Selainmoninpelien tietoturva ja huijauksenesto, esimerkkinä Laivanupotus

Miro Laukkanen

Haaga-Helia ammattikorkeakoulu

IT-tradenomi

Opinnäytetyö

2025

Tiivistelmä

Tekijä(t)
Miro Laukkanen

Tutkinto
Tietojenkäsittely, päivä

Raportin/Opinnäytetyön nimi
Selainmoninpelien tietoturva ja huijauksenesto, esimerkkinä Laivanupotus

Sivu- ja liitesivumäärä
55

Opinnäytetyön taustalla oli halu toteuttaa kaksin pelattava selainpohjainen Laivanupotus-peli hyödyntäen moderneja web-teknologioita. Pelin kehityksen myötä kävi ilmi, kuinka monimutkaista ja aikaa vievää on suunnitella tietoturvallinen ja huijauksenkestävä selainmoninpeli, vaikka itse pelimekaniikka olisi yksinkertainen. Tästä syntyi opinnäytetyön varsinainen aihe: tutkia selainmoninpelien tietoturvaa ja huijauksenestoa käyttäen esimerkkinä itse toteutettua Laivanupotus-peliä.

Työn tavoitteena oli selvittää, miten selainpohjaisen moninpelin tietoturva ja huijauksenesto voidaan suunnitella, toteuttaa, testata ja parantaa. Erityisesti tarkasteltiin peliviestinnän manipulointia, käyttäjien autentikointia ja pelilogiikan reiluuden varmistamista. Työssä haluttiin myös osoittaa, kuinka tietoturvatestausta voi käytännössä soveltaa omaan kehityksessä olevaan peliin.

Tietoturvatestaus toteutettiin useilla eri menetelmillä, kuten penetraatiotestauksella, input fuzzingilla, JWT-tokenien manipulointitestauksella sekä asiakassovelluksen ja rajapintojen analyysilla. Testaus suoritettiin paikallisessa kehitysympäristössä Burp Suite -työkalun avulla. Käytännön testauksessa havaittiin useita puutteita muun muassa autentikoinnin tarkkuudessa ja peliviestien käsittelyssä. Tulokset dokumentoitiin yksityiskohtaisesti ja niiden pohjalta esitettiin teknisiä korjausehdotuksia, kuten JWT-tokenien tarkemman validoinnin lisääminen ja virheilmoitusten selkeyttäminen.

Lopuksi pohdittiin, miten testausmenetelmiä voidaan soveltaa muihin selainmoninpeleihin ja mitä työn aikana opittiin tietoturvasta osana pelikehitystä. Työ osoittaa, että tietoturva ei ole pelkästään isojen sovellusten ongelma, vaan myös pienempien kehittäjien luomat verkkopelit voivat kohdata vakavia uhkia, ellei turvallisuuteen kiinnitetä riittävää huomiota jo kehitysvaiheessa.

Asiasanat
Tietoturva, Videopeli, WebSocket, Penetraatiotestaus

Sisällys

1 Johdanto

1.1 Projektin tausta

Selaimessa pelattavat videopelit eivät nauti nykyään samankaltaista suosiota kuin 1990, 2000 ja 2010- luvuilla, muiden pelialustojen yleistyessä pelaajien keskuudessa. Monet peleistä olivat yksinpelejä mutta moninpelit alkoivat yleistymään viimeistään 2000-luvun taitteessa nettiyhteyksien parantuessa. Pelasin selainpelejä paljon myös lapsena ja halusin nyt aikuisena ja osaavana koodarina kokeilla sellaisen toteuttamista modernin web-kehityksen keinoin.

Päätin toteuttaa kahden ihmisen selaimessa pelattavan laivanupotus-pelin. Päädyin laivanupotukseen sillä se vaikutti minulle sopivalta haasteelta. Halusin oppia lisää pelilogiikan luomisesta, käyttäjän- ja pelisession hallinnasta sekä reaaliaikaisesta vesiliikenteestä, ja laivanupotus piti kaikkia näitä sisällään.

Tätä laivanupotusta kehittäessäni huomasin kuinka monimutkaista ja aikaa vievää voi olla luoda tietoturvallinen sekä huijaamiselta vahvasti suojattu selainmoninpeli, vaikka itse peli olisi suhteellisen simppeli. Täten opinnäytetyön aiheekseni valiutui verkkoselainmoninpelien tietoturvan sekä huijauksenestonon testaaminen, jossa käytetään case-esimerkkinä luomaani laivanupotusta.

Verkkoselainpelit ovat usein pienempien tiimien tai yksityishenkilöiden julkaisemia projekteja ja joskin amatöörimäisempiä ja monesti tehty vain huvin vuoksi. Tälläisen hupituotoksen-tietoturvan toteutus ja testaaminen voi tuntua turhalta, mutta se on kuitenkin tärkeää ja ajankohtaista. Yksi moderneista ohjelmistokehityksen trendeistä on ”Vibe-coding”-, jossa amatöörimäisimmätkin ohjelmistokehittäjät pystyvät julkaisemaan web-sovelluksia ja verkkopelejä, jotka on luotu täysin generatiivisen tekoälyn malleilla, niille syötettyjen kehotteiden avulla. Nämä pelit ja sovellukset silti kohtaavat samoja tietoturvauhkia kuin muutkin verkkosivut. Niiden luoja usein kuitenkin hyväksyy generatiivisen tekoälyn tuotoksen mukisematta, mikä voi johtaa vakaviin tietoturvauhkiin tai mahdollisiin porsaanreikiin selainpelin huijauksenestossa.

Mikäli verkkosivu sitä vaatii, loppukäyttäjä käyttää usein itselleen tuttua sähköpostia ja salasanaa, varsinkin mitä vähemmän tärkeämpi verkkosivu on, säästyäkseen muistamisen vaivalta. Tämä pätee myös verkkosivuihin ja -peleihin, joita käytetään vain huvin vuoksi, mutta niidenkin tapauksessa niiden ylläpitäjä on vastuussa tietoturvasta.

1.2 Projektin tavoite

Tämän opinnäytetyön tavoitteena on selvittää, kuinka selainpohjaisen moninpelin tietoturva ja huijauksenesto voidaan toteuttaa, kuinka sitä voidaan arvioida taikka testata sekä parantaa. Lisäksi tarkastellaan kuinka tietoturvatestausta voi tehdä käytännössä omaa selainmoninpeliää vastaan kehitysympäristössä ja mitä ohjelmistoja siihen voi käyttää.

Työn tietoperustassa keskitytään erityisesti niihin teknologioihin ja arkkitehtuuriratkaisuihin, jotka liittyvät verkkoviestintään ja käyttäjien todennukseen, kuten JWT-tunnisteisiin ja WebSocket-pohjaiseen STOMP-protokollaan. Nämä teknologiat ovat läsnä useissa selaimessa pelattavissa moninpeleissä ja niissä ilmenevät monet tietoturvan haavoittuvuuksista. Tavoitteena on tunnistaa laivanupotuksen mahdollisia tietoturvaheikkouksia sekä pelin huijauksenestoon virheitä. Näitä on tarkoitus havainnollistaa, analysoimalla miten ne ilmenevät, ja miten ne voidaan minimoida, korjata tai estää. Empiirisessä osuudessa esitellään käytännön tietoturvan testausmenetelmiä laivanupotusta vastaan ja niillä saatuja tuloksia, jotka ohjaavat pelin tietoturvan jatkokehittämistä.

Opinnäytetyö toimii samalla ohjenuorana muille verkkokehittäjille, jotka miettivät, miten ja miksi verkkoselainpeliä tai muuta interaktiivista web-sovellusta kannattaa tarkastella tietoturvan näkökulmasta. Työssä esitetyt tekniset esimerkit ja testauskäytännöt tarjoavat konkreettista tukea turvallisemman ohjelmistokehityksen toteuttamiseen.

1.3 Projektin rajaukset

Tässä opinnäytetyössä keskitytään laivanupotus-verkkopelin huijaamisenestoon ja tietoturvaan liittyviin keskeisiin osiin. Sovelluksen lähdekoodia, toteutusta ja teknologioita käsitellään vain siltä osin kuin ne ovat olennaisia pelin tietoturvan arvioimiseksi. Muut toiminnallisuudet ja niiden tekniset ratkaisut jäävät työn ulkopuolelle.

Pelaajien tunnistautumista tarkastellaan tietoturvan näkökulmasta, mutta työ ei syvenny autentikointiprotokollien tai ulkopuolisten kirjautumisratkaisujen tarkkaan vertailuun tai tekniseen analyysiin, sillä näiden kattava käsittely ylittäisi työn laajuuden.

Työssä käsitellään tietoturvauhkia joihin pelin ohjelmoija voi realistisesti vaikuttaa koodissaan kehitysympäristössä. Palvelinpuolen viitekehykset web-ohjelmistoille sekä web-sovellusten julkaisualustat tarjoavat usein automaattisesti suojan yleisimmille tietoturvauhille. Tästä esimerkkinä laajamittaiset verkkohyökkäykset, kuten palvelunestohyökkäykset, rajataan työn ulkopuolelle. Sen sijaan tarkastelun kohteena ovat sovellukseen kohdistuvat pelikohtaiset tietoturvauhat, kuten

viestiliikenteen manipulointi, pelilogiikan väärinkäyttö ja autentikoinnin kiertäminen. Tietoturvatestauksessa keskitytään niihin menetelmiin ja uhkiin, jotka ovat merkityksellisiä juuri tämän sovelluksen toimintaympäristössä.

1.4 Keskeiset käsitteet

WebSocket on tietokoneiden viestintäprotokolla, joka tarjoaa samanaikaisen kaksisuuntaisen kommunikaatiokanavan.

Käyttöliittymä on se osa sovellusta, joka mahdollistaa vuorovaikutuksen sovelluksen ja käyttäjän välillä.

Pelaajan tunnistaminen on prosessi, jossa varmistetaan käyttäjän identiteetti ja oikeus käyttää palvelua. Verkkopeliympäristössä tunnistaminen toimii tietoturvan kulmakivenä estäen luvattoman pääsyn ja varmistaen, että ainoastaan autentikoidut käyttäjät voivat osallistua peliin. Tunnistaminen sisältää tyypillisesti rekisteröitymisen, kirjautumisen ja istunnonhallinnan mekanismit, jotka suojaavat pelikokemusta.

Asiakassovellus on käyttäjän selaimessa toimiva ohjelmisto, joka vastaa käyttöliittymän esittämisestä ja käyttäjän syötteiden käsittelystä sekä lähettämisestä palvelimelle.

Palvelin on verkkopalvelun taustalla toimiva ohjelmistokomponentti, joka käsittelee asiakassovelluksilta saapuvat pyynnöt, suorittaa keskitetyn pelilogiikan ja hallinnoi pelin kokonaistilaa.

Viestiliikenne tarkoittaa tiedonsiirtoa asiakassovelluksen ja palvelimen välillä.

Laivanupotus on lautapeli, jossa kaksi pelaajaa asettaa eri mittaisia laivoja omille ruudukoillensa toisen näkemättä. Pelaajat ”ampuvat” vuorotellen vastustajan ruudulle arvaamalla koordinaatteja. Tavoitteena on upottaa vastustajan laivasto.

Spring Boot avoimen lähdekoodin Java-viitekehys web-sovellusten nopeaan luontiin ja käyttöönottoon.

React Facebookin luoneen yhtiön Meta:n ylläpitämä käyttöliittymän kehitykseen tarkoitettu ohjelmistokirjasto.

2 Tietoperusta

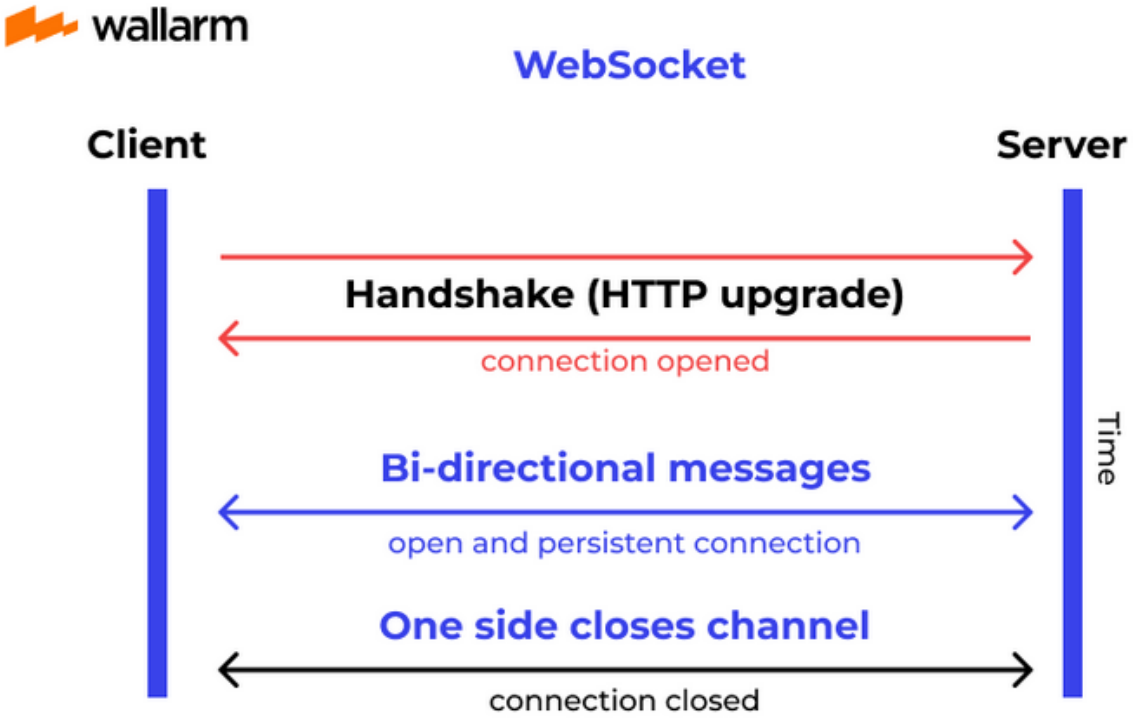
Tässä luvussa käydään läpi toteutetun laivanupotus-pelin kannalta keskeiset teknologiat ja käsitteet, jotka liittyvät pelin verkkoviestintään ja käyttäjän autentikointiin. Lisäksi käydään läpi mitä eroa on huijauksenestolla ja tietoturvalla tässä kontekstissa sekä yleisimpiä selainmoninpelien tietoturvauhkia sekä niitä vastaan olemassa olevia tietoturvamenetelmiä.

2.1 Websocket-protokollan perusteet ja toiminta

Websocket-protokolloa soveltuu erityisen hyvin käyttökohteisiin, joissa vaaditaan jatkuvaa ja viiveetöntä viestintää, kuten verkkopelien, chat-palveluiden ja yhteistyöalustojen kaltaisissa sovelluksissa. (Ably, 2024) Tämän vuoksi se on reaaliaikaisuudellaan oleellinen osa laivanupotuksen toteutusta.

WebSocket-protokolla on suunniteltu ratkaisemaan perinteisten HTTP-pyyntöihin perustuvien ratkaisujen rajoitteita, erityisesti reaaliaikaisessa tiedonsiirrossa. Toisin kuin tilaton HTTP-protokolla, WebSocket mahdollistaa pysyvän, kaksisuuntaisen yhteyden palvelimen ja asiakassovelluksen (client) välillä yhden TCP-yhteyden yli. (Fette & Melnikov. 2011).

Yhteys muodostetaan aluksi HTTP:n kautta niin sanotulla handshake-prosessilla, jonka jälkeen yhteys päivitetään WebSocket-protokollaksi. Kun yhteys on luotu, molemmat osapuolet voivat lähettää viestejä toisilleen ilman erillisiä pyyntöjä. Tämä vähentää verkkoliikenteen määrää ja parantaa suorituskykyä erityisesti tilanteissa, joissa viestien täytyy kulkea tiheästi tai jatkuvasti. (Wallarm. 2025)



Kuva 1. Websocket yhteyden elinkaari (Wallarm 2025)

2.2 Websocket protokollan hyödyt ja haitat

Laivanupotuksen vaatimaan reaaliaikaiseen tiedonsiirtoon on useita toteutusvaihtoehtoja (esim. http-long pollin) ja Websocket-valikoitui toteutustavaksi useasta eri syystä.

WebSocket-protokolla tarjoaa merkittäviä teknisiä ja toiminnallisia etuja verrattuna perinteisiin HTTP-pohjaisiin tiedonsiirtoratkaisuihin. Sen suurimmat hyödyt liittyvät viiveettömyyteen, alhaiseen latenssiin ja kaksisuuntaiseen kommunikointimalliin. Protokollan mahdollistama jatkuva yhteys vähentää tarvetta toistuville pyyntö-vastaus-sykleille, mikä parantaa sekä suorituskykyä että käyttökokemusta (Stack overflow, 2017).

WebSocketin **full-duplex**-ominaisuus – eli kyky lähettää ja vastaanottaa tietoa samanaikaisesti – tukee interaktiivisia sovelluksia tehokkaasti. Tämä arkkitehtuurimalli on selvästi tehokkaampi ja kevyempi vaihtoehto perinteiselle polling- tai long-polling-menetelmälle, jossa viive ja kuormitus kasvavat käyttäjämäärän mukaan (GeeksForGeeks, 2024).

WebSocket-protokollan käyttöön liittyy kuitenkin myös rajoitteita ja haasteita. Palvelimen on kyettävä ylläpitämään useita rinnakkaisia pitkäkestoisia yhteyksiä, mikä voi johtaa suorituskykyongelmiin erityisesti korkeassa kuormituksessa. Lisäksi yhteyden katkeaminen edellyttää erityistä logiikkaa sen uudelleenmuodostamiseen, jotta käyttäjä ei menetä pelikokemusta. (Ably, 2024)

Lisäksi turvallisuuden näkökulmasta WebSocket-protokolla ei sisällä omaa sisäänrakennettua salausmenetelmää. Tämän vuoksi sen suojattua versiota, **WSS-protokollaa** (WebSocket Secure), tulee käyttää aina, kun siirretään arkaluontoista tai pelitilaan liittyvää tietoa. WSS hyödyntää **TLS (Transport Layer Security)** -salausta samalla tavoin kuin HTTPS, suojaten viestiliikenteen ulkopuolisilta tarkkailijoilta ja välimieshyökkäyksiltä (IETF, 2011; OWASP, 2018).

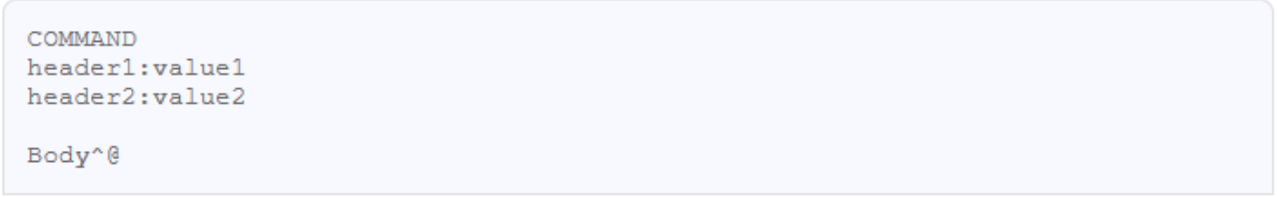
2.3 STOMP (Simple Text Oriented Messaging Protocol)

Monet WebSocket-kirjastot hyödyntävät viestinnässä STOMP-protokollaa ja niin on myös Laivanupotuksen tapauksessa.

