Juha Iijalainen

Verkkosovellusten tietoturva

Metropolia Ammattikorkeakoulu

Insinööri (AMK)

Tietotekniikan koulutusohjelma

Insinöörityö

31.5.2016

Metropolia

Tekijä Otsikko Sivumäärä Aika	Juha Iijalainen Verkkosovellusten tietoturva 37 sivua 31.5.2016
Tutkinto	Insinööri (AMK)
Koulutusohjelma	Tietotekniikka
Suuntautumisvaihtoehto	Ohjelmistotekniikka
Ohjaaja	Yliopettaja Jarkko Vuori

Insinöörityön tarkoituksena oli tutkia verkkosovellusten tietoturvaa rakentamalla itse sovellus. Työ toteutettiin käyttäen PHP-ohjelmointikieltä ja MySQL-tietokantaa. Verkkosovellukseksi valikoitui strategiapeli omien mieltymysten pohjalta.

Palveluiden siirtyessä Internetiin on myös niiden tietoturvaan kiinnitettävä yhä enemmän huomiota. Samalla Internetissä käytettävistä teknologioista löydetään jatkuvasti uusia tietoturvahaavoittuvuuksia. Internet on tämän vuoksi toimintaympäristönä kiinnostava ja haastava.

Sovellus rakennettiin suunnittelemalla ensin pelin mekaniikat ja sen jälkeen rakenne. Sovellus toteutettiin hyödyntäen MVC (Model-View-Controller)-malllia, jossa sovelluksen toimintalogiikka on eriytetty näkymien muokkauksen logiikasta. Tietoturvaratkaisuihin perehdyttiin laajasti, ja niistä valittiin mahdollisimman uudenaikaiset ja sovelluksen kokoon nähden sopivimmat toteutukset.

Työssä varmistettiin tietokannan turvallinen käyttö valmisteltujen lauseiden avulla, jolla estetään injektiohyökkäykset. Käyttäjien syötteiden oikeanlaatuisuus varmistettiin erikseen ennen niiden käyttöä toimintalogiikassa. Työssä perehdyttiin käyttäjätietojen hallintaan itserakennetulla hallintalogiikalla. Käyttäjätiedot tallennetaan tietokantaan ja salasanoista muodostetaan tiivistemerkkijot scrypt-funktiolla.

Samalla opittiin sovelluskehityksessä hyödyllisiä työtapoja ja menetelmiä, kuten versionhallinnan käyttö ja kehitysympäristön eriyttäminen tuotantoympäristöstä.

Erinäisten vastoinkäymisten takia projekti viivästyi ja pelilliset ominaisuudet jäivät osin toteuttamtta. Tietoturvapuoli on silti kunnossa. Loppupäätelmä on, että oikein käytettynä PHP-kieli on turvallinen, ja sillä on mahdollista toteuttaa tietoturvallisia verkkosovelluksia.

Avainsanat	PHP, tietoturva

Author Title Number of Pages Date	Juha Iijalainen Security of web-applications 37 pages 31 May 2016
Degree	Bachelor of Engineering
Degree Programme	Information Technology
Specialisation option	Software Engineering
Instructor	Jarkko Vuori, Principal Lecturer

The goal of this project was to study web application security by building an own application from scratch and learning security enhancing solutions by doing so. The main technologies used were the PHP programming language and MySQL database.

This thesis describes the use of the MVC architecture, as well as the procedure of storing user accounts using a self-made authentication system. Passwords were hashed in the study using a secure hashing algorithm. User data were validated, and database access was hardened with the use of prepared statements.

While the game aspects of the project never fully materialized, the choices made regarding security of the whole application were solid: security considerations of the application are up to 2016 standards.

The Internet is an ever changing environment with technologies and methods constantly under testing. As time passes on, new vulnerabilities will be found in the technologies used, and so in the future the security solutions need to be re-evaluated. Thus, it became apparent in the study that no web application can ever be considered as built to last.

Keywords	PHP, security

Sisällys

Lyhenteet

ASP.NET	Active Server Pages dotNet, .NET-sovelluskehyksen komponentti jolla rakennetaan verkkosivustoja.
CSRF	Cross-Site Request Forgery, hyökkäys, jossa sivustolle tunnistautunut käyttäjä suorittaa haluamattaan jonkin toiminnon.
CSS	Cascading Style Sheets, erityisesti HTML-sivujen tyylin muokkaukseen tarkoitettu kieli.
HTML	Hypertext Markup Language, web-sivujen ohjelmointiin tarkoitettu kieli.
MVC	Model–View–Controller, ohjelmistojen arkkitehtuurimalli, jossa ohjelmiston toimintalogiikka ja näkymien rakentaminen on erotettu toisistaan.
MySQL	Relaatiotietokantaohjelmisto, jota käytetään Internet-palveluissa.
PDO	PHP Data Objects, PDP-ohjelmointikielen rajapinta tietokantojen hallintaan.
PHP	PHP: Hypertext Preprocessor, ohjelmointikieli, jota käytetään web-ohjelmoinnissa.
POST	Web-palvelimelle lähetettävä pyyntö, jossa pyynnön parametrit sijatisevat viestin rungossa.
SIM	Subscriber Identity Module, älykortti, jota käytetään puhelimissa.
SSL	Secure Sockets Layer, tiedon salaamiseen käytetty protokolla.
SQL	Structured Query Language, tietokantakäskyjen kirjoittamiseen tarkoitettu kieli.
TCP/IP	Transmission Control Protocol / Internet Protocol, tiedonsiirtoprotokolla, jota käytetään Internetissä.

TKHJ	Tietokannan hallintajärjestelmä. Ohjelmisto, jonka avulla hallinnoidaan tietokantoja.
URL	Uniform Resource Identifier, osoitemerkkijono, joka kertoo jonkin tiedon paikan, esimerkiksi Internet-palvelimen paikan.
XSS	Cross-Site Scripting, hyökkäys, jossa turvalliselle nettisivulle saadaan ujutettua haitallista koodia.

1 Johdanto

Insinöörityön tavoitteena on saada aikaan verkkosovellus, joka on tietoturvallinen ja joka tukee monen käyttäjän samanaikaista vuorovaikutusta toistensa kanssa. Toteutettavan sovelluksen tyypiksi valikoitui strategiapeli omien mieltymysten ja harrastetaustan pohjalta. Projektin kannalta ei kuitenkaan ole tärkeää, ovatko sovelluksen pelilliset ominaisuudet mielenkiintoisia tai onko peliä hauska pelata. Olennaisinta on, että sovelluksen tietoturva on kunnossa.

Verkkosovelluskehitys on parhaimmillaan palkitsevaa ja nopeaa. Samalla yksikin tietoturvaan liittyvä erehdys saattaa asettaa koko sovelluksen alttiiksi vaaralle. Erilaisten Internetpalveluiden määrä kasvaa jatkuvasti, ja niihin tallennetaan myös henkilökohtaista tietoa. Siksi on tärkeää, että verkkosovellusten tietoturva on kunnossa.

Insinöörityössä esittelen verkkosovellusten rakennetta ja tietoturvan yleisellä tasolla, mitkä sovelluksen osat ovat haavoittuvaisia ja minkälaisia tietoturvauhkia verkkosovelluksiin liittyy. Sen jälkeen esittelen itserakennetun verkkosovelluksen käyttötarkoituksen, sekä sen toteutuksen, paneutuen tietoturvaratkaisuihin. Samalla kerron verkkosovelluksen kehitykseen liittyvistä toimista, kuten versiohallinnasta sekä kehitysympäristön ja tuotantoympäristön eriyttämisestä, jotka osaltaan tukevat ohjelmistokehitystä.

2 Verkkoteknologiat

Projektin näkökulma on, että jokainen palvelinpuolen ohjelmointikieli on oikein käytettynä turvallinen, ja lopputuloksena halutaan selkeä näkemys valitun ohjelmointikielen tietoturvaratkaisuista ja niitä tukevista hyvistä käytännöistä. Toinen vaihtoehto olisi ollut keskittyä vertailemaan eri teknologioiden tietoturvaratkaisuja ja lopputuloksena esittää perusteltu valinta tietoturvan kannalta parhaaksi teknologiaksi. Kattava tutkimus olisi kuitenkin vaatinut useiden teknologioiden samanaikaista omaksumista, ja projektin laajuus olisi muodostunut monta kertaa haluttua isommaksi.

Projektin palvelinpuolelle haluttiin yleisesti käytetty ja suosittu ohjelmointikieli, koska tämä on tietoturvan tutkimisen kannalta mielekkäintä. Suosituille ohjelmointikielille löytyy Internetistä paljon apua mahdollisiin ongelmiin. Samalla teknologian suosio takaa laajan tuen web-hotellipalveluissa.

Toinen kriteeri oli alkuun pääsemisen helppous, eli kuinka helppoa teknologian vaatimat ohjelmistot on asentaa ja konfiguroida ja kuinka nopeasti teknologian asentamisesta voi saada näkyviä tuloksia aikaiseksi.

Teknologioita voidaan valita useampiakin, mikäli yksi ei riitä kaikkiin projektin tarpeisiin. Esimerkiksi tietokantojen toteuttamiseen tarvitaan eri teknologia kuin itse muun toimintalogiikan ohjelmoimiseen. Selainpuolen teknologiat eivät ole tässä työssä yhtä oleellisia.

2.1 Palvelinpuolen teknologiat

Django

Django on ohjelmistokehys Python-ohjelmointikielelle. Se on julkaistu vuonna 2005. [1.]

Django koostuu neljästä pääkomponentista. URLConf määrittelee, mitä View-komponentteja missäkin URL:ssä näytetään. Kun käyttäjä lähettää pyynnön Django-palvelimelle, URLConf katsoo, minkä osoitteen käyttäjä pyynnössään esittää, ja sen perusteella suorittaa sille osoitteelle määrätyn View-komponentin. View't määrittävät

toiminnallisuuden, mitä kussakin polussa tapahtuu, esimerkiksi minkälaista tietoa käyttäjälle näytetään. [2, s. 21–37.]

Templatet eli sivupohjat määrittävät HTML-sivun rakenteen ja ulkonäön. Sapluunoiden avulla sivun toimintalogiikka voidaan erotella täysin sivujen ulkonäön määrittelyistä. HTML-sivun tapauksessa pohjassa on määritelty HTML-tunnisteet ja mahdollisesti tyylitiedot. Djangossa on määritelty oma sivupohjakieli, eli oma skriptikieli, jolla View'lta saatuja tietoja, kuten muuttujia, voidaan näyttää HTML-kielen lomassa. View'lta tuleville muuttujille voidaan määrittää esimerkiksi if- ja for-ehtolauseita ja muuttaa näkymää näiden perusteella. [2, s. 39–41.]

Modelit määrittävät yhteyden tietokantaan. Yleensä yksi Djangon model-komponentti vastaa yhtä tietokannan taulua. View-komponentissa voidaan modelien avulla käsitellä tietokannassa olevaa tietoa.[2, s. 72–79.]

Ruby on Rails

Rails on ohjelmistokehys Ruby-ohjelmointikielelle. Ruby ja Rails yhdessä muodostavat Ruby on Rails -verkkoteknologian. Ruby on Rails käyttää Model View Controller (MVC) -mallia, jossa sivuston graafinen ilme ja esityslogiikka on erotettu sovelluksen muusta sisäisestä logiikasta. Lisäksi Ruby on Rails käyttää Active Record -arkkitehtuurikaavaa, jossa halutut tietokantaoperaatiot kirjoitetaan Rubyllä ja sen sisäiset järjestelmät tulkitsevat ja muuttavat tietokantakomennot yhteensopiviksi projektin käyttämälle tietokannalle. Järjestelmä osaa myös tallentaa tietoa moneen erityyppiseen tietokantaan samalla kertaa. [3; 4, s. 20–28.]

ASP.net

ASP.net eli Active Server Pages on .NET-sovelluskehyksen osa verkkosivustojen kehittämiseen. Se julkaistiin alun perin vuonna 2002. .NET-sovelluskehyksen etuna on mahdollisuus käyttää useaa eri ohjelmointikieltä ohjelmien kehittämiseen. Näitä kieliä ovat muun muassa C# ja VisualBasic. [5; 6.]

