän tiedon suojaamisen ja käytettävyyden kannalta. Yhteiskunnan toimintojen ollessa entistä enemmän verkottunutta ja internetiin kytkettyä, on vaarana, että toiminnot voivat häiriintyä sovelluksissa olevien puutteiden ja haavoittuvuuksia hyväksikäyttävien tahojen toimesta (Valtiovarainministeriö 2013).

Haavoittuvuudet voivat mahdollistaa sovellusten käytön niiden alkuperäisen toimintaperiaatteen vastaisesti. Tällöin voidaan päästä käsiksi tietoon, jonka ei tulisi olla kaikkien saatavilla, muokata tietoa, jonka ei pitäisi olla muokattavissa tai estää pääsy tietoon, jonka tulisi olla saatavilla. Luottamus, eheys ja saatavuus ovat tietoturvallisuuden keskeiset elementit.

Verkkosovellusten haavoittuvuudet ovat yleisiä ja yleistyvät entisestään yhteiskunnan digitalisoituessa - viimeisen kolmen vuoden aikana haavoittuvuuslistauspalvelu CVE Detailsiin on kirjattu enemmän haavoittuvuuksia kuin koko muuna vuosikymmenenä yhteensä (The MITRE Corporation 2020c). Tämä kertoo siitä, että palveluja verkkoon siirrettäessä niiden turvallisuuteen ei välttämättä panosteta riittävästi. Shah ja Mehtre (2015) esittävät artikkelissaan, että haavoittuvuuksista yli puolet ovat ennestään tuntemattomia nollapäivähaavoittuvuuksia. Tällaisiin haavoittuvuuksiin ei ole olemassa vielä korjaavia päivityksiä ja niiden löytäminen on hankalampaa kuin tunnettujen haavoittuvuuksien.

Haavoittuvuuksia etsitään tyypillisesti haavoittuvuustestauksella. Testauksella etsitään virheellisiä asetuksia ja haavoittuvia komponentteja sekä testataan, miten sovellus selviytyy odottamattomista virhetilanteista ja väärinkäyttöyrityksistä (Valtiovarainministeriö 2013). Sovellusten tietoturvassa testaus on vain yksi osa, ja tietoturva tulisikin ottaa huomioon koko kehitysprosessissa, ei vain testausvaiheessa. Testausvaiheen tarkoituksena on varmentaa

kehitysvaiheen oikein toteutetut tietoturvatoimet, jotta sovellus voidaan ottaa käyttöön.

Tutkimuksessa tarkastellaan viimeisimpiä keinoja tutkia ja analysoida verkkosovellusten tietoturvaa. Tutkimukseen sisältyy käytännön osuus, jossa testataan yhden verkkosovelluksen tietoturvaa erilaisilla työkaluilla, kuten haavoittuvuusskannereilla. Tutkimuksessa kuvataan erilaiset keinot löytää haavoittuvuuksia ja esitetään ratkaisuja niiden korjaamiseksi. Pääosa testauksista on tehty dynaamisella blackbox-menetelmällä, eli testaamalla miten sovellus vastaa erilaisiin sille annettuihin syötteisiin. Lisäksi käytetään staattista analyysia, joka tehdään analysoimalla sovelluksen lähdekoodia.

Tietoturvaa, haavoittuvuuksia ja niiden testaamista käsitteleviä artikkeleja ja tutkielmia on paljon. Tämän tutkielman tarkoitus on koostaa viimeaikaisia tuloksia ja testausmenetelmiä yhteen kokonaisuuteen sekä palvella yksittäisen verkkosovelluksen tietoturvaa antamalla kehitysehdotuksia sen parantamiseksi.

Luvussa 2 käsitellään tutkimuksen teoreettinen viitekehys. Viitekehys on kerätty kirjallisuuskatsauksella, ja se sisältää tietoturvallisuuden, verkkosovellusten haavoittuvuuksien ja niiden testaamiseen liittyvien keskeisten käsitteiden ja prosessien määrittelyn. Luvussa 3 esitellään tutkimusmenetelmä ja tutkimuksen toteutus. Tutkimus on konstruktiivinen tutkimus, jossa konstruktio on sekä haavoittuvuustestauksen tulokset että annetut tietoturvaan liittyvät kehitysehdotukset. Luvussa 4 käsitellään verkkosovelluksen haavoittuvuustestauksen prosessia ja tehdään alusta loppuun kattava haavoittuvuustestaus. Testauksen tulokset käsitellään yleisesti tässä luvussa. Luvussa 5 aiemmin havaittujen puutteiden perusteella esitetään ehdotuksia siitä, miten verkkosovellusta tulisi kehittää tietoturvallisempaan suuntaan. Lopuksi luvussa 6 kootaan yhteen saadut tulokset ja arvioidaan tutkimuksen luotettavuutta ja eettisyyttä sekä aihetta jatkotutkimukselle.

2 Teoreettinen viitekehys

Tutkielman teoreettinen viitekehys rakentuu tietoturvallisuuden ja verkkosovellusten haavoittuvuuksien sekä niiden testaamisen ympärille. Kirjallisuuskatsauksessa tietoturvallisuus jaetaan eri osa-alueisiin, verkkosovelluksia tarkastellaan pääasiassa toiminnallisesta näkökulmasta ja haavoittuvuuksia esitellään käsitteiden määrittelyn ja erilaisten luokittelumenetelmien perusteella. Lisäksi esitellään erilaisia tietoturvallisuuden arviointi- ja testausmenetelmiä.

Tietoturvallisuus on moninainen käsite, joka pitää sisällään muun muassa haavoittuvuuden käsitteen. Siksi on tärkeää ymmärtää, mitä tietoturvallisuus tarkoittaa haavoittuvuustestauksen viitekehyksessä. Usein haavoittuvuudet heikentävät jotakin tietoturvan osa-aluetta, jonka seurauksena tietoja käytetään tiedon omistajan näkökulmasta väärin. Verkkosovelluksissa tiedolla voidaan tarkoittaa palvelimen tai sovelluksen asetuksia, toimintalogiikkaa, käyttöliittymiä (Nahari ja Krutz 2011) tai käyttäjien tietoja. Tietoturvatestauksessa pyritään etsimään haavoittuvuuksia, joiden avulla saadaan oikeudeton pääsy tietoon, estetään oikeellinen pääsy tietoon tai muutetaan oikeellista tietoa hallitsemattomasti tai luvatta.

Alaluvussa 2.1 tietoturvan eri osa-alueet avataan ja käsitellään verkkosovellusten näkökulmasta. Alaluvun 2.2 aiheena on verkkosovelluksen ja haavoittuvuuden käsitteet. Lisäksi perehdytään hieman haavoittuvuuksien nykytilaan ja vaikutuksiin maailmassa. Merkittävimpiä verkkosovellusten haavoittuvuuksia käsitellään alaluvussa 2.3 ja haavoittuvuuksien testausmenetelmiä ja testausprosessia alaluvussa 2.4.

2.1 Tietoturvallisuus

Tietoturva jaotellaan perinteisesti kolmeen osa-alueeseen, joita ovat luottamuksellisuus, saatavuus ja eheys. Osa-alueita kutsutaan myös CIA-triadiksi niiden englanninkielisten nimien mukaan (*confidentiality*, *integrity* ja *availability*). CIA-triadille ei ole löydettävissä alkuperäistä lähdettä, sillä kyseessä on käytäntöön vakiintunut tapa jakaa tietoturvallisuutta. Osa-alueiden keskinäinen tärkeysjärjestys vaihtelee suojattavasta kohteesta riippuen. Esimerkiksi turvallisuusviranomaisten ja muiden turvallisuuskriittisten toimijoiden tiedon luottamuksel-

lisuus on ensiarvoisen tärkeää. Vaikka kaupallisissa palveluissakin luottamuksellisuudella on merkitystä, tiedon eheys ja oikeellisuus korostuu pääsyn rajoittamista enemmän (Clark ja Wilson 1987). Saatavuus korostuu erityisesti palveluissa, joissa tarjotaan reaaliaikaista, muuttuvaa tietoa.

2.1.1 Luottamuksellisuus

Luottamuksellisuudella (engl. *confidentiality*) tarkoitetaan oikeudettoman pääsyn estämistä tietoihin. Luottamuksellista tietoa sovelluskehitysympäristössä voivat olla esimerkiksi asiakkaan vaatimusmäärittelyasiakirjat, sovellusarkkitehtuuri, sovelluksen lähdekoodi, testi- ja konfiguraatiotiedostot sekä sovelluksen tietokannoissa sijaitseva data (Johnson ja Stevens 2018).

Tiedon ollessa tarkoitettu vain tietyille käyttäjille, tulee pääsyä siihen rajata. Eräs konsepti pääsyn rajaamiseksi on rooliperusteinen pääsynhallinta (engl. *role-based access control*), joka on kehitetty 1970-luvulla (Sandhu ym. 1996). Vakiintunut tapa pääsynhallinnan toteuttamiseksi on kirjautuminen käyttäjätunnukselle ja salasanalla (O'Gorman 2003). Käyttäjät kuuluvat oletuksena johonkin ryhmään, jolla on oikeuksia tehdä vain tiettyjä toimenpiteitä järjestelmässä. Laajimpia pääkäyttäjän oikeuksia annetaan vain tarvittaessa, jotta voidaan minimoida tästä aiheutuvia riskejä, kuten kriittisten tietoaineistojen poistamista (Sandhu ym. 1996).

Verkkosovelluksissa luottamuksellisuuden kannalta on elintärkeää, miten tietoja säilytetään. Käyttäjän salasanoja ja muita tunnistetietoja ei tulisi tallentaa selväkielisenä tietokantaan, vaan salata ne. Tällöin pääsy tietokantaan ei vielä mahdollista pääsyä kaikkien käyttäjien tietoihin ja oikeuksiin. Myös tietoliikenteen tulee olla salattua, jottei tietoa voida kuunnella sen välittyessä osapuolelta toiselle. (Nahari ja Krutz 2011).

2.1.2 Saatavuus

Saatavuus (engl. *availability*) tarkoittaa, että oikeutetuilla käyttäjillä on tarvittaessa pääsy tietoon. Jotta saatavuus voidaan taata, sovelluksen on kestettävä myös suuret käyttäjämäärien vaihtelut tai häiriötilanteet. Suuri käyttäjämäärä voi olla seurausta palvelun suosion kasvusta

tai palvelunestohyökkäyksestä (engl. *denial-of-service attack*, DOS), jonka tarkoituksena on saada palvelu hetkellisesti tai pitkäaikaisesti pois käytöstä. Saatavuuden takaamiseksi palveluita ei tulisi rakentaa siten, että yksittäisen komponentin rooli kasvaa liian suureksi. Mikäli hyökkäys kohdistetaan tällaiseen yksittäiseen heikkoon kohtaan (engl. *single point of failure*), palvelun saatavuus voidaan katkaista pienilläkin resursseilla. (Nahari ja Krutz 2011).

Saatavuuteen voidaan päästä joko lisäämällä sen sietokykyä erilaisiin häiriötilanteisiin, jolloin palvelin kestää paremmin siihen kohdistuvaa kuormaa (Sundharam, Lakshmi ja Abarajithan 2010; Subil, Mathews ja Johnson 2005) tai monitoroimalla ja suodattamalla pois kuormittavaa tai muuten haitallista verkkoliikennettä (Yuan ja Mills 2005).

2.1.3 Eheys

Eheydellä (engl. *integrity*) tarkoitetaan sitä, että tieto säilyy muuttumattomana tilanteissa, joissa sen ei ole tarkoitus muuttua. Esimerkiksi järjestelmän virhetilanteissa tieto ei muutu hallitsemattomasti eivätkä järjestelmän käyttäjät voi muokata tietoja ilman, että muokkauksesta jää jälki. Tietokantaan tehtävät muutokset tulisi olla jäljitettävissä ja käyttäjien roolit sen mukaiset, ettei yksi käyttäjä voi poistaa tai muokata sekä itse tietoa että tiedon palauttamiseen käytettävää jäljitettävyystietoa (Clark ja Wilson 1987).

Eheyden turvaamiseksi datan aitouden varmistaminen tulisi aina olla mahdollista. Tähän voidaan käyttää yksilöllisiä tiivistearvoja (engl. *hash code*), joilla voidaan luotettavasti tunnistaa tiedon vastaavuus alkuperäisen tiedon kanssa. Tiedon alkuperä tulisi dokumentoida järjestelmään, samoin kuin siihen tehtävät muutokset. Järjestelmään kohdistuvissa häiriötilanteissa tieto tai sen alkuperä voi muuttua hallitsemattomasti ja aiheuttaa ongelmia eheydelle. Eheyteen liittyviä häiriötilanteita voivat aiheuttaa järjestelmän oma toiminta, joka virheestä johtuen muuttaa tietoja hallitsemattomasti, tai ulkopuolinen toiminta, jonka avulla jotakin haavoittuvuutta hyväksikäyttäen tietoa muutetaan. (McBride ym. 2018).

2.2 Verkkosovellukset ja niiden haavoittuvuudet

Tässä alaluvussa esitellään verkkosovelluksen ja haavoittuvuuden käsitteet ja tarkastellaan haavoittuvuustilannetta maailmassa.

2.2.1 Verkkosovellus

Verkkosovellus on palvelu, jossa käytetään verkkoselainta tietosisällön hakemiseen palvelimelta ja toisaalta myös uuden sisällön viemiseen palvelimelle. Verkkosovellus poikkeaa perinteisestä verkkosivustosta siten, että sovelluksessa sisältö on dynaamista ja vaihtelee käyttäjän antamien parametrien, asetusten ja aiemman käyttöhistorian mukaan, kun puolestaan verkkosivustolla sisältö on staattista, esimerkiksi tekstisivuja tai ladattavia tiedostoja. Nykyisin lähes kaikki verkkosivustot ovat dynaamisia ja niitä voidaan nimittää verkkosovelluksiksi. Verkkosovelluksia ovat esimerkiksi verkkokaupat, suoratoistopalvelut ja sosiaalisen median alustat. (Shklar ja Rosen 2009).

Verkkosovellusten toiminta voidaan jakaa frontendiin ja backendiin. Frontend tarkoittaa käyttäjälle näkyvää osaa sovelluksesta, kuten painikkeita, lomakkeita ja ulkoasua. Käyttöliittymän tiedot välitetään backendille, joka sisältää palvelimen ja tietokannan. Verkkosovelluksen frontend ja backend keskustelevat keskenään ilman, että se näkyy käyttäjälle. (Abdullah ja Zeki 2014). Backend voi tarjota lisäksi ohjelmointirajapinnan (engl. *application programming interface*) eli API:n. API:n kautta palvelimen tietoja voidaan hakea määrämuotoisena esimerkiksi toisiin sovelluksiin. (Ashby ja Jensen 2018).

Modernien verkkosovellusten kehittämisessä käytetään usein ohjelmointikirjastoja, joissa erilaisia valmiita komponentteja hyödyntäen saadaan tuotettua sovelluksiin haluttuja ominaisuuksia. Ehkä nykyisin tunnetuin esimerkki tällaisesta kirjastosta on React.js, joka on Facebookin kehittämä JavaScript-ohjelmointikielellä toteutettu kirjasto pääasiassa käyttöliittymäkomponenttien luomiseen (Facebook Inc. 2020).

Huolimatta itse sovelluksen toteutustavasta, verkkosovelluksissa käytetään vakiintuneesti *Hypertext Transfer Protocol* -protokollaa (HTTP) tiedon välittämiseen asiakkaan ja palvelimen välillä. HTTP on protokolla, joka on kehitetty asiakkaan ja palvelimen väliseen kommunikointiin ja on edelleen käytössä nykyaikaisissa verkkosovelluksissa. Protokolla toimii rajapintana asiakkaan ja palvelimen välillä, piilottaen tarkemman toimintalogiikan kummaltakin osapuolelta. Näin ollen osapuolten ei tarvitse huolehtia kuin siitä, että viestit lähetetään HTTP-protokollan mukaisella tavalla. (IETF 2014a).

HTTP-protokollan viestinvaihto koostuu pyynnöistä ja vastauksista, joita on eri tyyppisiä esi-

merkiksi tiedon hakemiseen (`GET`), lähettämiseen (`POST`) ja päivittämiseen (`PUT`). (IETF 2014a). HTTP-protokollan viestit sisältävät datan lisäksi otsaketietoja (engl. *header*), joilla palvelin määrittää selaimelle muun muassa sisällön tiedostomuodon ja pituuden. Otsakkeilla on merkitystä myös sovelluksen turvallisuuden kannalta, sillä niillä voidaan määritellä selaimelle välimuistin tyhjentämistä tai estää joitakin haitallisia toiminnallisuuksia. Jokaisen pyynnön vastauksessa on myös statuskoodi, joka kertoo asiakkaalle vastauksen tyypin. Kun pyydetään olemassa olevaa verkkosivua `GET`-pyynnöllä, siihen vastataan statuskoodilla `200 Found`. Mikäli sivulle on asetettu uudelleenohjaus toiselle sivulle, statuskoodi on `301 Moved Permanently`. Jos sivua ei ole olemassa, statuskoodi on `404 Not Found`. (IETF 2014b).

Nykyisin suurimmassa osassa internetin suosituimpia palveluita käytetään HTTP:n salattua versiota, HTTPS-protokollaa (Helme 2019). HTTPS ei eroa merkittävästi toiminnallisuudeltaan verrattuna HTTP-protokollaan. Siinä salaus on toteutettu *Transport Layer Security* (TLS) -protokollaa käyttäen (vanhemmissa versioissa *Secure Sockets Layer* (SSL)). (IETF 2005).

TLS-protokollan tärkein tehtävä on taata turvallinen kommunikaatiokanava vastaanottajan ja lähettäjän välille. Kun välitettävä data on luettavissa selkokielellä vain päätepisteissä, pystytään tiedon luottamuksellisuus takaamaan. Tällöin tietoa ei myöskään voida muokata, mikä takaa sen eheyden. Ennen tiedon välittämistä kommunoivat tahot sopivat yhteyden muodostamiseksi tarvittavien salausavainten vaihdosta menettelyllä, jota kutsutaan kättelyksi. Kättelyn jälkeen turvallinen viestinvaihto voi alkaa. (IETF 2018).

HTTP on tilaton protokolla, mikä tarkoittaa, että sillä voidaan välittää viestejä, mutta pelkästään niiden perusteella ei tiedetä mihin istuntoon viestit liittyvät. HTTP ei siis itsessään mahdollista verkkokauppojen ostoskorin säilyttämistä tai sovelluksiin kirjautumista, vaan näiden tietojen ylläpitämiseen tarvitaan evästeitä (engl. *cookies*). Evästeet ovat HTTP-protokollan tapa ylläpitää sovelluksen istuntoja. Tieto evästeestä välitetään HTTP-protokollan otsaketiedoissa, jolloin sekä selain että palvelin tietävät, mihin istuntoon tietty pyyntö liittyy. (IETF 2011).

Verkkosivustojen sisällä ja niiden välillä navigoidaan *Uniform Resource Locator* -osoitteita

(URL) käyttäen. URL on uniikki tunniste, joka kertoo tiedon hakemiseen käytettävän protokollan (esimerkiksi `https://`) ja osoitteen (`example.com`). (IETF 2000) Verkkosivustot esitetään *Hypertext Markup Language* -merkintäkieltä (HTML) käyttäen. HTML-kielellä kuvataan verkkosivun sisältö ja rakenne. (WHATWG 2020)

2.2.2 Haavoittuvuus

Haavoittuvuudelle on monia eri määritelmiä. Turvallisuuskomitea (2018) esittää kyberturvallisuuden sanastossaan, että haavoittuvuus on *"mikä tahansa heikkous, joka mahdollistaa vahingon toteuttamisen tai jota voidaan käyttää vahingon aiheuttamisessa"*. Tämä määritelmä on varsin laaja, sillä se ei rajaa haavoittuvuutta vain tietojärjestelmien käsitteeksi, vaan haavoittuvuuksia voi ilmetä myös prosesseissa tai ihmisten toiminnassa. Lisäksi haavoittuvuudella tarkoitetaan sanaston mukaan alttiutta uhkille, jotka kohdistuvat aiemmin käsiteltyihin tietoturvallisuuden osa-alueisiin - luottamuksellisuuteen, saatavuuteen ja eheyteen.

