Roni Akiola

Web-sovellusten tietoturva – OWASP Top 10 -uhkakuvat ja haavoittuvuuksien testaaminen

Metropolia Ammattikorkeakoulu

Insinööri (AMK)

Tieto- ja viestintätekniikka

Insinöörityö

18.4.2024

Tiivistelmä

Tekijä:	Roni Akiola
Otsikko:	Web-sovellusten tietoturva – OWASP Top 10 -uhkakuvat ja haavoittuvuuksien testaaminen
Sivumäärä:	50 sivua
Aika:	18.4.2024
Tutkinto:	Insinööri (AMK)
Tutkinto-ohjelma:	Tieto- ja viestintätekniikka
Ammatillinen pääaine:	Mediatekniikka
Ohjaajat:	Lehtori Matti Peltoniemi

Insinöörityössä perehdyttiin web-sovellusten tietoturvaan havainnoimalla tunnettuja riskitekijöitä, niiden aiheuttajia, vaikutuksia ja torjuntatoimenpiteitä. Työn viitekehyksenä toimi Open Worldwide Application Security Project -säätiön laatima OWASP Top 10 -lista.

Tavoitteena työlle oli koostaa kattava dokumentaatio web-sovellusten tietoturvauhkien korkean riskin kategorioista ja havainnollistaa näiden vahingollisia vaikutuksia web-sovellukseen ja sen käyttäjiin kohdentamalla haavoittuvuuksiin hyökkäyksiä ja testejä.

Insinöörityö toteutettiin kokoamalla tietoturva-aiheista aineistoa yhtenäiseksi kokonaisuudeksi sekä pystyttämällä virtuaalitietokoneelle testausympäristö, jossa tietoturvahaavoittuvuuksien testaaminen voitiin toteuttaa turvallisesti. Virtuaalitietokoneen käyttöjärjestelmäksi valittiin Kali Linux ja testattavaksi kohteeksi OWASP Juice Shop -alusta, minkä lisäksi avoimen lähdekoodin työkaluja hyödynnettiin testauksen aikana. Työn alussa laadittiin ohjeet testausympäristön alustamisesta, minkä jälkeen hyökkäyskohteeseen suoritettiin tiedustelu- ja kartoitustoimenpiteitä. Edeltävän vaiheen pohjalta löytyneisiin potentiaalisiin haavoittuvuuksiin tutustuttiin tarkemmin tukeutuen kerättyihin tietoihin, ja lopuksi suoritettiin hyökkäysvaihe järjestelmällisesti edeten.

Tuloksena muodostui laadullisten menetelmien kautta koostettu dokumentaatio, joka kokonaisuutena ilmentää aiheen ajankohtaisuutta ja tärkeyttä web-sovelluskehityksessä. Tätä työtä voidaan käyttää tutkittujen tietoturvariskien alkeiden oppimiseen, perehdytykseen ja käytännön toteutukseen.

Avainsanat: web-sovelluskehitys, tietoturva, OWASP, penetraatiotestaus

---

Abstract

Author:	Roni Akiola
Title:	Web Application Security – OWASP Top 10 Threats And Testing Of Vulnerabilities
Number of Pages:	50 pages
Date:	18 April 2024
Degree:	Bachelor of Engineering
Degree Programme:	Information and Communication Technology
Professional Major:	Media Technology
Supervisors:	Matti Peltoniemi, Senior Lecturer

The goal of this final year project was to study web application security by observing known risk factors, their causes, effects, and mitigation measures. The OWASP Top 10 list of the Open Worldwide Application Security Project foundation was used as a framework for conducting this study.

The goal was to compile comprehensive documentation on high-risk categories of web application security threats and illustrate their damaging effects on web applications and their users by directing attacks and tests on vulnerabilities.

In practice, a testing environment was set up on a virtual machine, where testing of security vulnerabilities could be carried out safely. Kali Linux was chosen as the operating system for the virtual machine, and OWASP Juice Shop was selected as the target for testing. In addition, multiple open-source tools were utilized during the testing. At the beginning of the work, instructions on how to set up the testing environment were composed, followed by reconnaissance and mapping procedures on the target application. Subsequently, the potential vulnerabilities that were identified in the previous phase were examined in more detail, and with the help of the information presented in theoretical chapters of the thesis, the attack phase was systematically carried out.

As a result, documentation that reflects the relevance and importance of security in web application development in general was compiled through qualitative methods. This work can be used for learning the basics and how to execute the researched security risks in practice.

Keywords: web application development, information security, OWASP, penetration testing

Sisällys

1 Johdanto

Tässä insinöörityössä perehdytään web-sovellusten tietoturvaan käyttäen viitekehyksenä Open Worldwide Application Security Project -säätiön (OWASP) laatimaa dokumenttia kymmenestä web-sovellusten yleisimmästä tietoturvauhkasta. Viimeisin julkaisu on laadittu vuonna 2021, ja se toimii myös tämän insinöörityön viitekehyksenä.

Tavoitteena tällä insinöörityöllä on syventyä web-sovellusten tietoturvaan sellaisella tasolla, että työn voidaan katsoa hyödyttävän laatijaa tulevaisuuden työelämässä, sekä perehdyttää lukija aiheen perusteisiin tutkimalla kriittisimmät uhkakuvat modernissa web-sovelluskehityksessä riskeineen ja aiheuttamine vahinkoineen. Työn tarkoituksena on toimia ponnahduslautana tiedon lisäämiseen ja ymmärtämiseen web-sovellusten tietoturvan kontekstissa, minkä pohjalta on miellyttävää etsiä lisää tietoa ja perehtyä aiheeseen yhä syvällisemmin.

Insinöörityön ensimmäinen luku sisältää yleistietoa web-sovellusten tietoturvasta, mikä toimii johdantona tuleville kappaleille. Toisessa luvussa perehdytään OWASP Top 10 -listaan, jossa tutkitaan listan sisältämät tietoturvaongelmat, näiden aiheuttajat, riskit ja mahdolliset torjuntatoimenpiteet uhkakohtaisesti. Kolmannessa osassa perehdytään lyhyesti penetraatiotestauksen teoriaan ja tarkoitukseen sekä pystytetään insinöörityössä käytetty testiympäristö. Viimeisessä luvussa tutkitaan ja testataan OWASP Juice Shop -alustaa mahdollisilta haavoittuvuuksilta, joihin myös kohdennetaan hyökkäyksiä, ja tuodaan esiin löydetyt havainnot.

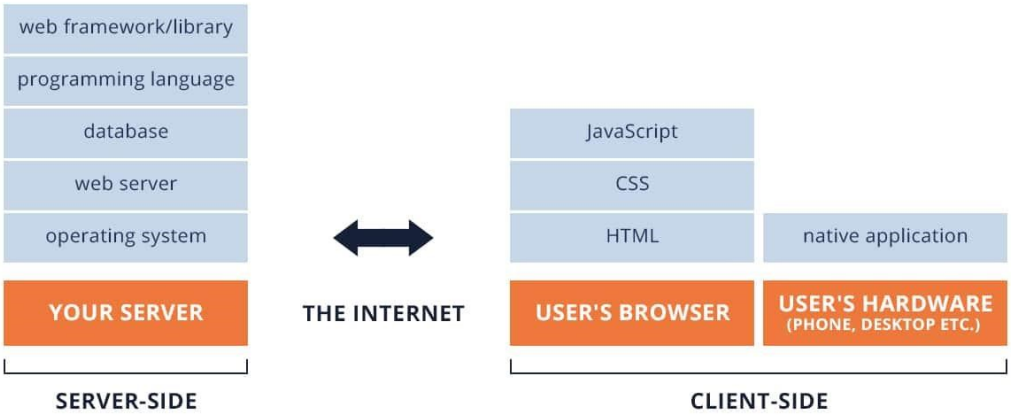
Tämä insinöörityö on rajattu sisältämään OWASP Top 10 -listan sisältämät tietoturvauhkat, eikä työ perehdy syvällisesti listan ulkopuolella oleviin uhkakuviin, kuten arkaluonteisen tiedon säilyttämiseen ja sen lakisääteisiin vaatimuksiin, tai tietokonekomponenttien haavoittuvuuksiin, vaikka edellä mainittuja asioita paikoin työssä sivutaankin.

2 Web-sovellusten tietoturva

2.1 Yleistä web-sovelluksista ja tietoturvasta

Web-sovellusten eli verkkosovellusten tietoturva on laaja käsite, mutta lyhykäisyydessään sillä tarkoitetaan verkkosivujen, -sovellusten ja -palveluiden sekä näihin liittyvien järjestelmien ja datan suojaamista tietoturvariskeiltä. Palveluita voivat esimerkiksi olla erilaiset julkiset tai yksityiset ohjelmointirajapinnat. Web-sovellusten tietoturva käsittää kaikki sovelluksen infrastruktuurin sisältämät tasot eli vähintään käyttöjärjestelmän, ohjelmointikielen, tietokannan ja palvelimen. Myös käyttäjän selaimessa suoritettava koodi on huomioitava tietoturvassa. [1.]

Kuvassa 1 on esitetty web-sovelluksen tasot jaettuna palvelin- ja käyttäjäpuoleen, jotka kommunikoivat HTTP/HTTPS-protokollan avulla internetin välityksellä. Kuvassa vasemmalla on palvelimen eri tasot ylhäältä alas järjestyksessä: ohjelmistokehys/-kirjasto, ohjelmointikieli, tietokanta, palvelinohjelmisto ja käyttöjärjestelmä. Vastaavasti oikealla puolella on käyttäjäpäätteen käyttämät teknologiat selaimessa: JavaScript, CSS, HTML. Aivan oikealla käyttäjän laitteisto, joka on web-sovellusta käytettäessä yleisimmin tietokone tai älypuhelin. Jokainen näistä tasoista voi olla alttiina erilaisille tietoturvauhkille yhdessä tai erikseen.



Kuva 1. Kuvaus web-sovellusten eri tasoista jaettuna palvelimeen vasemmalla ja käyttäjäpäätteeseen oikealla [2].

Eri tasojen yhteyden ja tärkeyden havainnollistamiseksi esitetään skenaario, jossa hyökkääjä pyrkii käyttäjäpäätteen selaimesta tekemään injektiohyökkäyksen palvelimen tietokantaan. Hyökkääjä on selvittänyt sovelluksen käyttämän tietokannan ja teknologiat sekä selaimessa suoritettavan ohjelmistokehyksen ja kielen. Hän toteuttaa SQL-injektion suoraan kirjautumissivulla, mutta syötekentät on puhdistettu jo selaimen ohjelmassa. Hyökkääjä kuitenkin pystyy kiertämään selaimen käyttäjäpäätteen heikot varmennukset käyttäjäsyötteille katkaisemalla selaimesta lähetetyn HTTP-pyynnön palvelimelle käyttäen proxy-työkalua. Nyt hyökkääjä pystyy manuaalisesti manipuloimaan keskeytetyn HTTP-pyynnön parametrien arvoja ja muokkaamaan niistä haluamansa, minkä jälkeen hän lähettää pyynnön eteenpäin palvelimelle. Jos palvelinpuolen ohjelmassa ei ole toteutettu syötteiden validointia ja puhdistusta tai se on toteutettu väärin, on hyvin mahdollista, että hyökkääjän kirjoittama käsky ajetaan tietokantaan.

Edellä kuvattu esimerkki on hyvin yksinkertainen, mutta demonstroi, kuinka web-sovelluksen eri tasot ovat yhteydessä toisiinsa ja kuinka yksi haavoittuvuus voi aiheuttaa muuten turvalliseen sovellukseen merkittävän tietoturva-aukon.

Web-sovellusten turvallisuuden varmistamisen tarkoituksena on pitää sovelluksen toiminta luotettavana ja kitkattomana sekä suojella sovellusta ja sen käyttäjiä siihen kohdistuvilta haitallisilta hyökkäyksiltä, kuten tietovarkauksilta ja

vandalismilta [3]. Koska web-sovellukset sisältävät monta eri tasoa, on tietoturvallisen sovelluksen kehittäminen vaativaa työtä, joka edellyttää sovelluskehittäjältä ammattitaitoa ja syvällistä perehtymistä sovellustasojen toimintaan sekä niihin kohdistuviin uhkiin.

Web-sovellusten tietoturvaa tarkastellessa ei ole yhtä oikeaa ratkaisua, joka ratkaisisi kaikki ongelmat ja heikkoudet kerralla, vaan tietoturvaa tulee aina tarkastella sovellustyypin mukaisesti [4]. Vaadittavan tietoturvan taso vaihtelee paljon sovelluksen laajuuden, käsiteltävän tiedon ja näihin liittyvien riskien perusteella. Esimerkiksi staattiset verkkosivut ovat luonnostaan tietoturvan suhteen melko vahvassa asemassa, jos ne eivät käytä tietokantaa tai käsittele ollenkaan käyttäjätietoja ja -syötteitä. Vaikka tämän tyyppiseen staattiseen verkkosivuun kohdistuisikin esimerkiksi injektiohyökkäys, on todennäköisyys vahingoittavalle tietomurrolle matala, parhaimmillaan olematon.

Aikaisemmin internet-sivujen yleistyessä ja suosion kasvaessa hakkerit kohdensivat hyökkäyksiään juuri staattisille verkkosivuille erilaisten motiivien perusteella. Tarkoituksena saattoi olla esimerkiksi huomion kerääminen kanssahakkerien keskuudessa, ja toiminta olikin useasti niin sanottua verkkosivuvandalismia tai -turmelemista. Ero nykypäivään on suuri, sillä nyt hyökkääjiä yleensä motivoi taloudellisen hyödyn tavoittelu laittomin keinoin. [5.]

Dynaamisten web-sovellusten yleistyessä tietoturvauhkat ovat moninaistuneet, ja niiden aiheuttamat haitat ovat kasvaneet [5]. Hyökkäyksistä johtuvat haitat riippuvat paljolti hyökkäyksen vakavuudesta, mutta pahimmillaan tappiot websovelluksen omistajalle tai sen asiakkaille voivat olla rahallisesti tai maineellisesti mittavat. Kun yritysten liiketoiminta on yhä enemmän verkossa, myös henkilötiedot ovat alttiina hyökkäyksille – näin kävi Suomessa psykoterapiakeskus Vastaamon 33 000 asiakkaille, kun heikosti salasanasuojattuun palvelimeen murtauduttiin, ja hyökkääjä vaati lunnaita anastettuja henkilötietoja vastaan [6].

Teknologian kehittyessä jatkuvasti voisi maallikko olettaa web-sovellusten tietoturvaratkaisujen saavuttavan pisteen, jossa hyökkäykset voidaan pysyvästi

torjua. Valitettavasti näin ei ole kuitenkaan käynyt, vaan alati kasvavan liiketoiminnan osuus verkossa, ja sitä myöten syntyvien teknisten ratkaisujen ohessa myös hyökkäysten määrä on kasvanut ja laatu moninaistunut. Voidaankin sanoa, että tietoturva alana on vielä kaukana täydellisyydestä, mikä on toisaalta hyvin luonnollista sen takia, että hyökkäävät tahot ja heitä vastaan ratkaisuja etsivät ohjelmistokehittäjät ovat keskenään ikuisessa kilpajuoksussa.

2.2 Tiedustelu ja kartoitus

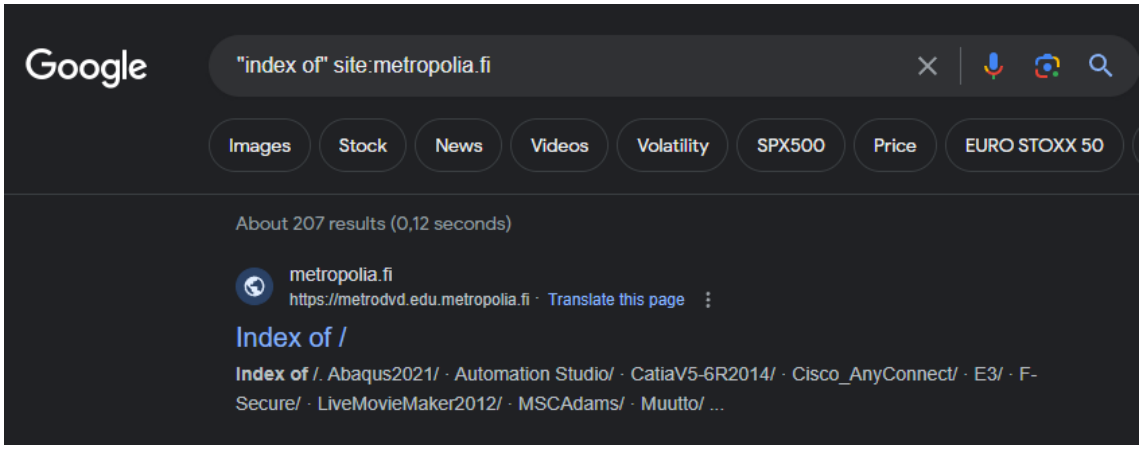
Web-sovelluksen hyödyntäminen hyökkäykseen vaatii hyökkäävältä taholta laajaa osaamista ja työkaluja, kuten ymmärrystä verkkoprotokollista, yleisesti käytetyistä komponenteista ja niiden heikkouksista, sovellukseen rakennetuista toiminnallisuuksista sekä tietenkin siitä, millä ohjelmistokehyksillä ja teknologioilla verkkosivu on rakennettu. Hyökkääjällä on oltava näistä syvä ymmärrys, jotta kohteena olevaan sovellukseen käytettävät hyökkäysmetodit voidaan rajoittaa oikein ja kohdistaa tehokkaasti. [3; 7.]

Ennen varsinaisen hyökkäyksen aloittamista on yleistä, että hyökkääjä suorittaa tiedustelu- ja kartoitusvaiheen, jossa hän pyrkii saamaan lisätietoja edellä mainituista asioista ja syventää sitä kautta ymmärrystään sovelluksen toiminnasta ja haavoittuvuuksista. Ilman tätä vaihetta olisi hyvin vaikeaa kohdentaa hyökkäyksiä tuloksekkaasti. Tiedusteluvaiheella pyritään esimerkiksi kartoittamaan sovelluksesta toiminnallisuuksia, jotka eivät ole tavalliselle käyttäjälle saavutettavissa, kuten esimerkiksi ylläpitäjälle tarkoitetut toiminnot ohjelmointirajapinnassa. [7.]

Tiedustelu- ja kartoitusvaihe voidaan suorittaa suorin tai epäsuorin keinoin. Epäsuoraa tiedustelua suositaan etenkin hyökkäyksen alkuvaiheessa, koska sillä pyritään välttämään suoraa kontaktia kohteena olevaan tahoon. Epäsuoraan tiedusteluun hyökkääjä käyttää hyödykseen julkisesti saatavilla olevia resursseja, verkkosivuja ja työkaluja. Yksinkertaisimmillaan erilaisia operaattoreita käyttämällä ja taitavasti muotoilulla Google-haulla saattaa pystyä kartoittamaan joidenkin verkkosivujen mahdollisia haavoittuvuuksia ja etsiä pääsyä

resursseihin, jotka eivät ole itse verkkosivulla näkyvissä. Tätä kutsutaan Google-hakkeroinniksi. [7.]