Perusrakenteeltaan ASP.netissä rakennetaan HTML-sivun pohja, johon asetetaan ohjaimia, esimerkiksi nappeja ja lomakkeita. Sen jälkeen määritellään erikseen ohjainten

toiminnallisuus: mitä tapahtuu, kun ohjaimia käytetään, esimerkiksi sivulla olevaa nappia painetaan. [7, s. 13–19.]

Aina kun sivun koodiin tehdään jokin muutos, sivu on käännettävä uudestaan, ennen kuin se voidaan näyttää sitä pyytäneelle asiakkaalle. Kääntäminen tehdään automaattisesti, kun käyttäjä tekee ensimmäisen pyynnön muuttuneelle sivulla. Tämän jälkeen kokoamista ei tarvitse tehdä, ellei koodi muutu uudestaan. [7, s. 7.]

PHP

PHP (Hypertext Preprocessor) on dynaamisten ja interaktiivisten internetsivujen rakentamiseen suunnattu ohjelmointikieli. Dynaamisella verkkosivulla tarkoitetaan, että sisältö voi muuttua joka kerta, kun sivu ladataan. Esimerkiksi pelkällä HTML-kielellä sivut ovat staattisia, ne näyttävät joka kerta samoilta. Interaktiivisuudella taas tarkoitetaan, että käyttäjä voi vaikuttaa sivuun jollain tavalla. Esimerkiksi keskustelufoorumeilla käyttäjä voi lähettää kirjoituksen, ja se näkyy, kun sivu seuraavan kerran ladataan. [8, s. 3–4.]

PHP-ohjelmat sijaitsevat verkkopalvelimella tai rakentavat asiakkaille verkkosivuja palvelupyynnön saadessaan. Ohjelmat prosessoivat tietoa ja rakentavat siitä HTML-muotoisen verkkosivun. PHP-skriptin joukossa voi lisäksi olla suoraan HTML-kieltä, jota ei prosessoida. Verkkosivuja voi näin rakentaa helposti ja nopeasti kumpaakin kieltä käyttäen. [8, s. 3–4.]

2.2 Teknologian valinta

Palvelinpuolen teknologiaksi valikoitui PHP. Eniten tähän vaikutti alkuun pääsemisen helppous: omalle koneelle saatava Xampp-virtuaalipalvelin ja sen mukana tuleva PHP oli erittäin nopea asentaa. Xamppissa on mukana myös MySQL-tietokanta ja sen hallinnointiin tarkoitettu phpMyAdmin-työkalu. Koodia pääsee kirjoittamaan heti, ja näkyviä tuloksia on helppo saada aikaan. Toinen syy oli PHP:n suosio: W3Techs-sivuston tutkimusten mukaan PHP on tasaisesti kasvattanut suosiotaan vuosien ajan. [9.]

2.3 PHP lähikuvassa

PHP:n ja HTML:n yhteistoiminta

Jokaisessa tiedostossa PHP-koodi on laitettava <*?php*-tunnisteen sisään. Tämä kertoo PHP-moottorille että, alue on prosessoitava PHP-koodina. Osio lopetetaan *?>*-tunnisteella. Näiden tunnisteiden ulkopuolelle jäävä alue tulkitaan tavalliseksi HTML-koodiksi, eikä sille tehdä mitään. Jos tiedostossa ei ole ollenkaan <?php-tunnisteita, PHP-moottori ei tee sivulle mitään vaan lähettää sen suoraan selaimelle. [8 s. 25]

Koska vain edellä mainittujen tunnisteiden sisältö käsitellään PHP-koodina, HTML-kieltä voi käyttää vapaasti kaikkialla muualla. Voi esimerkiksi käyttää kaikkia HTML:n tunnisteita ja lomakkeita sekä muotoilla sivua CSS-tyyleillä. Yhteistoiminta HTML:n kanssa on PHP:n suurin vahvuus: dynaamisten verkkosivujen tuottaminen on näiden yhteistyöllä erittäin nopeaa.

Väljästi tyypitetty kieli

PHP on niin sanottu väljästi tyypitetty kieli. Muuttujaa määriteltäessä muuttujan tyyppiä ei tarvitse määritellä, ja muuttujan tyyppi vaihdetaan sen mukaan, minkälaista arvoa muuttujaan ollaan säilömässä tai missä yhteydessä muuttujaa käytetään. [8, s 36–39.]

Muuttujan tyypin voi aina tarkistaa *gettype()*-funktiolla, joka palauttaa muuttujan tyypin. Muuttujan tyypin voi myös vaihtaa *settype()*-funktiolla. [8, s 36–39.]

PHP komentoriviltä käytettynä

PHP:tä voidaan käyttää myös komentoriviltä ilman verkkopalvelimen asentamista, ja sitä voi käyttää perinteisenä komentosarjakielenä, samaan tapaan kuin esimerkiksi Linux-ympäristöissä käytettäviä Bashia tai Perliä. Ne voidaan asettaa suorittumaan automaattisesti esimerkiksi kerran viikossa tai päivässä. Lisäksi muut PHP-ohjelmat tai muut ohjelmat voivat kutsua niitä suorittamaan tietyn tehtävän. [8, s 765–773.]

Skriptit voidaan ajaa komentoriviltä muutamalla eri tavalla. Yksinkertaisimmillaan skriptitiedoston nimi annetaan parametrina PHP-moottorille, eli komentoriville kirjoitetaan ”*php hello.php”.* [8, s 765–773.]

Unix-ympäristössä on kuitenkin järkevintä muuttaa skriptitiedoston käyttöoikeuksia niin, että sen voi suorittaa suoraan komentokehotteelta, joko ./hello.php tai path/to/hello.php. Tämä tehdään käyttäen komentoa ”chmod u+x hello.php”, jolloin nykyinen käyttäjä saa suoritusoikeuden hello.php-tiedostoon. [8, s 766.]

Windows-ympäristössä paras ratkaisu on kirjoittaa komentojonotiedosto, jolla on sama tiedostonimi kuin PHP-skriptillä. Se sitten kutsuu php.exe-ohjelmaa. Jos siis PHP-skriptin tiedosto on nimeltään hello.php, kirjoitetaan komentojonotiedosto nimeltä hello.bat ja sen sisällöksi ”*C:\path\to\php\php5.3.0\php.exe %~n0.php %*”. Nyt kijoittamalla ”hello” käynnistyy hello.php. [8, s 766.]

Komentoriviparametrit tekevät komentoriviohjelmista selkeästi monipuolisempia. PHP-komentoriviskriptit tukevat parametrejä siinä missä muutkin. Jokaisessa komentoriviskriptissä on automaattisesti muuttujat $argc ja $argv. $argc kertoo ohjelman saamien parametrien määrän. $argv on parametritaulukko, johon kaikki parametrit on saatu. Jokaisella ohjelmalla on aina vähintään yksi argumentti, eli skriptitiedoston nimi. [8, s 767.]

Parametreja otetaan muodossa ”*./hello.php --nimi=Ville --sukupuoli=mies*”. Todellisuudessa PHP lukee argumentiksi kaikki skriptin nimen jälkeen tulevat välilyönnillä erotetut merkkijonot. ”--”- ja ”**=**”-merkit ovat kuitenkin selkeä tapa erottaa parametrin nimi ja arvo toisistaan. [8, s 767.]

Skripteillä pitää olla mahdollisuus vuorovaikutukseen käyttäjän kanssa. Linux-ympäristössä tämä onnistuu merkkijonovirtojen avulla. PHP tukee kolmea eri merkkijonovirtaa. *Stdin*:llä luetaan syötteitä käyttäjältä, ja sitä edustaa vakio STDIN. *STDOUT* lähettää tulosteita komentorivitulkille, joka voi tulostaa ne käyttäjän komentoriviikkunaan. STDERR-virta on tarkoitettu virheilmoitusten lähettämiseen. Virtoja voi käyttää, kuin ne olisivat tiedostoja. Kaikkia tiedostokäsittelyfunktioita voi myös käyttää niiden kanssa. [8, s 768–770.]

Komentoriviskriptejä voi asettaa suoritettavaksi tietyin väliajoin. Näin esimerkiksi vitaalit ylläpitotoimet voidaan asettaa suoritettavaksi automaattisesti. Unix-ympäristössä ajastus tehdään *contab*-tiedostoa muokkaamalla. Ensin kirjaudutaan sisään sillä käyttäjällä, jona skripti halutaan ajaa, ja kirjoitetaan terminaalissa *contab -e*. Tämä avaa contab-tiedoston oletustekstieditorissa. [8, s 770–772.]

2.4 HTTP-protokolla

HTTP-protokolla (Hypertext Transfer Protocol) on internetissä käytetty tiedonsiirtoprotokolla. TCP/IP-viitemallissa se sijaitsee ylimmässä eli sovelluskerroksessa. [10; 11.]

Yksi HTTP:n konsepteista on, että tiedostot sisältävät referenssejä toisiin tiedostoihin. Jokaisessa HTTP-palvelimessa on verkkosivujen lisäksi HTTP Daemon, ohjelma, joka odottaa ja käsittelee HTTP-pyyntöjä. Käyttäjän internetselain on HTTP-asiakasohjelma, joka lähettää pyyntöjä HTTP-palvelimille käyttäjän syöttäessä URL:n tai napsauttaessa verkkosivulla olevaa hypertext-linkkiä. Selain koostaa pyynnön halutusta verkkosivusta ja lähettää sen URL:n osoittamaan IP-osoitteeseen. [10.]

Evästeillä (engl. cookies) tarkoitetaan tietoa, jota verkkosivustot tallentavat käyttäjän koneelle. Evästeitä käytetään säilyttämään käyttäjän sivustolle antamia tietoja hänen omalla tietokoneellaan. Käyttäjän navigoidessa verkkosivustolle selainohjelmisto lähettää automaattisesti aikaisemman vierailun yhteydessä sivustolta saamansa evästeen takaisin. Näin sivusto saa aiemmalta vierailulta tallennetut tiedot uudelleen käyttöönsä. Käyttäjä voi kuitenkin itse päättää, antaako hän tallentaa evästeitä vai ei. [12.]

2.5 Tietokannat

Tietokannat ovat tiedon varastoja, joissa samankaltaiset tai toisiinsa liittyvät tiedot on järjestetty koherenttiin järjestykseen. Sähköisissä tietokannoissa, erotuksena paperisiin, on itse tiedon ja sen rakenteen lisäksi myös tietokannan hallintajärjestelmä eli TKHJ. Hallintajärjestelmä toimii välikätenä tiedon ja käyttäjän välillä. Se suorittaa käyttäjien pyynnöt ja käskyt ja huolehtii tietojen oikeanlaisesta tallentamisesta ja muista toimenpiteistä. TKHJ automatisoi monet tietokannan toiminnalle hyödylliset toiminnot. [13, s 4–5.]

2.6 Relaatiotietokannat

Relaatiotietokannat ovat sellaisia tietovarastoja, jotka noudattavat relaatiomallia. Relaatiomalli on alun perin E. F. Coddin IBM:llä kehittämä sääntöihin perustuva malli. [14.]

Relaatiomallissa tietokanta koostuu relaatioista, eli tauluista, jotka sisältävät tietyntyyppistä tietoa. Jokaisen erilaisen taulun ajatellaan esittävän jonkin oikean maailman asian tietoja, vaikkapa ihmistä, taloa tai lentokonetta. Jos taulu vastaa autoa, taulun sisällä on tällöin usean eri auton tiedot, esimerkiksi merkki, malli, rekisterinumero ja sarjanumero. Jokaisen näistä tietoryppäistä ajatellaan olevan ainutlaatuinen, eli se kuvaa jotain tiettyä autoyksilöä. Yleensä jokaiselle tietoryppäälle annetaan lisäksi tiedot yksilöivä tunniste, esimerkiksi juoksevalla numeroinnilla numerosta 1 alkaen. [14.]