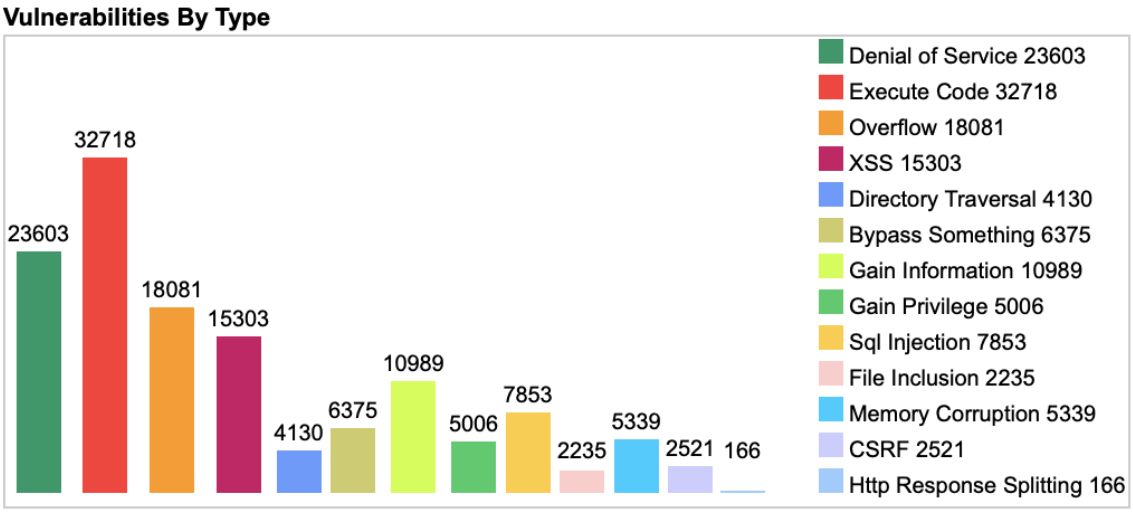
Muiden määritelmien mukaan haavoittuvuus on virhe tai heikkous ohjelmassa, jota hakkerit voivat käyttää saadakseen pääsyn järjestelmään (Wang ym. 2009) tai jota vihamielinen käyttäjä voi hyväksikäyttää aiheuttaen vahinkoa järjestelmälle (Pfleeger ja Pfleeger 2003). Raggad (2008) esittää, että haavoittuvuus on sellainen ominaisuus tai heikkous suojattavan kohteen komponentissa, joka lisää siihen kohdistuvan uhkan todennäköisyyttä.

Vielä yhden määritelmän mukaan haavoittuvuus on aukko tai heikkous sovelluksessa, jonka avulla hyökkääjä voi aiheuttaa vahinkoa sovelluksen käyttäjille, omistajalle tai muille sovelluksesta riippuvaisille tahoille. Haavoittuvuus voi olla seurausta virheestä joko sovelluksen suunnittelun tai käyttöönoton aikana. (OWASP Foundation 2016).

Kaikkien määritelmien ollessa varsin lähellä toisiaan, tässä tutkimuksessa haavoittuvuudella tarkoitetaan nimenomaisesti viimeksi esitettyä tapausta. Haavoittuvuus on siis tekninen ominaisuus, joka on seurausta sovelluksen virheellisestä suunnittelusta. Tämän tutkimuksen haavoittuvuuden määritelmä ei kata niitä hallinnollisten prosessien puutteita, joita ei voida arvioida puhtaasti teknisten ominaisuuksien perusteella. Haavoittuvuus erotetaan uhkista ja hyökkäyksistä, jotka ovat sovellukseen tosiasiassa tai teoreettisesti kohdistuvia aktiivisia toimenpiteitä sen toiminnan häiritsemiseksi.

Nollapäivähaavoittuvuus on tietojärjestelmässä oleva haavoittuvuus, johon ei ole saatavilla korjausta. (Turvallisuuskomitea 2018). Tällaiset haavoittuvuudet jäävät yleensä taka-alalle haavoittuvuustestauksessa, sillä testauksessa pyritään etsimään olemassaolevia haavoittuvia komponentteja.

Kuviossa 1 esitetyllä tavalla suurin osa haavoittuvuuksista jakautuu tyypin mukaan luvattoman koodin suorittamiseen, palvelunestoihin ja ylivuotoihin (engl. *overflow*), joita voivat sovelluksessa olla esimerkiksi muistialueen ylivuoto. Kuviossa on esitetty vuodesta 1999 alkaen kaikki luokitellut haavoittuvuudet.



Kuvio 1. Verkkosovellusten haavoittuvuudet tyypin mukaan. (The MITRE Corporation 2020d)

Haavoittuvuuksien luokitteluun käytetään *Common Vulnerability Scoring System* -pisteytystä (CVSS). Haavoittuvuudet luokitellaan vakavuuden perusteella asteikolle nollasta kymmeneen taulukon 1 mukaisesti. CVSS-pisteet määritetään perustuen haavoittuvuuden yleisiin tietoihin, ajallisiin tietoihin ja ympäristötietoihin. Yleistiedoissa pisteytyksen suuruuteen vaikuttavat muun muassa hyökkäystapa, hyökkäyksen kompleksisuus sekä minkälainen vaikutus onnistuneella haavoittuvuuden hyväksikäytöllä on tietoturvallisuuden osa-alueisiin. Ajallisilla tiedoilla viitataan siihen, onko haavoittuvuudelle olemassa todellista hyväksikäyttömahdollisuutta, esimerkiksi toimivaa ohjelmakoodia, vai onko haavoittuvuus teoreettinen. Lisäksi arvioidaan sitä, onko haavoittuvuuden korjaamiseksi saatavilla päivitystä. Ympäris-

tötietojen arvioinnissa osa-alueet pisteytetään siten, kuinka vakavia seurauksia tietyn tietoturvan osa-alueen vaarantuminen aiheuttaa. (FIRST.org 2019).

Taulukko 1. CVSS-pisteytyksen vakavuustasot (FIRST.org 2019)

Arviointi	CVSS-pisteet
Ei arviota	0.0
Matala	0.1–3.9
Keskitaso	4.0–6.9
Korkea	7.0–8.9
Kriittinen	9.0-10.0

Luokittelun lisäksi kaikki ilmoitetut haavoittuvuudet saavat oman *Common Vulnerabilities and Exposures* -tunnisteen (CVE), jonka avulla haavoittuvuustietoa jakavat tahot voivat luotettavasti tietää keskustelevansa samasta haavoittuvuudesta. CVE-tunnisteella voidaan hakea eri palveluntarjoajien tuottamaa tietoa haavoittuvuudesta. Tällaisia tietoja ovat muun muassa CVSS-pisteet, alustat ja versiot joita haavoittuvuus koskee, haavoittuvuuden syyt ja vaikutukset sekä mahdolliset esitetyt korjaustoimenpiteet. (The MITRE Corporation 2020e).

Haavoittuvuudet voidaan luokitella myös niiden tyypin mukaan *Common Weakness Enumeration* -luokkiin (CWE). CWE-listaus sisältää yleisiä ohjelmistojen ja laitteistojen heikkouksia. Heikkous on määritelty suunnittelu-, ohjelmointi- tai käyttöönottovirheeksi tai haavoittuvuudeksi, joka voi aiheuttaa alttiutta hyökkäyksille. Heikkous on laajempi käsite kuin haavoittuvuus, sillä heikkous voi sisältää esimerkiksi huonoja ohjelmointikäytänteitä, jotka eivät kuitenkaan suoranaisesti vaikuta sovelluksen turvallisuuteen. (The MITRE Corporation 2020a).Yleisimpiin ja vaarallisimpiin CWE-luokituksiin kuuluvat muun muassa erilaiset käyttäjät syötteen virheelliset käsittelyt ja puutteelliset rajoitustoimet, jotka mahdollistavat ohjelmassa eri muistialueille kirjoittamisen. (The MITRE Corporation 2019).

2.2.3 Yleiskatsaus haavoittuvuuksien tilanteeseen ja vaikutuksiin

Verkkosovellukset yleistyvät jatkuvasti ja samoin myös niihin liittyvät haavoittuvuudet. Haavoittuvuudet eivät ole vain IT-alan ongelma, sillä ne koskettavat nykyisin kaikkia yhteiskun-

nan toimintoja. Vuosina 2018-2019 raportointiin kymmenen kriittistä haavoittuvuutta, jotka jakautuivat eri sektorien kesken taulukon 2 mukaisesti. (ENISA 2019).

Taulukko 2. Kriittiset haavoittuvuudet sektoreittain (ENISA 2019)

Sektori	Kriittiset haavoittuvuudet
Energia	5
Vesi	2
Kemikaali	1
Elintarvike ja maatalous	1
Liikekiinteistöt	1

Yritykset ympäri maailman kokevat, että haavoittuvuuksia hyväksikäytettäessä niillä on suurin vaikututus yrityksen liiketoimintaan sekä asiakkaiden korvausvaatimusten että mainehaitan takia. Lisäksi vaikutuksia voi olla myös asiakasvaihtuvuuteen ja osakehintojen laskuun. (Radware 2019).

Traficom (2020) raportoi vuosijulkaisussaan, että haavoittuvuuksia hyväksikäytetään entistä nopeammin. Tämä tarkoittaa sitä, että kun uusi haavoittuvuus julkaistaan, sitä yritetään hyödyntää mahdollisimman pian, kun organisaatiot eivät ole tehneet korjaavia toimenpiteitä.

Mounika, Yuan ja Bandaru (2019) vertailivat vuosien 1999-2019 haavoittuvuustietoja, ja tämän perusteella ulkoisia XML-viittauksia hyödyntävät haavoittuvuudet sekä puutteellinen pääsynhallinta ja käyttäjän todentaminen ovat koko vertailujakson aikana yleistyneet, kun taas XSS-haavoittuvuudet vähentyneet.

Security Headers on palvelu, jossa voi tarkastaa minkä tahansa verkkosivun otsaketietojen turvallisuuden. Sivu myös ylläpitää tietoa kaikista tarkastetuista sivuista. Kirjoitushetkellä sivustolla oli tarkastettu noin 80 miljoonaa uniikkia sivustoa. Näistä yli puolet, 44 miljoonaa, luokiteltiin huonoimpaan F-luokkaan otsakkeiden turvallisuuden näkökulmasta. Toisaalta myös A- ja A+ -luokissa oli lähes 10 miljoonaa sivustoa. (Helme 2020).

2.3 Merkittävimmät haavoittuvuudet

Tässä alaluvussa esitellään merkittävimmät verkkosovellusten haavoittuvuudet OWASP Top Ten -listauksen mukaan. Listaus on muutaman vuoden välein päivitettävä, maailmanlaajuisesti tunnistettu kuvaus merkittävimmistä verkkosovellusten haavoittuvuuksista. Se on ensisijaisesti tietoisuutta luova työkalu, jota voidaan käyttää sovellusten turvallisuuden parantamiseen (OWASP Foundation 2017).

Alaluvuissa on esitetty kunkin OWASP Top Ten 2017 -listaukseen kuuluvan haavoittuvuuden ominaisuudet. Vertailtavina ominaisuuksina ovat hyödynnettävyys, yleisyys, havaittavuus ja vaikutukset. Hyödynnettävyydellä tarkoitetaan sitä, kuinka helposti haavoitettavuus on hyödynnettävissä. Matalan hyödynnettävyyden haavoittuvuuksissa tarvitaan usein tietyt olosuhteet, kun taas korkean hyödynnettävyyden haavoittuvuudet toimivat lähes varmasti. Yleisyys kertoo haavoittuvuuden ilmenemisen suhteessa kaikkiin haavoittuvuuksiin. Havaittavuudella kuvataan sitä, kuinka hyvin haavoittuvuus tai sen hyväksikäyttö on havaittavissa kohteessa. Vaikutus tarkoittaa sitä, millaiset mahdollisuudet onnistuneella hyväksikäytöllä on vaikuttaa kohteen toimintaan ja tietoturvallisuuteen.

2.3.1 Injektiot

Injektioilla tarkoitetaan tilannetta, jossa palvelimelle tehtävän kyselyn mukana lähetetään dataa, joka tulkitaan palvelimella muuna kuin kyselynä. Injektiolla voidaan saada suoritettua ohjelmakoodia tai hakea tietoa, jonka ei pitäisi olla muuten saatavilla. Injektiohyökkäys voi kohdistua esimerkiksi tietokantaan tai käyttöjärjestelmän rajoitetulle osiolle. (OWASP Foundation 2017).

Tietokantaan kyselykielellä tehtävä injektio eli SQL-injektio on määritelty luokkaan CWE-89, mutta kaikkiaan erilaisia injektioita CWE-luokituksessa on kymmenen. (The MITRE Corporation 2020f). SQL-injektio on määritelty tilanteeksi, jossa kyselykielessä käytettäviä erikoismerkkejä ei sanitoida kyselystä ja niillä voidaan muokata varsinaisen kyselyn rakennetta. Esimerkiksi kirjautumislomakkeen kautta voidaan saada tietokantaan syötettyä kysely, joka palauttaa nähtäville listan kaikista käyttäjistä heidän tietoineen ja jopa salasanoineen. (Al-Khurafi ja AlAhmad 2015).

Taulukko 3. Injektiot (OWASP Foundation 2017)

Hyödynnettävyys	Yleisyys	Havaittavuus	Vaikutus
korkea	keskitaso	korkea	korkea

2.3.2 Puutteellinen käyttäjän todentaminen

Kun sovelluksen toiminnot käyttäjän todentamiseen ja istunnonhallintaan on toteutettu virheellisesti, hyökkääjät voivat varastaa käyttäjätunnuksia, avaimia ja istunnon tunnuksia tai muuta tietoa (OWASP Foundation 2017). CWE-listaus määrittää puutteelliseen todentamiseen muun muassa huonosti suojatut tunnisteet tai virheellisesti toteutetun todentamisen ja heikot käyttäjän salasanaan liittyvät hallintatoimet (The MITRE Corporation 2020b). Tällaisia ovat esimerkiksi heikot oletussalasanat, monivaiheisen todennuksen puute tai selkokielisten salasanojen tallentaminen tietokantaan (OWASP Foundation 2017).

Esimerkki heikosta todennustavasta on käyttäjän istunnon tunnuksen välittäminen palvelimelle URL-osoitteessa. Tällöin kuka tahansa osoitteen saanut pääsee käsiksi istuntoon ja voi nähdä ja muokata käyttäjän tietoja. Myös automaattisen istunnon vanhenemisen puute tai vanhenemisen pitkä aika saattavat altistaa käyttäjät tietojen vaarantumiselle, erityisesti mikäli sovellusta käytetään yhteiskäyttöisellä päätelaitteella. (Al-Khurafi ja AlAhmad 2015).

Taulukko 4. Puutteellinen käyttäjän todentaminen (OWASP Foundation 2017)

Hyödynnettävyys	Yleisyys	Havaittavuus	Vaikutus
korkea	keskitaso	keskitaso	korkea

2.3.3 Arkaluonteisen tiedon paljastuminen

Usein sovellukset tai API:t eivät suojaa arkaluonteista tietoa riittävästi. Tietoa ei esimerkiksi suojata, kun sitä välitetään eri tahoille tai käytetään liian heikkoja suojauksia, jotka ovat helposti murrettavissa. Hyökkääjät voivat saada käsiinsä tai muokata tällaista heikosti suojattua tietoa. Viimeisten vuosien aikana tämän haavoittuvuustyypin hyväksikäytöllä on saatu aikaan suurimpia vaikutuksia hyökkäyksen ollessa onnistunut. (OWASP Foundation 2017).

Vaikka arkaluonteisen tiedon paljastuminen on merkitty omaksi haavoittuvuudekseen, se on kuitenkin usein seurausta jonkun muun haavoittuvuuden hyväksikäytöstä. Esimerkiksi injektioilla tai puutteellisella käyttäjän todentamisella voidaan päästä käsiksi arkaluontoisiin tietoihin. (OWASP Foundation 2017).

Taulukko 5. Arkaluonteisen tiedon paljastuminen (OWASP Foundation 2017)

Hyödynnettävyys	Yleisyys	Havaittavuus	Vaikutus
keskitaso	korkea	keskitaso	korkea

2.3.4 XML-kielen ulkoiset viittaukset

Extensible Markup Language (XML) on merkintäkieli, jota käytetään dokumenttien rakenteen kuvaamiseen. XML-kieltä prosessoidaan paikallisesti käyttäjän selaimella. Vanhempien selainten XML-prosessorit voivat sallia viittaukset ulkoisiin tahoihin, esimerkiksi haitallisille verkkosivustoille. Tällaista haavoittuvuutta kutsutaan *XML External Entity* -haavoittuvuudeksi (XXE). Sitä voidaan käyttää tiedon varastamiseen, komentojen suorittamiseen ja myös palvelunestohyökkäyksiin. (OWASP Foundation 2017). CWE-luokituksessa on määritetty kaksi haavoittuvuustyyppiä, jotka molemmat liittyvät puutteellisiin prosessoinnin rajoituksiin.

Tyypillinen esimerkki palvelunestohyökkäyksestä, joka on toteutettu XXE-haavoittuvuutta hyväksikäyttäen, on niin sanottu *billion laughs*. Tässä hyökkäyksessä hyökkääjä sisällyttää XML-dokumenttiin lukuisia viittauksia ulkoisiin tahoihin, jolloin prosessointi vaatii useita gigatavuja käyttömuistia, aiheuttaen lopulta palvelun toimimattomuuden. (Rasheed, Dietrich ja Tahir 2019).

Taulukko 6. XML-kielen ulkoiset viittaukset (OWASP Foundation 2017)

Hyödynnettävyys	Yleisyys	Havaittavuus	Vaikutus
keskitaso	keskitaso	korkea	korkea

2.3.5 Puutteellinen pääsynhallinta

Puutteellisella pääsynhallinnalla tarkoitetaan sitä, että käyttäjien toimia ei ole rajoitettu oikealla tavalla. Puutteelliset rajoitukset mahdollistavat esimerkiksi sen, että toisten käyttäjien tietoja päästään katsomaan tai muuttamaan sekä myös sen, että voidaan päästä käsiksi palvelimella oleviin konfiguraatiotiedostoihin ja muuttaa käyttöoikeuksia. Usein ongelmia ilmenee silloin, kun sovelluksessa ei ole alusta alkaen toteutettu asianmukaista pääsynhallintaa, vaan sitä on kehitetty ajan saatossa. Tällöin lähdekoodissa voi olla siellä täällä komponentteja, jotka toteuttavat pääsynhallintaa, mikä hankaloittaa pääsynhallinnan kokonaiskuvan ja kattavuuden hahmottamista. (OWASP Foundation 2017).

Usein sivuston ylläpitäjillä on hallintapaneeli, jota käytetään päivityksiin ja hallintaan julkisen verkon ylitse. Tällaiset ylläpitäjän käyttöliittymät ovat usein ensisijainen hyökkäyskohde pääkohde, sillä niiden avulla voidaan päästä käsiksi merkittäviin tietoihin. Hallintapaneelien näkyvyyttä tulisi rajoittaa ja kehittää lisää turvatoimia, jotta ne eivät olisi alttiita väärinkäytöksille. (OWASP Foundation 2017).

Taulukko 7. Puutteellinen pääsynhallinta (OWASP Foundation 2017)

Hyödynnettävyys	Yleisyys	Havaittavuus	Vaikutus
keskitaso	keskitaso	keskitaso	korkea

2.3.6 Virheellinen turvallisuuskonfiguraatio

Virheet konfiguraatioissa ovat yleisimpiä haavoittuvuuksia aiheuttavia tekijöitä. Tämä käsittää niin turvattomat oletusasetusasetukset, väärin konfiguroidut HTTP-otsakkeet ja paljastavat virheviestit, jotka sisältävät sensitiivistä tietoa palvelimen asetuksista. Virheenkäsittelyssä tietoja virheestä ei tulisi näyttää käyttäjälle, vaan uudelleenohjata käyttäjä geneeriselle virhesivustolle. Riskejä muodostuu myös, mikäli palvelimen tarpeettomia toimintoja ei ole otettu pois käytöstä, mutta niitä ei myöskään valvota. (OWASP Foundation 2017).

Virheellinen konfiguraatio on erityisen haitallinen silloin, kun se johtaa muiden haavoittuvuuksien, kuten XSS-hyökkäysten tai injektioiden hyväksikäyttöön (Eshete, Villafiorita ja Weldemariam 2011). Sovelluskehityksessä käytettävät debuggaustyökalut saattavat jäädä

piilottamatta peruskäyttäjiltä. Näiden avulla hyökkääjä voi esimerkiksi ohittaa käyttäjän todentamiseen käytettäviä menetelmiä ja saada pääsyn sensitiiviseen dataan. (Wainakh, Wabbi ja Alkhatib 2014).

