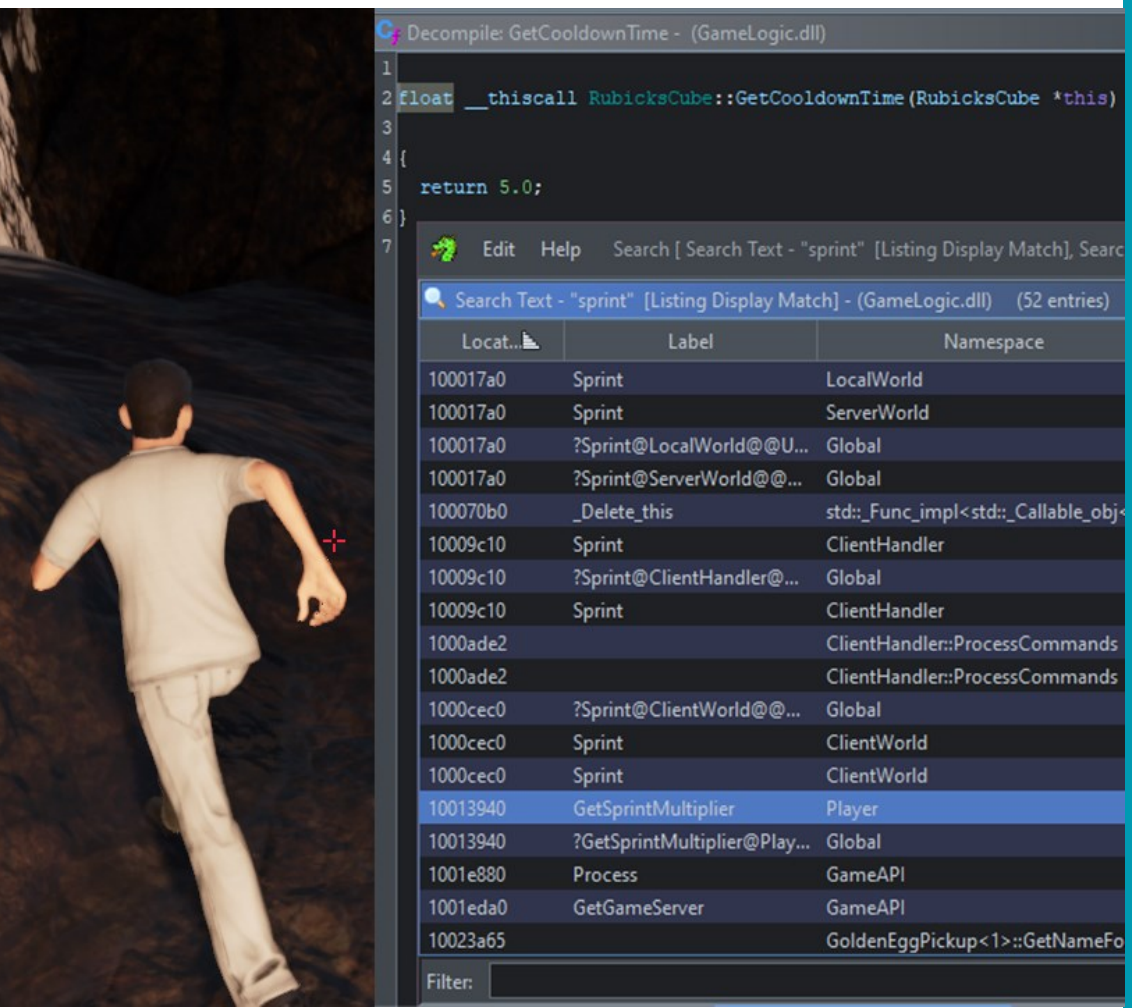
Emil Kilpeläinen

Takaisinmallinnus peleissä ja uudelleentoteutus



Opinnäytetyö

Tradenomi, tietojenkäsittely

Kevät 2024

KAMK • University of Applied Sciences

Tiivistelmä

Tekijä(t): Kilpeläinen Emil

Työn nimi: Takaisinmallinnus peleissä ja uudelleentoteutus

Tutkintonimike: Tradenomi, tietojenkäsittely

Asiasanat: videopelit, takaisinmallinnus, pelipalvelin

Kieli: suomi

Tässä opinnäytetyössä tutkittiin videopelin takaisinmallinnusprosessia ja sen käyttöä pelipalvelimen luomiseen. Työn tarkoituksena oli laajentaa lukijan tietoa takaisinmallinnuksesta ja kuinka sitä voi käyttää peleihin. Erityisesti keskityttiin projektiosuudessa siihen, kuinka takaisinmallinnettua peliä voidaan käyttää oman palvelimen kehittämisessä. Tutkimuksen intohimo syntyi aiemmista kokemuksista takaisinmallinnuksen kanssa ja tahdosta syventää osaamista.

Aluksi käytiin läpi takaisinmallinnuksesta: mikä on takaisinmallinnus, millaista se on videopeleissä, sen laillisuutta, estämismenetelmiä ja projektiosuuden toteutukseen käytettyjä sovelluksia, kuten Visual Studio, Ghidra, Wireshark ja kohdepeli PwnAdventure 3. Nämä antoivat hyvän perustan takaisinmallinnuksen toteuttamiselle. Työssä käsiteltiin myös palvelimia ja palvelinarkkitehtuureja, minkä tarkoitus oli auttaa ymmärtämään niiden roolia ja toimintaa peliympäristössä. Projektiosuudessa takaisinmallinnettiin kohdepeliä ja sen pohjalta kehitettiin omaa pelipalvelinta.

Projektin tulokset osoittivat, että takaisinmallinnuksen avulla voidaan luoda oma pelipalvelin sekä syventää ymmärrystä takaisinmallinetun pelin rakenteesta ja sen sisäisistä toiminnoista. Oma palvelin kykeni emuloimaan alkuperäisen pelipalvelimen toimintoja onnistuneesti, myös mahdollistaen soveltamisen. Tulokset myös osoittivat takaisinmallinnuksen potentiaalin pelien turvallisuuden parantamisessa.

Tutkimuksella sai paremman käsityksen takaisinmallinnuksen hyödyistä sovellusten kehityksessä. Se auttoi ymmärtämään eri tapoja, kuinka estää pelien väärinkäyttöä sekä peleissä huijaamista. Toiminnallisessa osuudessa näki, että takaisinmallinnus on myös hyödyllinen taito ja voimakas työkalu ohjelmoijalle. Työn olisi pystynyt myös tekemään hyödyntäen pääomaisesti Wiresharkia ja takaisinmallintamalla pelin alkuperäisen palvelimen pelin sijasta, mutta pelin takaisinmallintaminen kokeutui monikäyttöisemmäksi ja mahdolliseksi ilman pyörivää palvelinta.

KAMK • University of Applied Sciences

Abstract

Author(s): Kilpeläinen Emil

Title of the Publication: Reverse Engineering and Re-enactment

Degree Title: Bachelor of Business administration, Business Information Technology

Keywords: video games, reverse-engineering, game server

Language: Finnish

This thesis examined the process of reverse engineering video games and its use in creating a game server. The purpose of the work was to expand the reader's knowledge about reverse engineering and how it's applicable to video games. The project part of the thesis specifically focused on how a reverse-engineered game could be used in the development of one's own server. The idea for this research came from previous experiences with reverse engineering and the desire to deepen the knowledge with it.

Initially, the study covered the basics of reverse engineering: what it is, how it applies to gaming, it's legality, prevention, and the applications used in the work, such as Visual Studio, Ghidra, Wireshark and the target application PwnAdventure 3. These provided a solid foundation for starting to reverse engineer. The thesis also covers servers and server architectures, aiming to help the reader understand the server's role and functionality in a gaming environment. In the project part of the thesis, the game was reverse-engineered and used for the development of a custom game server.

The project results showed that reverse engineering can be used to create own servers and deepen the understanding of the reverse-engineered game's structure and internal functions. The custom server was able to successfully emulate the functionality of the original server, also allowing for modifications. The results also showed the potential of reverse engineering in enhancing games' security.

The research provided a better understanding of the benefits of reverse engineering in application development. It provided insight into different methods for preventing misuse and cheating in video games. The project's implementation showed that reverse engineering is also a useful skill and a powerful tool for a programmer. The project could have been implemented using Wireshark and by reverse engineering the original server instead of the game, but reverse engineering the game proved to be more versatile and feasible without a running server.

Sisällys

Termiluettelo

Assembly Konekielen symbolinen esitystapa. Jokaisella prosessoriarkkitehtuurilla on oma assembly-kielensä.

Hookkaus Eng. *Hooking* on tekniikka, jota käytetään ohjelmien funktioiden kaappaamiseen.

Dekompilaatio Eng. *Decompilation* on prosessi, jolla tietokoneella luettava koodi muunnetaan takaisin ihmisille luettavaksi lähdekoodiksi alkuperäisillä kooditoiminnoilla.

Botti Tietokoneohjelma, joka suorittaa sille määrättyjä prosesseja itsenäisesti.

Callback Funktio, joka omaa osoittimen toiseen funktioon ja se funktio kutsuu tätä call-back-funktiota.

Pointer Toiselta nimeltä osoitin, yleisimmin kutsutaan vaan nimellä pointer, muuttuja, joka viittaa keskusmuistissa sijaitsevaan muuttujan arvoon.

Mod (pelissä) Modilla tarkoitetaan modifikaatioita, yleisesti videopeleissä tulkitaan tavoilla muuntaa originaalia rakennetta lisäämällä esim. uusia pelimuotoja tai hahmoja peliin.

Debugger Työkalu, jonka avulla voi jäljittää sovelluksen toiminnallisuutta.

Socket Palvelintermi, joka tarkoittaa linkin luomista palvelimen ja yhteyden luovan välillä.

Välityspalvelin Eng. *Proxy* toimii resurssien välittäjänä asiakkaan ja palvelimen välillä. Välittää resurssit parantaen yksityisyyden sekä turvallisuuden.

Pseudokoodi Ohjelmointikielen kaltaista tekstiä, jonka tarkoitus on olla helposti ymmärrettävässä muodossa. Tätä käytetään auttaakseen ohjelmoijia hahmottamaan koodin toimintalogiikkaa.