Relaatiomallissa tauluilla voi lisäksi olla viittauksia toisiin tauluihin. Esimerkiksi jos autolle halutaan liittää kuljettaja, se voidaan toteuttaa lisäämällä auton tietokantatauluun uusi tietue: kuljettajaId, joka viittaa suoraan johonkin kuljettajataulun id-kenttään. [14.]

Relaatiotietokannan viittausten avulla voidaan vähentää tallennettavan tiedon määrää ja täten tietokannan kokoa. Jokainen auto ja kuljettaja tallennetaan kerran. Jos yksi kuljettaja on merkitty useaan eri autoon, viitataan kuljettajaan ainoastaan id:llä. Mikäli tällaista viittausta ei olisi, jouduttaisiin jokaisen auton tietoihin tallentamaan myös kuljettajan tiedot joka kerta erikseen.

Relaatiotietokannat ovat suosittuja tietovarastoja tietotekniikassa, ja relaatiomallin mukaan suunniteltuja tietokantoja on useita. Tähän projektiin tietokannaksi valittiin MySQL.

3 Verkkosovellusten rakenne

3.1 Sovelluskehykset

Sovelluskehys on kokoelma kirjastoja ja funktioita, joilla järjestelmään voidaan suoraan lisätä tai helposti toteuttaa yleisiä verkko-ohjelmissa tarvittavia toimintoja. Sovelluskehysten tehtävä on helpottaa ja nopeuttaa ohjelmoijien työtä, jolloin työpanoksen voi keskittää juuri tietylle ohjelmistolle erityisten toiminnallisuuksien toteuttamiseen. [15, s. 208.]

Sovelluskehykset ovat yleensä ilmaisia ja vapaasti kaikkien käytössä, joten niiden käyttäjäkunta on myös suuri. Virheet ja tietoturvapuutteet huomataan helpommin, koska niin moni silmäpari tutkii koodia. Puutteet on dokumentoitu, jos ei kehittäjien toimesta niin Internetin keskusteluissa. Sen sijaan kun jokin kriittinen ominaisuus, esimerkiksi käyttäjähallinta, toteutetaan itse, on myös itse pidettävä huolta, että se on tietoturvallinen ja toimiva. [16.]

Sovelluskehyksiä kannattaa käyttää, mikäli sovellukseen halutaan vähänkin enemmän toiminnallisuuksia. Tässä projektissa sovelluskehyksen käytöstä olisi suuri apu. Päätös oli kuitenkin se, että järjestelmä rakennetaan alusta asti itse, jolloin rakennusprosessista opitaan mahdollisimman paljon. Samalla tietoturvakysymyksiin voidaan keskittyä yleisellä tasolla niin, ettei käytössä oleva sovelluskehys vaikuta ratkaisuihin.

3.2 Hakemistorakenne

Hakemistorakenteella tarkoitetaan sitä, miten projektin tiedostot on organisoitu toimivaksi kokonaisuudeksi. Järkevä hakemistorakenne ei ole pelkästään apu ohjelmoijalle projektin kokonaisuuden hallinnassa, vaan sillä on myös vaikutusta ohjelman toimintaan. PHP-ohjelmoinnissa tärkein hakemistorakenteeseen liittyvä asia on pitää koodista ainoastaan välttämättömät asiat näkyvissä ulkomaailmaan. Tämä tarkoittaa, että projektilla on yksi verkkosivujen juurihakemisto (ns. document root -hakemisto), joka sijaitsee palvelimella. Kun asiakas ottaa yhteyden verkkosivustoon, palvelin etsii verkkosivuja tästä kansiosta ja näyttää ne sitten asiakkaalle. Koska ainoastaan palvelin tulkkaa PHP-koodia, voi verkkosivujen juurikansiossa oleva PHP-tiedosto viitata muual-

la palvelimella oleviin PHP-sivuihin. Asiakas siis näkee ainoastaan yhden hakemiston (ja sen alihakemistot), vaikka suurin osa suoritettavasta PHP-koodista sijaitsisikin muissa hakemistoissa. [17; 18; 19.]

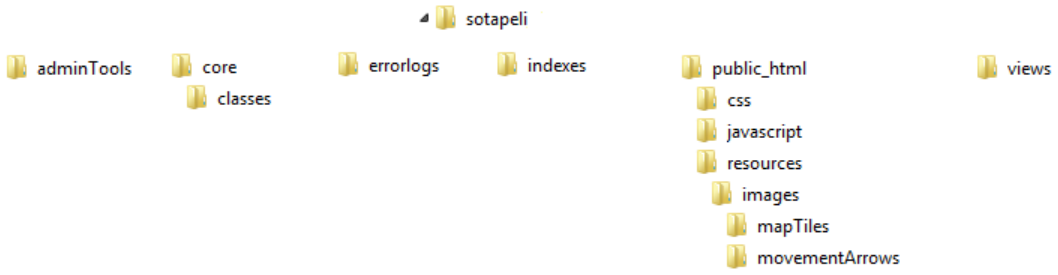
Kun tiedostot eivät näy Internetiin, niihin ei myöskään päästä ulkopuolelta käsiksi. Jos esimerkiksi jokin osa sivustosta on suojattu salasanalla ja tarvitse tunnistautumisen, eivät käyttäjät voi ohittaa sitä.

Tietoturvan kannalta on merkittävää, että mikäli koodi on kaikkien nähtävillä, mahdollinen hyökkääjä pystyy selvittämään ohjelmiston rakenteen ja saa näin enemmän tietoa, mikä saattaa auttaa hyökkäyksessä sivustoa vastaan. Jotta ohjelmisto olisi täysin suojattu, täytyy myös ohjelmakoodi tehdä niin, ettei se mahdollista käyttäjän tekemiä suoria viittauksia ohjelmiston tiedostoihin esimerkiksi osoiterivin kautta. Koodin piilottaminen ei poista tarvetta varmentaa käyttäjien autentikoitumista.

Juurihakemistoon ja sen alihakemistoon kannattaa laittaa ainoastaan sellaiset tiedostot, joihin selaimen on päästävä käsiksi. Näitä ovat JavaScript-tiedostot, kuvat ja tyylitiedostot.

Juurihakemiston ulkopuolella tehtävä hakemistorakenteen organisointi helpottaa projektin kehitystä. Se on myös hyvän ohjelmointitavan mukaista. Tässä projektissa Views-kansion tiedostoissa määritellään funktiot, joiden avulla pelin näkymät piirretään selaimeen. Indexes-kansion tiedostot lataavat ja suorittavat kukin views-kansion koodin. Kaikki lokitiedostot sijaitsevat errorlogs-kansiossa.

Kuvasta 1 selviää projektin hakemistorakenne. AdminTools-hakemisto sisältää pelin aloittamiseen ja lopettamiseen liittyvät toiminnot. Core-hakemisto sisältää pelin toimintalogiikan. Classes-alihakemisto sisältää pelin komponenttien määrittelyt sekä tietokanta, -ja validaatioluokat. Errorlogs-hakemisto sisältää tiedot pelin aikana mahdollisesti tapahtuvista virheistä. Indexes-hakemistossa sijaiseva koodi sisältää tiedot siitä, mitä kullakin pyynnöllä käsitellään. Views-hakemistossa sijaitseva koodi sisältää logiikan sille, miten asiat näytetään. Se käyttää apuna luokkia Classes-alihakemistosta.



Kuva 1. Insinöörityön verkkosovelluksen hakemistorakenne.

Public_html-kansio sisältää ulkomaailmaan näkyvät indeksitiedostot ja JavaScript-tiedostot ja kuvat omissa alihakemistoissaan.

Indexes-, views-, core- ja classes-hakemistot muodostavat yhdessä kokonaisuuden joka noudattaa Model – View – Controller -arkkitehtuurimallia, joskaan ei ihan täydellisesti.

4 Verkkosovellusten tietoturva

Verkkosovellukset sijaitsevat palvelimilla, jotka ovat yhteydessä Internetiin. Jotta asiakkaat voivat käyttää sovelluksia, on palvelinten oltava ulkomaailmaan avoinna. Siksi on tärkeää rakentaa sovellukset niin, että käyttäjät voivat suorittaa vain heille ennalta määrättyjä toimintoja. Mikäli tämä tavoite epäonnistuu, on käyttäjän mahdollista saada palvelin hallintaansa ja käyttää sitä omiin tarkoituksiinsa. Lisäksi palvelimella sijaitsevien sovellusten toiminta häiriintyy ja niiden tiedot voivat joutua vääriin käsiin. Näiden seikkojen takia sovelluksista on rakennettava niin tietoturvallisia kuin suinkin mahdollista. [15, s. 3–10.]

4.1 Käyttäjien hallinta

Kun rakennetaan verkkopalvelua, usein vähintään osa sen ominaisuuksista halutaan tarjota ainoastaan rekisteröidyille käyttäjille. Rekisteröityminen yksilöi käyttäjän ja tarjoaa hänelle esimerkiksi yksilöllisen käyttäjänimen tai pääsyn vain hänelle rajattuun tietoon. Kun rekisteröity käyttäjä haluaa käyttää palvelua, hänen on tunnistauduttava. Näin voidaan varmistua, että henkilö todella on se joka hän väittää olevansa. [15, s. 95.]

Erilaisia tapoja käyttäjän tunnistamiseksi

Yleinen vaihtoehto tunnitautumiseksi on yksilöllisen käyttäjänimen ja salasanan luominen jokaiselle rekisteröityvälle käyttäjälle. Nämä käyttäjätiedot tallennetaan sovelluksen omaan tietokantaan. Tunnistautumisessa käyttäjä syöttää sekä käyttäjänimensä että salasanansa palveluun. Palvelu varmistaam että annetulla käyttäjänimellä löytyy käyttäjä ja sille tallennettu salasana vastaa syötettyä salasanaa. [15, s. 97]

Tupas-tunnistuspalvelu hyödyntää Suomessa toimivien pankkien tunnistautumisjärjestelmää. Kirjautuessaan verkkopalveluun käyttäjä valitsee oman pankkinsa ja tunnistautuu pankin järjestelmään samalla tavoin kuin asioidessaan verkkopankissa. Kun tunnistaminen on suoritettu, tunnistustiedot välitetään verkkopalveluun. Tupas-palvelu toimii ainoastaan henkilökohtaisilla pankkitunnuksilla, ei esimerkiksi yrityksen verkkopankkitunnuksilla. [20.]

Tupas-palvelun etuja ovat käyttäjän vahva tunnistaminen: pankkitunnukset ovat henkilökohtaiset ja pankin järjestelmät tietoturvallisia. Haittana on vaivalloisuus: pankkiin tunnistautuessa käytetään usein erillistä avainlukulistaa tunnuksen ja salasanan lisäksi.

Mobiilivarmenne on tunnistautumispalvelu, joka hyödyntää matkapuhelimen SIM-korttia. Kuin käyttäjä on kirjautumassa verkkopalveluun, hän syöttää puhelinnumeronsa. Numero välitetään tunnistautumispalveluun, joka lähettää viestin käyttäjän puhelimeen. Käyttäjä vastaa viestiin omalla salasanallaan. Syötetty salasana välittyy ainoastaan SIM-kortille. Jos salasana hyväksytään, siitä välittyy tunnistautumispalvelun kautta tieto verkkopalveluun, ja tunnistautuminen on suoritettu. Mobiilivarmenne vaatii toimiakseen sitä tukevan SIM-kortin. [21.]

Facebook tarjoaa palvelua, jossa käyttäjä voi kirjautua verkkopalveluihin käyttäen omia Facebook-tunnuksiaan. Samalla palveluun voidaan välittää muitakin tietoja, esimerkiksi syntymäpäivä, kotikaupunki ja senhetkinen sijainti [22]. Useat verkkopalvelut kysyvät käyttäjiltä muitakin tietoja kuin vain tunnuksen ja salasanan. Facebook-tunnusten avulla käyttäjän ei tarvitse kirjoittaa samoja tietoja erikseen eri palveluihin.