Kuvassa 2 on esimerkki Google-haun tuloksesta Metropolian verkkosivulle. Linkkiä seuraamalla päästään käsiksi sivun kansiorakenteeseen, jota tutkimalla hyökkääjä saattaa löytää mahdollisia haavoittuvuuksia.



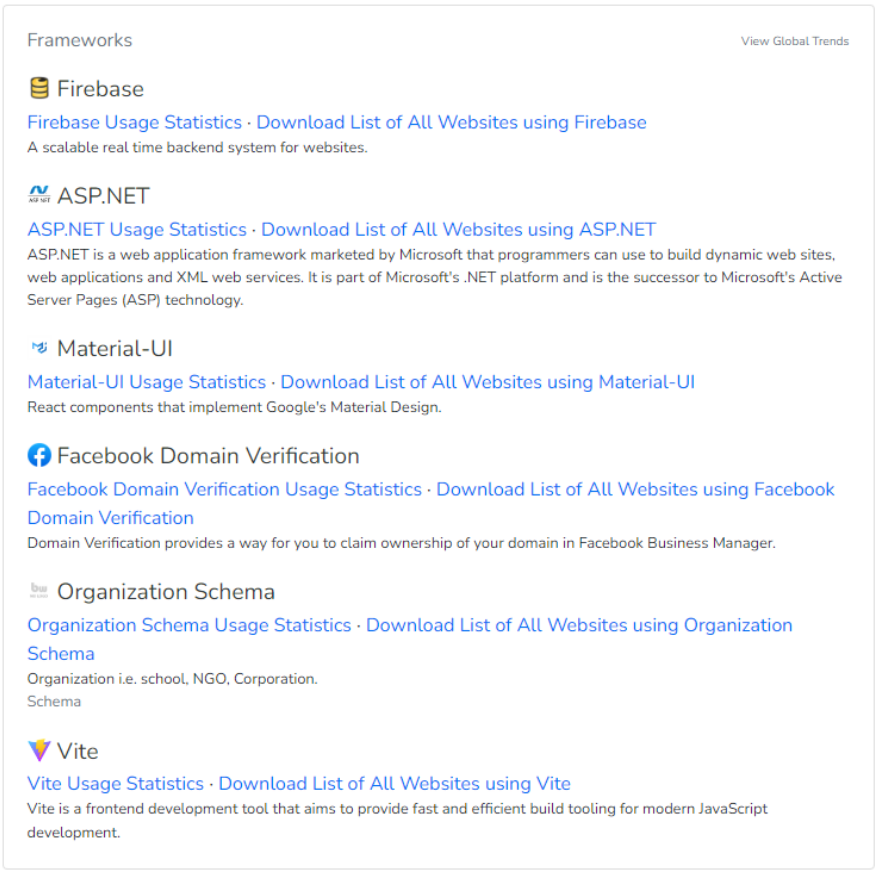
Kuva 2. Kuvakaappaus selainhausta erikoisoperaattoreilla.

Erikoisoperaattoreilla suoritettavissa Google-hauissa on huomioitava, että Google hyvin nopeasti estää toistuvat peräkkäiset haut CAPTCHA-sivuilla, joka pyytää suorittamaan satunnaisgeneroidun tehtävän, esimerkiksi valitsemaan kaikki kuvat, joissa on kissoja. Jos tästä huolimatta kyselyt jatkuvat, Google saattaa estää IP-osoitteen, josta pyynnöt tulevat. Tämän takia automatisoitu Google-hakkerointi ei ole kauaskantoisesti kannattavaa. Lisäksi on huomionarvoista, että vain tekstipohjaiset haut Googlen välimuistiin varmistavat anonymiteetin. Jos sivulle mennään, jää siitä jälki. Tämä on kuitenkin kierrettävissä esimerkiksi proxy-palvelulla. [7.]

Toinen epäsuora tiedustelutapa on esimerkiksi julkisesti verkosta löytyvä palvelu Built With, joka analysoi verkkosivun rakentamiseen käytetyt teknologiat, kuten ohjelmointikielen, komponentit, ohjelmistokehykset, käytetyt rajapinnat, palvelut ja niin edespäin. Built With analysoi sivustoja keräämällä niiden antamia niin sanottuja signaaleja [8]. Näitä signaaleja voi esimerkiksi olla

selaimessa suoritettava käyttäjäpäätteen lähdekoodi, joka saattaa sisältää koodia, josta ilmenee käytetyt teknologiat selainpuolella. Lisäksi palvelinpuolen teknologioita voidaan etsiä palvelimelle tehtyjen pyyntöjen ja vastausten avulla, joiden HTTP-otsikkotiedoista voi löytyä palvelimella käytettävien teknologioiden tyypillisiä ominaisuuksia. Käytännössä tämän työn voisi tehdä myös manuaalisesti itse, mutta Built With -palvelua käyttämällä nopeutetaan ja tehostetaan tiedusteluprosessia olematta suorassa kontaktissa tiedusteltavaan web-sovellukseen.

Kuvassa 3 on suoritettu analyysi Kotipizzan verkkosivujen käyttämistä teknologioista Built With -palvelulla ja rajattu sen käyttämät viitekehykset. Kuvasta voidaan nähdä, että selainpuolella on käytetty esimerkiksi Material-UI -komponenttikirjastoa Reactille. Jos Material-UI sisältäisi jonkin haavoittuvuuden, josta hyökkääjä on tietoinen, voisi hän käyttää tätä tietoa hyväkseen varsinaista hyökkäystä suorittaessaan.



Kuva 3. Kuvakaappaus Built With -palvelulla suoritetusta analyysistä Kotipizzan verkkosivuista ja sen ohjelmistokehyksistä.

Suorat tiedustelu- ja kartoitusmetodit ovat suorassa kontaktissa hyökättävään tahoon ja niillä pyritään löytämään syvällistä tietoa web-sovelluksen toiminnallisuudesta, kuten aliverkkotunnuksia, ohjelmointirajapintojen päätepisteitä, heikkoja kohtia ja kansiorakennetta.

Yksinkertaisimmillaan vierailu verkkosivulla ja kirjaamalla ylös sen tekemiä verkkopyyntöjä auttaa ymmärtämään, miten sovellus toimii ja kommunikoi. On kuitenkin huomattava, että useimpien web-sovellusten käyttöliittymät ja toiminnallisuudet ovat harvoin täysin saatavilla tavalliselle käyttäjälle, vaan ne sisältävät usein paljon ominaisuuksia, jotka ovat piilotettuja ja käytettävissä vain etuoikeutetuille käyttäjille, kuten ylläpitäjälle. [5.]

Edellisessä kappaleessa mainittu huomioon ottaen, on hyökkääjälle erittäin arvokasta selvittää piilossa olevia ominaisuuksia, kuten ohjelmointirajapinnan päätepisteitä, joihin vain etuoikeutetuilla käyttäjillä on pääsy. Pelkästään tieto päätepisteiden olemassaolosta on tärkeää web-sovelluksen toiminnallisuuksien kartoittamisen kannalta, mutta pahimmillaan hyökkääjä saattaa jopa löytää tietoturva-aukon, jota hyödyntämällä hän pääsee käsiksi päätepisteisiin, vaikkei suoraan olisi oikeutettu käyttämään niitä [5]. Päätepisteiden ja toiminnallisuuksien kartoittamisessa voidaan hyödyntää Burp Suite -ohjelmaa, jonka avulla kartoitus voidaan automatisoida, vaikka se harvoin tuottaa yhtä laadukkaita tuloksia kuin manuaalinen kartoitus [7].

Burp Suiten lisäksi suoria tiedustelu- ja kartoitusmetodeja on useita, ja niiden käyttö riippuu selvitettävän informaation luonteesta. Esimerkiksi jos Google-hakkeroinnilla on löydetty alustavasti mielenkiintoinen kansiorakenne, voidaan lisätietoa löytää hyödyntämällä DirBuster-ohjelmaa. DirBuster skannaa web-sovelluksen kansio- ja tiedostorakenteen käyttämällä ennalta määriteltyä tai kustomoitua sanalistaa. Tavoitteena on löytää yleisesti verkkosivuilta löytyviä kansioita ja tiedostoja, esimerkiksi ylläpitäjäkansioita, väliaikaisia tiedostoja, varmuuskopioita tai konfiguraatiotiedostoja. [7.] Tämänlaista tiedustelu- ja

kartoitustoimintaa kutsutaan brute force -hyökkäykseksi, eli väsytyshyökkäykseksi, ja se tulisi jättää viimeiseksi tiedusteluvaiheessa, sillä väsytyshyökkäykset ovat helposti havaittavissa ja hidastettavissa, jopa estettävissä kokonaan [5].

3 OWASP Top 10

Open Worldwide Application Security Project (OWASP) on vuoden 2001 loppupuolella perustettu voittoa tavoittelematon ja kaikille avoin säätiö, joka työskentelee edistääkseen ohjelmistojen tietoturvaa ja jonka toiminnan tavoitteina on tuoda sovelluskehittäjien ja organisaatioiden tietoisuuteen tietoturvaseikkoja sekä ohjeistaa turvallisempien ohjelmistojen kehitystyössä. Näiden tavoitteiden tueksi OWASP on julkaissut useita avoimen lähdekoodin projekteja, dokumentaatioita ja työkaluja, jotka ovat ilmaiseksi kenen tahansa käytettävissä. Tunnetuin projekteista on OWASP Top 10 -lista, johon on kerätty ajankohtaisimmat web-sovelluksiin kohdistuvat tietoturvauhkat. [9; 10]

OWASP Top 10 -lista on avoin projekti, jonka päälaatijoina ovat toimineet Andrew van der Stock, Brian Glas, Neil Smithline ja Torsten Gigler, mutta on kuitenkin huomionarvoista, että projektin toteutukseen on mittavasti vaikuttaneet useat vapaaehtoiset organisaatiot ja yksityishenkilöt [11]. Lista on tarkoitettu matalan kynnyksen viitekehykseksi kohti turvallisempaa ohjelmistokehitystyötä tarjoamalla informaatiota ajankohtaisista web-sovellusten tietoturvauhkista ja niiden torjuntatoimenpiteistä [12]. Listaa ei ole kuitenkaan tarkoitettu kaiken kattavaksi standardiksi, jonka pohjalta jonkin tietyn ohjelmiston tietoturva voidaan täysin varmistaa, vaan dokumentiksi, joka toimii lähtöpisteenä organisaatioille ja web-sovelluskehittäjille tietoturvaongelmien huomioimiseksi kehitystyössä [13]. Listaa päivitetään aina muutaman vuoden välein, jotta sen sisältö pysyisi mahdollisimman relevanttina jatkuvasti muuttuvassa toimintaympäristössä. Viimeisin versio on julkaistu 24.9.2021, ja siihen myös tässä insinöörityössä viitataan [11].

OWASP Top 10 -listan kategoriat on johdettu web-sovelluksista löytyneiden haavoittuvuuksien pohjalta kerätystä datasta, josta kahdeksan eniten esiintyvää

tietoturvakategoriaa on sisällytetty listalle. Lisäksi tietoturvayhteisölle teetetyn kyselyn perusteella kaksi merkittävimpänä pidettyä kategoriaa lisätään listalle. Nämä kaksi kategoriaa valitaan kuitenkin sellaisin perustein, että ne eivät saa olla samoja kategorioita kuin jo listalla entuudestaan esiintyvät, datan perusteella valitut kategoriat. Metodia on perusteltu datan keruun ja koostamisen hitaudella, minkä tueksi tarvitaan alan ammattilaisten ja yhteisön mielipide ajankohtaisista tietoturvauhkista, jotka eivät välttämättä vielä näy kerätyssä datassa. Data itsessään on kerätty eri organisaatioilta, joiden toiminta keskittyy haavoittuvuuksien ja uhkien löytämiseen sekä testaamiseen. [14.] Dataa vuoden 2021 listalle kertyi yhteensä yli 500 000 web-sovelluksesta [15].

Vaikka OWASP on yleisellä tasolla luotettu säätiö, voidaan kritiikkiä esittää siitä, että raakadata testatuista sovelluksista ei ole avoimesti saatavilla ja että konteksti ja selitteet kategoriakohtaisessa datassa ovat paikoin puutteellisia ja jättävät auki kysymyksiä siitä, miten jotkin tulokset on mitattu ja millaisin perustein.

Kuvassa 4 on esitelty OWASP Top 10 -listan kategoriat ja niihin liittyvää dataa. CWE eli Common Weakness Enumeration tarkoittaa tietyn tyyppistä heikkoutta ohjelmistossa, mikä voi johtaa haavoittuvuuteen. CVE eli Common Vulnerabilities and Exposures taas tarkoittaa listaa tunnetuista haavoittuvuuksista. Kartoitetut CWE:t kuvastavat, kuinka monta heikkoustyyppiä on sisällytetty kyseiseen kategoriaan. Esiintymistiheys kertoo prosentuaalisen määrän sovelluksista, joista löytyi kategorian sisältämiä heikkouksia. Hyödynnettävyys ja vaikutus on normalisoitu kymmenpisteasteikoksi CVSSv2- ja CVSSv3-pistejärjestelmien kesken. Kuvassa ilmoitetut pisteet ovat kategoriakohtaisesti kaikkien havaittujen CVE:ten pisteet, jotka on kartoitettu heikkoustyyppeihin ja joista on laskettu keskiarvo.

Kategoria	Kartoi-tetut CWE:t	Esiintymis-tiheys, MAX	Esiintymis-tiheys, KA	Hyödynnet-tävyys, painotettu KA	Vaikutus, painotettu KA	Kattavuus, MAX	Kattavuus, KA	Esiintyminen, yht.	CVE:t, yht.
1. Heikko pääsyoikeuksien hallinta	34	55,97 %	3,81 %	6,92	5,93	94,55 %	47,72 %	318 487	19 013
2. Salaustekniset virheet	29	46,44 %	4,49 %	7,29	6,81	79,33 %	34,85 %	233 788	3 075
3. Koodi-injektiot	33	19,09 %	3,37 %	7,25	7,15	94,04 %	47,90 %	274 228	32 078
4. Epäturvallinen suunnittelu	40	24,19 %	3,00 %	6,46	6,78	77,25 %	42,51 %	262 407	2 691
5. Virheet konfiguraatiossa	20	19,84 %	4,51 %	8,12	6,56	89,58 %	44,84 %	208 387	789
6. Haavoittuvaiset ja vanhentuneet komponentit	3	27,96 %	8,77 %	5,00	5,00	51,78 %	22,47 %	30 457	0
7. Tunnistus- ja todentamisvirheet	22	14,84 %	2,55 %	7,40	6,50	79,51 %	45,72 %	132 195	3 897
8. Ohjelmistojen ja tiedon eheysvirheet	10	16,67 %	2,05 %	6,94	7,94	75,04 %	45,35 %	47 972	1 152
9. Puutteet kirjauksessa ja valvonnassa	4	19,23 %	6,51 %	6,87	4,99	53,67 %	39,97 %	53 615	242
10. SSRF	1	2,72 %	2,72 %	8,28	6,72	67,72 %	67,72 %	9 503	358

Kuva 4. OWASP Top 10 -kategoriat ja niihin liittyvää dataa [17; 18; 19; 20; 21; 22; 23; 24; 25; 26].

Mitä korkeampi pisteytys hyödynnettävyydellä on, sitä helpommin kyseistä heikkoutta on hyödyntää. Vaikutuksella taas korkeampi pisteytys kuvastaa suurempaa vahinkoa. Kattavuus tarkoittaa niiden sovellusten prosentuaalista määrää, jotka on testattu kategoriaan liittyviltä heikkoustyypeiltä. Yhteenlasketut esiintymiset tarkoittaa sovellusten määrä, joista kategorian heikkouksia löytyi. Yhteenlasketut CVE:t, eli kaikki erityyppiset haavoittuvuudet kategorian sisällä ovat kuvan taulukon viimeisessä sarakkeessa oikealla. [16.] Aikaisemmin esitettyä kritiikkiä dataa kohtaan on tässä korostettava, sillä tarkkaa selitettä siitä, miten esiintymistiheyden maksimi ja keskiarvo on johdettu, ei ole esitetty. Sama ongelma on havaittavissa kattavuuden kohdalla.

3.1 Heikko pääsyoikeuksien hallinta

Pääsyoikeuden hallinnalla tarkoitetaan sitä, että käyttäjä ei pääse käsiksi heille asetettujen oikeuksien ulkopuolella oleviin tietoihin ja toimintoihin. Virheet pääsyoikeuksien hallinnassa saattavat johtaa salatun tiedon paljastumiseen, luvattomaan manipulointiin tai tuhoutumiseen. [27.] Riskinä on siis käyttäjän

mahdollisuus suorittaa vahingollisia toimintoja tai saavuttaa dataa, eli tietoa, joka pahimmillaan voi aiheuttaa suuren luokan tietoturvaloukkauksia.

Tyypillinen esimerkki puutteellisesta pääsyoikeuksien hallinnasta on sellainen sovellus, joka ei suojaa sensitiivistä tietoa ja sen muokkaamista todentamisella, tai jättää sovelluksen toiminnallisuuksia vapaasti käytettäväksi ilman käyttäjän käyttöoikeuksien tarkistamista [27]. Esimerkiksi ylläpitäjän on tarkoituksenmukaista pystyä lisäämään uusia käyttäjiä sovellukseen, mutta jos tätä toimintoa ei ole suojattu tarpeeksi hyvin, ja tavallinen käyttäjä pääsee lisäämään uusia käyttäjiä, voisi hän mahdollisesti lisätä uuden käyttäjän ylläpitäjän oikeuksilla, mikä aiheuttaisi merkittävän tietoturvauhkan koko sovellukselle ja sen infrastruktuurille. Tämänlainen uhka voi toteutua, jos tavallisella käyttäjällä on esimerkiksi pääsy rajoitetuksi tarkoitettuihin ohjelmointirajapintojen metodeihin tai hänellä on mahdollisuus ohittaa todentaminen muokkaamalla URL-osoitetta [17].

Kuten kaikkia tietoturvauhkia, myös pääsyoikeuksiin kohdistuvia uhkia voidaan torjua ja riskejä vähentää. Pääsyoikeuksien hallinnan tarkoituksena on pitää tietyt resurssit luvattomien käyttäjien saavuttamattomissa. Sovelluksen olisi tärkeä vahvistaa käyttäjätiedot ja -oikeudet joka kerta palvelimelta, kun tehdään sellaisia toimintoja, jotka oikeuksia tarvitsevat [27]. Jos oikeuksia ei pystytä vahvistamaan, tulisi sovelluksella olla sellainen toiminnallisuus, että käyttäjän toiminta voidaan keskeyttää ja sallia vasta, kun vahvistus voidaan varmistaa. Myös istuntoavaimet tulisi invalidoida tietyn ajan jälkeen, tai heti, kun käyttäjä on kirjautunut ulos. [17.]