Hashing Algoritmi, jota käytetään tietorakenteiden toteuttamisessa.

Tavu Digitaalinen mittayksikkö, joka voi esittää 256 eri arvoa.

1 Johdanto

Tämän opinnäytetyön tavoitteena on käydä läpi takaisinmallinnuksen prosessia, mihin takaisinmallinnuksesta saatua tietoa voidaan käyttää ja kuinka tarkalleen ottaen videopeleistä saanutta materiaalia voidaan käyttää pelipalvelimen luomiseksi.

Takaisinmallinnuksessa tärkeintä on, miten sitä käytetään. Väärissä tilanteissa, missä tekijänoikeudella suojattua peliä yritetään takaisinmallintaa, se voi johtaa lain rikkomiseen. Mutta on olemassa tapauksia, jolloin pelinkehittäjät ovat sallineet takaisinmallinnuksen peleilleen, mikä mahdollistaa oman luovuuden tuomisen kyseiseen peliin ja uusien materiaalien luomisen.

Olen kiinnostunut aiheesta, koska videopelien takaisinmallinnuksesta ja käytöstä löytyy useita samoja vaiheita kuin omien videopelien tuottamisesta. Silloin niitä tarkastelemalla voi myös oppia, kuinka omien pelien takaisinmallinnus ja huijaukset voidaan estää. Keskityn aiheessa lähinnä pelimaailman ympärillä pyörivään takaisinmallinnukseen ja palvelimen käyttöön, sillä tavoitteena on selventää ja käydä läpi videopelien takaisinmallinnuksen perusteet ja kyseisen tiedon käyttöönotto palvelimella. Kyseistä takaisinmallinnusta ja käyttöä käyn läpi vaiheissa termien ja kuvien avustuksella.

2 Takaisinmallinnus

Takaisinmallinnus on olemassa olevan tuotteen, yleisesti laitteen, tietojärjestelmän tai sovelluksen, toiminnallisuuden selvittämistä. Sillä voidaan kaivaa kyseisen tuotteen funktioiden suorituksia ulos, joita voidaan myöhemmin käyttää hyödyksi, jotta saadaan sovelluksen toiminnallisuus selville ja hyödynnetään sitä.

Takaisinmallinnusta käytetään usein yrityksissä, jotta saadaan selville sovelluksen yhteensopivuus toisen käyttöjärjestelmän kanssa, kun alkuperäisen sovelluksen uudelleenrakentaminen toiselle käyttöjärjestelmälle ei ole esimerkiksi enää mahdollista. Takaisinmallinnusta käytetään useasti myös lopetettujen tuotteiden rakenteiden selvittämiseen tai kun haluaa esimerkiksi löytää oman alkuperäisen logonsa, jonka alkuperää ei ole enää muualla tallessa. [1.]

2.1 Videopeleissä

Videopeleissä takaisinmallinnusta voidaan käyttää samoihin käyttöihin kuin normaalisti tietojärjestelmien ja laitteiden kanssa eli tiedon hakemiseen, tiedolla tarkoitetaan pelialustan rakenteen ja toiminnallisuuksien ymmärrystä, ja verkkopaketteihin. ”*Verkkopaketit sisältävät tietoa esimerkiksi toisen pelaajan sijainnista tai saapuvista viesteistä ja niiden sisällöistä.*” [2, s. 24]. Tätä tietoa käytetään eri tarkoituksiin eri tietojärjestelmien ja laitteiden kanssa.

Verkkopeleihin voidaan luoda botteja, jonka avulla voi nopeuttaa eri prosesseja, kuten hahmon kehittämistä. Kaikenlaisten pelien grafiikkoja voi muokata, esim. vaihtamalla käyttöliittymän tekstuureja, muokkaamalla pelin 3D-malleja jne. Joissain tapauksissa, varsinkin yksinpeleissä, hahmojen attribuutit, kuten eri hahmojen liikkumisnopeudet, haetaan salatuista tiedostoista ja muokataan omiin käyttötarkoituksiin. Sen, että jotain tiedostoa käytetään pelissä tietyssä kohtaa, saa selville takaisinmallintaessa peliä ja löytäessä callback-funktion kyseiseen tiedoston polkuun.

2.2 Laillisuus

Tärkeää ennen takaisinmallinnusta on huomioida, että pelin hyväksikäyttö on laitonta, jos se on tekijänoikeudella suojattu tai lähdekoodi on salainen. Monet asiat vaikuttavat takaisinmallinnuksen laillisuuteen, mutta yksinkertaisuudessaan itse takaisinmallinnus, kun se on tehty ohjelman

idean ja periaatteen selvityksenä, on sallittua kyseisen soveltuvan tekijänoikeuslain 25j § mukaan. Tämä laki selventää sen, että ohjelmiston takaisinmallinnus on tehtävä "ohjelman tietokoneen muistiin lukemisen tai ohjelman näyttämisen, ajamisen, siirtämisen tai tallentamisen yhteydessä." [3, l. 25j] Samassa pykälässä mainitaan, että sopimuksen ehto, jolla rajoitetaan 2–4 momentin mukaista käyttöä, on tehoton.

Seuraavassa pykälässä 25k § mainitaan *"Ohjelman koodin kopioiminen ja sen muodon kääntäminen on sallittua, jos nämä toimenpiteet ovat välttämättömiä sellaisten tietojen hankkimiseksi, joiden avulla voidaan saavuttaa yhteentoimivuus itsenäisesti luodun tietokoneohjelman ja muiden ohjelmien välillä, ja seuraavat edellytykset täyttyvät-"* [3, l. 25k].

Edellytyksiin kuuluu, että sen suorittaa henkilö, jolla on oikeus käyttää ohjelman kappaletta, yhteentoimivuuteen tarpeellinen tieto ei ole ollut helposti tai nopeasti aiemmin maininneen oikeuden omistavan henkilön käsissä ja viimeiseksi takaisinmallinnuksen toimenpiteet olivat tarpeellisia yhteentoimivuuden saavuttamiseen. Esimerkki tällaisesta tapauksesta on, että ohjelman käyttämälle tiedostolle ei ole tiedostoformaattia dokumentoitu, jolloin takaisinmallinnuksella kyseisen asian selvittäminen olisi sallittua, jos kyseessä on sellaisen sovelluksen kirjoittaminen, joka lukee kyseistä tiedostoa omassa formaatissaan. [3.]

Näissä lakipykälissä on siis kysessä poikkeussäädöksiä, jotka sallivat sovellusten lukemisen ja dekompilaation tietyin ehdoin. Yleisesti tekijänoikeuksilla suojatulla materiaalilla koodin dekompilaatio ei ole sallittua, jos et ole tekijänoikeuksien omistaja.

Tähänkin löytyy erikoistapauksia, kuten Rockstar Games. Rockstar Games selvensi pelaajilleen omassa artikkelissaan, "*After discussions with Take-Two, Take-Two has agreed that it generally will not take legal action against third-party projects involving Rockstar's PC games that are single-player, non-commercial, and respect the intellectual property (IP) rights of third parties."* [4]. Tämän ansiosta pelaajille on annettu mahdollisuus modata peliä lisäämällä omia hahmoja, ajoneuvoja, aseita yms., kunhan kyseiset asiat kunnioittavat artikkelin lopussa mainittuja asioita, kuten asioiden pitäminen yksinpelissä, jotta voidaan välttää muihin pelaajiin kohdistuvia epäreiluuksia.

2.3 Estäminen

Nyt kun tiedetään, mitä takaisinmallinnuksella voidaan tehdä, voi olla järkevää käydä läpi tapoja estää se, varsinkin verkkopeleissä, joissa modaus voi luoda epäreiluuksia pelaajien kesken. Kun puhutaan henkilökohtaiselle tietokoneelle asennetusta sovelluksesta, on hyvä tietää, että sovelluksen takaisinmallinnusta ei voida kokonaan estää. On kuitenkin tapoja hidastaa sitä ja estää tietyt toiminnallisuudet.

Pelaajan yrittäessä huijata pelissä, he yrittävät etsiä tiettyjä muuttujia ja hankaloittaakseen niiden löytämistä, muuttujat kannattaa rakentaa funktioiden ympärille, jonka avulla arvoja haetaan ja muutetaan. Verkkopelien kohdalla arvojen muuttaminen kannattaa käsitellä palvelimen puolella ja pelaajat voivat vain lähettää pyyntöjä niihin, jotta niiden luotettavuus voidaan aina tarkastaa. Yleisimmät nimitetavat funktioille, millä muuttujia käytetään on ‘Get’- ja ‘Set’-nimisiä: toisella päätetään muuttujan arvo ja toisella haetaan muuttujan arvo. Tämä hidastaa takaisinmallinnusta, sillä tämä luo abstraktiotason, joka voi hidastaa huijaajia selvittämään, miten muuttujia käytetään. [5.]

Joskus huijareiden kohteena ei ole itse pelissä huijaaminen, mutta rakenteen selvittäminen vaikka modien luomiseksi. Joskus näissä tapauksissa takaisinmallintajien pitää vain selvittää, miten hahmot ladataan ja käsitellään pelissä sekä löytyykö niiden tiedostoille jonkinlainen enkryptaus. Tätä tapaa sekä aiempaa tapaa huijata voidaan yrittää estää Windowsin käyttöjärjestelmän työkaluilla. Windowsin Dynamic Link Library:sta (DLL) löytyy pari funktiota ja luokkaa, joita voidaan hyödyntää tässä tilanteessa. [6.]

