Kaksivaiheinen tunnistautuminen tekstiviestiyhdyskäytävän avulla

HAMK
HÄMEEN AMMATTIKORKEAKOULU
HÄME UNIVERSITY OF APPLIED SCIENCES

Ammattikorkeakoulututkinnon opinnäytetyö

Riihimäki, tietotekniikan koulutusohjelma

kevät, 2018

Juha Kylmälä

HÄMEEN AMMATTIKORKEAKOULU
HÄME UNIVERSITY OF APPLIED SCIENCES

Tietotekniikan koulutusohjelma
Riihimäki

Tekijä Juha Kylmälä **Vuosi** 2018

Työn nimi Kaksivaiheinen tunnistautuminen tekstiviestiyhdyskäytävän avulla

Ohjaajan nimi Toni Laitinen

TIIVISTELMÄ

Tämän opinnäytetyön tarkoituksena oli parantaa PHP-pohjaisen websovelluksen sisäänkirjautumisen tietoturvaa. Työssä rakennettiin toimiva tekstiviestiyhdyskäytävä ja sitä käyttävä kaksivaiheisen tunnistautumisen vaativa sisäänkirjautumissovellus.

Vaiheessa yksi käyttäjältä pyydetään sähköpostia ja salasanaa. Syötettyään tiedot oikein järjestelmä lähettää kertakäyttöisen numerosarjan käyttäjälle tekstiviestiyhdyskäytävän kautta. Vaiheessa kaksi käyttäjä tunnistetaan edellä mainitun numerosarjan avulla.

Tekstiviestiyhdyskäytävä rakennettiin luottokortin kokoisella Raspberry Pi -tietokoneella, johon liitettiin Huawein operaattorivapaa nettitikku SIM-kortilla varustettuna. Webpalvelin ja tietokantapalvelin sijaitsivat erillisellä kannettavalla tietokoneella. Sovelluksen rakentamisessa käytettiin PHP-ohjelmointikieltä olio-ohjelmoinnilla ja MVC arkkitehtuurilla.

Lopputuotoksena muodostui sovellus, jossa puutteellista autentikointia on rajattu varmistamalla käyttäjä kahdella toisistaan riippumattomalla tavalla. Yksistään jo tämä keino parantaa käyttäjän todennusta huomattavalla tavalla. Lisäksi sovelluksen tietoturvaa on parannettu suojaamalla arkaluontoinen tieto tietokannassa ja estämällä SQL injektiot käyttäjän syötteistä. Sovellukseen luotiin myös yksinkertainen lokikirja ja valvonta käyttäjän virheellisistä kirjautumisyrityksistä.

Avainsanat MVC, OWASP, PHP, raspberry, tekstiviestiyhdyskäytävä

Sivut 23 sivua, joista liitteitä 1 sivu

HÄMEEN AMMATTIKORKEAKOULU
HÄME UNIVERSITY OF APPLIED SCIENCES

Information Technology
Riihimäki

Author Juha Kylmälä **Year** 2018

Subject Two-step authentication through SMS gateway

Name of Supervisor Toni Laitinen

ABSTRACT

The purpose of this project was to develop better login security for a PHP-based web application. In this project I built real SMS gateway and a two-step authentication login application using the SMS gateway.

During the first step user is requested his email and password. When user inputs are correct the application sends disposed series of numbers to the user through the SMS gateway. During the second step the user is authenticated by his series of numbers.

The SMS gateway was built with a credit-card sized Raspberry Pi -computer attached to the Huawei network operator-free USB modem with a SIM-card. The web- and database servers were located in a standalone laptop. The web application was created with PHP programming language using object-oriented programming and MVC architecture.

As an outcome application user is verified by two separated authentications. By this way user authentication is significantly safer. In addition, web application information security level is higher by protecting sensitive data at the database and preventing SQL-injections from user inputs. The web application has also simple monitoring of failed user logins.

Keywords MVC, OWASP, PHP, raspberry, SMS gateway

Pages 23 pages including appendices 1 page

SISÄLLYS

1 JOHDANTO

Facebookin käyttäjien tietovuoto ja Euroopan Unionin tietosuoja-asetus ovat tänä vuotena nousseet suuren yleisön tietoisuuteen ja keskustelun aiheeksi. Websovelluksista on tullut massiivisia henkilötietojen varastoja, joten sovellusten tietoturvakin nousee entistä tärkeämpään rooliin. Viime aikoina käyttäjän kaksivaiheiset kirjautumismenetelmät ovat yleistyneet.

Työn tarkoituksena on parantaa PHP-pohjaisen websovelluksen sisäänkirjautumisen tietoturvaa. Työssä rakennetaan toimiva tekstiviestiyhdyskäytävä ja sitä käyttävä kaksivaiheisen tunnistautumisen vaativa sisäänkirjautumissovellus. Vaiheessa yksi käyttäjältä pyydetään sähköpostia ja salasanaa. Syötettyään tiedot oikein järjestelmä lähettää kertakäyttöisen numerosarjan käyttäjälle tekstiviestiyhdyskäytävän kautta. Vaiheessa kaksi käyttäjä tunnistetaan edellä mainitun numerosarjan avulla. Lopullisessa sovelluksessa on tarkoituksena, että sovelluksen tietoturva on parantunut.

Sovelluksen rakentamisessa keskitytään sisäänkirjautumiseen liittyviin toimintoihin, jolloin esimerkiksi rekisteröitymistoiminto on rajattu työn ulkopuolelle. Lisäksi työssä keskitytään erityisesti palvelinpuolen ohjelmointiin, koska asiakaspuolen turvallisuuteen liittyvät vaatimukset ovat suhteellisen helppo ohittaa esimerkiksi kieltämällä JavaScriptin käyttö sivustolla.

Opinnäytetyön alussa tutustutaan OWASP:n (Open Application Security Project) listaukseen, jossa esitellään kymmenen yleisintä ja kriittisintä websovellusten turvallisuushaavoittuvuutta. Kolmannessa kappaleessa esitellään tekstiviestiyhdyskäytävänä toimivan järjestelmän rakenne ja käyttöönottaminen. Neljännessä kappaleessa esitellään työssä käytetyt tekniikat ja rakennetaan PHP-pohjainen sisäänkirjautumissovellus, jonka jälkeen siirrytään työn yhteenvetoon.

Työn lähdeaineisto koostuu englanninkielisistä verkkolähteistä, joissa on ajantasaisinta tietoa työssä käytetyistä laitteista, tekniikoista sekä ohjelmointikielistä. Painettu materiaali jää nopeasti vanhaksi, kun tekniikka kehittyy jatkuvasti.

2 YLEISIMMÄT JA KRIITTISIMMÄT HAAVOITTUVUUDET (OWASP)

The Open Web Application Security Project (OWASP) on maailmanlaajuinen yleishyödyllinen yhteisö, joka keskittyy ohjelmistojen tietoturvan parantamiseen. OWASP:lla on useita erilaisia tietoturvallisuuden parantamiseen keskittyviä projekteja, joiden kautta se tuottaa oppaita ja ohjeistuksia tietoturvan saralla. OWASP TOP 10 (engl. Top Ten Project) -projektin tarkoituksena on esitellä kymmenen yleisintä ja kriittisintä web sovelluksien turvallisuushaavoittuvuutta. Listassa esitellään haavoittuvuuksien tuomat riskit ja annetaan ohjeita haavoittuvuuksien tunnistamiseksi sekä torjumiseksi. Lisäksi lista sisältää runsaasti linkkejä lisämateriaalin äärelle. Listan uusin versio julkaistiin vuonna 2017, kun sitä edeltävä versio oli julkaistu vuonna 2013. Vuosien saatossa projektin luomasta listasta on muodostunut eräänlainen de facto -standardi websovellusten alalle. (OWASP 2017.)