STOMP on yksinkertainen (suoratoistoon soveltuva) tekstipohjainen viestinvälitysprotokolla. Se tarjoaa yhteensopivan tiedonsiirtoformaatin, jonka ansiosta STOMP-asiakassovellukset voivat viestiä minkä tahansa STOMP-yhteensopivan viestinvälityspalvelimen kanssa. Tämä mahdollistaa viestien helpon ja laajasti yhteensopivan välityksen eri ohjelmointikielten, alustojen ja välityspalvelinten välillä. (STOMP, 2012 a).

Kuten tietoperustan kappaleessa 2.2 Websocket protokollan hyödyt ja haitat mainittiin, on Websocket-protokolla soveltuva reaaliaikaiseen tiedonsiirtoon eli suoratoistoon. STOMP-protokolla täten sopii hyvin WebSocketin kanssa työskentelyyn.

STOMP perustuu kevyisiin teksti viesteihin, jotka koostuvat komennoista (kuten SEND, SUBSCRIBE, MESSAGE), otsikkokentistä (header) ja viestirungosta (Body) jos komento on SEND, MESSAGE tai ERROR. Tätä kokonaisuuta kutsutaan nimellä ”frame”. (STOMP, 2012 b).



Kuva 2. Framen mahd. koostumus (STOMP, 2012 b)

```
>>> CONNECT
Authorization:Bearer eyJhbGciOiJIUzUxMiJ9.eyJ
accept-version:1.2,1.1,1.0
heart-beat:4000,4000
```

Kuva 3. kuvankaappaus STOMP-viestistä selaimen konsolissa

Kuvassa 3 STOMP-viestin komento on CONNECT, ja sillä on kolme headeria eli otsikkoa.

Authorization: sisältää autentikoinnissa käytettävän JWT-tokenin.

accept-version: kertoo palvelimelle hyväksyttävät STOMP-protokollan versiot.

heart-beat: kertoo palvelimelle, kuinka monen millisekunnin välein lähetetään ”ping”-viesti, joka testaa onko yhteys vielä kunnossa.

Yksi STOMP:in keskeisistä ominaisuuksista myös on aihe-/tilaajapohjainen (pub/sub) viestintämalli, jossa viestit kohdennetaan tiettyihin aiheisiin (topics), ja vain aiheeseen tilanneet vastaanottavat ne. Tämä lähestymistapa mahdollistaa tehokkaan jaottelun ja skaalautuvuuden, kun useat eri asiakkaat voivat käsitellä rinnakkaisia viestivirtoja ilman häiriöitä. STOMP-viestien mukana voidaan myös kuljettaa metatietoa otsikkokentissä, kuten autentikointitietoja (JWT-tunniste), mikä mahdollistaa turvallisemman ja konteksisidonnaisen viestinvälityksen.

```
>>> SUBSCRIBE
id:sub-0
destination:/topic/game/1

>>> SUBSCRIBE
id:sub-1
destination:/topic/game/1/move

>>> SUBSCRIBE
id:sub-2
destination:/topic/game/1/connections
```

Kuva 4. Esimerkki erilaisista sub-viesteistä, kuvankaappaus selaimen konsolista.

2.4 JWT-autentikaatio selainverkkopeleissä

JWT eli JSON Web Token (myöhemmin JWT-tunniste) on olennainen osa toteutuneen laivanupotuksen tunnistautumista ja istunnon hallintaa. Se on osa myös laivanupotuksen tietoturvaa ja huijauksenestoa ja sen toimintaperiaatteiden ymmärtäminen on tärkeää työn kokonaisuuden kannalta.

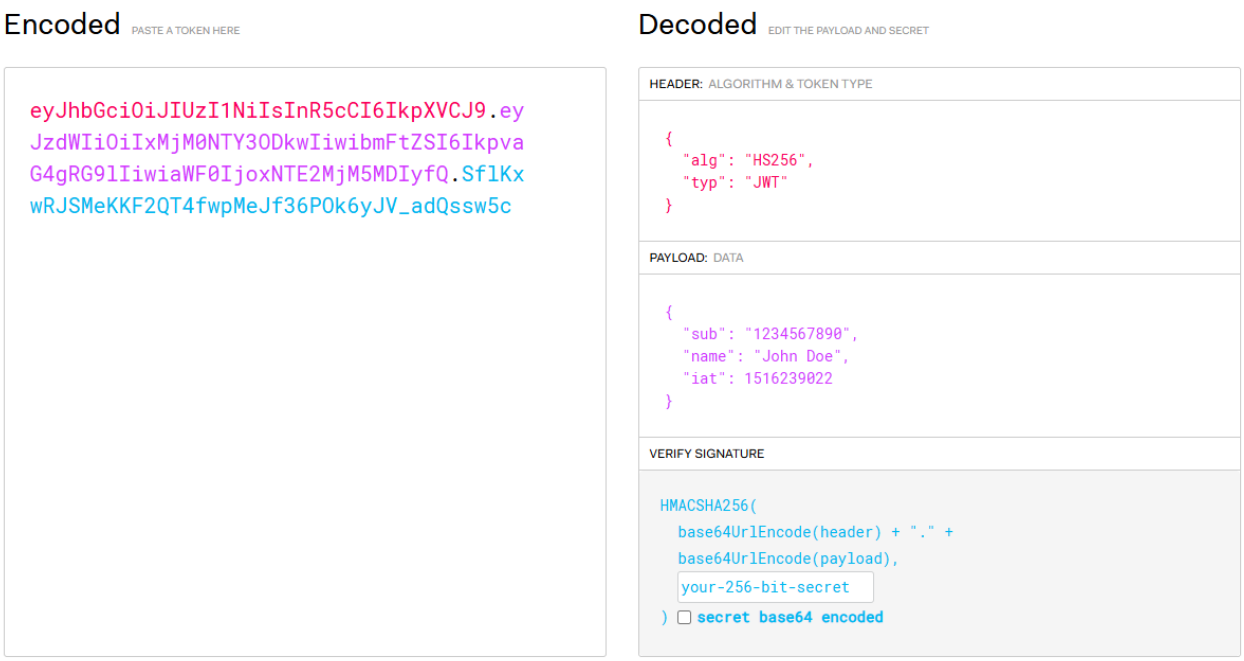
JWT (JSON Web Token) on avoin standardi, joka tarjoaa kompaktin ja turvallisen tavan välittää osapuolten välillä tietoa JSON-objektina. JWT-tokenit koostuvat kolmesta osasta: headerista, payloadista ja allekirjoituksesta, mikä tekee niistä digitaalisesti varmennettavia ja vaikeita väärentää. Tämä ominaisuus on erityisen hyödyllinen web-sovelluksissa, joissa käyttäjien tunnistaminen ja viestiliikenteen autentikointi on kriittistä (Auth0, s.a.).

JWT-tunniste koostuu kolmesta osasta, Header (otsikko), Payload (sisältö) ja Signature (allekirjoitus)

JWT:n otsikko sisältää tyypillisesti kaksi kenttää: tunnisteen tyyppi (esimerkiksi JWT) sekä allekirjoitusalgoritmi, kuten HMAC, SHA256 tai RSA. Näiden avulla palvelin tietää, miten tunniste on rakennettu ja miten sen eheys voidaan tarkistaa (Serban 11.5.2025).

Payload-osaan sisältyy väitteitä (claims), jotka voivat koskea esimerkiksi käyttäjää tai muuta entiteettiä. Väitteet voivat olla vakiomuotoisia, julkisia tai yksityisiä, ja niillä voidaan ilmaista esimerkiksi käyttöoikeuksia, käyttäjärooleja tai istunnon voimassaoloa. JWT:tä käytetään usein myös niin, että käyttäjän onnistuneen kirjautumisen jälkeen palvelin lähettää tokenin, jonka asiakas liittää myöhemmin pyyntöihin Authorization-headerissa. Näin tunnistautuminen onnistuu ilman erillistä istunnonhallintaa (Serban 11.5.2025).

JWT:n eheys ja aitous varmistetaan allekirjoituksella, joka luodaan yhdistämällä otsikko- ja sisältöosat sekä allekirjoittamalla ne valitulla algoritmilla ja salaisella avaimella. Näin voidaan varmistaa, ettei tunnistetta ole muokattu ja että se on peräisin luotetusta lähteestä (Serban, 2023).



Kuva 5. Kuvankaappaus JWT-tokenin purkuun tarkoitetulta web-sivulta. (JWT.IO, 2025)

Kuvassa 5. vasemmalla on enkoodattu JWT-tunniste ja oikealla puretun tokenin sisältö-

Kuvan 5. Tokenin ”Payload”-koostuu kuvassa kolmesta osasta:

”sub” eli subject eli aihe. Uniikki tunniste entiteetille mitä JWT-edustaa

”name” eli käyttäjä, jonka Token yksilöi.

”lat” eli issued at eli aikaleima tokenin luontihetkestä millisekunteina.

JSON Web Tokenien (JWT) turvallisessa käytössä tulee kiinnittää huomiota useisiin seikkoihin. Salaisuuksien tulee olla pitkiä, satunnaisia ja turvallisesti säilytettyjä, erityisesti käytettäessä symmetrisiä algoritmeja kuten HMAC SHA256. Tokenin voimassaolo kannattaa rajoittaa määrittelemällä sille vanhentumisaika, mikä pienentää väärinkäytön riskiä. Tokenin sisällöstä on lisäksi syytä jättää pois arkaluontoinen tieto, sillä kuka tahansa tokenin saanut voi sen sisällön purkaa. Tokenien aitous ja käyttötarkoitus on varmistettava tarkistamalla esimerkiksi allekirjoitus sekä olennaisten kenttien tiedot. Koska JWT on tilaton, on suositeltavaa käyttää esimerkiksi mustia listoja tai lyhyitä voimassaoloaikoja mahdollisten mitätöintitilanteiden hallintaan. (Serban 11.5.2025)

2.5 Selainverkkopelien huijauksenesto ja tietoturva

Tietoturvalla tarkoitetaan tärkeän tiedon suojaamista luvattomalta käytöltä, paljastumiselta, väärinkäytöltä, muokkaamiselta tai häirinnältä. Sen tavoitteena on varmistaa, että luottamuksellinen tieto on saatavilla vain valtuutetuille käyttäjille, säilyy muuttumattomana ja on käytettävissä silloin kun sitä tarvitaan (IBM, 2024).

Videopelissä huijaaminen tarkoittaa etulyöntiaseman saamista tavalla, jota ei ole tarkoitettu pelin normaalin kulun osaksi. Yksinpeleissä huijaamisen tarkoituksena on usein helpottaa pelaamista, kun taas moninpeleissa se antaa pelaajalle epäoikeudenmukaisen kilpailuedun muihin pelaajiin nähden (NCES, 2024).

Näin ollen tietoturva ja huijauksenesto jakavat useita yhteisiä tavoitteita: molemmat suojaavat järjestelmää väärinkäytöksiltä, ja hyvin toteutettu tietoturva tukee myös huijauksenestoa. Esimerkiksi palvelinpuolen autentikointi, syötteiden validointi ja viestien salaus paitsi suojaavat järjestelmää tietomurroilta, myös estävät epäreilua pelaamista.

Videopeleille, jotka ovat suoraan mallinnettu olemassa olevasta kortti- tai lautapelistä voi huijauksenestoon toteuttaminen olla todella suoraviivaista. Fyysisen pelin säännöt täytyy vain mallata videopelin pelilogiikkaan ja tämä logiikka siirtää tietoturvallisen palvelimen hallintaan.

2.6 Selainverkkopelien tietoturvahaasteet

Selainmoninpelit, kuten toteutettu Laivanupotus, ovat pohjimmiltaan web-sovelluksia, jotka kohtaavat samoja tietoturvahaasteita kuin muutkin verkkopalvelut. Web-sovellusten tietoturvahaasteita on kuitenkin valtava määrä, eikä ole tarkoituksenmukaista tai mahdollista käydä niistä jokaista läpi tämän opinnäytetyön puitteissa. Tässä luvussa kuvataankin yleisellä tasolla keskeisiä web-sovelluksiin kohdistuvia tietoturvahaasteita OWASP Web Service Security Cheat Sheetin (OWASP, 2020) pohjalta, joka on tunnustetun tietoturvaorganisaation laatima ohjeistus nimenomaan web-palveluiden suojaamiseen.

Esitetyt kategoriat ovat yleisesti sovellettavissa useisiin eri arkkitehtuureihin ja teknologioihin. Tarkempi soveltaminen Laivanupotus-pelin kontekstiin käsitellään empiirisessä osassa.

2.6.1 Kuljetuksen luottamuksellisuus ja enkoodaus (Transport Confidentiality & Encoding)

Kuljetuksen luottamuksellisuus (Transport Confidentiality) suojaa verkkopalveluiden välistä viestintää salakuuntelulta ja man-in-the-middle-hyökkäyksiltä.

Sääntö: Kaikki viestintä, joka liittyy sensitiivisiä toimintoja sisältäviin verkkopalveluihin, todennettuihin istuntoihin tai arkaluonteisten tietojen siirtoon, on salattava hyvin konfiguroidulla TLS-yhteydellä. Tämä suositus koskee viestintää myös silloin, kun itse viestit ovat erikseen salattuja, sillä TLS tarjoaa viestin kuljetuksen suojauksen lisäksi useita muita etuja, kuten tiedon eheyden varmistamisen, toistojenkestävyyden sekä palvelimen todennuksen. (OWASP, 2020)

Enkoodaus (Transport Encoding) puolestaan viittaa tapaan, jolla data muunnetaan tiedonsiirtoa varten. Yleensä tällä tarkoitetaan http-protokollan ”Transfer Encoding”-otsikkoa eli headeria.

Transfer-Encoding tukee useita menetelmiä, kuten "chunked", "compress", "deflate" ja "gzip", joista erityisesti "chunked" mahdollistaa datan lähettämisen osissa ilman tarvetta määritellä sisällön kokonaispituutta etukäteen (IETF 2014). Epäyhtenäiset tulkinnat näistä koodausmenetelmistä voivat altistaa järjestelmät hyökkäyksille, kuten HTTP Request Smuggling -tekniikalle, jossa hyökkääjä hyödyntää tulkintaeroja välityspalvelinten välillä (OWASP 2020).

2.6.2 Palvelimen autentikointi (Serve Authentication)

Palvelimen aitous on varmennettava asiakassovelluksen (esim. selaimen) toimesta. Tämä tarkoittaa sitä, että sivustolla käytetty TLS-sertifikaatti on luotetun tahon allekirjoittama, voimassa oleva ja vastaa palvelimen osoitetta. Tämä suojaa käyttäjää väärennetyiltä palvelimilta. (OWASP, 2020)

2.6.3 Käyttäjän autentikointi (User authentication)

Web-sovelluksen tulee varmistaa käyttäjän identiteetti turvallisesti. OWASP ei suosittele Basic Authenticationin käyttöä, sillä se paljastaa salaista tietoa http-pyynnön otsikossa (headerissa). Sen sijaan OWASP kehottaa hyödyntämään vahvempia ratkaisuja, kuten JWT-tunnisteet, OAuth tai Mutual TLS. Autentikoinnin on aina tapahduttava salatun yhteyden yli, esim. TLS. (OWASP, 2020)

2.6.4 Viestien eheys ja salaus (Message Integrity & Confidentiality)

TLS suojaa viestien eheyttä tiedonsiirron aikana, mutta joissain tapauksissa voidaan tarvita viestikohtaisia allekirjoituksia tai salausta (esim. XML-allekirjoitukset tai JSON Web Signature) viestin lähettäjän identiteetin varmistamiseksi. Tämä on tärkeää, jos viestit säilytetään salattuna tai kuljetetaan useiden palveluiden läpi. (OWASP, 2020)

2.6.5 Auktorisointi ja käyttöoikeudet

Käyttäjän tai järjestelmän tulee saada käyttöoikeus vain niihin toimintoihin, joihin sillä on lupa. Tämä tarkoittaa roolipohjaista pääsynhallintaa, resurssien rajaamista ja eriyttämistä esim. hallinnollisiin toimintoihin. Tarkistukset tulee suorittaa jokaisella pyynnöllä. (OWASP, 2020)

2.6.6 Skeema- ja sisältövalidointi (Schema & Content Validation)

OWASP painottaa, että kaikki syötteet on validoitava tarkasti. Esimerkiksi XML-tiedostoja hyödyntävässä sovelluksessa on tärkeää varmistaa, että viestit vastaavat skeemaa (XSD) jossa määritellään sisällön maksimi- sekä minimipituus sekä jokaiselle parametrille sallittu merkistö. Tämän lisäksi sen tulisi varmistaa, että sisällöt ovat oikein muotoiltuja. OWASP painottaa myös puutteellisen validoinnin mahdollistamia JSON/XML-pommien ja entiteettiliitosten uhkia sekä suosittelee pitämään listaa sallituista URL:stä (OWASP, 2020)

2.6.7 Ulostulon koodaus (Output Encoding)

Web-palvelujen tuloste, joka lähetetään asiakassovellukselle, on koodattava asianmukaisesti datana, eikä ohjelmakoodina. Tämä estää Cross-Site Scripting (XSS)- hyökkäykset. Tämä koskee erityisesti tilanteita, joissa syöte tai palaute renderöidään käyttöliittymässä, sillä usein käyttöliittymässä näkyvä koodi suoritetaan selaimen toimesta osana sivun rakennetta. (OWASP, 2020)

2.6.8 Haitallisten liitteiden torjunta (Virus Protection)

Jos web-sovellus sallii tiedostojen lähettämisen, ne tulee tarkistaa virusten ja haittaohjelmien varalta. Virustorjuntaohjelmisto tulee olla päivitetty ja mielellään automaattisesti tarkistava ennen tiedoston tallentamista tai käsittelyä. (OWASP, 2020)

2.6.9 Palvelunestohyökkäykset ja resurssien kuormitus (DoS & Resource Limiting)

Web-sovellukset voivat olla alttiita DoS-hyökkäyksille, kuten ylisuurten viestien lähettämiselle tai liian monien yhteyksien avaamiselle. Suojauskeinot sisältävät rajoituksia viestin koolle, istuntojen määrälle, muistin- ja prosessorin käytölle sekä pyyntöjen käsittelynopeudelle. (OWASP, 2020)

2.6.10 Päätepisteiden turvallisuus ja yhteensopivuus (Endpoint Security Profile)

Web-palvelujen kannattaa olla yhteensopivia vakioprofiilien, kuten WS-I Basic Profile kanssa, mikäli käytössä on SOAP-pohjainen rajapinta. (OWASP, 2020)

Vaikka REST- ja WebSocket-pohjaisissa ratkaisuissa tämä ei ole suoraan sovellettavissa, rajapintojen määrittely ja hallinta tulee silti olla selkeää ja turvallista hyvin toteutetulla auktorisoinnilla.

2.7 Tietoturvatestauksen menetelmiä

Tietoturvatestauksen tavoitteena on tunnistaa ja arvioida web-sovelluksiin kohdistuvia haavoittuvuuksia ennen kuin ne voivat johtaa tietomurtoon, tietovuotoon tai muunlaiseen väärinkäytökseen. Äskeisessä kappaleessa esitelty OWASP:in Web Security Cheat Sheet:in (OWASP, 2020) kymmenen kohtaa toimivat perustana testausstrategian muodostamiselle. Sen lisäksi tarkastellaan, onko laivanupotuksessa huijaaminen mahdollista asiakassovelluksen puolelta.

