LAHDEN AMMATTIKORKEAKOULU
Lahti University of Applied Sciences

Web-kehitys ja tietoturva

LAHDEN
AMMATTIKORKEAKOULU
Liiketalouden ala
Tietojenkäsittelyn koulutusohjelma
Opinnäytetyö
Syksy 2013
Risto-Pekka Nykänen

Lahden ammattikorkeakoulu
Tietojenkäsittelyn koulutusohjelma

Nykänen, Risto-Pekka Web-kehitys ja tietoturva

Tietojenkäsittelyn koulutusohjelman opinnäytetyö, 31 sivua, 0 liitesivua

Syksy 2013

TIIVISTELMÄ

Yhtenä suurimpana ongelmana web-kehityksessä on erilaiset tietoturva-aukot ja haavoittuvuudet. Yleisimpiä syitä web-sovelluksen haavoittuvuuksille on sovelluskehittäjän tekemät virheet tai riittämätön tietämys tietoturvasta.

Tämän opinnäytetyön tarkoituksena on tarkastella tietoturvallisen web-sovelluksen kehitystä ja sitä, mihin erilaisiin tietoturvaongelmiin web-kehittäjän on varauduttava kehittäessään sovellusta käytettäväksi verkkoympäristössä. Opinnäytetyö pyrkii etsimään vastausta siihen, millä keinoin ja mitä asioita huomioon ottaen voidaan luoda turvallinen web-sovellus.

Opinnäytetyössä esitellään yleisimmät web-sovelluksen haavoittuvuudet ja tietoturva-aukot, jotka voivat syntyä käytettäessä vääränlaisia kehitystekniikoita tai inhimillisten virheiden seurauksena. Läpi käydään muun muassa se, milloin sovellus voi mahdollisesti sisältää haavoittuvuuden ja miten haavoittuvuutta voidaan hyväksikäyttää. Tämän jälkeen tarkastellaan haavoittuvuuden ehkäisemiseen käytettäviä tapoja ja se, miten haavoittuvuudet ehkäistään kooditason ratkaisuilla. Tietoturvallisia ratkaisuja tarkastellessa hyödynnetään kahta yleisesti web-kehityksessä käytettyä tekniikkaa. Nämä tekniikat ovat PHP-ohjelmointikieli sekä MySQL-tietokantasovellus.

Opinnäytetyössä ei tarkastella palvelimen tai tietokannan konfiguroimisella saavutettaviin tietoturvaparannuksiin. Tarkastelun ulkopuolelle jää myös Julkaisujärjestelmien, ohjelmistokehyksien tai muiden kolmansien osapuolien tekemien sovelluksien käyttämisessä huomioitavat asiat.

Opinnäytetyön tuloksena syntyy kokoelma erilaisia tapoja ja käytänteitä, joita hyödyntämällä oikealla tavalla voidaan luoda tietoturvallinen sovellus.

Avainsanat: Web-kehitys, web-sovellus, tietoturva, PHP, MySQL

Lahti University of Applied Sciences
Degree Programme in Information Technology

NYKÄNEN, RISTO-PEKKA Web Development and Security

Bachelor's Thesis in Information technology 31 pages, 0 pages Of appendices

Autumn 2013

ABSTRACT

One of the biggest problems in web development is the different kinds of security issues and vulnerabilities. Probably the most common reason for vulnerabilities in web software is the mistakes made by web developers.

This thesis will introduce the most common vulnerabilities and security holes which may occur due to wrong kinds of development methods or human errors. It will also introduce the situations where the software may be vulnerable to certain type of attacks. The development techniques used to study this matter were PHP programming language and MySQL database management system.

This thesis only studies code level approaches on preventing security issues. Server and database configurations are excluded as is usage of third party applications like frameworks, class libraries and content management systems.

The result of this thesis is a collection of methods and techniques which may be used to create secure web application.