Seuraavassa esittelen TOP 10 -listan mukaisesti websovellusten yleisimmät ja kriittisimmät haavoittuvuudet.

2.1 Injektio

OWASP:n listauksen mukaan injektio on websovellusten yleisin haavoittuvuus. Injektiohaavoittuvuus muodostuu, kun tunkeutuja pystyy lähettämään haitallista koodia kyselyn tai komennon yhteydessä palvelimelle. Tämä voi tapahtua esimerkiksi silloin, kun websovellus pyytää käyttäjältä tietoja, joiden perusteella haetaan tietoa tietokannasta. Yleensä injektiohaavoittuvuuksia löytyy SQL, LDAP, XPath tai NoSQL kyselyiden yhteydessä. Injektiohaavoittuvuudet havaitaan yleensä tutkimalla koodia ja tätä varten on olemassa myös erilaisia haavoittuvuutta etsiviä skannereita. (OWASP 2017.)

Pahimmillaan haavoittuvuus voi johtaa koko järjestelmän haltuunottoon, mutta yleisimmin haavoittuvuus johtaa massiivisiin tietojen menetyksiin, varastamisiin tai niiden luvattomaan muuttamiseen. (OWASP 2017.)

2.2 Puutteellinen autentikointi

Puutteellisesta autentikoinnista on kyse silloin, kun tunkeutuja pystyy kirjautumaan järjestelmään luotettuna käyttäjänä, vaikka ei sitä olisikaan. Tämä voi tapahtua esimerkiksi silloin, kun käyttäjän on annettu syöttää järjestelmään jokin heikko salasana, kuten numerosarja ”1234”. Edistyneemmissä hyökkäyksissä tunkeutuja voi käyttää niin sanottua väsytyshyökkäystä (engl. brute force), jossa automaattinen ohjelma järjestelmällisesti yrityksen ja erehdyksen kautta kokeilemalla löytää oikean käyttäjätunnus ja salasana yhdistelmän. Vielä helpompaa hyökkäys on, jos tunkeutuja

käyttää toisesta järjestelmästä ryöstettyjä käyttäjätunnuksia ja salasanoja (engl. credential stuffing). (OWASP 2017.)

Puutteellisesta autentikoinnista on kyse myös silloin, jos jo kirjautunutta käyttäjää ei pystytä tunnistamaan luotettavasti järjestelmän sisällä tai jos käyttäjän kirjautumistietoja ei hävitetä oikealla tavalla uloskirjautumisen yhteydessä (engl. session management). Tällöin tunkeutuja pystyy kirjautumaan järjestelmään jopa ilman käyttäjätunnusta tai salasanaa. (OWASP 2017.)

Haavoittuvuus voi johtaa yksittäisen käyttäjätilin haltuunottoon ja esimerkiksi identiteettivarkauteen. Järjestelmän sisällä haavoittuvuuden vakavuus on riippuvainen haavoittuneen tilin käyttöoikeuksista. (OWASP 2017.)

2.3 **Arkaluontoisen tiedon paljastuminen**

Tässä haavoittuvuudessa yleensä arkaluontoinen tieto, kuten salasanat, terveystiedot tai luottokorttinumerot on tallennettu tietokantaan joko selkokielisenä tai huonosti salattuna. Toisaalta sovellus voi myös lähettää arkaluontoista tietoa asiakkaan selaimelta selkokielisenä palvelimelle tai päinvastoin, jolloin hyökkääjä voi saada tietoja haltuunsa ns. välistävetomenetelmän (engl. man-in-the-middle) avulla. (OWASP 2017.)

Käyttäjätilien määrällä mitattuna Suomen kolmanneksi suurin tietovuoto paljastui huhtikuussa 2018, kun Viestintävirasto ilmoitti erään suomalaisen websovelluksen tietomurron yhteydessä paljastuneen jopa 130 000 käyttäjän käyttäjätunnukset ja selväkieliset salasanat. (Viestintävirasto 2018.)

Viime aikoina myös Euroopan Unioni (EU) on kiinnittänyt huomiota tietosuojan sääntelyyn. Toukokuusta 2018 lähtien EU:n tietosuoja-asetus (GDPR eli General Data Protection Regulation) sääntelee mm. henkilötietojen keräämistä, käsittelyä ja luovuttamista sekä näihin liittyviä oikeuksia ja velvollisuuksia. Tietosuoja-asetuksen säännöt ovat samat kaikille EU:ssa toimiville henkilötietoja käsitteleville tahoille kotipaikasta riippumatta. (Suomen Yrittäjät ry 2018.)

2.4 **XML entiteetti -injektio (XML External Entities – XXE)**

XML on laajennettava merkintäkieli ja se tulee englannin kielen sanoista eXtensible Markup Language. XML -tiedosto sisältää pelkästään tekstiä ja sen tarkoitus on säilyttää ja siirtää tietoa, ei esittää sitä. Koska XML sisältää pelkästään tekstiä, niin sillä on helppo siirtää tietoa ohjelmointikielestä, sovelluksista tai järjestelmistä riippumatta toisiin. (W3Schools.com n.d.)

Tämä on uusi haavoittuvuus, jota ei ollut edellisessä vuoden 2013 versiossa. Hyökkääjä pystyy käyttämään haavoittuvuutta hyväkseen, jos hän

pääsee lataamaan omia XML -tiedostoja kohteen palvelimelle. Vaihtoehtoisesti hyökkääjä voi päästä muokkaamaan sovelluksen käyttämää XML -tietoa ja syöttämään sinne haitallista koodia. (OWASP 2017.)

Toiminta voi johtaa muun muassa luotettavan tiedon paljastumiseen palvelimelta tai palvelunestohyökkäykseen (engl. denial-of-service attack). (OWASP 2017.)

2.5 Rikkinäinen pääsynhallinta

Rikkinäisessä pääsynhallinnassa on kyse siitä, että käyttäjä näkee tai pääsee muokkaamaan sellaista tietoa, johon hänellä ei pitäisi olla oikeutta tai suorittaa sellaisia toimia, joita hänellä ei pitäisi olla. (OWASP 2017.)

2.6 Virheelliset turvallisuusasetukset

Tämä haavoittuvuus keskittyy lähinnä websovelluksen käyttämän palvelimen ja sinne asennettujen palveluiden konfiguraatioasetuksiin. Usein hyökkääjät pyrkivät etsimään näistä haavoittuvuuksia, kuten suojaamattomia tiedostoja taikka kansioita, järjestelmien oletussalasanoja jne. (OWASP 2017.)

2.7 Cross-Site Scripting (XSS)

OWASP:n mukaan XSS on toiseksi yleisin haavoittuvuus, jota on löytynyt jopa kahdesta kolmasosasta tutkittuja websovelluksia. XSS -hyökkäys ei varsinaisesti kohdistu websovellukseen, vaan sen käyttäjiin. Sovelluksen käyttäjille voidaan XSS-haavoittuvuutta hyväksikäyttäen syöttää selainpuolella suoritettava haitallinen koodi. Haittakoodi on saatu websovellukseen esimerkiksi lomakkeeseen syötettyjen tietojen avulla. Haittakoodi suoritetaan käyttäjän selaimessa näkymättömissä, joten käyttäjä ei välttämättä edes tiedä mitä on tapahtunut. (OWASP 2017.)

Tämän haavoittuvuuden kautta hyökkääjä voi saada käyttöönsä käyttäjän istunto- ja evästetietoja, joiden avulla hyökkääjä pääsee järjestelmään. Pahimmillaan hyökkääjä voi asentaa käyttäjän selaimeen esimerkiksi näppäimistön tallennusohjelman, jolloin hyökkääjälle välittyy kaikki käyttäjän selaimessa tekemät näppäinten painallukset. (OWASP 2017.)

