Jaakko Juvonen

VERKKOSOVELLUSTEN KRIITTISIMMÄT HAAVOITTUVUUDET JA MITEN NIITÄ EHKÄISTÄÄN

JYVÄSKYLÄN YLIOPISTO
INFORMAATIOTEKNOLOGIAN TIEDEKUNTA
2024

TIIVISTELMÄ

Juvonen, Jaakko
Verkkosovellusten kriittisimmät haavoittuvuudet ja miten niitä ehkäistään
Jyväskylä: Jyväskylän yliopisto, 2024, 34 s.
Tietojärjestelmätiede, kandidaatin tutkielma
Ohjaaja(t): Lampi, Anna

Viimevuosien aikana verkkosovellukset ovat kasvattaneet suosiotaan verkkoselaimella saavutettavan universaalin saatavuuden, sekä helppokäyttöisyyden vuoksi. Tämä muutos ei kuitenkaan ole saapunut ilman seuraamuksia. Verkkosovellukset omaavat suuren hyökkäyspinta-alan pahantahtoisia toimijoita kohtaan, pääasiassa monimutkaisuutensa sekä sovelluksissa käytettävien teknologioiden suuren määrän takia. Tämä tutkimus keskittyi kriittisimpiin verkkosovellusten haavoittuvuuksiin, sekä siihen, miten niitä voidaan ehkäistä. Tutkimus toteutettiin systemaattisena kirjallisuuskatsauksena. Haavoittuvuudet johdettiin tuoreimmasta OWASP Top Ten- listauksesta kriittisimmistä verkkosovellusten haavoittuvuuksista, sekä haavoittuvuudet kategorisoitiin neljään eri kategoriaan niiden ominaisuuksien mukaan. Tutkimuksessa huomattiin, että vaikka monia haavoittuvuuksista voidaan lievittää tai vähentää implementoimalla puolustautuvaa ohjelmointia, suuren osan ajasta skannaukset ja testaukset suoritetaan legacy-sovelluksille. Tämä korostaa dynaamisen ja staattisen analyysin merkitystä. Toinen suuri tekijä turvallisessa verkkosovelluksen kehityksessä on inhimillisen virheen minimointi, joka on paljolti edustettuna loogiseen rakenteeseen liittyvissä haavoittuvuuksissa sekä alustan konfiguroinnissa.

Asiasanat: verkkosovellusten haavoittuvuudet, tietoturva, OWASP Top Ten

ABSTRACT

Juvonen, Jaakko
The most critical vulnerabilities in web applications and how to prevent them
Jyväskylä: University of Jyväskylä, 2024, 34 pp.
Information Systems, Bachelor's Thesis
Supervisor(s): Lampi, Anna

In recent years, web applications have grown in popularity due to their universal accessibility via browsers and their ease of use. However, this change has not come without its consequences. Web applications possess a large attack surface for malicious actors, mainly because of the complexity and number of different technologies implemented in these applications. This study focused on the most critical web application vulnerabilities, and how to detect and avoid them. The study was made as a systematic literary review. The vulnerabilities were derived from the latest OWASP Top Ten listing of the most critical web application vulnerabilities, and they were further categorized into four different types by their properties. It was found that while most of the vulnerabilities can be mitigated by implementing defensive coding practices, most of the time vulnerability scanning and testing is made on legacy applications. This highlights the importance of dynamic and static analysis. Another major factor in secure web application development is minimizing human error, which is massively represented on vulnerabilities regarding the web applications logical correctness or the configuration of the platform.

Keywords: web application vulnerability, information security, OWASP Top ten

KUVIOT

TAULUKOT

SISÄLLYS

1 JOHDANTO

Internetin alkuaikoina verkkosivut olivat lähinnä staattisia HTML-sivuja, joissa ei ollut tarjolla juurikaan toiminnallisuuksia. Nykypäivänä staattiset sivut ovat jääneet historiaan, ja merkittävä osa verkkosivuista on toteutettu dynaamisten verkkosovellusten muodossa. Dynaamisten verkkosovellusten monimutkaisuuden vuoksi ohjelmistoon voi jäädä haavoittuvuuksia, jotka mahdollisesti altistavat verkkosovelluksen kyberhyökkäykselle. Verkkosovelluksien haavoittuvuuksia on tärkeä tutkia, sillä niiden osuus taloudellisissa, hallinnollisissa sekä terveydenhuollon organisaatioissa on kasvussa (Chaudhari & Vaidya, 2014). Verkkosovellusten haavoittuvuuksien onnistunut väärinkäyttö voi lievimmillään johtaa esimerkiksi verkkosivujen turmelemiseen (engl. *deface*), mutta pahimmassa tapauksessa arkaluontoinen tieto, kuten liiketoimintasalaisuudet tai henkilötiedot voivat päätyä vääriin käsiin. Suomessa on nähty esimerkiksi Vastaamon tapauksessa suuren mittakaavan seuraukset huonosta kyberturvallisuuden implementoinnista. Kyseisessä tapauksessa kymmenet tuhannet potilastiedot päätyivät Tor-verkkoon, kun psykoterapiaa tarjoavan yritykseen Vastaamoon kohdistuneisiin kiristyslunnaisiin ei vastattu. Seurauksina olivat muun muassa mainehaitta yritykselle sekä jopa kymmeniä tuhansia rikosilmoituksia (Kortesoja, 2022).

Haavoittuvuuksia on hyvin vaikea löytää, mikäli ei tiedä miten ja mistä etsiä. Ohjelmistokehityksessä turvallinen ohjelmointi ei takaa sitä, etteikö haavoittuvuuksia pääsisi verkkosovellukseen, sillä haavoittuvuuksia voivat aiheuttaa inhimillisten virheiden lisäksi suuri määrä erilaisia tekijöitä, kuten vanhentuneet komponentit tai rikkinäinen pääsynhallinta. Tutkielman tavoitteena on esittää verkkosovellusten kriittisimpiä haavoittuvuuksia, havainnollistaa miten näitä haavoittuvuuksia voidaan käyttää verkkosovellusta vastaan sekä esittää keinoja haavoittuvuuksien havaitsemiseen sekä ehkäisyyn.

Tutkielma on tehty systemaattisen kirjallisuuskatsauksen muodossa (Salminen, 2023). Koska tietojärjestelmätieteen julkaisuista ei juurikaan löytynyt artikkeleita tutkielmaan liittyen, artikkelit ovat suurimmalta osin tietotekniikan alan konferenssijulkaisuja. Tutkielmassa yleisimmin käytetyt tietokannat ovat Google Scholar, IEEE Xplore, Springer, ScienceDirect sekä Scopus.

Hakulausekkeita tutkielmaa tehdessä ovat olleet *web application vulnerabilities, input validation vulnerabilities, secure web development, session management vulnerabilities, logic vulnerabilities* sekä *web platform vulnerabilities.* Osasta haavoittuvuuksia on etsitty lähteitä niiden englanninkielisellä nimikkeellä. Nimikkeet ovat esiteltyinä taulukossa 1. Vertaisarvioituja artikkeleita aiheesta löydettiin yhteensä 79 kappaletta, joista tutkimukseen otettiin relevanteimmat ja luotettavimmat lähteet, joita oli 47 kappaletta. Lisäksi tutkimuksessa hyödynnettiin kirjoja sekä sovelluskehitykselle omistautuneita verkkosivuja. Lähteiden arviointiperusteina käytettiin viittausten määrää artikkeliin, julkaisufoorumin luokitusta, mikäli sellainen julkaisulla on, julkaisevan tahon luotettavuutta sekä artikkelin ajantasaisuutta ja sen sisältöä. Vaikka muutamalla tutkimuksessa käytetyistä konferenssijulkaisuista ei ole julkaisufoorumin luokitusta, nämä artikkelit ovat arvioitu käyttökelpoisiksi tutkimukseen.

Tutkielman tarkoituksena on vastata seuraaviin tutkimuskysymyksiin: *Miten verkkosovellusten haavoittuvuuksia hyväksikäytetään? Miten verkkosovellusten haavoittuvuuksia voidaan havaita ja ehkäistä?* Kirjallisuuskatsauksen perusteella haavoittuvuudet on jaoteltu neljään eri pääkategoriaan: syötteiden validointiin, istunnon hallintaan, loogiseen rakenteeseen sekä alustan ja sen rakenteen haavoittuvuuksiin. Kirjallisuuskatsaukseen on kerätty tietoa yleisimmistä haavoittuvuuksista sekä niiden vaikutuksista verkkosovellukseen, sekä esitetty käytänteitä kyseisten haavoittuvuuksien havainnointiin sekä ehkäisyyn. Tutkimuksen avulla varsinkin verkkosovelluskehityksen vasta-alkajat kykenevät vähentämään projekteistaan löytyvien haavoittuvuuksien riskejä, sekä saavat tietoa mahdollisista löytyvistä haavoittuvuuksista.

Tutkielmassa käydään aluksi läpi verkkosovelluksen rakennetta ja toiminnallisuuksia, jotta käsiteltäviä haavoittuvuuksia olisi helpompi ymmärtää. Luvussa 2 käydään läpi Lin ja Xuen (2014) kategorioittain verkkosovelluksien kriittisimpiä haavoittuvuuksia sekä sitä, miten niitä voidaan hyväksikäyttää. Kolmannessa luvussa esitetään keinoja samaisten haavoittuvuuksien havaitsemiseen sekä esitetään konkreettisia keinoja haavoittuvuuksien ehkäisyyn. Yhteenvetokappaleessa käsitellään tutkimusaihe, tutkimusmenetelmät, tulokset ja johtopäätökset sekä jatkotutkimusehdotukset.

1.1 Verkkosovellusten rakenne

Conallenin (1999) määritelmä verkkosovellukselle on hyvinkin löyhä; verkkosovellus on verkkojärjestelmä, jossa käyttäjän syöte vaikuttaa järjestelmään. Shklarin ja Rosenin (2003) määritelmän mukaan verkkosovelluksella tarkoitetaan asiakas-palvelin -sovelluksia, jotka tarjoavat interaktiivisia palveluja internetin tai intranetin välityksellä. Molemmille määritelmille yhteistä on se, että siinä missä verkkosivut tarjoavat sisältöä staattisista tiedostoista, verkkosovellus tarjoaa dynaamisesti muuttuvaa sisältöä.

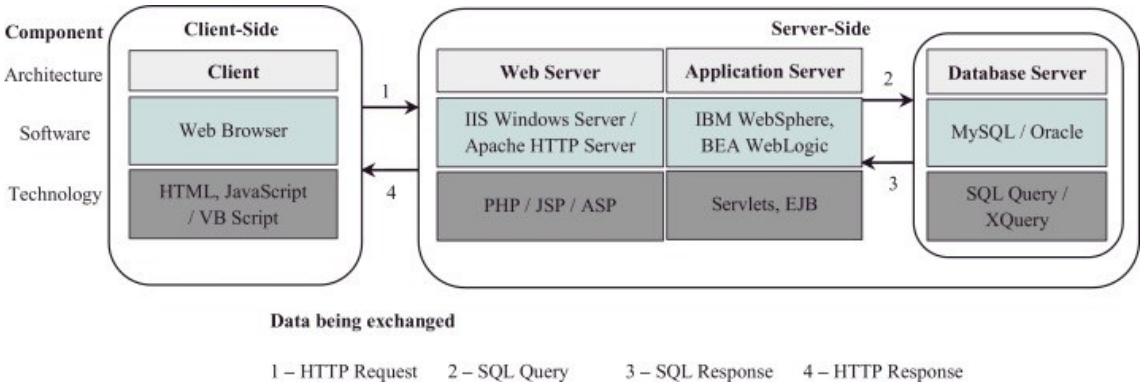
Tässä alaluvussa käsitellään verkkosovellusten rakennetta, jotta tutkimuksessa esitettyjä haavoittuvuuksia sekä niiden ehkäisykeinoja on helpompi ymmärtää. Hyökkääjät käyttävät verkkosovellusten erilaisia haavoittuvuuksia hyväkseen saadakseen käsiinsä esimerkiksi arkaluontoista tietoa, kuten käyttäjätietoa tai liiketoimintasalaisuuksia.

Verkkosovellukset ovat hyvinkin monimutkaisia sovelluksia, jotka sisältävät paljon erilaisia ohjelmistokomponentteja. Verkkosovellusten rakenne koostuu yleisesti ottaen asiakas- ja palvelinkomponenteista. (Abdullah & Zeki, 2014.) Asiakaskomponentit ovat komponentteja, joita käytetään käyttäjän laitteella, ja palvelinpuolen komponentit sisältävät backend-komponentit, joilla esimerkiksi tallennetaan ja prosessoidaan tietoa tietokantoja käyttäen (Chaudhari & Vaidya, 2014).