Yksi näistä on nimeltään **ProcessDebugFlags.** Tätä luokkaa voidaan käyttää tietääkseen, onko taustalla pyörimässä jonkinlainen debuggeri. Kuvassa 1 näkyy esimerkki, kuinka kyseistä luokkaa on käytetty debuggerin löytämiseen. [6.]

```
C++                                                    Shrink ▲
// CheckProcessDebugFlags will return true if
// the EPROCESS->NoDebugInherit is == FALSE,
// the reason we check for false is because
// the NtQueryProcessInformation function returns the
// inverse of EPROCESS->NoDebugInherit so (!TRUE == FALSE)
inline bool CheckProcessDebugFlags()
{
    // Much easier in ASM but C/C++ looks so much better
    typedef NTSTATUS (WINAPI *pNtQueryInformationProcess)
        (HANDLE ,UINT ,PVOID ,ULONG , PULONG);

    DWORD NoDebugInherit = 0;
    NTSTATUS Status;

    // Get NtQueryInformationProcess
    pNtQueryInformationProcess NtQIP = (pNtQueryInformationProcess)
        GetProcAddress( GetModuleHandle( TEXT("ntdll.dll") ),
        "NtQueryInformationProcess" );

    Status = NtQIP(GetCurrentProcess(),
            0x1f, // ProcessDebugFlags
            &NoDebugInherit, 4, NULL);

    if (Status != 0x00000000)
        return false;

    if(NoDebugInherit == FALSE)
        return true;
    else
        return false;
}
```

Kuva 1. Joshua:n artikkelin esimerkkikoodinpätkä, millä tarkistetaan, onko debuggaustyökaluja läsnä [6]

Lisää informaatiota muista tavoista estää debuggaus ja takaisinmallinnusta löytyy Joshua Tully:n artikkelista, vuodelta 2008, “An Anti-Reverse Engineering Guide” [6].

2.4 Projektiin valitut sovellukset

Käytän projektin dokumentointiin Wordia ja opinnäytetyön säilytykseen OneDriveä. Opinnäytetyön projektiin tarvitaan kohdepeli, jota takaisinmallintaa, ja työkaluja, jotka avustavat takaisinmallinnuksessa. Valitsemani sovellukset takaisinmallinnukseen eivät tietenkään ole pakollisia, mutta hyvin suosittuja, yleisiä, ja ne ovat olleet aiemmin omassa käytössäni.

2.4.1 PwnAdventure

PwnAdventure on Rusty Wagnerin tekemä peli, jossa tarkoitus on hyödyntää pelin tahallisia heikkouksia huijaamiseen. Pelin alkuperäinen käyttötarkoitus oli tehdä se Ghost in the Shellcode -kyberturvallisuuskilpailua varten, mutta kilpailun jälkeen peli ja palvelimen tiedostot on jätetty julkiseksi. Peli on hyvä kohde takaisinmallinnusprojektille, koska se on luotu pelin heikkousten

hyödyntämistä varten. [7.] Julkisia pelejä, kuten World of Warcraft, ei voi ottaa tällaisen projektin kohteeksi, koska omien palvelimien luontiin ei ole lupaa [8].

Ghost in the Shellcode -kilpailu on “Capture the Flag” -haaste. “Capture the Flag” tarkoittaa suomeksi lipunryöstöä. Kyberturvallisuudessa se kuitenkin tarkoittaa kilpailua, jossa löydetään sovellusten tai verkkosivujen heikkouksia. Traficomin kyberharjoitusohjeesta lainaten: *”… tavoitteena on etsiä tietoteknisistä järjestelmistä ”lippuja”, jotka kerryttävät kilpailijan tai joukkueen pistesaldoa. Liput voivat olla esimerkiksi tietynlaisia merkki- ja kirjainjonoja, jotka harjoittelija syöttää pisteytysjärjestelmään ne löydettyään.”*. [9.]

Lisää informaatiota pelistä ja pelin lataamisesta löytyy PwnAdventuren verkkosivuilta [7].

2.4.2 Microsoft Visual Studio

Visual Studio on Microsoftin kehittämä ohjelmointiympäristö. Se tukee useita ohjelmointikieliä ja on käytännöllinen ohjelmoidessa, koska se on toiminnallisuuksiltaan rikas. [2, s. 14.]

Käytän Visual Studiota, koska olen oppinut koodaamisen sen ympärillä ja opetellut sen käyttöliittymän jo aiemmin, ja erittäin monipuolisen toiminnallisuuden takia ei ole tarvinnut vaihtaa toiselle alustalle. Visual Studiolle löytyy myös laaja yhteisö täynnä avuliaita henkilöitä ja YouTubesta löytyy myös hyvin videotutoriaaleja sen käyttämiseen.

2.4.3 Ghidra

Ghidra on tietokoneohjelmien takaisinmallinnustyökalu, joka on tehty avoimella lähdekoodilla. Ghidran kehitteli Yhdysvaltain kansallinen turvallisuusvirasto NSA (*engl. National Security Agency*). [2, s. 15.]

Olen käyttänyt Ghidraa aiemmin, sillä en löytänyt aikanaan muita ilmaisia vaihtoehtoja, jotka tukevat tiettyjä tiedostotyyppejä kuten Ghidra. Hyvällä harjoittelulla Ghidra on varmaan yksi parhaimmista opeteltavista sovelluksista takaisinmallinnukseen. Ghidra on myös hyvin kehittyvä alusta, sillä sovellus perustuu avoimeen lähdekoodiin.

Käytän Ghidraa takaisinmallintaakseen PwnAdventurea, ja myöhemmin näytän myös kuvankaappauksia Ghidrasta.

2.4.4 Wireshark

Wireshark on ilmainen pakettien analysointityökalu, jolla voi sekä kaapata, tallentaa, että analysoida tietoliikennettä. Wiresharkissa on helppo käyttöliittymä, joka toimii Windowsilla, Linuxilla ja macOSilla. [10.]

Tietoliikenteen analysointia varten on monta erinäköistä työkalua, ja haluaisin sanoa työkalun valitsemisen olevan preferenssikysymys. Valitsin Wiresharkin, koska olen käyttänyt sitä ennen. Olen nähnyt Wiresharkista paljon oppaita YouTubessa ja opin pakettien käsittelystä sen kautta. Vastaavia sovelluksia ovat esimerkiksi: Tcpdump, Kismet ja EtherApe.

Wireshark tulee olemaan projektissa vain analysointityökaluna, sillä keskityn projektissa takaisinmallintamaan ja uudelleentuottamaan käytökset oman palvelimen kautta. Wireshark on silti hyödyllinen, sillä jumittaessa palvelinta ohjelmoidessa paketteja voi käydä läpi ja verrata oman palvelimen ja pelin palvelimen välillä ymmärtääkseen, onko palvelimien rakenteet samanlaisia.

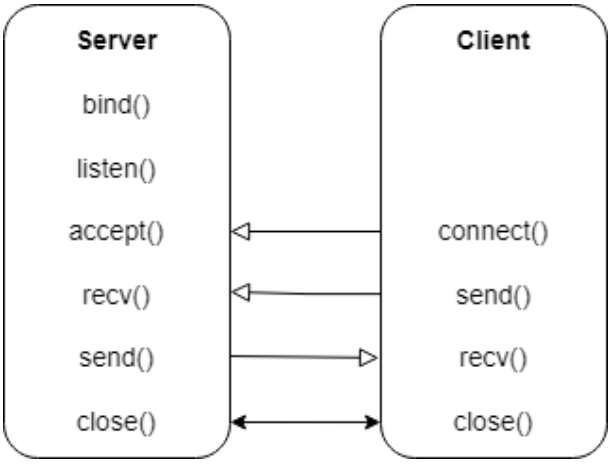
3 Palvelimet ja palvelinarkkitehtuurit

Palvelimia on kahdenlaisia: palvelintietokone ja palvelinohjelmisto. Palvelintietokone on tietokone, jonka käyttötarkoitus on suorittaa palvelinohjelmistoa. Yleisesti se on käyttöjärjestelmältään graafisesti kevyt (esim. Windows Server). Palvelinohjelmisto on se, mihin käyttäjä luo yhteyden sekä kommunikoi verkon välityksellä. Peleissä palvelinohjelmiston tehtävänä on ohjeistaa pelin toimintoja, jotka vaativat verkon kautta kulkevan välityksen, jotta voidaan tallentaa pelaajan tietoja tai kuljettaa informaatiota toisille pelaajille. Verkkopeleissä palvelin voi säilyttää pelaajien sijaintitietoja ja välittää ne kaikille läsnäoleville pelaajille, jotta he näkevät toisensa. [11.]

Pelimaailmassa palvelinarkkitehtuuri voi muuttua riippuen esimerkiksi pelin kulusta, pelaajien lukumäärästä tai pelaajien välisistä interaktioista. Kolme hyvin yleistä esimerkkiä pelipalvelimien arkkitehtuureista ovat Client-Server -, Multi-Server -, sekä Peer-to-Peer -arkkitehtuuri.

Client-Server tarkoittaa palvelimen ja asiakasohjelmiston välistä kommunikointia. Asiakasohjelmisto ja palvelin käyttävät eri sovelluksia. Multi-Serverissä käytetään useita palvelimia, ja sitä voidaan periaatteessa kutsua verkoksi, jossa on Client-Server -yhteyksiä. Multi-Server -arkkitehtuurissa voi lisäksi esiintyä välityspalvelimia. Peer-to-Peerissä ei käytetä erikseen palvelinohjelmistoa, vaan dataa kuljetetaan suoraan pelaajien välillä. P2P-malli ei ole parhain vaihtoehto, vaikka se on ollut tosi yleinen. Mallissa huijaaminen on tehty mahdolliseksi, koska palvelin ei ole prosessoimassa syötteitä ja asiakasohjelmistot pääsevät kirjoittamaan kaiken. Datan kulku riippuu pelaajien verkkoyhteydestä, ja palomuuri voi aiheuttaa käyttäjille ongelmaa kuljettaa dataa. [12.]

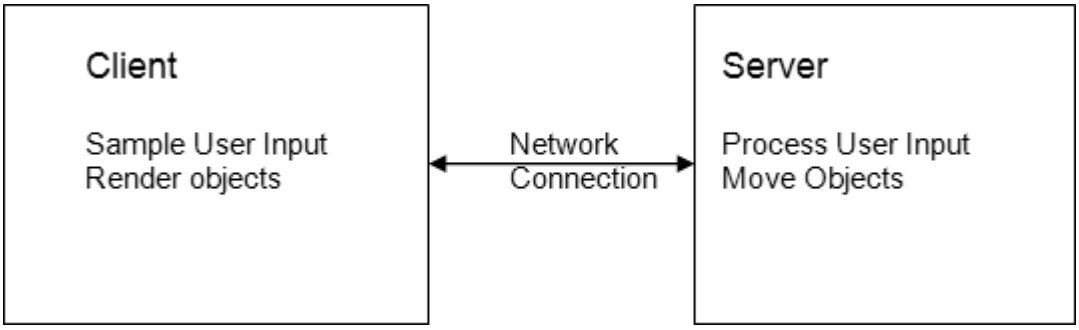
Kuvassa 2 näkyy palvelinohjelmiston ja asiakasohjelmiston välinen toiminta. Palvelin ensimmäiseksi sitoo itsensä bind()-komennolla tiettyyn IP-osoitteeseen ja porttiin, jotta se voi kuunnella tulevia yhteyksiä listen()-komennolla, jonka jälkeen asiakasohjelmistot voi yhdistää siihen, ja kommunikoida lähettämällä ja vastaanottamalla viestejä.