Tiivistefunktiot

Tiiviste-funktioiden tarkoituksena on luoda selväkielisestä tekstistä tietynpituinen tiivistemerkkisarja niin, ettei selväkielisen tekstin sisältöä tai pituutta pystytä jälkikäteen selvittämään. Tiiviste toimii alkuperäisen tiedon varmenteena: jos alkuperäistä merkkijonoa muutetaan, myös sen tiiviste muuttuu täysin erilaiseksi. Alkuperäisen tiedon aitous voidaan varmistaa luomalla muuttuneeksi uskotusta merkkijonosta tiiviste ja vertaamalla sitä alkuperäiseen. Jos tiivisteet ovat samat, alkuperäinen tieto ei ole muuttunut. [23.]

Funktion turvallisuuteen vaikuttaa niin kutsuttu törmäyskestävyys (engl. collision resistance) eli se, kuinka helposti on löydettävissä kaksi erilaista merkkijonoa, joilla on täysin sama tiiviste. [24.]

Tiivistefunktioita käytetään käyttäjien salasanojen turvaamiseen. Kun salasanat tallennetaan tietokantaan, on huolehdittava, ettei tietokanta joudu vääriin käsiin. Käyttäjät saattavat käyttää samaa salasanaa useissa muissakin palveluissa, jolloin yhden palvelun käyttäjälistan altistumisella saattaa olla kauaskantoiset seuraukset.

Tiivistefunktiot tuovat tähän tilanteeseen suojaa. Vaikka hyökkääjä pääsisikin palvelun tietokantaan käsiksi, hän näkee ainoastaan salasanojen tiivisteet. Tiivisteistä ei näe alkuperäistä salasanaa, joten selkokielistä salasanaa ei saada näkyviin. Kirjautuessaan palveluun käyttäjä syöttää käyttäjätunnuksen ja salasanansa. Salasanasta muodostetaan tiiviste, ja sitä verrataan tietokannassa olevaan tiivisteeseen. Mikäli tiivisteet vastaavat toisiaan, on salasana oikein ja käyttäjä kirjataan sisään. On tärkeää varmistaa, ettei selkokielisiä salasanoja tallenneta palveluun missään vaiheessa. [25.]

Vaikka salasanaa ei voikaan palauttaa tiivisteestä, on hyökkääjällä muita tapoja sen selvittämiseksi.

Raaka voima -tyyppisessä hyökkäyksessä kokeillaan kaikki mahdolliset salasanat eli merkkijonot läpi yksi kerrallaan [26]. Tieto salasanan minimi- ja maksimipituudesta sekä sallituista merkeistä auttaa rajaamaan etsittävien merkkijonojen määrää. Nämä tiedot saa usein selville palveluun rekisteröidyttäessä, koska käyttäjän salasanan valintaan vaikuttaa salasanan maksimipituus. Teoriassa mahdollisia yhdistelmiä on hyvin suuri määrä. Vaikka vahvojen salasanojen merkitystä usein korostetaan, kaikki ihmiset eivät tätä noudata vaan valitsevat lyhyitä ja helposti muistettavia [27]. Lyhyt salasana voidaan selvittää raaka voima -metodilla verrattain helposti.

Tiivistefunktion törmäyskestävyys määrittää, kuinka usein kahdella erilaisella merkkijonolla on sama tiiviste. Jos funktion törmäyskestävyys on huono, on verrattain helppoa löytää palvelun käyttäjätiedoista tiivisteitä aivan eri merkkijonoilla kuin mitä käyttäjien salasanat olivat. Koska salasana tarkistetaan aina tiivisteestä, pääsee hyökkääjä kirjautumaan sisään väärällä salasanalla.

Sanakirjahyökkäys on raaka voima -metodin muoto, jossa käytetään apuna sanakirjojen sanoista ja muista yleisistä salasanoista muodostettua listaa. Listalla olevia sanoja kokeillaan raaka voima -metodin tapaan, jolloin kaikki listan salasanojen käyttäjät paljastuvat. [25.]

Salasanalistoista voidaan myös muodostaa hakutaulukkoja, jossa on salasanoja ja niiden etukäteen muodostettuja tiivisteitä parina. Tällä tavoin tiivisteiden vertaaminen on nopeampaa kuin tiivisteiden muodostaminen erikseen. [25.]

Suola

Edellä mainitut hyökkäykset toimivat, koska käyttäjien salasanat ovat lyhyitä tai helposti pääteltävissä. Ongelma voidaan korjata lisäämällä salasanaan ylimääräisiä merkkejä, eli niin kutsuttu suola. Suolan lisäyksen jälkeen salasanasta muodostetaan tiiviste kuten aikaisemminkin, jolloin saadaan erilainen merkkijono kuin pelkästä salasanasta. Suola tallennetaan tietokantaan erilliseen tekstikenttään. Kun käyttäjä kirjautuu, hänen syöttämäänsä salasanaan lisätään suola, ennen kuin tiiviste muodostetaan. Vaikka hyökkääjällä olisikin tietokanta hallussaan, hän ei voi käyttää sanakijahyökkäystä tai ennalta muodostettua hakutaulukkoa tiivisteiden läpikäymiseen. Merkkijonon lisäämisen jälkeen tiivisteen muodostama salasana on pidempi, jolloin raaka voima -hyökkäys vaikeutuu. Suolan on hyvä olla pitkä ja muodostua mielivaltaisista merkeistä. [25.]

Jos käyttäjän salasana on ”hevonen”, se voidaan suomalaisista sanoista koostuvalla sanakirjahyökkäyksellä löytää helposti. Jos tälle salasanalle arvotaan suolaksi ”HaDhLqQmssXModfgWI64YOD”, tulee tiivisteen muodostavaksi merkkijonoksi ”hevonenHaDhLqQmssXModfgWI64YOD”. Tämä vaikeuttaa hyökkääjän työtä huomattavasti.

4.2 Web-hotellipalveluihin liittyvät riskit

Web-hotellilla tarkoitetaan palvelua, jossa samalla verkkopelvelimella sijaitsee monen eri tahon (yritysten, järjestöjen tai yksityisten) verkkosivustoja. Asiakas vuokraa kiintolevytilaa, jonne hän voi verkkosivustonsa laittaa. [28.]

Web-hotelleihin liittyy monia tietoturvariskejä, mutta sivuston ohjelmoijan on mahdollista minimoida niistä osa.

Sessiotiedot web-hotelleissa

Verkko-sivustot tallentavat käyttäjän koneelle evästeitä. Yksi tapa evästeiden hyödyntämiseen on tallentaa niihin käyttäjän yksilöivä merkkijono. Koska evästeet lähetetään joka pyynnöllä takaisin verkkosivustolle, voidaan tällä tekniikalla tunnistaa, että pyynnöt tulevat samalta käyttäjältä. Näin käyttäjän asioinnista palvelimella syntyy sessio.

Palvelimet käyttävät usein Linux-käyttöjärjestelmää. Linux-ympäristössä sessiotiedot sijaitsevat oletusarvoisesti /tmp-hakemistossa, johon kaikilla käyttäjillä on kirjoitusoikeus. Tavalliset käyttäjät eivät pääse lukemaan toisten /tmp-kansioon kirjoittamia tiedostoja. Sessiodatan kirjoittaa kuitenkin aina sama verkkopalvelinohjelmiston käyttäjä (esimerkiksi Apache-verkkopavelinohjelmistossa käyttäjä on nimeltään *nobody)*, joten sillä on pääsy kaikkeen sessiodataan riippumatta siitä, miltä pavelimelta tulevalta sivustolta data on peräisin. Hyökkääjän on näin ollen mahdollista kirjoittaa PHP-tiedosto, joka lukee kaikki sessiotiedot, ja näin päästä käsiksi toisten sivuston sessiotietoon. [29.]

Sessiotietojen lukeminen onnistuu esimerkiksi kuvan 2 skriptillä.

```
70  <?php
71  header('Content-Type: text/plain');
72  session_start();
73  $path = ini_get('session.save_path');
74  handle = dir($path);
75  while($filename = $handle->read())
76  {
77      if(substr($filename, 0, 5) == 'sess_')
78      {
79          $data = file_get_contents("$path/$filename");
80          if(!empty($data))
81          {
82              session_decode($data);
83              $session = $_SESSION;
84              $_SESSION = array();
85              echo "Session [" . substr(filename, 5) ."]\n";
86              print_r($session);
87          }
88      }
89  }
```

Kuva 2: Sessioiden luku levyltä.

Skripti etsii ensin polun, jonne sessiotiedot tallennetaan, minkä jälkeen se etsii sess_-alkuisia tiedostoja. Kun sellainen löydetään, tiedot jäsennetään ja tulostetaan.

Sessiotiedot on syytä pitää poissa hyökkääjien ulottuvilta. Ratkaisu tähän on, että muun herkän tiedon tavoin myös sessiotiedot tallennetaan tietokantaan. PHP:ssä on sisäänrakennettuna sessiohallintafunktiot. Niiden tilalle voi kuitenkin asettaa omat funktiot *session_set_save_handler()*-funktiolla. [30.]

```
session_set_save_handler('_open','_close','_read','_write','destroy','_clean');
```

Funktiolle annetaan uusien sessiokäsittelyfunktioiden nimet. Funktiot voi nimetä haluamikseen. Session_set_save_handler():ä kutsutaan ennen session_start()-funktiota, mutta varsinaiset sessionkäsittelyfunktiot voidaan määritellä missä tahansa koodissa. Ainoa muutos koodissa ovat nämä käsittelyfunktiot; $_SESSION-muuttuja toimii edelleen samalla tavalla ja kaikki sessioihin liittyvät konfiguraatiot toimivat.

4.3 Evästeet ja istunnon kaappaaminen

Evästeet mahdollistavat käyttäjän yksittäisten pyyntöjen niputtamisen yhdeksi istunnoksi; käyttäjä tekee palvelimelle yksittäisiä pyyntöjä mutta palvelin tietää niiden tulevan samalta käyttäjältä. Saadessaan pyynnön käyttäjältä palvelin voi sisällyttää vastaukseensa niin kutsutun Set-Cookien. Tämä on pyyntö käyttäjälle, että hän lähettäisi vastaavan evästeen tulevissa pyynnöissään. Mikäli evästeeseen sisällytetään yksilöivä tunniste, pystytään tietyn käyttäjän pyynnöt yhdistämään toisiinsa. Tällä tavoin palvelin voi seurata käyttäjän toimia ja niputtaa kaiken toiminnan yhdeksi istunnoksi. [31.]

Edellä mainittu voidaan hoitaa PHP:n sisäänrakennetulla istuntomekanismilla. Kun session_start()-komento suoritetaan, PHP katsoo onko istunnon tunniste sisällytetty nykyiseen kutsuun. Mikäli näin on, kyseessä olevan istunnon tiedot luetaan ja tallennetaan $_SESSION-nimiseen superglobaaliin taulukkoon. Mikäli näin ei ole, PHP generoi istunnon tunnisteen ja uuden merkinnön istuntojen tietovarastoon. [31.]

Vaikka tämä kaikki onnistuukin helposti, PHP:n istuntomekaniikkaan ei ole luontaisesti toteutettu juurikaan tietoturvaa. Tosin istunnon tunniste on tarpeeksi satunnainen niin, ettei sitä käytännössä voida ennustaa. Ohjelmoijan on itse rakennettava tarvittavat suojamekanismit, jotta istunnosta saataisiin turvallinen.

Yksi evästeisiin liittyvä riski on, että hyökkääjä saattaa varastaa ne. Mikäli istunnon tunniste on sisällytetty evästeeseen, mahdollistaa varastaminen istunnon kaappauksen.

Kaksi yleisintä tapaa, miten evästeet vaarantuvat, ovat selaimen turvallisuuspuutteet sekä Cross Site Scripting (XSS).

Session Fixation -hyökkäyksessä ideana on harhauttaa käyttäjä käyttämään istunnon tunnistetta, jonka hyökkääjä on valinnut. Yksinkertaisimmillaan harhautus voidaan toteuttaa tarjoamalle käyttäjälle linkki, johon on sisällytetty istunnon tunniste. [31.]