Taulukko 8. Virheellinen turvallisuuskonfiguraatio (OWASP Foundation 2017)

Hyödynnettävyys	Yleisyys	Havaittavuus	Vaikutus
korkea	korkea	korkea	keskitaso

2.3.7 Cross-Site Scripting (XSS)

Cross-Site Scripting -haavoittuvuuksissa (XSS) verkkosovellus hyväksyy uudelle verkkosivulle dataa tai päivittää vanhaa sivua ilman sen sisällön asianmukaista tarkastamista. Tällöin hyökkääjä voi muuttaa esimerkiksi verkkosivujen sisältöä, ohjata käyttäjän haitalliselle sivustolle tai kaapata istuntoja. (OWASP Foundation 2017).

XSS-haavoittuvuudet jaetaan pysyviin ja ei-pysyviin. Ei-pysyvissä eli reflektoiduissa hyökkäyksissä hyödynnetään esimerkiksi URL-osoitteeseen kirjoitettua skriptiä, joka toimii vain jos hyökkääjä saa linkin jaettua uhreilleen, jotka sen jälkeen vierailevat sivustolla linkin kautta. Skripti voi esimerkiksi pakottaa uudelleenohjauksen toiselle, haitalliselle sivustolle tai se voi lähettää uhrin istuntotiedot hyökkääjälle. Haitallista sivustoa voidaan käyttää edelleen haittaohjelman levittämiseen tai tietojenkalasteluun (engl. *phishing*). (Al-Khurafi ja AlAhmad 2015).

Taulukko 9. XSS (OWASP Foundation 2017)

Hyödynnettävyys	Yleisyys	Havaittavuus	Vaikutus
korkea	korkea	korkea	keskitaso

2.3.8 Turvaton sarjallistamisen purku

Sarjallistamisella tarkoitetaan ohjelmointikielen ominaisuutta, jossa olio (engl. *object*) muutetaan binäärimuotoon esimerkiksi tiedostoon tallentamisen tai verkon yli lähettämisen yhteydessä. Sarjallistamisen purulla taas tarkoitetaan sitä, että vastaanotettu binäärimuoto pu-

retaan ohjelmointikielen olioksi. Haavoittuvuus voi ilmetä, jos sarjallistettua dataa puretaan varmentamattomista lähteistä, jolloin se voi olla hyökkääjän muokkaamaa. (Khunphet 2019). Tällainen varmentamaton tiedon lukeminen voi johtaa haitallisen ohjelmakoodin suorittamiseen sovelluksessa, joka mahdollistaa monia vakavia hyökkäyksiä. (OWASP Foundation 2017).

Taulukko 10. Turvaton sarjallistamisen purku (OWASP Foundation 2017)

Hyödynnettävyys	Yleisyys	Havaittavuus	Vaikutus
matala	keskitaso	keskitaso	korkea

2.3.9 Haavoittuvien komponenttien käyttö

Verkkosovellukset käyttävät lukuisia eri komponentteja, jotka ovat usein riippuvaisia vielä suuremmasta määrästä erilaisia komponentteja. Nämä komponentit voivat olla ohjelmointikirjastoja tai muita moduuleja. Mikäli edes jokin näistä komponenteista on haavoittuva, mahdollistaa se väärinkäytöt koko sovelluksen tasolla, sillä komponentit suoritetaan samoilla käyttöoikeuksilla kuin itse ohjelma. Haavoittuvien komponenttien käyttö mahdollistaa lukuisia erilaisia hyökkäystyyppejä. (OWASP Foundation 2017).

Haavoittuvien komponenttien käyttö voi myös kohdistaa hyökkäyksiä sovellukseen, sillä hyökkääjät voivat automaattisesti skannata verkkoa tunnettuja haavoittuvuuksia tunnistavilla palveluilla. Tällöin hyökkäyksen kohteeksi voi joutua myös sovellus, joka ei muuten olisi kiinnostava. (Cadariu ym. 2015).

Taulukko 11. Haavoittuvien komponenttien käyttö (OWASP Foundation 2017)

Hyödynnettävyys	Yleisyys	Havaittavuus	Vaikutus
keskitaso	korkea	keskitaso	keskitaso

2.3.10 Riittämätön lokitus ja valvonta

Mikäli sovelluksen toimintaa ei valvota, ei voida myöskään havaita hyökkääjien vihamielisiä toimia. Jos hyökkääjillä on vapaa liikkuvuus, he voivat parantaa jalansijaansa järjestelmäs-

sä. Vuonna 2016 tietomurron havaitseminen kesti keskimäärin yli puoli vuotta. Tänä aikana hyökkääjä voi tehdä käytännössä mitä vain toimenpiteitä, jos niihin ei kohdistu minkäänlaista valvontaa. Lokitettavia tietoja ovat yleensä onnistuneet ja epäonnistuneet kirjautumiset, virheviestit sekä verkkoliikenne erityisesti skannausten osalta. (OWASP Foundation 2017).

Taulukko 12. Riittämättön lokitus ja valvonta (OWASP Foundation 2017)

Hyödynnettävyys	Yleisyys	Havaittavuus	Vaikutus
meskitaso	korkea	matala	keskitaso

2.4 Tietoturvallisuuden testausmenetelmiä

Tässä luvussa esitetään yleisesti menetelmiä, joilla verkkosovellusten tietoturvallisuutta voidaan arvioida ja testata. Testaus voidaan tehdä manuaalisesti, automatisoidusti tai molempia yhdistelemällä. Usein menetelmät täydentävät toisiaan, sillä siinä missä automaattisella testauksella saavutetaan kattavuutta, manuaalisilla menetelmillä päästään syvemmälle ja voidaan löytää tosiasiallisia aukkoja sovelluksissa (Raggad 2008). Testaustyökalut jaotellaan dynaamisiin (engl. *dynamic application security testing*, DAST) ja staattisiin (engl. *static application security testing*, SAST). Jälkimmäisestä menetelmästä käytetään myös nimitystä staattinen analyysi. (Phadke 2016).

Haavoittuvuustestauksen hyödyt testattavan kohteen tietoturvallisuuden tilan selvittämiseksi ovat selkeät: sillä saadaan yksityiskohtaista tietoa sovelluksesta sekä löydetään mahdollisia ohjelmointi- ja konfiguraatiovirheitä, jotka saattavat vaarantaa tietoturvallisuutta. Näillä tiedoilla sovelluksen tietoturvaa voidaan parantaa, jolloin sovelluksen ja sen käyttäjien tiedot säilyvät luottamuksellisina, eheinä ja saatavilla. Toisaalta haavoittuvuustestausta on vaikeaa tehdä niin kattavasti, että sillä voitaisiin sulkea pois haavoittuvuuksien olemassaolon mahdollisuus kokonaan. Parhaimmillaan testaus ovat vain katsaus sovelluksen tilanteeseen sillä ajanhetkellä ja niissä olosuhteissa, joissa testaus on toteutettu. (Khera ym. 2019). On myös huomioitava, että tässä tutkimuksessa esitetyt testausmenetelmät eivät ota kantaa esimerkiksi järjestelmän ylläpito- tai hallintaprosesseihin, vaan puhtaasti teknisiin ominaisuuksiin.

Testaus voidaan toteuttaa joko organisaation sisä- tai ulkopuolelta. Sisäpuolelta tehtävä tes-

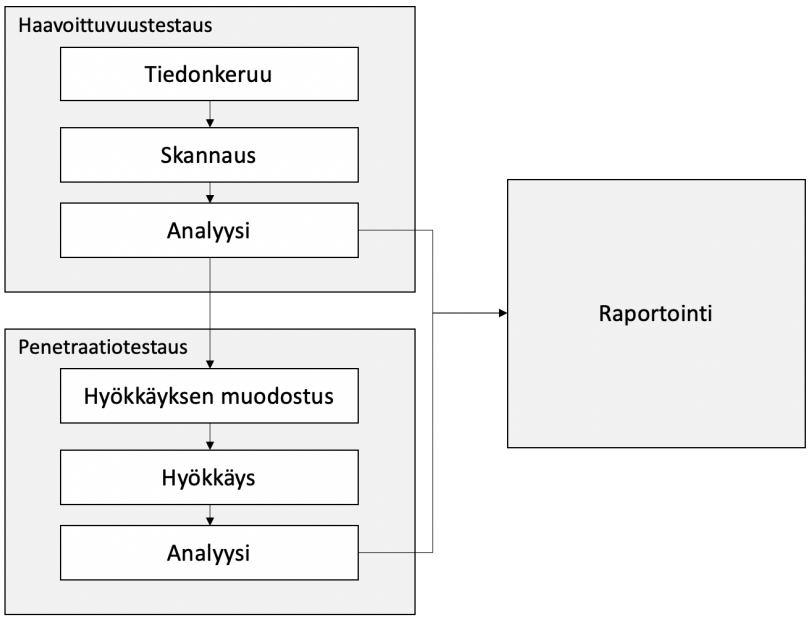
taus simuloi organisaation omaa työntekijää, joka voi toimia haitallisesti. Ulkopuolinen testaus näyttää, millä tavalla ulkopuoliset voivat löytää ja hyödyntää haavoittuvuuksia sovelluksesta. (Kumar 2014).

Testauksen luonteeseen liittyy myös näkyvyys testattavaan kohteeseen. Automatisoidut dynaamiset testaukset toteutetaan blackbox-menetelmällä (Awang ja Manaf 2013). Blackbox-testauksella tarkoitetaan sitä, että ilman tarkempaa tietoa järjestelmän sisäisestä rakenteesta siihen kohdistetaan erilaisia tavallisia, haitallisia ja sattumanvaraisia syötteitä. Näiden avulla pyritään saamaan aikaan haluttu lopputulos tai tietoa järjestelmän tarkemmasta toiminnasta, joita se antaa vastaamalla syötteisiin tietyllä tavalla. Staattisesta testauksesta käytetään myös nimitystä whitebox-testaus. Siinä sovelluksen lähdekoodi on saatavilla, jolloin sitä voidaan tutkia ja yrittää kohdistaa toimenpiteitä tiettyjä komponentteja kohtaan. Järjestelmän tarkempi toiminta on tiedossa ja sen testaus on osin helpompaa, eikä haavoittuvuuksien löytäminen välttämättä edes vaadi testausajoja. (Zeeshan ym. 2017).

Lukuisissa viimeaikaisissa lähteissä jaotellaan erikseen haavoittuvuustestaus (engl. *vulnerability assessment*, myös *vulnerability testing*) ja penetraatiotestaus (engl. *penetration testing*), jotka muodostavat haavoittuvuus- ja penetraatiotestauksen kokonaisuuden. Tämän tutkimuksen painopiste on sovellukseen tehtävässä haavoittuvuustestauksessa, mutta myös penetraatiotestausta käsitellään pintapuolisesti. Edellä esitetty jaottelu ei kuitenkaan ole selvä kaikessa aiheeseen liittyvässä kirjallisuudessa. Esimerkiksi Bacudio ym. (2011) esittävät, että haavoittuvuustestaus on osa penetraatiotestausta. Näin voidaan ajatella, sillä penetraatiotestausta ei voida toteuttaa järkevästi ilman ensin toteutettua haavoittuvuustestausta. Zeeshan ym. (2017) mukaan penetraatiotestaus taas jatkaa siitä, mihin haavoittuvuustestaus loppuu.

Sekä haavoittuvuus- että penetraatiotestauksen voidaan nähdä noudattavan prosessia, jossa ensin suunnitellaan ja valmistellaan testausta, tehdään kohteeseen liittyvää tiedustelua ja toteutetaan itse testaus. Tämän jälkeen analysoidaan havaitut poikkeamat ja raportoidaan ne. Jos haavoittuvuustestaus nähdään penetraatiotestauksen osana, tulee välivaiheeksi lisäksi myös haavoittuvuuksien hyödyntäminen (Jiménez 2016). Bacudio ym. (2011) esittävät, että testaus koostuu valmistelu-, testaus- ja analyysivaiheista. Varsinainen testausvaihe koostuu tiedon keruusta, haavoittuvuuksien analysoinnista ja haavoittuvuuksien hyödyntämisestä.

Shinde ja Ardhapurkar (2016) kuvaavat prosessit haavoittuvuus- ja penetraatiotestauksen osalta erillisinä kuvion 2 mukaisesti. Haavoittuvuustestauksen prosessi koostuu tiedonkeruusta, skannauksesta ja analyysista, josta voidaan jatkaa joko raportointiin tai penetraatiotestaukseen, joka koostuu hyökkäyksen muodostamisesta, hyödyntämisestä ja analyysistä. Prosessi alkaa aina haavoittuvuustestauksella, jolloin penetraatiotestaus vaatii ensin toteutetun haavoittuvuustestauksen.



Kuvio 2. Haavoittuvuus- ja penetraatiotestauksen prosessi (Shinde ja Ardhapurkar 2016, mukailtu).

2.4.1 Haavoittuvuustestaus

Haavoittuvuustestauksen tarkoituksena on löytää tunnistettuja haavoittuvuuksia ja virhetilanteita sovelluksesta ja saada tietoa haavoittuvuuksien määrästä, laadusta ja vakavuudesta. Haavoittuvuustestauksessa käytetään useimmiten automatisoituja skannaustyökaluja, joiden toiminta perustuu siihen, että ne skannaavat järjestelmällisesti yksittäiset sivut ja asiakkaan ja palvelimen välisen kommunikaation sekä palvelimen tiedot havaitakseen joko palveluiden haavoittuvia versioita tai virheellisiä konfiguraatioita. (Nagpure ja Kurkure 2017).

Raggad (2008) esittää, että haavoittuvuustestauksen tarkoituksena on tuottaa nopeita tulok-

sia, joilla voidaan tunnistaa järjestelmän heikkouksia ja niiden mahdollisia vaikutuksia. Koska haavoittuvuustestaus on varsin nopeaa ja perustuu tunnisteiden vertailuun, usein on varsin todennäköistä havaita myös haavoittuvuuksia, joita sovelluksessa ei tosiasiassa ole. Näitä kutsutaan false positive -tuloksiksi. Niiden poistamiseksi testauksessa voidaan käyttää useita eri työkaluja sekä manuaalista varmistamista.

Automaattisen haavoittuvuustestauksen etuna on sen kustannustehokkuus, sillä järjestelmän skannaus voidaan tehdä sen koosta riippuen minuuteissa tai muutamassa tunnissa. Skannaus voidaan myös ajastaa tehtäväksi tietyin väliajoin, jolloin mahdolliset uudet haavoittuvuudet tulevat tietoon. Manuaalisen testaukseen vaaditaan aina testaamiseen ja kohdesovelluksen teknologioihin perehtyneitä ammattilaisia, jolloin sekä rahalliset että ajalliset kustannukset voivat kasvaa suuriksi. Toisaalta automaattinen testaus on harvemmin räätälöity juuri kohdesovellusta varten, koska siinä käytetään valmiita kaupallisia työkaluja. Tällöin on riski, että testaus ei löydä merkittäviä puutteita. (Kumar 2014).

2.4.2 Penetraatiotestaus

Penetraatiotestauksella hyödynnetään haavoittuvuustestauksen tuloksia ja löydetään uusia haavoittuvuuksia, sekä yritetään hyväksikäyttää haavoittuvuuksia, jotta päästään sisään järjestelmään. Siinä missä haavoittuvuustestaus on automaattista, penetraatiotestaus on lähes poikkeuksetta manuaalista. Penetraatiotestauksessa yritetään hyödyntää haavoittuvuuksia, joilla saavutetaan pääsy järjestelmään tai saadaan kaapattua järjestelmän tietoja. Karkeana erotuksena voidaan sanoa, että haavoittuvuustestauksella pyritään saamaan kokonaiskuva järjestelmän haavoittuvuuksista ja niiden vakavuudesta, kun taas penetraatiotestauksessa pyritään löytämään ne aukot, joita hyväksikäyttämällä voidaan tosiasiassa toteuttaa hyökkäys. (Nagpure ja Kurkure 2017).

Sovelluksiin kohdistuva penetraatiotestaus selvittää millä tavalla löytyneitä haavoittuvuuksia voidaan hyväksikäyttää, jotta ne mahdollistaisivat pääsyn sovellukseen tai sovelluksen tietoihin sekä palvelinta ylläpitävään järjestelmään. Penetraatiotestaus voidaan tehdä joko sovelluksen alustalta tai ulkopuolelta. Jälkimmäisessä nähdään paremmin mihin ulkopuolelta tunkeutuvat hyökkääjät kykenevät. (Shebli ja Beheshti 2018). Manuaalinen penetraatiotes-

taus on syväluotaava menetelmä, jolla ammattilaisten toimesta varmennetaan automaattisilla työkaluilla löydettyjä haavoittuvuuksia sekä mahdollisesti löydetään myös uusia haavoittuvuuksia (Kumar 2014).

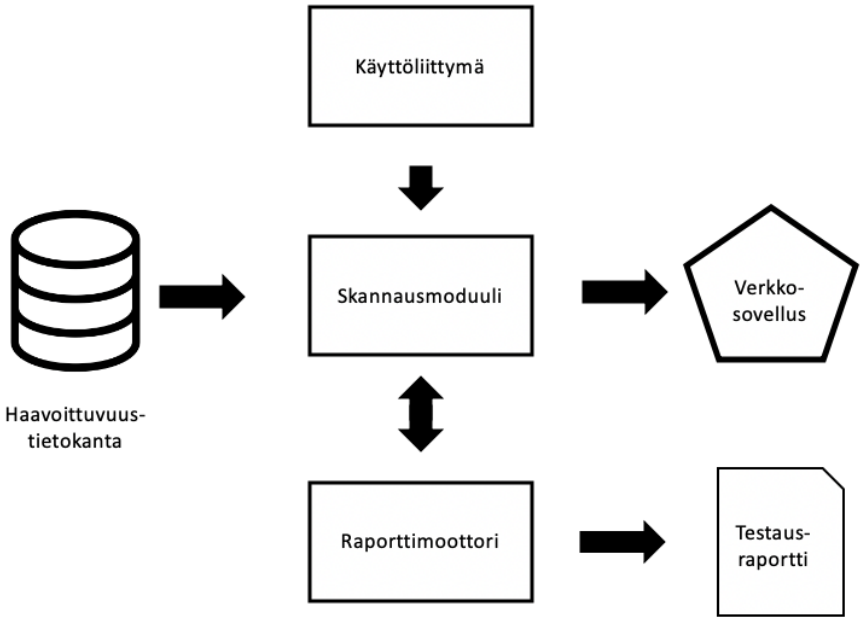
2.4.3 Dynaamiset skannaustyökalut

Dynaamiset skannaustyökalut ovat suuressa osassa haavoittuvuuksien löytämistä. Haavoittuvuusskannerit voidaan jakaa isäntä- ja verkkopohjaisiin (engl. *host- and network-based*) skannereihin (HKSAR 2008). Isäntäpohjaiset skannerit asennetaan isäntälaitteelle, esimerkiksi palvelimelle, ja ne skannaavat matalan tason tietoa, kuten esimerkiksi konfiguraatiotiedostoja. Isäntäpohjaiset skannerit tekevät tiedostojärjestelmään tarkastuksia, joihin verkkoskannerit eivät ulotu. (HKSAR 2008) Tässä tutkimuksessa ei syvennytä tähän skanneritYYppiin, sillä niillä saadaan paljon yksityiskohtaisempaa tietoa kuin mitä verkkosovellusten haavoittuvuuksien arviointiin yleensä sisältyy. Verkkopohjaisilla skannereilla tarkoitetaan sellaisia skannereita, jotka etsivät haavoittuvuuksia verkon kautta, esimerkiksi juuri verkkoon kytketystä palvelimesta (Bairwa, Mewara ja Gajrani 2014). Porttiskannerit ja haavoittuvuusskannerit kuuluvat molemmat tähän kategoriaan.

Porttiskannereilla ei itsessään voida löytää haavoittuvuuksia, mutta niillä voidaan selvittää palveluita, joita verkkosovelluksen palvelin tarjoaa. Palveluiden tietyissä versioissa voi olla haavoittuvuuksia, ja porttiskannauksella etsitään tietoa juuri tällaisista palveluista. (Skaggs ym. 2002). Usien myös varsinaiset haavoittuvuusskannerit sisältävät porttiskannerien ominaisuuksia (Christopher 2002). Porttiskanneri toimii yksinkertaisuudessaan siten, että sille syötetään skannattavan kohteen IP-osoite, jonka eri portteihin se lähettää paketteja. Vastausten perusteella päätellään, onko kyseinen portti auki eli tarjoaako se jotakin palvelua (Bairwa, Mewara ja Gajrani 2014).