Tässä luvussa käsitellään testausmenetelmiä, jotka soveltuvat erityisesti web-palveluiden yleisten uhkien tunnistamiseen ja analysointiin – riippumatta käytetystä ohjelmistokehyksestä tai teknologiasta. Taaskin tarkempi tietoturvatestauksen soveltaminen Laivanupotus-pelin kontekstiin käsitellään empiirisessä osassa.

Taulukko 1.Tietoturvatestauksen menetelmät, niitä vastaavat uhat sekä testauksen olennaisuus.

Tietoturvatestauksen menetelmä	**OWASP Cheat Sheet -uhka, jota menetelmä kattaa**	**Olennainen laivanupotuksen toteutuksessa**
Penetraatiotestaus & input fuzzing	Viestien eheys ja salaus, auktorisointi ja käyttöoikeudet, Rajapintojen turvallisuus.	Kyllä, viestien eheys ja tietoturva
Input fuzzing	Skeema- ja sisältövalidointi, Haitallinen syöte	Kyllä, haitallinen syöte mahdollistaa epäreilun edun.
Tunnistetason manipulointitestaus	Käyttäjän autentikointi, Viestien eheys, Auktorisointi ja käyttöoikeudet	Kyllä, käyttäjä autentikoidaan JWT-tunnistetta vasten.
Asiakassovellus-puolen eheyden varmistaminen	Syötteiden validointi, Auktorisointi ja käyttöoikeudet, Viestien eheys (jos asiakassovellus voi ohittaa logiikan)	Kyllä, laivanupotusta pelataan käyttäjän selaimesta käsin.

TLS-sertifikaatin ja kuljetustason suojaus	Kuljetuksen suojaus ja enkoodaus, Palvelimen autentikointi, Viestien eheys ja salaus	Ei, laivanupotuksen kehitysympäristössä TLS-sertifikaatti ei ole olennainen. Tuotannossa julkaisualusta huolehtii tästä.
Rajapintojen turvallisuuden arviointi	Auktorisointi ja käyttöoikeudet, Rajapintojen turvallisuus, Viestien eheys	Kyllä, laivanupotuksen palvelinpuolta käytetään REST-rajapinnan kautta.
Käsin tehty ulostulon katselmointi	Ulostulon koodaus (XSS:n torjunta)	Ei, XSS ei Cross site scripting on opinnäytetyön aiheen ulkopuolella.
Manuaalinen HTTPS/TLS-konfiguraation tarkistus	Palvelimen autentikointi, Kuljetuksen suojaus	Ei, laivanupotuksen kehitysympäristössä https-ei ole olennainen. Julkaisualustat huolehtivat tästä automaattisesti.
Automatisoitu virustarkistusjärjestelmä	Haitallisten liitteiden torjunta	Ei, Laivanupotuksessa ei käytetä liitetiedostoja.
Sovellus	Palvelunestohyökkäykset ja resurssien kuormitus (DoS & Resource Limiting)	Ei, julkaisualustat huolehtivat tästä automaattisesti.

2.7.1 Penetraatiotestaus

Penetraatiotestaus on tietojärjestelmän tai verkon tietoturvan arviointimenetelmä, jossa pyritään löytämään haavoittuvuuksia hyödyntämällä niitä hallitusti. Testauksen suorittavat eettiset hakkerit, jotka käyttävät samoja menetelmiä kuin haitalliset hyökkääjät, mutta ilman vahingon aiheuttamista järjestelmälle tai organisaatiolle. Haavoittuvuudet voivat liittyä esimerkiksi virheellisiin asetuksiin, tunnistautumiseen, tunnettuihin ohjelmistoaukkoihin tai liiketoimintalogiikkaan. Penetraatiotestaus auttaa tunnistamaan näitä heikkouksia ja arvioimaan nykyisten suojaustoimien toimivuutta. Tavoitteena on osoittaa, että ulkopuolinen hyökkääjä voisi löytää ja hyödyntää tietoturva-aukon. Testaus yhdistää manuaalista ja automatisoitua testausta, ja siihen kuuluu erilaisten työkalujen käyttö

ennalta sovitussa laajuudessa ja luvalla. Testauksen tuloksista laaditaan asiakkaalle raportti, joka sisältää tiedot testauksen kattavuudesta, löydetyistä haavoittuvuuksista, niiden vakavuudesta ja korjausehdotuksista (Imperva s.a.).

2.7.2 Input fuzzing

Fuzz-testaus eli fuzzing on automatisoitu ohjelmistotestauksen menetelmä, jossa testattavaan järjestelmään syötetään virheellisesti tai satunnaisesti muodostettua syötedataa tarkoituksena löytää toteutusvirheitä. Kyseessä on niin sanottu Black Box -testaustekniikka, eli testaus tehdään ilman tietoa järjestelmän sisäisestä rakenteesta (OWASP. s.a.)

Vaikka OWASP (OWASP. s.a.) kertoo Fuzz-testauksen olevan automaattista voi sitä suorittaa myös manuaalisesti siihen soveltuvan sovelluksen avulla.

Lisäksi Fuzz-testausta voidaan toteuttaa useilla tavoilla, joista yksi on API-fuzzing. Siinä lähetetään satunnaista tai odottamatonta syötedataa sovellusrajapintaan (API), jotta voidaan tunnistaa esimerkiksi puutteita syötteiden validoinnissa tai muita tietoturvahaavoittuvuuksia (Imperva s.a.). Tässä opinnäytetyössä suoritetaan yllä kuvailtua API-fuzz-testausta.

2.7.3 Token-tason manipulointitestaus

nykyaikaista tietoturvatestausta, erityisesti web-sovelluksissa, joissa käytetään tunnistepohjaisia autentikointimekanismeja kuten JSON Web Tokeneita (JWT). Testauksessa selvitetään, voiko hyökkääjä manipuloida tokenin sisältöä tai sen allekirjoitusta siten, että järjestelmä hyväksyy sen virheellisesti. Tarkoituksena on tuoda esiin, kuinka luotettavasti sovellus validoi tokenin eheyden ja käsittelee vanhentuneita, väärennettyjä tai muokattuja tunnisteita. Esimerkiksi heikko allekirjoitusavain, väärin käsitellyt algoritmit (kuten "none"-algoritmin hyväksyminen) tai puutteellinen aikaleimojen tarkistus voivat paljastaa kriittisiä haavoittuvuuksia (OWASP 2018; PortSwigger. s.a. b.).

2.7.4 Asiakassovelluksen integriteetti ja pelilogiikan eheys

Asiakassovelluksen tietoturvariskit ovat niin laaja käsite, että niitä kaikkia ei ole tarkastella tämän opinnäytetyön puitteissa. Sen sijaan keskitytään varmistamaan, että asiakassovellus ei näytä ylimääräistä sinne kuulumatonta tietoa. OWASP (OWASP, s.a b) määrittelee sen yhdeksi kymmenestä yleisimmistä asiakasovellusten tietoturvariskeistä: ”Asiakassovelluksen koodissa olevat luottamukselliset tiedot voivat paljastaa liiketoiminnan logiikkaa, kehittäjän kommentteja, omia algoritmeja tai järjestelmän tietoja, jos ne sisältyvät asiakassovelluksen koodiin tai tallennettuun dataan.

Tämä voi aiheuttaa vakavia tietoturvariskejä, sillä hyökkääjät voivat saada pääsyn näihin tietoihin ja hyödyntää niitä väärin (OWASP, s.a b).”

Sen lisäksi tarkastellaan ja testataan voiko asiakassovelluksen pelilogiikan eheyttä kiertää esim. selaimen konsolityökalulla.

2.7.5 Rajapintojen turvallisuuden arviointi

REST- rajapinnat ovat kriittisiä komponentteja web-sovellusten turvallisuuden näkökulmasta. Näiden rajapintojen testauksessa tarkastellaan, ovatko päätepisteet asianmukaisesti autentikoituja ja auktorisoituja. Esimerkiksi rikkinäinen autentikointi voi mahdollistaa käyttäjien identiteettien väärentämisen tai jopa järjestelmänvalvojan oikeuksien haltuunoton, kuten Redditin vuonna 2018 kokemassa tietomurrossa tapahtui (Tal 11.5.2025).

Lisäksi rajapintojen kautta kulkee usein arkaluontoista tietoa, kuten käyttäjätunnuksia ja henkilötietoja, joiden suojaamatta jättäminen voi johtaa vakaviin yksityisyyden loukkauksiin (Tal 11.5.2025) Turvallisuuden arvioinnissa on tärkeää varmistaa, ettei järjestelmä paljasta tarpeettomia tietoja esimerkiksi virheilmoitusten kautta. Rajapintojen arviointi on keskeinen osa palvelun eheyttä ja käyttöoikeuksien hallintaa.

Laivanupotuksen tapauksessa REST-sekä WebSocket-rajapintojen turvallisuutta arvioidaan näiltä kahdelta näkökulmalta. Näihin rajapintoihin kuuluu muitakin riskejä, mutta usein luotetut web-sovellusten julkaisualustat sekä web-kehityksen viitekehykset ottavat nämä uhat automaattisesti huomioon kehittäjän puolesta.

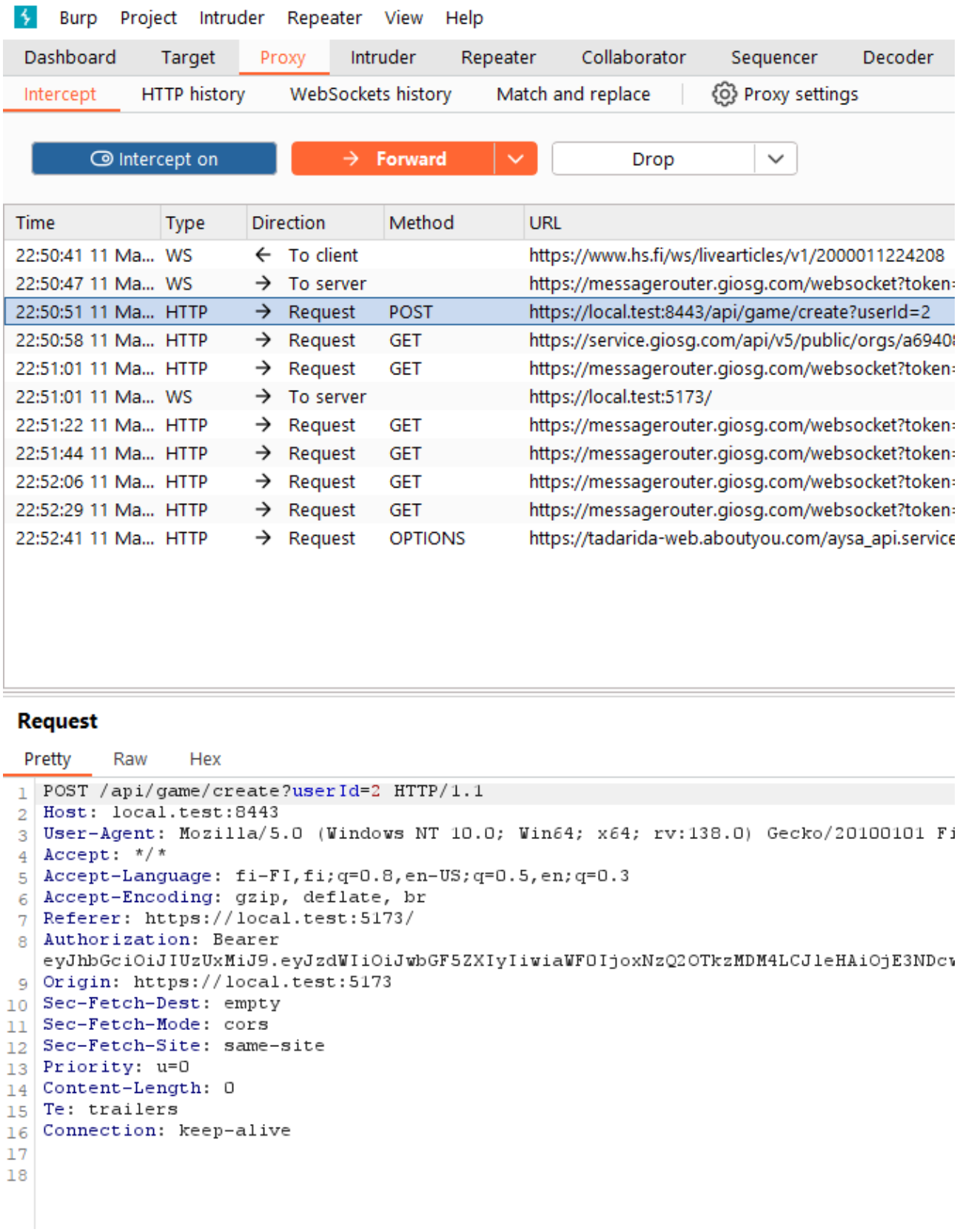
2.8 BURP-Suite työkaluna

Tämän opinnäytetyön empiirisessä osassa suoritetaan tietoturvatestausta laivanupotus-pelille. Suuri osa siitä koostuu viestiliikenteen tarkastelusta asiakassovelluksen ja palvelimen välillä. Tähän on monta työkalua olemassa ja tässä opinnäytetyössä päädyttiin BURP-Suite ohjelmistoon sen ilmaisen version ja käytettävyyden vuoksi.

Burp Suite on Portswigger-yhtiön kehittämä työkaluohjelmisto, jota käytetään verkkosovellusten haavoittuvuuksien testaamiseen. Se tarjoaa monipuolisen työkalupaketin tietoturvatestaajille, ja sen toiminnallisuuksia voidaan laajentaa lisäosilla, joita kutsutaan BAppseiksi. Burp Suite on erityisesti tietoturva-ammattilaisten ja bug bounty -metsästäjien suosiossa helppokäyttöisyytensä vuoksi. Se tarjoaa kolme eri versiota: ilmainen Community Edition, maksullinen Professional

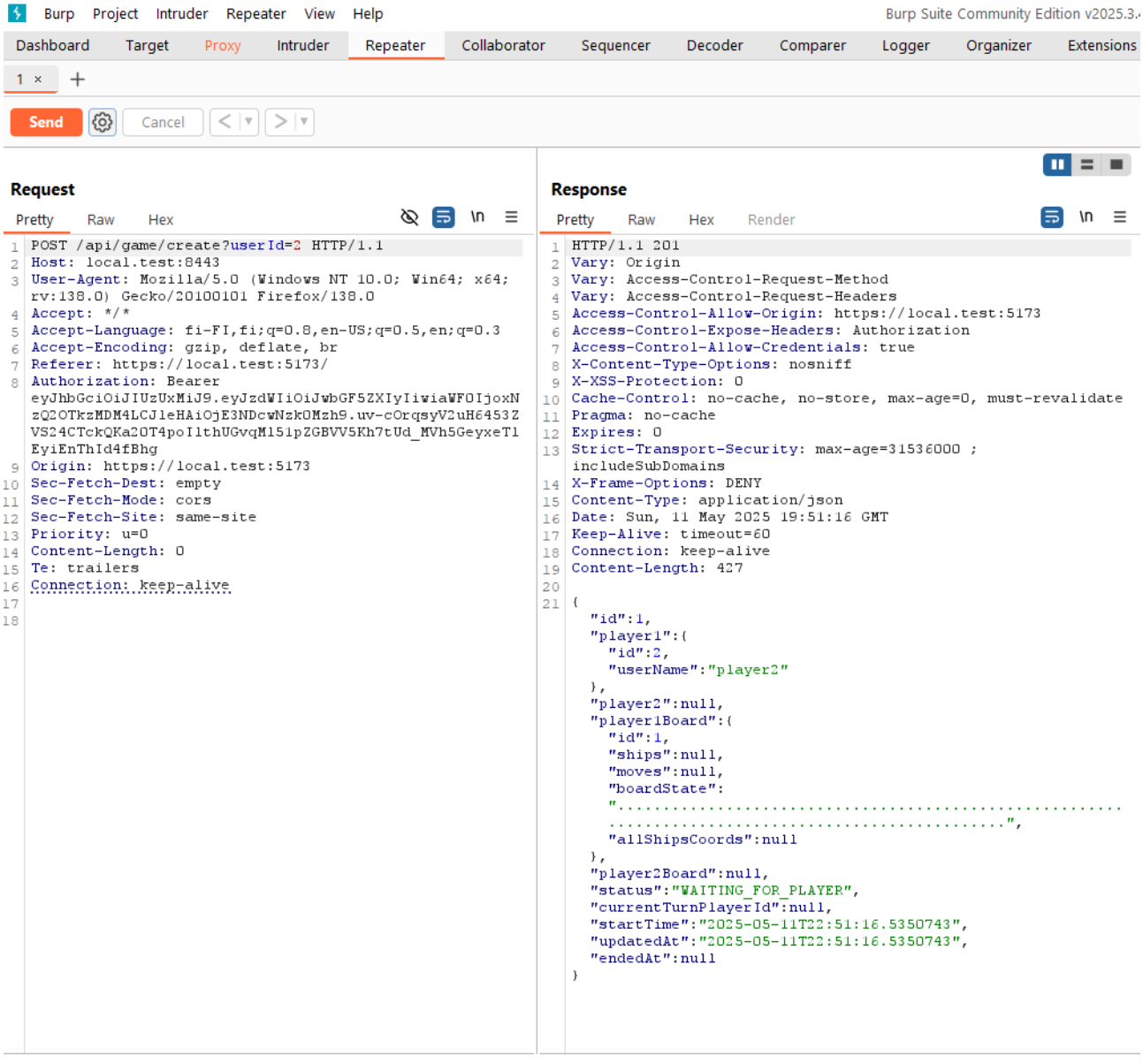
Edition (399 USD/vuosi) sekä laajempi Enterprise Edition (3999 USD/vuosi). (GeeksForGeeks, 2025). Rajallisen budjetin vuoksi tässä opinnäytetyössä päädyttiin käyttämään Community Edition:ia.

Burp Suiten tärkeimpiin ominaisuuksiin kuuluu sieppaava välityspalvelin (intercepting **proxy**), jonka avulla käyttäjä voi tarkastella ja muokata HTTP- ja WebSocket pyyntöjä ja -vastauksia reaaliajassa. Proxy mahdollistaa myös pyyntöjen siirtämisen suoraan muihin Burp Suiten työkaluihin ilman manuaalista kopiointia. Välityspalvelin voidaan konfiguroida käyttämään tiettyä IP-osoitetta ja porttia, ja sen suodatusasetuksilla voidaan rajata tarkasteltavia pyyntö- ja vastauspareja. (GeeksForGeeks, 2025).



Kuva 6. Kuvankaappaus BURP-Suiten proxy välilehdeltä.

Repeater on Burp Suiten työkalu, jonka avulla käyttäjä voi lähettää muokattuja HTTP-pyyntöjä toistuvasti. Työkalun avulla voidaan testata, miten palvelin reagoi erilaisiin syötteisiin ja kuinka tehokkaasti se validoi sekä puhdistaa käyttäjän antamat tiedot. Repeaterilla voidaan selvittää esimerkiksi, millaisia arvoja palvelin odottaa syötteissä, kuinka hyvin syötteiden käsittely toteutetaan ja millaisia suojauksia kuten CSRF-tarkistuksia on käytössä. Lisäksi työkalulla voidaan tunnistaa istuntokoodeihin liittyviä evästeitä. (GeeksForGeeks, 2025).



Kuva 7. Kuvankaappaus BURP-Suiten Repeater-välilehdeltä.