Hyökkääjien hallussa olevalla sivulla voi olla esimerkiksi html-linkki “Paina tästä”. Mikäli käyttäjä painaa sitä, hän siirtyy esimerkki.fi-sivustolle ja antaa siellä omaksi sessiotunnisteekseen merkkijonon ”9876”. Mikäli tämä onnistuu, voi hyökkääjä vierailla samalla sivustolla käyttäen samaa istunnon tunnistetta ja päästä näin käsiksi huijatun käyttäjän tietoihin. Jos esimerkiksi käyttäjä kirjautuu palveluun sisään, pääsee hyökkääjä myös palveluun sisään. Sama voidaan myös tehdä headereilla: header('Location: http://www.esimerkki.fi/index.php?PHPSESSID=9876'); -komento ohjaa käyttäjän selaimen automaattisesti esimerkki.fi-sivustolle. [31.]

Yksinkertainen ratkaisu ongelmaan on antaa käyttäjälle uusi istunnon tunniste aina kun hän kirjautuu sisään (kuva 3).

```
8   $_SESSION['kirjautunut']=FALSE;
9       if(check_login())
10      {
11          session_regenerate_id();
12          $_SESSION['kirjautunut']=TRUE;
13      }
```

Kuva 3: Session id:n uudelleenluonti.

Kuvan check_login-funktion oletetaan olevan käyttäjän tunnistautumisen tarkistus. Mikäli tarkistus onnistuu, session id luodaan uudelleen.

4.4 Cross site request forgery -hyökkäys

Cross Site Request Forgery (CSRF) on hyökkäys, jossa halutaan käyttäjän tekevän jokin, usein autentikoitumista vaativa, toimenpide sivustolla ilman käyttäjän suostumusta. Käyttäjä huijataan lähettämään pyyntö palvelimelle, esimerkiksi http-linkin avulla. Palvelin näkee pyynnön käyttäjältä ja suorittaa sen toimenpiteet, mikäli käyttäjällä on siihen oikeudet. Palvelin ei osaa erotella, tekikö käyttäjä pyynnön omasta tahdostaan vai huijasiko joku hänet painamaan linkkiä. Pyyntö on rakennettu niin, että suorittaessaan sen palvelu tekee jonkin käyttäjän kannalta epäedullisen toimenpiteen, esimerkiksi muuttaa käyttäjän salasanan. [32.]

Ongelman ydin on, ettei palvelin osaa erotella, tekikö käyttäjä pyynnön omasta tahdostaan vai jostain muusta syystä. Ratkaisuna on varmentaa haluttu pyyntö vielä käyttäjältä, esimerkiksi kysymällä tämän salasanaa. Toinen tapa on varmentaa pyyntö lisäämällä siihen yksilöllinen avain. Avain luodaan, kun käyttäjä saapuu sivulle jossa CSRF-hyökkäykselle altis kutsu pitää lähettää. Avain tallennetaan pyyntöön, esimerkiksi lomakkeen salattuun-kenttään ja käyttäjän evästeeseen. Kun pyyntö lähetetään, palvelin tarkistaa, että sama avain löytyy sekä käyttäjän evästeestä että lähetetystä pyynnöstä. Näin voidaan varmistua, että käyttäjä vieraili lomakkeen sisältämällä sivulla. [32; 33; 34.]

5 Hyvät työskentelytavat

Ohjelmistoprojektien kanssa työskennellessä on yleensä tapana kirjoittaa koodi eri paikassa kuin sitä käytetään eli missä asiakkaat voivat ottaa järjestelmään ulkomaailmasta yhteyttä. Näin projektin kehittäjä voi rakentaa uutta versiota järjestelmästä ilman, että nykyisten käyttäjien käyttö häiriintyy. Koodi kirjoitetaan yleensä kehittäjän omalla työasemalla, kun taas asiakkaat ottavat yhteyttä palvelimeen. Tätä jakoa kutsutaan kehitysympäristöksi ja tuotantoympäristöksi.

Versionhallinnalla tarkoitetaan ohjelmistoa, joka pitää kirjaa projektin tiedostojen ja rakenteen muutoksista. Käyttäjät voivat kirjoittaa koodia omilla työasemillaan ja siirtää sitä versionhallinnan avulla tietovarastoon. Versiohallinnan avulla kaikki käyttäjät ovat tietoisia toistensa tekemistä muutoksista. Kun joku tekee omaan koodiinsa muutoksen, muut voivat päivittää muutoksen omiin tiedostoihinsa. Tiedostoja ei kirjoiteta kokonaan alusta, vaan ainoastaan muuttuneet kohdat päivitetään. [35.]

Versionhallinnan käyttö sujuvoittaa työskentelyä, koska muiden tekemien muutosten päivittäminen omalla työasemalla olevaan versioon hoidetaan ohjelmallisesti. Koska muutoksista pidetään kirjaa, voidaan myös siirtyä käyttämään jotain aiempaa versiota, mikäli uudemmat huomataan toimimattomiksi. Versionhallinta helpottaa myös koodin viemistä tuotantoympäristöön, kun voi kopioida tiedostot tietovarastosta ja tietää täsmälleen, minkä version on tuotantoon viemässä. [35.]

6 Strategiapelin toiminta

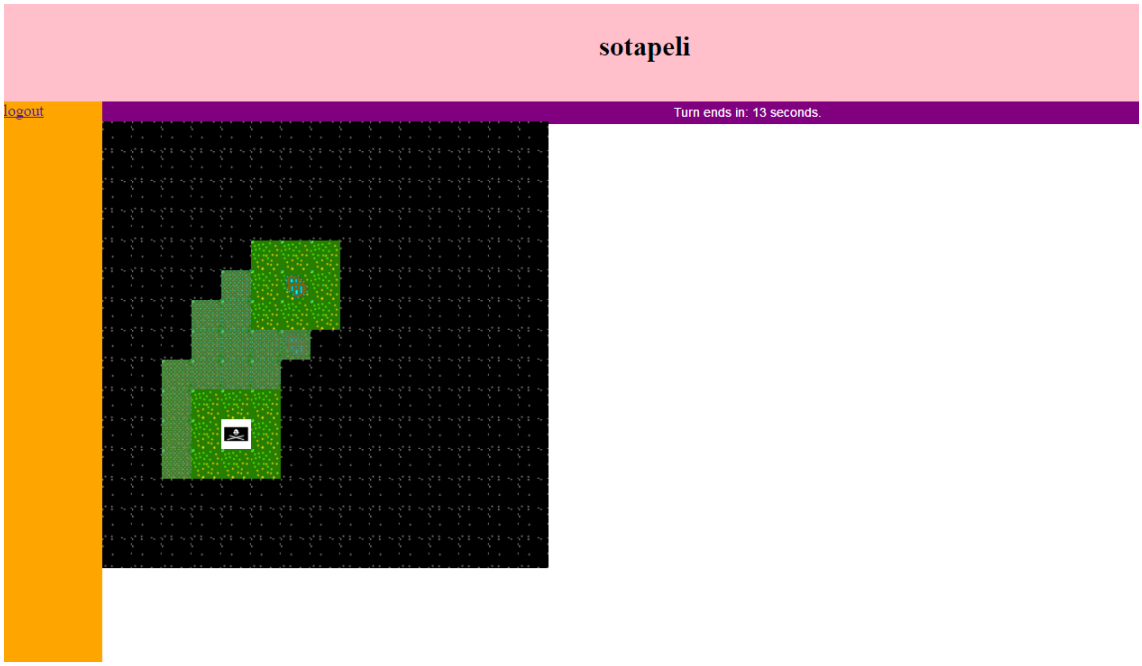
6.1 Pelin pelaaminen ja tavoite

Insinöörityön osana tehtiin verkkosovellus, selaimella pelattava strategiapeli. Pelissä on tavoitteena laajentua pelialueella, kasvattaa armeijoita ja valloittaa kaikki vastustajien hallussa olevat kaupungit. Pelin voittaa se pelaaja, jolla on viimeisenä hallussaan yksi tai useampia kaupunkeja.

6.2 Keskeiset komponentit

Kartta / shroud

Kartta on pelialue, jolla kaikki pelin tapahtumat taphtuvat: armeijat liikkuvat, taistelevat ja valtaavat alueita. Kartta on samalla pelin päänäkymä (kuva 4). Kartta koostuu neliönmuotoisista laatoista shakkilaudan tapaan. Jokaisella laatalla on x- ja y-koordinaatit. Erilaisia laattoja on muutamaa erilaista tyyppiä: avomaata, vuoristoa, rantaa ja kaupunkeja. Jokaisesta laatasta pääsee kulkemaan kahdeksaan ilmansuuntaan, kunhan kohdelaatta vain on vapaa.



Kuva 4: Armeija liikkuu pelin kartalla.

Kun karttaa tutkitaan, laajenee pelaajan käsitys pelikentästä. Vastustajien armeijat näkyvät kuitenkin ainoastaan, mikäli ne ovat kaupungin tai oman armeijan viereisessä laatassa.

Armeija ja yksiköt

Armeijat ovat kartalla liikkuvia joukko-osastoja, jotka koostuvat useista taistelijoista (myöhemmin yksiköistä). Yhteen armeijaan voi kuulua useita erityyppisiä yksiköitä, esimerkiksi miekkamiehiä, peitsimiehiä ja ratsumiehiä. Armeija on pelialueella aina yhdessä ruudussa kerrallaan. Armeijan maksimikokoa ei ole rajattu, eli siinä voi olla määrittelemättömän monta yksikköä kerrallaan.

Armeija voi liikkua vuorossa yhden ruudun haluamaansa suuntaan. Pelaaja voi myös halutessaan jakaa armeijan kahtia, eli käytännössä siirtää osan joukoista viereiseen ruutuun uudeksi armeijaksi tai liittää kaksi armeijaa yhdeksi liikuttamalla armeijan samaan ruutuun, jossa jo valmiiksi on pelaajan armeija.

Yksiköt ovat taistelijoita, jotka yhdessä muodostavat armeijoita. Yksiköitä rakennetaan kaupungeissa.

Yksikkötyypit eroavat toisistaan kestävyydessä, hyökkäysvoimassa taistelunopeudessa ja tuotantoajassa eli siinä, kuinka monta vuoroa yksikön valmistumisessa kestää.

Kaupungit

Kaupungit ovat pelin kartalla sijaitsevia tärkeitä alueita, jotka tuottavat yksiköitä.

Pelin alussa jokaisella pelaajalla on hallussaan yksi kaupunki. Kaikki loput kaupungit eivät ole kenenkään hallussa, ja ne voidaan valloittaa vapaasti omalle puolelle. Kun kaupunki on valloitettu, siinä voidaan aloittaa yksiköiden tuottaminen.

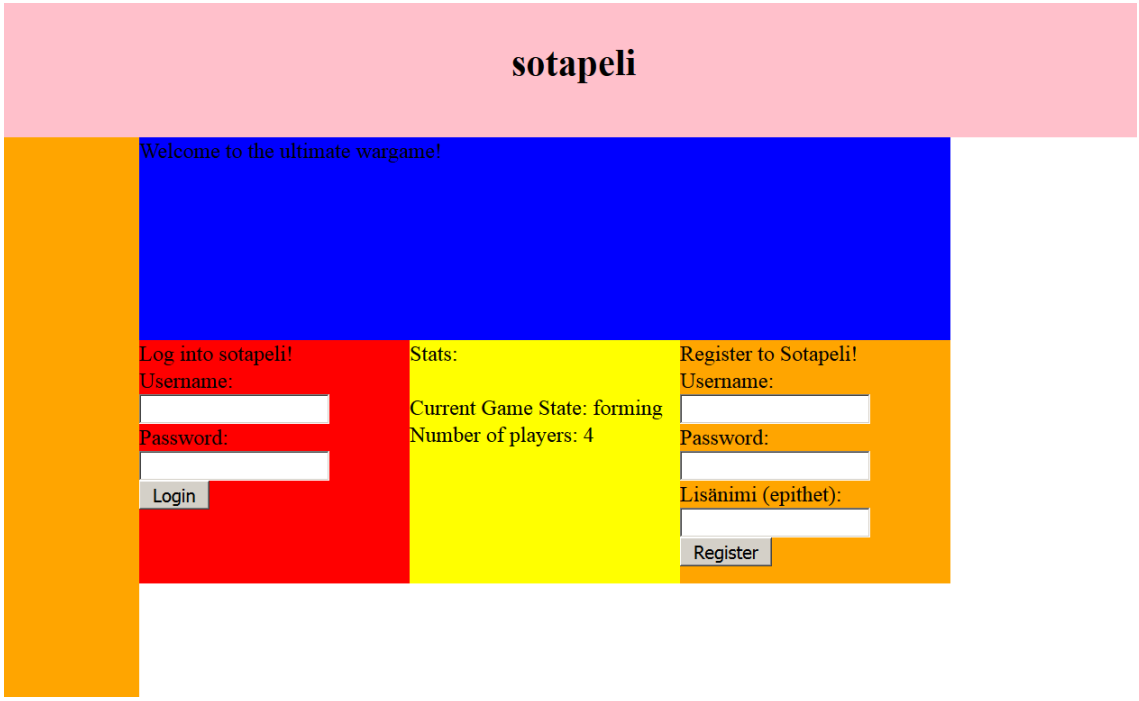
Kaupungit ovat pelaajan avain voittoon. Jokainen kaupunki tuottaa taisteluyksiköitä. Pelaaja voi valita joka kaupungille erikseen, mitä yksikkötyyppiä siellä tuotetaan. Yksikköjen tuottamiseen menee aina tietty määrä vuoroja, eli yksi tuotantosykli. Tuotettavana olevan yksikkötyypin voi vaihtaa joka vuorolla, mutta tällöin käynnissä oleva tuotantosykli menetetään.