Verkkosovellusten haavoittuvuusskannerit lähettävät pyyntöjä palvelimelle ja kuuntelevat vastauksia. Näitä vastauksia verrataan tietokantaan, jonka perusteella voidaan havaita tunnettuja haavoittuvuuksia. Haavoittuvuusskanneri koostaa havaituista haavoittuvuuksista raportin. (Makino ja Klyuev 2015). Yksinkertaistettu haavoittuvuusskannerin arkkitehtuuri on esitetty kuviossa 3. Yleisesti ottaen kaikkien skannerien toiminta koostuu skannausten ase-

tusten määrittämisestä, haavoittuvuusskannauksesta ja tulosten kokoamisesta. Skannauksen kesto vaihtelee kohteen suuruuden mukaan, ja esimerkiksi tuhannen IP-osoitteen skannaus voi kestää tunneista useisiin päiviin. (McMahon ym. 2018). Skannaustulosten epätarkkuus on myös yleistä, ja usein onkin järkevää käyttää useampia työkaluja eri konfiguraatioilla (El ym. 2017).



Kuvio 3. Haavoittuvuusskannerin arkkitehtuuri (Atymtayeva, Nurmyshev ja Tulemissova 2017, mukailtu).

2.4.4 Lähdekoodin analysointi

Kuten sovellusta testatessa, myös lähdekoodia voidaan testata automaattisesti tai manuaalisesti. Manuaalinen testaus tarkoittaa käytännössä koodin katselmointia, siis virheiden etsimistä ohjelmoijan ammattitaitoon perustuen. Automaattisen testauksen menetelmä on staattinen analyysi (engl. *static application security testing*, myös *static analysis*).

Staattinen analyysi tarkoittaa metodia, jossa sovelluksen lähdekoodista etsitään haavoittuvia osia, muun muassa ohjelmointivirheitä. Toisin kuin muissa esitetyissä metodeissa, staattisessa analyysissä nimensä mukaisesti ei ajeta ohjelmaa, eikä sen tarvitse olla käynnissä. Staattisessa analyysissä arvioidaan suoraan lähdekoodia, eikä ohjelmaa suoriteta missään vaihees-

sa. Staattinen analyysi on siis katsaus sovelluksen sisäisiin rakenteisiin eikä sen toimintaan. Sisäisiä rakenteita tutkimalla voidaan kuitenkin havaita haavoittuvuuksia, joita muut testausmenetelmät eivät löydä. Esimerkiksi harvoin suoritettavia ohjelmakoodin osia ei välttämättä havaita dynaamisessa testauksessa. (DuPaul 2013).

Staattista analyysia voidaan toteuttaa esimerkiksi vertaamalla ohjelmakoodia virhetietokantaan ja etsiämällä tunnettuja haavoittuvuuksia mahdollistavia ohjelmointivirheitä. Toinen vaihtoehto on analysoida ohjelmakoodia sääntöjen näkökulmasta, eli vastaako koodi tiettyjä ennalta määrättyjä sääntöjä. Tällaisia sääntöjä voivat olla esimerkiksi oikeuksien rajoittaminen siirryttäessä suorittamaan jotakin osaa ohjelmasta, joka ei tarvitse laajempia oikeuksia. (Li ja Cui 2010).

Staattisia analyysityökaluja on lukuisia, mutta niiden päällimmäinen heikkous on sidonnaisuus tiettyyn ohjelmointikieleen. Jokaiselle kielelle tarvitaan käytännössä oma työkalunsa (Li ja Cui 2010). Mikäli sovelluksessa käytetään useita kieliä, työkalujakin tarvitaan useampia. On myös olemassa analyysisovelluksia, jotka sisältävät usean eri ohjelmointikielen työkalut.

3 Tutkimuksen toteutus

Tässä luvussa kerrotaan tutkimuksen tarkempi toteututus. Alaluvussa 3.1 käsitellään tutkimuksen aiheen valintaa ja rajausta. Alaluvussa 3.2 esitellään tutkimuksen tavoite, tutkimusongelma ja tutkimuskysymykset, joiden avulla pyritään vastaamaan ongelmaan. Alaluvussa 3.3 kuvataan, miten teoreettisen viitekehyksen muodostus on toteutettu kirjallisuuskatsauksen keinoin. Alaluvussa 3.4 esitellään tutkimuksen menetelmäksi konstruktiivinen tutkimusote, ja kuvataan miten menetelmä sopii tähän tutkimukseen.

3.1 Aiheen valinta ja rajaus

Tutkimuksen aiheena on verkkosovellusten haavoittuvuuksien testaaminen. Aihe on valittu sen kiinnostavuuden ja ajankohtaisuuden vuoksi. Aiheeseen liittyvää tutkimusta on runsaasti, useimmiten tieteellisiä artikkeleita, joissa käsitellään uusia haavoittuvuuksia tai niiden havaitsemismenetelmiä. Tässä tutkimuksessa perehdytään tarkemmin haavoittuvuustestausprosessiin ja saadaan myös sitä kautta käytännön hyötyä verkkosovellukselle, johon testaus tehdään.

Aihe on rajattu käsittelemään otsikon mukaisesti haavoittuvuuksia verkkosovelluksissa. Tämä rajaa ulkopuolelle esimerkiksi haavoittuvuudet ohjelmistoissa, joita ei käytetä selaimen kautta sekä myös laitteistohaavoittuvuudet. Myöskään tietoliikenneprotokollien haavoittuvuuksiin ei oteta kantaa kuin siinä määrin, missä niillä on yhteinen rajapinta verkkosovellusten kanssa. Haavoittuvuuden määritelmään ei sisällytetä mahdollisia hallinnollisia tai organisatorisia prosesseja, vaan haavoittuvuus määritellään tämän tutkimuksen viitekehyksessä puhtaasti teknisenä ominaisuutena. Teoriaosuudessa käsitellään kaikentyyppisiä verkkosovelluksia, vaikka itse testaus tehdään sellaiseen avoimeen sovellukseen, joka ei mahdollista esimerkiksi kirjautumista, käyttäjän syötteen vastaanottamista tai muita yleisiä haavoittuvuuksia aiheuttavia tekijöitä. Näitä ei kuitenkaan ole mielekästä rajata teoriaosuuden ulkopuolelle, koska verkkosovellusten merkittävimmät haavoittuvuudet ovat yhtenäinen kokonaisuus. Sovelluksen ollessa kehitysvaiheessa on kaikkien haavoittuvuustyyppien tunnistaminen tärkeää, mikäli siihen päätetään lisätä uusia ominaisuuksia.

3.2 Tutkimuksen tavoite ja tutkimusongelma

Tutkimuksen tavoitteena on löytää toimintamalli, jota käyttämällä voidaan testata verkkosovelluksen haavoittuvuuksia ja tietoturvallisuutta. Tutkimuksessa esitetään erilaisia menetelmiä toteuttaa haavoittuvuustestaus ja arvioidaan niillä saatuja tuloksia eri näkökulmista. Menetelmien tueksi haetaan teoreettista tietoa tutkimusalan viimeaikaisesta kirjallisuudesta. Lopuksi verkkosovellukseen toteutetaan haavoittuvuustestaus ja arvioidaan, miten haavoittuvuudet voidaan minimoida ja siten parantaa sovelluksen tietoturvallisuutta.

Tutkimuksessa etsitään vastausta siihen, mitä haavoittuvuuksia kyseisestä verkkosovelluksesta löytyy ja miten toimenpiteitä niiden poistamiseksi voidaan tehdä. Täten tutkimuskysymyksinä ovat:

- Mitä haavoittuvuuksia verkkosovelluksesta löytyy?
- Miten sovelluksen tietoturvallisuutta voidaan parantaa?

Alakysymyksiä ovat:

- Mitkä ovat yleisimpiä ja merkittävimpiä verkkosovellusten haavoittuvuuksia?
- Miten verkkosovellusten haavoittuvuuksia voidaan testata?

Kysymyksiin vastataan keräämällä tietoa alan tieteellisistä julkaisuista ja tekemällä verkkosovellukseen kattava haavoittuvuustestaus.

3.3 Kirjallisuuskatsaus

Teoreettisen viitekehyksen materiaali kerättiin kirjallisuuskatsauksella. Materiaalin keräämisessä hyödynnettiin IEEE Xplore-, Semantic Scholar-, Google Scholar- ja Research Gate -julkaisuarkistoja. Lisäksi Google-hakukoneella tehtiin täsmähakuja kyseisiin palveluihin. Hakusanat on esitetty talukossa 13. Hakutulokset lajiteltiin relevanssin mukaan, ja luettavaksi otettiin muutamalta ensimmäiseltä hakutulossivulta otsikoiden tai tiivistelmien perusteella lupaavimmat julkaisut. On huomionarvoista, että osalla hakusanoista saatiin myös paljon muihin aihealueisiin kuin tietoturvallisuuteen liittyviä julkaisuja.

Tieteellisten artikkelien haussa käytettiin vain englanninkielisiä hakusanoja. Tuloksista rajat-

Taulukko 13. Hakusanat

Alaluku	Hakusana
2.1 Tietoturvallisuus	’information security’
	’confidentiality’
	’integrity’
	’availability’
2.2 Verkkosovellukset ja niiden haavoittuvuudet	’web application’
	’web application vulnerability’
	’vulnerability’
	’OWASP’
2.4 Tietoturvallisuuden arviointi- ja testausmenetelmiä	’vulnerability analysis’
	’vulnerability assessment’
	’vulnerability test’
	’vulnerability scanning’
	’penetration test*’
	’static analysis’
	’static application security test*’
	’whitebox test*’
	’blackbox test*’

tiin pääasiassa ulos ennen vuotta 2010 kirjoitetut artikkelit, koska tutkimuksen tarkoituksena on löytää nimenomaisesti tämän hetken merkittävimpiä haavoittuvuustestausmenetelmiä. Kuitenkin esimerkiksi tietoturvallisuuden käsitteenmäärittelyssä on käytetty huomattavasti vanhempia lähteitä määritelmän pysyvyyden vuoksi.

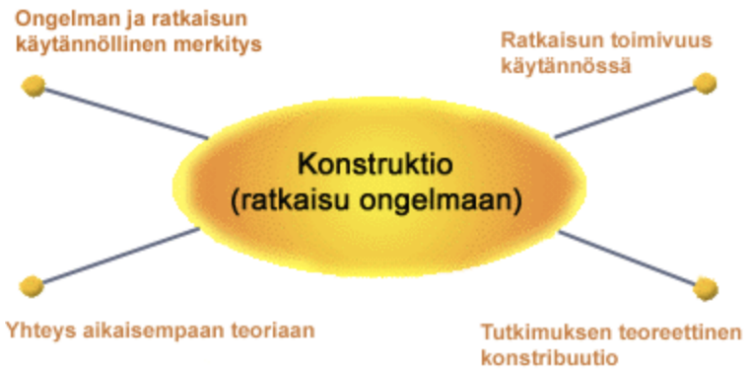
Lisäksi tietoa on haettu sellaisten organisaatioiden sivuilta, jotka ovat vahvasti kytköksissä aihealueeseen. Erityisesti verkkosovellusten protokollien määrittelyssä on tukeuduttu täysin IETF:n julkaisemiin RFC-dokumentteihin, jotka ovat standardeja vastaavassa asemassa. Haavoittuvuuksien luokittelua koskevassa kappaleessa viitataan pääosin MITRE:n verkkosi-

vustoille, koska kyseinen yritys on kehittänyt yleisesti käytettävät haavoittuvuuksien luokitteluun ja lajitteluun käytettävät menetelmät. Verkkosovellusten haavoittuvuuden nykytilaa selvitettäessä on käytetty tietoturvaorganisaatioiden vuosijulkaisuja niiltä osin, kun ne käsittelevät verkkosovelluksia. Verkkosovellusten merkittävimpiä haavoittuvuuksia kuvatessa on käytetty pääasiallisena lähteenä OWASP:n verkkosivustoa, sillä he tuottavat merkittävimpien verkkosovellusten luokittelussa käytettävää OWASP Top Ten -listausta. Lisäksi jokaista OWASP-haavoittuvuutta kohden on etsitty sen nimellä aiemmin mainituista tieteellisten lähteiden julkaisuarkistoista materiaalia.

Lisäksi lähdemateriaaleihin perehtyessä haettiin julkaisujen lähdeluetteloista lähteitä. Lähdeluetteloista löytyi usein relevanttia aineistoa, joka ei hakutuloksissa noussut esille hakusanojen muotoilun vuoksi.

3.4 Konstruktiivinen tutkimusote

Tutkimusmenetelmä on laadullinen tutkimus ja tutkimusotteeksi on valittu konstruktiivinen tutkimusote. Konstruktiivisessa tutkimusotteessa tarkoituksena on ratkaista reaalimaailman ongelmia teorian ja sen soveltamisen avulla. Konstruktiivisen tutkimusotteen keskeisinä elementteinä on reaalimaailmaan sovellettavuus, käytännönläheisyys ja konstruktion soveltamisyritys, linkitys aiempaan teoriaan ja teorian empiirinen reflektointi. (Lukka 2003)



Kuvio 4. Konstruktiivisen tutkimusotteen keskeiset elementit (Lukka 2014).

Kuviossa 4 esitetään konstruktiivisen tutkimusotteen keskeiset elementit. Tässä tutkimuksessa konstruktio, eli reaalimaailman ongelman ratkaisu on verkkosovelluksen haavoittuvuus-

testauksen malli, jota sovelletaan juuri tässä tutkimuksessa toteutettavaan haavoittuvuustestaukseen. Ongelma ja ratkaisu ovat kytköksissä reaalimaailmaan siten, että testattava sovellus on internetissä avoimesti saatavilla oleva sovellus, jonka tietoturvaa halutaan testata. Kehitettyä mallia voidaan soveltaa myös muihin vastaaviin sovelluksiin, mutta sellaisenaan se ei ole yleispätevä kaikkiin verkkosovelluksiin.

Konstruktiivisen tutkimuksen prosessi rakentuu seuraavista kokonaisuuksista (Lukka 2003):

1. Käytännön ongelma, jolla on potentiaalia teoreettiselle kontribuutiolle
2. Tutkimusyhteistyö kohdeorganisaation kanssa
3. Syvän ymmärryksen kehittäminen käytännöstä ja teoriasta
4. Kehitä ratkaisuidea ja kehitä konstruktio jolla teoreettista arvoa
5. Implementoi ja testaa ratkaisu
6. Pohdi ratkaisun sovellettavuuden laajuutta
7. Määritä ja analysoi teoreettien kontribuutio

Tutkimuksen käytännön ongelmana on verkkosovelluksen haavoittuvuuksien löytäminen ja parannusehdotusten antaminen. Tämä antaa merkityksellistä tietoa sovelluksen kehittäjille, mutta sillä on myös teoreettista kontribuutiota. Tutkimuksessa koostetaan prosessi sovelluksen tietoturvallisuuden arviointimenetelmäksi, jota voidaan hyödyntää lähes sellaisenaan sovelluksesta riippumatta, sillä myös työkalut ovat staattista analyysiä lukuunottamatta hyvin geneerisiä.

Tutkimuksessa on tehty yhteistyötä kehittäjien ja ylläpitäjien kanssa siten, että on saatu lupa testata sovellusta sellaisin menetelmin, jotka ilman lupaa olisivat laittomia (Korkein oikeus 2003). Lisäksi sovelluksen lähdekoodi on saatu käyttöön analysointia varten. Havaitut haavoittuvuudet raportoidaan tarkasti kohdeorganisaatiolle ja lisäksi ne esitetään yleisellä tasolla tässä tutkimuksessa.

Käytännöstä ja teoriasta on kehitetty syvällinen ymmärrys perehtymällä tieteelliseen kirjallisuuteen aiheesta sekä harjaantumalla teknisten työkalujen käytössä monipuolisesti. Ratkaisuna ja konstruktiona on prosessikuvaus ja havaittujen poikkeamien raportointi sekä kehitysehdotukset. Ratkaisun testaus tulee osana haavoittuvuuksien testaamista.

Ratkaisu on sovellettavissa laajemmin kuin tähän tutkimukseen, koska haavoittuvuustestausprosessi sopii mukailtavaksi eri kohteisiin. Vaikka itse testaustulokset eivät anna hyötyä muille kuin kohdeorganisaatiolle, nähdään niistä yleisesti minkälaisia haavoittuvuuksia tämäntyyppisissä sovelluksissa on. Teoreettinen kontribuutio tulee ensisijaisesti prosessin kuvaamisesta, mutta myös käytettyjen työkalujen toiminnan analysoinnista ja arvioinnista sekä kehitysehdotuksista.

4 Verkkosovelluksen haavoittuvuustestaus

Tässä luvussa käsitellään haavoittuvuustestausprosessia, tehdään sovellukseen testaus ja analysoidaan haavoittuvuustestauksen tulokset.

Alkuperäisiä testaustuloksia ei voida kokonaisuudessaan sisällyttää tutkielmaan, sillä ne käsittelevät yksityiskohtaista tietoa verkkopalvelusta ja sen toiminnasta. Nämä tiedot voivat vääriin käsiin päätyessään vaarantaa palvelun toiminnan. Tässä luvussa esitetyistä tuloksista on poistettu kaikki palvelua liiaksi yksilöivät tiedot.

Testaus toteutettiin palvelun ylläpitäjän luvalla kirjoittajan kotiverkosta. Sovellus on käynnissä virtuaalipalvelimelta, joka on ulkopuolisen palveluntarjoajan ylläpitämä. Tästä syystä testaus kohdistettiin pelkästään sovelluksen verkko-osoitteeseen, ei muuhun palvelinympäristöön. Koska verkkosovellus on vielä testausvaiheessa ja sitä ajetaan virtuaalipalvelimelta, haavoittuvuustestaus voitiin toteuttaa ilman, että siitä aiheutui erityistä haittaa palvelun tai ylläpitäjän toiminnalle.

Testauksessa käytettiin useita eri työkaluja ja käyttöjärjestelmiä. Testausten raportoinnin yhteydessä on kerrottu, mitä työkalua on käytetty ja kuvattu sen toiminta lyhyesti.

4.1 Testausprosessi

Ensimmäisessä vaiheessa kohteeseen tehtiin tiedonkeruuta, jonka tarkoituksena oli selvittää miten sovellus toimii, mikä on sen sijainti verkossa ja millaisia palveluita se tarjoaa. Tiedonhankinnan perusteella voitiin alustavasti selvittää haavoittuvuuksia ja rakentaa yleiskuvaa siitä, miten sovellus toimii. Tiedonkeruuvaiheeseen kuului reititystietojen selvittäminen, porttiskannaus ja haavoittuvuustietokannan läpikäynti palvelun versioiden osalta. Tiedonhankinnan toteutus on esitetty alaluvussa 4.2.

Toisessa vaiheessa kohteeseen tehtiin kattava haavoittuvuusskannaus usealla eri työkalulla. Lisäksi skannauksista saadut tulokset varmennettiin sovellusta manuaalisesti tarkastelemalla. Jokaisen työkalun kohdalla on myös analysoitu tulosten vaikutusta yleisesti verkkosovellusten turvallisuuteen. Haavoittuvuusskannauksen toteutus on esitetty alaluvussa 4.3.

Kolmannessa vaiheessa sovellukseen toteutettiin palvelunestohyökkäyksiä kolmella eri työkalulla. Palvelunestohyökkäysten tavoitteena oli häiritä palvelun toimintaa lähettämällä sille runsaasti erilaista liikennettä. Palvelunestohyökkäyksen toteutus ja tulokset on raportoitu alaluvussa 4.4.

Neljännessä vaiheessa sovelluksen lähdekoodia analysoitiin staattisen analyysin keinoin turvallisuusnäkökulmasta. Tarkoituksena oli löytää turvallisuuspuutteita, joita toisen vaiheen dynaamisilla skannereilla ei saada selville. Tulokset on esitetty alaluvussa 4.5.

Viidennessä vaiheessa kaikki havaitut puutteet koottiin yhteen ja luokiteltiin OWASP Top Ten -luokituksen mukaisesti. Lisäksi analysoitiin puutteiden vakavuutta ja merkitystä verkkosovelluksen tietoturvalle. Yhteenveto on esitetty alaluvussa 4.6.

Kuudennessa vaiheessa havaittujen puutteiden perusteella on esitetty parannusehdotuksia, joita tulisi tehdä sovelluksen tietoturvallisuuden parantamiseksi. Nämä parannusehdotukset käsitellään luvussa 5.