2.8 Suojaamaton serialisoinnin takaisinkääntäminen

Serialisointia (engl. serialize) voi verrata ohjelmoinnissa tiedon sarjoittamiseen. Joskus tietoa halutaan sarjoittaa, jolloin tieto on syötetty pitkäksi merkkisarjaksi. Merkkisarja sisältää yleensä yksittäisten tietojen tyypin, tiedon pituuden sekä sisällön. Seralisointia voidaan käyttää esimerkiksi syötettäessä tietoa toiselta ohjelmalta toiselle tai tallennettaessa tietoa

tietokantaan. Takaisinkääntämisessä (engl. deserialization) tieto luetaan merkkijonosta esimerkiksi takaisin muuttujien sisällöksi.

Suojaamattomana takaisinkääntämisessä hyökkääjä voi päästä syöttämään ohjelmalle itseluomiaan merkkijonoja, joissa on esimerkiksi muutettu käyttöoikeustasoa paremmaksi. Näin hyökkääjä saa itselleen lisää oikeuksia, jos ohjelma kääntää hyökkääjän syöttämän merkkijonon ilman tarkistuksia.

2.9 **Vanhentuneiden järjestelmän osien käyttö**

Jos palvelimen käyttämä käyttöjärjestelmä, tietokantajärjestelmä tai jokin muu järjestelmän osa pääsee vanhentumaan, niin kyseessä on turvallisuusuhka. Nykyään websovellusten rakentamista helpottavat muun muassa useat tarjolla olevat kirjastot (engl. library) ja valmiit kehykset (engl. framework). Näistäkin kuitenkin löydetään haavoittuvuuksia ja niitä päivitellään turvallisemmiksi ja paremmiksi ajan myötä. (OWASP 2017.)

2.10 **Tehottomat lokikirjat ja valvonta**

Onnistuneimmat hyökkäykset alkavat usein haavoittuvuuksien skannauksella, jolla hyökkääjä pyrkii selvittämään järjestelmässä olevia heikkouksia erilaisin menetelmin ja testauksin. Jos näitä skannauksia pystyy tekemään tarpeeksi paljon ja usein, niin hyökkäyksen onnistuminen voi olla täydellinen ja tuho laajamittainen. Vuonna 2016 hyökkäyksien selvittäminen kesti keskimäärin 191 päivää tehottomien lokien ja valvonnan vuoksi. (OWASP 2017.)

3 TIETOKANTAPOHJAISEN TEKSTIVIESTIYHDYSKÄYTÄVÄN RAKENTAMINEN

Tässä luvussa esittelen rakentamani tekstiviestiyhdyskäytävän rakenteen ja laitteistot. Lisäksi luvussa esitellään tekstiviestiyhdyskäytävän käyttämä tietokanta ja yhdyskäytävän asentaminen, konfigurointi sekä testaaminen.

3.1 Järjestelmän rakenne ja laitteistot

Myöhemmin esiteltävää demosovellusta varten tarvittiin PHP -ohjelmointikieleen soveltuvat web- ja tietokantapalvelimet. Palvelimet asennettiin kannettavalle tietokoneelle, jossa oli Windows -käyttöjärjestelmä. Palvelinten käyttöönotto tehtiin XAMPP -paketilla, joka on avoimen lähdekoodin paketti. XAMPP -sisältää Apachen webpalvelimen, MariaDB:n tietokantapalvelimen sekä PHP ja Perl -ohjelmointikielien tuet (Apache Friends 2018).

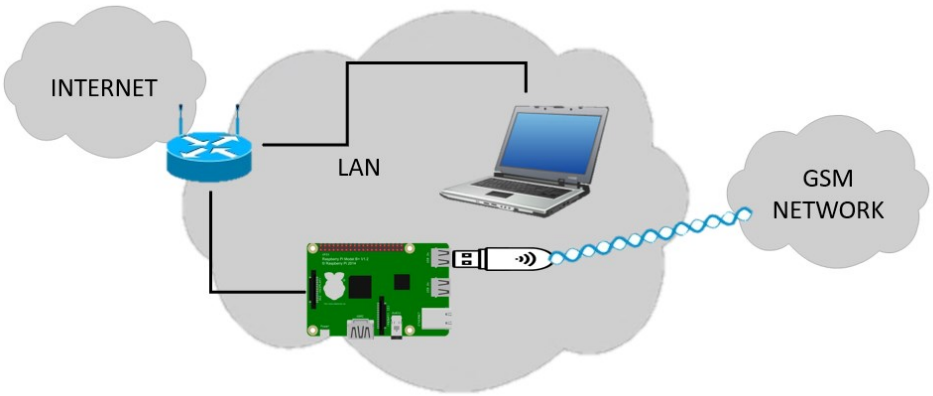
Tekstiviestiyhdyskäytävä on rakennettu käyttäen luottokortin kokoista Raspberry Pi -nimistä yhden piirilevyn tietokonetta. Sen on kehittänyt brittiläinen Raspberry Pi Foundation, mikä julkaisi ensimmäisen tietokoneen vuonna 2012. Uusin kolmannen sukupolven versio tietokoneesta on julkaistu vuonna 2016. Tässä työssä käytettiin tietokoneen toisen sukupolven versiota (Kuva 1), jossa on 32 -bittinen 900 MHz:n neliytiminen prosessori, 1 Gt:n näytönohjaimen kanssa jaettu sisäinen RAM -muisti ja muun muassa verkkosovitin, neljä USB 2.0 -porttia oheislaitteita varten, HDMI -portti näytön kytkemiseen ja Micro SD -muistikorttipaikka. Raspberry Pi:ssä ei ole lainkaan kiinteää kovalevyä, kuten perinteisissä tietokoneissa. Tietokoneen massamuistina toimii Micro SD -muistikortti, jossa käyttöjärjestelmä, ohjelmistot ja tiedostot säilytetään. Virtalähteenä toimii 5 V:n MicroUSB -liitin, jollaisia esimerkiksi matkapuhelinten laturit ovat. (Raspberry Pi Foundation 2018.)

Kuva 1. Työssä käytettiin Raspberry Pi -tietokoneen version 2 mallia B, joka julkaistiin vuonna 2015 (Raspberry Pi Foundation 2018).

Raspberry Pi -tietokonetta käytettiin uusimmalla Raspbian-nimisellä käyttöjärjestelmällä, joka on optimoitu juuri Raspberry Pi -tietokoneita varten (Raspbian.org n.d.).

Tekstiviestiyhdyskäytävän rakentamisessa varten tarvittiin tekstiviestien lähettämiseen soveltuva laite. Tässä työssä käytettiin Huawein operaattorivapaata mobiililaajakaistaliittymäksi soveltuvaa nettitikkua, joka kytkettiin Raspberry Pi -tietokoneen USB -porttiin. Nettitikkuun kytkettiin Telia -matkapuhelinoperaattorin prepaid tekstiviestiliittymä eli SIM -kortti, jolla saatiin yhteys matkapuhelinverkkoon.

Kuvassa 2 on havainnollistettu järjestelmän kokonaisrakenne laitteistoineen ja verkkoineen. Demosovellus ja tietokannat sijaitsevat kannettavalla tietokoneella, joka on yhteydessä lähiverkkoon ja internettiin kotireitittimen kautta. Lähiverkossa sijaitsee myös Raspberry Pi -tietokone, jolla on yhteys matkapuhelinverkkoon nettitikun avulla.