Kaupungit vallataan siirtämällä oma armeija kaupunkiin. Mikäli kaupungissa on vastustajan armeija, syntyy taistelu, aivan kuin jos pelaaja olisi hyökännyt tavallisesti armeijan kimppuun. Mikäli taistelun lopussa vastustajan armeija on täysin tuhottu, siirtyy kaupunki valloittajan haltuun. Muussa tapauksessa se pysyy puolustajan hallinnassa.

Mikäli pelaaja menettää viimeisen kaupunkinsa, on peli hänen osaltaan ohi.

Pelin aloitus ja ottelut

Pelin tilaa säätelee tietokannan gamestats-taulun gamestatus-kenttä. Tiloja on kolme: Forming-tilassa odotetaan pelaajia. Rekisteröityminen peliin on mahdollista mutta pelivuoroja ei simuloida. Pelin kulloisenkin tilan näkee kirjautumissivulta (kuva 5).



Kuva 5: Pelin pääsivu. Rekisteröitymisiä otetaan vastaan.

Running-tilassa peli on toiminnassa. Pelivuorot kuluvat ja pelaajat tekevät siirtojaan. Peli pysyy running-tilassa niin kauan, kunnes joku pelaaja on voittanut.

Kun peli on ohi, siirrytään ended-tilaan. Tässä tilassa vuorot eivät kulu, vaan pelaajat näkevät viimeisen pelitilanteen. Tässä vaiheessa pelaajilla on aikaa hengähtää ja arvioida omaa pelisuoritustaan. Ended-vaiheen lopussa kaikki pelaajiin liittyvät tieto poistetaan tietokannasta ja alustetaan uusi pelikierros.

Kaupunkien ja aloituspaikkojen arpominen

Pelin kartta pysyy samana eli pelikierroksella. Kaupunkien paikat arvotaan kuitenkin satunnaisesti, ja näin jokainen pelikerta on hieman erilainen. Kaupunkeja ei arvota keskelle vettä, koska armeijat eivät pysty ylittämään vesialueita.

Kuten kaupungit, niin myös pelaajien aloituspaikat arvotaan. Aloituspaikka on aina satunnaisesti valittu kaupunki.

6.3 Reaaliaikaisuus

Peliin haluttiin minuutin pituiset pelivuorot. Tämä tarkoittaa, että pelaajilla on minuutti aikaa päättää jokaiselle armeijalleen ja kaupungilleen yksi toiminto. Kun minuutti on kulunut, kaikki pelaajien valitsemat toiminnot toteutetaan.

Jotta skriptejä voitaisiin suorittaa jaksollisesti, tarvitaan ajastinohjelma. Unix-ympäristössä itsessään on cron-ohjelma, jolla voidaan ajaa komentoriviskriptejä ajastetusti. Pienin intervalli on yksi minuutti, ja se riittää pelin tarpeisiin.

Cronissa ajastus hoidetaan muokkaamalla crontab-tiedostoa. Syntaksi on muotoa

<minuutti> <tunti> <päivä> <kuukausi> <viikonpäivä> <komento>

Minuutti on väliltä 0–59, tunti on väliltä 0–23, päivä on väliltä 0–7, kuukausi on väliltä 1–12 ja viikonpäivä on väliltä 0–7. Jokainen aikaa ilmaiseva syöte voidaan korvata tähdellä (*), joka tarkoittaa, että komento suoritetaan jokaisena tätä aikaa ilmaisevana ajankohtana. Jos esimerkiksi <tunti> korvataan tähdellä, skripti suoritetaan joka tunti, kunhan kaikki muut ajankohdat täsmäävät myös.

<komento>-kohtaan lisätään suoritettava skripti tai komento. Komentoja voidaan suorittaa useita, mikäli kaikki komennot lisätään kaarisulkeiden sisään ja jokainen komento erotetaan puolipisteellä.

Palvelu saapumisjärjestyksessä

Pelaajien siirrot käsitellään siinä järjestyksessä, missä ne saapuvat palvelimelle. Vaikka pelaajilla on minuutti aikaa siirtojen tekemiseen, joissain tilanteissa on eduksi tehdä siirrot mahdollisimman nopeasti. Esimerkkinä on tilanne, jossa vastustaja on seuraavalla vuorollaan hyökkäämässä pelaajan kaupunkiin ja pelaajan oma armeija on kaupungin ulkopuolella. Jos kaupungissa on sillä hetkellä vain vähän puolustusta, on järkevää siirtää oma armeija kaupunkiin, ennen kuin vastustaja hyökkää sinne. Jos pelaaja ehtii siirtää ennen vastustajaa, ehtii kaupunki saamaan lisäjoukkoja. Jos taas vastustaja ehtii ensin, hyökkäys kaupunkia vastaan suoritetaan vuorosimulaatiossa ennen joukkojen siirtoa.

6.4 Taistelut ja kaupunkien valtaaminen

Mikäli pelaaja liikuttaa armeijan sellaiseen ruutuun, jossa on vastustajan armeija tai kaupunki, syntyy taistelu. Taistelussa ovat mukana molempien armeijoiden kaikki yksiköt. Taisteluissa on aina hyökkääjä ja puolustaja. Se, joka on liikkumassa toisen armeijan ruutuun, on hyökkääjä, ja toinen on puolustaja. Sama pätee, vaikka molemmat ovat liikkumassa samaan ruutuun: se jonka liikkumistoiminto kirjataan palvelimelle ensin, ehtii liikkumaan ruutuun, ja tällöin toinen pelaaja automaattisesti hyökkää ensin liikkuneen armeijan kimppuun. Puolustajalla on aina pieni etu hyökkääjään nähden.

Taistelut käydään aina samassa järjestyksessä yksiköiden hyökkäysnopeuden mukaan. Se yksikkö, jolla on suurin hyökkäysnopeus, saa hyökätä ensimmäisenä. Jos useilla yksiköillä on sama hyökkäysnopeus tai jos molemmissa armeijoissa on yksiköitä joiden hyökkäysnopeus on, ne hyökkäävät samanaikaisesti.

Yksiköiden hyökkäysvoima (tietokannan strength -sarake) lasketaan yhteen, ja sen jälkeen katsotaan vastustajan armeijan yksiköiden puolustusvoima (tietokannan armor-sarake). Yhteenlaskettu hyökkäysvoima jaetaan vastustajan yhteenlasketulla puolustusvoimalta, ja vastustaja menettää jakolaskun kokonaisluvun verran taistelijoita. Yksiköt, joilla on sama hyökkäysnopeus, saavat aina joka tapauksessa taistella toisiaan vastaan. Mutta vastustajalta saattavat tuhoutua ne yksiköt, joilla on alempi hyökkäysnopeus.

7 Sovelluksen toteutus

7.1 Versionhallinta

Projektin versiohallintateknologiaksi valittiin Subversion. Se on Apache Software Foundationin kehittämä ilmainen versionhallintajärjestelmä [36]. Tietovarasto perustettiin kotona sijaitsevalle Ubuntu-palvelimelle. Asiakasohjelmana, jolla tietovarastoon vietiin koodin versioita, käytettiin ilmaista TortoiseSVN-ohjelmaa.

Versionhallintaa käytettiin ohjelmakoodin ja pelin grafiikan tallentamiseen. Tietokantaa tallennettin tekemällä siitä erillinen varmuuskopiotiedosto, joka sitten voitiin tallentaa versiohallintaan. Varmuuskopiotiedostossa on MySQL-tietokannan taulujen luontilauseet, jotka sisältävät tarkat kuvauksen taulujen rakenteesta, sekä sisällönluontilauseet, jotka sisältävät tarkasti jokaisen sarakkeen tiedot.

Kaikkea kehitysympäristössä tietokantaan tallennettavaa tietoa ei ollut mielekästä tallentaa versiohallintaan. Esimerkiksi pelin rekisteröitymis- ja kirjautumismekanismia testattaessa peliin luotiin useita käyttäjiä. Tiedot olivat tärkeitä testausvaiheessa, mutta tuotantoympäristössä käyttäjälista koostuu ainoastaan oikeista käyttäjistä. Siksi tietokantaa ei tallennettu kokonaisuudessaan, vaan suurimmasta osasta tauluja tallennettiin ainoastaan niiden rakenteiden kuvaukset. Ainoastaan peliauleen kartan sisältämä map-tietokantataulu, yksiköiden tarkemmat tiedot sisältävä units-taulu ja maaston tiedot sisältävä terrain-taulu tallennettiin tietoineen päivineen, koska niiden sisältämiä tietoja voitiin käyttää suoraan myös tuotantoympäristössä.

7.2 Tuotantopalvelimen valinta

Projektin alkuvaiheessa ei ollut vielä selvillä, mille palvelimelle tuotantoversio sijoitettaisiin. Pelin Internet-kaistan käyttö arvioitiin pieneksi. Toisaalta peli on pseudoreaaliaikainen ja siinä on useita samanaikaisia pelaajia. On tilanteita, joissa siirrot pitäisi saada tehtyä nopeasti. Hidasteleva yhteys on käyttökokemuksen kannalta selkeä haitta. Koska kyseessä on ei-kaupallinen työ, ei maksullisia web-hotellipalveluja haluttu käyttää.

Seuraava vaihtoehto oli oppilaitoksen oma palvelin: jokaisella opiskelijalla on oppilaitoksen palvelimella oma tila, johon he voivat tallentaa omia tiedostojaan. Tässä tilassa sijaitsee myös erillinen osio, public_html-kansio, joka näkyy Internetiin. Niin ikään opiskelijoilla on mahdollista käyttää MySQL-tietokantaa. Oppilaitoksen ympäristö ei kuitenkaan soveltunut tälle projektille, sillä oppilaitoksen palvelimen asetuksissa on estetty viittaukset public_html-kansiossa sijaitsevista tiedoistoista sen ulkopuolelle.

Lopulta palvelimeksi valittiin kotona sijaitseva oma palvelin, jossa on Ubuntu-käyttöjärjestelmä. Tämä ei ollut optimiratkaisu, sillä kodin Internet-yhteys on aktiivisessa käytössä, mikä saattaa haitata pelin käyttökokemusta. Koska peli oli kuitenkin vasta kehitysasteella, ei tätä katsottu isoksi haitaksi.

Tuotantopalvelin koki laiterikon, kun projekti oli sijainnut siellä jonkin aikaa. Tämä oli merkittävä takaisku, joka keskeytti projektin etenemisen pitkäksi aikaa. Peruuttamatonta vahinkoa ei kuitenkaan tapahtunut. Vaikka sekä versionhallinta että tuotantoympäristö sijaitsivat tällä palvelimella, tiedot saatiin kuitenkin pelastettua, lukuun ottamatta versionhallinnan lokitiedostoja. Tämän jälkeen tehtiin päätös saattaa projekti loppuun kehitysympäristössä, keskittyen ainoastaan tietoturvaratkaisuihin.