Verkkosovellus on hajautettu sovellus, joka mahdollistaa dynaamisen informaation ja palvelun tarjonnan, ja se koostuu asiakas- ja palvelin- komponenteista (Chaudhari & Vaidya, 2014). Chaudharin ja Vaidyan mukaan asiakaspuolen komponentit sisältävät staattisen verkkosivun sekä siihen sisälletyt skriptauskielet, esimerkiksi JavaScriptin, jotka suoritetaan verkkoselaimella. Asiakasohjelma on se osa verkkosovellusta, jonka suoritus tapahtuu käyttäjän verkkoselaimella. Asiakasohjelma tuottaa verkkoselaimen avulla verkkosovelluksen käyttöliittymän sekä toiminnallisuudet. Yleisesti käytössä olevassa kolmitasomallissa palvelinpuolella on verkkopalvelin, sovelluspalvelin sekä tietokantapalvelin (Deepa & Thilagam, 2016).

Asiakaspuolen komponentteihin kuuluvat muun muassa HTML, CSS sekä JavaScript. HTML eli HyperText Markup Language on kieli, jolla määritellään verkkosivun rakenne ja tarkoitus, CSS eli Cascading Style Sheets määrittelee verkkosivun ulkoasun sekä JavaScriptillä voidaan toteuttaa funktionaalisuus ja sivun käyttäytyminen (*HTML*, 2023). Palvelinpuolen komponentteihin kuuluvat muun muassa tietokannat, palvelimet sekä backend ohjelmointikielet.

Verkkosovelluksen konfiguraatioissa ja teknologioissa on suuria eroja verkkosovellusten välillä. Esimerkiksi Facebook käyttää niin kutsuttua LAMP- kombinaatiota (Linux, Apache, MySQL, PHP). Kombinaatiossa Linux-käyttöjärjestelmällä ylläpidetään Apachen HTTP-palvelimia, tietokantapalvelut on toteutettu MySQL:ää käyttäen sekä PHP:tä käytetään pääasiallisena ohjelmointikielenä (Abdullah & Zeki, 2014)



Kuvio 1. (Deepa & Thilagam, 2016, s.2) Esimerkki verkkosovelluksen arkkitehtuurista.

HTML eli HyperText Markup Language on kieli, jolla verkkosivujen perustavanlaatuinen rakenne määritellään. Hypertext viittaa linkkeihin, joiden avulla verkkosivut yhdistyvät toinen toisiinsa joko sivustojen välillä tai sivuston sisäisesti (*HTML*, 2023). HTML syntaksissa kutsutaan elementtejä, joiden avulla verkkosivun rakenne muodostetaan. Elementit ympäröidään "<" sekä ">" merkeillä, ja merkkien sisään ilmoitetaan haluttu elementti. Esimerkiksi elementti "<*img*>"määrittää, että sivustolle upotetaan kuva (<*img*>, 2023). Tutkimuksen kannalta tärkeimmät elementit ovat "<*script*>" ja "<*input*>". Script- elementti tulkitaan verkkoselaimessa JavaScriptinä, ja input-elementti piirtää selaimeen syötekentän.

CSS eli Cascading Style Sheets on tyyliohjeistokieli, jota käytetään kuvailemaan HTML- tai XML-dokumenttien ulkoasu. CSS kuvailee, kuinka eri elementit tulisi kuvantaa (*CSS*, 2023).

JavaScript on komentosarjakieli, jonka avulla verkkosivuille voidaan luoda dynaamista sisältöä. Dynaamisuudella tarkoitetaan sitä, että kun HTML-dokumentin näkymää halutaan muokata, selain ei nouda uutta tai muutettua dokumenttia, vaan JavaScriptin avulla jo näkyvissä olevaa dokumenttia muokataan (*JavaScript*, 2023). JavaScript voidaan merkitä HTML-dokumenttiin "<*script*>"-elementillä, tai se voidaan linkittää erillisinä JavaScript-tiedostona.

Verkkopalvelimet, selaimet sekä välityspalvelimet kommunikoivat vaihtamalla HTTP viestejä keskenään (Shklar & Rosen, 2003). Palvelinpuolen komponentteina toimii usein useita erilaisia sovelluksia, joilla on omat tehtävänsä, kuten autentikaatio tai tiedon tallennus ja -hakuoperaatiot.

Palvelinpuolen järjestelmillä on monia erilaisia tehtäviä, jotka vaihtelevat sovelluksen tarkoituksen mukaan. Facebookin tapauksessa tehtävinä ovat esimerkiksi lokitietojen käsittely, PHP-skriptien käännös toiselle kielelle, ladattujen kuvien käsittely ja tallennus, SQL kyselyiden suoritus sekä kuormien tasaaminen (Abdullah & Zeki, 2014).

Palvelinpuolen järjestelmien sekä asiakaspuolen järjestelmien kommunikointi tapahtuu HTTP-protokollaa (HyperText Transfer Protocol) käyttäen. HTTP on tilaton ja kevyt protokolla, jonka takia palvelinpuolen järjestelmien ei tarvitse pitää auki yhteyksiä asiakaspuolen järjestelmiin. Tilattomuus tarkoittaa sitä, että pyyntöjä ja vastauksia käsitellään itsenäisinä tapahtumina, eivätkä ne ole kytköksissä toisiinsa. Tämän vuoksi protokollaan on lisätty evästeet, joiden avulla käyttäjiä voidaan yksilöidä ja tunnistaa pyynnöistä. (Stuttard & Pinto, 2011.)

HTTP on joukko sovellustason pyyntö- ja vastausprotokollia, jotka ovat käytössä nykyaikaisissa verkkosovelluksissa. HTTP piilottaa palvelun implementoinnin esittämällä yhdenmukaisia rajapintoja asiakkaille, jotka ovat riippumattomia tarjotuista resursseista. Tämä tarkoittaa sitä, että palvelinten ja asiakkaiden ei tarvitse olla tietoisia toisessa päässä tapahtuvista operaatioista ja asiakkaiden tarkoituksista; onnistuneelle tiedonsiirrolle riittää, että viestit ovat syntaksisesti oikeanlaisia. HTTP toimii myös välitysprotokollana, jolloin esimerkiksi välityspalvelimet voivat kääntää HTTP:ksi tietoa, joka ei alun perin ole HTTP:ksi kirjoitettua. (Fielding, Nottingham, Reschke, 2022.)

HTTP-protokollan viestinvaihto on toteutettu pyynnöillä ja vastauksilla. Tärkeimmät metodit tämän tutkimuksen osalta ovat GET, POST ja PUT. GET-metodia käytetään tiedon hakemiseen, POST-metodia tiedon lähettämiseen sekä PUT-metodia tiedon päivittämiseen (Fielding ym., 2022).

GET-metodilla pyydetään välittämään valittu esitysmuoto pyydetystä resurssista. Resurssilla voi olla tarjolla tai se voi olla kykenevä generoimaan useita erilaisia esitysmuotoja resurssin sen hetkisestä tilasta (Fielding ym., 2022). GET-pyyntö lähetetään esimerkiksi silloin, kun palvelimelta halutaan noutaa HTML-dokumentti. GET-metodilla ei pystytä kirjoittamaan tietokantaan uutta tietoa, tai muokata jo olemassa olevaa tietoa. GET-pyyntö saadaan aikaiseksi esimerkiksi klikkaamalla sivustolle asetettua linkkiä, joka ohjaa toiselle verkkosivulle tai tiedoston lataamiseen (Fielding ym., 2022).

POST-metodilla välitetään tietoa asiakkaalta palvelimelle. Tiedonsiirto tapahtuu HTTP-pyynnön viestikentän avulla. POST-metodin käyttötarkoituksia voivat olla esimerkiksi kirjautumistietojen lähetys. Metodia tulisi käyttää silloin, kun tarkoituksena on lähettää uutta tietoa palvelimelle, luoda resurssi, jota ei ole aikaisemmin ollut olemassa, lisätä tietoa jo valmiiksi olemassa olevaan esitysmuotoon tai lähettää tietoa käsiteltäväksi (Fielding ym., 2022.)

PUT-metodilla muutetaan kohteena olevan resurssin esitysmuotoa pyynnössä esitettyyn muotoon. Onnistuneen PUT-pyynnön jälkeen kohteena olevaan resurssiin kohdistuva GET-pyyntö palauttaa samanlaisen esitysmuodon kohteesta, mitä PUT-metodilla on muokattu (Fielding ym., 2022). Yksinkertaistettuna PUT-metodia tulee käyttää silloin, kun jo olemassa olevaa tietoa halutaan päivittää toiseen muotoon.

HTTP-pyyntöjen tilattomuuden takia tarvitaan metodeja istuntojen ylläpitoon ja käyttäjien tunnistamiseen. Tämä suoritetaan evästeiden (engl. *cookie)* avulla. Eväste asetetaan *Set-Cookie*- otsakkeella. Otsakkeet ovat osa http-pyyntöä, jossa jokainen otsake voi muuttaa tai määrittää viestin semantiikkaa, kuvailla

lähettäjää, määrittää sisältöä tai välittää lisää kontekstia viestiin (Fielding ym., 2022). *Set-Cookie*- otsakkeella palvelin voi lähettää merkkijonon HTTP-vastauksella. Tätä merkkijonoa käytetään *Cookie*- otsakkeessa asiakkaan puolella jatkossa, jotta asiakas voidaan tunnistaa jatkossa samaksi käyttäjäksi, jolle otsakkeessa määritetty istuntotunniste kuuluu. Evästeotsakkeessa voidaan myös säilyttää muuta tietoa, kuten ensisijainen kielivalinta (Barth, 2011).

Nykyisin verkkosovellukset käyttävät suurimmalta osin tavallisen HTTP-protokollan sijaan turvallisempaa HTTPS-protokollaa. HTTPS-protokollassa HTTP- pyynnöt ja vastaukset suojataan TLS (Transport Layer Security) -protokollalla, tai vanhemmissa versioissa SSL (Secure Socket Layer) -protokollaa käyttäen. HTTPS-protokolla käyttää eri porttia kuin HTTP-protokolla. HTTP-protokolla käyttää yleisesti ottaen porttia 80, kun taas HTTPS-protokolla käyttää yleisesti ottaen porttia 443 (Fielding ym., 2022).

2 MITEN HAAVOITTUVUUKSIA HYVÄKSIKÄYTETÄÄN?

Tutkimuksen seuraava luku keskittyy haavoittuvuuksien hyväksikäyttöön. Tutkimuksessa ei ole mahdollista käsitellä kaikkia haavoittuvuuksia, mitä verkkosovelluksiin kohdistuu, joten tutkimus on rajattu OWASP Top Ten- listaukseen (OWASP, 2021). OWASP Top Ten on noin neljän vuoden välein päivitettävä lista kriittisimmistä verkkosovelluksien haavoittuvuuksista. Uusin versio, jota tässä tutkimuksessa käytetään, on vuodelta 2021. Listalta löytyvät kymmenen haavoittuvuutta pyritään jakamaan Lin ja Xuen (2014) määrittelemiin kolmeen kategoriaan: syötteen validointiin, istunnon hallintaan sekä haavoittuvuuksiin loogisessa rakenteessa. Koska kaikki OWASP:n (2021) listaamat haavoittuvuudet eivät ole kategorisoitavissa Lin ja Xuen (2014) määrittelemiin kategorioihin, on tutkielmassa luotu yksi uusi kategoria: haavoittuvuudet alustassa ja sen rakenteessa. Nämä haavoittuvuudet koskevat itse alustan konfigurointia, esimerkiksi turvallisten komponenttien käyttöä. Seuraavassa taulukossa haavoittuvuudet on esitelty kriittisyysjärjestyksessä englanniksi sekä suomeksi.

Engl.	**Suom.**
Broken Access Control	Rikkinäinen pääsynhallinta
Cryptographic Failures	Kryptografiset häiriöt
Injection	Injektio
Insecure Design	Turvaton suunnittelu
Security Misconfiguration	Virheellinen turvallisuuskonfiguraatio
Vulnerable and Outdated Components	Haavoittuvaiset ja vanhentuneet komponentit
Identification and Authentication Failures	Identifioinnin ja autentikoinnin epäonnistumiset
Software and Data Integrity Failures	Ohjelmiston ja datan eheyden epäonnistumiset
Security Logging and Monitoring Failures	Turvallisuuslokien ja -monitoroinnin epäonnistumiset
Server-Side Request Forgery	Palvelinpuolen pyynnön väärentäminen

Taulukko 1. (*OWASP*, 2021.) Kriittisimmät haavoittuvuudet.