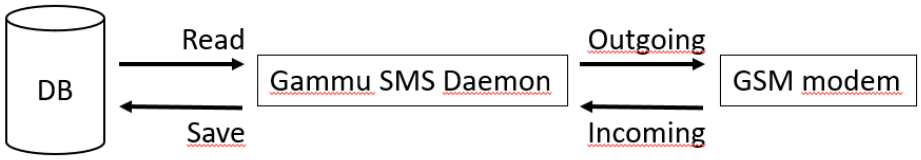
Kuva 2. Järjestelmän kokonaisrakenne

3.2 Tietokannan ja GSM -verkon yhdistäminen

Gammu on projektinimi ja komentopohjainen ohjelmointirajapinta, jolla voi ohjata matkapuhelimia. Nykyisin projektia johtaa Michal Čihař. Gammu on ohjelmoitu C -ohjelmointikielellä ja sillä pystyy mm. soittamaan, vastaanottamaan ja ohjaamaan puheluita sekä lähettämään ja vastaanottamaan tekstiviestejä sekä multimediaviestejä. Gammu tukee nykyään yli 3000 erilaista GSM -verkkoon yhdistyvää laitetta (matkapuhelimet, USB -tikut jne.) ja sen voi ladata niin Windows kuin Linux käyttöjärjestelmillekin. (Čihař n.d.)

Gammu projektissa on rakennettu myös Gammu SMS daemon (SMSD) ohjelma, joka käyttää hyväkseen edellä mainittua Gammu -kirjastoa. SMSD:llä matkapuhelinverkon laite voidaan yhdistää erilaisiin tietokantoihin, jolloin tekstiviestejä voidaan lähettää tietokannan kautta tai tulevia viestejä tallentaa tietokantaan. SMSD mahdollistaa tekstiviestien automatisoinnin ja massahallinnan. SMSD tukee MySQL, PostgreSQL, SQLite ja ODBC tietokantoja, kuten Microsoft Access ja Oracle. (Čihař n.d.)

Kuvassa 3 on havainnollistettu Gammu SMS Daemonin toimintaperiaate mukaillen käyttöohjetta (Čihař n.d.).

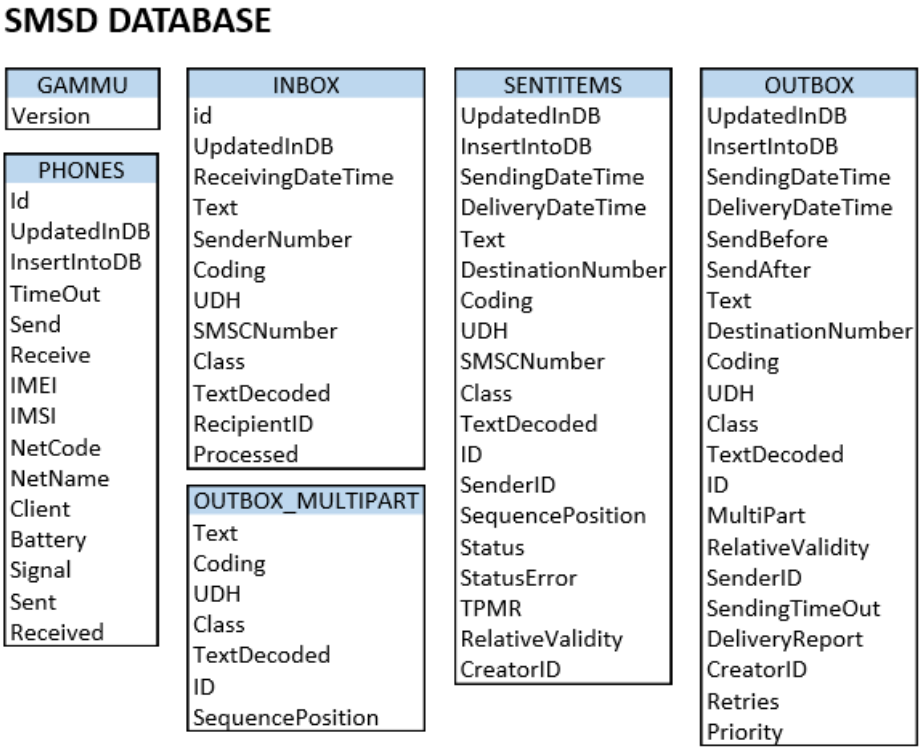


Kuva 3. Gammu SMS Daemonin toimintaperiaate

3.2.1 SMSD:n vaatima tietokanta

Ennen Gammu SMSD:n asentamista ja konfigurointia piti luoda sovelluksen käyttöä varten tietokanta. Tietokannan luomista varten Gammu -projektin internetsivuilta löytyi valmiit SQL -komennot, joilla saatiin tietokanta rakennettua.

SMSD tietokanta sisältää kuusi taulua, joita ovat; gammu, phones, inbox, outbox, outbox_multipart ja sentitems (Kuva 4). Seuraavassa esitellään lyhyesti jokaisen taulun tarkoitus ja sisältö.



Kuva 4. SMSD tietokannan taulut ja niiden sisältämien solujen nimet

Taulussa gammu määritellään ainoastaan tietokannan versionumero, jota tarvitaan versiohallinnassa. Tauluun phones tallennetaan yhdistettyjen

matkapuhelinverkon laitteiden tietoja, kuten IMEI-numero, akun ja signaalin voimakkuudet sekä tilastotietoa lähetetyistä ja vastaanotetuista tekstiviesteistä. Inbox-tauluun tallennetaan laitteeseen saapuneet tekstiviestit, lähettäjien numerot ja ajankohdat. Vastaavasti outbox -tauluun tallennetaan viestit, joita halutaan omalla laitteella lähettää. Outbox -taulu sisältää vastaanottajien numerot, viestien sisällön ja tilatietoa viestin lähetyksestä. Jos lähetettävä tekstiviesti on yli 160 merkkiä pitkä, niin järjestelmä käyttää apunaan outbox_multipart -taulua, johon se tallentaa pitkien viestien lähetystietoja. Viimeisenä tietokannasta löytyy sentitems-taulu, johon viestit siirretään, kun outbox-taulussa niitä ei enää tarvita.

3.2.2 SMSD:n asentaminen ja konfigurointi

Gammu SMSD ja sen tarvitsemat riippuvuudet asennettiin Raspberry Pi -tietokoneelle komennolla sudo app-get install.

Tämän jälkeen Gammu SMSD konfiguroitiin muuttamalla konfigurointitiedostosta tietoja (Kuva 5). Konfigurointitiedosto löytyi polusta /etc/gammu-smsdrc.

```
1  # Configuration file for Gammu SMS Daemon
2
3  # Gammu library configuration, see gammurc(5)
4  [gammu]
5  port = /dev/ttyUSB1
6  connection = at19200
7
8  # Debugging
9  logformat = textall
10 debuglevel = 255
11 logfile = omalogi.log
12
13 # SMSD configuration, see gammu-smsdrc(5)
14 [smsd]
15 service = sql
16
17 #Database connection
18 driver = native_mysql
19 user = smsd
20 password = smsd
21 database = smsd
22 host = xxx.xxx.xxx.xx:3306
23
24 CommTimeout = 5
25 ResetFrequency = 120
```

Kuva 5. Gammu SMSD:n konfigurointitiedoston sisältö

Kaikki # -merkinnällä olevat rivit ovat kommenttirivejä, joten ne eivät vaikuta sovelluksen toimintaan. Rivillä 5 määriteltiin portti, johon matkapuhelinverkon laite on kytketty ja seuraavalla rivillä määriteltiin AT-komentoihin perustuva yhteystapa, jonka nopeudeksi asetettiin 19200 kbit/s.

Riveillä 11-13 määriteltiin vianetsintää varten, että kaikki mahdollinen tieto (255) tallennetaan tekstimuodossa (textall) omalogi.log -nimiseen tiedostoon.

Rivillä 15 määritellään käytettäväksi SQL-palvelu, joka käyttää rivillä 18 määritettyä ohjainta. Riveillä 19-22 on määritetty tietokantayhteyttä varten tunnistetietoja, kuten tietokannan nimi, käyttäjänimi, salasana ja tietokantapalvelimen IP-osoite sekä TCP-portti (3306).