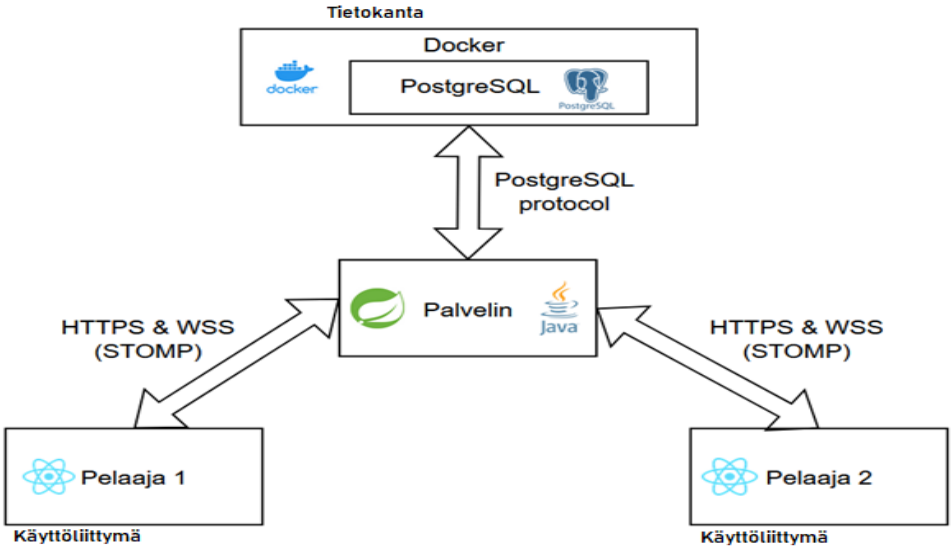
3 Empiirinen osa – Laivanupotus-selainmoninpelin tietoturvatestaus ja toteutus

Tässä kappleessa käydään läpi laivanupotus-pelin teknologisia valintoja sekä laivanupotuksen asiakassovelluksen ja palvelinpuolen välistä viestiliikennettä. Lisäksi laivanupotuksen tietoturvaa testataan konkreettisesti. Tietoturvatestaus suoritetaan niitä uhkia vasten, jotka on havaittu tietoperustan kappaleessa 2.7 taulukossa 1 olennaisiksi laivanupotuksen kannalta. Tällä saadaan selville, kuinka hyvin laivanupotuksen kehityksessä on otettu huomioon tietoturva ja huijauksenesto.

3.1 Laivanupotus-pelin tekninen toteutus

Tässä luvussa käsitellään Laivanupotus-pelin teknistä toteutusta erityisesti niiltä osin, joilla on merkitystä tietoturvan ja huijauksenestoon kannalta.

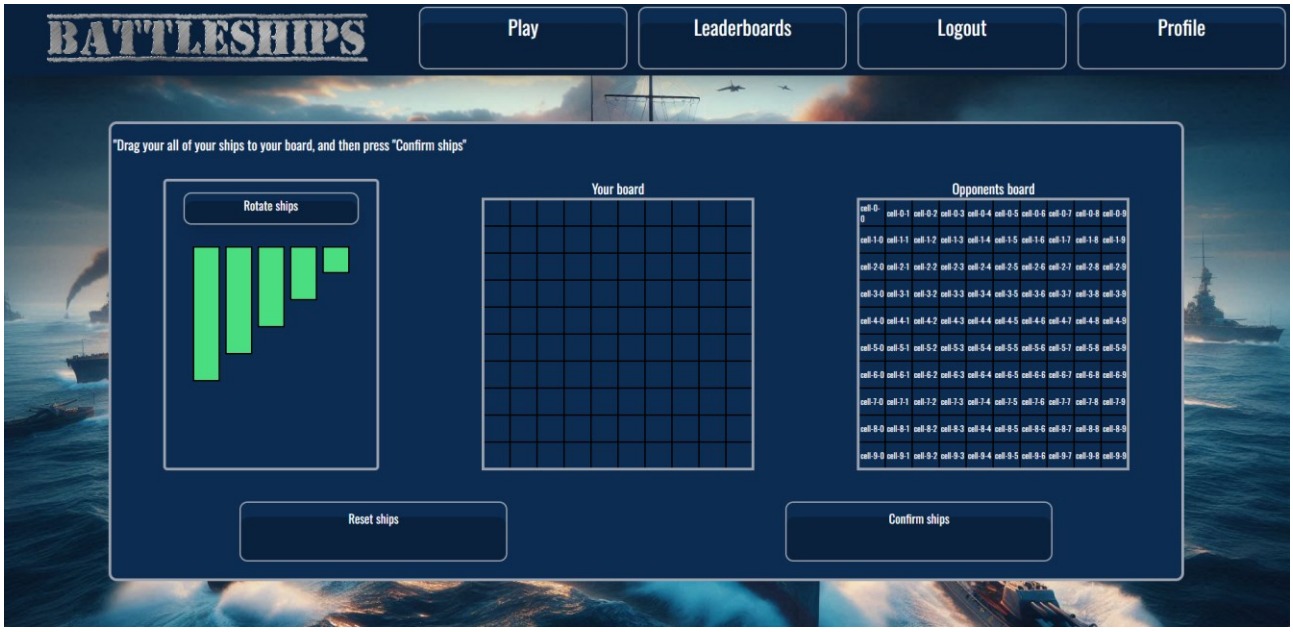
Pelin arkkitehtuuri perustuu moderniin client-server-malliin, jossa palvelin hallitsee keskeisiä pelilogiikan toimintoja ja tietoturvan valvontaa, kun taas selainpuoli tarjoaa käyttöliittymän ja käyttäjän ohjausmekanismit. (Britannica, 2025) Keskeisiä teknologioita ovat muun muassa Java (ohjelmointikieli), Spring Boot (Javan viitekehys Web-kehitykseen), React (Käyttöliittymäkirjasto), JWT-autentikointi sekä WebSocket/STOMP-pohjainen viestintä, joiden yhteistoiminta muodostaa järjestelmän toimintakaaren ytimen.



Kuva 8. Havainnekuva Laivanupotuksen sovellusarkkitehtuurista

3.1.1 Järjestelmäarkkitehtuuri ja teknologiavalinnat

Laivanupotus-peli noudattaa klassista client-server-arkkitehtuuria, jossa asiakassovellus kommunikoi palvelimen kanssa HTTP- ja WebSocket-protokollien kautta. Käyttöliittymä on rakennettu React-kirjastolla, jonka valintaan vaikuttivat sen laaja ja hyväksi todettu ekosysteemi ja henkilökohtainen React-osaaminen.



Kuva 9. Kuvankaappaus Laivanupotuksen käyttöliittymästä selaimessa.

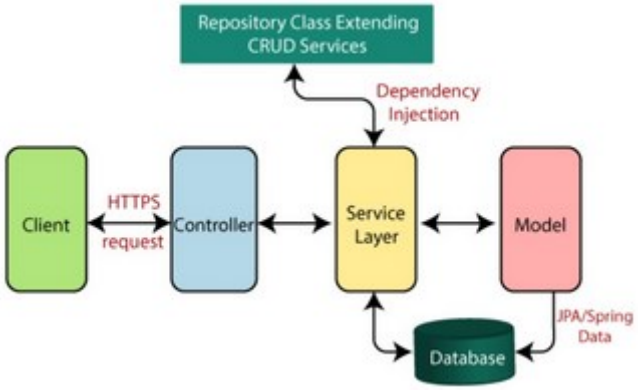
Palvelinpuoli on toteutettu Java-kielellä Spring Boot -viitekehyksen avulla, joka tarjoaa valmiit työkalut REST-rajapintojen, WebSocket-viestinnän ja käyttäjäautentikoinnin toteuttamiseen. Reaaliaikainen viestintä pelaajien välillä pelin ollessa käynnissä on toteutettu WebSocket-protokollalla, jonka päällä toimii STOMP-protokolla viestien rakenteen hallitsemiseksi. Tilatiedot ja tapahtumat jaetaan pelaajille WebSocketin kautta, kun taas kaikki pelitoimintaa muuttavat toiminnot, kuten liikkeet ja laivojen sijoittaminen, toteutetaan HTTP POST -pyyntöinä palvelimen REST-rajapintoihin.

Laivanupotuksen kehitysympäristössä on henkilökohtaisesta mielenkiinnosta konfiguroitu itse allekirjoitettu TLS-sertifikaatti, mutta se on epäolennaista tämän opinnäytetyön kannalta. Taasen tuotantoympäristössä on ehdottoman tärkeää käyttää TLS-salausta ja luotettavat julkaisualustat tarjoavat tämän automaattisesti.

Käyttäjän tunnistautuminen ja valtuutus tapahtuvat JWT (JSON Web Token) -tunnisteilla, jotka liitetään kaikkiin asiakaspään HTTP-pyyntöihin. Palvelin tarkistaa jokaisessa pyynnössä tunnisteen

aitouden ja oikeudet toiminnon suorittamiseen. Tarkempi kuvaus tästä löytyy kappaleesta 3.1.4 Viestiliikenteen rakenne ja viestintäprotokolla.

Pelin tietovarastona toimii PostgreSQL-tietokanta, johon tallennetaan käyttäjätiedot, pelihistoria ja pelitilojen tilanteet. Tietojen käsittely palvelimella noudattaa kerrosarkkitehtuuria, jossa controller-kerros ohjaa asiakassovelluksen pyyntöjen vastaanottoa, palvelukerros (service layer) hoitaa pelilogiikan (esim. huijauksenesto, vuoronhallinta ja tietokantakerros (repository) vastaa tietojen pysyvästä säilytyksestä.



Kuva 10. Havainnekuva palvelimen kerrosarkkitehtuurista (Datadisk. s.a.)

Tietoturvanäkökulmasta järjestelmä on rakennettu monikerroksisen suojauksen periaatteella. Kaikki kriittinen pelilogiikka käsitellään palvelinpuolella, mikä estää asiakassovelluksen manipulaatiosta aiheutuvat huijausyritykset. Syötteet validoidaan sekä asiakassovellus- että palvelinpuolella, ja pelitapahtumat kulkevat aina palvelimen kautta ennen kuin ne välitetään toiselle pelaajalle.

3.1.2 Autentikaation ja auktorisoinnin toteutus

Laivanupotus-pelin autentikaatio- ja auktorisointijärjestelmä on toteutettu hyödyntäen Spring Security -viitekehystä ja aikaisemmin tässä opinnäytetyössä tarkasteltuja JWT-tunnisteita. Käyttäjän rekisteröityessä järjestelmään salasana suolataan ja hashataan käyttäen Bcrypt-algoritmia ennen tietokantaan tallentamista, mikä suojaa salasanoja tietomurtojen varalta. Kirjautumisprosessissa käyttäjätunnus ja salasana lähetetään HTTPS-yhteyden kautta palvelimelle, jossa salasana validoidaan hashattua salasanaa vasten, minkä jälkeen onnistuneesta kirjautumisesta generoidaan allekirjoitettu JWT-tunniste.

Tässä toteutuksessa tokenin voimassaoloaika on rajattu 24 tuntiin, mikä pienentää varastetun tokenin väärinkäytön riskiä. Onnistuneen autentikaation jälkeen allekirjoitettu tunniste tallennetaan selaimen muistiin ja välitetään jokaisessa HTTPS-pyynnössä ja WSS-yhteyden yli lähetettävässä STOMP-viestissä Authorization-otsikon Bearer-kenttänä.

```
@PostMapping("/login")
public ResponseEntity<OwnUserProfileDto> loginUser(@RequestBody LoginRequestDto loginDto) {
    // Returns auth token as headers, if login successful.
    ResponseEntity<OwnUserProfileDto> loginResponse = userService.loginUser(loginDto);

    UserAuthenticator.checkUserRoles();
    return loginResponse;
};
```

Kuva 11. kuvankaappaus palvelimen lähdekoodin ”/login”-päätepisteestä, jota käytetään kirjautumiseen.

Laivanupotuksessa auktorisointi perustuu roolipohjaiseen pääsynhallintaan, jossa käyttäjällä voi olla USER- tai ADMIN-rooli. Pelitoimintojen auktorisointi on kuitenkin hienojakoisempaa ja perustuu pelisession tunnisteisiin ja pelaajan identiteettiin. STOMP-viestien auktorisoinnissa hyödynnetään sekä istuntotunnisteita että viestien sisältämää metadataa, kuten pelaajaidentiteettiä ja pelitunnistetta.

Palvelin hallitsee pelitilannetta ja tekee tarkistuksia jokaisen toiminnon kohdalla. STOMP-viesti kantaa headerissaan JWT-tunniste ja lähettää istunnon- ja pelaajan tunnisteen URL-parametreinä. Viestin saapuessa palvelimelle tunniste vahvistetaan ja tunnisteet tarkistetaan tietokantahakua vasten. Mikäli tietokannasta löytyy aktiivinen peli annetuilla tunnisteilla, siirrytään halutun toiminnon auktorisointiin ja suorittamiseen (esim. Siirto, laivojen sijoitus).

```
@PostMapping("/{matchId}/make-move")
public ResponseEntity<Move> makeMove(@PathVariable Long matchId, @RequestParam Long userId,
        @RequestBody Move move) {
    Move resultMove = gameService.makeMove(matchId, userId, move);
    return ResponseEntity.ok(resultMove);
};
```

Kuva 12. kuvankaappaus palvelimen lähdekoodin ”/{matchId}/make-move”-päätepisteestä, jota käytetään vastustajan laudalle ampumiseen.

Kuvassa 12 saadaan Move-tyyppiä oleva resultMove-niminen olio takaisin pelilogiikan suorittavan gameService.makeMove()-metodilta.

```java
@Override
public Move makeMove(Long matchId, Long playerId, Move move) {
    //Etsitään peli tietokannasta pelin ID:llä
    Match match = matchRepository.findById(matchId).orElseThrow(() -> new RuntimeException("Match not found!"));
    // Tarkistetaan, että peli on käynnissä
    if (match.getStatus() != GameStatus.IN_PROGRESS) {
        throw new RuntimeException("Match is not in progress!");
    }
    // Tarkistetaan, että pelaaja on oikeassa pelissä
    if (!playerId.equals(match.getCurrentTurnPlayerId())) {
        throw new RuntimeException("It's not your turn!");

    }

    // Määritetään pelaajan ID:stä kumpaan pelilautaan pelaajaan siirto kuuluu.
    Board boardToTarget = playerId.equals(match.getPlayer1().getId()) ? match.getPlayer2Board()
            : match.getPlayer1Board();
    // Tarkistetaan, että siirto on oikeassa pelilaudassa
    if (hasCoordinateAlreadyBeenHit(boardToTarget, move)) {
        throw new RuntimeException("You have already made a move on this coordinate!");
    }
    // Liitetään siirto pelilautaan
    boardToTarget.getMoves().add(move);
    // Tarkistetaan, onko siirto osuma
    Boolean isHit = isHit(boardToTarget, move);
    move.setHit(isHit);
    move.setPlayerBehindTheMoveId(playerId);
    // Luodaan viesti siirron tuloksesta
    String moveResponseMessage = getMoveResponseMessage(move, boardToTarget);
    // Siirretään viesti WebSocketille sopivaan muotoon.
    WebSocketMoveResponseDto message = WebSocketMoveResponseDto.builder().move(move).message(moveResponseMessage)
            .build();
    // Lisätään siirron tiedot pelin tilaan
    updateMatchState(match, boardToTarget, playerId);
    // Tallennetaan pelin uusi tila tietokantaan
    matchRepository.save(match);
    // Lähetetään siirron tulos WebSocketille, joka lähettää sen vastustajalle.
    webSocketHandler.notifyMoveMade(matchId, message);
    // Lähetetään pelaajalle siirron tulos.
    return move;

}
```

Kuva 13. Ampumisen pelilogiikka yhdessä funktiossa nimeltään makeMove. Kommentit kertovat mitä jokaisella funktion askeleella tehdään.

```java
public void notifyMoveMade(Long matchId, Object moveResult) {
    messagingTemplate.convertAndSend("/topic/game/" + matchId + "/move",
    moveResult);
};
```

Kuva 14. webSocketHandler-luokan notifyMoveMade-metodi.

Kuvassa 14. kuvailtua metodia käytetään lähettämään WebSocket-yhteyden yli ”/topic/game/”+matchId+”/move” päätepisteeseen tietoa toteutuneesta siirrosta.

Kuvan 13. metodilla varmistetaan, että pelaaja voi tehdä siirtoja vain omissa aktiivisissa peleissään ja vain omalla vuorollaan.

3.1.3 Pelilogiikan jakautuminen asiakassovellus- ja palvelinpuolelle.

Kuten muissakin selainvideopeleissä Laivanupotus-pelin client-puoli suoritetaan käyttäjän selaimella. Se koostuu JavaScript-koodista, CSS:stä sekä HTML:stä, joita voi helposti tutkia sekä manipuloida selaimen kehitystyökaluilla, mahdollisesti ohittaen asiakassovelluksessa olevia validaatioita pelitilanteen sekä peliliikkeiden oikeudenmukaisuudesta. Siksi olen toteuttanut tietoturvan ja huijauksenestonkannalta kriittisen pelilogiikan ja validoinnin palvelinpuolella.

Tämä pelilogiikan jako asiakassovellus- ja palvelinpuolelle noudattaa verkkokehityksessä tunnettua ”Never the trust the client”-periaatetta, jossa kaikki turvallisuuden kannalta merkittävät päätökset ja validoinnit tehdään palvelinpuolella. Tälle periaatteelle ei ole vakiintunutta lähdettä sillä sen odotetaan olevan maalaisjärkeä tietoturvan kanssa työskenteleville.