Haavoittuvuudet ovat suurimmalta osin vapaasti suomennettuja, sillä suoria suomennoksia monille haavoittuvuuksista ei löydy. Tutkielmassa käytetään haavoittuvuuksien osalta suomennettuja nimityksiä. Seuraavissa luvuissa käsitellään, miten haavoittuvuuksia hyväksikäytetään, sekä miten haavoittuvuuksia tulisi ehkäistä.

Haavoittuvuudet verkkosovelluksissa viittaavat uhkiin, jotka liittyvät vialliseen tai puutteelliseen ohjelmiston suunnitteluun, koodaamiseen, testaamiseen sekä implementaatioon (Rafique, Humayun, Hamid, Abbas, Akhtar, Iqbal, 2015). Rafiquen ym. (2015) mukaan verkkosovellukset ovat myös alttiimpia haavoittuvuuksille johtuen niiden julkisesta verkkokäyttöliittymästä. Seuraukset haavoittuvuuksien hyväksikäytöstä voivat olla esimerkiksi arkaluontoisen tiedon vuotaminen, joka voi johtaa taloudellisiin tappioihin sekä eettisiin ja juridisiin seurauksiin (Li & Xue, 2011).

2.1 Syötteen validointiin liittyvät haavoittuvuudet

Kun syötteitä ei riittävällä tasolla tarkasteta ja validoida, hyökkääjät voivat luoda haavoittavia syötteitä verkkosovellukselle saadakseen pääsyn verkkosovelluksen resursseihin (Li & Xue, 2014). OWASP:n (2021) esittämässä kymmenen kriittisimmän haavoittuvuuden listauksessa tähän kategoriaan kuuluvat injektiot sekä palvelinpuolen pyynnön väärentäminen. Injektioista tässä tutkimuksessa tarkastellaan SQL-injektioita sekä sivustojenvälistä komentosarjahyökkäystä (engl. *Cross-Site Scripting*).

SQL-injektiossa hyökkääjä onnistuu sisällyttämään syötteessä SQL-lausekkeen, jonka sovellus toteuttaa (Kieyzun, Guo, Jayaraman, Ernst, 2009). SQL-injektiossa siis onnistutaan toteuttamaan SQL-lauseke esimerkiksi verkkosovelluksen hakukentästä. Tällä tavoin tietokannasta voidaan saada haettua arkaluontoista tietoa, kuten käyttäjätunnukset ja salasanat. Uhka on vakava, sillä huonolla sanitoinnilla hyökkääjä voi saada jopa rajattoman pääsyn tietokantaan (Buja, Jalil, Ali, Rahman, 2014). Seuraavan kuvan (Shar & Tan, 2012c, s.3) avulla esitetään ote haavoittuvaisesta PHP-koodista, ja esimerkki haavoittuvaisen koodin hyväksikäytöstä. Esimerkissä on tietokantataulu nimeltä ”user”, ja siinä sarakkeet ”name” ja ”pwd”, tarkoittaen käyttäjänimeä ja salasanaa. SQL-injektio on mahdollinen, kun käyttäjän HTTP-pyynnön mukana tulleet parametrit käsitellään osana SQL-hakulauseketta.

```
$query = “SELECT info FROM user WHERE name =
  ‘$_GET[“name”]’ AND pwd = ‘$_GET[“pwd”]’”;
```

Kuvio 2. (Shar & Tan, 2012c, s. 3) Esimerkki haavoittuvaisesta PHP-koodista.

Yllä olevassa ohjelmistokoodissa query- muuttujaan sijoitetaan käyttäjän syötteen mukaan parametrit ”name” ja ”pwd”. Tarkoituksena on hakea ”info”-rivi taulusta ”user”, jossa nimi ja salasana täsmäävät syötteessä annettuihin parametreihin. Autentikointi voidaan kiertää syötteellä, jossa ”name” parametrille syötetään arvoksi ”x’ OR ‘1’=‘1”. Tällöin SQL-hakulausekkeesta muodostuu seuraavanlainen lauseke:

”SELECT info FROM user WHERE name = ’x’ OR ’1’=’1’ AND …”.

Hakulausekkeessa esitetty totuuslauseke 1=1 on aina totta, joten SQL-lausekkeen avulla hyökkääjä voi päästä käsiksi käyttäjätietoihin ilman oikeita käyttäjätietoja (Shar & Tan, 2012c). Lausekkeeseen voidaan myös lisätä merkit ”- -”, indikoiden SQL-syntaksin mukaan kommenttia, jota ei ajeta. Tällää tavoin voidaan kiertää esimerkiksi salasanan tarkastus vastaavanlaisilla parametreillä (Singh, Gupta, Singh, Ranjan, 2021).

Sivustojenvälisellä komentosarjahyökkäyksellä tarkoitetaan injektiota, jossa hyökkääjä onnistuu injektoimaan haitallisen komentosarjan luotetulle sivustolle. Haitallisen komentosarjan avulla hyökkääjällä on mahdollisuus päästä käsiksi arkaluontoiseen tietoon, kuten evästeihin ja istuntotunnisteisiin (Deepa & Thilagam, 2016).

Sivustojenväliset komentosarjahyökkäykset voidaan jakaa kolmeen eri kategoriaan niiden ominaisuuksien perusteella: ei-pysyvä (engl. *Non-persistent*), pysyvä (engl. *Persistent*) ja DOM-pohjainen (engl. *DOM-based*) (Rodríguez, Torres, Flores sekä Benavides, 2019). Kirjallisuudessa on esitetty monia eri nimityksiä sivustojenvälisten komentosarjahyökkäysten ei-pysyvälle ja pysyvällä variantille, mutta tässä tutkimuksessa nimityksinä käytetään Rodríguezin, Torresin, Floresin sekä Benavidesin (2019) esittämiä termejä.

Ei-pysyvässä sivustojenvälisessä komentosarjahyökkäyksessä hyökkääjä onnistuu haitallisen komentosarjan suorittamisessa esimerkiksi URL-osoitteeseen upotetun komentosarjan avulla (Rodríguez ym., 2020). Hyökkääjä pystyy komentosarjan suorituksen jälkeen esiintymään käyttäjänä, mikäli haitallisen komentosarjan avulla hyökkääjä onnistuu saamaan käyttäjän istuntotunnisteet tai evästetiedot haltuunsa. Toisin kuin DOM-pohjainen komentosarjahyökkäys, ei-pysyvä hyökkäys on palvelinpuolen haavoittuvuus, sillä haitallinen ohjelmistokoodi jäsennetään palvelinpuolella, eikä asiakaspuolella (Liu, Zhang, Chen, Zhang, 2019)

Pysyvässä sivustojenvälisessä komentosarjahyökkäyksessä hyökkääjä syöttää haitallisen ohjelmistokoodin suoraan haavoittuvaiselle sivustolle . Pääasiallinen riski tässä hyökkäyksessä kohdistuu siihen, että haitallinen komentosarja tallentuu verkkosivulle, ja jokainen vierailija, joka istunnon aikana aktivoi komentosarjan, ajaa tämän haitallisen komentosarjan kokonaisuudessaan sellaisena kuin se on kirjoitettu (Marashdih & Zaaba, 2017; Rodríguez ym., 2020).

DOM-pohjaisessa sivustojenvälisessä komentosarjahyökkäyksessä hyökkääjä luo linkin, jossa haitallinen komentosarja on. Kun käyttäjä avaa kyseisen linkin, ajetaan haitallinen komentosarja käyttäjän verkkoselaimessa. Kyseistä komentosarjaa ei missään vaiheessa ladata osaksi hyökättävän verkkosivun

lähdekoodia. DOM-pohjaisessa sivustojenvälisessä komentosarjahyökkäyksessä haitallinen komentosarja asentaa haitallisen ohjelmistokoodin käyttäjän verkkoselaimen tiedostoon, ja ajaa ohjelmistokoodin ilman varmennusta (Rodríguez ym., 2020).

Palvelinpuolen pyynnön väärentämisessä hyökkääjä onnistuu pakottamaan palvelinpuolen järjestelmän lähettämään esimerkiksi HTTP-pyynnön paikkaan, johon sen ei ole tarkoitettu lähettävän pyyntöjä, kuten sisäverkkoon palvelinpuolen muihin järjestelmiin. Riippuen käyttötavasta, hyökkääjä voi myös lähettää pyynnöt ulkoisiin järjestelmiin. Hyökkäyksessä hyökkääjä luo pyynnön, jonka palvelin tulkitsee luotettavaksi, ja palvelin tekee oman pyyntönsä hyökkääjän valitsemaan resurssiin (Luo, 2019). Luon (2019) mukaan hyökkäystä on vaikea havaita, sillä haitallisen pyynnön tekijä on palvelinpuolen järjestelmä.

Palvelinpuolen pyynnön väärentämiset voidaan jakaa Jabiyevin, Mirzaein, Kharrazin ja Kirdan (2021) mukaan kahteen eri kategoriaan: sisäiseen kaistaan (engl. *in-band*) ja ulkoiseen kaistaan (engl. *out-band*). Heidän mukaansa sisäistä kaistaa käyttävässä hyökkäyksessä hyökkääjä haitallinen tietosisältö on sisällettynä palvelimelle menevässä pyynnössä, kun taas ulkoista kaistaa käyttävässä hyökkäyksessä hyökkääjä lähettää pyynnössään viitteen haitalliseen tietosisältöön. Palvelinpuolen pyynnön väärennöksiä voidaan käyttää hyväksi myös esimerkiksi palvelunestohyökkäyksissä sekä alkuperän pesuhyökkäyksissä (Pellegrino, Catakoglu, Balzarotti, Rossow, 2016).

2.2 Istunnon hallintaan liittyvät haavoittuvuudet

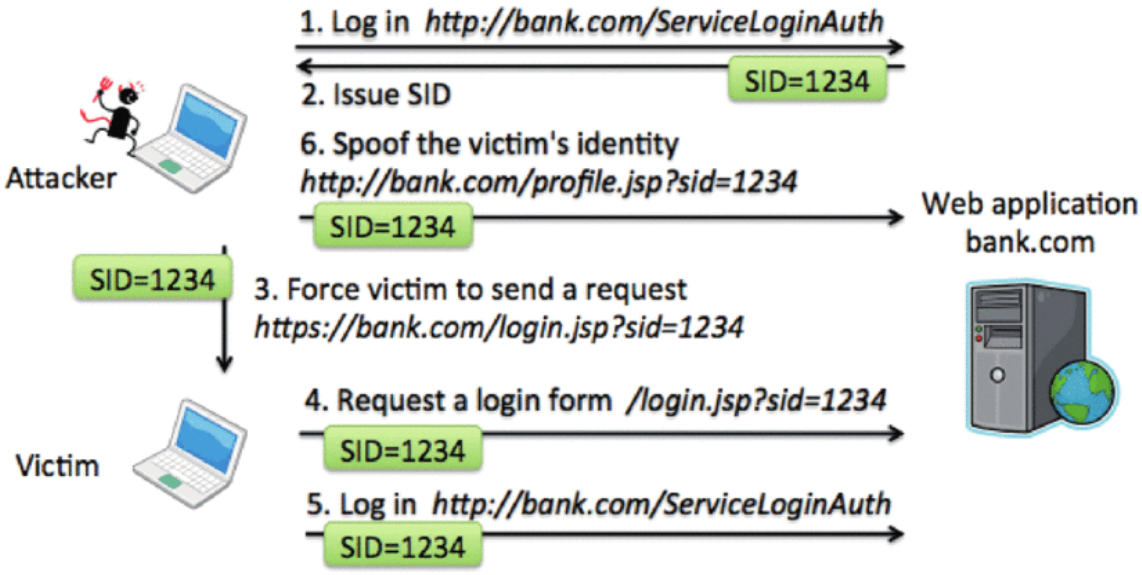
Istunnon hallinta suoritetaan joko asiakaspuolella esimerkiksi evästeiden avulla, tai palvelinpuolella tiedostomuodossa tai tietokantaan tallennettuna (Li & Xue, 2014). Istuntotunnisteiden avulla verkkosovellus pystyy tunnistamaan käyttäjänsä, eikä tätä istuntotunnisteen tulisi olla muiden saatavilla. Istuntotunnisteisiin ja evästeisiin pohjautuva istunnonhallinta on virhealtis ratkaisu, ja OWASP:n (2021) kymmenen kriittisimmän verkkosovellusten haavoittuvuuksien listauksesta tähän kategoriaan kuuluvat kaksi haavoittuvuutta: rikkinäinen pääsynhallinta sekä identifioinnin ja autentikoinnin epäonnistumiset. OWASP:n (2021) mukaan rikkinäinen pääsynhallinta sisältää muun muassa sivustojenvälisen pyyntöväärennöksen (engl. *Cross-site request forgery*), jota tarkastellaan tutkielmassa rikkinäisen pääsynhallinnan osalta. Identifioinnin ja autentikoinnin epäonnistumisia tarkastellaan istunnon kiinnittämisen (engl. *Session fixation*) näkökulmasta.