Viimeinen testausprosessin vaihe on raportointi, joka toteutetuu sekä tässä haavoittuvuustestausta käsittelevässä luvussa että parannusehdotuksia käsittelevässä luvussa 5. Kokonaisuudessaan testausprosessi on esitetty alla. Se noudattelee pääpiirteissään Shinde ja Ardhapurkar (2016) esittämää prosessia, vaikkakaan varsinaista penetraatiotestausta ei ole eriytetty. Varsinaisesti penetraatiotestausta tässä tutkimuksessa on vain palvelunestohyökkäys, eikä havaittuja puutteita ole yritetty hyödyntää järjestelmään murtautumiseksi.

1. Tiedonkeruu
2. Haavoittuvuusskannaus
3. Palvelunesto
4. Lähdekoodin analyysi
5. Tulosten koonti
6. Parannusehdotukset
7. Raportointi

4.2 Tiedonhankinta

Tiedonhankinnan tarkoituksena oli selvittää sovelluksen yksityiskohtaisia tietoja. Tässä selvitettiin, millaisia palveluita sovelluksen palvelin tarjoaa ja minkälaisia haavoittuvuuksia niistä löytyy palveluiden versionumeroiden perusteella. Lisäksi selvitettiin, millainen reitti testaajan verkosta on sovellukseen.

4.2.1 Traceroute

Aluksi tiedossa oli vain palvelun URL-osoite. Tiedonhankinta aloitettiin selvittämällä reitti kotiverkosta verkkosovelluksen palvelimelle. Traceroute on komentorivityökalu, jolla pyritään saamaan vastaus jokaiselta matkan varrelta olevalla reitittimeltä (Berkeley Distribution 2008).

Reitin selvitys tehtiin macOS-terminaalissa komennolla:

```
$ traceroute [palvelun URL-osoite]
```

Traceroute-sovelluksen tulokset on esitetty taulukossa 14, jossa yksilöivät IP-osoitteet on korvattu merkinnöillä A.A.A.A - F.F.F.F. Hypyllä tarkoitetaan sitä, monesko reititin on kyseessä testauskoneesta alkaen. Kirjoittajan kotireititin on rivillä 1 ja palveluntarjoajan ensimmäinen palvelin rivillä 9. Tästä eteenpäin reittiä ei pystytty määrittämään, koska sovellus on todennäköisesti suojattu palomuurilla, joka estää tai hävittää sille tulevat tämäntyyppiset pyynnöt. Kun reittipistettä ei ole voitu määrittää, on se merkitty kolmella tähdellä. RTT 1-3 tarkoittavat aikaa, joka testauskoneelta lähtevällä paketilla kestää saavuttaa kyseinen piste. Oleellista tulosten oikeellisuuden arvioinnissa on, ettei yhden reittipisteen arvot eroa merkittävästi toisistaan (InMotion Hosting 2020). Traceroute-tuloksilla ei tässä yhteydessä saatu merkityksellistä tietoa jatkotestausten kannalta.

4.2.2 Nmap

Testausta jatkettiin skannaamalla palvelimen portit, toisin sanoen selvittämällä mitä palveluita se kertoo ulospäin tarjoavansa. Skannaus toteutettiin yleisimmällä porttiskannaustyökalulla, Nmapilla. Palveluiden skannaamisen lisäksi selvitettiin niiden versiot ja palvelimen

Taulukko 14. Reitin selvitys palvelimelle

Hyppy	IP-osoite	RTT 1	RTT 2	RTT 3
1	192.168.8.1	6.170 ms	2.254 ms	1.382 ms
2	A.A.A.A	163.993 ms	176.257 ms	221.706 ms
3	B.B.B.B	29.013 ms	42.146 ms	37.913 ms
4	C.C.C.C	37.624 ms	40.698 ms	31.888 ms
5	D.D.D.D	36.389 ms	20.589 ms	31.937 ms
6	* * *			
7	E.E.E.E	52.592 ms		
8	* * *			
9	F.F.F.F	39.599 ms	33.118 ms	37.386 ms
10	* * *			

käyttöjärjestelmä. Porttiskannaus tehtiin Kali Linuxin terminaalisovelluksessa komennolla:

```
$ nmap -sV -O [palvelun IP-osoite]
```

Parametri `-sV` tarkoittaa palveluiden versionumeroiden selvittämistä ja `-O` kohteen käyttöjärjestelmän tunnistamista. Nmapin porttiskannauksen tulokset avoimien porttien osalta palveluiden versionumeroineen on esitetty taulukossa 15:

Taulukko 15. Porttiskannauksessa havaitut avoimet portit

Portti	Palvelu	Versio
21	FTP	vsftpd 3.0.3
22	SSH	OpenSSH 7.2p2
80	HTTP	nginx 1.10.3 (Ubuntu)
443	SSL/HTTP	nginx 1.10.3 (Ubuntu)
1723	PPTP?	-

Merkittävimmät palvelut, joita tarjotaan ovat skannauksen perusteella ovat tiedonsiirtoon käytettävä vsftpd, salattuun yhteydenmuodostukseen käytettävä OpenSSH sekä Nginx-palvelin.

4.2.3 Haavoittuvien versioiden hakeminen

Porttiskannauksen perusteella saatiin selville verkkosovelluksen palvelimella avoinna olevat palvelut versionumeroineen. versioita haettiin Exploit Database -haavoittuvuustietokannasta, jotta voitiin selvittää suoraan, onko niihin liittyen tunnistettuja haavoittuvuuksia. Taulukossa 16 on esitetty löydetyt haavoittuvuudet. Haavoittuvuudet liittyvät kaikki joko palvelunestohyökkäyksen mahdollistamiseen tai puutteelliseen käyttäjän todentamiseen ja pääsynhallintaan. Haavoittuvuuksia ei käsitellä tarkemmin tässä, sillä niitä ei kyetä varmentamaan, mutta mahdollisia keinoja niiden poistamiseen käsitellään alaluvussa 5.

Taulukko 16. Haavoittuvuustietokannasta löydetyt haavoittuvuudet

Palvelu	CVE	CVSS	Haavoittuvat versiot
nginx 1.10.3	CVE-2018-16843	7.8	< 1.15.6
	CVE-2018-16843	7.8	< 1.15.6
	CVE-2018-16845	5.8	< 1.15.6
openssh 7.2p2	CVE-2016-6515	7.8	< 7.3
	CVE-2015-8325	7.2	7.2p2
	CVE-2018-15919	5.0	< 7.8
	CVE-2017-15906	5.0	< 7.6
	CVE-2016-10708	5.0	< 7.4
	CVE-2016-6210	4.3	< 7.3

4.3 Haavoittuvuusskannaus

Haavoittuvuusskannaukset toteutettiin siten, että verkkosovellusta skannattiin useilla erityyppisillä työkalulla. Usean työkalun käyttö on perusteltua, koska yksittäiset tulokset eivät välttämättä ole tarkkoja. Ne voivat joko sisältää poikkeamia, joita ei todellisuudessa ole, tai ohittaa tunnettuja haavoittuvuuksia. Kukin skannaustyökalu toimii hieman eri tavalla. Osa työkaluista on komentorivipohjaisia, osa graafisen käyttöliittymän kautta käytettäviä sovelluksia ja osa selaimella käytettäviä verkkosovelluksia. Yhdistävänä tekijänä on se, että kaikista saadaan jonkinlainen raportti havaituista poikkeamista ja haavoittuvuuksis-

ta. Useimmat tässä testauksessa käytetyt työkalut luokittelivat haavoittuvuuksia kolmiportaiselle tasolle matalasta korkeaan. Osa työkaluista antoi lisäksi suoraan CVE-, CWE- tai CVSS-luokittelutietoa.

4.3.1 Nessus

Nessus on sovellus, jota voidaan käyttää verkkoselaimen tai komentorivin kautta. Nessus luokittelee haavoittuvuudet kolmeen luokkaan: korkea, keskitaso ja matala. Lisäksi on myös informatiivinen luokka, jonka poikkeamat eivät ole varsinaisia haavoittuvuuksia vaan tietoja, joita Nessus saa kerättyä sovelluksesta. Informatiivisessa luokassa voi kuitenkin olla myös sellaisia tietoja, jotka vaikuttavat sovelluksen turvallisuuteen. Lisäksi Nessuksen tuottamaan raporttiin tulevat tiedot haavoittuvuuksiin liittyvistä tunnisteista, kuten CVE-numerosta ja nimestä, mikäli sellaiset haavoittuvuudella on.

Nessus-skannaus kesti yhden tunnin ja 58 minuuttia. Nessus löysi 49 poikkeamaa, joista kaksi olivat keskitasoa ja 47 informatiivisia. Jälkimmäisiä ei analysoida tarkemmin, sillä ne tulevat oleellisilta osin käsiteltyä muiden skannereiden tuloksissa. Taulukossa 17 on esitetty havaitut poikkeamat. Havaitut keskitason poikkeamat olivat nimeltään SWEET32 ja BEAST. Molemmat liittyvät HTTPS-protokollassa käytettävän TLS/SSL-salauksen puutteisiin.

SWEET32-haavoittuvuudessa on kyse siitä, että TLS/SSL-salaukseen käytettävä algoritmi 3DES ei ole turvallinen. Kaappaamalla verkkoliikennettä tarpeeksi, salausalgoritmi alkaa toistaa itseään ja se voidaan murtaa. (Bhargavan ja Leurent 2016). BEAST-haavoittuvuus on vanhan TLS-version (1.0) haavoittuvuus, joka mahdollistaa selaimen kautta toteutettavan salauksen murtamisen (Duong 2011).

Haavoittuvuudet varmistettiin verkosta löytyvällä TLS/SSL-versioiden tarkastustyökalulla (Qualys 2020). Havaittiin, että sovellus käyttää TLS-versioita 1.0 ja 1.1, jotka ovat haavoittuvia. Chrome-selain tulee jatkossa näyttämään vain näitä TLS-versioita käyttävät sivustot turvattomiksi (Thompson 2019). Lisäksi todettiin, että sovellus käyttää heikkoa 3DES-salausalgoritmia. Nämä löydökset vahvistavat molempien haavoittuvuuksien olemassaolon. Sovelluksessa käytetään kuitenkin myös vahvempaa TLS 1.2 -versiota ja AES-salausalgoritmia.

Taulukko 17. Skannaustulokset: Nessus

Luokitus	Kuvaus	CVE	CVSS
Keskitaso	SWEET32	CVE-2016-2183	7.5
Keskitaso	BEAST	CVE-2011-3389	5.3

4.3.2 OpenVAS

OpenVAS on virtuaalikoneelle asennettava Linux-pohjainen järjestelmä. Varsinaista skanneria käytetään virtuaalikoneen tai isäntäkoneen verkkoselaimen kautta. OpenVAS:n skannaustuloksissa näytetään oletuksena vain korkeat ja keskitason haavoittuvuudet. Matalat ja informatiiviset suodatetaan pois, koska niiden hyödyntäminen vaatii syvällisempää tietoa sovelluksesta (Greenbone 2016). Korkeita ja keskitason haavoittuvuuksia löydettiin yhteensä 4. Skannaus testiin kahdella eri konfiguraatiolla, nopealla skannauksella ja syväskannauksella. Nopean skannauksen kesto oli 18 minuuttia ja syväskannauksen kesto oli kolme tuntia ja 11 minuuttia. Yhteensä löydettiin yksi korkean tason ja kolme keskitason haavoittuvuutta, jotka on esitetty taulukossa 18.

Taulukko 18. Skannaustulokset: OpenVAS

Luokitus	Kuvaus	CVE	CVSS
Korkea	Linksys Gozila CGI DOS	-	7.5
Keskitaso	FTP palvelunesto	-	5.0
Keskitaso	FTP salaamaton kirjautuminen	-	4.8
Keskitaso	SWEET32	CVE-2016-2183	5.0

Linksys-haavoittuvuudesta ei löytynyt paljoa tietoa avoimista lähteistä, mutta OpenVAS-skannausraportissa mainitaan, että reitittimeen voidaan kohdistaa palvelunestohyökkäys sen hallintapaneelin kautta. Koska tässä tutkimuksessa tarkoituksena on selvittää verkkosovellusten haavoittuvuuksia, sivuutettiin tämän haavoittuvuuden tarkempi tutkinta.

Toinen haavoittuvuus on yleisesti tunnettu tiedostonimiin liittyvä ongelma Windowsissa.

Windows-käyttöjärjestelmässä ei voi olla tietynnimisiä tiedostoja, koska ne on varattu muuhun käyttöön (Microsoft 2018). OpenVAS-skannausraportin perusteella oli mahdollista jäädyttää Windows-käyttöjärjestelmä lähettämällä FTP-protokollalla tällainen väärän niminen tiedosto. Tätä voidaan kuitenkin epäillä false positive -tulokseksi, sillä aiemman tiedonhankinnan perusteella palvelin on Linux-pohjainen, eikä sitä koske Windows-haavoittuvuudet.

Kolmas haavoittuvuus liittyi FTP-protokollan salaamattomaan kirjautumiseen. Käytännössä tämä tarkoittaa sitä, että kirjautuminen voidaan tehdä ilman, että tietoliikenne salataan. Tämä mahdollistaa esimerkiksi salasanojen kaappaamisen. Haavoittuvuus todennettiin ajamalla testauskoneella samanaikaisesti Wireshark-verkkoliikenteen pakettikaappaustyökalua ja yrittämällä kirjautua FTP-palvelimelle. Kirjautumista yritettiin macOS-terminaalissa komennolla:

```
$ ftp
ftp> open [palvelun URL-osoite]
```

Kirjautumiseen käytetty käyttäjätunnus ja salasana näkyivät selkokielisenä pakettikaappauksessa, mikä todensi löydetyn haavoittuvuuden.

Viimeinen haavoittuvuus on sama SWEET32-haavoittuvuus, joka havaittiin Nessuksella ja on kuvattu tarkemmin alaluvussa 4.3.1.

4.3.3 Arachni

Arachi on sovellus, jota voidaan käyttää komentorivillä tai selaimella. Arachnin haavoittuvuudet on luokiteltu kolmiportaiselle asteikolle. Erilaisia haavoittuvuuksia oli neljä kappaletta, joista kaksi keskitason ja kaksi matalan tason haavoittuvuuksia (taulukko 19). Skannaus kesti yhden tunnin ja 11 minuuttia.

Puuttuva `Strict-Transport-Security` -otsake tarkoittaa, että palvelin ei käytä *HTTP Strict Transport Security* (HSTS) -mekanismia. Tällöin käyttäjä voi halutessaan lähettää pyyntönsä selkokielisellä HTTP-protokollaa salatun HTTPS-protokollan sijaan. Vastausotsakkeen käyttöönotto pakottaa käyttäjän selaimen HTTPS-protokollaan, vaikka käyttäjä pyytäisi viestin välitystä HTTP-protokollalla. HTTP-protokollan salliminen voi vaarantaa tiedon

Taulukko 19. Skannaustulokset: Arachni

Luokitus	Kuvaus
Keskitaso	Puuttuva `Strict-Transport-Security` -otsake
Keskitaso	Yleinen hakemisto
Matala	Puuttuva `X-Frame-Options` -otsake
Malata	Turvaton `Access-Control-Allow-Origin` -otsake

luottamuksellisen tilanteissa, joissa verkkoliikennettä esimerkiksi kaapataan. (IETF 2012).

`X-Frame-Options` -otsakkeella säädellään, voiko sivuston sisältöä upottaa muille verkkosivuille. Jos upottaminen on sallittu, tämä mahdollistaa klikkausansojen (engl. *clickjacking*) tekemisen ulkopuolisille sivustoille. Esimerkiksi haittaohjelmien latauslinkkejä voidaan upottaa sivustolle näkymättömiin jonkin sellaisen kohteen päälle, jota käyttäjä haluaa klikata. Tällöin käyttäjä luulee klikkaavansa kohdetta, mutta todellisuudessa klikkaakin päällimmäisenä olevaa haitallista linkkiä. (IETF 2013).

`Access-Control-Allow-Origin` -otsake kertoo selaimelle, mistä lähteestä palvelimen resurssiin sallitaan pääsy. Lähteellä tarkoitetaan tässä yhteydessä sekä osoitetta että porttinumeroa. Esimerkiksi samassa URL-osoitteessa eri porteissa ajetut palvelut ovat eri lähteitä. Normaalisti selaimet sallivat JavaScript-koodin suorittaminen vain samasta lähteestä. Otsakkeessa voidaan sallia pääsy kaikista tai vain määritellyistä lähteistä. (W3C 2014).

Otsaketietoja tarkasteltiin Google Chrome -selaimen Network Inspector -työkalulla. Puuttuvat ja turvattomat otsakkeet raportoitiin pääasiassa juurisivulle, joten tarkastelu aloitettiin tästä. Todettiin, että juurisivun HTTP-vastauksesta puuttui Arachnin raportin mukaisesti tietyt otsaketiedot. Tarkastelua jatkettiin myös API:iin osoitteessa `/api`, jossa ilmeni yksi turvaton otsake. Tämän perusteella Arachni-skannaus on toiminut odotetusti ja puutteet ovat olemassa sovelluksessa.

X-Frame-Options-otsakkeen puutteellisuutta tarkasteltiin manuaalisesti upottamalla testattu verkkosovellus paikalliseen HTML-tiedostoon:

```
<iframe src="[palvelun␣URL-osoite]">
</iframe>
```

Verkkosivu onnistuttiin kokonaisuudessaan upottamaan toiselle sivulle, mikä mahdollistaa esimerkiksi klikkausansat toisilta sivuilta.

Viimeinen Arachnin havaitsema poikkeama oli yleinen hakemisto. Tällä tarkoitetaan sitä, että sovelluksen tuotantoversioon jää yleisten käytänteiden mukaisesti nimettyjä kansiorakenteita, joita ei kuitenkaan sovelluksessa käytetä. Arachni ilmoitti, että tällainen hakemisto oli osoitteessa `/doc`. Tätä hakemistoa yritettiin avata selaimella ja komentorivillä, mutta molemmissa tapauksissa palvelin ohjasi geneeriselle virhesivustolle. Network Inspectorilla havaittiin kuitenkin, että tässä tai muissakaan olemattomissa hakemistoissa ei vastata vakiintuneen käytänteen mukaisesti `HTTP 404` -statusviestillä, vaan `HTTP 200` -viestillä. Tällöin selain tai haavoittuvuusskanneri ei tunnista sitä, että kyseistä polkua ei todellisuudessa ole olemassa. Tämä oli siis turvallisuuden osalta false positive -tulos, vaikkakin paljasti hyvän käytänteen vastaisen toimintamallin (Wagner 2017).

4.3.4 OWASP ZAP

OWASP Zed Attack Proxy eli ZAP on Java-pohjainen sovellus, jota käytetään sovelluksen omalla graafisella käyttöliittymällä. ZAP luokittelee haavoittuvuudet kolmiportaiselle asteikolle ja informatiiviseen luokka Nessuksen tavoin. ZAP:lla löydettiin kaksi kesitason ja kolme matalan tason haavoittuvuutta (taulukko 20). Skannaustyökalu ei toiminut odotetusti, sillä se skannasi sivuja, joita ei ole rakenteessa. Tämä johtuu edellisessä alaluvussa kuvatusta `HTTP 404` -statuskoodin puuttumisesta. ZAP-skannaus pysäytettiin manuaalisesti, kun huomattiin sen tuntien jälkeen edelleen skannaavan sivuja, jotka vievät samalle virhesivustolle.

Content-Security-Policy-otsakkeella säädellään muun muassa sitä, mitä resursseja kyseinen sivu voi ladata (IETF 2016). Rajoittamalla resurssien lataamista voidaan estää esimerkiksi XSS-hyökkäyksiä (Mozilla 2019a). `X-Frame-Options` -otsakkeen puuttuminen

Taulukko 20. Skannaustulokset: OWASP ZAP

Luokitus	Kuvaus
Keskitaso	Puuttuva `X-Frame-Options` -otsake
Keskitaso	Puuttuva `Content-Security-Policy` -otsake
Matala	Puuttuva `Cache-Control` -otsake
Malata	Puuttuva `X-Content-Type-Options` -otsake
Malata	Puuttuva `X-XSS-Protection` -otsake

on käsitelty alaluvussa 4.3.3.

`Cache-Control` -otsakkeella voidaan rajoittaa välimuistiin tallentamista selaimelle esimerkiksi sellaisissa tilanteissa, jossa käsitellään käyttäjän sensitiivisiä tietoja. Jos sensitiivisiä tietoja käsittelevällä sivulla ei ole tätä otsaketta asetettu, voidaan välimuistista hakea aiempi sisältö, esimerkiksi luottokorttitietoja. (Saad, Meucci ja Mitchell 2020).