Lisäksi olen lisännyt riville 24 asetuksen, jolla Gammu SMSD tarkistaa 5 sekunnin välein, että löytyykö tietokannan outbox -taulusta lähetettäviä viestejä. Lopuksi varmistan rivillä 25 matkapuhelinverkon laitteen hereillä pysymisen antamalla sille pienen herätyksen 2 minuutin välein.

Lopuksi varmistetaan, että palvelu on toiminnassa komennolla

systemctl status gammu-smsd.service

ja että tekstiviestin lähettäminen onnistuu komennolla

gammu-smsd inject TEXT <puhelinnumero> -text ”omaviesti”.

4 SOVELLUKSEN TOTEUTTAMINEN

Seuraavaksi rakennetaan kaksivaiheisen tunnistautumisen sovellus. Ennen sitä esittelen ja perustelen lyhyesti työssä käytettyjä ja valitsemiani ohjelmointitekniikoita.

Kuvassa 6 esitellään sovelluksen toimintaperiaate käyttäjän näkökulmasta. Sovelluksen tarkoituksena on, että käyttäjä kirjautuu ensin käyttäen sähköpostia ja salasanaa. Tämän jälkeen järjestelmä lähettää käyttäjälle kertakäyttöisen koodin tekstiviestillä, jonka käyttäjä joutuu syöttämään järjestelmän pyytämälle sivulle päästäkseen järjestelmän sisälle. Tällä tavoin järjestelmä varmistaa käyttäjän kahdella toisistaan riippumattomalla tavalla, mikä parantaa sovelluksen tietoturvaa.

Kuva 6. Sovelluksen toimintaperiaate käyttäjän näkökulmasta

Lisäksi sovellukseen luodaan yksinkertainen valvontatyökalu käyttäjien virheellisistä kirjautumisyrityksistä. Käyttäjän tili lukitaan, jos käyttäjä yrittää liian monta kertaa kirjautua virheellisillä tiedoilla.

4.1 Ohjelmointikieli (PHP)

PHP on ohjelmointikieli, joka soveltuu erityisesti dynaamisten websovellusten luomiseen. PHP:tä käytetään mm. seuraavissa websovelluksissa:

- kaikkien tuntema Facebook,
- avoimen lähdekoodin oppimisalusta Moodle sekä
- sisällönhallintajärjestelmä WordPress.

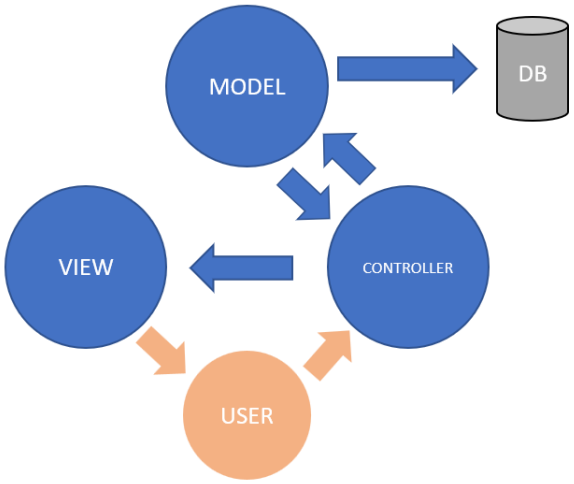
Ensimmäinen versio PHP:stä julkaistiin vuonna 1995 tanskalais-grönlantilaisen Rasmus Lendorfin toimesta ja uusin versio 7.2 julkaistiin toukokuussa 2018. PHP:tä voidaan käyttää käytännössä kaikilla tunnetuilla käyttöjärjestelmillä ja siitä löytyy laajat tuet useimpiin webpalvelimiin. PHP:n etuna voidaan pitää myös sen laajaa soveltuvuutta erilaisten tietokantapalvelimien kanssa.

4.2 Ohjelmointiparadigma (OOP) ja arkkitehtuuri (MVC)

Ohjelmointiparadigmalla tarkoitetaan ohjelmointikielen taustalla olevaa tapaa ajatella ja mallintaa ohjelmointitehtävän ratkaisu. PHP:tä voidaan käyttää niin proseduraalisena kuin oliopohjaisena.

Olio-ohjelmointi (engl. object-oriented programming, OOP) on ohjelmointiparadigma, jossa ratkaisut jäsennetään niin sanottujen olioiden yhteistoimintana. Olio käsittelee sisältämäänsä tietoa ja voi vastaanottaa viestejä ja lähettää tietoa muille olioille. Jokainen olio voidaan nähdä itsenäisenä pienenä koneena, jolla on tietty rooli tai vastuu. Olio-ohjelmointi selkiyttää ohjelmointia, jolloin ohjelmien laajentaminen ja ylläpito on helpompaa.

MVC tulee englannin kielen sanoista Model-View-Controller eli suomeksi Malli-Näkymä-Ohjain. MVC arkkitehtuurissa ohjelman koodi jaetaan kolmeen loogiseen osaan, eli komponenttiin (Kuva 7). Näin saadaan esimerkiksi graafinen käyttöliittymä erilleen ohjelman tietokantalogiikoista. Arkkitehtuuri helpottaa koodin jäsentelyä.



Kuva 7. MVC -arkkitehtuurin toimintaperiaate tässä työssä

Näkymä-komponentit sisältävät graafisen käyttöliittymän, jotka näytetään käyttäjälle. Kuvassa 6 on esitelty tässä työssä käytetyt kolme näkymää, jotka on rakennettu HTML ja CSS -ohjelmointikielillä.

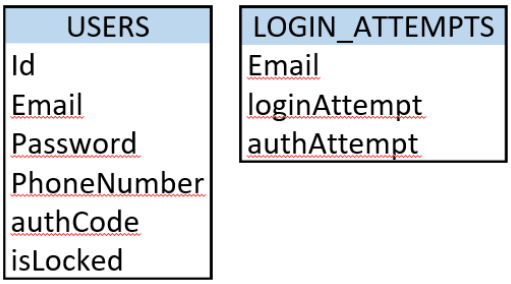
Mallit sisältävät käyttöperiaatteensa mukaisesti olioita omaan käyttötarkoitukseensa liittyen. Mallit voivat ottaa vastaan tietoa ja muokata sitä tai hakea tietoa tietokannoista.

Ohjaimet vastaavasti toimivat mallien ja näkymien välillä. Ohjain ottaa vastaan käyttäjän syötteitä ja toimittaa niitä malleille jatkokäsittelyyn. Ohjaimet myös vastaanottavat tietoa malleilta ja välittävät tiedot näkymien kautta käyttäjälle.

4.3 Tietokantataulut

Sovelluksen luominen aloitettiin lisäämällä aikaisemmin luotuun SMSD-tietokantaan kaksi taulua (Kuva 8). Users-taulu sisältää käyttäjän sähköpostin ja salasanan ensimmäistä kirjautumista varten. Kertakäyttöisen koodin tallentamista varten taulu sisältää paikan, johon koodi tallennetaan ja koodin lähettämistä varten taulusta löytyy käyttäjien puhelinnumerot. Lisäksi taulusta löytyy käyttäjän lukitustietoa kuvaava isLocked-solu.

Virheellisiä kirjautumisyrityksiä varten tietokantaan lisättiin myös login_attempts -niminen taulu, johon tallennetaan ensimmäisen vaiheen ja toisen vaiheen kirjautumisyritykset.