Sivustojenvälisessä pyyntöväärennöksessä hyökkääjä saa kirjautuneen käyttäjän lähettämään hyökkääjän muovaaman HTTP-pyynnön haavoittuvaiselle sivulle (Jovanovic, Kirda, Kruegel, 2006). Hyökkäyksen uhrilla on kirjautumisen takia käytössään validi istuntotunniste, jota selain käyttää tunnistaakseen käyttäjän. Tällä hyökkäyksellä hyökkääjä voi päästä käsiksi arkaluontoiseen tietoon tai suorittaa komentosarjoja esiintyen kirjautuneena käyttäjänä (Jovanovic ym. 2006).

Sivustojenvälisiä pyyntöväärennöksiä voidaan kategorisoida kahteen eri kategoriaan: heijastettuun (engl. *reflected*) ja tallennettuun (engl. *stored*). Heijastetussa hyökkäyksessä tietosisältö toimitetaan kolmannen osapuolen kautta, kuten toiselta verkkosivustolta, kun taas tallennetussa hyökkäyksessä tietosisältö on osa luotettua sivustoa, kuten esimerkiksi blogia (Shahriar & Zulkernine, 2010). Molempien hyökkäysten toimintatavat ovat kuitenkin hyvinkin samanlaiset. Hyökkäyksessä hyökkääjä esimerkiksi huijaa käyttäjää avaamaan linkin, joka sisältää haitallisen tietosisällön, jolloin haitallinen HTTP-pyyntö lähetetään haavoittuvaiselle sivustolle. Tämä HTTP-pyyntö voidaan sisällyttää linkin sijaan esimerkiksi sivuston <img> -elementissä, jolloin verkkosivun yrittäessä hakea resurssia, lähetetään haitallinen HTTP-pyyntö (Jovanovic ym., 2006).

Istunnon kiinnittämishyökkäyksessä hyökkääjä onnistuu kaappaamaan käyttäjän istunnon. Hyökkäyksessä hyökkääjä luo istunnon ja saa istuntotunnisteen istunnolleen, onnistuu saamaan käyttäjän kirjautumaan tällä istuntotunnisteella verkkosovellukseen ja lopuksi kaappaa tämän istunnon, koska tietää sen istuntotunnisteen (Takamatsu, Kosuga, Kono, 2012). Hyökkäys siis alkaa jo ennen kuin käyttäjä on kirjautunut sivustolle. Vlsaggion ja Blasion mukaan istunnon hallinnan mekanismit voidaan jakaa kahteen eri kategoriaan: Sallivaan (engl. *Permissive*) ja tiukkaan (engl. *Strict*). Sallivan mekanismin systeemi hyväksyy uudet tekstialkiot, kun taas tiukan mekanismin systeemi hyväksyy vain tiedetyt, aiemmin luodut tekstialkiot (Vlsaggio & Blasio, 2010). Tämä tarkoittaa sitä, että sallivan mekanismin istunnon hallinta hyväksyy minkä tahansa asiakkaan asettaman istuntotunnisteen, kun taas tiukka hyväksyy vain palvelinpuolen jo aikaisemmin määrittelemät istuntotunnisteet.

Alla olevassa kuvassa on havainnollistettu hyökkäyksen kulkua.



Kuvio 3. (Takamatsu, Kosuga, Kono. 2012, s.2) Istunnon kiinnittämishyökkäys.

Kuvassa hyökkääjä luo istunnon, jolloin hän saa palvelinpuolelta uniikin istuntotunnisteen. Hyökkääjä sen jälkeen onnistuu saamaan uhrinsa käyttämään tätä istuntotunnistetta kirjautuessaan palveluun, esimerkiksi sisällyttämällä

istuntotunnisteen sivuston URL:ssä. Uhri kirjautuu verkkosovellukseen käyttäen tätä istuntotunnistetta, jonka jälkeen hyökkääjä ottaa istunnon haltuunsa, koska tietää istunnon istuntotunnisteen. Hyökkääjä voi myös hyödyntää sivustojenvälistä komentosarjahyökkäystä asettaakseen käyttäjän istuntoon istuntotunnisteen, sillä useammat verkkosovellukset eivät nykyään hyväksy istuntotunnistetta URL:n välityksellä (Johns, Braun, Schrank, Posegga, 2011).

2.3 Loogiseen rakenteeseen liittyvät haavoittuvuudet

Loogisella rakenteella viitataan verkkosovelluksen toimintalogiikkaan ja operaatioiden oikeaoppiseen järjestykseen . Esimerkiksi autentikointi tulee suorittaa ennen kirjautumista, jotta vain kirjautuneet ja autentikoidut käyttäjät voivat käyttää heille tarkoitettuja ominaisuuksia ja toiminnallisuuksia (Li & Xue, 2014). Mikäli hyökkääjä kykenisi kiertämään autentikointiprosessin ja pystyisi suorittamaan vain tunnistetuille käyttäjille tarkoitettuja toiminnallisuuksia, tai esimerkiksi verkkokaupan verkkosovelluksessa hyökkääjä pystyisi käyttämään alennuskuponkia useammin kuin kerran, olisi kyseessä loogiseen rakenteeseen liittyvä haavoittuvuus. (Li & Xue, 2014.) OWASP:n (2021) kymmenen kriittisimmän haavoittuvuuden listalta tähän kategoriaan voidaan tunnistaa kryptografiset häiriöt.

Kryptografiset häiriöt voidaan luokitella häiriöiksi verkkosovelluksen kryptografisten algoritmien toiminnassa. Esimerkiksi salasanat ja henkilötiedot tulee kuljettaa salattuja kanavia pitkin verkossa, ja mikäli tätä käytäntöä laiminlyödään, on kyseessä kryptografinen epäonnistuminen. Lisäksi heikkojen tai vanhentuneiden kryptografisten algoritmien käyttö tai salasanojen tallentaminen tietokantoihin tekstimuodossa hash-tiivisteen sijaan voidaan laskea kryptografiseksi häiriöksi. (*A02 Cryptographic Failures - OWASP Top 10:2021*, 2021.)

Kryptografiset häiriöt voidaan kuvitella enemmänkin oireeksi kuin pohjimmaiseksi syyksi, ja mikä tahansa häiriö, joka altistaa salatun tiedon joutumisen vääriin käsiin tekstimuodossa voidaan laskea kryptografiseksi häiriöksi. Kryptografiset häiriöt ovat usein inhimillisistä virheistä johtuvia implementaatio-ongelmia, esimerkiksi yksi ylimääräinen koodirivi Applen SSL/TLS- implementaatiossa altisti miljoonia laitteita väliintulohyökkäykselle (Lazar, Chen, Wang, Zeldovich, 2014). Väliintulohyökkäyksessä kolmas osapuoli tunkeutuu asiakkaan ja palvelimen välille, ja pystyy täten muuttamaan tai tutkimaan kulkevaa informaatiota (Conti, Dragoni, Lesyk, 2016).

Kryptografisia häiriöitä hyväksikäytetään yleisesti ottaen muiden haavoittuvuuksien kanssa yhteistoimin. Mikäli hyökkääjä onnistuu penetroimaan verkkosovelluksen ja saa esimerkiksi palvelimen käyttäjätiedot haltuunsa, voi hän hyödyntää esimerkiksi tekstinä tallennettuja tai heikolla hash-tiivisteellä salattuja salasanoja (Aljabri, Aldossary, Al-Homeed, Alhetelah, Althubiany, Alotaibi & Alsaqer, 2022). Mikäli verkkosovellus käyttää viestinvälityksessä heikkoa salausalgoritmia, hyökkääjä voi myös käyttää väliintulohyökkäystä. Lisäksi heikot salausalgoritmit sekä riittämätön satunnaisuus voivat altistaa verkkosovelluksen väsytyshyökkäykselle (Lazar ym., 2014). Väsytyshyökkäyksessä

istuntotunnistetta yritetään arvata systemaattisesti yksi kerrallaan, yleensä jonkin ohjelmiston avustuksella (Gautam & Jain, 2015).

2.4 Alustaan ja alustan rakenteeseen liittyvät haavoittuvuudet

Vaikka verkkosovellukset ovat toisistaan hyvinkin eriäviä toiminnallisuuksien ja arkkitehtuurin suhteen, on niille yleensä yhteistä alusta, jolle ne rakennetaan, sillä palvelinpuolen alustaratkaisuja on rajallinen määrä tarjolla. Yksi käytetyimmistä kombinaatioista on Apachen palvelin, MySQL- tietokanta sekä PHP- ohjelmointikieli, sillä nämä ovat avoimen lähdekoodin palveluja. (Eshete, Villafiorita, Weldemariam. 2011.) Haavoittuvuudet alustassa ja alustan rakenteessa viittaavat verkkosovelluksen alustan konfigurointiin ja suunnitteluun. Se kattaa alustan arkkitehtuurin ja suunnitellut toiminnallisuudet, turvallisuus- ja toiminnallisuuskonfiguroinnin sekä palvelimen, palvelinohjelmiston ja sovelluskehyksen konfiguroinnin verkkosovelluksen toimintaympäristöön . Tähän kategoriaan OWASP:n (2021) kymmenen kriittisimmän haavoittuvuuden listauksesta sopivat viisi haavoittuvuutta: turvaton suunnittelu, virheellinen turvallisuuskonfiguraatio, haavoittuvaiset ja vanhentuneet komponentit, ohjelmiston ja datan eheyden epäonnistumiset sekä turvallisuuslokien ja -monitoroinnin epäonnistumiset.

Turvattomalla suunnittelulla tutkielmassa tarkoitetaan alustan suunnitteluun ja arkkitehtuuriin liittyviä haavoittuvuuksia. Turvaton suunnittelu ja turvaton implementaatio ovat kaksi erillistä haavoittuvuuden tyyppiä, sillä täydellinen implementaatio turvattomaan suunnitteluun altistaa verkkosovelluksen silti turvattomasta suunnittelusta johtuviin haavoittuvuuksiin, ja turvaton implementaatio turvalliseen suunnitteluun aiheuttaa omat haavoittuvuutensa verkkosovellukselle. OWASP:n (2021) mukaan esimerkiksi sensitiivistä informaatiota sisältävien häiriöviestien välitys tai turvattomasti tallennettujen kirjautumistietojen säilytys ovat esimerkkejä turvattomasta suunnittelusta. Mikäli verkkosovelluksen häiriöviesti paljastaisi esimerkiksi jonkin resurssin polun voisi hyökkääjä käyttää tätä tietoa hyväkseen laatiessaan hyökkäystä verkkosovellukseen. Kirjautumistietojen turvatonta säilytystä voidaan taas hyödyntää esimerkiksi oikeuksien nostamiseen tai käyttäjätietojen varastamiseen. Lisäksi sovelluksen toiminnallisuudet tulisi suunnitella turvalliskeksi todettujen kirjastojen avulla, sillä Decanin, Mensin ja Constantinoun (2018) mukaan noin neljännes ladattavissa olevista kirjastoista sisältävät haavoittuvuuksia.