Kuva 2. Oma esimerkki palvelimen ja ohjelmiston välisestä kommunikaatiosta

Valvelta löytyy esimerkki Client-Server -arkkitehtuurin implementoinnista. Valven kirjoittamassa esimerkissä pelaajan sovellus renderöi objekteja ja käsittelee pelaajan syötteet, jotka lähetetään palvelimelle prosessoitavaksi. Palvelin käsittelee käyttäjien antamat syötteet ja lopuksi ilmoittaa muutoksista käyttäjille. [13.]

On myös pelejä, joissa pelillä on suurempi päätösvalta, kuin kuvassa 3 näytetään, ja pelaajat pääsevät tässä tapauksessa manipuloimaan paikallista ympäristöään. Tämän ansiosta peleissä voi huijata suhteellisen helposti takaisinmallintamalla peliä tai muuntamalla paketteja, joita palvelimelle lähetetään. Etu tässä arkkitehtuurissa kuitenkin on, että pelaajat eivät koe mitään viivettä toimissaan, sillä he käsittelevät omat toimensa heti ja lähettävät sen eteenpäin muille. Tätä löytyy myös PwnAdventuresta, esimerkkinä pelaajan juoksemisen nopeus.



Kuva 3. Valven esimerkki Client-Server -arkkitehtuurista [13].

4 Projektin toteutus

PwnAdventuren oma palvelin on suljettu, mutta palvelimen tiedostot on jätetty julkiseen käyttöön, joten voin luoda itse tämän skenaarion, missä pelille löytyy palvelin, jonka kanssa kommunikoida ja voin kalastaa sieltä paketteja, jotta voin katsoa tarvittaessa, miten alkuperäinen palvelin toimii tietyissä kohdissa.

Pelin ja palvelimen välisten pakettien kalastaminen on huomattavasti helpompi prosessi, kuin pelistä funktioiden takaisinmallinnus. Ilman toista palvelinta avuksi pelin takaisinmallinnus palvelimen luomiseksi on suurimmalta osalta kokeilemalla oppimista, mikä voi kokeutua pitkäksi prosessiksi.

4.1 PwnAdventuren takaisinmallinnus

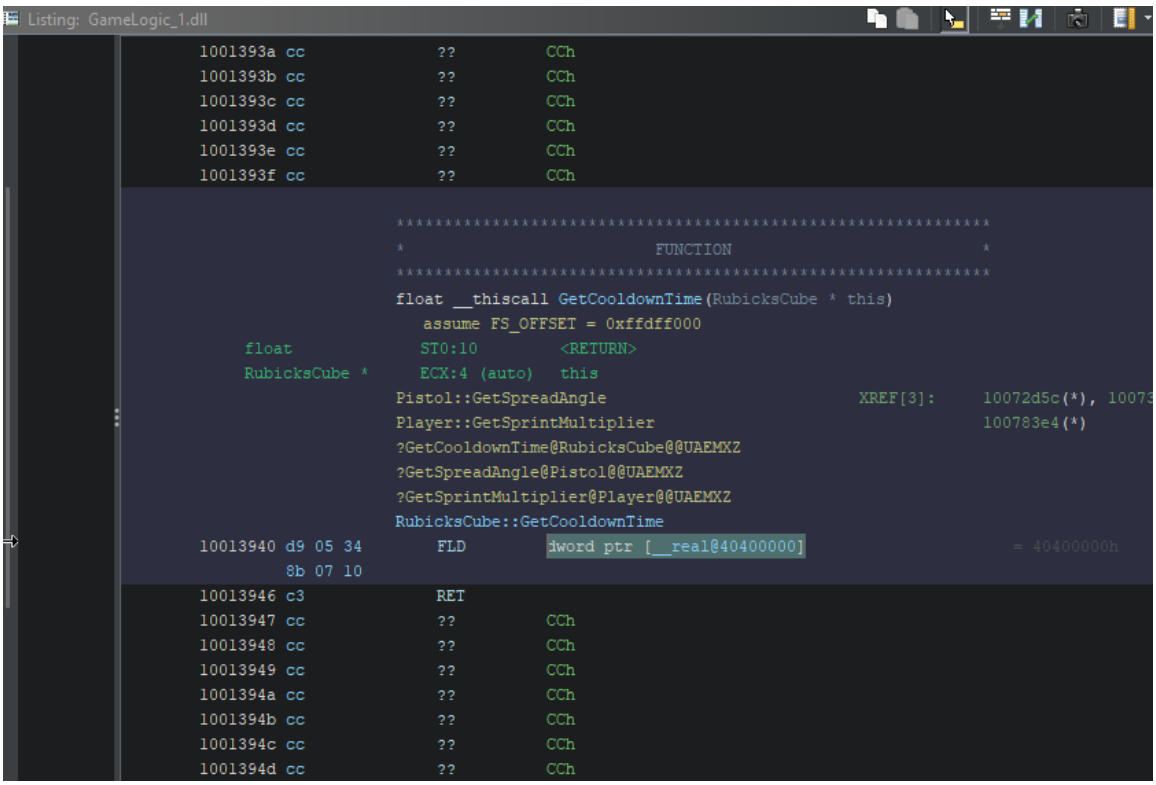
Takaisinmallinnusta suunniteltaessa pitää olla selvät tavoitteet, jotta tietää, mitä yritetään saada aikaan. Esimerkkinä tähän on yrittää kirjautua sisään, ja kun pelistä tulee virheilmoitus, virheen tekstin vastaavaa voi etsiä sovelluksen assemblystä, jolloin kyseinen funktio löydetään ja siitä voi aloittaa takaisinmallintamaan.

Jos takaisinmallinnuksen kohteena on oman palvelimen implementointi, verkkopelin takaisinmallinnus aloitetaan selvittämällä, missä peli haluaa yhdistää palvelimelle. PwnAdventuressa kyseinen tieto haetaan server.ini -tiedostosta. Väliin voi myös luoda välityspalvelimen ohjatakseen pelin eri osoitteeseen.

Palvelinta vielä ajattelematta, pelissä voi takaisinmallintaa paikallisia funktioita sekä muuntaa niiden arvoja, esim. pelaajan juoksemista. Pelaajan juoksemisen nopeus lasketaan paikallisesti pelissä kertoimella, joka haetaan binääritiedostosta nimeltä GameLogic.dll.

Kuvassa 4 osoitteesta 10013940 löytyy funktio, joka palauttaa numeerisen arvon. Tiedän, että muuntamalla tätä numeroa saan eri nopeuden. Kuvassa näkyvä “dword ptr [__real@40400000]” on referenssi osoitteeseen 10078b34 GameLogic.dll -tiedostossa Ghidrassa. Tuosta osoitteessa löysin hexadesimaalisen arvon 0x40400000, jonka desimaalinen arvo on 3.0. Alunperin kokeilin tätä nopeasti, joten otin suoraan alla olevan hexadesimaalin, jonka arvo oli 0x40a00000, desimaalina 5.0. Vaihdoin siis “dword ptr [__real@40400000]” > “dword ptr

[__real@40a00000]”, jolla viittaan tähän 5.0 desimaaliin. Nyt juokseminen pitäisi kertoutua kolmen sijasta viidellä.



Kuva 4. Kuvankaappaus funktiosta GameLogic.dll -tiedoston sisältä käyttäen Ghidraa

Seuraavaksi tallensin GameLogic.dll samaan kansioon, kuin alkuperäisen GameLogic -tiedoston, säästäen alkuperäisestä varmuuskopion ongelmien varalta. Sen jälkeen käynnistin pelin testatakseni sitä ja hahmo liikkui onnistuneesti melkein kaksi kertaa nopeampaa.

Lisää esimerkkejä vastaavista voi löytää katsomalla PwnAdventuren Capture The Flag -kilpailun vastauksia.

Peliä takaisinmallintaessa oikean funktion löytäminen voi koitua hankalaksi, jonka takia on pari eri tapaa varmistaa, että käsittelemme oikeata funktiota. Joissain takaisinmallinnustyökaluissa, kuten Ghidrassa, voi laittaa pysäytyspisteitä, jonka avulla sovellus pysähtyy sille riville, mihin pysäytyspiste on laitettu. Tämä varmistaa, että käsittelemme oikeata kohtaa sovelluksesta. Yksi toinen tapa varmistaa oikea funktio on käyttämällä debuggeria, joka voi nähdä sovelluksen funktioiden kutsuntahistorian, jonka avulla voi löytää helpommin, mitä funktiota käytetään.

4.2 Palvelimen luonti teoriassa

Ensimmäinen vaihe on luoda yhteys pelin ja palvelimen välillä. Palvelinta luodessa pitää tietää, mistä peli löytää IP-osoitteen ja portin, mihin yhdistää, ja vaihtaa ne oman palvelimen osoitteiseen.

Portin tehtävä on toimia päätepisteenä, joka mahdollistaa pakettien lähettämisen palvelimen ja pelin välillä tietoliikenneprotokollan määritysten mukaisesti. Protokollista on huomioida, että niitä on kaksi tavallista tyyppiä, TCP ja UDP. TCP on luotettavampi protokolla, koska se luo yhteyden viestittäjien välillä ja tarkistaa viestien perille kulkemisen, jolloin paketit ja niiden sisältö eivät jää matkan varrelle. Huono puoli TCP:ssä on, että se on hitaampi kuin UDP, sillä UDP ei tarkista saapuvan paketin muuttumattomuutta, eikä takaa sen perille saapumista tai jos paketit tulivat oikeassa järjestyksessä. Syy, miksi UDP:ta silti käytetään, on juurikin sen yksinkertaisuus. Se ei sisällä ylimääräistä tietoa ja sen prosessoiminen on huomattavasti kevyempää. [14.] Yrittäessä yhdistää pelillä palvelimeen, jos peli ei hyppää suoraan kenttään, todennäköisesti kyseessä on ensin TCP-yhteyden luonti.