Key words: web development, web application, Security, PHP, MySQL

SISÄLLYS

1 JOHDANTO

Tämän opinnäytetyön tarkoitus on tuoda esiin yleisimmät tietoturvaongelmat, jotka voivat vaarantaa sovelluksen käyttäjän tietoturvan, tai sovellukseen luovutettujen tietojen turvallisuuden. Opinnäytetyössä käydään läpi yleisimmät web-sovelluksissa kohdattavat tietoturva-aukot ja tietoturvan ongelmakohdat ja se, kuinka ongelmakohdat voidaan ratkaista erilaisilla kooditason ratkaisuilla.

1.1 Aiheen rajaus

Tietoturvan tutkimisessa hyödynnetyt tekniikat valittiin niiden suuren suosion, sekä lähdekoodien avoimuuden takia. PHP on yksi suosituimmista palvelinpuolen skriptikielistä. MySQL-tietokantoja puolestaan käyttävät muunmuassa Google ja Facebook. Näiden seikkojen lisäksi monet avoimen lähdekoodin palvelinratkaisut tukevat kyseisten tekniikoiden käyttöä. Nämä tekniikat ovat helposti kaikkien web-kehityksestä kiinnostuneiden käyttöönotettavissa ja opittavissa.

Opinnäytetyössä keskitytään vain siihen, millaisilla kooditason ratkaisuilla voidaan vaikuttaa web-sovellusken tietoturvaan. Tämä tarkoittaa sitä, että tietoturvaongelmiin pyritään löytämään ratkaisu ruohonjuuritasolta. Tällainen lähestymistapa auttaa tietoturva-haavoittuvuuksien ymmärtämisessä. On mahdotonta luoda sellaista ohjeistusta, joka pätee aivan kaikissa mahdollisissa tilanteissa. Tämän takia ohjelmoijan on syytä tuntea web-sovelluksen haavoittuvuuksien perusteet ja periaatteet niiden hyödyntämisen takana. Tämä auttaa tunnistamaan sellaiset tilanteet, joissa tietoturva-aukkojen syntyminen on mahdollista.

Opinnäytetyö ei käsittele palvelimen ja tietokannan konfiguroinnin tuomia tietoturvahyötyjä. Ohjelmoijalla ei välttämättä ole mahdollisuutta tai osaamista vaikuttaa käytetyn palvelimen tai tietokannan asetuksiin.

Opinnäytetyön ulkopuolelle jäävät myös kaikki kolmansien osapuolien tarjoamat ohjelmistokehykset, julkaisujärjestelmät sekä kirjastot. Edellämainitut sovellukset tuovat yleensä mukanaan omat tietoturvaratkaisunsa, mutta niistä voi löytyä sellaisia haavoittuvuuksia, joiden paikkaaminen vaatii sovelluksen erityistä

tuntemista.

1.2 Tietoturvan merkitys

Internetissä vapaasti käytössä oleva web-sovellus on miljoonien ihmisten ulottuvilla. Suuri osa näistä ihmisistä ei ole tietoisia web-sovellusten tietoturva-asioista tai siitä, mitä riskejä haavoittuvan sovelluksen käytöllä voi olla. Osa näistä ihmisistä kykenee taitojensa puolesta tekemään web-sovelluksella sellaisia asioita, joita kyseisellä sovelluksella ei ole alun perin tarkoitus tehdä. Kyseessä voi olla puhdas haitanteko, omien taitoijen esittely tai jopa henkilökohtaisen hyödyn tavoittelu. Tietoturvan ylenkatsomisella web-kehityksessä voikin olla äärimmäisiä seurauksia. Jos web-sovelluksella on liiketoiminnan kannalta oleellista merkitystä, voi rahalliset menetykset olla suuria. Liiketoiminnalle kriittisen web-sovelluksen alasajaminen, korjaaminen ja uudelleenkäynnistys voi tapauksesta riippuen viedä paljon aikaa ja näinollen maksaa suuria rahamääriä (Matbouli & Gao 2012, 5). Asiakkaiden luottamuksen menettämisestä ei voi välttämättä edes rahassa mitata.