Virheellisellä turvallisuuskonfiguraatiolla tutkimuksessa tarkoitetaan kaikkea verkkosovelluksen turvallisesta konfiguraatiosta poikkeavaa konfiguraatiota. Tämä voi tarkoittaa esimerkiksi avoimia portteja, oletuskäyttäjätunnuksien ja salasanojen käyttöä, turvallisuusotsakkeiden käyttämättömyyttä tai ohjelmiston haavoittuvuutta johtuen vanhentuneesta versiosta. Esheten ym. (2011) mukaan AMP eli Apache, MySQL ja PHP- ympäristössä yleisiä haavoittuvuuksien aiheuttajia ovat konfiguraatiotiedostot ja niiden vääränlainen konfiguraatio. Näistä konfiguraatiotiedostojen turvattoman konfiguraation takia hyökkääjä voi esimerkiksi saada arvokasta tietoa ohjelmiston ja palvelimen versioista ja

hyökätä juuri näille versioille tiedossa olevilla hyökkäyksillä verkkosovellukseen tai saada häiriöviestissä tietoa esimerkiksi tietokantapalvelimen sarakkeista (Eshete ym., 2011). Esheten ym. (2011) esittämien turvallisuuskonfiguraatiovirheiden lisäksi hyökkääjä voi hyödyntää haavoittuvaisia ja vanhentuneita komponentteja, joihin hyökkääjät voivat käyttää tunnettuja hyökkäyksiä, sekä monitoroinnin ja testaamisen puutteellisuutta pysyäkseen näkymättömänä (Loureiro, 2021).

Haavoittuvaiset ja vanhentuneet komponentit antavat hyökkääjälle erinomaisen hyökkäysvektorin, sillä vanhentuneisiin komponentteihin, oli kyse sitten ohjelmistosta tai laitteistosta, löytyy internetistä suuri määrä kirjattuja haavoittuvuuksia sekä proof-of-concept-tiedostoja, joita muokkaamalla hyökkääjä voi hyökätä verrattain pienellä vaivalla verkkosovellukseen. Uhka ei jää vain palvelinpuolen laitteistoon ja ohjelmistoon, sillä vanhentuneita komponentteja JavaScript-kirjastoissa voidaan hyödyntää hyökkäyksessä esimerkiksi sivustojenvälisessä komentosarjahyökkäyksessä (Lauinger, Chaabane, Arshad, Robertson, Wilson, Kirda. 2017). Aljabrin ym. (2022) mukaan haavoittuvuudesta tekee erityisen vaarallisen se, että vanhentuneet ohjelmistokomponentit ajetaan samoilla käyttöoikeuksilla kuin itse ohjelmat.

Ohjelmiston ja datan eheyden epäonnistumisella tarkoitetaan organisaation toimia, jotka eivät suojaa ohjelmistoa ja dataa tiedon muokkaukselta tai tuhoamiselta. Haavoittuvuus voi syntyä esimerkiksi automaattisista päivityksistä, mikäli eheyden tai sertifikaattien tarkastusta ei riittävällä tasolla tehdä. Historiassa yksi merkittävimmistä ohjelmiston ja datan eheyteen liittyvistä hyökkäyksistä oli vuoden 2020 SolarWinds- hyökkäys, jossa hyökkääjät onnistuivat sisällyttämään luotetun sovelluksen päivitykseen omat haittaohjelmansa (Alkhadra, Abuzaid, AlShammari, Mohammad. 2021). Ohjelmiston ja datan eheyden epäonnistumisilla voi olla suurempi merkitys kuin datan luottamuksellisuuden häiriöillä, sillä kuten SolarWinds- esimerkkitapauksessa, datan luvaton muokkaaminen antaa hyökkääjälle mahdollisuuden sisällyttää omaa pahantahtoista sisältöä, kuten haitallisia komentosarjoja tai kokonaisia haittaohjelmia hyökkäyksen kohteena olevan organisaation ohjelmistoon tai dataan (Sivathanu, Wright, Zadok. 2005)

Turvallisuuslokien ja -monitoroinnin epäonnistumiset liittyvät hyökkäyksen oikea-aikaiseen huomaamiseen ja estämiseen. Verkkosovelluksen tulisi kirjata ylös muun muassa kirjautumiset, epäonnistumiset kirjautumisessa sekä korkean prioriteetin tiedonvälitykset. Lisäksi näitä tietoja tulisi monitoroida joko ihmisen tai automatiikan avulla. Tällä tavoin hyökkäykset voidaan huomata jopa reaaliajassa. Mikäli verkkosovellus ei pidä asianmukaista lokia ja monitorointia lokitapahtumista, voi hyökkääjä edetä hyvinkin pitkälle verkkosovelluksessa herättämättä huomiota.(*A09 Security Logging and Monitoring Failures - OWASP Top 10:2021*, 2021.) Tämä haavoittuvuus ei suoranaisesti anna hyökkääjälle lisää hyökkäysvektoreita, vaan helpottaa hyökkääjän tekemiä hyökkäyksiä. Suuri ongelma monitoroinnissa on datan määrä, ja kuinka haastavaa sitä on manuaalisesti monitoroida(Aljabri ym., 2022)

3 MITEN HAAVOITTUVUUKSIA TULISI EHKÄISTÄ?

Tutkielman seuraava luku keskittyy esitettyjen haavoittuvuuksien ehkäisykeinoihin. Lin ja Xuen (2014) mukaan tekniikat haavoittuvuuksien ehkäisyyn voidaan jakaa kolmeen eri kategoriaan: uusien verkkosovellusten turvalliseen rakenteeseen, legacy-sovellusten turvallisuusanalyysiin ja -testaukseen sekä legacy-sovellusten ajonaikaiseen suojaamiseen, kun taas Deepan ja Thilagamin (2016) mukaan verkkosovellusten elinkaaren aikana on huomattava määrä erilaisia toimia, joita voidaan hyödyntää turvallisemman verkkosovelluksen rakennuksessa. Heidän mukaansa sovelluksen rakennuksen aikana tulisi noudattaa puolustautuvan koodauksen periaatteita, testausvaiheessa hyödyntää staattista ja dynaamista analyysiä, sekä käyttöönoton jälkeen tulisi ottaa käyttöön hyökkäyksen havainnointia tai estämistä edistäviä järjestelmiä. Staattinen ja dynaaminen analyysi eroavat siten, että staattinen analyysi suoritetaan esimerkiksi lähdekoodin katselmuksella, ja dynaaminen analyysi suoritetaan ajonaikaisina toimintoina (Li & Xue, 2014). Näitä ohjeistuksia voidaan pitää peruspilareina kaikkien verkkosovellusten haavoittuvuuksien ehkäisyssä, sillä ne ovat universaalisti implementoitavissa. Tutkielmassa haavoittuvuuksien ehkäisyssä keskitytään pääasiallisesti Deepan ja Thilagamin (2016) esittämiin keinoihin verkkosovellusten suojauksesta haavoittuvuuksia vastaan. Tutkimuksessa tarkastellaan etenkin haavoittuvuuksien havainnointia, sekä annetaan esimerkkejä kirjallisuudessa esitetyistä suojauskäytänteistä, jotta suojausmekanismit olisivat mahdollisimman helposti ymmärrettävissä.

3.1 Syötteen validointiin liittyvät haavoittuvuudet

Kirjallisuudessa on esitetty monenlaisia ratkaisuja syötteen validointiin liittyvien haavoittuvuuksien ehkäisykeinoiksi (esim. Brinhosa ym., 2008, 2012; Li & Xue, 2014). Yhteistä näille ratkaisuille on se, että syöte halutaan tarkastaa

mahdollisimman aikaisessa vaiheessa, jotta haitallinen tietosisältö ei pääse penetroimaan järjestelmää syvälle. Esimerkiksi Brinhosa, Westphall ja Westphall (2008) esittävät mallissaan ratkaisun, jossa syöte lähetetään XML-ohjelmalle tarkistettavaksi, ja myöhemmässä työssä Brinhosa, Westphall ja Westphall (2012) esittävät ratkaisun, jossa syöte tarkistetaan erillisellä palvelimella ennen prosessointia. Itse syötteen tarkastus on haastavaa, sillä kaikkia erikoismerkkejä ei voida vain asettaa suoraan kieltolistalle, vaan monimutkaiset algoritmit ovat tarpeen syötteen tarkastuksessa. Esimerkiksi merkkiä ” ’ ” ei voida asettaa välttämättä kieltolistalle, sillä se esiintyy kansainvälisesti useissa eri nimissä. Tämä hankaloittaa esimerkiksi SQL-injektioiden vastatoimia, sillä hyökkääjä voi käyttää SQL-injektiossa merkkiä karatakseen odotetusta SQL-hakulausekkeesta ja syöttää pahantahtoisen syötteen, kuten aikaisemmassa luvussa esitetyn hakulausekkeen.

SQL-injektioiden vastatoimet koostuvat puolustautuvasta ohjelmoinnista, injektion havaitsemisesta sekä ajonaikaisesta vastatoimenpiteistä (Shar & Tan, 2012c). Puolustautuvalla ohjelmoinnilla tarkoitetaan esimerkiksi dynaamisten hakulausekkeiden korvaamista parametrisoiduilla hakulausekkeilla tai tallennetuilla toimintosarjoilla, datan tyypin validointia sekä sallittujen ja ei-sallittujen merkkien suodatusta. Mikäli parametrit ovat sidottuja määriteltyyn SQL-struktuuriin, ylimääräinen SQL-hakulauseke tai sen osa on mahdotonta saada osaksi hakulauseketta (Shar & Tan, 2012b).

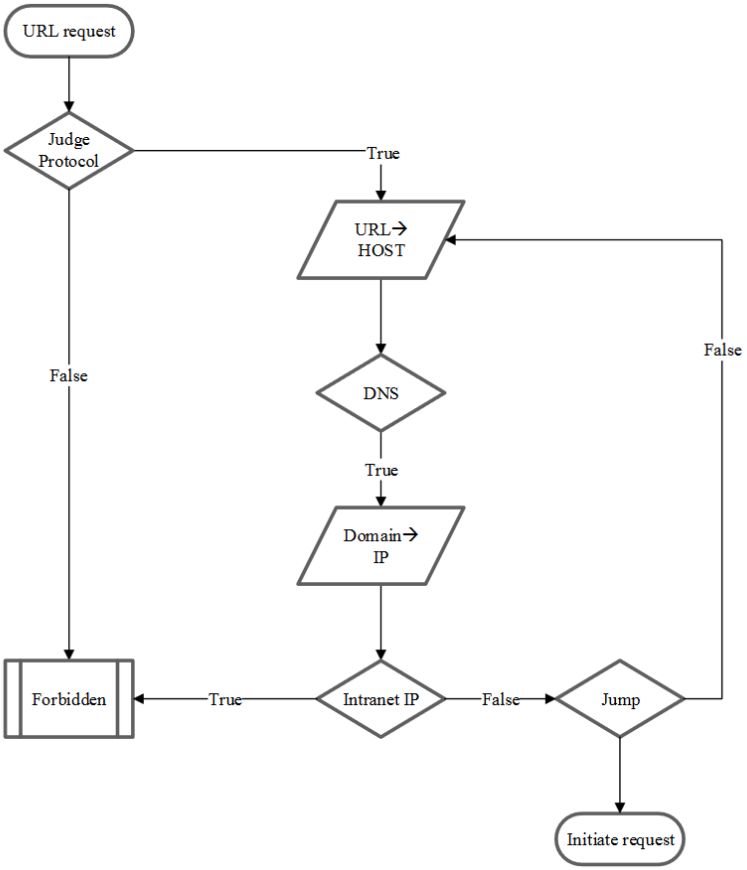
SQL-injektion havaitsemisesta on esitetty kirjallisuudessa erilaisia malleja. Buja, Jalil, Ali ja Rahman (2014) esittävät mallin, jonka avulla injektiot voidaan havaita verkkosovelluksesta hakurobottia hyödyntämällä. Kuitenkin Sharin ja Tanin (2012) mukaan erilaisia testausohjelmia on markkinoilla tarjolla hyvinkin paljon, ja verkkosovelluksen kehittäjien tulisi hyödyntää näitä. Automatisoiduilla työkaluilla saadaan testattua erilaisia hyökkäysvektoreita huomattavasti tehokkaammin kuin manuaalisella testaamisella.

Sivustojenväliselle komentosarjahyökkäykselle on esitetty monia erilaisia puolustuskeinoja, joista tutkimuksessa perehdytään kahteen erilaiseen keinoon. Shar ja Tan (2012b) esittävät algoritmin mahdollisesti vaarallisen syötteen havainnointia ja estämistä varten. Heidän mukaansa ohjelmiston lähdekoodista tulisi etsiä jokainen solmu, jossa käyttäjän syötettä käsitellään HTML-koodina. Kun syötteen käsittelykohdat on löydetty, ajetaan algoritmi, joka tutkii HTML-kontekstin syötteen ympäriltä ja tarkistaa, ettei syötteessä ole ylimääräisiä HTML-tunnisteita. Mikäli syötteestä löytyisi ylimääräisiä HTML-tunnisteita, syöte havaittaisiin vaaralliseksi eikä sitä käytettäisi validina syötteenä. Tuoreemmassa tutkimuksessa Lei, Chen, He ja Li (2020) esittävät ratkaisun, jossa he hyödyntävät neuroverkkoja ja pitkäkestoista lyhytkestomuistia saavuttaakseen automatisoidun syötteentarkastusohjelman. Lein ym. (2020) esittämä ohjelma havaitsi hyökkäykset yli 99 prosentin tarkkuudella, kilpailevien teknologioiden saavuttaessa 95,6–97,2 prosentin tarkkuuden.