Kuvassa 5 on yksinkertainen TCP-sovellus, joka vastaanottaa yhteyksiä. Tämä koodinpätkä luo paikallisen palvelimen, joka kuuntelee TCP-yhteyksiä portille 3333. Se luo yhteyden socketin avulla käyttäjän ja palvelimen välille mahdollistaen näiden kahden välisen kommunikoinnin. Jos PwnAdventure yrittäisi ottaa TCP-yhteyttä palvelimen IP-osoitteeseen käyttäen samaa porttia, palvelimelta tulisi viesti “Client Connected” -konsoliin.

```
0 references
static void Main(string[] args)
{
    int port = 3333;
    string IpAddress = "127.0.0.1";
    Socket ServerListener = new Socket(AddressFamily
        .InterNetwork, SocketType.Stream, ProtocolType.Tcp);
    IPEndPoint ep = new IPEndPoint(IPAddress.Parse(IpAddress), port);
    ServerListener.Bind(ep);
    ServerListener.Listen(100);
    Console.WriteLine("Connecting...");
    Socket ClientSocket = default(Socket);
    int counter = 0;
    while (true)
    {
        counter++;
        ClientSocket = ServerListener.Accept();
        Console.WriteLine(counter + " Client Connected");
    }
}
```

Kuva 5. Oma koodinpätkä TCP-palvelimesta

Socket.Listen(int) -funktiolla voidaan määrittää, kuinka monta yhteyttä palvelimeen voidaan luoda. Esimerkkiin laitoin **ServerListener.Listen(100)**, joka tarkoittaa, että palvelimeen voi luoda 100 eri yhteyttä.

Kun peli saa yhteyden palvelimeen ja avaa yhteyden socketiin, peli ja palvelin haluaa tehdä SSL/TLS-kättelyn. Tässä vaiheessa peli lähettää 'ClientHello'-viestin palvelimelle ja palvelin vastaa ServerHello-viestillä. Hello-viesteissä vahvistetaan SSL/TLS-versio ja todennetaan palvelin julkisella avaimella. [15.] Tämän jälkeen yhteys on luotu ja verkkopelin kommunikoinnin rakentamisen voi aloittaa.

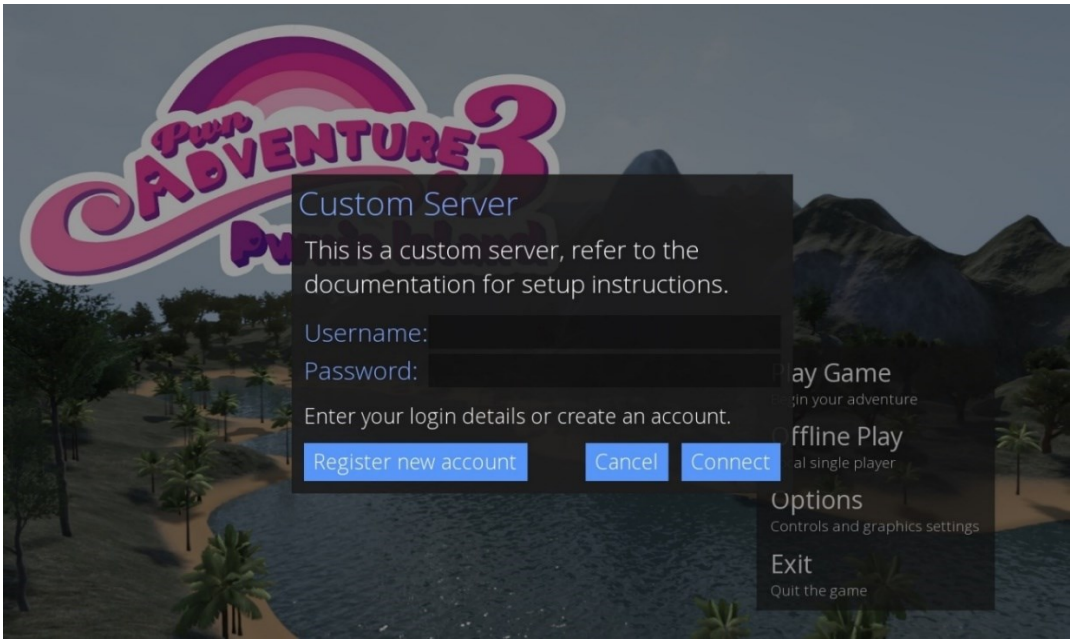
4.3 Palvelimen takaisinmallinnus ja implementointi

Ennen kuin aloitan oman palvelimen implementoinnin, laitan alkuperäisen PwnAdventuren palvelimen pyörimään virtuaalipalvelimeen. Näin voin luoda realistisen skenaarion palvelimen uudelleentoteutukselle.

PwnAdventuren sivuilla lukee, että palvelinta on pyöritetty Ubuntu 14.04 64-bittisellä versiolla, ja suosituksena oli laittaa ainakin 2GB RAM. Virtuaalikoneita voi luoda helposti VirtualBox:in avulla.

Tein virtuaalikoneen, johon asensin Ubuntu 14 -käyttöjärjestelmän ja 4GB RAM-muistia. Siirsin sovelluksen ja tarvittavat pelitiedostot SFTP:n avulla PwnAdventuren sivuilta omalle Ubuntu-palvelimelleni. Virtuaalikoneiden kanssa pitää muistaa yleensä aukaista SFTP:lle portti virtuaalikoneen verkkoasetuksista.

Alkuperäinen palvelin pyörii nyt onnistuneesti. Palvelimeen saa yhteyden ja pelin sisälle pystyy kirjautumaan. Kuvassa 6 näkymä yhdistäessä palvelimelle onnistuneesti.



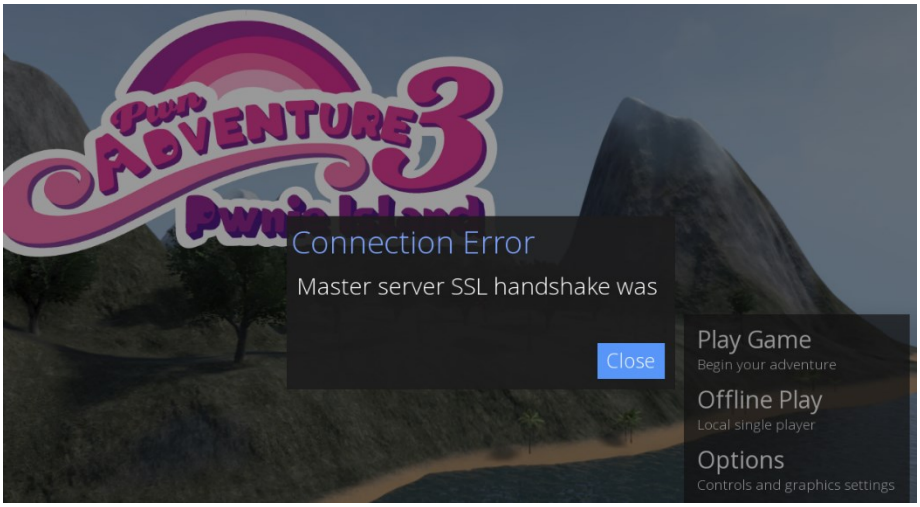
Kuva 6. Kuvankaappaus näkymästä, kun yhdistää alkuperäiselle palvelimelle

Seuraavaksi on aika alkaa ohjelmoida omaa palvelinta. Pohjana käytän aiemmin näytettyä koodinpätkää 5. kuvasta ja palaan takaisin pelin alkuperäiseen palvelimeen tarvittaessa.

Varmistan aluksi, että PwnAdventure yrittää yhdistää samaan osoitteeseen kuin server.ini:ssä. Tämän jälkeen käynnistän pelin ja kokeilen yhdistää sen palvelimeeni. Jos kaikki on mennyt hyvin, vastaan pitäisi tulla onnistunut yhteys.

Palvelimella tilanne näyttää tässä kohtaa hyvältä, mutta pelissä tulee virhe “Master server SSL handshake was invalid”.

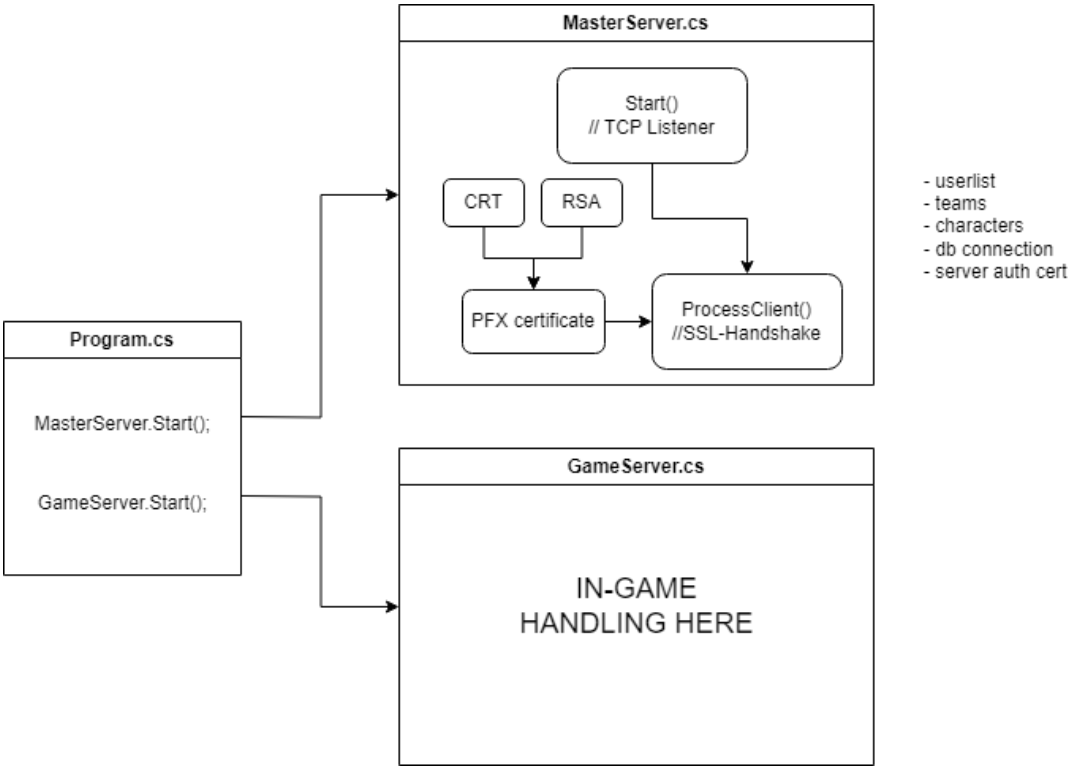
Kuvan 7 virheestä oletan, että peli yrittää seuraavaksi suorittaa kättelyä palvelimen kanssa, mutta en ole vielä implementoinut mitään tapaa vastaanottaa kättelyä ja vastata siihen.