7.3 Tuotantoonvienti ja sen ongelmat

Ohjelman siirto tuotantopalvelimelle tehtiin ottamalla versionhallinnasta ohjelmiston uusin versio. Se kopioitiin verkkoyhteyden kautta tuotantopalvelimelle vanhan ohjelmakoodin päälle. Mikäli tietokantaa oli versioiden välillä muutettu, vanha kanta poistetiin ja tilalle siirrettiin uusi. Mitään tietoja tuotantopalvelimen kannasta ei säästetty, vaan aina aloitettiin puhtaalta pöydältä.

7.4 Kehitysympäristön vaihto

Kun ohjelmisto siirrettiin tuotantokäyttöön, huomattiin, että ohjelma ei toimi samalla lailla kehityskäytössä kuin tuotantokäytössä. Tästä syystä kehitysympäristö siirrettiin Ubuntu-käyttöjärjestelmään, joka on lähempänä tuotantokäyttöä kuin Xampp-palvelin Windows-käyttöjärjestelmässä. Ubuntu asennettiin VMWare-virtuaalikoneeseen. Tämä helpotti koodin kirjoittamista.

8 Tietoturvan kannalta merkittävät toteutukset

8.1 Käyttäjien hallinta

Käyttäjien tunnistamistekniikka

Sovelluksen käyttäjien tunnistautumismekaniikaksi valittiin käyttäjänimi-salasanakirjautumisjärjestelmä. Tässä projektissa on tärkeintä, ettei kukaan muu pääse pelaamaan toisen pelaajan käyttäjällä. Pelaajan varsinaisella henkilöllisyydellä ei ole merkitystä. Samalla itse tehty tunnistautumisjärjestelmä tarjoaa tietoturvan kannalta mielenkiintoisimpia haasteita.

Järjestelmään rekisteröidytään ja kirjaudutaan sovelluksen etusivulta. Tiedot syötetään lomakkeeseen ja lähetetään POST-pyynnöllä palvelimelle. Tiedonsiirtoa ei ole salattu.

Kirjautumisfunkioon lisättiin viive usleep()-komennolla, joka vaihtelee satunnaisesti viiden mikrosekunnin ja neljän sekunnin välillä.

```
usleep(rand(5, 4000000));
```

Mikäli joku kokeilee käyttäjätunnuksia ja salasanoja mielivaltaisesti, hän ei pyynnön käsittelemisessä kuluvasta ajasta voi mitenkään päätellä, oliko käyttäjätunnus olemassa.

Jos kirjautuminen epäonnistuu, virheilmoitus on joka tapauksessa sama riippumatta siitä, onko käyttäjänimi olemassa tai onko salasana väärä. Näin mahdollinen hyökkääjä, joka kokeilee mielivaltaisesti eri käyttäjänimiä, ei tätä kautta saa tietoonsa edes käyttäjänimiä. Hyökkääjälle voi esimerkiksi olla arvokasta tietoa, että järjestelmässä on käyttäjänimi nimeltä ”Admin”.

Tiivistefunktion valinta

Tiivistefunktioksi valittiin Scrypt, joka on alun perin vuonna 2009 esitelty algoritmi. Se vaatii sekä paljon laskenta tehoa että paljon muistia ja on siksi vaikea murtaa. Isot las-

kentateho- ja muistivaatimukset tekevät tiivisteen muodostamisesta hitaampaa, jolloin myös mahdollisissa murtoyrityksissä kestää kauemmin.

Scrypt ei tule PHP-jakelupakettien mukana, vaan se pitää asentaa erillisenä moduulina. Tämän vuoksi asennettiin kryptaus-, dekryptaus- ja tiivistefunktioita sisältävä Libsodium-kirjasto.

Scryptin prosessori- ja muistivaatimuksiin voidaan vaikuttaa parametreilla. Riittävän monimutkaisen tiivisteen takaamiseksi funktiota käytetiin kirjaston mukana tulevilla vakioarvoilla.

Suola

Salasanoihin liitettiin suola-merkkijono, jotta niiden murtaminen olisi mahdollisimman vaikeaa. Merkkijonon pituudeksi valittiin sama kuin scrypt-tiivisteen pituus eli 101 merkkiä.

Kryptografisesti turvalliset merkkijonot ovat sellaisia, ettei niitä voi helposti ennustaa. Niiden luomiseksi tarvitaan satunnaislukugeneraattori. PHP tarjoaa rand()-funktiota, mutta sen tuottamat luvut eivät ole tarpeeksi satunnaisia. Siksi satunnaislukujen generointiin päädyttiin käyttämään random_compat-kirjastoa ja sen random_int()-funktiota.

Suolan muodostusfunktio rakennettiin hashes.php-tiedostoon. Suola ei ole tiivistefunktio, mutta sen luonteva paikka oli tiivisteen luontiin käytettävän funktion seurana. Random_compat-kirjaston lisäämisen jälkeen määritellään suolaan käytettävät merkit. Sen jälkeen niistä arvotaan suolan mittainen merkkijono kuvan 6 mukaisesti.

```
26
27      function produceSalt($saltSize = 101)
28      {
29        include_once "../core/random_compat-2.0.2/lib/random.php";
30
31        $salt = '';
32        $characters = 'abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ0123456789';
33        $max_character = mb_strlen($characters, '8bit') - 1;
34        for($i=0;$i<$saltSize;$i++)
35        {
36            $random_character = $characters[random_int(0, $max_character)];
37            $salt=$salt.$random_character;
38        }
39        return $salt;
40      }
```

Kuva 6: Suola-merkkijonon luonti random_compat-funktion avulla.

8.2 Tietokannan käyttö ja SQL-injektioiden torjunta

Tietokantaan pääsyä varten rakennettiin oma Database–luokka, joka hoitaa kaikki transaktiot peliä ohjaavien funktioiden ja tietokannan välillä. Jokaiselle eri tietokanta-operaatiolle rakennettiin oma metodinsa Database-luokkaan.

Tietokantakutsut hoidetaan PHP Data Objects (PDO) -rajapinnalla. PDO tukee useimpia käytettyjä tietokantatyyppejä. Database-luokka ja PDO-rajapinta alustetaan kuvan 7 mukaisella tavalla.

```
function __construct()
{
  try
  {
    include '../core/db_settings.php';
    $this->database = new PDO("mysql:host=$dbhost;dbname=$dbname", $dbuser, $dbpass);
    $this->database->setAttribute( PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION );
  }
  catch(Exception $e)
  {
    $message = "Method: ".__METHOD__." ".$e->getMessage();
    file_put_contents('../errorlogs/PDOErrors.txt', $message, FILE_APPEND);
    die("Error occured");
  }
  $this->validator = new Validator();
}
```

Kuva 7: Database-luokan alustus.

Yhteydenmuodostus tietokantaan tehdään database-luokan konstruktorissa. PDO ei hyväksy, että yhteyden muodostamiseen käytettävät käyttäjänimi, salasana ja tietokannan nimi haettaisiin ympäristömuuttujista. Siksi päädyttiin määrittelemään erillinen tietokannan asetustiedosto db_settings.php, johon tiedot lisätään. Kun tiedot sijaitsevat muualla, niitä on jälkikäteen helppo muuttaa tuotantoon viennin yhteydessä. Näillä tiedoilla muodostetaan PDO-luokan instanssia, joka määrätään ottamaan yhteys MySQL-tietokantaan. PDO-instanssi säilytään $database-muuttujaan. Seuraavaksi PDO-asetetaan tuottamaan poikkeus, mikäli tietokantaoperaatioissa tapahtuu virhe.

Tietokantalauseet kirjoitetaan poikkeuksenkäsittelylauseiden sisään. Toimintalogiikka sijaitsee try-lohkossa. Jos tässä logiikassa tapahtuu virhe, ohjelma tuottaa poikkeuksen. Catch-lohkossa tapahtuu poikkeuksen käsittely: Virhe otetaan kiinni ja kirjoitetaan levylle errorlogiin yhdessä sen metodin nimen kanssa, jossa ongelma tapahtui. Metodin nimi otetaan __METHOD__-vakiomuuttujasta. Database-luokan konstruktorissa vir-

heen tapahtuessa ohjelman ajo lopetetaan ja käyttäjälle tulostetaan geneerinen virheilmoitus.

Varsinaisissa tietokannan käsittelyoperaatioissa kutsutaan haluttua database-luokan metodia ja annetaan sille tarvittavat muuttujat. Metodin sisällä saadut muuttujat varmistetaan konstruktorissa määritellyn Validation-instanssin metodeilla. Jos validointi epäonnistuu, tietokantaoperaatioita ei suoriteta.

Tietokantaoperaatiot toimivat Kuvan 8 mukaisesti.

```
$query = "INSERT INTO map VALUES (?,?,?,?,?)";
$statement = $this->database->prepare($query);
$parameters = array($x, $y, $data, 0, 0);
$statement->execute($parameters);
```

Kuva 8: PDO-luokan käyttö tietoturvallisen syöttölauseen muodostamiseen.

Ensin muodostetaan SQL-lause, jossa muuttujat on korvattu kysymysmerkeillä. Seuraavaksi kyselylauseesta muodostetaan PDOn prepare-metodia käyttäen statement-luokan instanssi. Tällä tavalla estetään SQL-injektion mahdollisuus. Validoiduista paremetreistä rakennetaan taulukko. Lopulta kutsutaan $statementin execute-komentoa, jolle muodostettu taulukko annetaan parametrinä.

8.3 Käyttäjän syötteiden validointi

Kaikkea käyttäjältä saatavaa tietoa pidetään riskinä ohjelman turvallisuudelle, ja se validoidaan. Sen takia kaikki validointi tehdään palvelimen puolella.

Syötteiden validointiin rakennettiin oma Validation-luokkansa. Kun validaatio halutaan suorittaa, muodostetaan luokan olio ja kutsutaan sopivaa validointimetodia sopivilla käyttäjän syöttämillä parametreillä. Jos syötteet eivät läpäise prosessia, haluttua toimintoa ei suoriteta loppuun.

Jokaiselle validoitavalle tyypille määriteltiin enimmäispituus. Esimerkiksi id-tyyppisille muuttujille se on neljä merkkiä, nimimerkeille ja salasanoille 30 merkkiä. Pituuden ylittyessä validaatio palauttaa virheilmoituksen. Numeroita validoitaessa testataan, että

arvo todella on numeerinen, nimiä ja salasanoja validoitaessa, että ne ovat merkkijonoja.

Lopulta tarkistetaan myös, että syötteissä on ainoastaan ennalta määrättyjä merkkejä. Tähän käytetään säännöllisiä lausekkeita (engl. ”Regular Expression”), jota PHP:llä käytetään preg_match-funktiolla: ”if (!preg_match("/^[A-Za-z0-9-\s]{1,30}$/", $input))”. If-lause tarkistaa, että syöte sisältää ainoastaan isoja ja pieniä kirjaimia, numeroita ja väliviivan.

8.4 Muuttujien tulostaminen

Sovelluksessa on tarkkaan rajattu, mitä tietoa käyttäjät voivat syöttää: käyttäjänimi, salasana ja lisänimi validoidaan ja niissä sallitaan ainoastaan kirjaimet, numerot, välilyönti ja yhdysmerkki. Tällä varmistetaan, ettei tietoihin voi lisätä HTML- tai JavaScript-koodia. Tulevaisuudessa saatetaan sallia suurempi määrä erikoismerkkejä. On varmistuttava, että vaikka tietokantaan joutuisi sinne kuulumatonta tietoa, sitä ei tulosteta sellaisenaan.

Kaikki näkymissä tulostettavat muuttujat käsitellään htmlspecialchars()-funktion avulla. Funktio muuttaa html-kielelle merkitsevät merkit erilaiseen muotoon, ettei niistä muodostu HTML-koodia. Esimerkiksi &-merkki muutetaan muotoon & . Selain osaa näyttää &-tekstin oikein &-merkkinä, mutta HTML-koodia tästä ei synny.