Tutkimuksessaan Shar ja Tan (2012b) esittävät yleispäteviä ohjeita sivustojenvälisen komentosarjahyökkäyksen puolustuskeinoksi. Kuten SQL-injektion kohdalla, puolustusmekanismit koostuvat heidän mukaansa puolustautuvasta

ohjelmoinnista, haavoittuvuuden havaitsemisesta staattisen ja dynaamisen testauksen avulla sekä ajonaikaisista vastatoimenpiteistä. Heidän mukaansa puolustautuva ohjelmointi koostuu sallittujen ja ei-sallittujen merkkien suodattamisesta eri keinoin. Sovelluksen tulisi suodattaa merkit, joilla on asiakaspuolen järjestelmässä jokin erikoismerkitys, kuten HTML-tunnisteet, sekä muuttaa tai poistaa erikoismerkit syötteestä. Sharin ja Tanin (2012b) mukaan kiellettyjen merkkien lähestymistapa ei ole yhtä turvallinen kuin sallittujen merkkien lähestymistapa, mutta sallittujen merkkien lähestymistavalla saatetaan aiheuttaa monien validien syötteiden hylkäys. Sallittujen merkkien lähestymistavassa määritellään erikseen syötteen sallitut merkit, sekä kiellettyjen merkkien lähestymistavassa määritellään erikseen kielletyt merkit.

Palvelinpuolen pyynnön väärentämisen ehkäisyksi Luo (2019) esittää yksinkertaisen prosessikaavion, jota verkkosovelluksen tulisi noudattaa.



Kuvio 4. (Luo, 2019, s.4) Resurssin tarkastuksen prosessi.

Prosessikaaviossa ensimmäisenä syötteestä tarkastetaan protokolla. Mikäli protokolla on kielletty, tulee yhteys kieltää. Mikäli yhteys on sallitulla protokollalla, tulee URL-osoitteesta parsia halutun resurssin isäntäosoite. Tämän jälkeen osoitetta vastaava IP-osoite haetaan DNS-protokollalla, ja mikäli osoite on osa organisaation intranettiä, yhteys kielletään. Tällä tavoin voidaan asettaa selkeät rajoitukset resurssien hakemiselle.

Haavoittuvuuden havaitsemiseen Al-Talak ja Abbas (2021) esittävät erilaisia keinoja koneoppimista hyödyntäen. Kuten Lein ym. (2020) tutkimuksessa sivustojenvälisestä komentosarjahyökkäyksestä, Al-Talakin ja Abbasin (2021) tutkimuksessa hyödynnetään pitkäkestoista lyhytkestomuistia. Heidän esittämällä mallilla tunnistettiin noin 96 prosenttia vaarallisista syötteistä.

3.2 Istunnon hallintaan liittyvät haavoittuvuudet

Yhteistä istunnon hallintaan liittyvien haavoittuvuuksien ehkäisylle on istuntotunnisteiden vahva implementaatio. Tutkimuksessa tutkittavien haavoittuvuuksien, eli sivustojenvälisen pyyntöväärennöksen sekä istunnon kiinnityshyökkäyksen, pohjimmaiset syyt ovat pitkälti istuntotunnisteiden käytön virheissä (Huluka & Popov, 2012). Istuntotunnisteiden tulisi olla vahvoilla kryptografisilla algoritmeilla suojattuja ja kertakäyttöisiä tunnisteita, joita ei olisi mahdollista arvata väsytyshyökkäyksillä (Huluka & Popov, 2012).

Jovanovic, Kirda ja Kruegel (2006) esittävät tutkimuksessaan yleisesti käytössä olevia vastatoimia sivustojenväliselle pyyntöväärennökselle, sekä tarjoavat oman ratkaisunsa mahdollisimman hyvälle puolustautumiselle. Heidän mukaansa yleisiä tapoja välttää sivustojenväliset pyyntöväärennökset ovat HTTP-protokollan POST-metodin käyttö GET-metodin sijaan, ”Referer”-otsakkeen tarkistus, istuntotunnisteen poisto URL-osoitteesta sekä istuntotunnisteen muuttaminen jaetuksi salaisuudeksi. Jaetuksi salaisuudeksi muutettu istuntotunniste tarkoittaa sitä, että asiakkaalla sekä palvelimella on yhteinen salausavain, joka tulee käyttää molemmilla osapuolilla istuntotunnisteen kanssa. Tätä salausavainta kutsutaan tässä kappaleessa nimellä *token.* Nämä vastatoimet ovat kuitenkin heidän mukaansa riittämättömiä, ja he esittävät ratkaisuksi välityspalvelinpohjaisen algoritmin. Jovanovicin ym. (2006) esittämässä algoritmissa tarkistetaan ensiksi, onko pyynnössä istuntotunnistetta. Mikäli ei, pyyntö lähetetään eteenpäin, sillä tunnistautumaton asiakas voi nähdä vain tunnistautumattomalle asiakkaalle saatavilla olevat resurssit. Mikäli istuntotunniste esiintyy, tarkistetaan istuntotunniste token-taulukossa, jossa esiintyvät istuntotunnisteet ja niitä vastaavat tokenit. Mikäli taulukossa ei esiinny istuntotunnisteelle tokenia, voidaan olettaa tämän olevan uusi istunto, ja uusi token luodaan tälle istuntotunnisteelle, sekä pyyntö lähetetään eteenpäin uutena istuntona. Mikäli pyynnössä on validi istuntotunniste sekä sitä vastaava token, pyyntö lähetetään eteenpäin. Täten voidaan varmistaa, että pelkällä istuntotunnisteella ei voida suorittaa sivustojenvälistä komentosarjahyökkäystä. Jovanovic ym. (2006) huomauttavat, että kyseinen algoritmi ei suojaa verkkosovellusta esimerkiksi sivustojenväliseltä komentosarjahyökkäykseltä.

Kuten aikaisempienkin haavoittuvuuksien kohdalla, istunnon hallintaan liittyvien haavoittuvuuksien automatisoitua testausta erilaisilla työkaluilla pidetään tärkeänä tekijänä verkkosovelluksen turvallisuuden kannalta (Lukanta, Asnar, Kistijantoro, 2014; Takamatsu ym., 2012). Lukanta ym. (2014) esittävät Nikto -työkalun avulla yksinkertaisen prosessin istunnon kiinnityshyökkäyksen

tunnistamiseksi, kun taas Takamatsu ym. (2012) esittävät manuaalisen sekä Amberate-työkalua hyödyntävän tavan havaita sekä istunnon kiinnityshyökkäys että sivustojenvälinen pyyntöväärennöshyökkäys. Molemmissa tutkimuksissa kohteena olevista verkkosovelluksista onnistuttiin havaitsemaan työkalujen avulla haavoittuvuuksia suorittamalla aitoja hyökkäyksiä simuloiduissa ympäristöissä.

3.3 Loogiseen rakenteeseen liittyvät haavoittuvuudet

Loogisen rakenteen haavoittuvuuksia voi olla hyvinkin vaikea huomata, sillä Lin ja Xuen (2014) mukaan niistä on vaikea löytää juurisyytä. Tämä tarkoittaa sitä, että loogiseen rakenteeseen liittyvät haavoittuvuudet ovat yleensä monimutkaisia sovelluksen toimintaan liittyviä ongelmia. Loogisen rakenteen haavoittuvuuksia voi olla vaikeata havaita ilman manuaalista testaamista, sillä haavoittuvuudet liittyvät sovelluksen toimintalogiikkaan, eivätkä välttämättä yksittäisiin ohjelmointivirheisiin.

Li ja Xue (2014) esittävät useita erilaisia ratkaisuja ohjelmiston suojaamiselle. Heidän mukaansa markkinoilla on useita erilaisia malleja ja viitekehyksiä, jotka tuovat puolustautuvan ohjelmoinnin periaatteita ohjelmistokehitykseen. He suosittelevat käyttämään Jif-pohjaisia ohjelmointikieliä, sillä ne sisältävät mahdollisuuden seurata informaatiovirtaa sekä tarjoavat yhtenäistetyn viitekehyksen. Tätä väitettä tukevat myös Deepa ja Thilagam (2016). Lazar ym. (2014) painottavat kryptografisten häiriöiden ehkäisyksi sovelluskehityksen ohjelmointivaiheessa turvallisten ohjelmistorajapintojen käyttöä sekä suosittelevat komponentteja, jotka eivät säilytä muistissa tietoa, joka olisi kytköksissä salaiseen tietoon. Lisäksi heidän mukaansa ohjelmistokoodin tulisi pystyä ajaa ilman virheitä kääntäjän tiukimmillakin asetuksilla.

Loogiseen rakenteeseen liittyvien haavoittuvuuksien havainnoinnista Felmetsger, Cavedon, Kruegel ja Vigna (2010) esittävät yhdeksi ratkaisuksi Walerohjelman käytön. Felmetsgerin ym. (2010) mukaan Walerin avulla voidaan automaattisesti löytää sovelluskohtaisia virheitä. Lisäksi Deepa ja Thilagam (2016) esittävät tutkimuksessaan kaksi eri skanneria, joilla pystytään havainnoimaan pääsynhallinnan loogista rakennetta verkkosovelluksissa. Kryptografisten häiriöiden havainnoinnista Lazar ym. (2014) esittävät useamman työkalun, joilla voidaan havaita staattisen analyysin avulla huomattava määrä kryptografisia häiriöitä, kuten riittämättömät salaukset tai kovakoodatut avaimet. Esitettyjä työkaluja tulisi käyttää monipuolisesti ohjelmiston testausvaiheessa, jotta sovelluksesta saadaan havainnoitua ja poistettua haavoittuvuuksia aiheuttavat ominaisuudet.

3.4 Alustaan ja alustan rakenteeseen liittyvät haavoittuvuudet

Scottin ja Sharpin (2002) mukaan verkkosovelluksen turvallinen suunnittelu ja ohjelmointi on haastavaa, sillä verkkosovelluksen kehityksessä käytetään montaa eri ohjelmointikieltä. Heidän mukaansa verkkosovellus tulisi suunnitella toimimaan SWAP-työkalujen, eli The Secure Web Applications Project työkalujen kanssa. SWAP-työkalut ovat heidän mukaansa työkaluja, joiden avulla pyritään ratkaisemaan sovelluskerroksen verkkoturvallisuutta korkealla tasolla.

Erilaisia turvallisuusmalleja arvioivat Dalai ja Jena (2011). He esittävät tutkimuksessaan erilaisia verkkosovelluksen turvallisuuteen liittyviä malleja ja esittävät, millaisiin haavoittuvuuksiin kyseisiä malleja voidaan hyödyntää. Heidän mukaansa verkkosovelluksen ja sen alustan konfiguraatioon liittyviin haavoittuvuuksiin suunnitteluvaiheen ratkaisuja ovat erilaisten turvallisuusmallien implementoinnit, kuten todennustoiminnallisuuksien sisällys sovellukseen sekä tiedon salattu säilytys. Lisäksi Pathak, Singh ja Sharma (2017) esittävät viitekehyksen turvalliselle verkkosovelluksen suunnittelulle käyttäen UML 2.0-notaatiota. Heidän viitekehyksessään kyse on oliosuuntautuneesta mallinnuksesta, jonka avulla korkean tason malleista kehitetään toiminnallisia komponentteja verkkosovellukseen. Pathakin ym. (2017) viitekehystä pystytään käyttämään Dalain ja Jenan (2011) esittämien turvallisuusmallien kanssa, lisäten turvallisuutta verkkosovelluksen suunnittelussa.