Kuva 7. Pelin ja palvelimen välinen kättelyvirhe

Tulevaisuuden varalta, jos palvelimelle tulee paljon koodia, ohjelma olisi hyvä jo näin alkuvaiheessa hajauttaa useisiin eri prosesseihin. Ongelma ensimmäisessä koodissa on, vaikka se erittäin yksinkertainen onkin, että ohjelma joka kerta jonkun liittyessä lisää laskurin lukuun yhden ennen kuin palaa kuuntelemaan seuraavaa yhteyttä. Pienissä määrin tämä ei koidu ongelmalliseksi, mutta optimaalisempaa on jakaa tämä eri prosesseihin, eli yksi prosessi vastaanottaa pelin yhteyden ja lähettää sen seuraavaan prosessiin vapauttaen jonoa seuraavalla yhteydelle.

Seuraava vaihe on jakaa ohjelma kuvan 8 mukaisesti kahteen eri luokkaan, MasterServer ja GameServer. MasterServer -luokkaan siirretään TCP-yhteyksien vastaanotto. Sitten siirrytään seuraavaan vaiheeseen eli SSL-kättelyn avainten selvittämiseen, jotta palvelin voi todentaa ja vahvistaa pelin kanssa kättelyn.



Kuva 8. Diagrammi, kuinka suunnittelin jakavani palvelimen eri prosesseihin. MasterServerillä pidetään tietokantoja ja käsitellään pelaajien yhdistäminen.

Tutkiessa PwnAdventuren tiedostoja, löysin tiedoston PwnAdventure3\PwnAdventure3\Content\Server\server.crt. Tämä on palvelimen julkinen X.509 PEM -sertifikaatti, jota peli hyödyntää vahvistaakseen palvelimen luotettavuuden. SSL-kättelyssä peli lähettää palvelimelle 'ClientHello'-viestin, johon palvelin vastaa 'ServerHello'-viestillä. 'ServerHello'-viestin mukana pelille lähetetään palvelimen julkinen avain, jota peli katsoo verraten omassa kansiossa olevaan versioon. Jos palvelimen lähettämä sertifikaatti ei vastaa pelillä olevaa versiota palvelimen sertifikaatista, peli hylkää yhteyden. Peli ei myöskään anna yhdistää palvelimeen, jos sertifikaatin poistaa kansiosta, turvatakseen yllä mainitun varatoimen.

Seuraavaksi luodaan oma avainpari palvelimelle ja kopioidaan avainparin julkinen sertifikaatti samaan kansioon kuin aiempi pelillä oleva sertifikaatti. Helppo tapa luoda nämä avaimet on OpenSSL:n avulla.

Yksityisavaimen voi luoda RSA algoritmillä: *openssl genpkey -algorithm RSA -out private.key*. Seuraavaksi pitää tehdä CSR (Certificate Signing Request) allekirjoittaakseen avaimet: *openssl req -new -key private.key -out csr.pem*. Tämän jälkeen voi tehdä itse allekirjoittaman X.509-tyyppisen sertifikaatin käyttäen SHA256 hashing -algoritmia: *openssl x509 -req -days 365 -in csr.pem -sign-*

key private.key -out certificate.crt -sha256. Nyt on yksityisavain RSA-muodossa, ja X.509 -sertifikaatti PEM-muodossa. Seuraavaksi sertifikaatti pitää kopioida samaan kansioon kuin alkuperäinen pelin käyttämä sertifikaatti ja sen jälkeen pitää vielä pakata yksityisavain ja julkinen avain yhteen. Tein tämän tekemällä niistä pfx-tiedoston (Personal Information Exchange): *openssl pkcs12 -export -out cert.pfx -inkey private.key -in certificate.crt*.

Seuraavaksi palvelimen pitää yhdistää peliin SSL:n avulla ja suorittaa kättely, missä käytetään aiemmin luotua sertifikaattia.

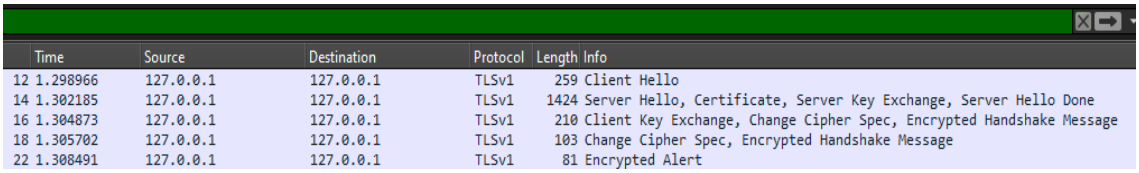
Jos pelille olisi yrittänyt lähettää takaisin alkuperäistä sertifikaattia (server.crt) ilman palvelimen yksityisavainta, konsoliin tulostuisi tämä virhe: “Exception: The server mode SSL **must use a certificate with the associated private key**.”. Eli kättely epäonnistuisi, koska yksityisavainta ei lähetetty sertifikaatin kanssa.

Kuitenkin jos palataan aiemmin luotujen avainten pariin ja yritetään suorittaa kättely kuvan 9 mukaisesti, konsoliin tulee virheilmoitus “Master server is not responding”. Tässä vaiheessa päätin kurkata Wiresharkiin ja varmistaan, missä kohtaa meillä tulee ongelma.

```
private void ProcessClient(TcpClient client) {
	var sslStream = new SslStream(client.GetStream(), false);
	try {
		sslStream.AuthenticateAsServer(X509Certificate, false, true);
	}
	catch (AuthenticationException e){
		Console.WriteLine("Exception: {0}", e.Message);
		if (e.InnerException != null) {
			Console.WriteLine("Inner exception: {0}", e.InnerExcep-
		tion.Message);
		}

		Console.WriteLine("Authentication fail - closing connection.");
	}
	finally {
		sslStream.Close();
		client.Close();
	}
}
```

Kuva 9. Koodinpätkä SSL:n suorittamisesta

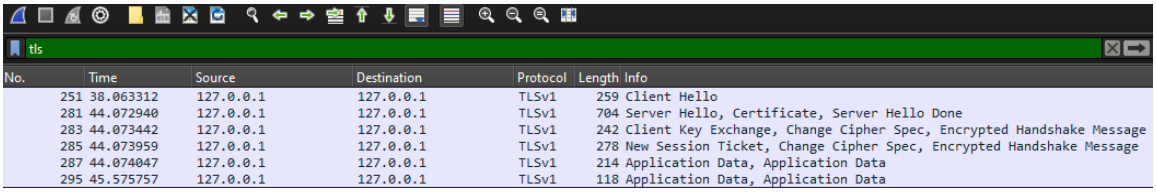
Kuvassa 10 on pelin ja oman palvelimen välinen kättely, josta näkee, että kättely muuten suoriutuu oikein paitsi lopusta puuttuu jonkinlainen paketti, minkä peli haluaisi palvelimelta.



	Time	Source	Destination	Protocol	Length	Info
12	1.298966	127.0.0.1	127.0.0.1	TLSv1	259	Client Hello
14	1.302185	127.0.0.1	127.0.0.1	TLSv1	1424	Server Hello, Certificate, Server Key Exchange, Server Hello Done
16	1.304873	127.0.0.1	127.0.0.1	TLSv1	210	Client Key Exchange, Change Cipher Spec, Encrypted Handshake Message
18	1.305702	127.0.0.1	127.0.0.1	TLSv1	103	Change Cipher Spec, Encrypted Handshake Message
22	1.308491	127.0.0.1	127.0.0.1	TLSv1	81	Encrypted Alert

Kuva 10. Oman palvelimen ja pelin välinen kommunikointi Wiresharkissa

Molemmista kuvista 10 ja 11 huomataan, että palvelin ei katkaise yhteyttä, mutta alkuperäinen palvelin lähettää SSL:n jälkeen lisää dataa pelille, jota se oletettavasti tarvitsee jatkaakseen tästä eteenpäin. Tässä kohtaa molemmat, peli ja palvelin, on kuitenkin suorittanut kättelyn, joten oman sertifikaatin ja avaimen luonti onnistui oikein. Seuraavaksi pitää selvittää, mistä datasta on kyse, joten palataan takaisinmallintamaan peliä ja selvittämään, missä kohtaa kättelyn jälkeen peli yrittää vastaanottaa lisää dataa.



tls

No.	Time	Source	Destination	Protocol	Length	Info
251	38.063312	127.0.0.1	127.0.0.1	TLSv1	259	Client Hello
281	44.072940	127.0.0.1	127.0.0.1	TLSv1	704	Server Hello, Certificate, Server Hello Done
283	44.073442	127.0.0.1	127.0.0.1	TLSv1	242	Client Key Exchange, Change Cipher Spec, Encrypted Handshake Message
285	44.073959	127.0.0.1	127.0.0.1	TLSv1	278	New Session Ticket, Change Cipher Spec, Encrypted Handshake Message
287	44.074047	127.0.0.1	127.0.0.1	TLSv1	214	Application Data, Application Data
295	45.575757	127.0.0.1	127.0.0.1	TLSv1	118	Application Data, Application Data