`X-Content-Type-Options`-otsake estää *MIME sniffing* -haavoittuvuudet, joita voidaan käyttää esimerkiksi XSS-hyökkäysten toteuttamiseen. Normaalisti otsakkeissa asetetaan tiedostomuoto `Content-Type` -otsakkeella. Tämän tiedon ollessa puutteellinen, selain voi yrittää selvittää tiedostomuotoa MIME sniffingillä. Kuitenkin haitallista ohjelmakoodia voidaan piilottaa tiedoston sisään, jolloin selain suorittaa koodin selvittäessään tiedostomuotoa. `X-Content-Type-Options` -otsakkeen käyttöönotto estää MIME sniffingin käytön selaimessa. (Mozilla 2016).

`X-XSS-Protection` -otsakkeella pyritään havaitsemaan ja estämään XSS-hyökkäyksiä. Nykyisin verkkosovelluksissa käytetään `Content-Security-Policy` -otsaketta, mutta vanhemmat selaimet eivät välttämättä tue sen käyttöä. (Mozilla 2019b).

4.3.5 Nikto

Nikto on komentorivipohjainen työkalu yksinkertaiseen skannaukseen. Skannaus kesti yhdeksän minuuttia. Niktolla ei onnistuttu löytämään uusia haavoittuvuuksia, joita ei olisi jo jollain muulla työkalulla havaittu (taulukko 21. Nikto-skannaus tehtiin Kali Linuxin terminaalisovelluksessa komennolla:

```
$ nikto [palvelun URL-osoite]
```

Taulukko 21. Skannaustulokset: Nikto

Kuvaus
Puuttuva `X-Frame-Options` -otsake
Puuttuva `X-XSS-Protection` -otsake
Puuttuva `X-Content-Type-Options` -otsake

4.4 Palvelunesto

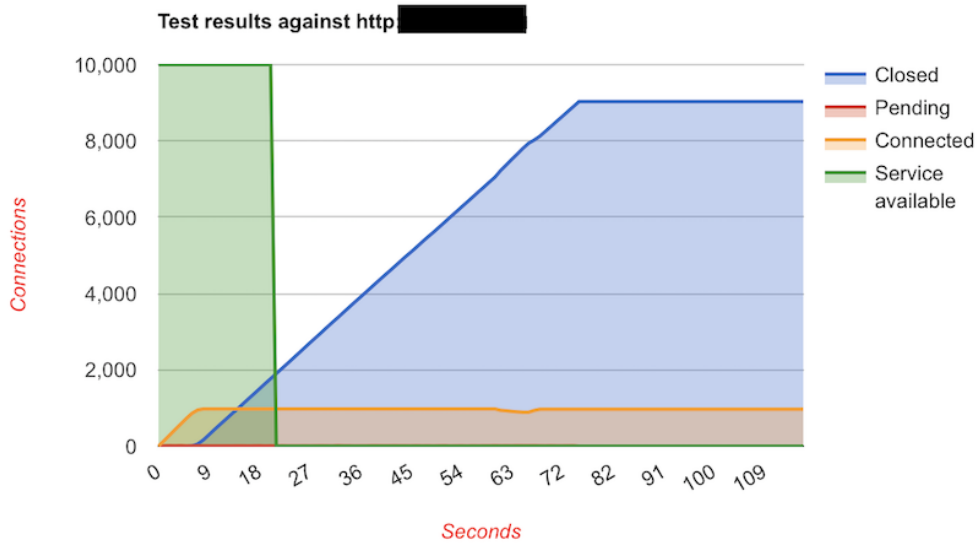
Palvelunestohyökkäystä yritettiin kolmella eri sovelluksella, slowhttptest, LOIC ja owasp-dos-http-post.

Slowhttptest-palvelunestohyökkäys tehtiin Kali Linuxin terminaalisovelluksessa komennolla:

```
$ slowhttptest -c 10000 -H -g -o ./output_file -i 20 -r 200
-t GET -u [palvelun URL-osoite] -x 24 -p 2
```

Annetuilla parametreilla tarkoittaan, että kohteeseen yritetään muodostaa 10 000 erillistä yhteyttä. Hyökkäys toteutetaan lähettämällä puutteellisia `GET`-pyyntöjä 20 kertaa sekunnissa ja kymmenen sekunnin välein `POST`-pyyntöjä. Muut parametrit liittyvät tulosten tallentamiseen tiedostoon.

Kuviossa 5 on esitetty, miten slowhttptest-sovelluksessa toteutettu hyökkäys toimii. Todellisuudessa palvelu ei lakannut toimimasta missään vaiheessa, joten lienee todennäköistä, että kyseiset paketit suodattuivat pois ja siksi yhteys näyttää katkenneelta. Myöskään LOIC- tai owasp-dos-http-post -työkaluilla ei saatu palvelua häirittyä, joten on perustelua arvioida,



Kuvio 5. Palvelunestohyökkäyksen vaikuttavuus slowhttptest-sovelluksella testattuna

että palveluntarjoajan palomuuri suodattaa hyvin tehokkaasti samasta IP-osoitteesta tulevat lukuisat pyynnöt.

4.5 Staattinen analyysi ja koodin katselmointi

Testausta varten saatiin pääsy verkkosovelluksen ohjelmakoodiin GitLabissa, josta se ladattiin testauskoneelle. Sovelluksen frontend on toteutettu Reactilla ja backend Expressillä. Express on ohjelmistokehys (engl. *software framework*), joka on toteutettu Node.js-alustaa käyttäen. Staattisen analyysin työkaluja etsittiin erikseen backendille ja frontendille, vaikka molemmat on toteutettu JavaScriptillä. Backendin skannaukseen päädyttiin käyttämään NodeJSScan-työkalua, koska se on tarkoitettu nimenomaisesti Nodea käyttävien sovellusten testaamiseen. Reactille ei löydetty omaa testaussovellusta, joten frontendin osalta päädyttiin käyttämään usealle eri ohjelmointikielelelle soveltuvaa testausalustaa SonarQubea ja sen staattisen analyysin työkalua SonarScanneria (Sonarqube 2019).

4.5.1 Backend

NodeJSScan ladattiin ja käynnistettiin konttityökalu Dockeria käyttäen macOS:n terminaalissa komennolla:

```
$ docker pull opensecurity/nodejsscan
$ docker run -it -p 9090:9090 opensecurity/nodejsscan:latest
```

Tämän jälkeen selaimella navigoitiin osoitteeseen `0.0.0.0:9090`, jossa oli sovelluksen web-käyttöliittymä. Lähdekoodi ladattiin `.zip`-tiedostomuodossa sovellukseen ja käynnistettiin skannaus. Skannaus kesti noin minuutin ja tuloksena saatiin raportti lähdekoodin puutteista ja virheistä. Raportissa esitettiin seitsemän puuttuvaa HTTP-otsaketta, mutta varsinaisia ohjelmointivirheitä ei havaittu. Tätä selittää ensisijaisesti se, että sovelluksessa ei lueta syötettä käyttäjältä, jolloin myös haavoittuvuuspinta-ala on pienempi.

Express tarjoaa valmiin työkalun nimeltä Helmet, jota voidaan käyttää otsakkeiden turvalliseen asettamiseen (Helmet 2020). Kuten haavoittuvuusskannauksissa havaittiin, osoitteessa `/api` otsakkeet oli asetettu pääosin oikein. Kun tarkasteltiin lähdekoodia `/src/index.js`-tiedostossa, havaittiin että Helmet oli käytössä oletusasetuksilla.

Taulukossa 22 on esitetty analyysin tulokset.

Taulukko 22. Skannaustulokset: NodeJSScan

Kuvaus
Puuttuva `X-Frame-Options` -otsake
Puuttuva `X-Content-Type` -otsake
Puuttuva `Strict-Transport-Security` -otsake
Puuttuva `Public-Key-Pins` -otsake
Puuttuva `X-Download-Options` -otsake
Paljastava `X-Powered-By` -otsake
Puutteellinen `Set-Cookie` -otsake

`Public-Key-Pins` -otsake on vanhentunut turvallisuuskäytänne, joka ei ole enää käytössä moderneissa selaimissa (Mozilla 2020b). Tämän osalta havainnon voidaan katsoa olevan false positive.

`X-Download-Options` -otsake on Internet Explorer 8 -selaimen otsake, jolla on säädelty ladattavien HTML-tiedostojen käsittelyä. (Klingsgeim 2020). Tämän otsakkeen turvallisuusvaikutuksia ei käsitellä tarkemmin, koska IE8 on nykyisin harvinainen selain. (W3Counter 2020).

`X-Powered-By` -otsake on standardoimaton otsake, jolla voidaan välittää tietoa palvelimella käytettävästä teknologiasta. Otsakkeen käyttö voi paljastaa palvelimen tarkan version, jolloin haavoittuvuuksien etsiminen on hyökkääjän näkökulmasta helpompaa. (Mozilla 2020a).

`Set-Cookie` -otsaketta käytetään evästetietojen välittämiseen. (Mozilla 2020a). NodeJSScan havaitsi `httpOnly`-arvon puuttuvan kyseiseltä otsakkeelta, joka voi johtaa XSS-hyökkäyksiin tai evästetietojen vuotamiseen. `httpOnly`-arvo estää evästeiden tietojen lukemisen automatisoidulla skriptillä. (Microsoft 2008).

4.5.2 Frontend

SonarQube ladattiin ja käynnistettiin Dockeria käyttäen macOS-terminaalissa komennolla:

```
$ docker pull sonarqube:latest
$ docker run -d --name sonarqube -p 9000:9000 -p 9092:9092 sonarqube
```

Tämän jälkeen selaimella navigoitiin osoitteeseen `localhost:9000` ja kirjauduttiin sisään oletuskäyttäjätunnuksella `admin:admin`. SonarQubeen luotiin uusi projekti nimeltä `TESTAUS`. SonarScanner käynnistettiin macOS-terminaalissa komennolla:

```
$ sonar-scanner \
  -Dsonar.projectKey=TESTAUS \
  -Dsonar.sources=. \
  -Dsonar.host.url=http://localhost:9000 \
  -Dsonar.login=[projektin token]
```

Analyysi kesti noin puoli minuuttia, jonka jälkeen tulokset olivat tarkasteltavissa. Varsinaisia turvallisuushaavoittuvuuksia ei havaittu. SonarQube kuitenkin antoi kahdeksan manuaalisesti tarkastettavaa turvallisuuteen liittyvää huomiota koodista. Huomiot eivät välttämättä tarkoita haavoittuvuuksia, sillä ne ovat kohtia koodissa, joissa haavoittuvuuksia voi yleises-

ti esiintyä. Todelliset vaikutukset sovelluksen turvallisuuteen on selvitettävä manuaalisella tarkastelulla. Kaksi analyysin tuottamaa huomiota olivat korkean prioriteetin kohteita, jotka liittyvät injektioihin. Loput kuusi huomiota olivat keskitason prioriteetin kohteita ja koskivat mahdollista palvelunestoa.

Injektiouhka liittyi ohjelmakoodiin tiedostossa `argv.js`, jossa luetaan komentoriviargumentti:

```
module.exports = require('minimist')(process.argv.slice(2));
```

Tätä moduulia käytetään myöhemmin tiedostossa `index.js`:

```
const argv = require('./argv');
...
const customHost = argv.host || process.env.HOST;
const host = customHost || null;
...
app.listen(port, host, (err) => {
  if (err) {
    return logger.error(err.message);
  }
...
}
```

Yksinkertaistettuna ohjelmakoodi lukee komentoriviltä syötteenä osoitteen, josta sovellus käynnistetään. Oletuksena käytetään paikallista localhost-osoitetta, mikäli syötettä ei anneta. Turvallisuuden näkökulmasta tässä ei ole todellista uhkaa injektiolle, sillä käyttäjillä ei ole pääsyä sovelluksen käynnistämiseen tarkoitettuun ohjelmakoodiin. Injektion mahdollisuus oli kuitenkin tarkastettava, koska joissain sovelluksissa käyttäjältä voidaan lukea palvelimella sellaisenaan suoritettavaa syötettä. Injektioiden osalta sovellus katsottiin turvalliseksi.

Palvelunestohaavoittuvuus liittyi säännöllisiin lausekkeisiin. Näitä evaluoimalla voitaisiin kuluttaa palvelimen resursseja paljon, jos lauseke on rakennettu siten että sen laskeminen kestää pitkään. Tässä tapauksessa käyttäjän syötettä ei evaluoida, vaan säännölliset lausekkeet liittyvät käyttöliittymän sisäisten kieliasetusten määrittämiseen.

Koska SonarQube on laadunvarmistustyökalu, se havaitsi koodissa myös erilaisia huonoja

käytänteitä, kuten käyttämättömiä komponenttien tuonteja (engl. *component imports*). Näitä ei kuitenkaan käsitellä tässä tutkimuksessa, sillä niillä ei ole merkittäviä turvallisuusvaikutuksia. Kokonaisuudessaan frontendin osalta ei havaittu turvallisuuteen viittaavia puutteita.

4.6 Havaintojen koonti ja luokittelu

Pääosa havaituista puutteista kohdistui otsakkeisiin. Näiden puuttuminen tai virheelliset määritykset varmistettiin Network Inspector -työkalulla manuaalisesti. Puutteet koskivat suoraan verkkosovelluksen käyttöliittymää juuriosoitteessa, vaikka sen alapuolella rakenteissa, esimerkiksi ohjelmointirajapinnassa, otsakkeet olivat pääosin turvallisia.

Väärin konfiguroidut tai puuttuvat otsakkeet luokitellaan OWASP Top Ten -listauksessa luokkaan virheellinen turvallisuuskonfiguraatio (alaluku 2.3.6). Tässä tapauksessa puuttuvat otsakkeet voivat mahdollistaa sen käyttäjiin kohdistuvat hyökkäykset, kuten XSS:n, klikkausansat tai sensitiivisen tiedon kaappaamisen. Varsinaisesti sovelluksen toimintaa tai tietoja ne eivät vaaranna.

FTP-tiedonsiirtoprotokollaa eivät yleensä käytä peruskäyttäjät, vaan ylläpitäjät siirtääkseen tiedostoja palvelimelle. Tässä yhteydessä puutteet protokollan salauksessa voivat tosiasiallisesti vaarantaa myös palvelun toimintaa paljastamalla ylläpitäjän tunnuksen ja sitä kautta muita arkaluonteisia tietoja (alaluku 2.3.3). Mikäli ylläpitäjän tunnuksilla kirjaudutaan salaamattoman yhteyden yli, voivat tunnukset vuotaa kaappaajille. Lisäksi haavoittuvien TLS/SSL-versioiden käyttö ei ole suositeltavaa, koska ne ovat murrettavissa. Salauksen puutteet luokitellaan puutteelliseksi käyttäjän todentamiseksi (alaluku 2.3.2).

Hakemalla palveluiden versionumeroita haavoittuvuustietokannasta, havaittiin haavoittuvuuksia monissa palveluissa. Nämä luokitellaan haavoittuvien komponenttien käytöksi (alaluku 2.3.9. Näitä haavoittuvuuksia ei yritetty hyödyntää tässä testauksessa, mutta niiden olemassaolo on todennäköistä versionumeroihin perustuen.

Taulukossa 23 on lajiteltu sekä skannereilla että manuaalisesti havaitut haavoittuvuudet OWASP Top Ten -luokkiin ja false positive -tulokset on poistettu. Yli puolet haavoittuvuuksista luokitellaan virheelliseksi turvallisuuskonfiguraatioksi, mitä selittää puutteellisten otsakkeiden

suuri määrä. Haavoittuvien komponenttien käyttöä on seuraavaksi eniten, mitä puolestaan selittää vanhojen versioiden käyttö. Käyttäjän todentamiseen ja arkaluonteisen tiedon paljastumiseen liittyvät haavoittuvuudet tulevat vanhentuneiden salausmenetelmien käytöstä.

Taulukko 23. OWASP Top Ten -luokitellut haavoittuvuudet

Haavoittuvuus	Määrä
Virheellinen turvallisuuskonfiguraatio	11
Puutteellinen käyttäjän todentaminen	2
Arkaluonteisten tietojen paljastuminen	1
Haavoittuvien komponenttien käyttö	2
False positive	3

Koonti kaikista havainnoista skannaustyökaluittain on esitetty taulukossa 24. Arvioidut false positive -tulokset on esitetty merkinnällä (F), ja muut todelliset haavoittuvuudet skannaustyökalun antaman luokituksen mukaan: informatiivinen (I), matala (L), keskitaso (M) ja korkea (H). Niktossa ja NodeJSScan:ssa haavoittuvuuksia ei erikseen luokitella, joten niiden osalta merkintä on X. On huomionarvoista, että samat haavoittuvuudet voidaan luokitella eri työkaluilla eri vakavuusluokkaan. Esimerkiksi Nessus luokittelee kaikki otsakkeisiin liittyvät haavoittuvuudet informatiivisiksi, kun taas muissa skannereissa ne ovat vähintään matalan tason, osassa jopa keskitason haavoittuvuuksia. Muilta osin luokitteluperusteita on vaikea arvioida, sillä SWEET32 oli ainoa haavoittuvuus, joka ei liittynyt otsakkeisiin ja havaittiin useammalla kuin yhdellä skannerilla.

Taulukko 24. Haavoittuvuusskannereiden kootut tulokset

Haavoittuvuus	Nessus	OpenVAS	Arachni	OWASP ZAP	Nikto	NodeJSScan
SWEET32	M	M				
BEAST	M					
Linksys Gozila CGI DOS	(F)					
FTP palvelunesto		(F)				
Salaamaton FTP		M				
Yleinen hakemisto			(F)			
Puuttuva `Content-Security-Policy`	I			M		
Puuttuva `Strict-Transport-Security`	I		M			X
Puuttuva `X-Frame-Options`	I		L	M	X	X
Turvaton `Access-Control-Allow-Origin`			L			
Puuttuva `X-Content-Type-Options`				L	X	X
Puuttuva `Cache-Control`				L		
Puuttuva `X-XSS-Protection`				L	X	
Puuttuva `Public-Key-Pins`						(F)
Puuttuva `X-Download-Options`						(F)
Paljastava `X-Powered-By`						X
Puutteellinen `Set-Cookie`						X

5 Verkkosovelluksen tietoturvallisuuden parantaminen

Tässä luvussa esitetään ratkaisuja, miten havaittuja haavoittuvuuksia ja järjestelmiä yleensä tulisi kehittää haavoittuvuuksien korjaamiseksi ja välttämiseksi. Jo alkuun on huomioitava, että pelkkä haavoittuvuuksien löytäminen ei ota kantaa siihen, aiheuttaako se tietoturvariskin kyseiselle sovellukselle. Tutkimuksessa testatussa sovelluksessa ei esimerkiksi lueta käyttäjältä tekstimuotoista dataa, jolloin injektioiden hyökkäyspinta-ala vähenee. Lisäksi sovelluksen kehittäjien on itse arvioitava, missä määrin halutaan varautua riskeihin ja minkä riskien kanssa tullaan toimeen. Tässä esitetyt parannusehdotukset pyrkivät korjaamaan kaikki haavoittuvuudet, kuitenkin siten, että arvioidaan myös mahdollisten korjaustoimenpiteiden laajuutta käyttötarkoitukseen nähden.

5.1 Versiopäivitykset

Pitämällä sovelluksen komponentit ajan tasalla voidaan välttyä tunnetuilta haavoittuvuuksilta. Vaikka aiemmin mainitusti nollapäivähaavoittuvuuksia hyödynnetään yhä enemmän, se ei poista riskiä siitä, että hyökkääjät käyttävät hyväksi myös tunnettuja haavoittuvuuksia ja etsivät niitä käyttäviä palveluita aktiivisesti.

Porttiskannauksella havaittiin, että Nginx ja OpenSSH käyttävät haavoittuvia versioita. Vaikkei näitä palveluita välttämättä käytetä aktiivisesti, on niiden pitäminen ajan tasalla tärkeää. Käyttämällä hyväksi vanhojen versioiden haavoittuvuuksia voidaan päästä sisään kohteeseen tai aiheuttaa muuta vahinkoa. Versiopäivityksiä tehtäessä on kuitenkin perehdyttävä muutoksiin ja arvioitava, tuleeko uuden version kanssa mahdollisia yhteensopivuusongelmia, jotka voivat vaarantaa saatavuuden.

Käytetty Nginx-versio 1.10.13 on julkaistu tammikuussa 2017. Nginx-palvelimeen julkaistaan päivityksiä tiheästi kerran kuukaudessa. Viimeisin versio 1.17.10 on huhtikuulta 2020. Ylläpitäjää suositellaan päivittämään Nginx uusimpaan versioon, sillä versiohistorian perusteella lähes kaikissa päivityksissä on korjattu joitain ohjelmointivirheitä (nginx 2020).