Turvallisuuskonfiguraatioista ja sen oikeaoppisesta implementaatiosta Eshete ym. (2011) esittävät AMP-ympäristölle SCAAMP-nimisen automatisoidun työkalun. Työkalun tarkoitus on heidän mukaansa löytää ja korjata virheet, kuten oletuskäyttäjätunnusten käytön turvallisuuskonfiguraatiosta. Työkalun avulla voidaan todeta, että oletuskonfiguraatiot AMP-ympäristössä eivät ole tarpeeksi hyvällä turvallisuustasolla verkkosovellusten kehitystä varten. Tästä voidaan päätellä, että käyttäjätunnukset tulisi vaihtaa mahdollisimman vahvoiksi, jotta virheellistä turvallisuuskonfiguraatiota voidaan välttää. Myös verkkosovelluksen palomuurin konfigurointia voidaan testata automatisoidusti, sillä Chowdharyn, Jhan ja Zhaon (2023) mukaan generatiivista kilpailevaa verkostoa hyödyntävällä sovelluksella verkkosovelluksen palomuuria voidaan testata generatiivisella syötteellä, saaden aikaan kattavan testauksen.

Haavoittuvaisten ja vanhentuneiden komponenttien käyttöön on hyvin yksinkertainen ratkaisu: organisaation tulee huolehtia, että käytössä olevat ohjelmistokomponentit, kuten JavaScript -kirjastot, ovat ajan tasaisia versioita (Lauinger ym., 2017). Kuitenkin Decanin ym. (2018) mukaan noin 15 prosenttia haavoittuvuuksista korjataan vasta niiden julkistuksen jälkeen, tai ei ollenkaan, joten komponenttien ajantasaisuus ei välttämättä poista haavoittuvuutta verkkosovelluksesta. Lisäksi ohjelmiston kirjastoja tulisi hankkia vain virallisilta sivustoilta turvallisten linkkien kautta, jotta vältyttäisiin myös datan eheyden ongelmilta (*A06 Vulnerable and Outdated Components - OWASP Top 10*, 2021). Organisaation tulisi aktiivisesti tutkia käytössä olevia ohjelmistokomponentteja haavoittuneiden komponenttien varalta, ja päivittää ohjelmistoaan sen mukaisesti.

Datan eheyden varmistamiseksi on kirjallisuudessa esitetty monia ratkaisuja, joista verkkosovelluksille erikoistunut ratkaisu on Verena (Karapanos, Filios, Popa, Capkun, 2016). Verena on viitekehys, joka varmistaa datan eheyden kehittäjän määrittämällä käytännöllä, ja käyttäjän selain tarkistaa, että haettu verkkosivu täyttää kehittäjän määrittelemät käytännöt. Verena siis tarkistaa, että asiakkaan hakema sivu vastaa palvelimen tarjoamaa sivua, eikä sitä ole päästy muokkaamaan. Verena toimii, vaikka koko verkkosovellus olisi hyökkääjän hallinnan alla. Datan eheyden varmistavan viitekehyksen lisäksi Alkhadra ym. (2021) esittävät useita erilaisia standardeja, joita noudattamalla organisaatio pystyy turvaamaan oman toimintansa SolarWinds -hyökkäyksen kaltaisia hyökkäyksiä vastaan. He esittävät standardeja, jotka kattavat esimerkiksi henkilöstön koulutuksen, ohjelmiston dokumentaation sekä kehitysverkon suojauksen. Heidän mukaansa näitä standardeja seuraamalla SolarWinds -hyökkäys olisi suuremmalla todennäköisyydellä osattu välttää, joten standardien, kuten W3C-standardien noudattaminen verkkosovelluksen kehityksessä on suotavaa. W3C standardit ovat verkkosovellusten parhaiden toimintatapojen standardeja, jotka käsittelevät muun muassa verkkosovellusten saatavuutta ja turvallisuutta (*Web Standards*, 2023)

4 YHTEENVETO

Tutkimuksen tarkoituksena oli selvittää, miten kriittisimpiä verkkosovelluksia koskevia haavoittuvuuksia hyväksikäytetään, sekä miten niitä tulisi ehkäistä. Tutkielma toteutettiin systemaattisena kirjallisuuskatsauksena, ja aineisto kerättiin käyttäen tietokantoja Google Scholar, IEEE Xplore, Springer, ScienceDirect sekä Scopus. Tutkimus on ajankohtainen, sillä verkkosovellukset ovat kasvattaneet suosiotaan viime vuosien aikana esimerkiksi taloudellisissa, hallinnollisissa sekä terveydenhuollon organisaatioissa (Chaudhari & Vaidya, 2014). Yksi merkittävä syy verkkosovellusten suosion kasvussa on niiden helppo implementaatio, sillä verkkosovelluksia pystytään käyttämään verkkoselaimella laitteesta riippumatta. Johtuen verkkosovelluksen julkisesta käyttöliittymärajapinnasta, verkkosovellus on kuitenkin normaalia alttiimpi kyberhyökkäykselle (Rafique ym., 2015). Tutkimuksessa havaittiin, että verkkosovellusten haavoittuvuudet ovat hyvinkin monipuolisia, johtuen verkkosovellusten monipuolisista teknologioista koostuvasta rakenteesta.

Luvussa 2 käsiteltiin verkkosovellusten yleisimpiä haavoittuvuuksia hyökkääjän näkökulmasta. Luvussa haavoittuvuuksia kuvattiin hyvinkin yksinkertaisten esimerkkien avulla, jotta haavoittuvuuksien pääpiirteet ovat mahdollisimman helposti ymmärrettävissä. Haavoittuvuuksien toteutuksia tarkastellaan lähinnä prosessitasolla, sillä monissa haavoittuvuuksista tarvittavan ymmärryksen haavoittuvuuden toiminnasta saa kuvaamalla hyökkäyksen prosessia. Luvussa 3 haavoittuvuuksia tutkittiin puolustautuvalta näkökulmalta. Luvun tarkoituksena oli antaa konkreettisia keinoja haavoittuvuuksien havainnointiin sekä ehkäisyyn. Luvussa keskityttiin haavoittuvuuksien havainnointiin sekä testaamiseen, sillä suurin osa haavoittuvuus- ja penetraatiotestauksesta suoritetaan legacy-sovelluksilla.

Tutkielmassa käsiteltiin kirjallisuutta verkkosovellusten haavoittuvuuksiin liittyen. Tutkimusta varten kerättiin aineistoksi 79 artikkelia tai konferenssijulkaisua, joista 47 julkaisua hyväksyttiin mukaan tutkimukseen. Lisäksi tutkimuksessa käytettiin hyväksi kirjoja sekä verkkosovellusten kehitykselle omistautuneita verkkosivuja. Lähteitä arvioitiin julkaisijan, viittausten määrän,

julkaisufoorumin luokituksen, mikäli sellainen julkaisulla oli, julkaisevan tahon luotettavuuden sekä artikkelin ajantasaisuuden avulla.

Tutkielmassa vastattiin kysymyksiin *miten verkkosovellusten haavoittuvuuksia hyväksikäytetään?* sekä *miten verkkosovellusten haavoittuvuuksia voidaan havaita ja ehkäistä?* Havaittiin, että kaikki haavoittuvuudet eivät lisää hyökkääjälle uusia hyökkäysvektoreita, vaan osa tutkimuksessa esitetyistä haavoittuvuuksista helpottavat hyökkääjän etenemistä verkkosovelluksessa. Osasta haavoittuvuuksia pystyttiin antamaan teknisiä esimerkkejä kirjallisuuden avulla, mutta osasta haavoittuvuuksia kirjallisuutta oli hyvinkin vaikea löytää. Käsiteltävät haavoittuvuudet onnistuttiin jakamaan neljään eri kategoriaan niiden ominaisuuksien mukaan: syötteen validointiin, istunnon hallintaan, loogiseen rakenteeseen sekä alustaan ja alustan rakenteeseen liittyviin haavoittuvuuksiin. Jokaiselle kategorialle löydettiin kyseisen kategorian haavoittuvuuksille omintakeisia piirteitä.

Suurimmassa osassa haavoittuvuuksien havainnointia suosittiin automatisoituja työkaluja, sekä tuoreimmissa tutkimuksissa hyödynnettiin koneoppimisen tekniikoita haavoittuvuuksien löytämiseksi. Haavoittuvuuksien poistamiselle annettiin kirjallisuudessa hyvinkin vähän painoarvoa, sillä kirjallisuus aiheesta on hyvinkin keskittynyt haavoittuvuuksien havainnointiin sekä testaamiseen legacy-sovelluksista. Voidaan siis päätellä, että haavoittuvuuksien paikkaamiset ovat hyvin yksilöllisiä projekteja verkkosovelluksen ominaisuuksien mukaan, eikä kaikista haavoittuvuuksista voida antaa yleispätevää ohjeistusta. Yleisesti päteviä turvallisen ohjelmoinnin perusteita kirjallisuudessa oli esitetty osalle käsitellyistä haavoittuvuuksista, ja näitä kuvattiin muun muassa prosessikaavioiden sekä algoritmien avulla. Tutkielmassa käytettyjen lähteiden osalta aiheen tutkimus oli hyvin yhteneväistä, eikä ristiriitaista tietoa esiintynyt lähteiden välillä. Lisäksi tuoreiden sekä vanhempien tutkimusten eroavaisuudet olivat lähinnä käytettyjen teknologioiden eroavaisuuksia, eikä tutkimustuloksissa ollut ristiriitaa. Haavoittuvuuksien ehkäisymetodit eivät juurikaan ole vuosien aikana muuttuneet, mutta havainnointityökaluja ja -teknologioita on selkeästi kehitetty, esimerkiksi uusimpien tutkimusten koneoppimista hyödyntävien ohjelmien muodossa.

Koska tutkimuksessa keskityttiin vain OWASP:n (2021) esittämiin haavoittuvuuksiin, jatkotutkimusta olisi syytä tehdä muistakin haavoittuvuuksista, esimerkiksi palvelunestohyökkäyksistä ja kiristysohjelmista, sillä ne jäivät kokonaan tutkimuksen ulkopuolelle. Tutkimus keskittyy tällä hetkellä hyvin paljon OWASP:n (2021) esittämiin haavoittuvuuksiin, joten tutkimusta olisi hyvä tehdä myös listauksen ulkopuolisista haavoittuvuuksista. OWASP:n (2021) esittämässä listassa esiintyvät vain kymmenen kriittisintä haavoittuvuutta, joten suuri määrä haavoittuvuuksia jäi kirjallisuuskatsauksesta pois. Lisäksi tutkimus ei käsittele haavoittuvuuksien hyväksikäytön tai puolustautumisen teknistä toteutusta, joten laajamittainen tutkimus verkkosovelluksen turvallisen kehityksen käytänteistä ja testauksesta olisi tarpeen. Lisäksi tietojärjestelmätieteen alan lähteiden puutteellisuuden takia tutkielmassa ei päästy tarkastelemaan esimerkiksi johtoportaan roolia tai kaupallisia vaikutuksia OWASP:n (2021) haavoittuvuuksien ehkäisyssä.

Tutkimuksessa käsitellyistä haavoittuvuuksista löytyi tietoa hyvin vaihtelevasti. Etenkin turvallisuuslokien ja -monitoroinnin epäonnistumisista oli hyvin vaikeata löytää ollenkaan kirjallisuutta, joten tätä aihetta tulisi tutkia huomattavasti lisää. Lisäksi muitakin vuoden 2021 OWASP Top Ten -listaukseen uutena lisättyjä haavoittuvuuksia tulisi tutkia enemmän, sillä tietoa löytyy runsaasti vain jo pidempään kriittisinä pidetyistä haavoittuvuuksista.