Kuva 8. SMSD tietokantaan lisätyt taulut

4.4 Tiedonsiirto ja tietokantaliityntä

Käyttäjän selaimen ja palvelimen välinen tiedonsiirto voidaan toteuttaa joko HTTP GET tai POST metodeilla. Tässä sovelluksessa käyttäjän syötteiden välityksessä käytetään HTTP POST metodia. Kyseinen metodi on turvallisempi kuin GET, sillä POST ei tallenna tietoja selaimen historiaan taikka lokitietoihin. POST -metodi ei myöskään näytä syötettyjä tietoja käyttäjän selaimen osoiterivillä selkokielisenä, kuten GET -metodi tekee. (W3Schools.com n.d.)

PHP tarjoaa useita vaihtoehtoja tietokantaliitynnän rakentamiseksi sovelluksen ja tietokannan välille. Itselleni tunnetuimmat ovat PDO (PHP Data Objects) ja MySQL. PDO on hiukan monipuolisempi, sillä siitä löytyy tuki jopa 12 eri tietokantaohjaimelle ja se tukee nimettyjä parametrejä. (Marjanovic 2012.)

PDO:n monipuolisuudesta huolimatta valitsin tietokantaliitynnän rakentamiseksi MySQLi:n, koska se on minulle tutumpi ja tietokanta toimi minulla MySQL -palvelimella. Kuvassa 9 on esitelty tietokantaliitynnän muuttujat ja yhteyden avaaminen. Lisäksi rivillä 8 asetetaan tietokantaliitynnälle merkistökoodaukseksi UTF-8, jolloin myös ääkköset toimivat.

```
1  //connect to database
2  $server = 'localhost';
3  $user = 'root';
4  $pass = '';
5  $db = 'smsd';
6  $charset = 'utf8';
7  $con_smsd= new mysqli($server, $user, $pass, $db);
8  $con_smsd->set_charset($charset);
```

Kuva 9. Tietokantaliitynnän muuttujat ja yhteyden avaaminen

4.5 SQL -injektioiden estäminen

Sovelluksessa käytetään tietokantakyselyissä PHP:n sisäisesti valmisteltuja kyselyitä (engl. prepared statements). Valmistelluissa kyselyissä kysely valmistellaan ensin prepare -funktiolla, kuten kuvassa 10 rivillä 1. Massiivisia tietovuotoja pyritään suojaamaan asettamalla SQL -kyselylle raja-arvoksi (engl. limit) 1, joka tarkoittaa, että haetaan tietokannasta vain yksi osuma näytettäväksi.

Hakusanojen paikoille kyselyyn asetetaan kysymysmerkki. Seuraavaksi asetetaan kysymysmerkin paikalle käyttäjän syöttämä tieto ja ilmoitetaan, että ko. tieto on tyyppiä ”s” eli string. Tämän jälkeen suoritetaan kysely tietokannalle, tallennetaan tulos ja syötetään kerätty tieto eri muuttujiin rivillä 5.

```
1  $stmt = $this->con_smsd->prepare("SELECT id, phoneNumber,
   password FROM " . $this->db_table_users . " WHERE email = ?
   LIMIT 1");
2  $stmt->bind_param("s", $email);
3  $stmt->execute();
4  $stmt->store_result();
5  $stmt->bind_result($id, $phoneNumber, $hash);
6  $stmt->fetch();
```

Kuva 10. Esimerkki tietokantakyselystä

4.6 Käyttäjän syötteiden suodattaminen ja varmentaminen

Sovelluksessa käyttäjä pääsee syöttämään vain sähköpostiosoitteen, salasanan ja kertakäyttöisen koodin. Salasanaa ei tarvitse suodattaa eikä varmentaa, koska se salataan 60 merkkisiksi merkkijonoiksi tekniikalla, joka esitellään seuraavassa kappaleessa.

Sen sijaan käyttäjän syöttämä sähköpostiosoite tulee suodattaa ja varmentaa, jotta käyttäjä ei pääse syöttämään järjestelmään mitään haitallista koodia. Ensin varmennetaan, että käyttäjän syöttämä tieto on oikeanlaista ja tämän jälkeen vielä suodatetaan käyttäjän syöttämä tieto.

Kuvassa 11 on esitelty käyttämäni suodatus- ja varmennusfunktiot, jotka löytyvät sisäänrakennettuina valmiina PHP -ohjelmointikielestä. Molemmat funktiot ottavat vastaan käyttäjän syöttämän sähköpostiosoitteen muuttujassa $email ja tekee tarvittavat toimenpiteet. Suodattaminen (sanitizeEmail) palauttaa suodatetun sähköpostiosoitteen takaisin. Vastaavasti varmentaminen (validateEmail) palauttaa varmennuksesta tiedon TYHJÄ (NULL) tai EPÄTOSI (FALSE).

```
1   public function sanitizeEmail($email)
2   {
3       $email = filter_var($email, FILTER_SANITIZE_EMAIL);
4       return $email;
5   }
6
7   public function validateEmail($email)
8   {
9       $result = NULL;
10      if(!filter_var($email, FILTER_VALIDATE_EMAIL))
11      {
12          $result = FALSE;
13      }
14      return $result;
15  }
```

Kuva 11. Käyttäjän syöttämän sähköpostiosoitteen suodatus ja varmennus

4.7 Arkaluontoisen tiedon suojaaminen ja tarkistaminen

Tässä sovelluksessa arkaluontoista tietoa ovat käyttäjän salasana ja kertakäyttöinen koodi, joka lähetetään käyttäjälle. Jotta sovelluksen tietoturva on parempi, niin nämä kaksi tietoa tulee salata tietokantaan. PHP:stä löytyy erilaisia tiedon suojaamiskeinoja. Osa keinoista on jo vanhentuneita, eikä niitä suositella käytettäväksi.

PHP:n ohjekirjasta löytyi hyvät ohjeet salasanojen tai vastaavien arkaluontoisten tietojen tallentamiseksi tietokantaan. Ohjeiden perusteella laadin olion, joka ottaa kuvan 12 rivillä 1 vastaan salattavan tiedon muuttujassa $password. Tämän jälkeen olio luo tiedosta 60 merkkiä pitkän salatun merkkijonon ja palauttaa sen takaisin esimerkiksi tallennettavaksi tietokantaan.

Vastaavasti salattua tietoa voidaan verrata käyttäjän syöttämään tietoon antamalla rivillä 10 oliolle ensin käyttäjän syöttämä tieto muuttujassa $inputPass ja tämän jälkeen haetaan salattu tieto tietokannasta muuttujaan $hash. Vertaus tehdään password_verify -funktiolla ja tuloksesta riippuen palautetaan joko TOSI (TRUE) tai EPÄTOSI (FALSE).

```php
public function securePassword($password)
{
    $options = [
        'cost' => 12,
    ];
    $hashed = password_hash($password, PASSWORD_BCRYPT, $options);
    return $hashed;
}

public function verifyPassword($inputPass, $hash)
{
    $check = FALSE;
    if(password_verify($inputPass, $hash))
    {
        $check = TRUE;
    }
    return $check;
}
```

Kuva 12. Arkaluontoisen tiedon suojaaminen ja tarkistaminen

4.8 Toisen vaiheen kirjautuminen

Tässä kappaleessa esittelen toisen vaiheen kirjautumisessa tarvittavat lähdekoodit MVC-arkkitehtuurin mukaisesti. Ensimmäisenä esitellään näkymän lähdekoodi (view), jonka jälkeen siirrytään ohjaimen (controller) esittelyyn ja lopuksi esitellään mallit (models).

Kuvassa 13 esitellään HTML-kielellä rakennetun lomakkeen lähdekoodi, jonka tuottama tulos näkyy luvun 4 alussa olevassa kuvassa 6. Tämä lomake näytetään käyttäjälle, kun ensimmäinen kirjautuminen käyttäjätunnuksella ja salasanalla on onnistunut.