Käytetty OpenSSH-versio 7.2p2 on julkaistu maaliskuussa 2016. Päivityksiä on julkaistu

kaksi kertaa vuodessa ja viimeisin versio 8.2 on helmikuulta 2020 (OpenSSH 2020). Myös tämä kehotetaan päivittämään uusimpaan versioon yhteensopivuus huomioiden.

Vsftpd:n käytetty versio 3.0.3 on julkaistu heinäkuussa 2015 ja uudempaa versiota ei kirjoitushetkellä ole saatavilla (Evans 2020). Versioon ei kuitenkaani liity tunnettuja haavoittuvuuksia.

5.2 Puutteet salauksessa

Salauksessa havaitut puutteet johtuvat vanhojen versioiden käytöstä. Yleensä vanhoja versioita käytetään, jotta yhteensopivuus vanhempien selaimien kanssa säilyy. Hyökkääjät voivat kuitenkin pakottaa palvelimen viestimään heikommalla salauksella, jolloin se on helpommin murrettavissa. Tässä sovelluksessa puutteina olivat SSL/TLS-protokollan heikko salausalgoritmi 3DES ja vanhentuneen, turvattoman TLS 1.0 -version käyttö. Nämä mahdollistavat SWEET32- ja BEAST-hyökkäykset, joissa salaus voidaan murtaa. Esimerkiksi OpenSSL ei ole vuoden 2016 versiopäivityksen jälkeen sisällyttänyt 3DES-algoritmia tuotteeseensa (Salz 2016) ja Google Chrome näyttää jatkossa turvattomina verkkosivut, jotka käyttävät TLS 1.0- tai 1.1-versioita (Thompson 2019).

Koska sovellukseen on kuitenkin konfiguroitu myös vahvempien salausten käyttö, kehotetaan poistamaan vanhemmat ja turvattomat versiot käytöstä. AES-salausalgoritmilla suositellaan korvattavan 3DES kokonaan. Lisäksi suositellaan siirtymään käyttämään vain turvallisempia TLS 1.2- ja 1.3-versioita. On kuitenkin huomioitava, että vanhempien versioiden poistaminen voi aiheuttaa ongelmia osalle käyttäjistä, jotka käyttävät vanhoja käyttöjärjestelmiä ja selaimia.

5.3 Turvalliset otsakkeet

Haavoittuvuusskannauksissa ilmeni, että yhdeksän otsaketietoa on asetettu turvattomiksi. Näiden merkitys on käsitelty tarkemmin alaluvuissa 4.3.3 ja 4.3.4. Tässä alaluvussa annetaan suosituksia, miten palvelun otsakkeita voidaan asettaa turvalliseksi modernit käytänteet huomioiden. Suositukset pohjautuvat Mozillan ja OWASP Secure Headers -projektin tietoihin

kyseisistä otsakkeista (Mozilla 2020a; OWASP Foundation 2020). Taulukkoon 25 on koottu tämän alaluvun suositukset.

`Access-Control-Allow-Origin`-otsakkeen avulla voidaan asettaa rajoituksia, mistä lähteistä voidaan päästä palvelun resursseihin. Ilman määrityksiä pääsy on vain samasta lähteestä. Kaikista lähteistä tulisi sallia pääsy vain silloin, kun kyseessä on julkinen ohjelmointirajapinta. Palvelun osoitteessa `/api` otsake oli määritelty siten, että pääsy kaikista sallittiin kaikista lähteistä. Mikäli kyseessä on täysin julkinen rajapinta, tämä on hyväksyttyä eikä vaadi muutoksia. Muussa tapauksessa otsake voidaan säätää sallimaan pääsy vain yksittäisestä sivusta, tai ottaa se pois, jolloin oletuksena pääsyä ei ole kuin samasta lähteestä. Juurisivustolla tätä otsaketta ei ole asetettu.

`X-Frame-Options`-otsakkeella voidaan estää sivun upottaminen toisille sivuille. Kyseistä otsaketta ei oltu asetettu, jolloin sivu pystyttiin upottamaan mille tahansa toiselle sivulle. Jos upotus halutaan sallia samasta lähteestä, voidaan käyttää arvoa `SAMEORIGIN`. Upotus voidaan kokonaan estää asettamalla arvoksi `DENY`. Mikäli upotus halutaan sallia tietyillä sivuilla, tulee se säätää otsakkeella `Content-Security-Policy`.

`Content-Security-Policy` -otsakkeella voidaan säätää monia sivuston ominaisuuksia. Tässä merkittävä on `frame-ancestors` -arvo, jolla voidaan sallia upottaminen vain tietyiltä sivuilta. Esimerkiksi arvo `frame-ancestors example.com` sallii, että sivustolle `example.com` voidaan upottaa tämän sivun sisältöä, mutta millekään muulle sivustolle ei.

`Strict-Transport-Security` -otsakkeella kerrotaan selaimelle, ettei sivustoa tulisi ikinä ladata salaamattoman HTTP-protokollan yli. Otsake hyväksyy arvot `max-age` ja `includeSubdomains`, joilla annetaan aika, kuinka kauan selain muistaa tämän tiedon sekä mikäli säännön halutaan sisältävän myös sivuston alaosoitteet, kuten `/api`.

`X-Content-Type-Options` -otsakkeella voidaan estää automaattinen tiedostomuodon selvittäminen. Tämä tehdään asettamalla otsakkeen arvoksi `nosniff`. Tämän jälkeen selain ei selvitä tiedostomuotoa, jolloin on tärkeää, että tiedostomuodon kertova `Content-Type` otsake on asetettu oikein.

`X-XSS-Protection` -otsakkeen määrittämisen jälkeen selain lopettaa sivun lataamisen, mikäli se havaitsee XSS-hyökkäyksen. Otsakkeen arvo `1` ottaa käyttöön suodattamisen ja sanitoi sivun sisällön havaitessaan hyökkäyksen. Lisäksi voidaan asettaa arvo `mode=block`, joka estää sivuston lataamisen sanitoinnin sijaan. XSS-hyökkäyksiä voidaan rajoittaa myös `Content-Security-Policy` -otsakkeella.

`Cache-Control` -otsakkeella säädellään sovelluksen välimuistiin tallentamista. Tällä hetkellä arvo on säädetty `public, max-age=0`, jolla tarkoitetaan, että sivuston sisältö tallennetaan aina välimuistiin ja ladataan sieltä, ellei se ole muuttunut. Koska sovelluksessa ei käsitellä arkaluonteisia tietoja, tämän otsakkeen käyttö asetetuilla arvoilla on hyväksyttävää.

`X-Powered-By`-otsake kertoo, millä teknologialla palvelin on toteutettu. Tässä tapauksessa otsakkeen arvo on `Express`, joka kertoo, että palvelin on ohjelmoitu käyttäen JavaScript-pohjaista Express.js-kirjastoa. Vaikka tämän tiedon paljastaminen ei ole merkittävä uhka tietoturvallisuudelle, voi se kertoa tavanomaiselle hyökkääjälle tarvittavia tietoja sovelluksesta. Tämä otsake suositellaan poistettavaksi.

`Set-Cookie`-otsakkeella asetetaan eväste ja määritetään sen tietoja. Otsakkeelle tulisi määrittää `httpOnly`-arvo, jolloin istunnon tiedot ovat paremmin suojassa erilaisilta hyökkäysyrityksiltä.

Skannereiden tulosten lisäksi sovellukseen tehtiin manuaalista tarkastamista, jonka perusteella havatiiin vielä yksi turvallisuuteen vaikuttava otsake. `Referrer-Policy` -otsake määrittää, mitä tietoja välitetään `Referer`-otsakkeessa siirryttäessä toiselle sivulle esimerkiksi linkin välityksellä. Pahimmassa tapauksessa ulkopuoliselle sivustolle voidaan välittää tiedot, jotka sisältävät arkaluonteisia tietoja esimerkiksi URL-osoitteen parametreinä. Otsake tulisi säätää siten, että toiselle sivustolle välitetään vain lähdetiedot, eli sovelluksen osoite, mutta ei parametrejä. Tämä voidaan asettaa `strict-origin` -arvolla, joka lisäksi estää referer-tietojen välittämisen salaamatonta HTTP-protokollaa käyttäen.

Verkkosovelluksessa oli käytössä turvallisten evästeiden asettamiseksi käytetty Helmet oletusasetuksilla. Suositellaan, että oletusasetuksen lisäksi säädetaan tarkemmin yksittäisten otsakkeiden turvallisuutta. Suurin osa tässä luvussa mainituista otsakkeista voidaan asettaa Helmetillä.

Taulukko 25. Suositellut otsaketiedot sovellukselle

Otsake	Arvo
`Access-Control-Allow-Origin`	`*`
`X-Frame-Options`	`SAMEORIGIN`
`Content-Security-Policy`	`frame-ancestors example.com`
`Strict-Transport-Security`	`max-age=15552000;` `includeSubDomains`
`X-Content-Type-Options`	`nosniff`
`X-XSS-Protection`	`1; mode=block`
`Cache-Control`	`public; max-age=0`
`Referrer-Policy`	`strict-origin`
`Set-Cookie`	evästeen tiedot; `httpOnly`
`X-Powered-By`	suositellaan poistettavaksi

5.4 Muutosten turvallinen implementointi ja jatkuva haavoittuvuuksien hallinta

Koska uusia haavoittuvuuksia havaitaan jatkuvasti, ei tietoturvallisuuden arviointi voi olla vain yksittäisten skannausten varassa. Tästä syystä olisi järkevää yhdistää turvallisuustestaus muutosten yhteyteen. Sovelluksen lähdekoodi on GitLabissa, johon on saatavilla SAST-automaatio, joka suorittaa NodeJSScan-skannauksen jokaisen muutoksen yhteydessä.

Haavoittuvuuksia tulisi testata dynaamisilla työkaluilla säännöllisesti. (Zeeshan ym. 2017) esittävät haavoittuvuuksien hallinnan mallin, jossa haavoittuvuuksien löydön jälkeen ne arvioidaan ja raportoidaan. Tämän jälkeen tehdään korjaustoimenpiteet ja varmistetaan niiden toimivuus. Useissa työkaluissa, kuten esimerkiksi Nessuksessa, on saatavilla ajastettuja skannauksia. Ajastetut skannauksen voidaan kohdistaa yöaikaan, jolloin oletettavasti sovelluksella on vähemmän käyttäjiä. Sopiva aikataulutus voisi olla esimerkiksi kerran kuukaudessa tai neljännesvuosittain.

6 Yhteenveto ja pohdinta

Tutkielman viimeisessä kappaleessa kootaan yhteen tutkimuksen tavoitteet ja tulokset. Lisäksi pohdintaosiossa arvioidaan tutkimuksen luotettavuutta, eettisyyttä ja aihetta jatkotutkimukselle.

6.1 Yhteenveto

Tietoturvallisuus koostuu tiedon luottamuksellisuudesta, saatavuudesta ja eheydestä. Verkkosovelluksessa näiden tiedon osa-alueiden toteutumista uhkaavat haavoittuvuudet, jotka ovat virheitä sovelluksen suunnittelussa, ohjelmoinnissa tai konfiguraatioissa. Merkittäviä haavoittuvuuksia ovat esimerkiksi injektiot, haavoittuvien komponenttien käyttö ja puutteet salauksessa tai käyttäjän todentamisessa. Näitä haavoittuvuuksia hyväksikäyttämällä sovelluksen arkaluonteisiin tietoihin voidaan päästä käsiksi tai estää sovellusta toimimasta halutulla tavalla. Haavoittuvuuksia etsitään skannaamalla sovellusta haavoittuvuusskannereilla ja analysoimalla ohjelmakoodia staattisesti. Lisäksi haavoittuvuuksia voidaan löytää selvittämällä sovelluksen komponenttien versioita ja etsimällä tunnettuja haavoittuvuuksia niistä.

Tutkimus toteutettiin konstruktiivista tutkimusotetta käyttäen. Teoreettinen viitekehys rakennettiin kirjallisuuskatsauksella aiheeseen liittyviin artikkeleihin ja kirjallisuuteen. Tutkimuskysymyksiksi esitettiin, mitä haavoittuvuuksia eräästä verkkosovelluksesta löytyy ja miten sen tietoturvallisuutta voidaan parantaa. Tutkimuskysymykseen vastattiin tekemällä haavoittuvuustestaus eri menetelmiä käyttäen. Löydettyihin haavoittuvuuksiin esitettiin parannusehdotuksia kirjallisuuden ja testaustyökalujen antamien korjausten perusteella. Sovellukseen tehtiin määritellyn testausprosessin mukaisesti tiedonhankintaa sekä dynaamista ja staattista haavoittuvuusskannausta. Lisäksi löydetyt haavoittuvuudet varmistettiin manuaalisella testauksella. Haavoittuvuuksia löytyi vanhentuneista ohjelmistoversioista, käytetyistä salausmenetelmistä sekä puutteellisista HTTP-protokollan otsaketiedoista.

Haavoittuvuuksien korjaaminen tulisi tehdä päivittämällä uusimmat ohjelmistoversiot, poistamalla käytöstä turvattomat salausmenetelmät ja ohjelmoimalla turvallisuusevästeet käyttöön. Yleisellä tasolla sovelluksen tietoturva on kohtuullisella tasolla ja käytettäessä valmiita

kirjastoja sovelluskehittäjien vastuuta turvallisesta koodista voidaan siirtää kirjaston kehittäjille. Yleisesti käytettävissä olevissa kirjastoissa olevat haavoittuvuudet yleensä myös korjataan nopeasti. Tässä sovelluksessa havaitut puutteet on suhteutettava tarvittavaan tietoturvan tasoon. Sovelluksessa itsessään ei lueta käyttäjän syötettä tai kirjauduta sisään, jolloin injektioihin ja istunnon kaappaamiseen liittyvät haavoittuvuudet eivät ole merkittävä uhka. Sovelluksen tietoturvan lisäksi tulisi kuitenkin huomioida myös erilaiset reflektoidut hyökkäykset, kuten XSS-hyökkäykset tai klikkausansat, joissa sovellusta käytetään välikappaleena käyttäjän tietojen varastamiseen. Koska toteutettu tietoturvatestaus on kuitenkin vain kuva tietoturvan tilasta testaushetkellä, turvallisuus tulisikin ottaa huomioon jatkuvana prosessina sovelluskehityksessä.

Tutkimus hyödyttää ensisijaisesti testatun sovelluksen kehittäjiä, jotka saavat konkreettista tietoa sovelluksen tietoturvallisuuden tilasta sekä valmiita ehdotuksia, joilla tietoturvallisuutta voidaan entisestään kehittää. Lisäksi tutkimus on kattava kuvaus tämänhetkisistä työkaluista ja prosessista, joilla verkkosovellusten haavoittuvuuksia voidaan testata, joten sillä on myös tieteellistä arvoa.

6.2 Luotettavuuden arviointi ja eettisyys

Luotettavuuden osalta haavoittuvuustestauksessa ongelmiksi tunnistettiin automaattisten skannaustyökalujen epätarkkuus ja testauksen aikasidonnaisuus. Yksittäisen skannaustyökalun epätarkkuus voi heikentää tulosten tarkkuutta ja oikeellisuutta, mutta tätä ongelmaa on vähennetty käyttämällä useita eri tavalla toimivia skannaustyökaluja sekä haavoittuvuuksien tarkastamista manuaalisin keinoin. Haavoittuvuusskannerien tulosten väliset erot johtuvat sekä työkalun toimintaperiaatteesta että käytetystä haavoittuvuustietokannasta. Esimerkiksi HTTP-otsakkeiden osalta kaikki skannerit eivät luokittele niitä haavoittuvuuksiksi ja myös skannereiden välillä on eroa siinä, mitkä otsakkeista liittyvät turvallisuuteen ja mitkä eivät. Tuloksiin sisällytettyjen otsakkeiden osalta on arvioitu selainvalmistajien tietoa otsakkeiden turvallisuudesta sekä skannerin tuottaman raportin perusteluja haavoittuvuudesta. Kaikki esitetyt haavoittuvuudet on todettu vähintään yhdellä haavoittuvuustyökalulla ja manuaalisella varmistamisella, mutta suurimmassa osassa useampi haavoittuvuusskanneri on havainnut saman haavoittuvuuden.

Haavoittuvuustulokset ovat vahvasti aikasidonnaisia, jolloin niiden toisintaminen voi olla haastavaa. Uusia haavoittuvuuksia voi ilmetä nopeastikin ja toisaalta vanhoja voidaan korjata, jolloin uudessa testauksessa ne eivät enää nouse esille. Tulosten esittämisen yhteydessä on kuvattu mahdollisimman tarkasti käytetyt työkalut ja niiden konfiguraatiot, jolloin tulosten toisintaminen on mahdollista. Tarkkoja testausajankohtia tai testattua versiota ei ole sisällytetty tutkielmaan, mutta ne ovat tarvittaessa saatavilla tulosten oikeellisuuden varmistamiseksi. Näiltä osin tutkimuksen luotettavuuden voidaan katsoa täyttävän tarkkuuden ja toistettavuuden vaatimukset.

Aiemmissa tutkimuksissa ei ole erityisesti korostettu haavoittuvuustutkimuksen eettisyyttä. Koska tässä tutkimuksessa on käsitelty haavoittuvuuksia ja niiden hyödyntämistä, on tärkeää nostaa esille myös eettisyyteen liittyvät kysymykset. Haavoittuvuustietojen päätyminen vääriin käsiin on haitallista erityisesti tilanteissa, joissa puhutaan uusista haavoittuvuuksista. Vaikka tässä tutkimuksessa ei etsitty tai löydetty tällaisia nollapäivähaavoittuvuuksia, testaukset muodostavat kuitenkin niin kattavan kuvan sovelluksen haavoittuvuuksista, että sitä on mahdollista käyttää myös väärin. Tästä syystä kaikki verkkosovellusta yksilöivät tiedot on poistettu tuloksista, eikä tutkimuksessa myöskään kerrota tarkasti sovelluksen toimintaperiaatetta.

Haavoittuvuustutkimus itsessään ei ole laitonta tai epäeettistä, varsinkaan kun sen tarkoituksena on parantaa testattavan sovelluksen tietoturvallisuutta. Kaikki kriittiset palvelua tai sen käyttäjiä uhkaavat haavoittuvuudet tulisi raportoida välittömästi palveluntarjoajalle. (Matwyshyn ym. 2010). On totta, että käsiteltyjä työkaluja ja menetelmiä voidaan yleisesti käyttää vahingontekotarkoituksessa. Työkalut ja ohjeet niiden käyttöön ovat kuitenkin julkisesti saatavilla, joten tältä osin tutkimus ei tuo mitään uutta haitallista menetelmää esille, jota vahinkoa haluava taho ei muuta kautta saisi selville.

6.3 Jatkotutkimus

Tutkimuksen teon aikana esille nousi mahdollisia jatkotutkimusaiheita. Tässä tutkimuksessa ei kiinnitetty huomiota haavoittuvuuksien etsimisen havainnointiin. Tutkimuksen näkökulman voisi kääntää palveluntarjoajan suuntaan siten, että arvioidaan menetelmiä, joilla

voidaan havaita palveluun kohdistuvia skannausyrityksiä. Tutkimuksessa palveluntarjoajalta pyydettiin lupa testausten toteuttamiseen, jolloin he tiesivät tarkat skannausajankohdat. Palvelun turvallisuuden näkökulmasta verkkoliikenteen monitorointi on tärkeää haavoittuvuuksien etsimisen lisäksi. Tästä näkökulmasta tutkimus voitaisiin kohdistaa kykyyn analysoida tunnettuja tai tunnistamattomia poikkeamia verkkoliikenteessä, jotka voivat olla uhka tietoturvallisuudelle. Skannauksen havaitseminen on mahdollista vain kun skannaus tehdään dynaamisilla työkaluilla, sillä staattista lähdekoodiin tehtävää analyysia ei voida havaita, koska se ei aiheuta verkkoliikennettä.

Toinen mielenkiintoinen aihe voisi olla tarkempi analyysi haavoittuvuusskannerien tarkkuudesta keskenään vertailtuna. Tässä tutkimuksessa haavoittuvuusskannerien osin ristiriitaiset tulokset esitettiin, mutta niitä ei erityisesti vertailtu keskenään muuta kuin luotettavuuden näkökulmasta. Tutkimusaiheena voisi olla haavoittuvuusskannauksen tekeminen useisiin eri sovelluksiin ja arvioida miten ja miksi tulokset poikkeavat toisistaan. Tällainen tutkimus voisi kehittää ymmärrystä haavoittuvuusskannerien toiminnasta ja rajoitteista.