3.2 Salaustekniset virheet

Salaustekniset eli kryptograafiset virheet liittyvät salaustekniikoiden virheiden tai niiden puutteellisuuden aiheuttamiin tietoturvariskeihin. Salaustekniikoita käytetään web-sovelluksissa monilla eri tasoilla suojaamaan tietoa, joka ei ole tarkoitettu avoimeksi tai kaikille tarjottavaksi. Esimerkiksi tietokantaan tallennetut käyttäjätiedot ja salasanat yleensä suojataan eli kryptataan jollain algoritmilla, kuten SHA-tiivistefunktiolla selkotekstin salaamiseksi. [28.] Salattu teksti näyttää

ulospäin hölynpölyltä, mutta salausavaimella salakirjoituksen voi muuntaa takaisin selkotekstiksi ja näin lukea sen sisällön.

Heikko tai olematon salausalgoritmi voi altistaa arkaluontoista dataa, kuten salasanoja, potilastietoja, liikesalaisuuksia tai maksutietoja, ulkopuolisten osapuolien saataville ja varastettavaksi. Muita yleisiä salausvirheitä on esimerkiksi tiedon välittäminen selkotekstinä, vanhojen algoritmien käyttö, kovakoodatut salasanat asetustiedostoissa, väärät käytännöt salausavainten hallinnassa ja niin edespäin [29].

Jotta web-sovelluksen turvallisuus voidaan varmistaa salausteknisten seikkojen osalta, on tärkeätä ymmärtää dataa ja sen eri laatuja. Kaikkea tietoa ei ole perusteltua salata, jos sen tarkoitus on olla saavutettavissa avoimesti kaikille. Turvallisuuden vuoksi sovelluksen keräämää ja käyttämää dataa tulisi monitoroida ja luokitella sekä salata sen perusteella, onko tieto arkaluonteista tai luottamuksellista. Arkaluonteinen data tulisi poistaa heti, kun sillä ei ole tosiasiallista käyttöä. Istuntoavaimet, jotka sisältävät arkaluontoista tietoa, tulisi korvata sijaisavaimella, joka ei arkaluonteista dataa sisällä. [29.]

3.3 Koodi-injektiot

Injektiohyökkäys tarkoittaa haitallista toimintaa, jossa hyökkääjä yrittää syöttää haitallista koodia tai komentoja tietokonejärjestelmään tai verkkosivustolle vihamielistä tarkoitusta varten. Tämä voi johtaa tiedon varastamiseen, virusten levittämiseen, järjestelmän hallitsemattomaan käyttöön tai minkä tahansa muun haitallisen toiminnan toteuttamiseen [30]. Injektiohyökkäykset ovat yleisiä ja ne voivat kohdistua mihin tahansa verkkoon kytkettyyn järjestelmään, mukaan lukien palvelimet, verkkosivustot, tietokannat ja mobiililaitteet.

Esimerkkejä injektiohyökkäyksistä:

1. SQL-injektio: SQL-injektio on yksi yleisimmistä injektiohyökkäyksistä, joka kohdistuu tietokantoihin. Hyökkääjä pyrkii lähettämään haitallisia

komentoja web-sovelluksen kautta tietokantaan, joka hyökkääjän onnistuessa suorittaa lähetetyn komennon. Tuloksena voi olla tietokannan sisällön paljastuminen, muokkaaminen, manipulointi tai poistaminen. [30.]

2. Komentorivi-injektio: Komentorivi-injektio on hyökkäys, joka tapahtuu, kun hyökkääjä lähettää haitallista tietoa järjestelmän komentoriville. Tämä voi johtaa järjestelmän kaatumiseen tai mahdollistaa hyökkääjän pääsyn järjestelmään ja sitä kautta aiheuttaa suurta ja laajaa vahinkoa. [30.]

3. XSS (Cross-Site Scripting): XSS-hyökkäys kohdistuu verkkosivustoihin. Hyökkääjä lähettää haitallista koodia (tyypillisesti JavaScriptiä) verkkosivustolle, joka suorittaa haitallisen toiminnon. Kun käyttäjä vierailee verkkosivulla, suorittaa hänen selaimensa haitallisen koodin ja altistaa käyttäjän tietoturvariskille. [30.]

4. LFI (Local File Inclusion): Tämä injektiohyökkäys kohdistuu verkkosivustoihin ja mahdollistaa hyökkääjän pääsyn järjestelmän tiedostoihin ilman oikeuksia. LFI-hyökkäys on yleinen ongelma PHP-pohjaisissa web-sovelluksissa. [30.]

5. RCE (Remote Code Execution): RCE-hyökkäys mahdollistaa hyökkääjän suorittaa koodia etänä ilman oikeuksia. Tämä voi johtaa järjestelmän kaatumiseen tai mahdollistaa hyökkääjän pääsyn järjestelmään. [30.]

Injektiohyökkäyksen torjuminen vaatii tietoturvallisuuden asiantuntijoilta ja järjestelmän kehittäjiltä palvelimen ja tietokannan tarkkaa suunnittelua ja toteutusta, jotta haavoittuvuudet voidaan tunnistaa ja korjata, ennen kuin hyökkääjä pääsee niihin käsiksi. Tietoturvaa voidaan parantaa esimerkiksi validoimalla, suodattamalla ja puhdistamalla kaikki käyttäjäsyötteillä lähetetty data, joka saapuu palvelimelle tai verkkosivustolle, sekä säännöllisesti tarkistamalla sovellus haavoittuvuuksien varalta ja monitoroimalla vilpillistä toimintaa. [19].

3.4 Epäturvallinen suunnittelu

Epäturvallinen suunnittelu on laaja käsite web-sovelluskehityksessä, mutta tiivistettynä sillä tarkoitetaan erilaisia heikkouksia sovelluksen suunnittelussa ja arkkitehtuurissa [20]. Turvallinen suunnittelu usein jää kehittäjiltä huomioimatta, sillä sen ei ole tavallisesti ajateltu kuuluvan perinteisten tietoturvauhkien piiriin, kuten muut web-sovellusten uhkat, joita tässäkin tutkielmassa on esitelty [31]. Epäturvallinen suunnittelu kuitenkin voi johtaa näihin tietoturvauhkiin. Koska turvallisen suunnittelun konseptissa on kyse suunnittelun filosofiasta osana sovelluskehityksen prosesseja, on se ymmärrettävästi jäänyt vähemmälle huomiolle konkreettisten tietoturvauhkien joukossa. Epäturvallista suunnittelua ei kuitenkaan pidä sekoittaa epäkohtiin implementaatiossa, sillä vaikka nämä asiat risteävät helposti keskenään, on niillä eri juurisyyt ja keinot korjaamiseen [20].

Turvallinen suunnittelu tulisi omaksua osaksi sovelluskehityksen prosesseja jo heti alkuvaiheessa ja istuttaa koko kehittäjätiimin työskentelykulttuuriin. Tämä pitää sisällään tietoturvan testaukset ja niiden tuloksien analysoinnin sekä ratkaisujen pohtimisen kehitystyötä tekevän henkilöstön kesken. Turvallisen suunnittelun on oltava holistinen ajattelutapa sovelluskehityksen koko elinkaaren aikana. [31.] Näin saavutetaan kehitysprosessi, jossa tietoturva huomioidaan kaikissa kehitystyön vaiheissa heti alusta alkaen ja lisätään kehitystyötä tekevän henkilöstön vastuuntuntoa tietoturvasta.

3.5 Virheet konfiguraatiossa

Virheet palvelimen tai verkkosivun tietoturva-asetuksissa johtuvat alustamisvaiheessa tehdyistä virheistä ja puutteellisesta konfiguroinnista. Käyttöönotto on esimerkiksi saatettu tehdä oletusasetuksilla, mikä on erittäin vakava riski tietoturvalle [32]. Väärin konfiguroidun web-sovelluksen riskit kasvavat sovelluksen jokaisella kerroksella sekä mahdollisen pilvipalvelun tarjoajalla [21].

Web-sovellusten tietoturvassa on useita mahdollisia aukkoja laajalla alueella, kun tarkastellaan koko sovellusta sisältäen sen kaikki kerrokset, eli eri ohjelmat,

komponentit ja rajapinnat, jotka ovat sovelluksen käytössä. Tyypillinen virhe sovelluksen arkkitehtuurissa on jättää turhat ominaisuudet, kuten käyttämättömät verkkoportit, palvelut ja sivut auki, sekä tarpeettomat käyttäjät ja käyttöoikeudet päälle [21]. Näitä auki jätettyjä ominaisuuksia voidaan käyttää hyökkäyksissä helposti hyödyksi monella eri tavalla. Käyttämättömiä ominaisuuksia ei välttämättä myöskään seurata usein, joten ne helposti jäävät päivittämättä ja täten vanhenevat. Vanhentuneet versiot ominaisuuksissa ja ajurit komponenteissa ovat tietoturvariski. Myös oletuskäyttäjien jättäminen toimintaan on vakava tietoturvariski, sillä usein näiden käyttäjien käyttäjänimet ja salasanat ovat yleisessä tiedossa tai muuten helposti arvattavissa, mikä altistaa hyökkäyksille, jotka pahimmillaan vaarantavat koko sovelluksen tai jopa koko yrityksen infrastruktuurin ja arkaluonteiset tiedot. [21.] Muita virheitä ovat esimerkiksi turvattomat asetukset palvelimella, ohjelmointiviitekehyksissä ja -kirjastoissa sekä tietokannassa. Lisäksi virheenhallinnan liian avoimilla ja runsaasti tietoa sisältävillä virheraporteilla voidaan paljastaa hyökkääjälle hyödyllistä tietoa sovelluksen toiminnasta. [21.]

Jotta tietoturva-aukot sovelluksen eri tasoilla voidaan paikata, on ensin opittava ja ymmärrettävä sovelluksen toiminnot ja se, miten nämä ominaisuudet käyttäytyvät. On tärkeää olla tietoinen sovellusta koskevan infrastruktuurin koostumuksesta reaaliajassa ja hahmottaa sen tarpeettomat osat. Kun systeemi ymmärretään, voidaan eliminoida turhat ominaisuudet ja puutteelliset tietoturva-asetukset korjata. [32.] Tulevaisuudessa puutteita voidaan minimoida tekemällä julkaisuympäristöjen konfiguraatiosta toistettava prosessi sekä sitten automatisoimalla se. Konfiguraatio tulisi tehdä identtisesti, mutta eri pääsytiedoin, jotta useaan ympäristöön ei voida käyttää samoja salasanoja. Näin vältetään inhimilliset virheet ja uusien turvapuutteiden havainnoimisesta tulee helpompaa. [21.]

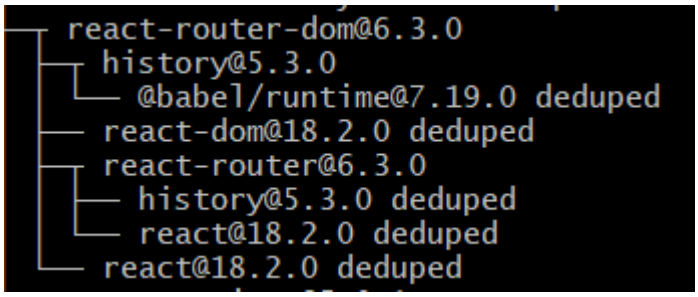
3.6 Haavoittuvat ja vanhentuneet komponentit

Web-kehityksessä ulkoisten komponenttien eli kolmansien osapuolten kirjoittamaa koodia, ohjelmistoja, kirjastoja tai kehyksiä käytetään hyvin yleisesti organisaation sisällä tuotetun koodin rinnalla. Tarkasteltaessa komponentteja web-

kehityksessä laajasti, kattaa niiden käyttö kaikki web-sovelluksen tasot aina käyttöjärjestelmästä kolmansien osapuolten koodikirjastoihin. [33.]

Komponenttien käyttö on kieltämättä hyödyllistä, sillä valmiiden ratkaisujen hyötykäyttö nopeuttaa kehitettävän ohjelmiston julkaisua, mutta silloin niiden käytön ympärille on syytä luoda toimintaperiaatteet, joilla sovelluksen tietoturva varmistetaan. Hyökkääjille tunnetut heikkoudet komponenteissa tarjoavat oivallisia pisteitä kohdennetuille hyökkäyksille [33]. Tietyissä tapauksissa ja parhaimmillaan komponenttien käyttö voi kuitenkin olla turvallisempi vaihtoehto kuin itse toteutettu osa ohjelmaa: aloittelevan web-kehittäjän toteuttama kirjautumissivu web-sovelluksessa on luultavasti huomattavasti epäturvallisempi kuin integroidun OAuth-kirjautumispalvelun käyttö.

Kolmansien osapuolten komponenttien käytössä on huolehdittava siitä, että tiedetään täydellisesti, mitä komponentteja kehitettävässä ohjelmassa käytetään ja mikä niiden tarkoitus on [33]. Kehitystyössä saatetaan esimerkiksi käyttää sisäkkäisiä komponentteja, jotka sisältävät riippuvuuksia muihin komponentteihin, minkä takia kokonaisuuden havaitseminen on haastavaa. Riippuvuuksien tarkastaminen voidaan toteuttaa manuaalisesti, esimerkiksi npm-paketinhallintaohjelmalla suorittamalla npm ls -a -komento. [5.] Kuvassa 5 näkyy komennon tuloste konsolissa, josta nähdään ohjelmiston käyttämän React Router -komponentin sisältämät riippuvuudet.



Kuva 5. React Router -komponentin riippuvuuspuu.

Paketin- ja riippuvuuksienhallintaohjelmat auttavat merkittävästi komponenttien päivityksessä ja ylläpidossa sekä kertovat, jos jokin niistä on vanhentunut. Ne

eivät kuitenkaan kerro, jos komponenttiin kohdistuu jokin uhka. Lisäksi manuaalisesti komponenttien hallinta ja turvallisuuden varmistaminen on hyvin työlästä, joten prosessi on syytä automatisoida, esimerkiksi iteroimalla komponenttien riippuvuuspuu ja vertaamalla sen sisältämiä komponentteja johonkin tunnettuun CVE-tietokantaan. Näin komponenttien versiokohtaiset uhat voidaan selvittää tehokkaasti. [5.]

Komponenttien turvallinen käyttäminen siis kokonaisuudessaan vaatii ohjelmiston ja sen osien tuntemista, järjestelmällistä päivittämistä, tietoturvauhkien automatisoitua skannaamista, komponenttien uusien versioiden testaamista sekä ongelmien korjaamista. On myös suotavaa, että edellä mainitut toimenpiteet suoritetaan tarpeeksi usein, jotta sovellus tai ohjelmisto ei ehdi olla mahdollisen uhan alla kohtuuttoman pitkää aikaa. Tietoturvauhkiin reagoimiseen auttaa myös kolmansien osapuolten komponenttien tarjoajien turvatiedotteet, joita voi usein tilata sähköpostiin. [22; 33.]

3.7 Virheet tunnistamisessa ja todentamisessa

Useat web-sovellukset tarjoavat käyttäjilleen mahdollisuuden kirjautua sisään rekisteröimillään tunnuksilla. Tätä kutsutaan todentamiseksi. Todentamisprosessin aikana käyttäjä pyritään tunnistamaan, eli yksinkertaisesti sanottuna varmistetaan, että käyttäjä on se, joksi väittää itsensä olevan. Yleisin tapa toteuttaa todentaminen on klassinen käyttäjänimen ja salasanan yhdistelmä, joskin tämän rinnalle on nykyään noussut myös muita todentamisen keinoja, kuten OAuth tai SSO, eli single sign-on. [34.] Esimerkiksi OAuth siirtää web-sovelluksen tarjoajan käyttäjänimen ja salasanan huolehtimisen kolmannelle osapuolelle, mutta huomionarvoista on, että jos käyttäjän alkuperäinen tili joutuu vääriin käsiin, voi sillä olla massiiviset vahingot käyttäjän muihin OAuth-palvelulla valtuutettuihin tileihin [5].

Hyökkäyskohteena verkkosivun todentamisvaihe on erittäin houkutteleva, etenkin jos hyökkääjä onnistuu pääsemään käsiksi sovelluksen ylläpitäjän käyttäjätunnuksiin. Varastettuja käyttäjätunnuksia voidaan käyttää hyödyksi useilla eri

tavoilla, kuten kiristämällä, disinformaation levittämisellä sosiaalisessa mediassa, rahallisen edun tavoittelulla petoksilla tai tunnusten myynnillä pimeässä verkossa. [33.] Varastettujen tunnusten kautta aiheutettu vahinko riippuu verkkosivun luonteesta, mutta ei rajaudu siihen, jos käyttäjä on käyttänyt vuotanutta käyttäjätunnusta ja salasanaa useammassa palvelussa ja hyökkääjä onnistuu laajentamaan niiden käyttökohdetta.

Todentamispalvelun tarjoaminen verkkosovelluksessa vaatii myös muita toimintoja käyttäjätilin hallintaan – jos käyttäjä esimerkiksi unohtaa salasanansa, on salasana pystyttävä palauttamaan. Tämän lisäksi muita toimintoja voi olla salasanan vaihto ja kirjautumisen muistaminen, jotta käyttäjän ei tarvitse aina näppäillä tunnuksiaan uudestaan. Valitettavasti nämä edellä mainitut ominaisuudet sovelluksessa kasvattavat hyökkäyspintaa, mikä on huomioitava kehitystyössä. Esimerkiksi jos käyttäjä näkee uutta tunnusta rekisteröidessään ilmoituksen verkkosivulla, missä sanotaan, että tämä käyttäjänimi on varattu, antaa tämä vilpilliselle taholle mahdollisuuden enumeroida eli listata jo olemassa olevia käyttäjänimiä. Ominaisuus on kuitenkin tarpeellinen, joten tällöin on syytä asettaa aikarajoitus useamman yrityksen jälkeen tai implementoida ominaisuuteen CAPTCHA-testi, jolloin automatisoitua enumerointia saadaan hidastettua. [34.]

Edellä mainittuja verkkosovelluksen enumeroituja käyttäjänimiä voidaan käyttää brute-force-hyökkäyksessä hyödyksi iteroimalla löydettyjä käyttäjänimiä listaan, joka sisältää yleisimmin käytettyjä heikkoja salasanoja. Lisäksi todentamisvaihe voi olla haavoittuvainen myös esimerkiksi XSS-, CSRF- ja injektiohyökkäyksille sekä istunnon kaappauksille [34].