LÄHTEET

<img>: The Image Embed element - HTML: HyperText Markup Language | MDN. (14.9.2023). https://developer.mozilla.org/en-US/docs/Web/HTML/Element/img

A02 Cryptographic Failures - OWASP Top 10:2021. (ei pvm.). Noudettu 24. tammikuuta 2024, osoitteesta https://owasp.org/Top10/A02_2021-Cryptographic_Failures/

A06 Vulnerable and Outdated Components - OWASP Top 10:2021. (ei pvm.). Noudettu 5. joulukuuta 2023, osoitteesta https://owasp.org/Top10/A06_2021-Vulnerable_and_Outdated_Components/

A09 Security Logging and Monitoring Failures - OWASP Top 10:2021. (ei pvm.). Noudettu 24. tammikuuta 2024, osoitteesta https://owasp.org/Top10/A09_2021-Security_Logging_and_Monitoring_Failures/

Abdullah, H. M. & Zeki, A. M. (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. https://doi.org/10.1109/ACSAT.2014.22

Aljabri, M., Aldossary, M., Al-Homeed, N., Alhetelah, B., Althubiany, M., Alotaibi, O. & Alsaqer, S. (2022). Testing and Exploiting Tools to Improve OWASP Top Ten Security Vulnerabilities Detection. *2022 14th International Conference on Computational Intelligence and Communication Networks (CICN)*, 797–803. https://doi.org/10.1109/CICN56167.2022.10008360

Alkhadra, R., Abuzaid, J., AlShammari, M. & Mohammad, N. (2021). Solar Winds Hack: In-Depth Analysis and Countermeasures. *2021 12th International Conference on Computing Communication and Networking Technologies (ICCCNT)*, 1–7. https://doi.org/10.1109/ICCCNT51525.2021.9579611

Al-talak, K. & Abbass, O. (2021). Detecting Server-Side Request Forgery (SSRF) Attack by using Deep Learning Techniques. *International Journal of Advanced Computer Science and Applications*, *12*(12). https://doi.org/10.14569/IJACSA.2021.0121230

Barth, A. (2011). *HTTP State Management Mechanism* (Request for Comments RFC 6265). Internet Engineering Task Force. https://doi.org/10.17487/RFC6265

Brinhosa, R. B., Westphall, C. B. & Westphall, C. M. (2008). A Security Framework for Input Validation. *2008 Second International Conference on Emerging Security Information, Systems and Technologies*, 88–92. https://doi.org/10.1109/SECURWARE.2008.67

Brinhosa, R. B., Westphall, C. M. & Westphall, C. B. (2012). Proposal and development of the Web services input validation model. *2012 IEEE Network Operations and Management Symposium*, 643–646. https://doi.org/10.1109/NOMS.2012.6211976

Buja, G., Jalil, K. B. A., Ali, F. Bt. H. M. & Rahman, T. F. A. (2014). Detection model for SQL injection attack: An approach for preventing a web application from the SQL injection attack. *2014 IEEE Symposium on Computer Applications and*

Industrial Electronics (ISCAIE), 60–64. https://doi.org/10.1109/ISCAIE.2014.7010210

Chaudhari, G. R. & Vaidya, M. V. (2014). *A Survey on Security and Vulnerabilities of Web Application*. *5*.

Chowdhary, A., Jha, K. & Zhao, M. (2023). Generative Adversarial Network (GAN)-Based Autonomous Penetration Testing for Web Applications. *Sensors*, *23*(18). Scopus. https://doi.org/10.3390/s23188014

Conallen, J. (1999). Modeling Web application architectures with UML. *Communications of the ACM*, *42*(10), 63–70. https://doi.org/10.1145/317665.317677

Conti, M., Dragoni, N. & Lesyk, V. (2016). A Survey of Man In The Middle Attacks. *IEEE Communications Surveys & Tutorials*, *18*(3), 2027–2051. https://doi.org/10.1109/COMST.2016.2548426

CSS: Cascading Style Sheets | MDN. (22.7.2023). https://developer.mozilla.org/en-US/docs/Web/CSS

Dalai, A. K. & Jena, S. K. (2011). Evaluation of web application security risks and secure design patterns. *Proceedings of the 2011 International Conference on Communication, Computing & Security - ICCCS '11*, 565. https://doi.org/10.1145/1947940.1948057

Deepa, G. & Thilagam, P. S. (2016). Securing web applications from injection and logic vulnerabilities: Approaches and challenges. *Information and Software Technology*, *74*, 160–180. https://doi.org/10.1016/j.infsof.2016.02.005

Eshete, B., Villafiorita, A. & Weldemariam, K. (2011). Early Detection of Security Misconfiguration Vulnerabilities in Web Applications. *2011 Sixth International Conference on Availability, Reliability and Security*, 169–174. https://doi.org/10.1109/ARES.2011.31

Felmetsger, V., Cavedon, L., Kruegel, C. & Vigna, G. (ei pvm.). *Toward Automated Detection of Logic Vulnerabilities in Web Applications*.

Fielding, R. T., Nottingham, M. & Reschke, J. (2022). *HTTP Semantics* (Request for Comments RFC 9110). Internet Engineering Task Force. https://doi.org/10.17487/RFC9110

Gautam, T. & Jain, A. (2015). Analysis of brute force attack using TG — Dataset. *2015 SAI Intelligent Systems Conference (IntelliSys)*, 984–988. https://doi.org/10.1109/IntelliSys.2015.7361263

HTML: HyperText Markup Language | MDN. (17.7.2023). https://developer.mozilla.org/en-US/docs/Web/HTML

Huluka, D. & Popov, O. (2012). Root cause analysis of session management and broken authentication vulnerabilities. *World Congress on Internet Security (WorldCIS-2012)*, 82–86. https://ieeexplore.ieee.org/abstract/document/6280203

JavaScript | MDN. (25.9.2023). https://developer.mozilla.org/en-US/docs/Web/JavaScript

Johns, M., Braun, B., Schrank, M. & Posegga, J. (2011). Reliable protection against session fixation attacks. *Proceedings of the 2011 ACM Symposium on Applied Computing*, 1531–1537. https://doi.org/10.1145/1982185.1982511

Jovanovic, N., Kirda, E. & Kruegel, C. (2006). Preventing Cross Site Request Forgery Attacks. *2006 Securecomm and Workshops*, 1–10. https://doi.org/10.1109/SECCOMW.2006.359531

Karapanos, N., Filios, A., Popa, R. A. & Capkun, S. (2016). Verena: End-to-End Integrity Protection for Web Applications. *2016 IEEE Symposium on Security and Privacy (SP)*, 895–913. https://doi.org/10.1109/SP.2016.58

Kieyzun, A., Guo, P. J., Jayaraman, K. & Ernst, M. D. (2009). Automatic creation of SQL Injection and cross-site scripting attacks. *2009 IEEE 31st International Conference on Software Engineering*, 199–209. https://doi.org/10.1109/ICSE.2009.5070521

Kortesoja, M. (2022). Tapaus Vastaamo: Symptomaattinen luenta potilastietosuojan murtumisen yhteiskunnallisista syistä ja seurauksista. *Tutkimus & kritiikki*, *2*(1), 9–32. https://doi.org/10.55294/tk.113346

Lauinger, T., Chaabane, A., Arshad, S., Robertson, W., Wilson, C. & Kirda, E. (2017). Thou Shalt Not Depend on Me: Analysing the Use of Outdated JavaScript Libraries on the Web. *Proceedings 2017 Network and Distributed System Security Symposium*. https://doi.org/10.14722/ndss.2017.23414

Lazar, D., Chen, H., Wang, X. & Zeldovich, N. (ei pvm.). *Why does cryptographic software fail? A case study and open problems*.

Lazar, D., Chen, H., Wang, X. & Zeldovich, N. (2014). Why does cryptographic software fail?: a case study and open problems. *Proceedings of 5th Asia-Pacific Workshop on Systems*, 1–7. https://doi.org/10.1145/2637166.2637237

Lei, L., Chen, M., He, C. & Li, D. (2020). XSS Detection Technology Based on LSTM-Attention. *2020 5th International Conference on Control, Robotics and Cybernetics (CRC)*, 175–180. https://doi.org/10.1109/CRC51253.2020.9253484

Li, X. & Xue, Y. (ei pvm.). *A Survey on Web Application Security*.

Li, X. & Xue, Y. (2014). A survey on server-side approaches to securing web applications. *ACM Computing Surveys*, *46*(4), 1–29. https://doi.org/10.1145/2541315

Liu, M., Zhang, B., Chen, W. & Zhang, X. (2019). A Survey of Exploitation and Detection Methods of XSS Vulnerabilities. *IEEE Access*, *7*, 182004–182016. https://doi.org/10.1109/ACCESS.2019.2960449

Loureiro, S. (2021). Security misconfigurations and how to prevent them. *Network Security*, *2021*(5), 13–16. Scopus. https://doi.org/10.1016/S1353-4858(21)00053-2

Lukanta, R., Asnar, Y. & Kistijantoro, A. I. (2014). A vulnerability scanning tool for session management vulnerabilities. *2014 International Conference on Data and Software Engineering (ICODSE)*, 1–6. https://doi.org/10.1109/ICODSE.2014.7062682

Luo, H. (2019). SSRF Vulnerability Attack and Prevention Based on PHP. *2019 International Conference on Communications, Information System and Computer Engineering (CISCE)*, 469–472. https://doi.org/10.1109/CISCE.2019.00109

Marashdih, A. W. & Zaaba, Z. F. (2017). Cross Site Scripting: Removing Approaches in Web Application. *Procedia Computer Science*, *124*, 647–655. https://doi.org/10.1016/j.procs.2017.12.201

OWASP Top Ten | OWASP Foundation. (ei pvm.). Noudettu 25. syyskuuta 2023, osoitteesta https://owasp.org/www-project-top-ten/

Pathak, N., Singh, B. M. & Sharma, G. (2017). UML 2.0 based framework for the development of secure web application. *International Journal of Information Technology*, *9*(1), 101–109. https://doi.org/10.1007/s41870-017-0001-3

Pellegrino, G., Catakoglu, O., Balzarotti, D. & Rossow, C. (2016). Uses and Abuses of Server-Side Requests. Teoksessa F. Monrose, M. Dacier, G. Blanc & J. Garcia-Alfaro (toim.), *Research in Attacks, Intrusions, and Defenses* (s. 393–414). Springer International Publishing. https://doi.org/10.1007/978-3-319-45719-2_18

Rafique, S., Humayun, M., Hamid, B., Abbas, A., Akhtar, M. & Iqbal, K. (2015). Web application security vulnerabilities detection approaches: A systematic mapping study. *2015 IEEE/ACIS 16th International Conference on Software Engineering, Artificial Intelligence, Networking and Parallel/Distributed Computing (SNPD)*, 1–6. https://doi.org/10.1109/SNPD.2015.7176244

Rodríguez, G. E., Torres, J. G., Flores, P. & Benavides, D. E. (2020). Cross-site scripting (XSS) attacks and mitigation: A survey. *Computer Networks*, *166*, 106960. https://doi.org/10.1016/j.comnet.2019.106960

Salminen, A. (2023). *Mikä kirjallisuuskatsaus? : Johdatus kirjallisuuskatsauksen tyyppeihin ja joihinkin hallintotieteellisiin sovelluksiin*. Vaasan yliopisto. https://osuva.uwasa.fi/handle/10024/15470

Scott, D. & Sharp, R. (2002). Developing secure Web applications. *IEEE Internet Computing*, *6*(6), 38–45. https://doi.org/10.1109/MIC.2002.1067735

Shahriar, H. & Zulkernine, M. (2010). Client-Side Detection of Cross-Site Request Forgery Attacks. *2010 IEEE 21st International Symposium on Software Reliability Engineering*, 358–367. https://doi.org/10.1109/ISSRE.2010.12

Shar, L. K. & Tan, H. B. K. (2012a). Defending against Cross-Site Scripting Attacks. *Computer*, *45*(3), 55–62. https://doi.org/10.1109/MC.2011.261

Shar, L. K. & Tan, H. B. K. (2012b). Automated removal of cross site scripting vulnerabilities in web applications. *Information and Software Technology*, *54*(5), 467–478. https://doi.org/10.1016/j.infsof.2011.12.006

Shar, L. K. & Tan, H. B. K. (2012c). Defeating SQL Injection. *Computer*, *46*(3), 69–77. https://doi.org/10.1109/MC.2012.283

Shklar, L. & Rosen, R. (2003). *Web application architecture: principles, protocols and practices*. Wiley.

Singh, N. K., Gupta, P., Singh, V. & Ranjan, R. (2021). Attacks on Vulnerable Web Applications. *2021 International Conference on Intelligent Technologies (CONIT)*, 1–5. https://doi.org/10.1109/CONIT51480.2021.9498396

Sivathanu, G., Wright, C. P. & Zadok, E. (2005). Ensuring data integrity in storage: techniques and applications. *Proceedings of the 2005 ACM Workshop on Storage Security and Survivability*, 26–36. https://doi.org/10.1145/1103780.1103784

Stuttard, D. & Pinto, M. (2011). *The Web Application Hacker's Handbook: Finding and Exploiting Security Flaws*. John Wiley & Sons.

Takamatsu, Y., Kosuga, Y. & Kono, K. (2012). Automated detection of session management vulnerabilities in web applications. *2012 Tenth Annual International Conference on Privacy, Security and Trust*, 112–119. https://doi.org/10.1109/PST.2012.6297927

Vlsaggio, C. A. & Blasio, L. C. (2010). Session management vulnerabilities in today's web. *IEEE Security & Privacy Magazine*, *8*(5), 48–56. https://doi.org/10.1109/MSP.2010.114