Server-puolella toteutettu pelilogiikka sisältää:

- Pelisession luomisen ja hallinnoinnin

- Laivojen sijoittelun validoinnin ja tallentamisen

- Pelaajien vuorojärjestyksen hallinnan

- Ampumistoimintojen validoinnin ja tulosten määrittelyn

- Pelin lopputuloksen määrittelyn ja tilastojen tallentamisen

Client-puolella toteutettu pelilogiikka keskittyy käyttöliittymän toiminnallisuuksiin:

- Pelilaudan visualisointi ja interaktiivisuus

- Laivojen sijoittelun käyttöliittymä

- Pelitilan paikallinen säilytys käyttäjäkokemuksen parantamiseksi

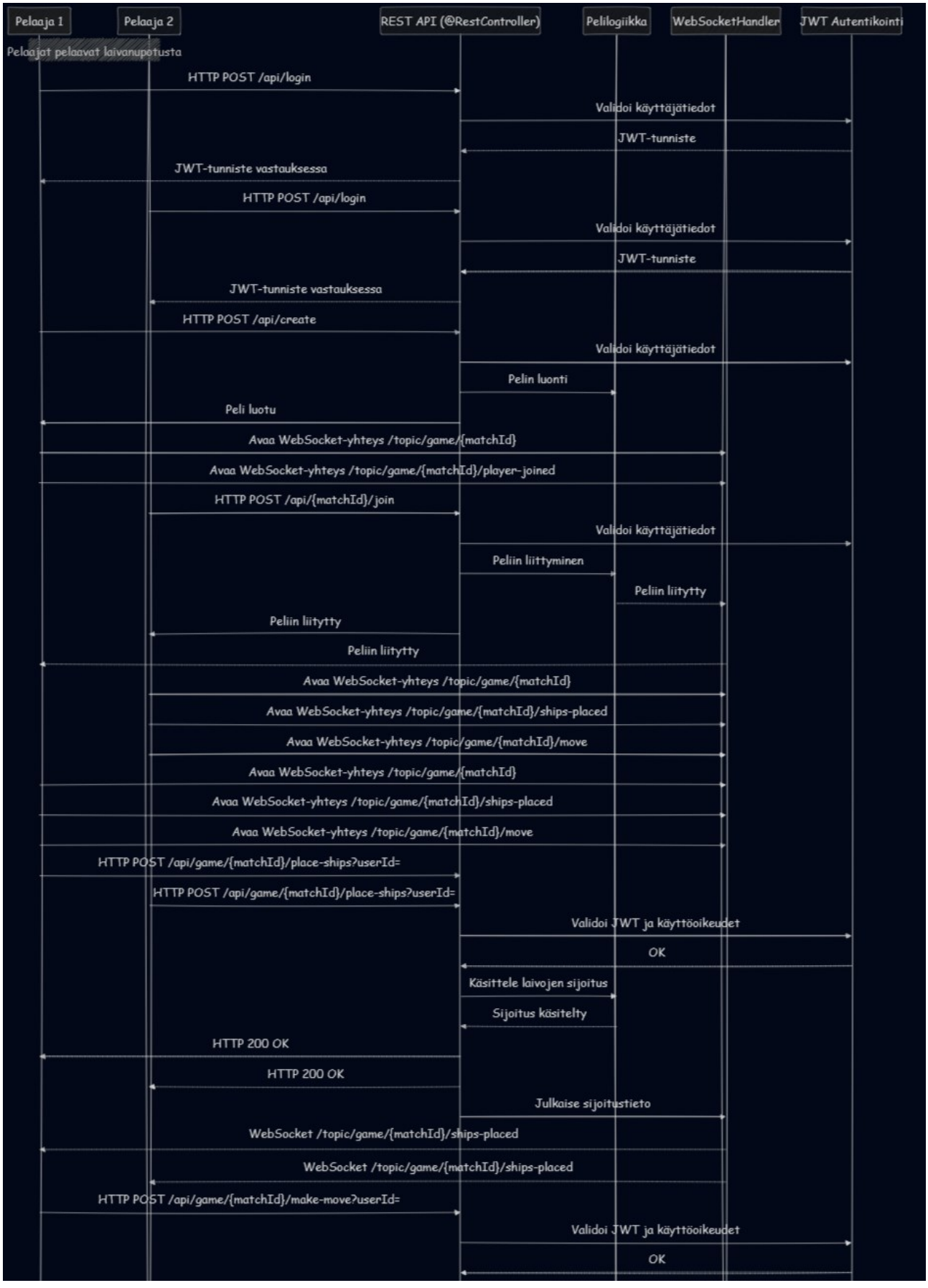
- Animaatiot ja visuaaliset efektit

3.1.4 Viestiliikenteen rakenne ja viestintäprotokolla

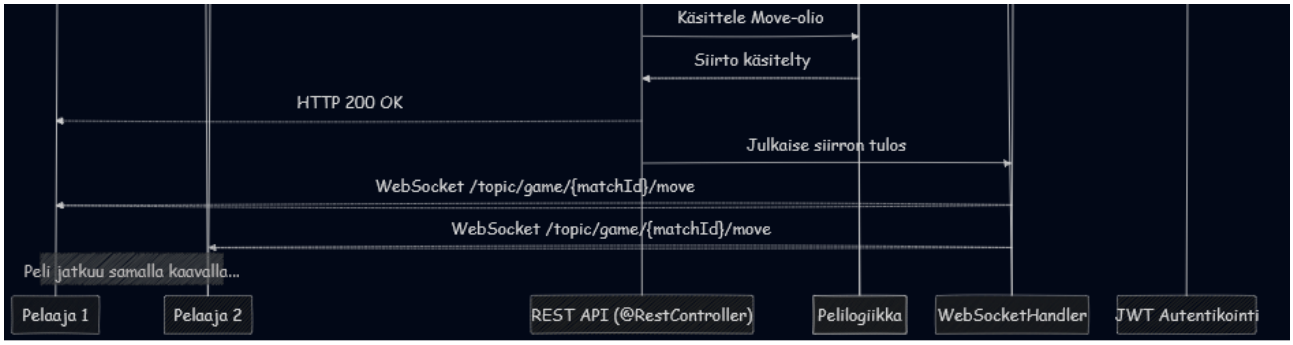
Laivanupotus-sovelluksessa viestiliikenteen arkkitehtuuri rakentuu kahden rinnakkaisen teknologian – HTTP REST -rajapintojen ja WebSocket-viestien – varaan. Pelitoiminnot, kuten laivojen sijoittaminen ja ampuminen, välitetään palvelimelle HTTP-pyyntöinä, minkä jälkeen palvelin julkaisee näiden toimintojen tulokset WebSocketin kautta molemmille pelaajille. Näin yhdistetään HTTP-protokollan tarjoama turvallinen kontrolli ja WebSocketin mahdollistama reaaliaikaisuus.

Spring Bootin REST-rajapinnat määritellään @RestController-annotoiduissa luokissa. Esimerkiksi POST /api/game/{matchId}/make-move?userId= vastaanottaa pelaajan ampumiskomennon. Pyyntö sisältää Move-olion, joka validoidaan ja käsitellään palvelinpuolella. Kun pelilogiikka on suoritettu WebSocket-kanavaan /topic/game/{matchId}/move, jolloin molemmat pelaajat saavat tiedon siirrosta välittömästi.

HTTP API -rajapintojen käyttö mahdollistaa vahvan autentikoinnin JWT-tunnisteiden avulla. Jokaisessa pyynnössä validoidaan käyttäjän identiteetti ja rooli, sekä tarkistetaan oikeus suorittaa kyseinen toiminto. WebSocketia käytetään pelkästään pelin tilapäivitysten ja tapahtumailmoitusten jakeluun, eikä sen kautta vastaanoteta käyttäjän syötteitä.


Pelaaja 1
Pelaaja 2
REST API (@RestController)
Pelilogiikka
WebSocketHandler
JWT Autentikointi
Pelaajat pelaavat laivanupotusta
HTTP POST /api/login
Validoi käyttäjätiedot
JWT-tunniste
JWT-tunniste vastauksessa
HTTP POST /api/login
Validoi käyttäjätiedot
JWT-tunniste
JWT-tunniste vastauksessa
HTTP POST /api/create
Validoi käyttäjätiedot
Pelin luonti
Peli luotu
Avaa WebSocket-yhteys /topic/game/{matchId}
Avaa WebSocket-yhteys /topic/game/{matchId}/player-joined
HTTP POST /api/{matchId}/join
Validoi käyttäjätiedot
Peliin liittyminen
Peliin liitytty
Peliin liitytty
Peliin liitytty
Avaa WebSocket-yhteys /topic/game/{matchId}
Avaa WebSocket-yhteys /topic/game/{matchId}/ships-placed
Avaa WebSocket-yhteys /topic/game/{matchId}/move
Avaa WebSocket-yhteys /topic/game/{matchId}
Avaa WebSocket-yhteys /topic/game/{matchId}/ships-placed
Avaa WebSocket-yhteys /topic/game/{matchId}/move
HTTP POST /api/game/{matchId}/place-ships?userId=
HTTP POST /api/game/{matchId}/place-ships?userId=
Validoi JWT ja käyttöoikeudet
OK
Käsittele laivojen sijoitus
Sijoitus käsitelty
HTTP 200 OK
HTTP 200 OK
Julkaise sijoitustieto
WebSocket /topic/game/{matchId}/ships-placed
WebSocket /topic/game/{matchId}/ships-placed
HTTP POST /api/game/{matchId}/make-move?userId=
Validoi JWT ja käyttöoikeudet
OK

Kuva 15. Kaaviolla esitetty laivanupotus pelin kulku viestiliikenteen kannalta



Kuva 16 jatkoa kuvalle 15.

Taulukko 2. Http-api rajapinnat pelilogiikkaa koskien laivanupotuksen palvelimelta

HTTP-metodi	**Polku**	**Kuvaus**
POST	/api/game/create	Luo uusi peli
POST	/api/game/{matchId}/join	Pelaaja liittyy peliin
POST	/api/game/{matchId}/make-move	Suorittaa ampumistoiminnon
POST	/api/game/{matchId}/place-ships	Asettaa laivat laudalle
GET	/api/game/{matchId}/gamestate	Palauttaa pelin tilanteen
POST	/api/game/{matchId}/authorize	Sallii ja evää pääsyn peleihin liittymiseen

Taulukko 3. WebSocket-kanavat tiedonvälitystä varten laivanupotuksen palvelimelta

Kanava	Viestin julkaisutapahtuma
/topic/game/{matchId}	Yleiset pelitilan päivitykset esim, voitto, häviö
/topic/game/{matchId}/move	Pelaajan siirron tulos
/topic/game/{matchId}/player-joined	Uusi pelaaja liittyi peliin
/topic/game/{matchId}/ships-placed	Vastustaja sijoitti laivat laudalle
/topic/game/{matchId}/opponent-disconnected	Vastustaja katkaisi yhteyden

3.2 Tietoturvatestauksen metodologia

Tässä luvussa kuvataan opinnäytetyössä sovellettava tietoturvatestauksen metodologia, jonka pohjalta laivanupotuksen tietoturvatestaus suoritetaan. Luku esittelee testauksen kohteet, käytettävät testausmenetelmät ja niiden perustelut, testausympäristön rakenteen sekä dokumentointitavan.

Luvun sisältö toimii viitekehyksenä seuraavassa kappaleessa kuvatulle käytännön testaustoteutukselle.

Testaussuunnitelma, -menetelmät sekä dokumentointiapa myös pohjautuu työn tekijän omaan analyysiin ja tuntemukseen pelin arkkitehtuurista sekä sen haavoittuvista kohdista, eikä suoraan nojannut mihinkään valmiiseen testauskehikkoon tai ulkoiseen ohjeistukseen.

3.2.1 Testaussuunnitelma ja testattavat komponentit

Suunnitelma keskittyy niihin komponentteihin, joiden kautta käyttäjä on vuorovaikutuksessa järjestelmän kanssa, ja näiden lisäksi pisteisiin sovelluksessa, joissa syöte ja viestit käyttäjältä voivat vaikuttaa pelin kulkuun tai eheyttä uhkaaviin tekijöihin. Tämä lähestyminen suunnitelmassa varmistaa, että tekniset sekä loogiset puutteet, jotka mahdollistaisivat huijaamisen tai muun tietoturvaloukkauksen, havaitaan testauksella.

Suunnitelman testattavat osa-alueet voidaan jakaa kolmeen päätasoon:

1. Frontend (Asiakassovellus)

   Kappaleessa 2.7.6 todettiin että asiakassovelluksen tietoturvaa ja huijauksenestoa käsitellään datan integriteetin ja pelilogiikan eheyden kannalta. Täten tärkeimpiä testattavia asioita ovat:

   - Onko asiakassovelluksessa ylimääräistä tieota (esim. vastustajan käyttäjänimi tai laivojen sijainti)
   - Voitko asiakassovelluksessa olevia rajoituksia ohittaa käyttäjän toimesta (esim. ampuminen vastustajan vuorolla)?
   - Onko asiakassovelluksella mahdollisuus suorittaa pelin eheyden kannalta epäluotettavia toimintoja ilman palvelimen vahvistusta?

2. Backend (palvelinpuoli)

   Laivanupotuksen palvelinpuoli sisältää kaiken kriittisen pelilogiikan. Testauksen kohteena on täten:

   - REST-rajapintojen autentikointi ja auktorisointi
   - Käyttäjän syötteiden validointi ja virheenkäsittely
   - Pelisessioiden hallinta sekä reiluuden valvonta

3. Viestiliikenne (WebSocket + STOMP ja http)

Manipuloimaton viestiliikenne takaa pelin eheyden ja rehellisyyden. Testeissä pyritään selvittämään:

- Onko viestejä mahdollista siepata, muokata, tai lähettää uudelleen?
- Onko palvelimella tarpeelliset vahvistukset jokaisen viestin kohdalla, viestin vilpittömyydestä ja korrektiudesta?
- Käytetäänkö jokaisen viestien yhteydessä luotettavaa ja turvallista autentikointia?

Näiden kolmen tason testaaminen mahdollistaa kokonaisvaltaisen arvion järjestelmän turvallisuudesta. Lisäksi testauksessa keskitytään erityisesti niihin kohtiin, joissa pelin eheys voisi vaarantua ilman, että käyttäjän toiminta rikkoisi varsinaisia pelisääntöjä — esimerkiksi ampumiskomentojen toistaminen, tunnisteiden väärentäminen tai session tilan manipulointi.

Testauksen aikana ylläpidettiin tarkkaa kirjanpitoa jokaisesta testatusta komponentista, käytetystä menetelmästä ja tuloksista. Tämä dokumentaatio toimii pohjana luvussa 3.3 esitettäville konkreettisille testituloksille ja löydöksille.

3.2.2 Valitut testausmenetelmät ja niiden perustelut

Tietoperustassa (kappale 2.7) esiteltyjä tietoturvatestauksen menetelmiä sovelletaan tässä opinnäytetyössä käytännönläheisesti Laivanupotus-pelin suojausmekanismien arviointiin.

Taulukko 4. Testausmenetelmä ja sen sovellus laivanupotuksessa

TESTAUSMENETELMÄ	Soveltaminen Laivanupotus-pelissä
Penetraatiotestaus	Testataan, voiko REST-rajapintojen kautta ohittaa siirtovuorot (esim. ampua kahdesti peräkkäin), tai manipuloida WebSocket-viestejä ilman oikeaa tunnistautumista.
Input fuzzing	Syötetään HTTP-rajapinnoille väärinmuotoiltuja koordinaatteja ja pelitapahtumia, testaten kuinka palvelin käsittelee virheellistä syötettä laivojen sijoittamisessa ja ampumisessa.

Token-tason manipulointitestaus	Muokataan JWT-tokenin sisältöä, ja testataan estääkö palvelin peliliikkeet, jotka on tehty väärennetyllä tunnisteella.
Asiakassovelluksen integriteetti ja pelilogiikan eheys.	Tutkitaan selaimen kehitystyökaluilla, voiko käyttäjä nähdä esimerkiksi vastustajan laivojen sijainnit tai ohittaa asiakassovellus-puolen siirtorajoituksia ilman palvelimen tarkastusta.
Rajapintojen turvallisuuden arviointi	Arvioidaan laivanupotuksen REST- ja WebSocket rajapintojen autentikointi- ja auktorisointikäytännöt esimerkiksi peliliikkeiden tekemisen ja pelihuoneeseen liittymisen yhteydessä.

3.2.3 Testausympäristön kuvaus

Laivanupotus-videopeli pyörii paikallisessa kehitysympäristössä (localhost) ja testaus suoritettiin tässä samaisessa ympäristössä. Testauksen työasemana toimii tietokone Windows-10 käyttöjärjestelmällä. Seuraavat teknologiat kattavat sovelluksen kokoonpanon:

- Palvelin: Java Spring Boot
- Asiakassovellus: React 18, Vite-kehityspalvelin portilla 5173
- Selain: Mozilla Firefox
- Tietokanta: Docker-sovelluksessa pyörivä PostgreSQL-kontti portilla 5433

Käyttäjäautentikointi hoidettiin JWT-tunnisteiden sekä Spring- viitekehyksen Spring Security kirjastolla. Pelisessioiden simulointiin käytettiin kolmea eri testikäyttäjää, jotka luodaan palvelimen käynnistyessä. Näiden avulla simuloitiin pelitilannetta.

```java
@Configuration
public class StartupConfig {

    @Profile("dev")
    @Bean
    public CommandLineRunner cmdRunner(UserService userService) {
        return args -> {
            System.out.println("Initializing database with test data in dev environment");
            if (userService.findByUsername("player1").isEmpty()) {
                Player player1 = new Player();
                player1.setUserName("player1");
                player1.setPassword("password123");
                player1.setEmail("player1@test.com");
                player1.setRoles("ROLE_USER");
                userService.registerUser(player1);
            }
            if (userService.findByUsername("player2").isEmpty()) {
                Player player2 = new Player();
                player2.setUserName("player2");
                player2.setPassword("password123");
                player2.setEmail("player2@test.com");
                player2.setRoles("ROLE_USER");
                userService.registerUser(player2);
            }
            if (userService.findByUsername("player3").isEmpty()) {
                Player player2 = new Player();
                player2.setUserName("player3");
                player2.setPassword("password123");
                player2.setEmail("player3@test.com");
                player2.setRoles("ROLE_USER");
                userService.registerUser(player2);
            }
        };
    };
```

Kuva 17. Java koodia, joka suoritetaan laivanupotuksen palvelimen käynnistyessä.

Kuvassa 17. Tietokantaan rekisteröidään kolme käyttäjää (player1, player2 ja player3) testaamiseen tarkoitetuilla sähköposteilla ja salasanoilla. Näillä käyttäjillä voi kirjautua laivanupotukseen sen käyttöliittymän kautta.

3.2.4 BURP-suiten konfigurointi paikalliseen kehitysympäristöön

Selaimen (Mozilla Firefox) konfigurointi BURP Suitelle oli teknisesti suoraviivaista: selain asetettiin käyttämään HTTP(S)-välityspalvelinta osoitteessa 127.0.0.1:8080, ja BURP Suiten varmenne asennettiin Firefoxin luotettuihin juurivarmenteisiin. Haasteita kuitenkin aiheutti se, että modernit selaimet ja käyttöjärjestelmät kohtelevat localhost-osoitetta erityistapauksena, eivätkä salli BURP Suiten väliintuloa HTTP(S)-yhteyksiin tähän osoitteeseen ilman lisäkonfiguraatiota.

Ratkaisuksi tähän ongelmaan localhost-osoitteen sijasta käyttöön otettiin aliaksena toimiva nimi local.test, joka ohjattiin käyttöjärjestelmän IP-osoitteita verkkotunnuksiin ylläpitävää hosts-tiedostoa muokkaamalla 127.0.0.1:een.

```
# Copyright (c) 1993-2009 Microsoft Corp.
#
# This is a sample HOSTS file used by Microsoft TCP/IP for Windows.
#
# This file contains the mappings of IP addresses to host names. Each
# entry should be kept on an individual line. The IP address should
# be placed in the first column followed by the corresponding host name.
# The IP address and the host name should be separated by at least one
# space.
#
# Additionally, comments (such as these) may be inserted on individual
# lines or following the machine name denoted by a '#' symbol.
#
# For example:
#
#      102.54.94.97     rhino.acme.com          # source server
#       38.25.63.10     x.acme.com              # x client host

# localhost name resolution is handled within DNS itself.
#   127.0.0.1       localhost normally. Changed to local.test for Burp Testing
#   ::1             localhost
127.0.0.1       local.test
```

Kuva 18. Kuvankaappaus hosts-tiedostosta

Kuvassa 18. vasemmassa alalaidassa IP-osoite (127.0.0.1) ja sille annettu uusi verkkotunnus (local.test). Vanha välityspalvelimen nimi ”localhost” on kommentoitu pois #-merkillä rivin alussa.

Tämä mahdollisti sen, että BURP Suite pystyi estämättä purkamaan liikenteen HTTPS- ja WSS-yhteyksissä, koska liikenne ei enää kohdistunut suoraan localhost-osoitteeseen. Sekä React-kehityspalvelin että Spring Boot -palvelin konfiguroitiin kuuntelemaan osoitteessa local.test, ja samalla React-sovelluksen API-kutsut muutettiin osoittamaan https://local.test:8433.

Tämän konfiguraation myötä kaikki sovelluksen viestiliikenne kulki salattuna ja BURP Suiten tarkasteltavissa olevana myös paikallisesti, mukaan lukien WebSocket-viestit (wss://).