Kuva 11. Alkuperäisen palvelimen ja pelin välinen kommunikointi Wiresharkissa

Aloitin Ghidrassa seuraamalla MasterServeriin liittyviä funktioita, mutta suurinosa on assemblyä, joten selaamalla ei välttämättä saada mitään irti funktioista helposti. Hain myös koodista string-tekstejä, joissa lukisi “master”, ja heti löytyi referenssejä, joita käydä läpi. Löysin pari tekstiä, mitkä oli tullut meille vastaan pelin sisällä, esim: *“Master server is not responding.”*

Kuvassa 12 näkyy se funktio, mihin peli jää tällä hetkellä jumiin. Funktion rakenteen selvitettyä tiedän, että saan ohjattua pelin seuraavaan askeleeseen, kun pelille lähetetään paketti, jossa ensimmäinen tavu vastaa heksadesimaalia 0x334e5750, tavutaulukkona [80, 87, 78, 51], ja selkokielellä “PWN3”.

```
388 local_78 = (void *)((uint)local_78 & 0xffffff00);
389 std::basic_string<>::assign
390           ((basic_string<> *)&local_78,"Master server is not responding."
391 local_8 = 0x2a;
392 local_4c = 0xf;
```

Kuva 12. Koodinpätkä Ghidrassa pelin tämänhetkisestä virheilmoituksesta

Seuraavaksi peli haluaa toisen tavun, jossa on kokonaisluku. Tämän avulla peli osaa määrittää, mitä pitää näyttää näytöllä. Vaihtoehtoina ovat erilaiset virheilmoitukset, kuten mikä kuvassa 12 näkyy sekä yksi sellainen if-lause, joka tarvitsee vielä enemmän dataa lähettämästä paketista.

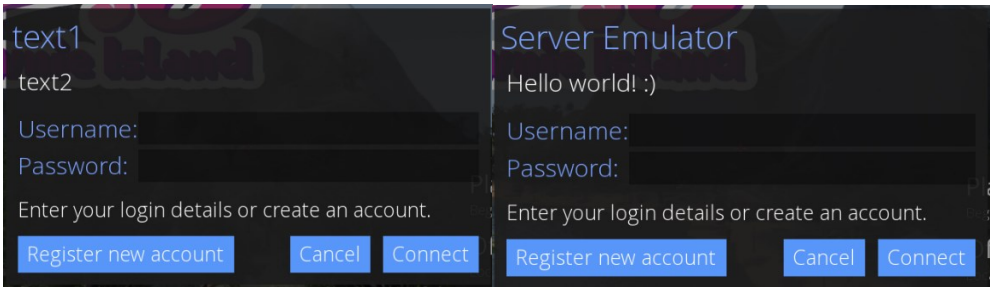
Peli haluaa SSL-kättelyn jälkeen paketin, jossa ilmoitetaan: “PWN3”, 5, 0x00, teksti1 pituus, 0x00, teksti1, 0x00, teksti2 pituus, 0x00, teksti2, 0x00. Luon tälläisen paketin tavuista seuraavaksi palvelimella.

Kuvassa 13 näkyy minun koodipätkä, miten käsittelin SSL-kättelyn jälkeisen paketin. Testatessa huomasin, että nyt pelissä aukeaa sisäänkirjautumisvalikko ja paketissa lähetetyt muuttujat ‘text1’ ja ‘text2’ ovat otsikoita tässä valikossa. Olin syöttänyt näihin aiemmin ‘a’ arvolle “text1” ja ‘b’ arvolle “text2”.

```
public static byte[] CreateWelcomePacket(int version, string a, string b)
{
	var buffer = new List<byte>();
	byte[] welcome = Encoding.ASCII.GetBytes("PWN3");
	buffer.AddRange(welcome);

	buffer.Add((byte)version);
	buffer.Add(0x00);

	var aSize = a.Length;
	buffer.Add((byte)aSize);
	buffer.Add(0x00);
	byte[] aPacket = Encoding.ASCII.GetBytes(a);
	buffer.AddRange(aPacket);

	var bSize = b.Length;
	buffer.Add((byte)bSize);
	buffer.Add(0x00);
	byte[] bPacket = Encoding.ASCII.GetBytes(b);
	buffer.AddRange(bPacket);

	return buffer.ToArray();
}
```

Kuva 13. Funktio, joka rakentaa tarvittavan paketin

Kuvissa 14 ja 15 näkyy, miten aiemman kuvan funktio vaikuttaa käyttöliittymään pelin sisällä.



Kuva 14. ja 15. Sisäänkirjautumisikkunat

Tästä eteenpäin palvelimen ohjelmointi perustuu samaan pelin ja palvelimen välisen kommunkoinnin selvittämiseen Wiresharkin ja Ghidran avulla.

Kokeilin pelissä painaa eri nappeja “Register new account”, “Register”, ja “Connect”, mutta joka syötteestä tuli vastaan “Master server connection lost.”. Hain tätä tekstiä Ghidrasta ja Ghidrassa näkyi useampi referenssi funktioihin, jotka hyödyntää tätä tekstiä. Nämä kaikki funktiot vaikuttaisivat olevan hyödyllisiä, joten näitä takaisinmallinnetaan seuraavaksi. Näiden avulla pääsee toivottavasti jo pelikentän sisälle.

Kuvassa 16 näkyy 6 eri funktiota, jotka kaikki käyttää tätä samaa tekstiä, joten tutkimalla noita seuraavia funktioita löytyy se kohta, mikä vaikuttaa kirjautumisvalikon nappeihin.

Kuva 16. Ghidrasta löytyi teksti “Master server connection lost.” liitettynä 6 eri funktioon

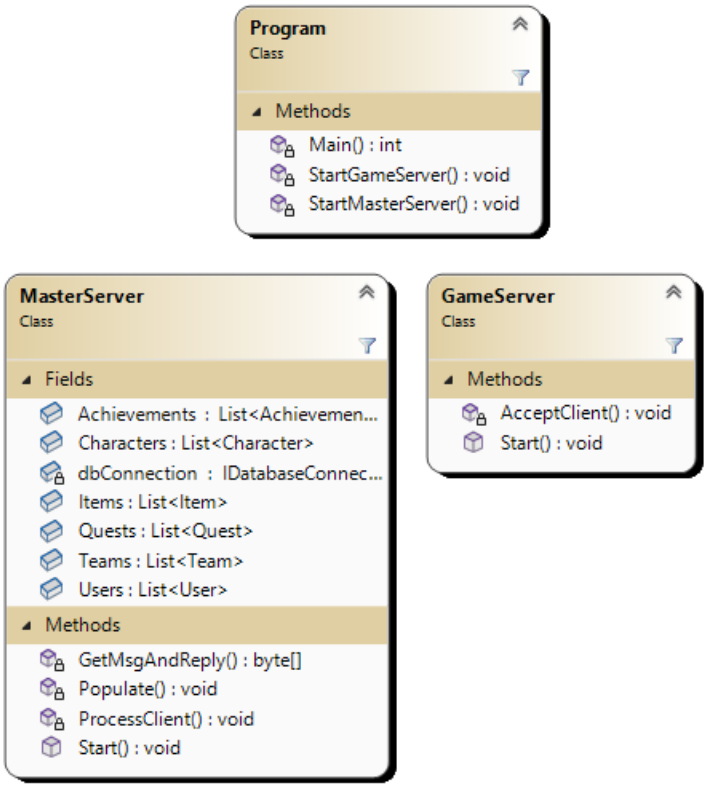
Kuvassa 17 piirsin prosessin, kun menin funktio kerrallaan takaisinmallintamassa peliä siihen pisteeseen, että se yrittää yhdistää GameServeriin. Seuraavaksi pitää siis siirtyä MasterServeristä GameServeriin.

Kuva 17. Seuraavaksi lisäämäni palvelimen toiminnallisuudet järjestyksessä

Aiemmin, kun suunnittelin kuvan 8 diagrammia olin vasta aloittamassa palvelimen ohjelmointia, mutta nyt on jo saatu MasterServerin puoli tehtyä hyvälle tasolle. Kerrataan eli palvelimella pystyy nyt: yhdistämään peliin, tekemään SSL-kättelyn ja välittämään paketteja, kuten rekisteröitymistä, kirjautumista ja hahmon valintaa varten.

Huomioi, että rekisteröintiä tehdessä kannattaa järjestää jonkinlainen tietokanta, minne tallentaa pelin lähettämät pelaajan tiedot.

Kuvassa 18 näkyy palvelimen eri prosessit. Aiemmin näytetyt paketit, kuten kuvassa 13 käyttämäni funktio, on erillään näistä luokista helpottaakseen projektin käsittelyä sen kasvaessa. Esimerkiksi pakettien käsittelyä varten laitoin vastaavat funktiot omaan luokkaan nimeltä PacketManager.cs.