Rivillä 3 määritellään, että lomake käyttää tiedonsiirtona HTTP POST -metodia, joka esiteltiin kappaleessa 4.4. Rivillä 6 luodaan tekstikenttä, johon käyttäjä voi syöttää tekstiviestillä saamansa kirjautumiskoodin. Tekstikenttä on nimetty ja yksilöity inputCode-tunnisteella. Ja lopuksi rivillä 8 luodaan painike, jolle on annettu nimeksi submit2.

```
1  //VIEW form of second authentication
2
3  <form class="form" method="post">
4  <h2 class="form">Please authenticate</h2>
5  <label for="authCode" class="label">Code</label>
6  <input type="text" id="inputCode" name="inputCode" class="form"
   placeholder="Code" required autofocus>
7  <br>
8  <button class="btn" name="submit2" type="submit">Sign in</button>
9  </form>
```

Kuva 13. Toisen vaiheen näkymän (view) HTML -koodi

Kun käyttäjä painaa submit2-painiketta, niin kuvassa 14 esitelty ohjain huomaa tämän rivin 4 ehtolauseessa ja alkaa toteuttamaan sille määriteltyjä tehtäviä. Ensimmäisenä ohjain syöttää rivillä 6 käyttäjän, lomakkeen

tekstikenttään, syöttämän pääsykoodin Validation-nimiselle mallille, josta löytyy olio nimeltään validateInputCode. Tämä olio varmentaa, että käyttäjän syöttämä pääsykoodi on oikeassa muodossa.

Seuraavaksi ohjain vertaa mallilta saatua vastausta ehtolauseessa ehtoon TRUE rivillä 8. Jos ehtolause on totta, niin ohjain syöttää seuraavaksi pääsykoodin Login-mallista löytyvälle check_auth_login -oliolle, joka tekee omat tarkistuksensa.

Jälleen rivillä 11 verrataan mallilta saatua vastausta ehtolauseessa. Jos kaikki on oikein, niin käyttäjä ohjataan lopuksi rivillä 13 Template-mallin kautta inside.php -näkymään.

Jos taas rivin 8 ehtolause ei ole totta, niin ohjain käskee Login-mallin olion resetAuthCode-töihin, ohjain itse tyhjentää ja tuhoaa istuntotiedot riveillä 19 ja 20, luo käyttäjälle varoituksen rivillä 21 Template-mallin kautta ja lopuksi ohjaa käyttäjän takaisin ensimmäiseen kirjautumiseen rivillä 22.

Ja mikäli submit2-painiketta ei ole painettu, niin ohjain ohjaa käyttäjälle näkyväksi ensimmäisen kirjautumissivun.

Tässä huomataan, että ohjain (controller) on erittäin keskeisessä roolissa sovelluksen toiminnassa, sillä se sisältää kaiken toiminnallisuuden. Ohjain on sovelluksen ydin, joka käyttää apunaan malleista löytyviä olioita ja näkymiä.

```
1  //CONTROLLER of second authentication
2
3  <?php
4  if(isset($_POST['submit2']))
5  {
6      $validateInputCode = $Validation-
       >validateInputCode($_POST['inputCode']);
7
8      if($validateInputCode === TRUE)
9      {
10         $accepted2 = $Login->check_auth_login($_POST['inputCode']);
11         if($accepted2 === TRUE)
12         {
13             $Template->load('app/views/inside.php', 'Sign in');
14         }
15     }
16     else
17     {
18         $Login->resetAuthCode();
19         session_unset();
20         session_destroy();
21         $Template->set_alert("Invalid code, try again", "warning");
22         $Template->load('app/views/home.php', 'Sign in');
23     }
24 }
25 else
26 {
27     $Template->load('app/views/home.php', 'Sign in');
28 }
```

Kuva 14. Toisen vaiheen ohjaimen (controller) PHP-koodi

Kuvaan 15 on koottu kaikki ohjaimen tarvitsemat oliot, jotka löytyvät varmennus (Validation), sisäänpääsy (Login) ja sapluuna (Template) -malleista (models).

```
1  //Second authentication functions from models
2
3  <?php
4
5  //from validation model
6  public function validateInputCode($code)
7  {
8      $result = FALSE;
9      if(filter_var($code, FILTER_VALIDATE_INT))
10     {
11         $result = TRUE;
12     }
13     return $result;
14 }
15
16 //from template model
17 public function load($url, $title = '')
18 {
19     if ($title != '') { $this->set_data('page_title', $title); }
20     include($url);
21 }
22
23 public function set_alert($value, $type = 'success')
24 {
25     $_SESSION[$type][] = $value;
26 }
27
28 //from login model
29 public function check_auth_login($inputCode)
30 {
31     $accepted = FALSE;
32     $id = $_SESSION["id"];
33
34     $stmt = $this->con_smsd->prepare("SELECT * FROM " . $this-
       >db_table_users . " WHERE id = ? AND authCode = ? LIMIT 1");
35     $stmt->bind_param("ii", $id, $inputCode);
36     $stmt->execute();
37     $stmt->store_result();
38
39     if($stmt->num_rows === 1)
40     {
41         $accepted = TRUE;
42     }
43     $stmt->close();
44     return $accepted;
45 }
46
47 public function resetAuthCode()
48 {
49     $RandomNumber = $this->createAuthCode();
50     $id = $_SESSION["id"];
51
52     $stmt = $this->con_smsd->prepare("UPDATE "  . $this->db_table_users
       . " SET authCode = ? WHERE id = ?");
53         $stmt->bind_param("ii", $RandomNumber, $id);
54         $stmt->execute();
55         $stmt->close();
56 }
```

Kuva 15. Toisessa vaiheessa tarvittavat oliot, jotka on koottu kuvaan Validation, Login ja Template -malleista (models)

Jokaisella mallin oliolla on oma pieni tehtävänsä. ValidateInputCode -olio varmentaa, että käyttäjän syöttämä pääsykoodi on varmasti sarja numeroita ja palauttaa ohjaimelle tuloksen kyllä (TRUE) tai ei (FALSE).

Load-olio ohjaa käyttäjän tiettyyn näkymään eli lataa käyttäjälle esimerkiksi kuvassa 13 esitellyn koodin.

Set_alert-olio asettaa istuntoon oletuksena onnistuneita (success) ilmoituksia, mutta ilmoituksen tyyppi voi olla myös tyyppiä varoitus. Lisäksi olio ottaa käyttäjälle määritellyn ilmoituksen vastaan muuttujaan $value, josta se näytetään käyttäjälle.

Check_auth_login-olio ottaa vastaan käyttäjän syöttämän pääsykoodin ja vertaa sitä SQL-kyselyllä tietokantaan tallennettuun pääsykoodiin. Tietokantaan pääsykoodi on tallennettu samanaikaisesti, kun pääsykoodi lähetettiin tekstiviestiyhdyskäytävän kautta käyttäjälle. Olio palauttaa ohjaimelle tuloksen kyllä (TRUE) tai ei (FALSE).

Ja resetAuthCode-olio vastaavasti luo tietokantaan käyttäjälle uuden satunnaisesti luodun pääsykoodin. Tätä oliota käytetään muun muassa silloin, kun käyttäjän syöttämä pääsykoodi on väärin.

Tästä esimerkistä huomataan, että mallit helpottavat mm. olioiden jäsentelyä huomattavalla tavalla.

4.9 **Sovelluksen logiikka**

Liitteessä 1 pyritään havainnollistamaan sovelluksen logiikkaa eli sitä, että mitä ja missä vaiheessa tehdään mitäkin toimintoja ja kutsutaan eri funktioita. Seuraavassa esittelen sovelluksen logiikkaa vaiheittain.