Voidaankin todeta, että hyökkäyspintaa todentamisessa on runsaasti, ja siksi onkin olennaista, että web-sovellus suojataan mahdollisimman huolellisesti implementoimalla suojaus automatisoituja hyökkäyksiä vastaan, estetään heikkojen salasanojen luonti, toimitetaan salasananpalautuslinkki ulkoista kanavaa pitkin ja asetetaan tälle aikaikkuna sekä huolehditaan, että istuntoavaimet ovat tarkoituksenmukaisesti valideja, tarvittaessa kertakäyttöisiä [23].

3.8 Ohjelmistojen ja tiedon eheysvirheet

Ohjelmistojen ja tiedon eheys ovat kriittinen osa web-sovelluksen turvallisuutta, millä varmistetaan ohjelmiston koodin ja infrastruktuurin suojaus valtuuttamattomilta muutoksilta. Edeltävissä kappaleissa käytiin läpi, että useat web-sovellukset rakennetaan käyttäen hyödyksi kolmansien osapuolien komponentteja, kuten koodikirjastoja. Ongelmia eheyden kanssa voi syntyä, kun näitä komponentteja käytetään varmentamattomista ja epäluotettavista lähteistä, kuten pakettivarastoista (engl. repository) tai sisällönjakoverkosta, joiden kautta sovelluksen infrastruktuuriin saattaa päästä peukaloitua tai sen toimintaa vaarantavaa sisältöä. Varmentamaton pakettivarasto, josta sovelluksen käyttämä komponentti hakee päivityksiä, voi olla vaarantunut, jolloin on riski, että sovellus lataa lähteestä tiedoston, jonka ei kuuluisi olla siellä tai jonka puutteelliset tietoturva-asetukset vaikuttavat kielteisesti sovelluksen turvallisuuteen. Jos sovellusta ei ole suojattu tämänlaisia hyökkäyksiä ja riskejä vastaan päivitysten eheyden tarkistamisella, sisältö ladataan suoraan sellaisenaan. [24; 35.]

Jatkuvan integroinnin ja julkaisun järjestelmissä, joissa sovellus hakee automaattisesti päivityksiä ja ajaa ne suoraan kehitykseen tai tuotantoon, voi ilmaantua edellisessä kappaleessa mainittuja riskitekijöitä, joiden kautta sovellus saattaa altistua haavoittuvuuksille ja hyökkäyksille. Automatisoiduissa päivityksissä riski nousee silloin, kun tarkoituksenmukaisesta eheyden tarkistamisesta ei ole huolehdittu CI/CD-putkessa, mikä antaa vilpilliselle toiminnalle hyvin laajan hyökkäyspinta-alan, joka pahimmillaan jää huomaamatta pitkäksi aikaa. [35.]

Eheysvirheiden välttämiseksi komponentteja ja muuta sovelluksen infrastruktuuriin kuuluvaa ladattaessa ulkoisista lähteistä, on suositeltavaa käyttää digitaalisia allekirjoituksia, joiden avulla ladattavan sisällön ja lähteen luotettavuus voidaan varmentaa. Paketinhallintajärjestelmiä käytettäessä tulisi tarkistaa, että vain luotetut pakettivarastot ovat sallittuja käytettäväksi. Joissain tapauksissa voi olla tarkoituksenmukaista järjestää organisaation sisäinen pakettivarasto, johon vain luotetut ja varmennetut ohjelmistot, kirjastot ja riippuvuudet on

varastoitu. Olennaista on myös, että koodin ja tietoturva-asetusten muutoksille on käytössä hyvien käytäntöjen mukainen katselmointi. [24.]

CI/CD-putkien suhteen on tärkeää asettaa pääsyoikeuksille vain tarpeelliset valtuudet, jotta oikeuksien eskaloinnilta vältytään. Lisäksi kehitys- ja julkaisuputkien eriyttämisellä voidaan haavoittuvuuden realisoituessa minimoida hyökkäyksen laajuutta. [35.]

3.9 Puutteet kirjauksessa ja valvonnassa

Web-sovelluksen turvallisuuden ja siihen kohdistuvien väärinkäytösten kannalta on olennaista, että sovelluksen toiminnallisuuksia ja käyttäjiä valvotaan (engl. monitoring) sekä näistä koostuvat tapahtumat kirjataan (engl. logging) talteen. Vaikka puutteet kirjauksessa ja valvonnassa eivät suoranaisesti ole haavoittuvaisuuksia, ne altistavat järjestelmän niille. [36.]

Tapahtumien kirjaaminen auttaa seuraamaan ja analysoimaan sovelluksen elinkaaren aikana tapahtuvia normaaleja toimintoja, kuten käyttäjätapahtumia, mutta myös varoituksia ja virhetilanteita, esimerkiksi sisäänkirjautumisessa tapahtuvan väärän salasanan syöttäminen. [36.] Puutteellinen kirjaus ja monitorointi johtaa siihen, että aktiivisia hyökkäysyrityksiä ei välttämättä havaita, mikä antaa vilpilliselle taholle vapaat kädet toimia rauhassa [25].

Käyttäjätapahtumien, kuten transaktioiden, sisäänkirjautumisen tai sen epäonnistumisen jättäminen tallentamatta ovat puutteita kirjauksessa [25]. On tarkoituksenmukaista, että esimerkiksi samasta IP-osoitteesta tulevaa epäonnistuneiden kirjautumisyritysten virtaa voidaan automaattisesti valvoa lokin pohjalta ja analysoida, onko kyseessä potentiaalinen hyökkäysyritys, kuten brute-force-hyökkäys. Tällöin hyökkäykseen voidaan reagoida nopeasti tarvittavin toimin, mielellään automaattisesti, jos ohjelmisto on kehitetty asianmukaisella kompetenssilla. Tämän lisäksi varoitus- ja virhetilanteista generoitujen ilmoitusten tulisi olla ymmärrettäviä ja sisällöltään riittävän kattavia [25].

Pelkkä lokien pitäminen ei ole riittävää, vaan niitä tulee myös valvoa, jotta epänormaali käyttäytyminen järjestelmässä, web-sovelluksessa ja sen käyttäjissä voidaan ajoissa havaita ja tunnistaa. Valvonnan piiriin kuuluu myös ohjelmointirajapinnat, sillä nämäkin ovat tyypillinen vilpillisen toiminnan kohde [25]. Valvonta on epäonnistunut, jos tietoturvamurtoja ei huomata. [36.]

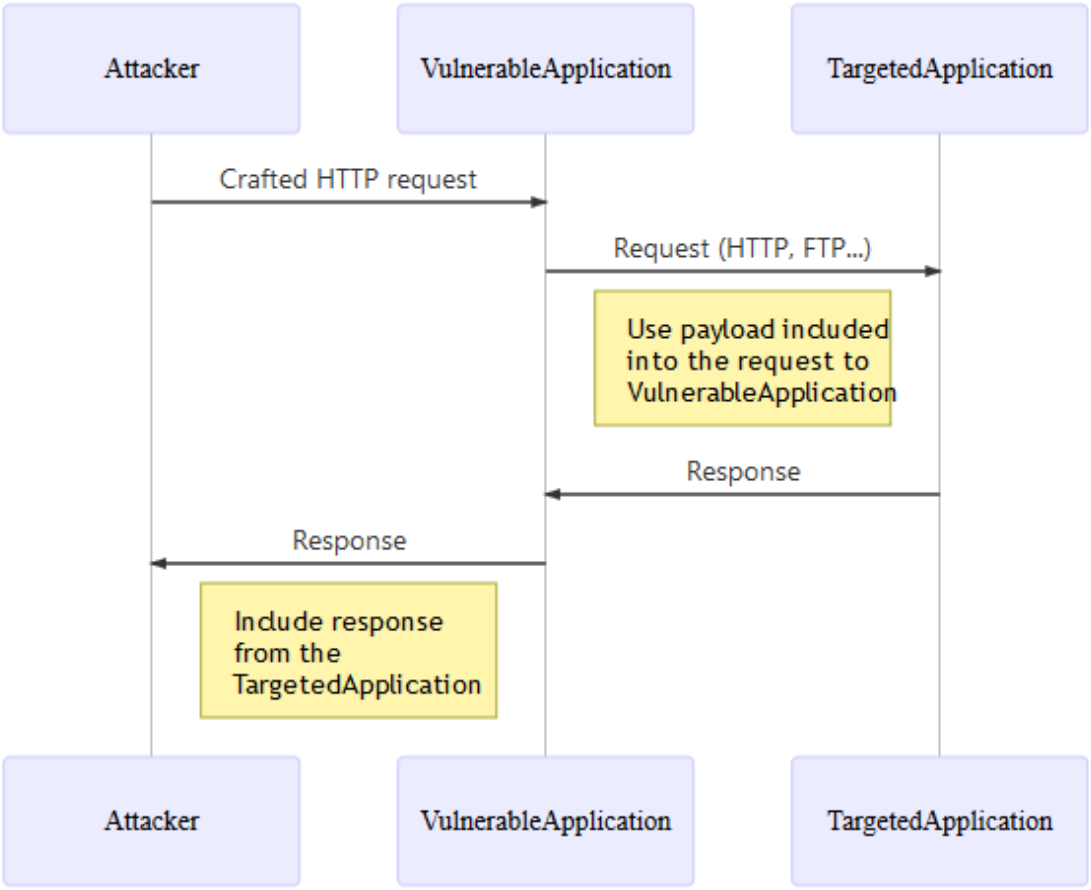
Virheiden ja puutteiden välttämiseksi kirjauksessa ja valvonnassa on varmistettava, että kaikki virhetapahtumat sisäänkirjautumisessa, pääsyoikeuksien hallinnassa ja syötteiden validoinnissa kirjataan lokiin tarpeellisin tiedoin sekä tunnistetaan näistä vilpillinen toiminta, johon myös tarvittaessa vastataan nopeasti [25]. Loki itsessään on myös syytä suojata injektiohyökkäyksiltä ja muilta siihen kohdistuvilta pahansuovilta hyväksikäytön yrityksiltä [36].

3.10 Server Side Request Forgery (SSRF)

SSRF-hyökkäyksessä vilpillinen taho pyrkii saamaan palvelinpuolen ohjelman tekemään pyynnön sellaisiin resursseihin, joihin normaalisti käyttäjällä ei pitäisi olla pääsyä, esimerkiksi organisaation sisäisiin resursseihin tai joihinkin muihin ulkoisiin resursseihin, joiden kautta hyökkääjä pääsee käsiksi sensitiiviseen tietoon [37]. Vaikka organisaation sisäiset resurssit olisi suojattu palomuurilla tai VPN:llä, eikä tavoitettavissa ulkoisesta verkosta, voi SSRF-hyökkäyksellä hyökkääjä saavuttaa nämä resurssit, sillä pyyntö niihin tulee näennäisesti luotettavasta lähteestä, eli organisaation omalta palvelimelta, tai localhost-osoitteesta, jos hyökkääjä yrittää käyttää hyödykseen loopback-käyttöliittymää. [38].

Web-sovelluksen riski altistua SSRF-hyökkäykselle ilmenee, kun sovellus noutaa dataa jostain osoitteesta, esimerkiksi käyttäjän profiilikuvan ladattavaksi ulkoisesta resurssista ilman, että käyttäjän syöttämää URL-merkkijonoa validoidaan, jolloin web-sovellusta voidaan käyttää hyväksi pakottamalla se tekemään peukaloitu pyyntö sellaiseen resurssiin, johon ei normaalisti käyttäjällä ole pääsyä [26]. Kun hyökkääjällä on mahdollisuus päästä osittain tai kokonaan käsiksi web-sovelluksen tekemiin pyyntöihin, lisää se riskiä SSRF-hyökkäykselle URL-manipuloinnin kautta [38].

Kuvassa 6 ilmenee yleisellä tasolla SSRF-hyökkäyksen toimintaperiaate, jossa hyökkääjä lähettää peukaloidun http-pyynnön haavoittuvalle palvelimelle, joka puolestaan tekee pyynnön kohteena olevaan ohjelmaan, sovellukseen tai resurssiin. Kohteena oleva entiteetti vastaa palvelimen pyyntöön ja palvelin vastaa hyökkääjän käyttäjäpäätteeseen. Näin hyökkääjä on valjastanut haavoittuvan palvelimen omaksi hyödykseen ja pääsee käsiksi saavuttamattomiksi tarkoitettuihin resursseihin tai tietoihin.



Kuva 6. SSRF-hyökkäyksen tapahtumavirta [39].

SSRF-hyökkäyksien riskien vähentämiseksi tehokas keino on ottaa käyttöön valkolista, joka sisältää ne osoitteet, joihin web-sovellus saa ottaa yhteyttä. Näin muut verkko-osoitteet suljetaan sovelluksen ulkopuolelle. [38.] Jos valkolistan sijaan käytössä on mustalista kielletyistä verkko-osoitteista, kaikki käyttäjältä tuleva data tulisi sanitoida ja validoida palvelinpuolella, jotta peukaloituja pyyntöjä

ei voida tehdä [26; 38]. Mustalistan käyttäminen tietoturvan näkökulmasta yleisellä tasolla ei kuitenkaan ole suotavaa, sillä sen ylläpitäminen on työlästä ja virhealttius suurta, mikä saattaa jättää runsaasti aukkoja hyökkäyksille [38]. Lisäksi on huomioitava, että jos sanitointi ja validointi on tehty vain käyttäjäpuolella, on palvelimelle tuleva pyyntö helppoa pysäyttää ja lähettää eteenpäin palvelimelle peukaloituna. Muita keinoja riskien vähentämiseksi ovat http-uudelleenohjausten ja käyttämättömien URL-skeemojen estäminen sekä siitä huolehtiminen, että palvelimelta ei sallita raakoja vastauksia (engl. raw response) käyttäjien tekemiin pyyntöihin.

4 Penetraatiotestaus

4.1 Yleistä penetraatiotestauksesta

Erilaisten web-sovellusten laajentuessa ja yleistyessä tietoturvan huomioiminen ohjelmistokehityksessä on lisääntynyt tieto- ja viestintätekniikan alalla, mikä näkyy myös organisaatioiden kasvavina investointeina tietoturvaan [40]. Nykyiset ohjelmat ovat toiminnoiltaan äärimmäisen monipuolisia ja käsittelevät valtavan määrän dataa, joka saattaa olla sensitiivistä. Lisäksi mobiililaitteiden yleistyttyä ja teknologian kulkiessa taskussa joka hetki, on modernissa yhteiskunnassa lähestulkoon kaikki palvelut saavutettavissa internetin välityksellä. Kehitys on kulkenut kauaksi internetin alkuajoista, jolloin verkkosivut olivat lähinnä staattisia HTML-sivuja, joissa interaktiivisuus käyttäjän kanssa oli hyvin minimaalista. Muutos on lisännyt mahdollisuuksia vilpilliselle toiminnalle, sillä ohjelmistojen arkkitehtuurien monimutkaisuus ja laajuus luovat otollisia tilaisuuksia väärinkäytölle. Web-sovellusten kompleksisuuden kasvaessa ja hyökkäysten luonteen muuttuessa on ollut tarpeellista kehittää keinoja testata sovelluksia heikoilta kohdilta.

Penetraatiotestaus on tietoturva-ammattilaisten suorittama prosessi, jossa tietoverkon, sovelluksen, järjestelmän, ohjelman tai muun infrastruktuuriin liittyvän aspektin tietoturvahaavoittuvuuksia testataan eettisen hakkeroinnin keinoin [41].

Tämän insinöörityön puitteissa testaamme web-sovellusta, joten jatkossa viittaukset penetraatiotestaukseen tapahtuvat myös web-sovellusten kontekstissa.

Penetraatiotestauksessa simuloidaan oikean maailman hyökkäyksiä sovellusta vastaan – joskus testiympäristössä, jotta oikealle järjestelmälle ei aiheutettaisi haittaa. Testauksen aikana toteutetut hyökkäykset voivat olla automatisoituja tai manuaalisia, ja niillä pyritään paikantamaan sovelluksen ja sen infrastruktuurin heikkoja kohtia. Huomionarvoista on, että pelkät automaattiset testit eivät ole riittäviä, eivätkä ne täytä penetraatiotestauksen standardeja. Tämä johtuu siitä, että automaattiset testit antavat usein vääriä hälytyksiä tai eivät huomaa kaikkia haavoittuvuuksia, etenkään monimutkaisia sellaisia. Penetraatiotestauksen tavoitteena on selvittää mahdolliset haavoittuvuudet ja puutteet tietoturvassa, mitkä dokumentoidaan ja raportoidaan yksityiskohtaisesti testattavalle taholle. Penetraatiotestauksen viimeisessä vaiheessa asiantuntijat tyypillisesti konsultoivat palveluntilaajaa, kuinka nämä puutteet tulisi korjata. [41; 42.]

4.2 Penetraatiotestauksen vaiheet

Yleisesti käytössä oleva Penetration Testing Execution Standard (PTES) jakaa penetraatiotestauksen seitsemään vaiheeseen.

Esivaiheessa penetraatiotestaaja keskustelee asiakkaan kanssa penetraatiotestauksen tavoitteista, laajuudesta, ajankohdasta ja metodeista. Tässä vaiheessa määritellään testin yksityiskohdat, joten kommunikaatio asiakkaan kanssa on erittäin tärkeää väärinkäsitysten välttämiseksi. Tässä vaiheessa myös sovitaan, kuinka paljon tietoa järjestelmästä annetaan testaajalle käytettäväksi: black-box test – ei yhtään, white-box test – täydellinen läpinäkyvyys järjestelmästä, gray-box test – osittaiset tiedot. [41; 43.]

Tiedonkeruuvaiheessa tiedustellaan ja kartoitetaan järjestelmän toimintaa. Tietoa kerätään julkisesti saatavilla olevista lähteistä ja selvittämällä, millaisia ohjelmistoja on käytössä. [43.] Luvussa 2.2 tiedustelutoiminta yksityiskohtaisesti.

Uhkien mallintaminen on vaihe, jossa kerätyn informaation pohjalta saatujen löydösten merkittävyyttä arvioidaan sekä onnistuneen uhkakuvan realisoitumisen vaikutusta asiakkaan järjestelmään analysoidaan. Käyttökelpoisten hyökkäysmetodien toimintasuunnitelma kehitetään. [43.]

Haavoittuvuusanalyysissä testaaja aktiivisesti etsii tarkemmin haavoittuvuuksia ja arvioi niihin kohdistettujen hyökkäysten toimivuutta. Haavoittuvuuksia etsitään automatisoiduilla ja manuaalisilla työkaluilla. [43.]

Hyökkäysvaiheessa löydettyjä haavoittuvuuksia hyväksikäytetään ja varsinaiset hyökkäykset toteutetaan. [43.]