Sovellus tai sovelluksen haltija ei ole välttämättä ainut, joka kärsii huonolla tietoturvalla varustetusta sovelluksesta. Jotkin hyökkäyksistä kohdistuvat suoraan käyttäjään palvelun sijaan. Käyttäjältä voidaan varastaa arkaluontoista tietoa tai tehdä muunlaista haittaa. Jos sovelluksen käyttäjällä on esimerkiksi sama salasana useammassa web-palvelussa, voi sovelluksesta varastetulla salasanalla päästä sisään myös muihin palveluihin. Käyttäjän luottaessa web-palveluun täysin, voi hyökkääjä web-palvelun heikkouksia hyväksikäyttäen tehdä lähes mitä tahansa. Pahimmillaan hyökkääjä voi saada palvelun käyttäjän tietokoneen haltuunsa.

Heikon tietoturvan omaava web-palvelu voi toisinsanoen aiheuttaa suuria vahinkoja sekä palvelulle itselleen sekä sen käyttäjille. Tästä syystä tietoturvan tulisi olla jo suunnitteluvaiheeta lähtien yksi web-sovelluksen avainkysymyksistä.

Huomiotavan arvoinen seikka on se, että tässä opinnäytetyössä mainitut web-kehitykseen liittyvät tietoturvaongelmat eivät päde pelkästään edellämainittuihin webkehitystekniikoihin, vaan ne ovat yleisetettävissä myös muihin mainitsematta jääviin tekniikoihin.

2 TUTKIMUSMENETELMÄ

Opinnäytetyön ongelmaa lähdetään ratkomaan design sciense-tutkimusmenetelmää hyödyntäen. Design sciense on eritoten ongelmien ratkomiseen soveltuva menetelmä ja sen lopputulemana on aina jonkinlainen konsepti, metodi, malli tai esimerkki ongelman ratkaisemiseksi.

tietoteknisien ongelmien ratkaisu on jaettavissa kuuteen osaan, joista neljä ensimmäistä on kriittisiä opinnäytetön kannalta. Ongelmanratkaisun ensimmäisenä vaiheena on ongelman tiedostaminen, tunnistaminen ja sen merkityksellisyyden ymmärtäminen. Tässä tapauksessa ongelmana on web-kehityksen tietoturva ja se, millä menetelmillä voidaan rakentaa tietoturvallinen web-sovellus. Opinnäytetyö keskittyykin web-kehityksen tietoturvaongelmien tunnistamiseen ja niiden ehkäisemiseen. (Geerts 2011. 144.)

Toinen vaihe lähestyy ongelmaa teoriatasolla. Olemassa olevalle ongelmalle määritellään ihanteellinen lopputulos, johon voidaan päästä ratkaisemalla ongelma tilanteeseen nähden sopivimmalla tavalla. Toisinsanottuna pyritään selvittämään erilaisia vaihtoehtoja ongelman ratkaisemiseen ja sitä, voisiko kyseisten ratkaisujen tuoma lopputulos vastata haluttua tavoitetta. Tämä on mahdollista vain, kun ongelma ja siihen käytettävissä olevat tekniikat tunnetaan riittävän hyvin. (Geerts 2011. 144.)

Web-sovelluksen luominen on usein tasapainottelua tietoturvan ja käytettävyyden välillä. Web-sovelluksen tietoturvan kannalta ihanteellinen ratkaisu on sellainen, joka ei ole ylimitoitettu käsillä olevaan ongelmaan nähden, eikä ratkaisu vähennä sovelluksen käytettävyyttä merkittävästi. (Ben-Asher, Mayer, Möller, & Englert 2009. 1a)

Kolmannessa vaiheessa hyväksikäytetään aiemmassa vaiheessa tutkittuja teorioita ja metodeita, joiden avulla ongelma voidaan ratkaista. Valituista metodeista ja teorioista valitaan parhaat, joita aletaan soveltaa. (Geerts 2011. 144.)

Tietoturvallista ohjelmakoodia tehdessä tämä tarkoittaisi oikeanlaisen ohjelmointitavan ja käytettyjen funktioiden luoman kokonaisuuden suunnittelua ja toteutusta.