Itse verkkosovelluksen tietoturvallisuuden osalta tämä tutkimus jättää myös kehittämisen varaa, koska varsinaista penetraatiotestausta ei tehty. Haavoittuvuuksia havaittiin ja niitä varmennettiin, mutta niitä ei yritetty käyttää tosiasiallisten hyökkäysten todentamiseen, koska tämä vaatii osaamis- ja aikaresursseja, joita ei enää tämän laajuiseen työhön saatu sisällytettyä. Toisaalta myös testattu verkkosovellus asetti rajoitteita, koska se ei mahdollista kirjautumista tai käyttäjän syötteen lukemista. Jatkotutkimuksessa voisi etsiä haavoittuvuuksia istuntoja ja lomakkeita sisältävästä palvelusta, jolloin todennäköisesti haavoittuvuuksiakin olisi enemmän.

Lähteet

Abdullah, Hanin Mohammed, ja Ahmed M. Zeki. 2014. “Frontend and Backend Web Technologies in Social Networking Sites: Facebook as an Example”. *2014 3rd International Conference on Advanced Computer Science Applications and Technologies:* 85–89.

Ashby, Dennis, ja Claus Jensen. 2018. *APIs for dummies.* 3. painos. Hoboken, NJ: Wiley.

Atymtayeva, Lyazzat, Serik Nurmyshev ja Gulfarida Tulemissova. 2017. *Proceedings of the Seventh International Symposium on Business Modeling and Software Design:* 136–145.

Awang, Nor Fatimah, ja Azizah Abd Manaf. 2013. “Detecting Vulnerabilities in Web Applications Using Automated Black Box and Manual Penetration Testing”. Teoksessa *Communications in Computer and Information Science, vol 381.* Toimittanut Ali Ismail Awad, Aboul Ella Hassanien ja Kensuke Baba, 230–239. Springer-Verlag.

Bacudio, Aileen G, Xiaohong Yuan, Bei Chu ja Monique M. Jones. 2011. “An Overview of Penetration Testing”. *International Journal of Network Security & Its Applications* 3:19–38.

Bairwa, Sheetal, Bhawna Mewara ja Jyoti Gajrani. 2014. “Vulnerability Scanners-A Proactive Approach To Assess Web Application Security”. *ArXiv* abs/1403.6955.

Berkeley Distribution. 2008. *BSD System Manager’s Manual: traceroute(8).*

Bhargavan, Karthikeyan, ja Gaëtan Leurent. 2016. “On the Practical (In-)Security of 64-bit Block Ciphers: Collision Attacks on HTTP over TLS and OpenVPN”. Teoksessa *CCS ’16.*

Cadariu, Mircea, Eric Bouwers, Joost Visser ja Arie van Deursen. 2015. “Tracking known security vulnerabilities in proprietary software systems”. *2015 IEEE 22nd International Conference on Software Analysis, Evolution, and Reengineering (SANER):* 516–519.

Christopher, Roger. 2002. “Port Scanning Techniques and the Defense Against Them”. *SANS Institute Information Security Reading Room.*

Clark, David, ja David Wilson. 1987. “A Comparison of Commercial and Military Computer Security Policies”. *IEEE Symposium on Security and Privacy:* 184–194.

Duong, Thai. 2011. “Beast”. Viitattu 30. huhtikuuta 2020. https://vnhacker.blogspot.com/2011/09/beast.html.

DuPaul, Neil. 2013. “Static Testing vs. Dynamic Testing”. Viitattu 19. huhtikuuta 2020. https://www.veracode.com/blog/2013/12/static-testing-vs-dynamic-testing.

El, Malaka, Emma McMahon, Sagar Samtani, Mark W. Patton ja Hsinchun Chen. 2017. “Benchmarking vulnerability scanners: An experiment on SCADA devices and scientific instruments”. *2017 IEEE International Conference on Intelligence and Security Informatics (ISI):* 83–88.

ENISA. 2019. “State of vulnerabilities 2018/2019: Analysis of Events in the life of Vulnerabilities”.

Eshete, Birhanu, Adolfo Villafiorita ja Komminist Weldemariam. 2011. “Early Detection of Security Misconfiguration Vulnerabilities in Web Applications”. *2011 Sixth International Conference on Availability, Reliability and Security:* 169–174.

Evans, Chris. 2020. “vsftpd: Probably the most secure and fastest FTP server for UNIX-like systems”. Viitattu 2. toukokuuta 2020. https://security.appspot.com/vsftpd.html.

Facebook Inc. 2020. “React: A JavaScript Library for building user interfaces”. Viitattu 18. huhtikuuta 2020. https://reactjs.org.

FIRST.org. 2019. “Common Vulnerability Scoring System version 3.1: Specification Document”. Viitattu 18. huhtikuuta 2020. https://www.first.org/cvss/v3.1/specification-document.

Greenbone. 2016. “Greenbone Security Manager Manual”. Viitattu 30. huhtikuuta 2020. https://docs.greenbone.net/GSM-Manual/gos-3.1/en/scanning.html.

Helme, Scott. 2019. “Alexa Top 1 Million Analysis - February 2019”. Viitattu 19. huhtikuuta 2020. https://scotthelme.co.uk/alexa-top-1-million-analysis-february-2019/.

Helme, Scott. 2020. “Security Headers: Scan your site now”. Viitattu 2. toukokuuta 2020. https://securityheaders.com.

Helmet. 2020. “Helmet: Express.js security with HTTP headers”. Viitattu 3. toukokuuta 2020. https://helmetjs.github.io.

HKSAR. 2008. “An Overview of Vulnerability Scanners”. Viitattu 11. huhtikuuta 2020. https://www.infosec.gov.hk/english/technical/files/vulnerability.pdf.

IETF. 2000. “RFC 3986: HTTP Over TLS”. Viitattu 19. huhtikuuta 2020. https://tools.ietf.org/html/rfc2818.

———. 2005. “RFC 2818: Uniform Resource Identifier (URI): Generic Syntax”. Viitattu 6. toukokuuta 2020. https://tools.ietf.org/html/rfc3986.

———. 2011. “RFC 6265: HTTP State Management Mechanism”. Viitattu 19. huhtikuuta 2020. https://tools.ietf.org/html/rfc6265.

———. 2012. “RFC 6797: HTTP Strict Transport Security (HSTS)”. Viitattu 30. huhtikuuta 2020. https://tools.ietf.org/html/rfc6797.

———. 2013. “RFC 7034: X-Frame-Options”. Viitattu 30. huhtikuuta 2020. https://tools.ietf.org/html/rfc7034.

———. 2014a. “RFC 7230: Hypertext Transfer Protocol (HTTP/1.1): Message Syntax and Routing”. Viitattu 18. huhtikuuta 2020. https://tools.ietf.org/html/rfc7230.

———. 2014b. “RFC 7231: Hypertext Transfer Protocol (HTTP/1.1): Semantics and Content”. Viitattu 19. huhtikuuta 2020. https://tools.ietf.org/html/rfc7231.

———. 2016. “RFC 7762: Initial Assignment for the Content Security Policy Directives Registry”. Viitattu 1. toukokuuta 2020. https://tools.ietf.org/html/rfc7762.

———. 2018. “RFC 8446: The Transport Layer Security (TLS) Protocol Version 1.3”. Viitattu 19. huhtikuuta 2020. https://tools.ietf.org/html/rfc8446.

InMotion Hosting. 2020. “How to Read a Traceroute”. Viitattu 29. huhtikuuta. https://www.inmotionhosting.com/support/website/ssh/read-traceroute/.

Jiménez, Rina Elizabeth López de. 2016. “Pentesting on web applications using ethical - hacking”. *2016 IEEE 36th Central American and Panama Convention (CONCAPAN XXXVI):* 1–6.

Johnson, Michael, ja Perdita Stevens. 2018. “Confidentiality in the Process of (Model-Driven) Software Development”. *2nd International Conference on Art, Science and Engineering of Programming.*

Khera, Yugansh, Deepansh Kumar, Sujay ja Nidhi Garg. 2019. “Analysis and Impact of Vulnerability Assessment and Penetration Testing”. *2019 International Conference on Machine Learning, Big Data, Cloud and Parallel Computing (COMITCon):* 525–530.

Khunphet, Phonlawat. 2019. “Insecure Deserialization”. Viitattu 26. huhtikuuta 2020. https://medium.com/blog-blog/insecure-deserialization-e5398e83defe.

Al-Khurafi, Ossama B., ja Mohammad A. AlAhmad. 2015. “Survey of Web Application Vulnerability Attacks”. *2015 4th International Conference on Advanced Computer Science Applications and Technologies (ACSAT):* 154–158.

Klingsgeim, André. 2020. “X-Download-Options”. Viitattu 6. toukokuuta 2020. https://www.nwebsec.com/HttpHeaders/SecurityHeaders/XDownloadOptions.

Korkein oikeus. 2003. “KKO 2003:36”. Viitattu 19. huhtikuuta 2020. https://www.finlex.fi/fi/oikeus/kko/kko/2003/20030036.

Kumar, Vivekanandan. 2014. “Ethical Hacking and Penetration Testing Strategies”. *International Journal of Emerging Technology in Computer Science & Electronics* 11:21–23.

Li, Peng, ja Baojiang Cui. 2010. “A comparative study on software vulnerability static analysis techniques and tools”. *2010 IEEE International Conference on Information Theory and Information Security:* 521–524.

Lukka, Kari. 2003. “The Constructive Research Approach”. *Turun kauppakorkeakoulun julkaisuja* B (1:2003): 83–101.

Lukka, Kari. 2014. “Kari Lukka: Konstruktiivinen tutkimusote”. Viitattu 13. huhtikuuta 2020. https://metodix.fi/2014/05/19/lukka-konstruktiivinen-tutkimusote/.

Makino, Yuma, ja Vitaly Klyuev. 2015. “Evaluation of web vulnerability scanners”. *2015 IEEE 8th International Conference on Intelligent Data Acquisition and Advanced Computing Systems: Technology and Applications (IDAACS)* 1:399–402.

Matwyshyn, Andrea M., Ang Cui, Angelos D. Keromytis ja Salvatore J. Stolfo. 2010. “Ethics in security vulnerability research”. *IEEE Security & Privacy* 8.

McBride, Tim, Anne Townsend, Michael Ekstrom, Lauren Lusty ja Julian Sexton. 2018. “Data Integrity: Recovering from Ransomware and Other Destructive Events”. *2018 IEEE Cybersecurity Development (SecDev):* 140–140.

McMahon, Emma, Mark Patton, Sagar Samtani ja Hsinchun Chen. 2018. “Benchmarking Vulnerability Assessment Tools for Enhanced Cyber-Physical System (CPS) Resiliency”. *2018 IEEE International Conference on Intelligence and Security Informatics (ISI):* 100–105.

Microsoft. 2008. “Mitigating Cross-site Scripting With HTTP-only Cookies”. Viitattu 6. toukokuuta 2020. https://docs.microsoft.com/en-us/previous-versions//ms533046(v=vs.85).

———. 2018. “Naming Files, Paths, and Namespaces”. Viitattu 30. huhtikuuta 2020. https://docs.microsoft.com/en-us/windows/win32/fileio/naming-a-file.

Mounika, Vanamala, Xiaohong Yuan ja Kanishka Bandaru. 2019. “Analyzing CVE Database Using Unsupervised Topic Modelling”. *2019 International Conference on Computational Science and Computational Intelligence (CSCI):* 72–77.

Mozilla. 2016. “Mitigating MIME Confusion Attacks in Firefox”. Viitattu 1. toukokuuta 2020. https://blog.mozilla.org/security/2016/08/26/mitigating-mime-confusion-attacks-in-firefox/.

Mozilla. 2019a. “Content-Security-Policy”. Viitattu 1. toukokuuta 2020. https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers/Content-Security-Policy.

———. 2019b. “X-XSS-Protection”. Viitattu 1. toukokuuta 2020. https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers/X-XSS-Protection.

———. 2020a. “HTTP headers”. Viitattu 3. toukokuuta 2020. https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers.

———. 2020b. “HTTP Public Key Pinning (HPKP)”. Viitattu 6. toukokuuta 2020. https://developer.mozilla.org/en-US/docs/Web/HTTP/Public_Key_Pinning.

Nagpure, Sangeeta, ja Sonal Kurkure. 2017. “Vulnerability Assessment and Penetration Testing of Web Application”. *2017 International Conference on Computing, Communication, Control and Automation (ICCUBEA):* 1–6.

Nahari, Hadi, ja Ronald L. Krutz. 2011. *Web Commerce Security: Design and Development.* 1. painos. Indianapolis, IN: Wiley.

nginx. 2020. “nginx change log”. Viitattu 2. toukokuuta 2020. http://nginx.org/en/CHANGES.

O’Gorman, Lawrence. 2003. “Comparing Passwords, Tokens, and Biometrics for User Authentication”. *Proceedings of the IEEE* 91 (12): 2019–2040.

OpenSSH. 2020. “OpenSSH Release Notes”. Viitattu 2. toukokuuta 2020. https://www.openssh.com/releasenotes.html.

OWASP Foundation. 2016. “OWASP Category: Vulnerability”. Viitattu 11. huhtikuuta 2020. https://wiki.owasp.org/index.php/Category:Vulnerability.

———. 2017. “OWASP Top Ten: Top 10 Web Application Security Risks”. Viitattu 13. huhtikuuta 2020. https://owasp.org/www-project-top-ten/.

———. 2020. “OWASP Secure Headers Project”. Viitattu 3. toukokuuta 2020. https://owasp.org/www-project-secure-headers/.

Pfleeger, C., ja S. Pfleeger. 2003. *Security in Computing*. 3. painos. Prentice Hall PTR.

Phadke, Apoorva. 2016. “SAST vs. DAST: What’s the best method for application security testing?” Viitattu 8. toukokuuta 2020. https://www.synopsys.com/blogs/software-security/sast-vs-dast-difference/.

Qualys. 2020. “SSL Server Test”. Viitattu 3. toukokuuta 2020. https://www.ssllabs.com/ssltest/analyze.html.

Radware. 2019. “The State of Web Application Security”. Viitattu 26. huhtikuuta 2020. https://www.radware.com/social/was-report-2019/.

Raggad, Belgacem. 2008. *Information Security Management: Concepts and Practice*. Boca Raton, FL: CRC Press.

Rasheed, Shawn, Jens Dietrich ja Amjed Tahir. 2019. “Laughter in the Wild: A Study Into DoS Vulnerabilities in YAML Libraries”. *2019 18th IEEE International Conference On Trust, Security And Privacy In Computing And Communications/13th IEEE International Conference On Big Data Science And Engineering (TrustCom/BigDataSE):* 342–349.

Saad, Elie, Matteo Meucci ja Rick Mitchell. 2020. “OWASP Web Security Testing Guide”. Viitattu 1. toukokuuta 2020. https://owasp.org/www-project-web-security-testing-guide/stable/.

Salz, Rich. 2016. “The SWEET32 Issue, CVE-2016-2183”. Viitattu 3. toukokuuta 2020. https://www.openssl.org/blog/blog/2016/08/24/sweet32/.

Sandhu, Ravi, Edward Coyne, Hal Feinstein ja Charles Youman. 1996. “Role-Based Access Control Models”. *IEEE Computer* 29 (2): 38–47.

Shah, Sugand, ja B. M. Mehtre. 2015. “An overview of vulnerability assessment and penetration testing techniques”. *J Comput Virol Hack Tech,* numero 11: 27–49.

Shebli, Hessa Mohammed Zaher Al, ja Babak D. Beheshti. 2018. “A study on penetration testing process and tools”. *2018 IEEE Long Island Systems, Applications and Technology Conference (LISAT):* 1–7.

Shinde, Prashant S., ja Shrikant B. Ardhapurkar. 2016. “Cyber security analysis using vulnerability assessment and penetration testing”. *2016 World Conference on Futuristic Trends in Research and Innovation for Social Welfare (Startup Conclave):* 1–5.

Shklar, Leon, ja Rich Rosen. 2009. *Web Application Architecture: Principles, Protocols and Practices.* 2. painos. West Sussex, England: Wiley.

Skaggs, Brandon, Brian Blackburn, Gavin Wylie Manes ja Sujeet Shenoi. 2002. “Network vulnerability analysis”. *The 2002 45th Midwest Symposium on Circuits and Systems, 2002. MWSCAS-2002.* 3:III–493.

Sonarqube. 2019. “SonarScanner”. Viitattu 5. toukokuuta 2020. `https://docs.sonarqube.org/latest/analysis/scan/sonarscanner/`.

Subil, Abraham, Thomas Mathews ja Thomas Johnson. 2005. “Enhancing Web Services Availability”. *IEEE International Conference on e-Business Engineering.*

Sundharam, R., M. Lakshmi ja D. Abarajithan. 2010. “Enhancing the Availability of Web Services for Mission Critical Applications”. *Trends in Information Sciences & Computing:* 149–151.

The MITRE Corporation. 2019. “CWE Top 25 Most Dangerous Software Errors”. Viitattu 18. huhtikuuta 2020. `https://cwe.mitre.org/top25/archive/2019/2019_cwe_top25.html`.

———. 2020a. “About CWE”. Viitattu 18. huhtikuuta 2020. `https://cwe.mitre.org/about/index.html`.

———. 2020b. “CWE CATEGORY: OWASP Top Ten 2017 Category A2 - Broken Authentication”. Viitattu 21. huhtikuuta 2020. `https://cwe.mitre.org/data/definitions/1028.html`.

———. 2020c. “CVE Details Vulnerabilities By Date”. Viitattu 11. huhtikuuta 2020. `https://www.cvedetails.com/browse-by-date.php`.

———. 2020d. “CVE Details Vulnerabilities By Type”. Viitattu 10. huhtikuuta 2020. `https://www.cvedetails.com/vulnerabilities-by-types.php`.

The MITRE Corporation. 2020e. “CVE Numbering Authority (CNA) Rules”. Viitattu 18. huhtikuuta 2020. https://cve.mitre.org/cve/cna/rules.html.

———. 2020f. “CWE-89: Improper Neutralization of Special Elements used in an SQL Command (’SQL Injection’)”. Viitattu 18. huhtikuuta 2020. https://cwe.mitre.org/data/definitions/89.html.

Thompson, Chris. 2019. “Chrome UI for Deprecating Legacy TLS Versions”. Viitattu 4. toukokuuta 2020. https://blog.chromium.org/2019/10/chrome-ui-for-deprecating-legacy-tls.html.

Traficom. 2020. “Tietoturvan vuosi 2019: Kyberturvallisuuskeskuksen vuosikatsaus”. *Traficomin julkaisuja* 5.

Turvallisuuskomitea. 2018. *Kyberturvallisuuden sanasto.*

W3C. 2014. “Cross-Origin Resource Sharing”. Viitattu 30. huhtikuuta 2020. https://www.w3.org/TR/cors/.

W3Counter. 2020. “Web Browser Usage Trends”. Viitattu 6. toukokuuta 2020. https://www.w3counter.com/trends.

Wagner, Eva. 2017. “Why Should You Configure 404 Error Pages”. Viitattu 10. toukokuuta 2020. https://en.ryte.com/magazine/why-should-you-configure-404-error-pages.

Wainakh, Aidmar, A A Wabbi Wabbi ja Bassel Alkhatib. 2014. “Design and Develop Misconfiguration Vulnerabilities Scanner for Web Applications”. *International Review on Computers and Software* 9:1682–1691.

Valtiovarainministeriö. 2013. *Sovelluskehityksen tietoturvaohje: VAHTI 1/2013.* 31. tammikuuta.

Wang ym. 2009. “Environmental metrics for software security based on a vulnerability ontology”. *Third IEEE International Conference on Secure Software Integration and Reliability Improvement:* 159–169.

WHATWG. 2020. “HTML: Living Standard”. Viitattu 6. toukokuuta 2020. https://html.spec.whatwg.org/multipage/introduction.html#introduction.

Yuan, Jian, ja Kevin Mills. 2005. “Monitoring the macroscopic effect of DDoS flooding attacks”. *IEEE Transactions on Dependable and Secure Computing* 2:324–335.

Zeeshan, Muhammad, Tazeen Majeed, Nayab Nasir, Saadia Anayat ja Shams Un Nisa. 2017. “Vulnerability Assessment and Penetration Testing: A proactive approach towards Network and Information Security”. *International Journal of Digital Information and Wireless Communications* 7:124–142.