Jälkivaiheessa kerätään lisätietoa penetraatiotestauksen kohteena olevasta järjestelmästä ja etsitään mahdollisuuksia uusille hyväksikäytön kohteille. Testaaja saattaa esimerkiksi etsiä hyödyllisiä tiedostoja, tai koittaa nostaa pääsyoikeuksien tasoa, millä voisi murtautua syvemmälle varsinaisen testattavan järjestelmän ulkopuolelle. Jälkivaihe on erittäin tärkeä vaihe penetraatiotestauksessa, sillä sen aikana arvioidaan hyökkäyksestä johdetut vaikutukset asiakkaaseen ja järjestelmään. [43.]

Raportointi on penetraatiotestauksen viimeinen vaihe, jossa asiakkaalle raportoidaan kattavasti ja ymmärrettävästi tehdyt löydökset ja niiden merkitys järjestelmän turvallisuudelle. Asiakasta myös neuvotaan järjestelmän tietoturvallisuuden parantamisessa. [43.]

4.3 Valitut teknologiat

Penetraatiotestauksessa käytettävät teknologiat valitaan tarkoituksenmukaisesti testattavan kohteen mukaan. Koska tämä insinöörityö käsittelee erityisesti websovellusten tietoturvaa, valittiin tähän näkökulmaan sopivat työkalut ja testausympäristö, jotka esitellään seuraavaksi alla.

Kali Linux on Debian-pohjainen penetraatiotestaajien, tietoturva-asiantuntijoiden ja valkolakkihakkereiden suosiossa oleva käyttöjärjestelmä. Kali Linux sisältää

esiasennettuna kattavan paketin työkaluja, joita voidaan hyödyntää penetraatiotestauksessa, minkä lisäksi käyttöjärjestelmän ympärillä on erittäin aktiivinen käyttäjäyhteisö, joten resursseja ja tietoa on avoimesti saatavilla kattavasti. Edellä mainitut seikat huomioiden Kali oli helppo valinta käyttöjärjestelmäksi.

OWASP Juice Shop on avoimen lähdekoodin projekti, joka tarjoaa testausympäristön web-sovellusten tietoturvan ja haavoittuvuuksien hyväksikäytölle. Web-sovellus on tarkoituksella kehitetty turvallisuusominaisuuksiltaan puutteelliseksi ja on erinomainen alusta harjoitella OWASP Top 10 -listalta löytyvien haavoittuvuuksien hyväksikäyttöä turvallisesti.

Burp Suite on PortSwiggerin kehittämä työkalu penetraatiotestausta varten, jonka proxy-toiminto mahdollistaa selaimessa toimivan web-sovelluksen ja palvelimen välisen tietoliikenteen väliintulon. Tämä tarkoittaa sitä, että yhdistettynä muihin Burp Suiten ominaisuuksiin, on penetraatiotestaajalla hyvin laajat mahdollisuudet tarkkailla sovelluksen toimintaperiaatteita sekä tunnistaa ja testata web-sovelluksen haavoittuvuuksia. Tässä työssä käytetään Kali Linuxiin esiasennettua Burp Suiten ilmaisversiota, jonka ominaisuudet ovat rajatut, mutta suhteellisen riittävät manuaaliseen testaamiseen.

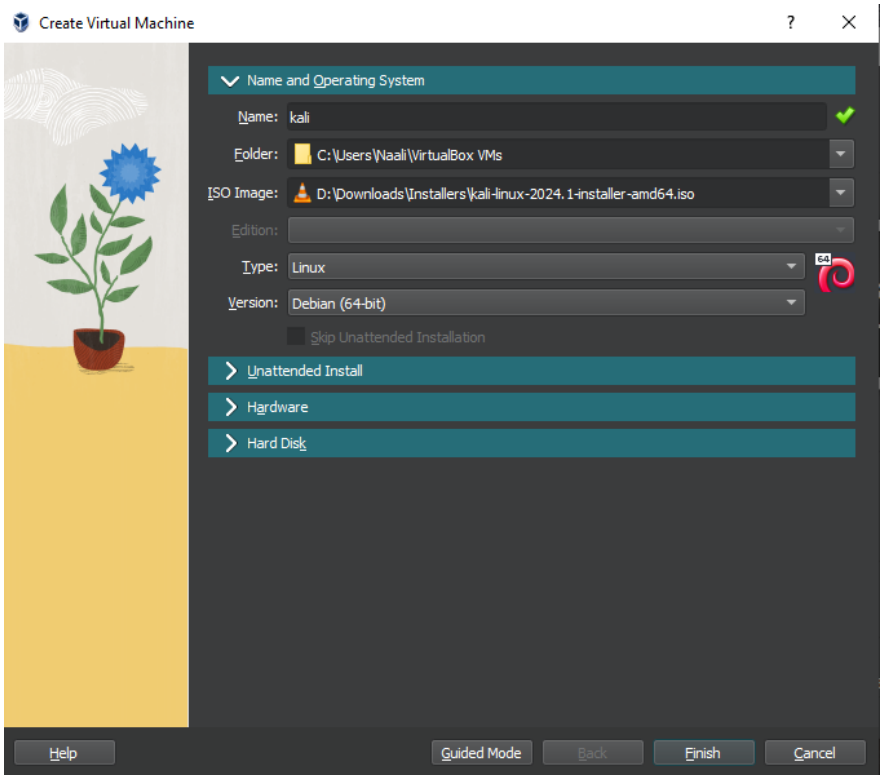
Zed Attack Proxy (ZAP) on penetraatiotestaukseen tarkoitettu työkalu, jonka toimintaperiaate on hyvin samanlainen kuin Burp Suitella. Koska tässä työssä käytetään Burp Suiten ilmaisversiota, otetaan ZAP sen rinnalle paikkaamaan puuttuvia ominaisuuksia, kuten verkkosivun automaattista skannausta ja haavoittuvuuksien paikantamista.

Muita työssä käytettyjä teknologioita ovat muun muassa *Wappalyzer*, *Sqlmap, Hashcat, John The Ripper* ja *FoxyProxy*.

Alustana testausympäristölle toimii *Oracle VM VirtualBox*, jolle Kali Linux asennus tehdään Debian 64-bittiselle virtuaalitietokoneelle.

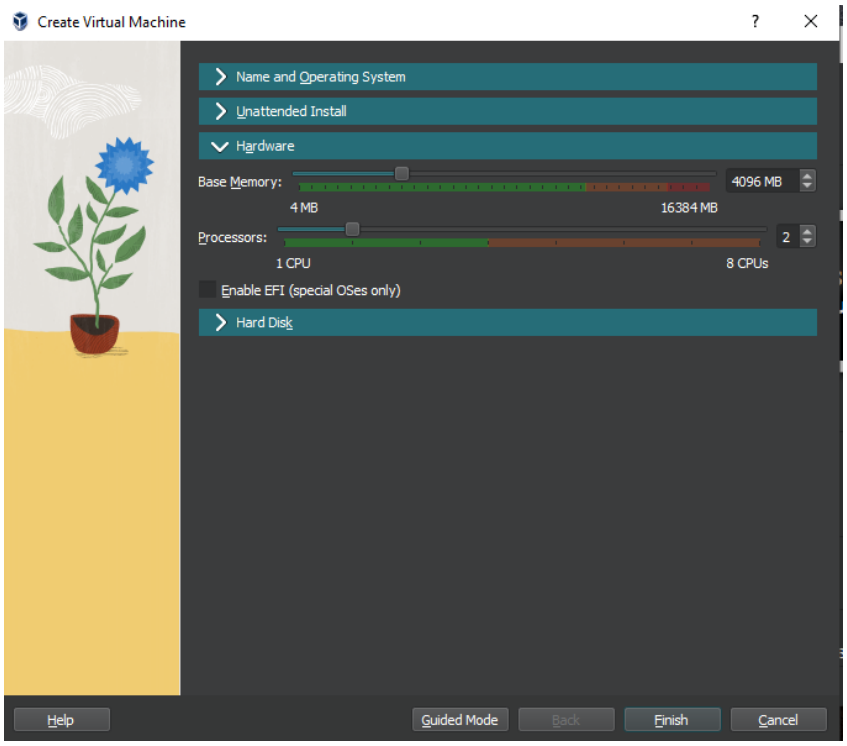
4.4 Testausympäristön pystytys

Ympäristön pystytyksen ensimmäisessä vaiheessa ladattiin Oracle VM VirtualBox -ohjelma ja Kali Linux -käyttöjärjestelmän ISO-tiedosto, joka tarvitaan käyttöjärjestelmän lataamiseen virtuaalitietokoneelle. Tämän jälkeen luotiin uusi virtuaalikone, jonka asetukset muokattiin ympäristön pystyttämistä varten sopivaksi. Kuvassa 7 nähdään virtuaalikoneen tallennuskansio ja käytettävä ISO-kuva sille asennettavasta käyttöjärjestelmästä. Virtuaalitietokoneen pohjana toimi 64-bittinen Debian Linux-jakelu.



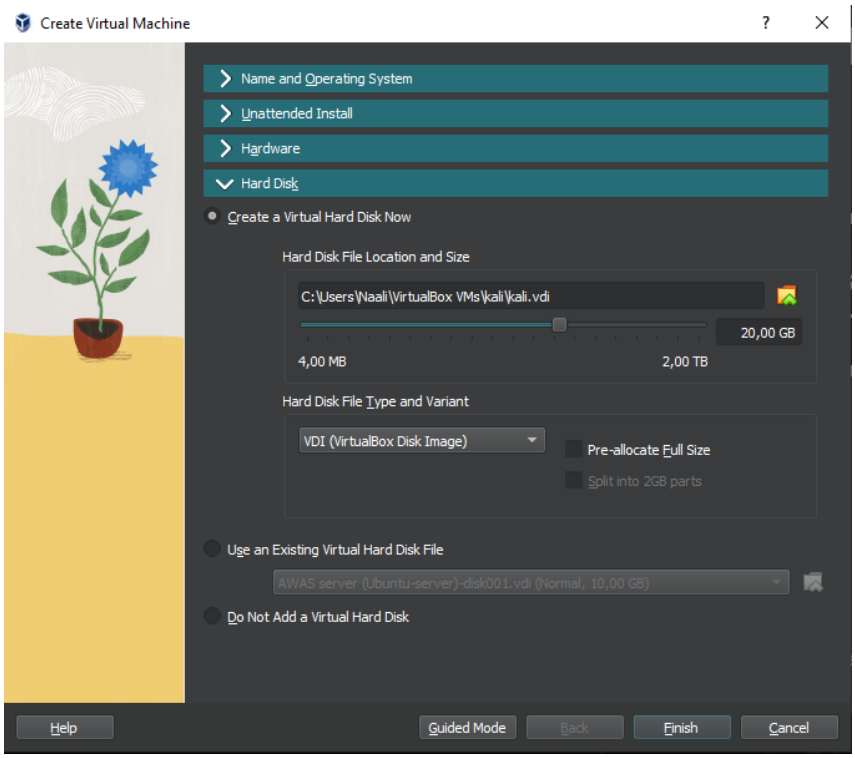
Kuva 7. Kuvakaappaus virtuaalikoneen asetusten ensimmäisestä vaiheesta.

Seuraavaksi allokoitiin virtuaalitietokoneelle neljä gigatavua muistia ja käytettäväksi kaksi prosessoria. Nämä asetukset ovat testausympäristön kannalta riittävät, mutta resursseja voitaisiin halutessa myös lisätä, mikä parantaisi ympäristön suorituskykyä. Asetukset näkyvät kuvassa 8.



Kuva 8. Kuvakaappaus virtuaalikoneen asetusten toisesta vaiheesta.

Kuvassa 9 varattiin virtuaalitietokoneelle käytettäväksi 20 gigatavua SSD-muistia, mikä on suositeltava minimivaatimus Kali Linux-asennukselle.



Kuva 9. Kuvakaappaus virtuaalikoneen asetusten viimeisestä vaiheesta

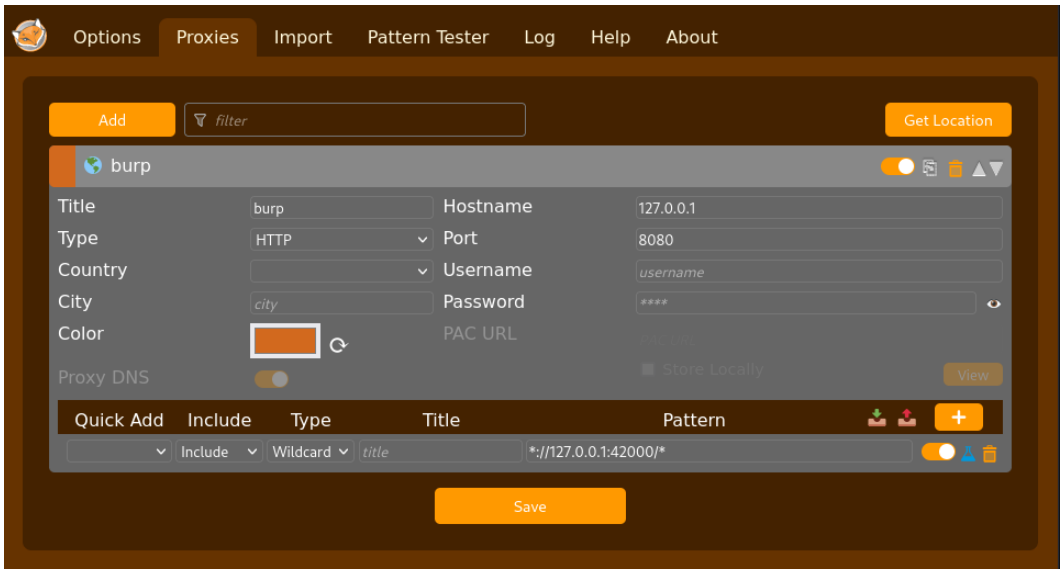
Kun virtuaalikoneen asetukset oli asetettu, alkoi käyttöjärjestelmän asentaminen. Nämä vaiheet on kuitenkin jätetty pois tästä työstä, sillä ne ovat paljolti preferenssikysymyksiä. Varsinaisen Kali Linuxin asennus sisältää esimerkiksi kohtia, joissa kysytään, haluaako tehdä asennuksen graafiselle käyttöliittymälle, vai pelkistetyn terminaalikäyttöjärjestelmän, sekä millä kielellä tahtoo käyttöjärjestelmää käyttää ja millaisen näppäimistöasettelun haluaa järjestelmälle.

Asennuksen valmistuttua varmistettiin, että käyttöjärjestelmä on ajan tasalla ja päivitettiin paketit, asennettiin Node.js, npm-paketinhallintajärjestelmä ja itse OWASP Juice Shop. Viimeisenä tarkistettiin, että Juice Shop toimii. Edellä mainitut toimenpiteet suoritettiin komennoilla terminaalissa, mitkä näkyvät esimerkkikoodissa 1.

```
#päivitetään paketit
sudo apt update
sudo apt upgrade

#asennetaan Node.js ja npm-paketinhallintajärjestelmä
sudo apt install nodejs
sudo apt install npm

#asennetaan OWASP Juice Shop
sudo apt install juice-shop

#käynnistetään Juice Shop
sudo juice-shop
```

Esimerkkikoodi 1. Suoritetut komennot terminaalissa. Risuaidalla merkittyä riviä ei ajeta terminaalissa, vaan sillä merkitään kommenttia.

Burp Suiten oman selaimen kanssa ilmeni ongelmia tuoreimman Kali-version kanssa, joten oletusselaimeksi valittiin Firefox. Jotta työskentely-ympäristö saatiin miellyttäväksi, Firefoxiin asennettiin FoxyProxy Standard -lisäosa, jolla verkkoliikenne voidaan ohjata Burp Suiteen. Oletusasetuksilla Burp Suite kuuntelee lokaalisti portissa 8080 ja apt-paketinhallintajärjestelmällä ladattu Juice Shop lokaalisti portissa 42000, joten kuvassa 10 esitetyillä FoxyProxyyn tehdyillä asetuksilla vain Juice Shopin verkkoliikenne kulkee Burp Suiten proxyn kautta.



Kuva 10. Kuvakaappaus FoxyProxyyn tehdyistä asetuksista.

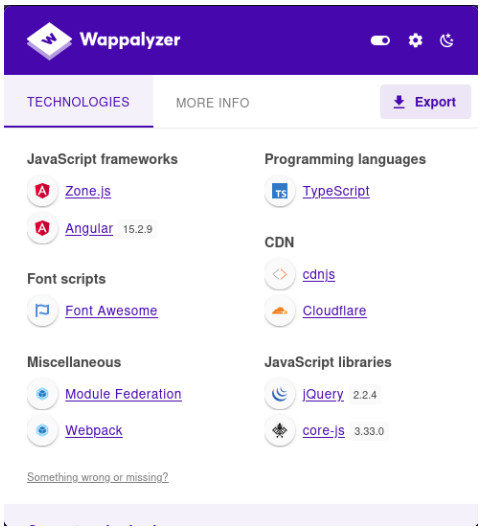
Valkolistaamalla Juice Shopin IP-osoite ohjataan muu verkkoliikenne ohi proxystä. Tämä siksi, että verkkoliikenteen väliintulo voi olla laitonta, joten on tärkeää pidättäytyä vain testattavassa, sallitussa ympäristössä.

5 Haavoittuvuuksien testaaminen

Tässä luvussa demonstroidaan haavoittuvuuksien paikantamista ja niitä vastaan hyökkäämistä käyttäen hyödyksi edellisessä kappaleessa esiteltyjä teknologioita ja ympäristöä. Penetraatiotestausta sen varsinaisessa muodossa ei kokonaisuutena suoriteta, vaikka tämä luku sisältääkin sille ominaisia vaiheita.

5.1 Juice Shopin kartoitus

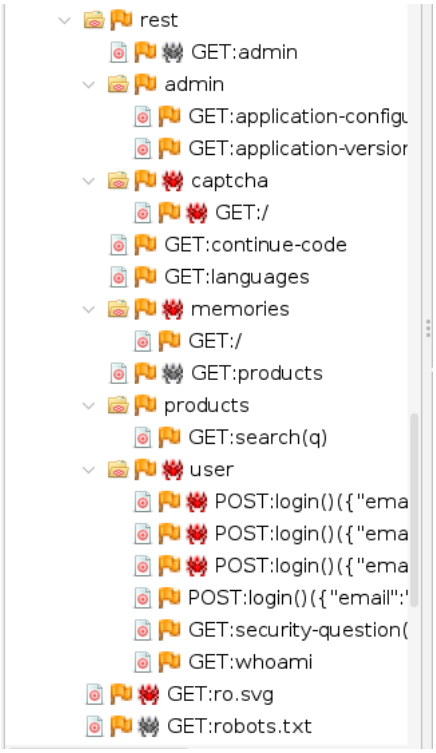
Juice Shop on avoimen lähdekoodin projekti, joten tiedot käytetyistä teknologioista olisivat helposti saatavilla nopealla Google-haulla. Tämän työn puitteissa on kuitenkin syytä demonstroida tiedustelu- ja kartoitusvaihetta käytännössä, joten alkuun suoritettiin selaimessa Wappalyzer-lisäosalla selvitys verkkosivun käyttämistä teknologoista, jonka tulokset ovat nähtävissä kuvassa 11.