Neljännessä vaiheessa esitellään edellämainitun kokonaisuuden käyttöä ongelmanratkaisutilanteessa. Tarkoituksena on todistaa käytettyjen ratkaisujen toimivuutta. Opinnäytetyössä esitellään koodiesimerkein tietoturvallisia ratkaisuja erilaisiin ongelmiin. Näiden vaiheiden jälkeen viidennessä vaiheessa pohditaan sitä, kuinka hyvin ratkaisun lopputulos vastaa ongelman ratkaisulle asetettuja tavoitteita. (Geerts 2011. 144.)

3 OPINNÄYTETYÖSSÄ KÄYTETYT TEKNIIKAT

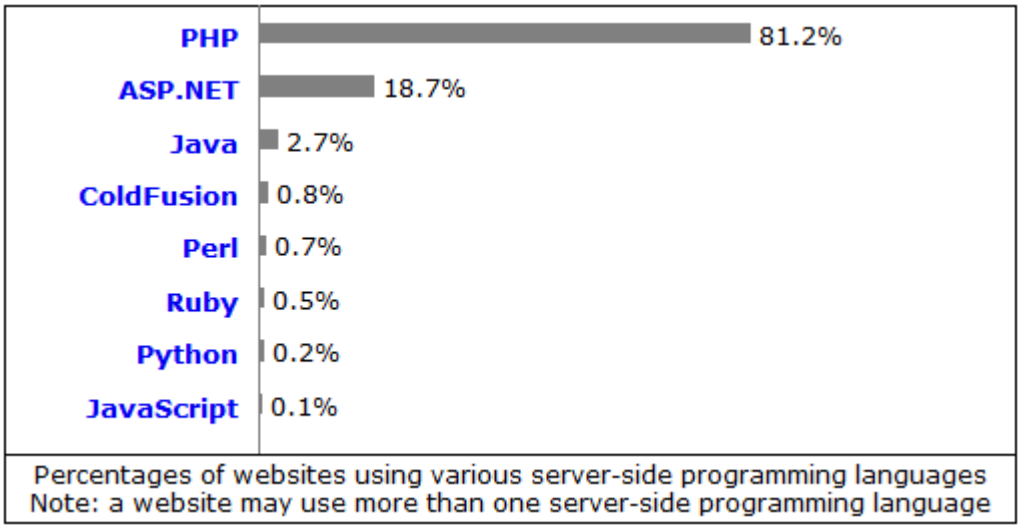
Kuten aikaisemmin on mainittu, opinnäytetyössä käytetyt tekniikat ovat PHP-ohjelmointikieli ja MySQL-tietokannat. Lisäksi on mainittava myös yleisesti web-kehityksessä käytetty merkkauskieli HTML ja CSS-tekniikka, joilla voidaan tehdä sovellukselle käyttöliittymä. Tässä kappaleessa kerrotaan hieman kyseisistä tekniikoista.

3.1 PHP-ohjelmointikieli

PHP (lyh. PHP: Hypertext Prosessor) on niin sanottu palvelinpuolen skriptikieli. Tämä tarkoittaa sitä, että PHP-koodia ei ajeta käyttäjän koneella. Web-sovelluksen käyttäjä lähettää web-selaimella pyynnön palvelimelle, joka toteuttaa halutun toiminnon. Toiminnon lopputulos palautetaan käyttäjän nähtäväksi halutulla tavalla. PHP:n avulla voidaan esimerkiksi prosessoida tietoa tai tallentaa tietoa tietokantoihin ja suorittaa muita web-sovellusten tarjoamia funktioita.

Interaktiivisia web-sivuja tai web-sovellusta ei voi yleensä tehdä pelkästään HTML:n kaltaisella merkkauskielellä, vaan käyttäjän syötteiden käsittelyyn ja erilaisten toiminnallisuuksien toteuttamiseen tarvitaan lähestulkoon aina jokin ohjelmointikieli.

PHP on suosituin web-kehityksessä käytettävistä ohjelmointikielistä. Syinä suureen suosioon voidaan pitää muun muassa käyttöönoton helppoutta, ohjelmointikielen suhteellisen matalaa oppimiskynnystä ja suuren yhteisön tarjoamaa tukea.