3.2.5 Testien dokumentointi ja raportointi

Penetraatiotestausta (ml. Input fuzzing ja Token-testaus) varten luotiin taulukko, johon dokumentoitiin jokaisen tietoturvatestin kohdalle omalle rivilleen muun muassa testin nimi, tarkoitus, testausmenetelmä. sekä testin tulos ja havainnot. Joidenkin testien kohdalla on myös lisätietoja esim.

palvelimen antamasta vastauksesta, mikäli ne ovat todettu havaintojen todennettavuuden kannalta sitä parantavaksi.

Luvussa 3.4 tätä testien taulukkodokumenttia hyödynnetään yhdistämällä testien tulokset analyysiin. Jokaisen testin löydöksen vaikutuksia, vakavuutta sekä mahdollisia korjaustoimenpiteitä arvioitiin. Dokumentti oli hyvä työkalu parantamaan testauksen toistettavuutta. Testitapauksien yhteyteen lisättiin tarvittava tekninen tieto testin toistamiseen myöhemmin esimerkiksi regressiotestauksen puitteissa. Tämä tekninen tieto sisältää muun muassa käytetyt http-pyynnöt, sekä testien käyttäjätunnukset. Testausdokumentaatio toimii täten kuin lokina testeistä mutta myös ohjeena mahdollisille tulevaisuuden testaajille tai projektin kehittäjille.

Testitapaus	Tarkoitus	Menetelmä / Työkalu	Suoritusympäristö	Tulos	Havainto / Suositus	Lisätiedot
JWT-tokenin manipulointi	Testata, hyväksyykö palvelin muokatun JWT:n	Manuaalinen testi Postmanilla: muokattu payload	localhost, Spring Boot + PostgreSQL	Hylkäys (200 → 401)	Tokenin eheys tarkistetaan oikein. Ei haavoittuvuutta.	Käytetty JWT: muokattu Base64-payload, alkuperäinen signature säilytetty. Lähetetty headerissa: Authorization: Bearer <token>
Siirron uudelleenlähetys	Testata voiko käyttäjä ampua kahdesti peräkkäin ohittaen vuorot	WebSocket STOMP-client, uudelleentoisto samaa siirtoa	localhost, 2 testikäyttäjää	Hylkäys	Serveri tarkistaa vuoron oikein.	Lähetetty STOMP-viesti: /topic/game/{matchId}/make-move käyttäen samaa siirtodataa kahdesti peräkkäin, JWT header mukana
Client-puolen manipulointi	Näkyykö vastustajan laivat selaimen konsolista?	Selaimen devtools (Mozilla Firefox)	React front-end portilla 5173	Ei näkyvyyttä	Ei vuotoa. DOM ei sisällä salaisia tietoja.	Inspect Element > Components / Network. Laivat eivät renderöidy DOMiin eikä niitä ladattu clientin muistissa
Input fuzzing: laiton koordinaatti (-1, 200)	Syötteiden validoinnin testaaminen	HTTP POST /make-move satunnaisilla syötteillä	Postman + käsin syötetty JSON	Hylkäys: 400 Bad Request	Validointi toimii, mutta virheilmoitusta voisi parantaa.	Pyyntö: POST /api/game/{matchId}/make-move, payload: { "x": -1, "y": 200 }, headerissa: validi JWT-token
TLS-yhteyden varmistus	Onko yhteys salattu oikein?	HTTPS- ja WSS-yhteyksien tarkastus selaimessa ja Netstatilla	localhost + selain	OK	Sertifikaatti voimassa (itse allekirjoitettu). Suositellaan virallista sertifikaattia tuotantoon.	Testattu selaimen turvallisuusraportilla ("Connection is secure"). WSS-yhteys muodostettu onnistuneesti porttiin :8443/wss

Kuva 19. Esimerkki kuvankaappaus penetraatiotestauksen Excel-testitaulukosta

Asiakasovelluksen integriteetin ja pelilogiikan eheyden testaamiselle ei tehty omaa taulukkoa sillä testitapaukset eivät ole yhtä selkeästi eroteltavissa kuin penetraatiotestauksessa eikä niitä ole ajallisesti järkevää kirjata ylös. Testien tuloksia varten tehtiin Word-tiedosto.

Rajapintojen turvallisuuden arvioinnissa käytettiin pohjana aiemmassa kappaleessa 3.1.4 esiteltyjä API-rajapintojen taulukoita 2 ja 3.

3.3 Tietoturvatestauksen toteutus

Tässä kappaleessa kuvataan käytännön tasolla, miten Laivanupotus-selainverkkopelin tietoturvatestaus toteutettiin. Testauksen lähtökohtana olivat luvussa 3.2.1 esitetyt ja testattavat komponentit, valitut testausmenetelmät sekä testien dokumentit. jotka perustuivat työn tietoperustassa (luku

2) esiteltyihin uhkamalleihin, teknologioihin ja huijausmahdollisuuksiin. Toteutus jakautuu osioihin, joissa käsitellään erikseen tärkeimmät testausalueet ja niihin liittyvät menetelmät.

Käytännön testauksessa hyödynnettiin sekä manuaalisia (selaimen konsoli) että ohjelmallisia työkaluja (BURP-Suite), ja kaikki testit suoritettiin kappaleessa 3.2.3 kuvaillussa ympäristössä. Testaus kohdistui erityisesti järjestelmän rajapintoihin, käyttäjän syötteiden käsittelyyn, viestiliikenteeseen, tunnistautumiseen sekä asiakassovelluksen haavoittuvuuksien tutkimiseen. Jokaisessa testausosiossa pyrittiin tunnistamaan mahdollisia haavoittuvuuksia, jotka voisivat vaarantaa pelin turvallisuuden, reiluuden tai käyttäjätietojen suojan.

Tavoitteena oli paitsi havaita teknisiä tai loogisia puutteita, myös arvioida kuinka hyvin järjestelmä torjuu tietoturvauhkia ja huijausyrityksiä käytännössä. Testitulokset dokumentoitiin kappaleessa 3.2.5 kuvatuilla tavoilla ja tulokset käydään läpi tarkemmin luvussa 3.4.

3.3.1 Penetraatiotestauksen toteutus ja työkalut

Penetraatiotestaus toteutettiin manuaalisesti käyttäen Burp Suiten proxy- ja repeater-työkaluja. Proxy mahdollisti asiakassovellukselta palvelimelle lähetettyjen HTTP- ja WebSocket-viestien sieppaamisen ja muokkaamisen ennen niiden lähettämistä eteenpäin, kun taas BURP-Suiten repeaterin avulla samoja muokattuja pyyntöjä voitiin toistaa helposti eri muunnelmilla. Testauksen päätavoitteena oli tunnistaa sovelluksesta loogisia ja teknisiä heikkouksia, joita hyökkääjä voisi hyödyntää pelin eheyden tai tietoturvan vaarantamiseen.

Testauksen aikana kävi ilmi, että muut menetelmät – kuten input fuzzing ja JWT-tunnisteiden manipulointitestaus – olivat käytännössä penetraatiotestauksen sateenvarjon alla. Nämä testit kirjattiin samaan testitapausten taulukkoon kuin muutkin penetraatiotestauksen testitapaukset ja suoritettiin yhdessä BURP-Suitella.

Input fuzzing sisältyi testauksen vaiheisiin, joissa HTTP- tai WebSocket-viestien sisältöä (payload) manipuloitiin tarkoituksellisesti virheellisillä tai odottamattomilla arvoilla. Tätä sovellettiin erityisesti tilanteissa, joissa pelaaja sijoitti laivoja laudalle, ampui koordinaatteihin tai kirjautui järjestelmään. Tavoitteena oli selvittää, miten palvelin käsittelee poikkeavia syötteitä ja löytyykö puutteita syötteiden validoinnissa tai virheenkäsittelyssä.

JWT-tokenien manipulointitestaus puolestaan sisälsi tilanteita, joissa käyttäjälle myönnettyä tunnistetta muokattiin. Tämä suoritettiin esim. muuttamalla tokenin ”payloadin” sisältöä ja sen jälkeen

testattiin, hyväksyykö palvelin muokatun tokenin. Tämä auttoi arvioimaan, kuinka tarkasti palvelin validoi tokenin eheyden ja aitouden ennen toiminnon hyväksymistä.

3.3.2 REST- ja WebSocket-rajapintojen turvallisuusarvio

Rajapintoja tutkittiin samanaikaisesti penetraatiotestauksen yhteydessä ja manuaalisesti rajapintojen ohjelmakoodia katselmoimalla.

3.3.3 Asiakassovelluksen analysointi

Asiakassovellusta analysoitiin manuaalisesti käyttämällä selaimen kehitystyökaluja, erityisesti konsolia ja verkkoliikenteen (network) tarkastelua. Testauksen lähtökohtana oli pelata laivanupotusta normaalilla tavalla, mutta samanaikaisesti seuraten ja koittaen manipuloida konsolin kautta sovelluksen toimintaa. Tarkoituksena oli selvittää, voiko asiakassovelluksen kautta saada näkyviin tietoa, jota pelaajan ei kuuluisi nähdä – kuten vastustajan käyttäjätunnus tai laivojen sijainnit – tai voiko pelilogiikkaa manipuloida esimerkiksi suorittamalla toimintoja väärään aikaan, kuten ampumalla vastustajan vuorolla, tai muokkaamalla HTML ja CSS-asetteluja.

3.4 Tulokset ja löydökset

Tässä luvussa raportoidaan empiirisessä osassa (kappale 3.3) suoritetuista testauksista saadut keskeiset havainnot. Aluksi esitellään tunnistetut haavoittuvuudet ja niiden vakavuusasteet, ja lopuksi tarkastellaan ehdotettuja korjaustoimenpiteitä ja arvioidaan, mitkä riskit jäävät vielä jäljelle ja miten niitä voidaan hallita.

3.4.1 Penetraatiotestauksen tulokset

Tässä kappaleessa esitellään testaustaulukko, jossa on testauksen tavoite, menetelmä, tulokset, havainto/suositus sekä lisätietoa toistettavuuden paranatamiseksi.

Taulukko 5 Penetraatiotestauksen, input fuzzinging sekä JWT-tunnisteen manipulaatiotestin tulokset

Tavoite	Menetelmä / Työkalu	Tulokset	Havainto / Suositus	Lisätiedot

Testata, torjuuko palvelin tunnistamattoman käyttäjän pelitoiminnon	Burp Suite – headerin Authorization poistaminen	403 Unauthorized	✅ Palvelin estää pyynnön oikein, virhekoodi tulee muuttaa 403 --> 401 alan yleisten standardien mukaisesti.	POST /api/game/{matchId}/make-move, ei Authorization-headeria
Selvittää, voiko pelaaja tehdä siirron toisen käyttäjän nimissä	Burp Suite – vaihdettu tokenin payload toisen käyttäjän ID:lle	200 OK	❌ Auktorisointi ei toimi täydellisesti. Vain tokenin allekirjoitus varmistetaan, ei muita ominaisuuksia.	JWT decoded, sub vaihdettu toiseksi käyttäjä-ID:ksi
Tarkistaa, onko mahdollista ampua kahdesti peräkkäin/vastustajan vuorolla	Burp Suite – resend samaa POST-pyyntöä	500 Internal server error. Liikeen hylkäys / ei vaikutusta	✅ Vuorojärjestys ja tilatarkistus toimii	Sama JSON-payload ja token, testattu heti peräkkäin
Tarkistaa, onko laivojen asetuksen logiikka suojattu, sopiva virheilmoitus	Burp Suite – sama POST-pyyntö kahdesti	Toisella kerralla epäsuora virheilmoitus.	◈ Epäsuora tarkistus, palvelin estää laivan sijoittamiseen samoilla koordinaateilla, ei palvelimella ei flagia "laivat sijoitettu".	POST /api/game/{matchId}/place-ships, testattu kahdesti
Tarkistaa, onko laivojen uudelleenasetus estetty uusilla koordinaateilla, sopiva virheilmoitus	Burp Suite – sama POST-pyyntö kahdesti, toisella kerralla muunnellulla payloadilla	Toisella kerralla järjetön virheilmoitus.	◈ Epäsuora esto, Java-ohjelmointikieli itsessään estää entiteetin (laivan) luonnin kahdesti samalla id:llä	POST /api/game/{matchId}/place-ships, testattu kahdesti
Tarkistaa voiko vastustajan hyökkäyspintaalaa vähentää poistamalla laivojen koordinaatteja	Burp Suite - POST pyynnän payloadista laivojen koordinaattien poisto	Palvelin hyväksyy laivojen virheelliset koordinaatit	❌ Palvelin ei tarkista että laiva täytää pituudeltaan/leveydeltään saman verran ruutuja	POST /api/game/{matchId}/place-ships
Tarkistaa voiko vastustajan hyökkäyspintaalaa vähentää poistamalla laivoja	Burp Suite - POST pyynnän payloadista laivojen poisto	Palvelin hyväksyy laivojen poissaolon	❌ Palvelin ei tarkista että laiva jokaista laivaa on yksi kappale	POST /api/game/{matchId}/place-ships

Tarkistaa, onko peliin liittyminen pakollinen tilan tarkasteluun	Postman – GET ilman peliin liittymistä	403 Forbidden	✅ Pelaajan peliin kuuluminen tarkistetaan	GET /api/game/{matchId}/gamestate heti pelin luonnin jälkeen
Testata, pääseekö peliin ilman tunnistautumista	Burp Suite – POST ilman JWT-tokenia	403 Forbidden	✅ Autentikointi toimii	POST /api/game/{matchId}/join, ilman Authorization-headeria
Tarkistaa, voiko WebSocketin kautta liittyä ilman tokenia	Burp Suite - Stomp Connect pyynnöstä Authorization Header poistettu	Error: Authorization Header not found	✅ STOMP-yhteys suojattu alusta asti	STOMP CONNECT ilman headeria → palvelin ei luo WebSocket-yhteyttä
Testaa, voiko lähettää siirron peliin johon ei kuulu	Burp Suite – väärä matchId	500 Internal server error - Käyttöliittymässä vuoro vaihtuu, palvelinpuolella ei.	✅ Peli-ID validoidaan	POST /api/game/{matchId}/makemove, Viesti ei aiheuta tilapäivitystä eikä päädy vastaanottajalle
Testaa, voiko vastustajan laudalle ampua ennen kuin peli on alkanut.	Burp Suite –	500 Internal server error	✅ Ampuminen estetään	POST /api/game/{matchId}/makemove, ampuminen ei mene läpi, sillä peli tilan ei salli sitä.

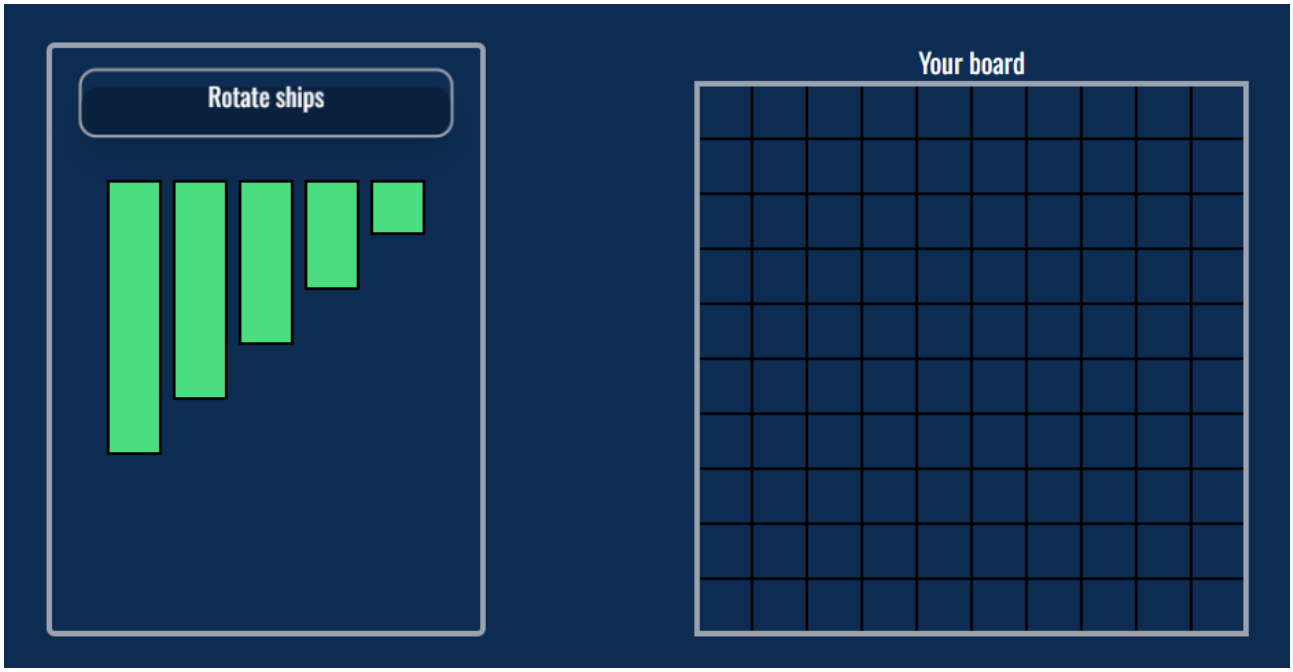
Voiko hyökkääjä pidentää tokenin voimassaoloa	Manuaalinen payload-muokkaus (Base64)	200 Success	❌ Token-signatuuria validoidaan vain repositorion token-avainta vasten, jolla token on allekirjoitettu. Tämä on, joka on jokaisessa tokenissa sama. Validoinnissa täytyy ottaa huomioon muutkin kentät, tokenien yksilöimiseksi.	exp lisätty + 24 h, signature ei vastannut, pyyntö meni silti läpi
Tarkistaa, voiko yhteyden muodostaa väärällä tokenilla	Burp Suite – vaihdettu JWT-token toisen käyttäjän tokeniin	Yhteys muodostuu	❌ Token-signatuuria validoidaan vain repositorion token-avainta vasten, jolla token on allekirjoitettu. Tämä on, joka on jokaisessa tokenissa sama. Validoinnissa täytyy ottaa huomioon muutkin kentät, tokenien yksilöimiseksi.	Suositus: validoida token jo yhteyden muodostusvaiheessa
Testataan selviääkö palvelin liian suuresta numerosta coordinates: "y": -2, "x": 99999999999999 999999999999 9999999999999999 99999999	400 Bad Request	Rajojen ylitys, integer overflow	✅ Viesti estetään Javan toimesta. Virhekoodi 500, kannattaa muuttaa informatiivisemmaksi	/api/game/1/placeships?userId=2
Viallinen laivan pituus "length": "abc"	400 Bad Request	Tyyppivirhe / formatointi	✅ Viesti estetään Javan toimesta. Virhekoodi 500, kannattaa muuttaa informatiivisemmaksi	/api/game/1/placeships?userId=2

Null-varmistus: "matchId": null, "x": 5, "y": 5	400 Bad Request	Puuttuvat arvot / null-check	✅ Viesti estetään Javan toimesta. Virhekoodi 500, kannattaa muuttaa informatiivisemmaksi	/api/game/null/placeships?userId=2
Tyhjävarmistus: "matchId": tyhjä , "x": 5, "y": 5	400 Bad Request	Puuttuvat arvot / null-check	✅ Viesti estetään Javan toimesta. Virhekoodi 500, kannattaa muuttaa informatiivisemmaksi	/api/game/placeships?userId=2
SQL-injektio testi: "matchId": "1; DROP TABLE players"	400 Bad Request / estetty	SQL-injektio (testi backendin validoinnista)	✅ Viesti muutetaan merkkijonoksi, eikä SQL-lukija ikinä suorita sitä.	/api/game/placeships?userId=2
Null byte testi: Käyttäjätunnus: "\0admin"	401 Unauthorized	Null-byte-hyökkäys	✅ Viesti estetään Javan toimesta. Virhekoodi 500, kannattaa muuttaa informatiivisemmaksi	/api/game/login

JSON-syntaksi rikki: { "username": "test" "password": }	400 Bad Request	Parserin sietokyky	✅ Viesti estetään Javan toimesta. Virhekoodi 500, kannattaa muuttaa informatiivisemmaksi	/api/game/login
Liian suuri JSON (esim. 5 MB tekstiä kenttään)	413 Payload Too Large / 400 Bad Request	Palvelimen kuormankesto / resurssien hallinta	✅ Palvelimen Spring Boot -viitekehys katkaisee yhteyden automaatisesti	/api/game/login

3.4.2 Havaitut huijausmahdollisuudet pelimekaniikassa

Laivanupotuksen penetraatiotestauksessa oli testitapaus, jossa pelaaja voi hyödyntää palvelimen puutteellista huijauksenestoa. Tämä tuli esiin, kun laivojen sijoittamiseen tarkoitettu http-pyyntö kaapattiin ja sen sisältö muutettiin.