9 Työn tulokset

Projektissa tietoturvan parantamiseen valitut metodit olivat ajantasaisia ja tehokkaita. Esitellyt tekniikat estävät monien yleisten hyökkäysmetodien käytön. Toki on muistettava, että tietoturva on juuri niin hyvä kuin järjestelmän heikoin lenkki; yksikin lipsahdus voi johtaa koko järjestelmän altistumiseen.

Projektin alussa päätettiin olla käyttämättä sovelluskehystä. Tämä oli sinänsä perusteltu ratkaisu. Koska projekti on kuitenkin verrattain laaja, olisi sovelluskehys luultavasti nopeuttanut projektin rakentamista huomattavasti. Tulevaisuudessa tämän kokoluokan projekteissa sovelluskehyksen käyttö on suositeltavaa.

Kun sovellusta alettiin siirtää tuotantoon, sen rakenteessa paljastui ongelmia esimerkiksi tiedostopolkujen kanssa: kehitysympäristössä tiedostopolut toimivat mutta tuotantopalvelimella kansioiden oikeudet olivat erilaiset, mikä esti kutsumasta tiedostoja samalla tavalla. Lisäksi itse tehty MVC-rakenne kasvoi nopeasti sekavaksi kokonaisuudeksi, jonka hallinnoiminen oli vaikeaa. Sovelluskehyksen mukana tuleva valmiiksi mietitty rakenne olisi helpottanut työn edistymistä.

Järkevä sovelluksen rakenne myös tukee turvallisen sovelluksen toteuttamista. Hyvin suunnitellussa ohjelmassa samaa koodia ei tarvitse toistaa useissa eri paikoissa, jolloin työmäärä ja mahdollisuudet virheiden tekemiseen pienenevät. Sekavassa projektissa helposti unohtaa tehdä muutokset kaikkiin paikkoihin. Samalla selkeä rakenne auttaa kokonaisuuden hahmottamisessa, jolloin mahdolliset ongelmakohdat tulevat paremmin esille.

Itse peli ei projektin aikana valmistunut. Tämä ei haitannut: vaikka tavoitteena olikin saada aikaan toimiva peli, pääosa oli kuitenkin luotettavan tietoturvan rakentamisessa. Siinä onnistuttiin.

10 Yhteenveto

Insinöörityön edetessä kävi selväksi, että turvallisen verkko-ohjelmiston rakentaminen on mahdollista PHP-ohjelmointikielellä. Verrattain pienellä panostuksella on mahdollista toteuttaa tietoturvan kannalta merkittäviä asioita. Samalla huomattiin, että tietoturvavalinnat ja niiden laajuus ovat tasapainoilua. Riippuu ohjelman käyttötarkoituksesta ja skaalasta, kuinka kattavat tietoturvaratkaisut järjestelmä vaatii tai on mielekästä toteuttaa.

Ohjelmointikielet ja -tekniikat ovat jatkuvasti kehittyviä kokonaisuuksia ja alituisen tarkastelun kohteena. Haavoittuvuuksia löydetään jatkuvasti, ja samalla laskentatehon kasvaessa vanhat tietoturvan kannalta riittävät keinot voivat menettää merkityksensä. Ohjelmoijan onkin hyvä seurata tarkasti käyttämiensä tekniikoiden ja ohjelmointikielten kehitystä ja tarvittaessa muuttaa rakentamiaan sovelluksia.

Ratkaisumalleja etsiessä pyörää ei kannata keksiä uudelleen: kun käytetään suosittua ohjelmointikieltä, on moni käyttäjä ollut samanlaisten haasteiden edessä aikaisemmin, jolloin tietoa on hyvin saatavilla. Tietoa etsiessä kannattaa kuitenkin olla kriittinen: kaikki tarjotut vastaukset eivät aina ole hyviä, eikä kehittäjäyhteisö ei aina ole asioista samaa mieltä. Esimerkiksi scrypt-tiivistefunktio on turvallinen ainoastaan, kun sitä käytetään oikeanlaisten parametrien kanssa.

Tulevaisuudessa strategiapelisovellus voitaisiin muuttaa käyttämään SSL-salausta ja lisätä tuki pelaajien omien näköiskuvien lisäämiseen. Tällöin voitaisiin tarkastella tiedostojen käsittelyyn liittyviä haasteita. Myös pelilliset ominaisuudet olisi hyvä rakentaa loppuun asti.

Lähteet

1 FAQ: General. Verkkodokumentti. Django Software Foundation. <https://docs.djangoproject.com/en/1.7/faq/general/#why-does-this-project-exist>. Luettu 23.5.2016.

2 Holovaty, Adrian & Kaplan-Moss, Jacob. 2009. The Definite Guide to Django: Web Development Done Right. Second Edition. Berkeley: Apress.

3 Getting Started with Rails. Verkkodokumentti. <http://guides.rubyonrails.org/getting_started.html>. Luettu 7.3.2012.

4 Hellsten, Christian & Laine, Jarkko. 2006. Beginning Ruby on Rails E-Commerce: From Novice to Professional. Berkeley: Apress.

5 Banga, Manish. Brief version history of ASP.NET with features. Verkkodokument-ti. 2015. Dot Net Study. <http://www.dotnetstudy.com/brief-version-history-of-aspnet-with-features?id=6>. Luettu 23.5.2016.

6 VB.Net - Web Programming. Verkkodokumentti. Tutorials Point. <http://www.tutorialspoint.com/vb.net/vb.net_web_programming.htm>. Luettu 23.5.2016.

7 Bochicchio, Daniele; Mostarda, Stefano & De Sanctis, Marco. 2011. ASPNET 4.0 in Practice. Shelter Island, NY: Manning Publications.

8 Doyle, Matt. 2010. Beginning PHP 5.3, Indianapolis: Wiley Publishing.

9 Historical yearly trends in the usage of server-side programming languages for websites. Verkkodokumentti. Q-Success <http://w3techs.com/technologies/history_overview/programming_language/ms/y>. Luettu 8.4.2016.

10 HTTP (Hypertext Transfer Protocol). 2006. Verkkodokumentti. TechTarget <http://searchwindevelopment.techtarget.com/definition/HTTP>. Luettu 22.4.2016.

11 Syme, Matthew & Goldie, Philip. Understanding Application Layer Protocols. 2004. Verkkodokumentti. Pearson Education, Informit. <http://www.informit.com/articles/article.aspx?p=169578>. Luettu 23.5.2016.

12 Cookies - Information that websites store on your computer. Verkkodokumentti. The Mozilla Foundation <http://support.mozilla.org/en-US/kb/cookies-information-websites-store-on-your-computer?redirectlocale=en-US&redirectslug=Cookies> Luettu 10.8.2012.

13 Hovi, Ari; Huotari, Jouni & Lahdenmäki, Tapio. 2005. Tietokantojen suunnittelu & indeksointi. Porvoo: Docendo

14 Litwin, Paul. 2003 Fundamentals of Relational Database Design. Verkkodokumentti. Rudy Limeback <http://r937.com/relational.html>. Luettu 15.4.2016.

15 Ballad, Tricia & Ballad, William. 2009. Securing PHP web applications. Upper Saddle River, NJ: Addison-Wesley

16 Cassim, Hussain. 2014. Using Frameworks to Build Websites and Web Applications. Verkkodokumentti. Open Source Training. <https://www.ostraining.com/blog/webdesign/frameworks/>. Luettu 12.4.2016.

17 Fells, David. 2005. PHP Application Development Part One. Verkkodokumentti. Developer Shed Network. <http://www.devshed.com/c/a/PHP/PHP-Application-Development-Part-One/1/>. Luettu 12.4.2016.

18 Apache's DocumentRoot directive. Verkkodokumentti. Dedicated Server School. <http://www.serverschool.com/dedicated-servers/apaches-documentroot-directive/>. Luettu 22.4.2016.

19 Document Root. Verkkodokumentti. Karelia Software. <http://karelia.com/sandvox/help/z/Document_Root.html>. Luettu 22.4.2016.

20 Pankkien Tupas–tunnistuspalvelun tunnistusperiaatteet. 2013. Verkkodokumentti. Finanssialan keskusliitto. <http://www.finanssiala.fi/maksujenvalitys/dokumentit/Tupas_tunnistusperiaatteet_v20c.pdf>. Luettu 17.4.2016.

21 Tunnistautuminen mobiilivarmenteella. 2015. Verkkodokumentti. Valtiokonttori. <http://www.suomi.fi/suomifi/suomi/asioi_verkossa/sahkoinen_tunnistus_ja_allekirjoitus/tunnistautuminen_mobiilivarmenteella/index.html>. Luettu 8.4.2016.

22 Facebook Login for Apps - Overview. Verkkodokumentti. Facebook. <https://developers.facebook.com/docs/facebook-login/overview/> verkkodokumentti. Luettu 8.4.2016.

23 Northcutt, Stephen. Security Laboratory: Cryptography in Business Series. Verkkodokumentti. SANS Technology Institute. <http://www.sans.edu/research/security-laboratory/article/hash-functions>. Luettu 9.4.2016.

24 Radack, Shirley, 2009. The cryptographic Hash Algorithm Family: Revision of the Secure Hash Standard and Ongoing Competition for New Hash Algorithms. Verkkodokumentti. National Institute of Standards and Technology. <http://csrc.nist.gov/publications/nistbul/March2009_cryptographic-hash-algorithm-family.pdf>. Luettu 9.4.2016.

25 Salted Password Hashing - Doing it Right. 2016. Verkkodokumentti. Defuse Security. <https://crackstation.net/hashing-security.htm>. Luettu 9.4.2016.

26 Brute Force Attack. Verkkodokumentti. Technopedia. <https://www.techopedia.com/definition/18091/brute-force-attack>. Luettu 9.4.2016.

27 Ms. Smith. 2014. Top 25 most commonly used and worst passwords of 2013. Verkkodokumentti. Network World. <http://www.networkworld.com/article/2226175/microsoft-subnet/top-25-most-commonly-used-and-worst-passwords-of-2013.html>. Luettu 9.4.2016.

28 Rouse, Margaret. 2005. Hosting (Web site hosting, Web hosting, and Webhosting). Verkkodokumentti. TechTarget. <http://fi.wikipedia.org/wiki/Webhotelli>. Luettu 5.9.2012.

29 Hodgetts, Benjamin. 2010. Setting the PHP Session Path. Verkkodokumentti. <http://robotbutler.org/article/37>. Luettu 21.4.2016.

30 Brown, Philip. 2013. How to save PHP Sessions to a database. Verkkodokumentti. Yellow Flag. <http://culttt.com/2013/02/04/how-to-save-php-sessions-to-a-database/>. Luettu 28.4.2016.

31 Shifflett, Chris. Essential PHP Security: Sessions and Cookies. Verkkodokumentti. <http://phpsecurity.org/ch04.pdf>. Luettu 21.4.2016.

32 Cross-Site Request Forgery (CSRF). 2015. Verkkodokumentti. OWASP Foundation. <https://www.owasp.org/index.php/Cross-Site_Request_Forgery_(CSRF)>. Luettu 28.4.2016.

33 Bulten, Wouter. 2009. Secure Your Forms With Form Keys. Verkkodokumentti. Envato Pty. <http://net.tutsplus.com/tutorials/php/secure-your-forms-with-form-keys/>. Luettu 28.11.2012.

34 Reviewing code for Cross-Site Request Forgery issues. 2010. Verkkodokumentti. OWASP Foundation. <https://www.owasp.org/index.php/Reviewing_Code_for_Cross-Site_Request_Forgery>. Luettu 28.11.2012.

35 Nagele, Chris. An introduction to version control. Verkkodokumentti. Wildbit. <http://guides.beanstalkapp.com/version-control/intro-to-version-control.html>. Luettu 11.4.2016.

36 Apache Subversion Features. Verkkodokumentti. The Apache Software Foundation. <https://subversion.apache.org/features.html>. Luettu 11.4.2016.