Kuva 11. Kuvakaappaus Wappalyzerin tunnistamista teknologoista.

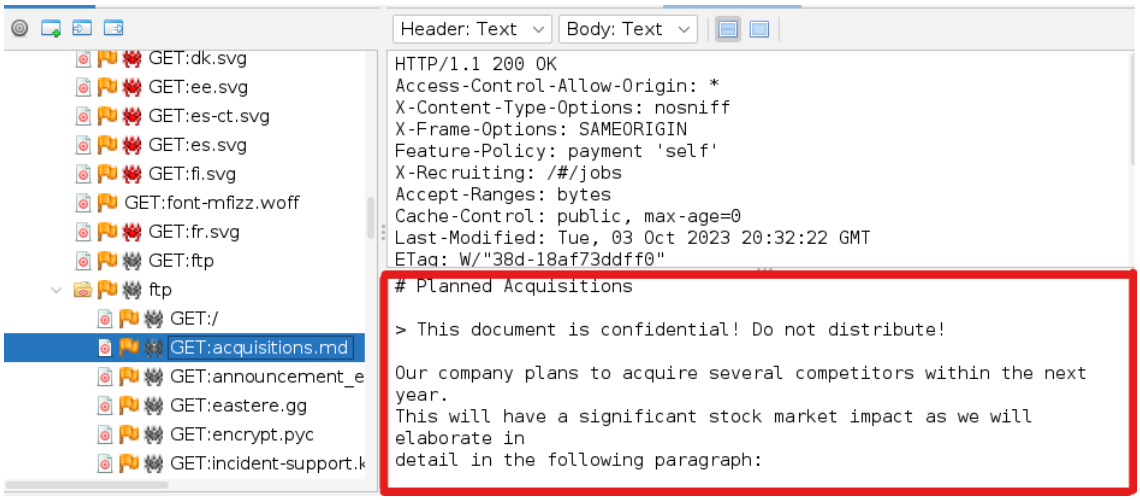
Tärkeä huomio tuloksista on, että web-sovellus on rakennettu käyttäen Angular-ohjelmistokehystä, mikä tarkoittaa sitä, että tietyt työkalut ja hyökkäykset eivät ole singe-page-sovellusta vastaan yhtä tehokkaita.

Seuraavaksi ajettiin Zed Attack Proxylla Spider- ja AJAX Spider -työkalut, joiden avulla saatiin muodostettua web-sovelluksen kansiorakenne ja löydettiin piilotettuja ja mielenkiintoisia tiedostoja, joita ei ole tarkoitettu saavutettavaksi. AJAX Spider on oleellinen työkalu single-page-sovellusten kanssa, koska se pyrkii käyttämään sovelluksen toiminnallisuuksia ja kartoittaa sitä myöten esimerkiksi erilaisia rajapintapyyntöjä. Kuvassa 12 näkyy osittainen kansiorakenne.



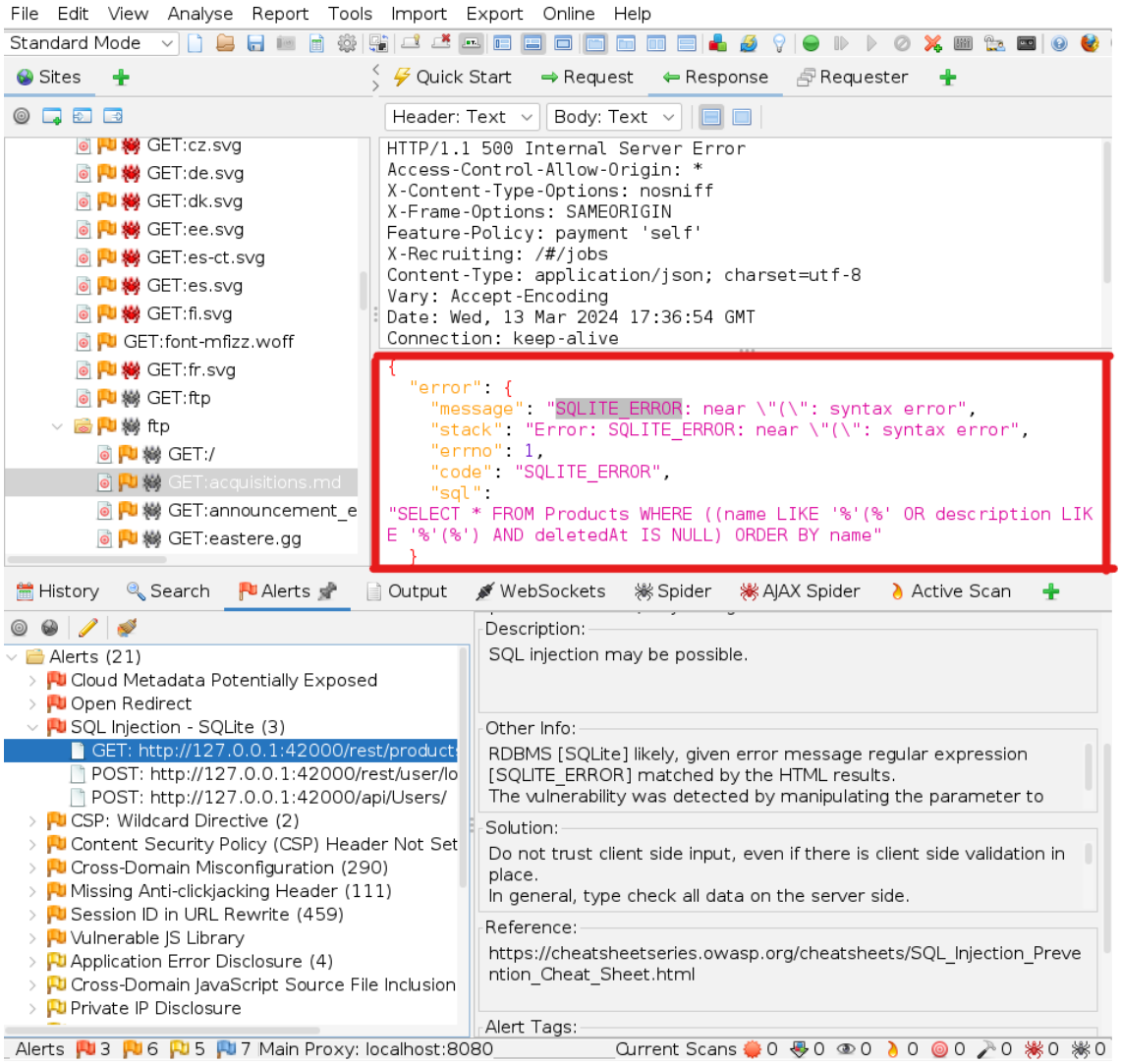
Kuva 12. Kuvakaappaus osittaisesta kansiorakenteesta.

Kansiorakenteesta löytyy heti muutamia kiinnostavia löydöksiä, kuten admin-, memories- ja ftp-kansio, joista viimeinen on jo ensimmäinen löydetty tietoturvariski, sillä se sisältää sensitiivistä dataa, joka on saatavilla erittäin helposti ulkoisille osapuolille. Kansio sisältää muun muassa varmuuskopiotiedostoja ja acquistions-tiedoston, joka on tarkoitettu luottamukselliseksi. Pohjustus on tehty, joten seuraavaksi tehtiin aktiivinen skannaus ZAPilla, jota ennen luotiin käyttäjätunnus Juice Shoppiin, jotta myös kirjautumisen taakse jäävät ominaisuudet voidaan käydä kattavasti läpi. ZAPin aktiivinen skannaus on automatisoitu hyökkäys, joka etsii ja hyväksikäyttää heikkouksia web-sovelluksessa. Aktiiviseen skannaukseen on mahdollista syöttää monipuolisesti erilaisia asetuksia – tässä tapauksessa siihen syötettiin kirjautumistiedot ja suodatettiin pois testit sellaisille teknologioille, joiden tiedetään olevan turhia tätä web-sovellusta vastaan. Kuvassa 13 näkyy vasemmalla kansiorakenne ja osa tehdyistä GET-http-pyynnöistä, oikealla ylälaatikossa tehty pyyntö ja sen alapuolella punaisella merkityssä laatikossa saatu vastaus, joka sisältää olennaisen tekstin paljastuneesta luottamuksellisesta acquisitions.md -tiedostosta.



Kuva 13. Kuvakaappaus ZAP-ohjelmasta.

Automaattinen, aktiivinen skannaus tuotti satoja hälytyksiä löydetyistä, potentiaalisista haavoittuvuuksista, joista iso osa oli kuitenkin jo nopealla tarkastuksella vääriä ja osa vain informatiivisia ilmoituksia. Esimerkiksi kaikki timestamp disclosure -hälytykset ovat vääriä ja ne voidaan jättää huomiotta. Tämä on hyvä esimerkki siitä, että automaattinen penetraatiotestaus ei ole riittävää, vaan löydöksien analysoiminen ja tehtävät jatkotoimenpiteet vaativat myös manuaalista työtä, jotta testattavan web-sovelluksen tietoturvasta saadaan realistinen tilannekuva. Automaattiset testit toimivat kuitenkin erinomaisena pohjana, kun alussa etsitään haavoittuvuuksia, ja tässäkin tapauksessa saatiin erityisen mielenkiintoinen löydös, joka on SQL-injektiosta syntynyt korkean prioriteetin varoitus, joka näkyy kuvassa 14. Samaisesta kuvasta on huomattavissa, että virheilmoitus http-vastauksessa on erittäin huonosti toteutettu, sillä se paljastaa käytettävän tietokannan teknologian, joka on SQLite.

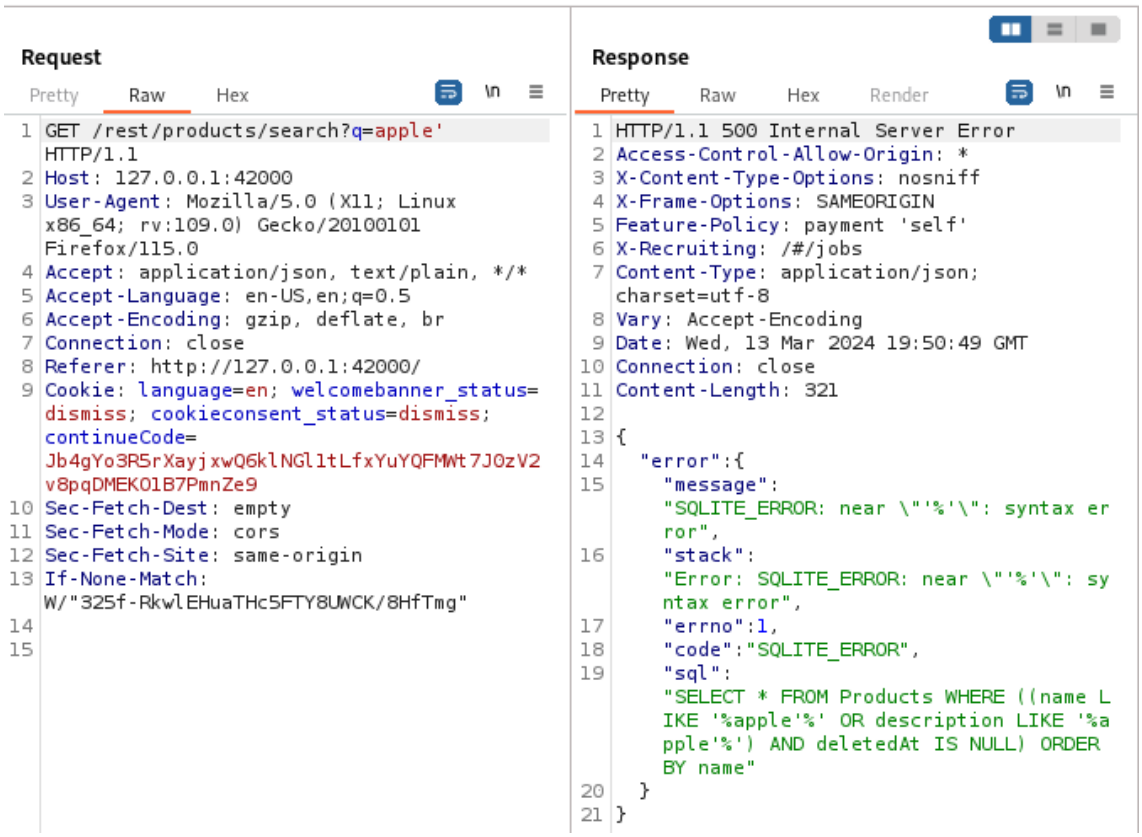


Kuva 14. Kuvakaappaus ZAP-ohjelmasta, jossa näkyy saatu virheilmoitus.

5.2 SQL-injektio

Edellisessä luvussa löytyi korkean prioriteetin varoitukset SQL-injektion mahdollisuudelle sisäänkirjautumissivulla ja tuotesivun hakutoiminnossa, joten seuraavaksi on hyvä varmistaa manuaalisesti SQL-injektiopisteet. Kartoituksen pohjalta selvisi, että Juice Shop käyttää SQLite-tietokantaa, joten injektiota voidaan testata heittomerkillä syötekentässä, mikä onnistuu erinomaisesti Burp Suiten Repeater-työkalulla. Kuvassa 15 sovelluksen hakutoiminnon tekemään GET-pyyntöön lisättiin hakusanan ”apple” perään heittomerkki, jolloin tietokanta

palauttaa virheilmoituksen virheellisestä syntaksista. Tämä tarkoittaa sitä, että syötettä ei sanitoida, jolloin haavoittuvuus on varmistettu.



Kuva 15. Kuvakaappaus Burp Suiten Repeater-työkalusta.

Sisäänkirjautumissivun haavoittuvuus osoittautui erittäin vakavaksi, sillä kokeilemalla klassista boolean-tyyppistä SQL-injektiota ” ’OR 1=1-- ”, avautui pääsy admin-käyttäjän tilille. Toimintalogiikka injektiossa perustuu siihen, että ensimmäinen hakamerkki sulkee alkuperäisen merkkijonon ja lisää sen perään ehdon, joka evaluoituu aina true-arvoksi. Lopuksi kaksi viivaa aloittaa kommentin, joka sulkee loput SQL-lausekkeesta ulos. Juice Shopin hakutoiminnosta löytynyt injektiohaavoittuvuus on astetta vaikeampi käyttää hyödyksi, joten apuna käytettiin Sqlmap-työkalua, jolla hyökkäys injektiopisteeseen automatisoitiin. Työkalun avulla saavutettiin tietoa tietokannan rakenteesta ja haavoittuvuuden alttiudesta ainakin union-tyyppiselle SQL-injektiolle, joskin huomionarvoisena seikkana testien aikana nousi työkalun hitaus ja epäluotettavuus. Työkalun hitaus mitä luultavimmin johtui virtuaalitietokoneen rajoitetusta tehosta ja

epäluotettavuus virheellisesti käytetyistä parametreistä, eli käyttäjän virheestä. Lopulta tietokannan lataaminen rajoitettiin Users-taulukkoon ja tulokset tallennettiin csv-tiedostoon, jonka data siistittiin Excelillä. Tämän lisäksi koodiesimerkissä 2 näkyvä, manuaalisesti toteutettu union-tyyppinen injektio onnistui, ja sen pohjalta saatua luotettavaa Users-taulukon dataa verrattiin automaattisesti kerättyyn dataan, josta tällöin ilmeni runsaasti virheitä. Lyhyesti sanottuna Sqlmap oli sijoitellut taulukon solujen sisältämää tietoa vääriin paikkoihin.

```
GET /rest/products/search?q=apple'))UNION+SELECT+id,role,email,pass-
word,username,totpsecret,deluxetoken,lastloginip,9+FROM+Users;--
```

Esimerkkikoodi 2. Burp Suiten Repeater työkalulla tehty SQL-injektio.

Kuvassa 16 on näkyvillä lopullinen ja siistitty data Excel-taulukkona. Taulukosta on rajattu pois epäoleellisia tietoja, kuten käyttäjätunnusten luonti-, päivitys- ja poistopäivät, sillä nämä eivät olleet työn kannalta merkityksellisiä.

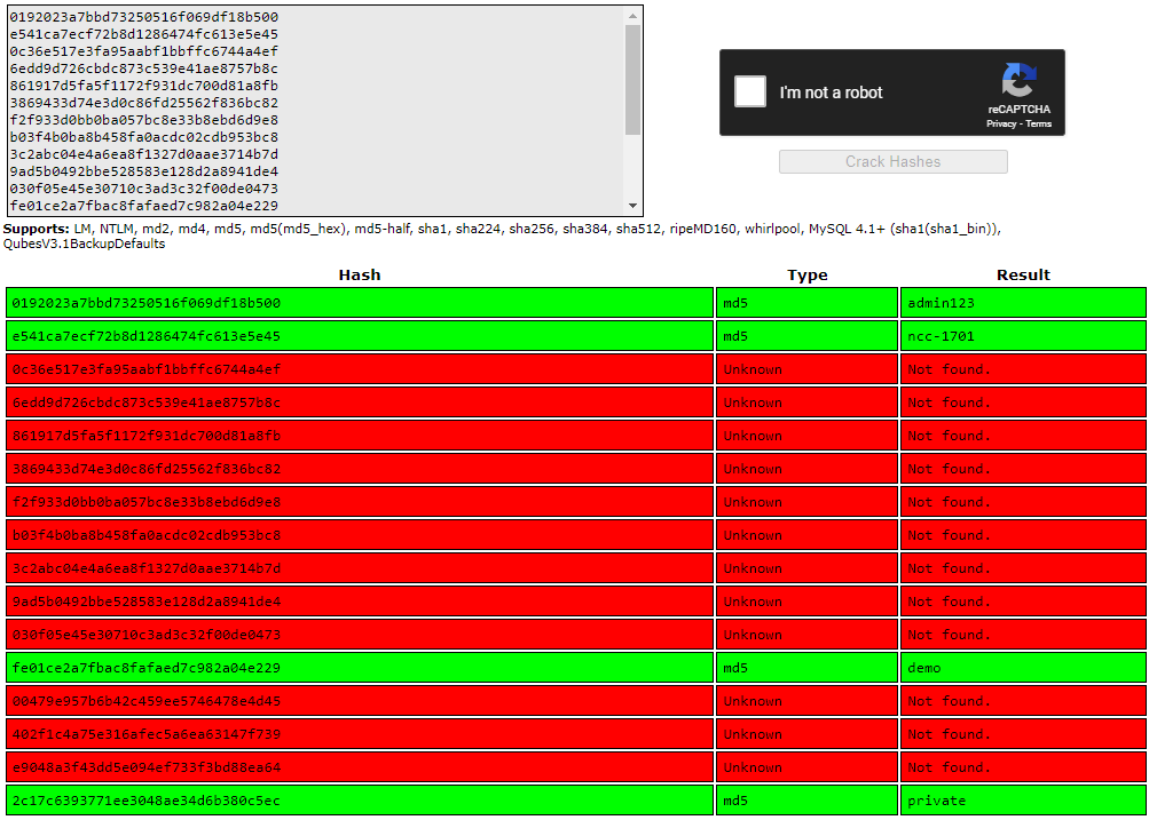
id	role	email	password
1	admin	admin@juice-sh.op	0192023a7bbd73250516f069df18b500
2	customer	jim@juice-sh.op	e541ca7ecf72b8d1286474fc613e5e45
3	customer	bender@juice-sh.op	0c36e517e3fa95aabf1bbffc6744a4ef
4	admin	bjoern.kimminich@gmail.com	6edd9d726cbdc873c539e41ae8757b8c
5	deluxe	ciso@juice-sh.op	861917d5fa5f1172f931dc700d81a8fb
6	admin	support@juice-sh.op	3869433d74e3d0c86fd25562f836bc82
7	customer	morty@juice-sh.op	f2f933d0bb0ba057bc8e33b8ebd6d9e8
8	customer	mc.safesearch@juice-sh.op	b03f4b0ba8b458fa0acdc02cdb953bc8
9	admin	J12934@juice-sh.op	3c2abc04e4a6ea8f1327d0aae3714b7d
10	admin	wurstbrot@juice-sh.op	9ad5b0492bbe528583e128d2a8941de4
11	customer	amy@juice-sh.op	030f05e45e30710c3ad3c32f00de0473
12	admin	bjoern@juice-sh.op	7f311911af16fa8f418dd1a3051d6810
13	deluxe	bjoern@owasp.org	9283f1b2e9669749081963be0462e466
14	customer	chris.pike@juice-sh.op	10a783b9ed19ea1c67c3a27699f0095b
15	accounting	accountant@juice-sh.op	963e10f92a70b4b463220cb4c5d636dc
16	customer	uvogin@juice-sh.op	05f92148b4b60f7dacd04cceebb8f1af
17	customer	demo	fe01ce2a7fbac8fafaed7c982a04e229
18	customer	john@juice-sh.op	00479e957b6b42c459ee5746478e4d45
19	customer	emma@juice-sh.op	402f1c4a75e316afec5a6ea63147f739
20	deluxe	stan@juice-sh.op	e9048a3f43dd5e094ef733f3bd88ea64
21	deluxe	ethereum@juice-sh.op	2c17c6393771ee3048ae34d6b380c5ec

Kuva 16. Osa kerätystä Users-taulukon datasta.