Kuva 18. Päivitetty diagrammi näyttää tämänhetkisen palvelimen prosessien rakenteen

Aloitin GameServer-luokan samalla tavalla kuin MasterServerin kanssa, hakemalla TCP-yhteyden, josta siirrytään toiseen funktioon, missä käsitellään pelaajat. “AcceptClient()” luo istunnon yhteyden saaneen, eli vasta kirjautuneen pelaajan kanssa. Sitten funktio alkaa vastaanottamaan paketteja, mutta tarkistaa myös, onko pelaajan istunto tyhjä tietääkseen, onko kyseessä uusi yhteys ja

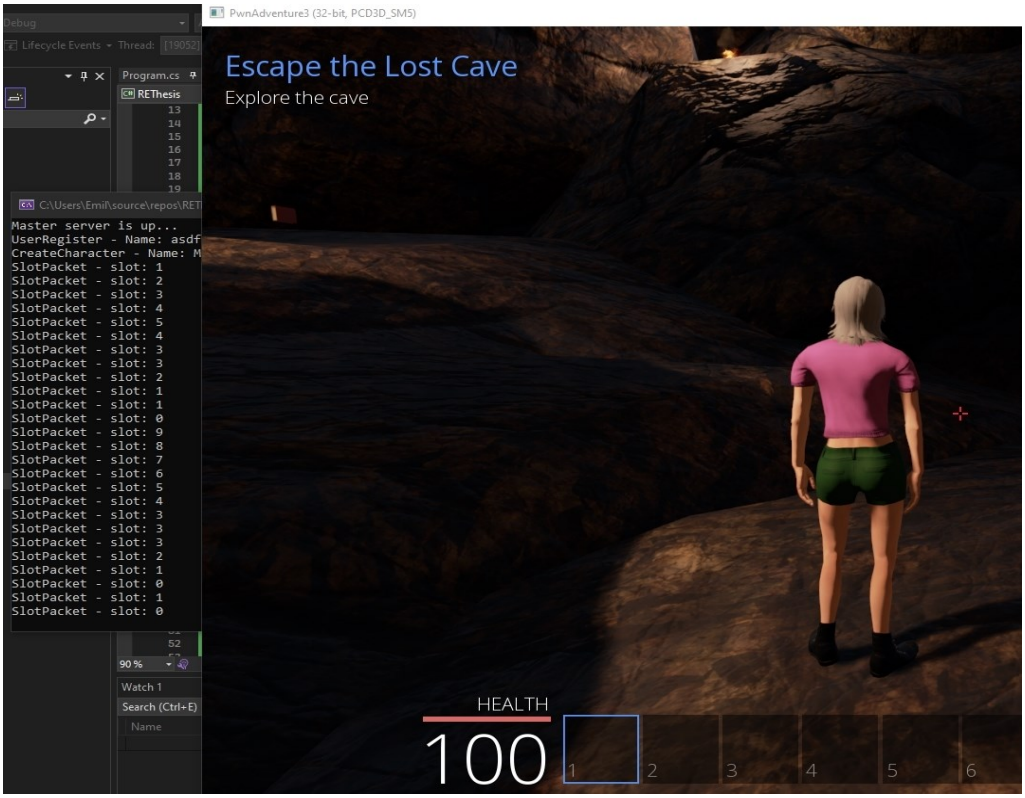
pitääkö pelaajan ‘ClientHello’-paketti prosessoida. Pelaaja lähettää kirjautuessaan peliin palvelimelle tiedon omasta istunnon tunnisteesta (sessionId) sekä hahmon tunnisteesta (characterId). Palvelimella tieto otetaan ylös staattiseen sanakirjaan, jossa pidetään yllä kaikkien pelaajien istuntoja. Nyt jos kirjautuu palvelimelle ja yrittää luoda hahmon, peli syöttäisi virhettä “Socket error 10060”, joka tarkoittaa, että pelin ja palvelimen välillä tuli yhteysaikakatkaisu. Peli ei saanut vastausta palvelimelta omaan komentoon eli tähän ‘ClientHello’-viestiin. Seuraavaksi pitää, samalla tavalla kuin MasterServerin kanssa, kirjoittaa funktio, joka vastaa ‘ClientHello’-viestiin.

Jatkettuani Ghidralla pelin tutkimista huomasin, että pelin haluama paketti lähettää pelaajan hahmon, aloitussijainnin sekä suunnan. Kuvassa 19 näkyy, kuinka toteutin lopulta funktion, joka kasaa paketin.

```
public static byte[] CreateServerHelloPacket(int charId, Vector3 position,
Rotation rotation) {
            var packet = new List<byte>();

            packet.Write32(charId);
            packet.WriteVector3(position);
            packet.WriteRotation(rotation);

            return packet.ToArray();
        }
```

Kuva 19. Paketti, joka lähetetään palvelimelta pelille. Kyseinen funktio antaa pelille tiedon pelaajan hahmon Id:stä, haluamasta sijainnista sekä suunnasta.

Olen nyt onnistuneesti saanut implementoitua oman version palvelimesta. Tästä eteenpäin, mitä enemmän paketteja seuraa ja peliä takaisinmallintaa, sitä enemmän pelistä saa toiminnallisuutta ulos. Kuvassa 20 näkee, että selvitin myös UI:n alaosasta työkalupalkin ja voin nyt liikutella työkalupalkin tavaroiden välillä.



Kuva 20. Kuvankaappaus pelin sisältä palvelimen konsoli auki vasemmassa laidassa

5 Yhteenveto/Lopputulos

Projektin tavoitteena oli takaisinmallintaa peli nimeltä PwnAdventure ja luoda kyseiselle pelille palvelinohjelmisto, joka emuloi alkuperäisen palvelimen käytöstä.

Opinnäytetyö alkoi takaisinmallinnuksen teoriasta, millaista takaisinmallinnus on videopeleissä, mikä takaisinmallinnuksessa on laillista, sekä miten sovellusten tekijät on vältellyt ongelmia takaisinmallinnuksen kanssa. Kirjoittaessa näitä opin enemmän takaisinmallinnuksen estämisestä sekä tarkemmin takaisinmallinnukseen koskevista laeista.

Projektiin siirtyessä aloitin pelin yhdistämisestä omaan palvelimeen. Yhteyden luomiseen kuului sertifikaattien ja avainten kanssa työskentely. Kokeilin erilaisia ratkaisuja SSL-kättelyn kanssa, kuten aiempien avainten käyttöä nähdäkseen, mitä peli syöttäisi ulos. Sen jälkeen projektissa oli vuoro käyttää Ghidraa ja selvittää PwnAdventurea sen avulla. Yhtenä kokeiluna muunsin aluksi pelaajan nopeutta, ja myöhemmin projektissa takaisinmallinsin tärkeämmät asiat, kuten rekisteröinnin, kirjautumisen, hahmon valitsemisen ja peliin pääsemisen.

Projektin lopussa pelissä pystyi rekisteröitymään, kirjautumaan, valitsemaan hahmon sekä liikkumaan kentässä. Laajennusvaraa jäi peliin aika paljon, sillä implementoin asioita vain kenttään saapumiseen asti. Palvelimella ei voi esimerkiksi aktivoida kentän sisällä olevia tapahtumia eikä käyttää pelaajan tavaroita tai taitoja. Sanoisin projektia silti hyvin onnistuneeksi, sillä pelipalvelimeni toteuttaa yleiset pelipalvelimen käyttötarkoitukset, kuten palvelimelle rekisteröinnin, liittymisen. Kaikki oleelliset takaisinmallinnus taidot on tullut käyttöön ja tästä eteenpäin palvelimen kehittäminen vaatisi vain enemmän aikaa käytettynä pelin purkamiseen, jotta ymmärtäisi, mitä lähettää sekä vastaanottaa pelin ja palvelimen välillä. Projekti olisi myös onnistunut ilman Wiresharkia, mutta on kiva oppia entistä enemmän Wiresharkista.

Takaisinmallinnuksella huijaamista peleissä voisi estää järjestämällä tärkeät asiat, kuten laskut ja toimitukset funktioineen, palvelimelle ja minimoida sitä määrää, mitä pelaajalla on valtaa päättää asioista. Tällä tavalla pelaaja voi vaikuttaa vain oman puolen asioihin huijauksilla, mutta palvelin vahvistaa kaiken palvelimella tapahtuvan.

Lähteet

1 Brian Hess. What is reverse engineering? [Internet]. [Viitattu 13.04.2024]. Saatavilla: https://astromachineworks.com/what-is-reverse-engineering/

2 Tomi Hietala. Autonomisesti peliä pelaavan ohjelman suunnittelu ja toteutus. [AMK-opinnäytetyö]. Karelia-ammattikorkeakoulu; 2021. Saatavilla: https://urn.fi/URN:NBN:fi:amk-2021060714749

3 L 8.7.1961/404. Tekijänoikeuslaki. 2022. Saatavilla: https://finlex.fi/fi/laki/ajan-tasa/1961/19610404

4 Rockstar Games. PC Single-Player Mods. [Internet]. [Viitattu 23.3.2024]. Saatavilla: https://support.rockstargames.com/articles/115009494848/PC-Single-Player-Mods

5 Matthew Pritchard. How to Hurt the Hackers: The Scoop on Internet Cheating and How You Can Combat It. [Viitattu 06.10.2022]. Saatavilla: https://www.gamedeveloper.com/design/how-to-hurt-the-hackers-the-scoop-on-internet-cheating-and-how-you-can-combat-it

6 Joshua Tully. An Anti-Reverse Engineering Guide. [Viitattu 23.3.2024]. Saatavilla: https://www.codeproject.com/Articles/30815/An-Anti-Reverse-Engineering-Guide

7 Vector 35. PwnAdventure. [Internet]. [Viitattu 22.11.2023]. Saatavilla: https://www.pwnadventure.com/

8 Blizzard Entertainment. Private WoW Servers. [Internet]. [Viitattu 15.4.2024]. Saatavilla: https://us.battle.net/support/en/article/13639

9 Liikenne- ja Viestintäviraston Kyberturvallisuuskeskus. Kyberharjoitusohje. [Internet]. [Viitattu 23.3.2024]. Saatavilla: https://www.kyberturvallisuuskeskus.fi/sites/default/files/media/file/Kyberharjoitusopas.pdf

10 Wireshark. [Internet]. [Viitattu 04.12.2023]. Saatavilla: https://www.wireshark.org/

11 Ionos. What is a server? [Internet]. [Viitattu 23.3.2024]. Saatavilla: https://www.ionos.com/digitalguide/server/know-how/what-is-a-server-one-term-two-definitions/

12 Matthias Schubert. Distributed Game Architectures. [Internet]. [Viitattu 29.3.2024]. Saatavilla: https://www.dbs.ifi.lmu.de/Lehre/mmmo/sose17/slides/MMMO-3-Network.pdf

13 Yahn W. Bernier. Server In-game Protocol Design and Optimization. [Internet]. [Viitattu 09.10.2022]. Saatavilla: https://developer.valvesoftware.com/wiki/Latency_Compensating_Methods_in_Client/Server_In-game_Protocol_Design_and_Optimization

14 Eino Niemi. Tietoliikenneohjelmointi. [Internet]. [Viitattu 29.3.2024]. Saatavilla: http://www.oamk.fi/~eniemi/TietolOhj/materiaali/TietoliikenneohjV004.pdf

15 Cloudflare. How does SSL/TLS work? [Internet]. [Viitattu 16.12.2023]. Saatavilla: https://www.cloudflare.com/en-gb/learning/ssl/how-does-ssl-work/