Kuva 20. Rajattu kuvankaappaus laivanupotuksen pelaajan näkymästä

Kuvassa 20. Vasemmalla laivanupotuksen ovat laivat. jokainen niistä on yhden ruudun leveä ja aina seuraavaa yhden ruudun pidempi kuin edellinen aina viidenteen asti.

```
[
  {
    "id":"2",
    "type":2,
    "coordinates":[
      {
        "y":2,
        "x":1
      },
      {
        "y":3,
        "x":1
      }
    ],
    "direction":"vertical",
    "isSunk":false,
    "height":2,
    "width":1
  },
  {
    "id":"1",
    "type":1,
    "coordinates":[
      {
        "y":5,
        "x":2
      }
    ],
    "direction":"vertical",
    "isSunk":false,
    "height":1,
    "width":1
  },
  {
    "id":"3",
    "type":3,
    "coordinates":[
      {
        "y":1,
        "x":7
      },
      {
        "y":2,
        "x":7
      },
      {
        "y":3,
        "x":7
```

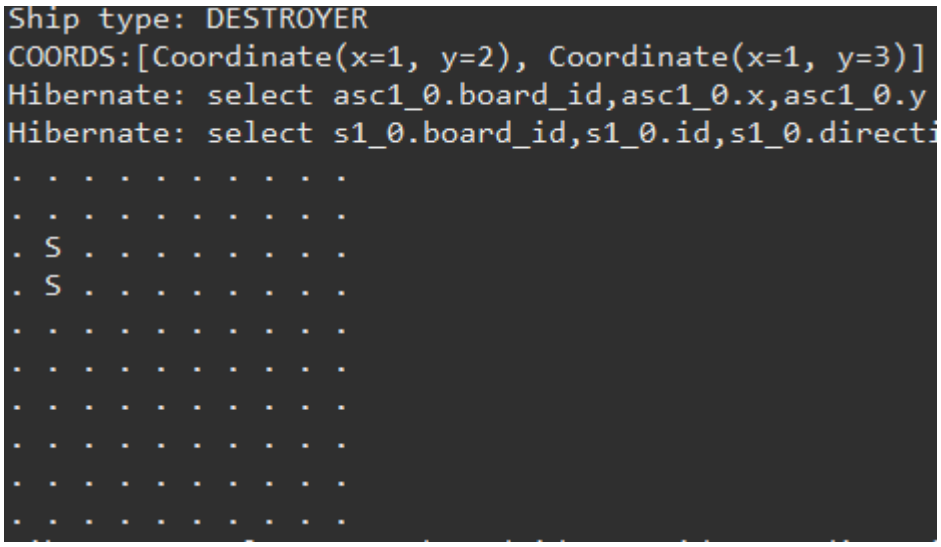

Kuva 21. Burp-Suitessa kaapatun laivojen sijoittamisen http-post viestin sisältöä.

Kuvassa 21. Näkyy taulukko (alkaa ”[”-merkistä) laivoja ja niiden koordinaatteja. Taulukossa on kaikki laivat mutta kuvaa on rajattu.

```
[
  {
    "id":"2",
    "type":2,
    "coordinates":[
      {
        "y":2,
        "x":1
      },
      {
        "y":3,
        "x":1
      }
    ],
    "direction":"vertical",
    "isSunk":false,
    "height":2,
    "width":1
  }
]
```

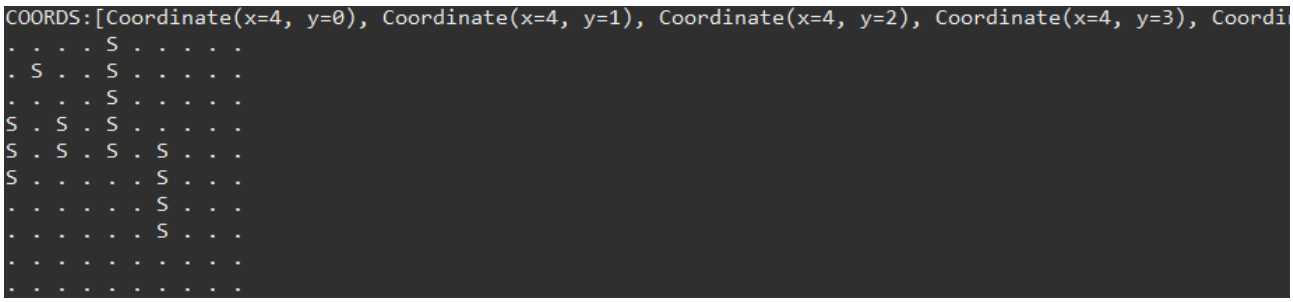

Kuva 22. Muokattu http-post pyynnön sisältö joka esiteltiin kuvassa 20.

Kuvassa 22 näkyy sama pyyntö mutta suurin osa laivoista on poistettu taulukosta jättäen vain yhden laivan jäljelle. Se on pituudeltaan kaksi ja leveydeltään yhden ruudun.



Kuva 23. Kuvankaappaus palvelimen konsolista

Kuvassa 23 näkyy pelitilaa ylläpitävään tietokantaan tallennettu lauta, jossa modifioitu laivan sijainti näkyy kahtena S-kirjaimena.



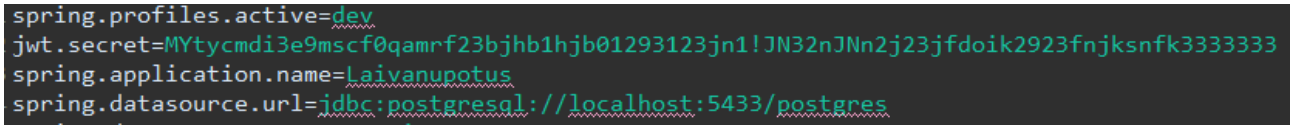
Kuva 24. Kuvankaappaus palvelimen konsolista

Kuvassa 24 näkyvät Vastustajan sijoittamat laivat. Tässä kuvassa on kaikki laivat sijoitettu oikein pituuksin ja leveyksin.

Palvelinpuolelle on implementoitava ominaisuus, joka tarkistaa laivojen sijoituspyynnön mukana tulevan datan oikeellisuuden.

3.4.3 Autentikoinnin ja auktorisoinnin puutteet

Penetraatiotestauksessa havaittiin merkittävä puute autentikoinnin ja auktorisoinnin toteutuksessa. Sovellus käytti JWT-tokenia käyttäjän tunnistamiseen, mutta palvelin tarkisti ainoastaan, että tunniste oli oikein allekirjoitettu palvelimen hallinoimalla salaisella avaimella — ei sitä, kenelle tunniste kuului. Tämä mahdollisti tilanteen, jossa toinen käyttäjä saattoi lähettää pyyntöjä toisen käyttäjän nimissä, kunhan käytössä oli validi, allekirjoitettu tunniste.



Kuva 25. Kuvankaappaus palvelimen application.properties-nimisestä ympäristömuuttujien tiedostosta

Kuvassa 25. ”jwt.secret”-muuttujan arvoksi on asetettu salainen avain, jolla JWT-Token allekirjoitetaan

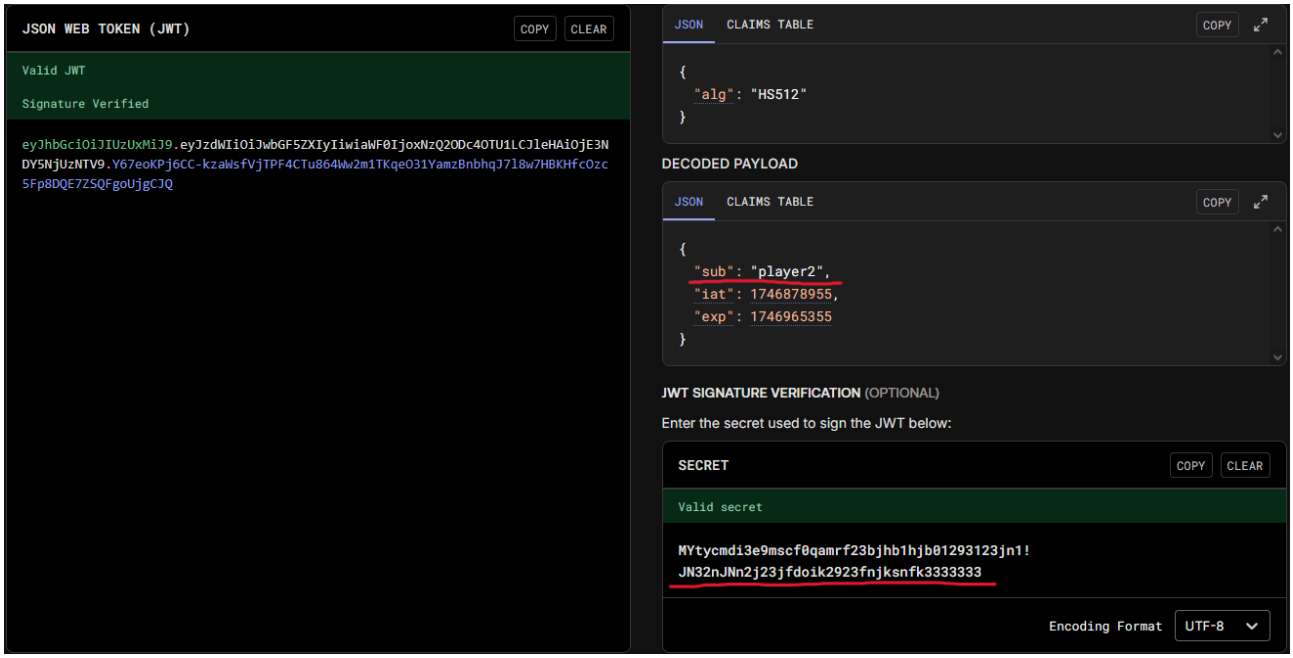
Haavoittuvuus paljastui, kun Burp Suite -työkalulla lähetettiin pyyntö, johon liitettiin toisen käyttäjän tunniste. Palvelin hyväksyi pyynnön, koska validateToken-metodi tarkisti vain, tokenin teknisen eheyden **(Jwts.parser().verifyWith(key).build().parseSignedClaims(token)),** ja allekirjoituksen, mutta ei vertaillut tokenin sub-kentän sisältämää käyttäjätunnistetta pyynnön käyttäjään tai sessioon.

```java
public boolean validateToken(String token) {
    try {
        Jwts.parser().verifyWith(key).build().parseSignedClaims(token);
        return true;
    } catch (Exception e) {
        return false;
    }
};
```

Kuva 26. Java metodi validateToken

Kuvassa 26. Java-metodi, joka varmistaa salaisella avaimella ”key” tokenin allekirjoituksen.

Kyseessä on vakava auktorisoinnin puute, sillä palvelin ei varmista, että tokenin omistaja vastaa pyynnön tekijää. Tämä mahdollistaa mm. muiden käyttäjien pelien manipuloinnin, toimien suorittamisen väärän käyttäjän puolesta sekä potentiaalisen tietojen vuotamisen. Oikea toteutustapa olisi purkaa tokenin sisältö ja varmistaa, että sen sisältämä käyttäjätunnus (esimerkiksi sub) vastaa pyynnössä tai sessiossa käytettyä käyttäjätietoa.



Kuva 27. Kuvankaappaus JWT-Tunnisteiden purkuun tarkoitetulta websivulta (JWT.IO, 2025)

Kuvassa 27. vasemmalla JWT-tunniste ja oikealla punaisella alleviivattuna ”sub” (subject, eli kohde)-kenttä sekä sama salainen avain kuin kuvassa 24 olevassa jwt.secret-muuttujassa.

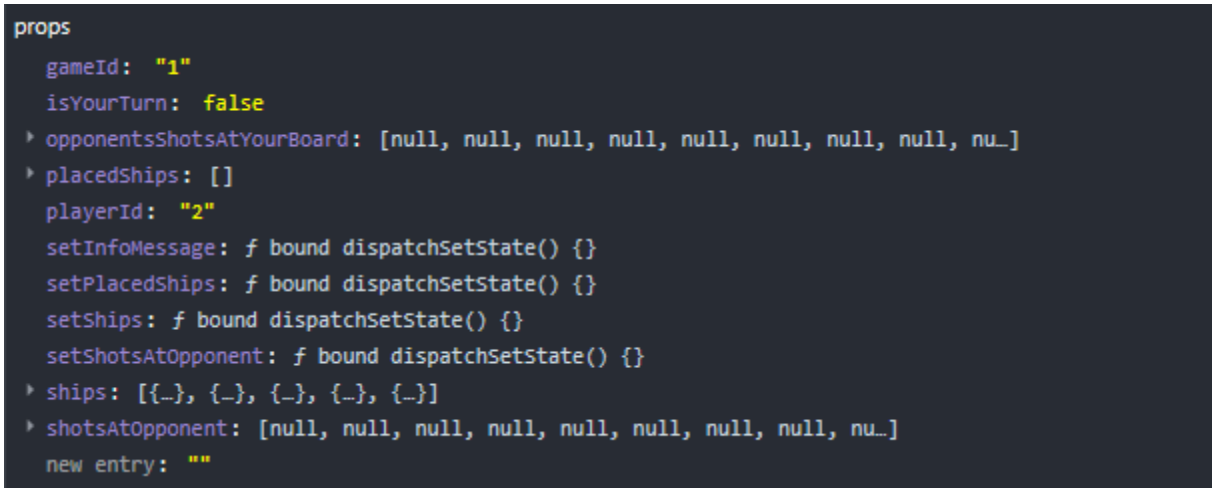
```
public boolean validateToken(String token, String userName) {
    try {
        Jwts.parser().verifyWith(key).build().parseSignedClaims(token);
        String userNameFromToken = getUserNameFromToken(token);

        if (userNameFromToken != null && !userNameFromToken.equals(userName)) {
            return false;
        }
        return true;
    } catch (Exception e) {
        return false;
    }
};

public String getUserNameFromToken(String token) {
    Claims claims = Jwts.parser().verifyWith(key).build().parseSignedClaims(token).getPayload();
    return claims.getSubject();

};
```

Kuva 28. Uusi validateToken-metodi

Kuvassa 28. olevassa metodissa on korjattu JWT-tunnisteen validaatio. Metodi getUserNameFromToken() palauttaa tokenin ”sub”-kentästä käyttäjänimen ja validateToken-metodi vertaa sitä parametrinään olevaan userName-merkkijonoon. (userName on http-pyynnön tehneen pelaajan käyttäjänimi).

3.4.4 Asiakassovelluksen havainnot

Käyttäjällä on täysi hallinta selaimessa ajettavaan JavaScript-koodiin. Esimerkiksi React sovellusten debuggaamiseen virheiden poistoon tarkoitettu React Developer Toolsin tai selainkonsolin avulla käyttäjä voi muuttaa komponenttien sekä muuttujien tilaa helposti. Tämän vuoksi client-puolen toteutukseen ei voida luottaa kriittisissä toiminnoissa, kuten vuoronvalvonnassa tai käyttäjän tunnistamisessa. Kaikki tärkeä validointi ja pelilogiikka tulee suorittaa palvelinpuolella, jotta client-puolen manipulointi ei johda sääntöjen kiertämiseen tai tietoturvaongelmiin.

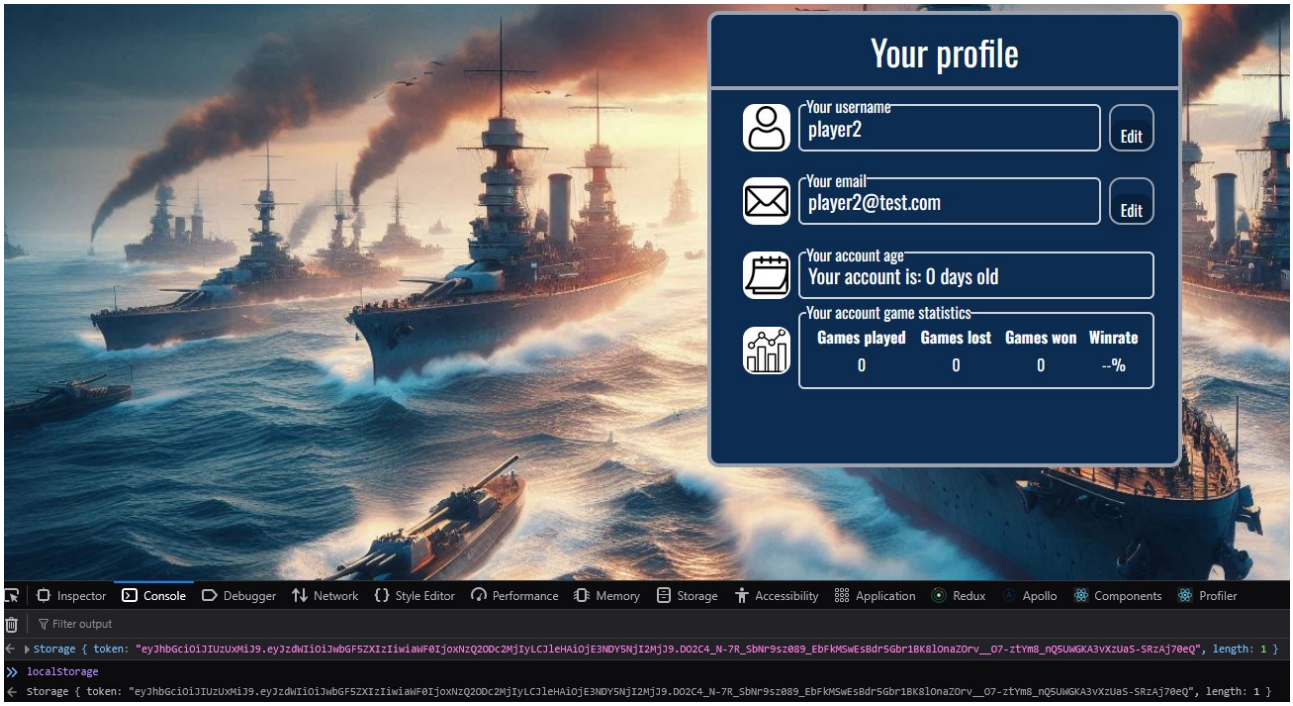


Kuva 29. Kuvankaappaus selaimen konsolista ”React Developer tools”-lisäosan välilehdestä

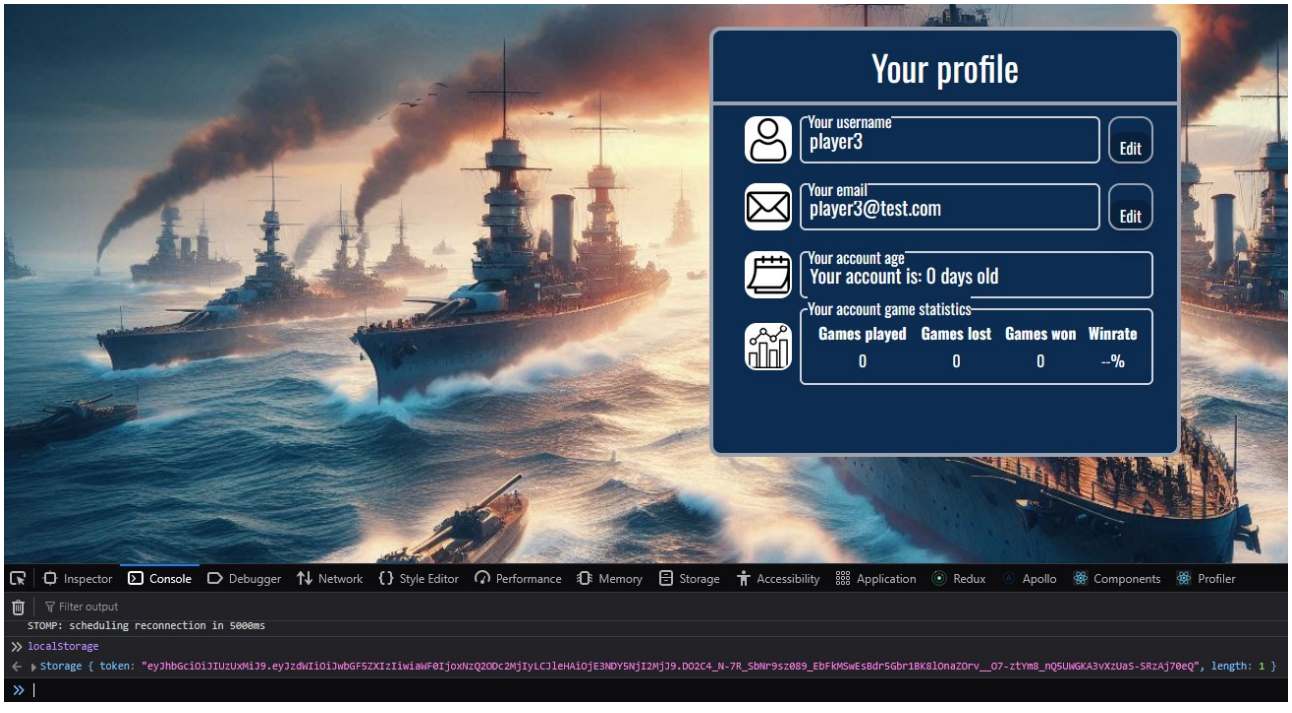
Kuvassa 29. Pelin sisäiset muuttujat sai näkyviin ja muutettua vaivatta.