5.3 Salasanojen murtaminen

Luvussa 5.2 kaikkien Juice Shopin käyttäjien kirjautumistiedot onnistuttiin lataamaan, mutta salasanat ovat suojattu salausalgoritmilla. Tämä on ongelma, koska nykyisessä muodossaan salasanoja ei voi käyttää hyväksi sellaisenaan, joten ne täytyy murtaa.

Ensimmäisenä on hyvä tarkistaa, onko löydettyjä salasanoja jo ennestään murrettu, joten CrackStation-palvelulla testattiin salasanatiedot. Kuvassa 17 nähdään, että salasanoista neljälle löytyy selkotekstinen vastine CrackStationin tietokannasta, eli joku on ne aiemmin murtanut. Salasanat on pyritty suojaamaan käyttäen MD5-algoritmiä, joka on tähän tarkoitukseen erittäin heikko ja vanhentunut. Kyseessä on siis salaustekninen virhe.



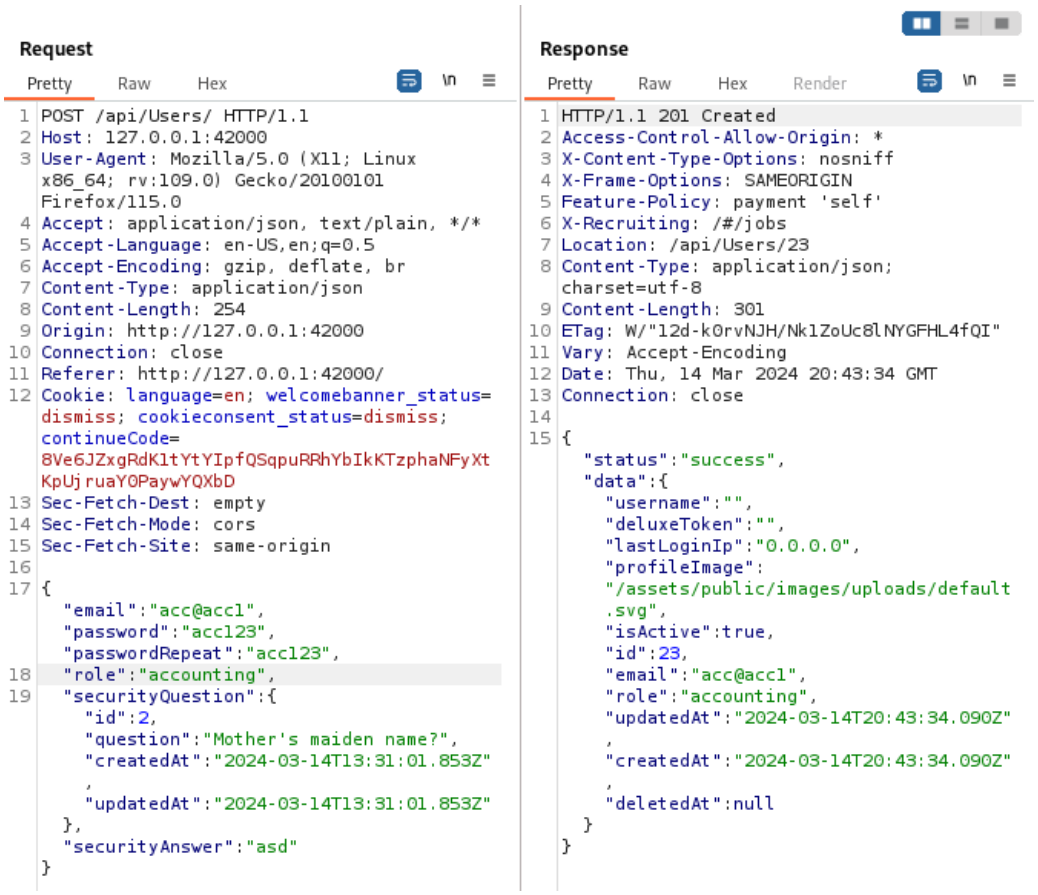
Kuva 17. Kuvakaappaus CrackStation-verkkosivulta.

Hashcat-työkalulla suoritetut testit eivät onnistuneet murtamaan muiden kuin neljän jo ennestään tunnetun salasanan salauksia, joten seuraavaksi yritettiin John The Ripper -työkalua, mikä jäi myös tuloksettomaksi yritykseksi murtaa muita salasanoja. Työkaluilla ajettiin löydetyt salasanat sanalistoja vastaan, mitkä todennäköisesti olivat tähän tarpeeseen puutteelliset, tai sitten osa salasanoista on suojattu suolauksella. Hash-analysointiin tarkoitetuilla palveluilla voidaan arvioida hajautusfunktioilla luotuja salasanasuojauksia, ja tuloksena oli kaikkien salasanojen kohdalla MD5-algoritmin käyttö ilman suolausta. Hashes.com-palvelun tietokannasta löytyi lopuksi vielä kolme ennestään murrettua salasanaa, joista yksi ei kuitenkaan ollut toimiva.

5.4 Järjestelmänvalvojan ja kirjanpidon hallintasivut

Aikaisemmissa luvuissa saavutettiin pääsy sensitiiviseen tietoon ja admin-käyttäjän kirjautumistiedot pystyttiin murtamaan. Näillä tunnuksilla kirjautuessa sisään havaitaan, että suoraa pääsyä järjestelmänvalvojan sivulle, asetuksiin tai muuhun sellaiseen ei ole saatavilla. Firefoxin kehittäjätyökalulla pikainen selaus admin-hakusanalla HTML- ja main.js -tiedostoissa tuottaa tulosta ja järjestelmänvalvojan sivu löytyy osoitteesta http://127.0.0.1:42000/#/administration. Sivu on niin helposti löydettävissä, että sen olisi voinut löytää arvaamalla, mutta samalla löytyi lukuisa määrä aikaisemmin kartoittamattomia polkuja, kuten http://127.0.0.1:42000/#/accounting sekä uusia kutsuja ohjelmointirajapintaan. Accounting- ja admin-sivu ovat molemmat lukittuna vain käyttäjille, joilla on asianmukainen rooli ja oikeudet. Aikaisemmin esitellyssä kuvassa 16 on näkyvillä käyttäjien roolit, joista voidaan päätellä, onko käyttäjällä oikeudet accounting- tai admin-sivuille. Sivuille pyrkiminen oikeudettomalla käyttäjällä ponnauttaa 403-ilmoituksen, mikä on huonoa suunnittelua.

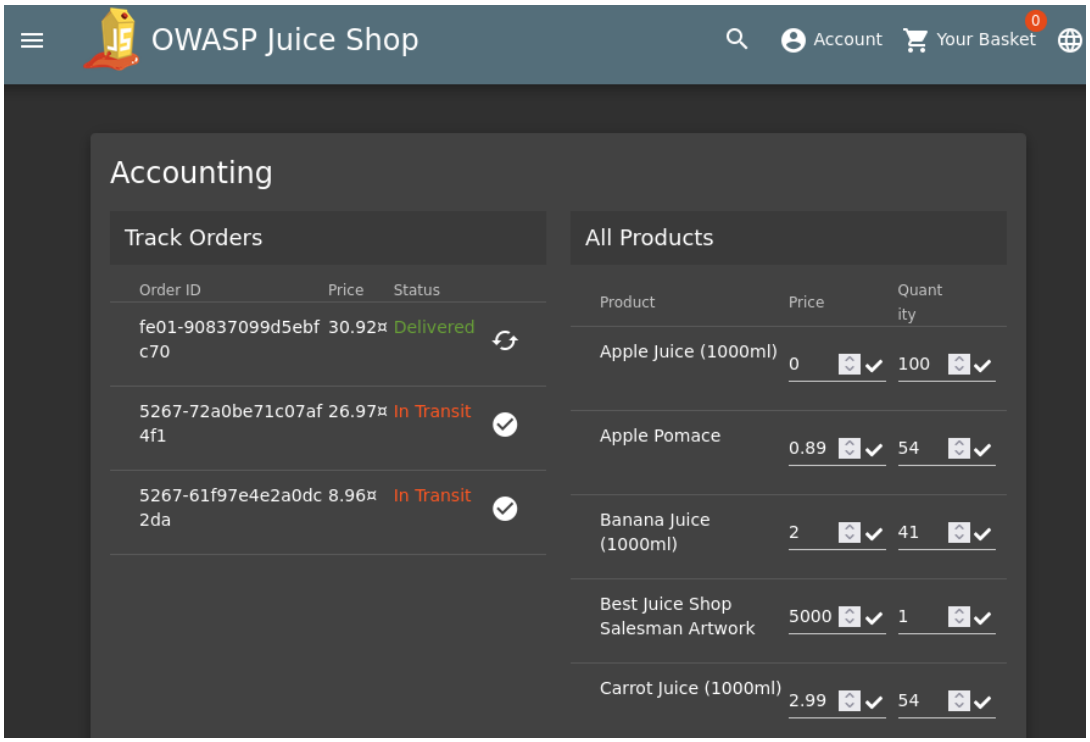
Juice Shopin toiminto käyttäjän rekisteröinnille on toteutettu heikosti, sillä palvelinpuolen ohjelma ei validoi dataa ja käyttäjän luominen accounting-roolilla onnistuu pysäyttämällä http-liikenne Burp Suitella ja manipuloimalla rekisteröintitietoja, mikä näkyy kuvassa 18. Samalla logiikalla myös admin-käyttäjän luominen on mahdollista.



Kuva 18. Kuvakaappaus Burp Suitesta ja onnistuneesta vilpillisestä rekisteröinnistä.

Accounting-sivulle pääsemällä avautuu mahdollisuus muokata olemassa olevia tuotteita ja hallita asiakkaiden tekemiä tilauksia, mutta tuotteiden määrän muokkaaminen on IP-osoitteen mukaan estetty.

Tämä osuus on osoittanut käytännössä, kuinka heikko pääsyoikeuksien hallinta yhdistettynä käyttäjän syöttämän datan puutteelliseen validointiin ja sanitointiin aiheuttaa helposti käyttäjäoikeuksien eskalaation, jolloin hyökkääjälle avautuu pääsy sellaisiin ominaisuuksiin ja toimintoihin, jotka eivät hänelle kuuluisi. Tehtyjä muutoksia tuotteisiin ja tilauksiin accounting-sivulla näkyy kuvassa 19.

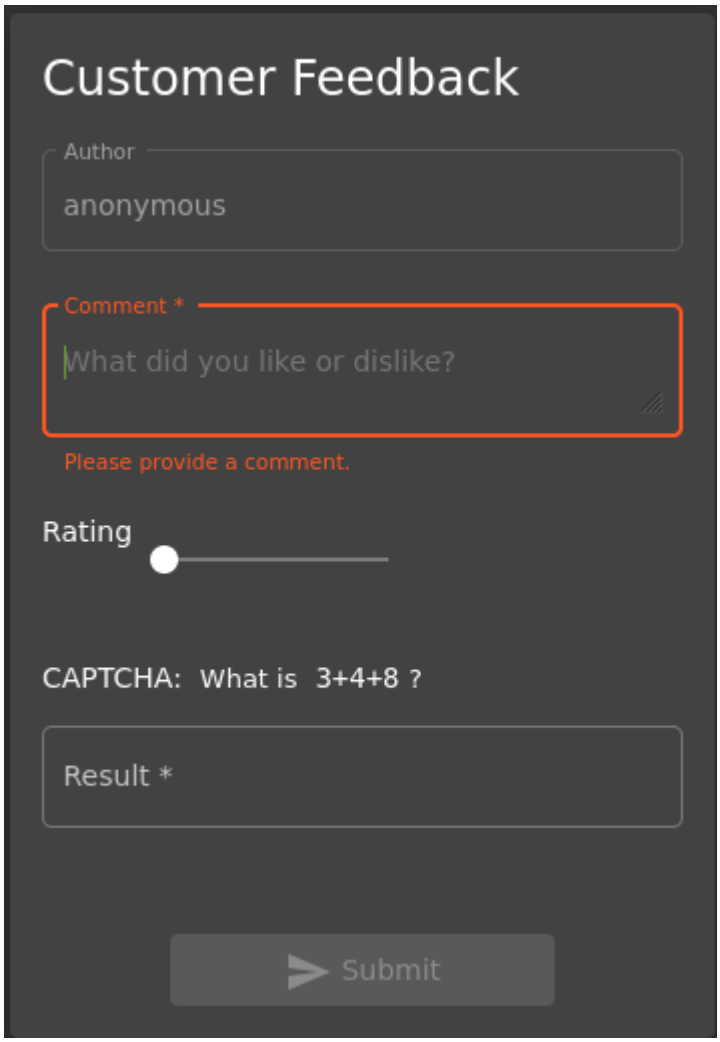


Kuva 19. Kuvakaappaus accounting-sivulta. Muutoksia on tehty tilauksiin ja tuotteiden hintoihin.

5.5 Heikkoudet pääsyoikeuksien hallinnassa

Tähän mennessä on tullut hyvin ilmi, että Juice Shopissa on runsaasti haavoittuvuuksia pääsyoikeuksien hallinnassa. Aikaisemman luvun toteutuksen aikana huomattiin, että järjestelmänvalvojan sivulla näkyy avoimesti kaikki palautelomakkeen kautta lähetetyt viestit, mikä on erityisen kiinnostava tieto, kun huomioidaan heikosti toteutettu pääsyoikeuksien hallinta yhdistettynä mahdollisuuteen lähettää palautetta kirjautumatta sisään.

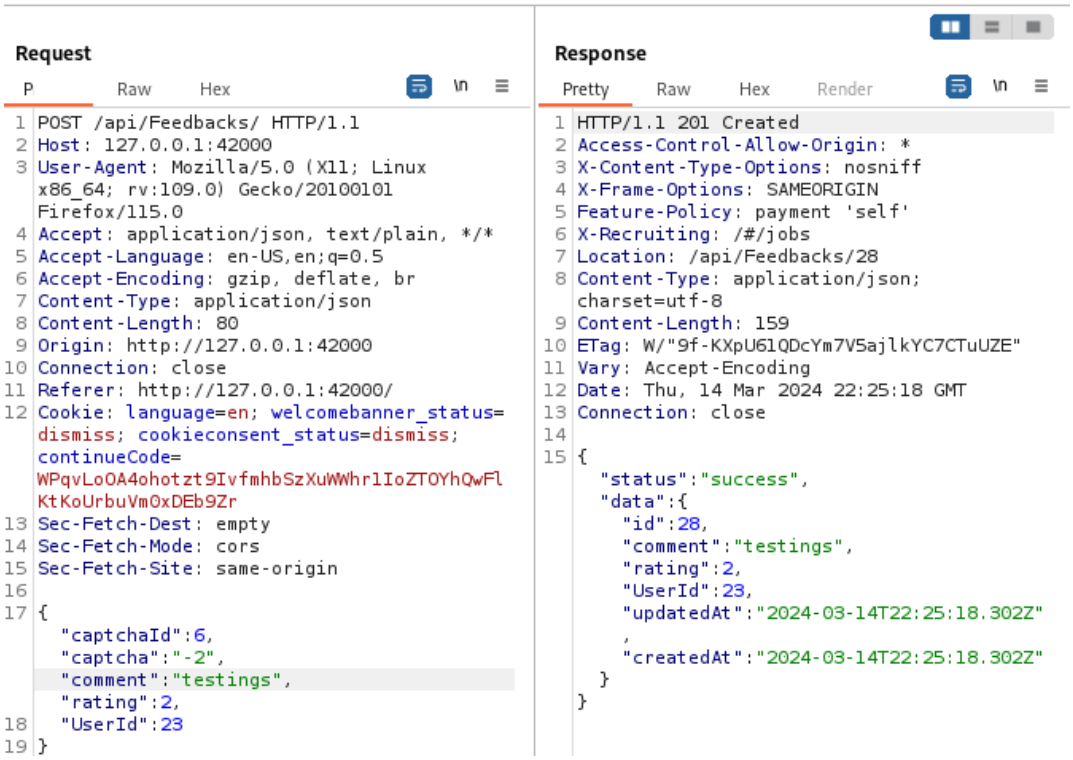
Koska palautelomake vaikuttaa kiinnostavalta toiminnallisuudelta haavoittuvuuksien kannalta, kannattaa sen toimintaan perehtyä tarkemmin ja selvittää, voisiko sitä käyttää hyökkäysvektorina. Kuvassa 20 näkyy asiakaspalautelomake, jonka tekemät pyynnöt tullaan jälleen pysäyttämään Burp Suiten proxyllä.