1. Kun käyttäjä tulee sovelluksen etusivulle (index.php), niin ohjain huomaa, että käyttäjä ei ole vielä kirjautunut ja täten ohjaa hänelle home.php -näkymän.
2. Käyttäjä syöttää sähköpostin ja salasanan sekä painaa nappia, jolloin ohjain huomaa tämän ja vastaanottaa käyttäjän syöttämät tiedot.
3. Ohjain syöttää ensimmäisenä käyttäjän sähköpostin varmennukselle, joka varmentaa, että sähköposti on oikeassa muodossa. Jos se ei ole, niin ohjain palauttaa käyttäjän home.php -näkymään.
4. Ohjain syöttää käyttäjän sähköpostin suodattimen läpi.
5. Ohjain pyytää isLocked -funktiota tarkistamaan, että onko käyttäjän tili lukittu. Jos on, niin ohjain palauttaa käyttäjän home.php -näkymään.
6. Ohjain pyytää checkLogin -funktiota tarkistamaan, että kirjautuminen on sallittua. checkLogin käyttää apunaan verifyPasswordia.
7. Jos kirjautuminen ei ole sallittua, niin tarkistetaan lokikirjasta käyttäjän kirjautumisyritysten lukumäärä.
8. Lukumäärän perusteella käyttäjä joko lukitaan tai tietokantaan tallennetaan yksi virheellinen kirjautumisyritys. Käyttäjä palautetaan home.php -näkymään.
9. Ohjain pyytää luomaan kertakäyttöisen satunnaisen koodin, jonka jälkeen koodi tallennetaan käyttäjän riville users -tauluun sekä

valmistellaan lähetettäväksi käyttäjälle tekstiviesti, joka tallennetaan outbox -tauluun.

10. Ohjain näyttää käyttäjälle auth.php -näkymän ja Gammu SMSD havaitsee outbox -taulun rivin, jolloin viesti välitetään nettitikulle, joka lähettää tekstiviestin käyttäjälle matkapuhelimeen.
11. Käyttäjä syöttää koodin näkymään ja painaa nappia, jolloin ohjain ottaa syötteen vastaan, tarkastaa sen oikeellisuuden ja vertaa sitä tietokannassa olevaan.
12. Jos syöte on virheellinen, niin ohjain pyytää nollaamaan tietokannasta koodin ja käyttäjä palautetaan home.php -näkymään uusinta kirjautumista varten. Vastaavasti oikein ollessaan ohjain näyttää käyttäjälle inside.php -näkymän.

5 YHTEENVETO

Opinnäytetyön tarkoituksena oli rakentaa sovellus, jossa on kaksivaiheinen tunnistautuminen. Ensimmäisessä tunnistautumisessa käyttäjä tunnistettiin perinteiseen tapaan sähköpostin ja salasanan yhdistelmällä ja toisessa vaiheessa käyttäjä tunnistettiin sovelluksen luomalla kertakäyttöisellä numerosarjalla.

Lopputuotoksena muodostui sovellus, jossa puutteellista autentikointia on rajattu varmistamalla käyttäjä kahdella toisistaan riippumattomalla tavalla. Yksistään jo tämä keino parantaa käyttäjän todennusta huomattavalla tavalla. Lisäksi sovelluksen tietoturvaa on parannettu suojaamalla arkaluontoinen tieto tietokannassa ja estämällä SQL-injektiot käyttäjän syötteistä. Sovellukseen luotiin myös yksinkertainen lokikirja ja valvonta käyttäjän virheellisistä kirjautumisyrityksistä.

Sovelluksessa on otettu huomioon neljä kymmenestä OWASP:n listauksen mukaisista turvallisuushaavoittuvuuksista. Tähän kun lisätään vanhentuneiden järjestelmän osien ajan tasalle saattaminen sekä palvelinpuolen turvallisuusasetusten päivittäminen, niin saadaan jo suhteellisen pienellä vaivalla kuusi kymmenestä turvallisuushaavoittuvuutta korjattua. Jäljelle jääneet turvallisuushaavoittuvuudet, kuten XXE ja XSS, Insecure Deserialization ja pääsynhallinta, on otettava huomioon siinä vaiheessa, kun sovellukselle luodaan jotain sisältöä kirjautumisen jälkeen.

Valittujen tekniikoiden, kuten MVC -arkkitehtuurin ansiosta sovellusta on helppo kehittää, ylläpitää tai tarvittaessa muokata.

Mielestäni tämä työ mahdollistaisi myös jatkoa, sillä olisi mielenkiintoista selvittää rakennetun tekstiviestiyhdyskäytävän soveltuvuus useampien modeemien kanssa samanaikaisesti, ja että kuinka suurien viestimäärien lähettäminen onnistuisi. Lisäksi vaihtoehtoisten tunnistautumiskeinojen, kuten biometriset tunnistautumiset (sormenjälki, kasvojentunnistus jne.), tutkiminen olisi aiheellista.

Työ oli opettavainen kokemus. Tieto ja ymmärrys erityisesti turvallisen ohjelman luomisen osalta laajeni huomattavalla tavalla. Lisäksi ohjelmointitaidot karttuivat, sillä toimivan ohjelman rakentaminen vaati paljon yrityksiä ja erehdyksiä. Aion tulevaisuudessa käyttää tämän projektin kautta opittuja taitoja omissa ohjelmointiprojekteissani. Myös Linux -käyttöjärjestelmän osalta tuli lisää osaamista.

LÄHTEET

Apache Friends (2018). What is XAMPP? Haettu 25.5.2018 osoitteesta https://www.apachefriends.org/index.html

Čihař, M. (n.d.) Gammu. Haettu osoitteesta 25.5.2018 https://wammu.eu/gammu/

Čihař, M. (n.d.) Gammu SMSD. Haettu 25.5.2018 osoitteesta https://wammu.eu/smsd/

Čihař, M. (n.d.) SMSD Database Structure. Haettu 25.5.2018 osoitteesta https://wammu.eu/docs/manual/smsd/tables.html

Čihař, M. (n.d.) SMSD Overview. Haettu 25.5.2018 osoitteesta https://wammu.eu/docs/manual/smsd/overview.html#overall-schema

Marjanovic, D. (2012). PDO vs. MySQLi: Which Should You Use? Haettu 20.5.2018 osoitteesta https://code.tutsplus.com/tutorials/pdo-vs-mysqli-which-should-you-use--net-24059

OWASP (2017). OWASP Top 10 – 2017. The Ten Most Critical Web Application Security Risks. Haettu 20.5.2018 osoitteesta https://www.owasp.org/images/7/72/OWASP_Top_10-2017_%28en%29.pdf.pdf

Raspberry Pi Foundation (n.d.) Raspberry Pi 2 Model B specifications. Haettu 25.5.2018 osoitteesta https://www.raspberrypi.org/products/raspberry-pi-2-model-b/

Raspian.org (n.d.) Welcome to Raspbian. Haettu 25.5.2018 osoitteesta http://www.raspbian.org/

Suomen Yrittäjät ry (2018). Yrittäjän tietosuojaopas. Opas päivitetty 13.4.2018. Haettu 20.5.2018 osoitteesta https://www.yrittajat.fi/yrittajan-abc/yritystoiminnan-abc/yrittajan-tietosuojaopas-570864

Viestintävirasto (2018). Suomalaisten selväkielisiä salasanoja paljastunut. Haettu 20.5.2018 osoitteesta https://www.viestintavirasto.fi/kyberturvallisuus/varoitukset/2018/varoitus-2018-01.html

W3Schools.com (n.d.) Introduction to XML. Haettu 20.5.2018 osoitteesta https://www.w3schools.com/xml/xml_whatis.asp

W3Schools.com (n.d.) HTTP Request Methods. Haettu 20.5.2018 osoitteesta https://www.w3schools.com/tags/ref_httpmethods.asp

Liite 1

SOVELLUKSEN LOGIIKKA FUNKTIOTASOLLA