Testauksessa havaittiin myös vakava client-puolen haavoittuvuus JWT-autentikoinnin toteutuksessa. Kirjautuessa asiakassovellukseen käyttäjätunnistetta edustava tunniste tallennettiin selaimen LocalStorageen, eli sisäiseen tallennustilaan. Kirjautuessa kahdella eri käyttäjätunnuksella eri välilehdissä, molemmat istunnot käyttivät edelleen samaa tunniste. Tämä johti tilanteeseen, jossa toinen käyttäjä toimi toisen valtuuksin — mahdollistaen esimerkiksi pääsyn väärään peliin tai toimintojen suorittamisen toisen puolesta.

Kyseessä oli autentikointiin liittyvä looginen virhe, jonka syynä oli laivanupotuksen kehittämisessä huomiotta jätetty käyttäjätilanne. Vastaavien tilanteiden estämiseksi jokaisen kirjautumisen tulisi tuottaa uusi käyttäjäkohtainen tunniste, vanha tulisi mitätöidä ja tokenin päivitys varmistaa käyttöliittymässä. Lisäksi uloskirjautumisen yhteydessä kaikki tunnistautumistieto tulisi poistaa selaimesta. Toinen vaihtoehto on siirtyä asiakassovelluksessa LocalStoragen käytöstä pois tunnisteiden talennustilana.



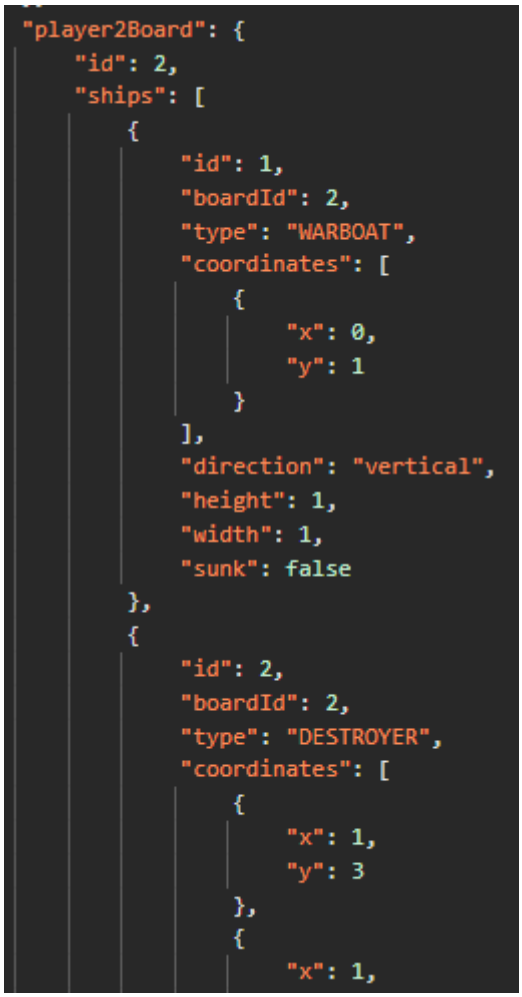
Kuva 30. Kuvankaappaus pelaajan player2 profiilista, JWT-Token näkyvillä selaimen konsolissa



Kuva 31. Kuvankaappaus pelaajan player3 profiilista, JWT-Token näkyvillä selaimen konsolissa

Kuvissa 30. ja 31. Huomaa käyttäjänimien ero ja konsolissa localStoragessa oleva sama JWT-Token.

Lisäksi havaittiin, että mikäli pelaaja (tässä tapauksessa Player1) asettaa laivan onnistuneesti POST-pyynnöllä api-rajapinnan osoitteeseen /api/game/{matchId}/place-ships voi vastaus sisältää ylimääräistä tietoa.



```
"player2Board": {
    "id": 2,
    "ships": [
        {
            "id": 1,
            "boardId": 2,
            "type": "WARBOAT",
            "coordinates": [
                {
                    "x": 0,
                    "y": 1
                }
            ],
            "direction": "vertical",
            "height": 1,
            "width": 1,
            "sunk": false
        },
        {
            "id": 2,
            "boardId": 2,
            "type": "DESTROYER",
            "coordinates": [
                {
                    "x": 1,
                    "y": 3
                },
                {
                    "x": 1,
```


Kuva 32. Kuvankaappaus vastauksesta laivojen sijoittamispyyntöön.

Kuvassa 32. on esimerkki ylimääräisestä tiedosta. Selaimen konsolin Network-välilehdellä http-vastausta tutkimalla, huomataan että se palautti pelaajan 1 laivojen sijainnin ja muun metadatan lisäksi pelaajan 2 laivojen koordinaatit.

http-vastauksesta pitää palvelinpuolella karsia ylimääräinen tieto pois enne sen lähetystä takaisin pelaajalle asiakassovellukseen.

3.4.5 Palvelinpuolen API-rajapinnan turvallisuus ja validointi

Palvelinpuolen REST- ja WebSocket-rajapintojen turvallisuus tuli testattua penetraatiotestauksen ohella. Testien tarkoituksena oli arvioida, miten hyvin palvelin validoi käyttäjien pyyntöjä ja estää luvattoman toiminnan esimerkiksi väärennetyillä tunnisteilla tai ohitetuilla siirtosäännöillä.

Testauksen lisäksi manuaalisen palvelinpuolen koodin katselmoinnin perusteella voidaan todeta, että palvelinpuolen suojausratkaisut perustuvat vahvasti JWT-tunnisteiden validointiin sekä Spring Security -kirjaston käyttöön. REST-rajapinnat oli konfiguroitu siten, että ainoastaan kirjautuneet käyttäjät voivat suorittaa pelin kannalta merkittäviä toimintoja. WebSocket-viestinnässä autentikointi ja auktorisointi tapahtuvat STOMP-headerin mukana kulkevan tokenin avulla, joka tarkistetaan ennen kuin käyttäjä saa yhteyden tai voi lähettää viestejä.

Lisäksi WebSocket-konfiguraatiossa on käytetty mekanismia, joka jokaisen palvelimelle tulevan WebSocket-viestin kohdalla suorittaa tokenin validoinnin, käyttäjän tunnistamisen ja tarvittaessa yhteyden katkaisun, mikäli viesti on puutteellisesti allekirjoitettu tai käyttäjää ei voida tunnistaa. Samalla tarkistetaan, että käyttäjä on oikeutettu tilaamaan tai lähettämään viestejä vain niihin pelikanaviin, joissa hän on osallisena. Nämä tarkistukset havaittiin toimiviksi, kunhan kappaleessa Empiirisen osuuden kohdassa 3.4.3 mainittu JWT-tokenin viallinen validointi, korjataan.

Yhteenvetona voidaan todeta, että rajapintojen validointi ja käyttöoikeuksien hallinta on toteutettu johdonmukaisesti sekä REST- että WebSocket-puolella, ja testien perusteella toteutus estää tehokkaasti yleisimmät väärinkäyttöyritykset.

3.4.6 Korjaustoimenpiteet ja niiden implementointi

Tietoturvatestauksen tulosten perusteella tunnistettiin neljä haavoittuvuutta, joihin laadittiin konkreettiset korjaustoimenpiteet. Osa löydöksistä liittyi suoraan palvelinpuolen validointien puutteisiin, osa taas asiakassovelluksen rakenteeseen ja tiedonhallintaan. Tässä kappaleessa käydään ne tiivistetysti läpi.

1. JWT-tokenin validoinnin korjaus:

Sovelluksen vakavin tietoturvapuutteista liittyi puutteelliseen auktorisointiin, jossa palvelin hyväksyi minkä tahansa allekirjoitetun tokenin ilman, että tarkistettiin, kuuluuko tunniste pyynnön tekijälle. Tämä mahdollisti toimien suorittamisen toisen käyttäjän puolesta. Korjauksena palvelinpuolelle lisättiin logiikka, joka vertaa tokenin sub-kenttää pyynnön yhteydessä käytettyyn käyttäjätunnisteeseen. Näin varmistetaan, että pelitoiminnot suoritetaan vain käyttäjän omilla valtuuksilla.

2. Laivojen sijoittamisen validointi:

Palvelin hyväksyi puutteelliset laivasijoitukset, mikä mahdollisti pelimekaniikan manipuloinnin ja etulyöntiaseman toista pelaajaa kohtaan. Korjauksena toteutettiin tarkistukset, jotka varmistavat, että jokainen laiva esiintyy täsmälleen kerran, oikealla pituudella ja muotoilulla.

3. Konsolin debug-tietojen siistiminen:

Testauksessa havaittiin, että selaimen konsolin ”network”-välilehdelle tulostui tarpeetonta ja arkaluonteista tietoa, mukaan lukien vastustajan laivojen sijainti, mikäli pelaaja asetti laivansa sen jälkeen, kun vastustaja oli asettanut ne. Palvelinpuolella korjauksena oli muokata palautettavan datan muotoa siten, että se palauttaa ainoastaan pelaajalle olennaista tietoja, kun hän lähettää http-post pyynnön laivojen sijoituksesta.

4. Asiakassovelluksen LocalStorage-hallinta:

JWT-tunniste tallennettiin LocalStorageen ilman asianmukaista hallintaa, mikä johti tilanteisiin, joissa eri käyttäjien sessiot sekoittuivat saman selaimen eri välilehdissä. Tämä korjattiin siten, että kirjautuminen aina generoi uuden tokenin, joka korvaa aiemman. Uloskirjautumisen yhteydessä tunniste poistetaan eksplisiittisesti. Myöhemmin harkitaan siirtymistä turvallisempaan tilahallintaan, pois localStoragesta.

Nämä toimenpiteet paransivat huomattavasti sovelluksen eheyttä, reiluutta ja tietoturvaa. Korjausten yhteydessä varmistettiin, että muutokset eivät vaikuttaneet sovelluksen normaaliin toimintaan ja että ne voidaan validoida uudelleentestauksella jatkokehityksen aikana.

3.4.7 Jäljelle jääneet riskit ja niiden hallinta

Testaaminen kattoi kattavasti tietoperustan osassa 2.7 taulukossa 1. luetellut laivanupotukselle olennaiset tietoturvauhat. Suurin riski tulevaisuudessakin laivanupotuksen tietoturvalle sekä huijauksenestolle muodostavat inhimilliset ohjelmointivirheet. Lisäksi laivanupotuksen tekninen toteutus nojaa todella paljon ulkoisiin kirjastoihin ja kehitysalustoihin. Näiden tietoturvapäivitykset ja haavoittuvuudet voivat vaikuttaa suoraan sovelluksen turvallisuuteen, joten jatkokehityksen kannalta on hyvä pysyä ajan tasalla näitä kirjastoja koskevista päivityksistä.

4 Pohdinta

Tässä kappaleessa käydään vielä lyhyesti läpi tärkeimpiä ajatuksia, joita heräsi tämän projektin aikana.

4.1 Tietoturvatestauksen ja huijauksenestonsovellettavuus muihin selainmoninpeleihin

Tässä työssä käytetty tietoturvatestaus ja huijauksenestonvarmistus, joka yhdistää manuaalisen penetraatiotestauksen, input fuzzingin ja asiakassovelluksen manipulointitestauksen, soveltuu hyvin muihinkin selainpohjaisiin moninpeleihin. Useimmat pelit jakavat samankaltaisia arkkitehtuurisia piirteitä, kuten WebSocket-pohjaisen reaaliaikaisen viestinnän, REST-API rajapinnan, johon tehdään http-pyyntöjä sekä JWT-pohjaisen tunnistautumisen ja käyttäjänhallinnan. Tämä tekee vastaavanlaisen testauksen toistettavaksi muissa konteksteissa vähäisin muutoksin.

4.2 Projektin arviointi ja oppimisprosessi

Projekti onnistui tavoitteessaan tunnistaa ja korjata tietoturvaan ja pelin eheyteen liittyviä puutteita. Suurimpina oppimiskokemuksina voidaan pitää manuaalisen testauksen roolia, testityökalujen käyttöä (esim. Burp Suite) sekä tietoturvamekanismien, kuten JWT-tunnisteiden toiminnan syvällistä ymmärtämistä. Lisäksi projektin aikana syventyi ymmärrys siitä, kuinka paljon asiakassovelluksen manipulointi voi vaikuttaa kokonaisuuteen, jos palvelin ei validoi syötteitä oikein. Itse projektiin kuuluvan laivanupotuksen kehityksestä myös ammennettiin paljon oppeja, oivalluksia sekä kokemusta.

4.3 Jatkokehitysmahdollisuudet

Jatkossa peliä voisi kehittää lisäämällä tarkemmat virheilmoitukset, paremmat käyttäjäistuntojen hallintamekanismit ja siirtymällä turvallisempaan tokenien säilytykseen. Lisäksi mahdollisia jatkokehitysalueita ovat mm. skaalautuvuuden testaus, kaverilista, reaaliaikainen pelinsisäinen chat, sekä itse sovelluksen julkaisu julkisessa internetissä kaikkien pelattavaksi.

Lähteet

Ably 2025. The challenge of scaling WebSockets. Luettavissa: https://ably.com/topic/the-challenge-of-scaling-websockets#what-is-vertical-scaling. Luettu: 10.5.2025.

Auth0. Introduction to JSON Web Tokens. Luettavissa: https://jwt.io/introduction. Luettu: 10.5.2025.

Britannica. 2025. Client-server architecture. Luettavissa: https://www.britannica.com/technology/client-server-architecture. Luettu: 10.5.2025.

DataDisk. s.a. Services. Luettavissa: https://www.data-disk.co.uk/html_docs/spring_boot/spring_boot_3.html. Luettu 11.5.2025.

Fette, I. & Melnikov, A. 2011. The WebSocket Protocol. IETF RFC 6455. Luettavissa: https://datatracker.ietf.org/doc/html/rfc6455. Luettu: 10.5.2025.

GeeksForGeeks. 2024. WebSockets protocol and long polling. Luettavissa: https://www.geeksforgeeks.org/websockets-protocol-and-long-polling/. Luettu: 9.5.2025.

GeeksForGeeks. 2025 What is Burp Suite? Luettavissa: https://www.geeksforgeeks.org/what-is-burp-suite/. Luettu 10.5.2025

IBM. 2024. What is information security? Luettavissa: https://www.ibm.com/topics/information-security. Luettu: 10.5.2025.

Imperva. s.a. What is penetration testing? Luettavissa: https://www.imperva.com/learn/application-security/penetration-testing/. Luettu: 9.5.2025.

NCES. 2024. What is considered cheating in a game? Luettavissa: https://www.ncesc.com/gaming-faq/what-is-considered-cheating-in-a-game/. Luettu: 10.5.2025.

OWASP. 2020. Web Service Security Cheat Sheet. Luettavissa: https://cheatsheetseries.owasp.org/cheatsheets/Web_Service_Security_Cheat_Sheet.html. Luettu: 26.4.2025.

OWASP. 2018. JSON Web Token (JWT) Cheat Sheet for Java. Luettavissa: https://cheatsheetseries.owasp.org/cheatsheets/JSON_Web_Token_for_Java_Cheat_Sheet.html. Luettu: 9.5.2025.

OWASP. s.a. a Fuzzing. Luettavissa: https://owasp.org/www-community/Fuzzing. Luettu 10.5.2025.

OWASP. s.a. b OWASP Top 10 Client-Side Security Risks. Luettavissa: https://owasp.org/www-project-top-10-client-side-security-risks/ Luettu 10.5.2025.

PortSwigger. s.a. b. JWT attacks. Luettavissa: https://portswigger.net/web-security/jwt. Luettu: 10.5.2025.

Serban, J. 2025. Demystifying JWT: A Beginner's Guide to JSON Web Tokens. Paul Serbanin henkilökohtainen blogi. Luettavissa: https://paulserban.eu/blog/post/demystifying-jwt-a-beginners-guide-to-json-web-tokens/. Luettu: 10.5.2025.

Tal, L 2024. How to secure a REST API? snyk-yhtiön blogi. Luettavissa: https://snyk.io/blog/how-to-secure-rest-api/. Luettu: 10.5.2025.

Stack Overflow. 2017. Why use websocket and what is the advantage of using it? Luettavissa: https://stackoverflow.com/questions/44898882/why-to-use-websocket-and-what-is-the-advantage-of-using-it. Luettu: 10.5.2025.

Stomp. 2012 a. Simple Text Oriented Message Protocol. Luettavissa: https://stomp.github.io/. Luettu: 9.5.2025.

Stomp. 2012 b. STOMP Protocol Specification, Version 1.2. Luettavissa: https://stomp.github.io/stomp-specification-1.2.html. Luettu: 11.5.2025

Wallarm. 2025. WebSocket protocol. Luettavissa: https://www.wallarm.com/what/a-simple-explanation-of-what-a-websocket-is. Luettu: 10.5.2025.