Kuva 20. Kuvakaappaus Juice Shopin asiakaspalautelomakkeesta.

Ensimmäiset testit suoritettiin kirjautumatta sisään, jolloin huomattiin, että http-vastauksessa arvo on *null* kohdassa *UserId*. Tämä tarkoittaa sitä, että potentiaalisesti pääsyoikeuksienhallinta voidaan ohittaa, jos palvelin ei tarkasta palautteen lähettäjän identiteettiä lähetettyä dataa vastaan. Toisena huomiona ilmaantui se, että CAPTCHA-testi voidaan ohittaa käyttämällä Burp Suiten Repeater-työkalua toistamaan samaa http-pyyntöä. Tämä on virhe automaation estossa.

Toisessa testissä UserId-kenttä populoitiin tunnetulla, olemassa olevan käyttäjän id-numerolla ja havaittiin, että toisen käyttäjän nimissä pystytään lähettämään palautelomakkeella tietoa esteettömästi, mikä näkyy kuvassa 21.



Kuva 21. Kuvakaappaus Burp Suitella tehdystä http-pyynnöstä ja saadusta vastauksesta.

Tehdyt havainnot testien perusteella osoittavat, että palautelomaketta voi olla mahdollista käyttää hyväksi erittäin vahingoittavasti. Johtopäätös perustuu siihen, että lähetetyn datan validoimatta jättäminen ja sen päätyminen järjestelmänvalvojien käyttämälle sivulle tekee palautelomakkeesta erittäin mielenkiintoisen hyökkäysvektorin, jonka potentiaali on hyvä selvittää.

5.6 XSS-hyökkäys palautelomakkeen kautta

Palautelomakkeen kautta lähetetyt tiedot tallennetaan tietokantaan, josta ne haetaan joka kerta, kun järjestelmänvalvojan sivulla vieraillaan. Kun lisäksi otetaan huomioon aikaisemmassa luvussa tehdyt havainnot, on palautelomake hyvin todennäköisesti erinomainen hyökkäysvektori pysyvälle XSS-hyökkäykselle. Myös Juice Shopin pistetaulukon haasteista ilmenee, että tämänlainen haavoittuvuus pitäisi löytyä.

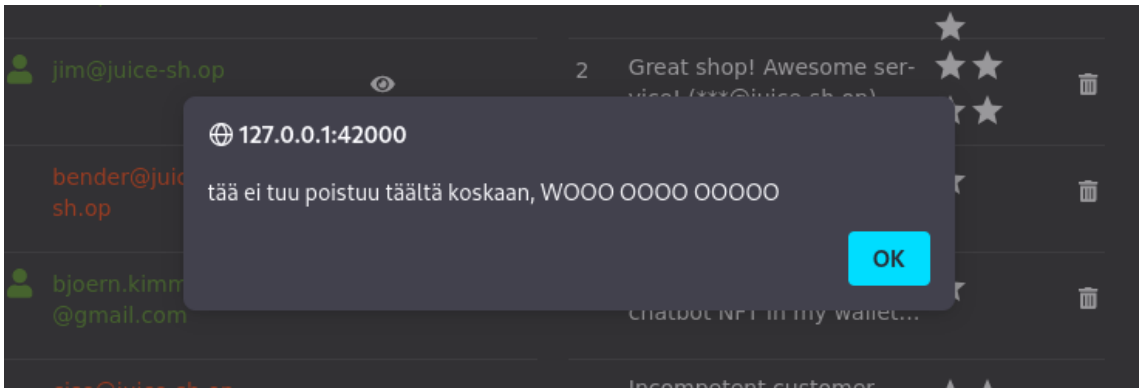
Heti hyökkäyksen alussa ilmeni ongelmia, sillä palvelinpuolella olikin jonkinlainen sanitointi käytössä, minkä vuoksi XSS-hyökkäykset eivät olleet heti toimivia. Palvelimella oleva ohjelma näytti puhdistavan ainakin script-elementit, HTML-elementtien kulmasulkeet sekä kaiken sisällön näiden sisältä. Pysyvän XSS-hyökkäyksen toteuttaminen tässä tapauksessa osoittautui yllättävän haastavaksi ja aikaa toimivan hyökkäyksen löytämiseen kului useita tunteja, joten lähestymiskulma sen suhteen tuskin oli kovin optimaalinen. Hyökkäys kohdistettiin palautelomakkeen kommenttikenttään ja se toteutettiin Burp Suiten Repeater-työkalulla, jolla hyökkäys eteni askel kerrallaan tarkkailemalla palvelimelta saatuja vastauksia ja analysoimalla, kuinka palvelin reagoi eri kulmasulkeiden ja elementtien yhdistelmiin. Luovuutta käytettiin myös siinä, että löydettiin tekijä, joka lopulta laukaisee suoritettavan JavaScript-koodin selaimessa.

Lopulta kuitenkin hyökkäys onnistui koodiesimerkissä 3 käytetyllä hyökkäyksellä, jossa img-elementin alkuun on sisällytetty ylimääräinen img-elementti, joka toimii puskurina palvelimen sanitaatiologiikalle jättäen hyökkäyksen toteuttavan img-elementin huomioimatta. Kuvan lähteeksi on asetettu risuaita, joka aiheuttaa kuvan lataamisessa virheen ja laukaisee onerror-ehdon, joka suorittaa siihen annetun JavaScript-koodin.

```
<<img></img>img src=# onerror=\"javascript:alert('tää ei tuu poistuu
täältä koskaan, WOOO OOOO OOOOO')\">
```

Esimerkkikoodi 3. Onnistunut XSS-hyökkäys.

Vaikka hyökkäyksen toteuttamiseen kului paljon aikaa, olisi oikealle, vaaralliselle hyökkäykselle aika sen arvoista, sillä joka kerta, kun järjestelmänvalvoja vierailee sivulla, koodi suoritetaan. Hyökkäyksellä olisi mahdollista jatkaa esimerkiksi tekemällä skripti, joka varastaa järjestelmänvalvojan istuntoavaimen ja lähettää sen vilpilliselle taholle, jolloin tämä saavuttaisi pääsyn järjestelmänvalvojan käyttäjälle. Lopputuloksena kuvasta 22 ilmenee, kuinka järjestelmänvalvojan sivulla vierailtaessa XSS-hyökkäys käynnistyy ja tervehtii pelottavalla viestillä.



Kuva 22. XSS-hyökkäyksen käynnistämä alert-tapahtuma.

6 Yhteenveto

Tämä insinöörityö oppimisprosessina ja aiheena oli hyvin antoisa, mielenkiintoinen ja hauska. Kokonaisuutena insinöörityön lopputulos vastaa sitä, mitä alussa tehdyissä suunnitelmissa tavoiteltiin. Tavoitteena oli koostaa kattava dokumentti, joka perehdyttää niin laatijan kuin lukijat web-sovellusten tietoturvan alkeisiin ja innostaa myös jatkossa syventymään aiheeseen enemmän.

Työn ensimmäiset kolme lukua ovat tiedon määrältään suuria ja lukujen työstäminen opetti nopeasti, kuinka laaja tämän insinöörityön aihe on. Aiheen käsitellessä näin laajaa aluetta, olisi tämä työ hyvin voinut olla kokonainen kirja. Prosessin alussa haasteena olikin aiheessa pysymisen vaikeus – kuinka tuoda esille oleellisimmat asiat ja mitä yksityiskohtia karsia pois, jotta työn laajuus pysyy sille asetetuissa rajoissa. Paikoin kirjallisen tiedon koostaminen sopiviksi kokonaisuuksiksi oli hyvin puuduttavaa, mikä näkyi ongelmina aikataulutuksen kanssa ja paikoin mekaanisena tekstintuottamisena. Loppujen lopuksi sisällöstä kuitenkin tuli suhteellisen ytimekäs ja kattava osa työtä, ja teorialukujen sisältö antaa hyvät lähtökohdat ymmärtämään viimeisessä käsittelyluvussa tehtyä haavoittuvuuksien testaamista.

Insinöörityön viimeisen käsittelyluvun osuus oli työssä kaikkein antoisin ja opettavaisin, sillä tekemällä käytännönläheistä työtä oppii ja omaksuu sivussa myös teoreettista tietoa hyvin tehokkaasti. Demonstraatioissa pyrittiin toimimaan

järjestelmällisesti seuraamalla löydettyjä havaintoja ja etenemään askel askeleelta eteenpäin. Tämän tarkoituksena oli oppia ja samalla havainnoida, kuinka pienistä haavoittuvuuksien jyväsistä voi eskaloitua sarja suurempia vahinkoja. Työssä olisi voitu vaihtoehtoisesti suorittaa pieniä testejä sattumanvaraisesti, mutta se ei olisi antanut tietoturvariskien vierivän vaikutuksen aiheuttamasta vahingollisuudesta yhtä realistista kuvaa, vaikkakin silloin testejä olisi ollut mahdollista tehdä laajemmin eri kategorioista. Kokonaisuudessaan käsittelyluku oli hyvin mielenkiintoinen, mutta harmillisesti osa sen aikana tehdyistä työvaiheista oli jätettävä tämän raportin ulkopuolelle, jotta jälleen kerran pysyttäisiin työlle asetetuissa rajoissa.

Loppusanoina voidaan todeta, että työn aihe on erittäin laaja, mutta hyvin opettavainen, ja luultavasti tulevaisuudessa jatkuvasti ajankohtaisempi. Kirjoittamisen hetkellä tätä työtä voisi jatkaa sellaisenaan täydentämällä jo opittua ja tehtyä, mutta teknologian kehittyessä ja sen myötä tietoturvaympäristön muuttuessa, on tämäkin työ joskus vanhentunutta tietoa. Tietoturva aiheena on riippuvainen ympäröivästä maailmasta, ja usein aikaisemmin toimineet ratkaisut eivät enää toimi tulevaisuudessa. Positiivisena puolena todettakoon, että aiemman tiedon päälle on aina helpompi rakentaa uutta, joten tähän aiheeseen syventyminen ei ole koskaan ajanhukkaa. Sen voisikin todeta tämän insinöörityön ydinopetukseksi – jatkuvan uuden oppimisen.

Lähteet

1 Awati, Rahul & Wigmore, Ivy. 2023. What is web stack? Verkkoaineisto. TechTarget. <https://www.techtarget.com/whatis/definition/Web-stack>. Luettu 12.2.2023.

2 Adam, John. 2022. Why the right technology stack for your web applications is such a crucial strategic decision and a checklist to get it right. Verkkoaineisto. <https://kruschecompany.com/technology-stack-web-applications/>. Luettu 13.2.2023.

3 What is web application security? Verkkoaineisto. Cloudflare. <https://www.cloudflare.com/en-gb/learning/security/what-is-web-application-security/>. Luettu 12.2.2023.

4 Wu, Hanqing & Zhao, Liz. 2015. Web Security: A WhiteHat Perspective. E-kirja. Auerbach Publishers, Incorporated.

5 Hoffman, Andrew. 2020. Web Application Security. E-kirja. O'Reilly Media, Inc.

6 Jäntti, Mari. 2023. Jopa 500 uhria ilmoittautunut psykoterapiakeskus Vastaamon mahdolliseen käräjäkäsittelyyn – näin poikkeuksellisen suuri istunto järjestetään. Verkkoaineisto. Yle. <https://yle.fi/a/74-20054424>. 11.10.2023. Luettu 22.10.2023.

7 Reconnaissance and Web Application Mapping. Opintomateriaali. Open Distributed European Virtual Campus on ICT Security. Päivitetty 15.2.2017. Luettu 22.10.2023.

8 Frequently Asked Questions about Built With. Verkkoaineisto. Built With. <https://builtwith.com/faq>. Luettu 1.11.2023.

9 About the OWASP Foundation. Verkkoaineisto. OWASP. <https://owasp.org/about/>. Luettu 6.11.2023.

10 Peter Loshin. 2022. What is the Open Web Application Security Project (OWASP)? Verkkoaineisto. TechTarget. <https://www.techtarget.com/searchsoftwarequality/definition/OWASP>. 2022. Luettu 14.11.2023.

11 Notice. 2021. Verkkoaineisto. OWASP. <https://owasp.org/Top10/0x00-notice/>. Luettu 14.11.2023.

12 OWASP Top Ten. 2021. Verkkoaineisto. OWASP. <https://owasp.org/www-project-top-ten/>. Luettu 14.11.2023.

13 How to use the OWASP Top 10 as a standard. 2021. Verkkoaineisto. OWASP. <https://owasp.org/Top10/A00_2021_How_to_use_the_OWASP_Top_10_as_a_standard/>. Luettu 14.11.2023.

14 What is your data collection and analysis process? 2021. Verkkoaineisto. OWASP. <https://owasp.org/Top10/A00_2021_Introduction/#what-is-your-data-collection-and-analysis-process>. Luettu 14.11.2023.

15 Thank you to our data contributors. 2021. Verkkoaineisto. OWASP. <https://owasp.org/Top10/A00_2021_Introduction/#thank-you-to-our-data-contributors>. Luettu 14.11.2023.

16 Data Factors. 2021. Verkkoaineisto. OWASP. <https://owasp.org/Top10/A00_2021_Introduction/#data-factors>. Luettu 14.11.2023.

17 A01:2021 – Broken Access Control. 2021. Verkkoaineisto. OWASP. <https://owasp.org/Top10/A01_2021-Broken_Access_Control/>. Luettu 14.11.2023.

18 A02:2021 – Cryptographic Failures. 2021. Verkkoaineisto. OWASP. <https://owasp.org/Top10/A02_2021-Cryptographic_Failures/>. Luettu 14.11.2023.

19 A03:2021 – Injection. 2021. Verkkoaineisto. OWASP. <https://owasp.org/Top10/A03_2021-Injection/>. Luettu 14.11.2023.

20 A04:2021 – Insecure Design. 2021. Verkkoaineisto. OWASP. <https://owasp.org/Top10/A04_2021-Insecure_Design/>. Luettu 14.11.2023.

21 A05:2021 – Security Misconfiguration. 2021. Verkkoaineisto. OWASP. <https://owasp.org/Top10/A05_2021-Security_Misconfiguration/>. Luettu 14.11.2023.

22 A06:2021 – Vulnerable and Outdated Components. 2021. Verkkoaineisto. OWASP. <https://owasp.org/Top10/A06_2021-Vulnerable_and_Outdated_Components/>. Luettu 14.11.2023.

23 A07:2021 – Identification and Authentication Failures. 2021. Verkkoaineisto. OWASP. <https://owasp.org/Top10/A07_2021-Identification_and_Authentication_Failures/>. Luettu 14.11.2023.

24 A08:2021 – Software and Data Integrity Failures. 2021. Verkkoaineisto. OWASP. <https://owasp.org/Top10/A08_2021-Software_and_Data_Integrity_Failures/>. Luettu 14.11.2023.

25 A09:2021 – Security Logging and Monitoring Failures. 2021. Verkkoaineisto. OWASP. <https://owasp.org/Top10/A09_2021-Security_Logging_and_Monitoring_Failures/>. Luettu 14.11.2023.

26 A10:2021 – Server-Side Request Forgery (SSRF). 2021. Verkkoaineisto. OWASP. <https://owasp.org/Top10/A10_2021-Server-Side_Request_Forgery_%28SSRF%29/>. Luettu 14.11.2023.

27 Clancy, Ryan. 2020. What Is Broken Access Control Vulnerability, and How Can I Prevent It? Verkkoaineisto. <https://www.eccouncil.org/cybersecurity-exchange/web-application-hacking/broken-access-control-vulnerability/>. Luettu 10.2.2023.

28 Surveillance Self-Defence. 2018. What Should I Know About Encryption? Verkkoaineisto. <https://ssd.eff.org/module/what-should-i-know-about-encryption>. Luettu 10.2.2023.

29 Sengupta, Sudip. 2022. OWASP Top 10 Cryptographic Failures A02 – Explained. Verkkoaineisto. <https://crashtest-security.com/owasp-cryptographic-failures/>. Luettu 10.2.2023.

30 Milzarek, René. 2020. Injection Attack Types and How to Best Protect Your Web Apps. Verkkoaineisto. <https://crashtest-security.com/different-injection-attack-types/>. Luettu 11.2.2023.

31 Schraff, Sam. 2021. Verkkoaineisto. OWASP Top 10 – #4 Insecure Design. <https://foresite.com/blog/owasp-top-10-insecure-design/>. Luettu 12.2.2023.

32 Dizdar, Admir. 2022. Security Misconfiguration: Impact, Examples and Prevention. Verkkoaineisto. <https://brightsec.com/blog/security-misconfiguration/>. Luettu 11.2.2023.

33 McDonald, Malcolm. 2020. Web Security for Developers. E-kirja. No Starch Press.

34 Attacking Authentication. Opintomateriaali. Open Distributed European Virtual Campus on ICT Security. Päivitetty 15.2.2017. Luettu 3.1.2024.

35 Guide For Preventing Software & Data Integrity Failure. Verkkoaineisto. Crashtest Security. <https://info.veracode.com/rs/790-ZKW-

291/images/software-data-integrity-failure-prevention-guide-en.pdf>. Luettu 6.2.2024.

36 Guide For Preventing Security Logging And Monitoring Failures Prevention. Verkkoaineisto. Crashtest Security. <https://info.veracode.com/rs/790-ZKW-291/images/security-logging-and-monitoring-prevention-guide-en.pdf>. Luettu 6.2.2024.

37 Server-side request forgery (SSRF). Verkkoaineisto. Port Swigger. <https://portswigger.net/web-security/ssrf>. Luettu 8.2.2024.

38 Muscat, Ian. 3.2.2022. What is server-side request forgery (SSRF)? Verkkoaineisto. Acunetix. <https://www.acunetix.com/blog/articles/server-side-request-forgery-vulnerability/>. Luettu 8.2.2024.

39 Server-Side Request Forgery Prevention Cheat Sheet. Verkkoaineisto. OWASP. <https://cheatsheetseries.owasp.org/cheatsheets/Server_Side_Request_Forgery_Prevention_Cheat_Sheet.html>. Luettu 8.2.2024.

40 Penetration Testing Standards. Opintomateriaali. Open Distributed European Virtual Campus on ICT Security. Päivitetty 15.2.2017. Luettu 13.2.2024.

41 What is penetration testing? Verkkoaineisto. IBM. <https://www.ibm.com/topics/penetration-testing>. Luettu 13.2.2024.

42 Fort, Julio. 8.5.2023. What is automated penetration testing? Hint – Not a pentest. Verkkoaineisto. Blaze Information Security. <https://www.blazeinfosec.com/post/automated-penetration-testing/>. Luettu 13.2.2024.

43 Weidman, Georgia. 2014. Penetration Testing. E-kirja. No Starch Press.