KUVIO 1: Käytetyimmät palvelinpuolen ohjelmointikielet. (PHP Group 2013)

Helppo käyttöönotto ja oppimiskynnys ovat osaltaan syitä, joiden takia useissa web-sovelluksista löytyy paljon tietoturvaongelmia. Web-sovelluksen kehittäjä ei välttämättä ole tietoinen kaikista sovelluksen tietoturvaa uhkaavista tekijöistä. PHP-ohjelmointikieli ei myöskään pakota tai ohjaa ohjelmoijaa käyttämään tietoturvan kannalta oleellisia ratkaisuja, vaan niiden käyttäminen jää ohjelmoijan vastuulle.

PHP-ohjelmointikieltä kehitetään jatkuvasti, joten siitä löytyvät rakenteelliset tietoturvaongelmat pyritään paikkaamaan heti niiden tultua ilmi. Tehdyt muutokset julkaistaan päivityksen muodossa. Yleisesti ottaen tietoturvan kannalta on suositeltavaa, että palvelin ja muut web-kehitykseen liittyvät ratkaisut ovat päivitettynä ja ajan tasalla.

3.2 MySQL-tietokantasovellus

MySQL on yksi maailman suosituimmista avoimen lähdekoodin relaatiotietokantaohjelmistoista. Relaatiotietokanta on tietokanta, joka koostuu tietokantatauluita. Näiden tietokantataulujen välille voidaan luoda suhteita. Relaatiotiotokantaohjelmiston tarkoituksena on toimia tiedon säilytyspaikkana. Tallennettavaa tietoa voi olla esimerkiksi web-ohjelmistoon kirjautumista varten tarvittavat käyttäjätiedot. MySQL-tietokantaohjelmisto sopii niin pieniin kuin suuriinkin sovelluksiin.

Tietokannan ja web-sovelluksen käyttöliittymä yhdistetään ohjelmoimalla niiden välille niin sanottu tietokantaliittymä. Tietokantaliittymän tarkoituksena on lähettää käyttäjän haluamat tietokantakyselyt tietokannanhallintajärjestelmälle. Liittymä myös mahdollisesti tulostaa tietokantakyselyjen tulokset käyttäjän nähtäville. Yksi esimerkki yleisesti suoritettavasta tietokantakyselystä on käyttäjätietojen kysely sovellukseen sisäänkirjautuessa.

PHP-ohjelmointikieli sopii hyvin yhteen MySQL-tietokantojen kanssa. PHP:sta löytyykin laajennusosat, jotka on tarkoitettu vain ja ainoastaan MySQL-tietokantojen käyttämiseen. PHP:n kanssa voi halutessaan käyttää myös muita tietokantasovelluksia.

Suurimmaksi osaksi MySQL-tietokantasovelluksen käyttöön liittyvät haavoittuvuudet perustuvat huolimattomasti tehtyihin tietokantaliittymiin. Huonosti tehty tietokantaliittymä mahdollistaa tietoturva-aukkojen hyödyntämisen. Esimerkiksi jotkin injektio-tyyppiset hyökkäykset hyödyntävät huolimattomasti tehtyjä tietokantaliittymiä.

3.3 HTML ja CSS

HTML (Hypertext Markup Language) ja CSS (Cascading Style Sheet) ovat web-sovelluksen ulkoasun tekemiseen ja muokkaamiseen käytettyjä tekniikoita. HTML on merkkauskieli, jolla web-sovelluksen ulkoasu rakennetaan. Web-selaimen tehtävä on lukea HTML-muotoinen tekstidokumentti ja piirtää tämän dokumentin sisältö käyttäjälle näkyvään muotoon.

CSS:ää käytetään määrittelemään se, miten HTML-dokumentti tulee piirtää. Sen avulla voidaan muunmuassa asemoida eri HTML-elementtejä, vaihtaa niiden väriä tai muotoa tai esitystapaa.

Pelkällä HTML:llä ja CSS:llä tehty staattinen web-sivu ei sisällä itsessään minkäänlaisia haavoittuvuuksia. Ainoat haavoittuvuudet olisivat tässä tapauksessa palvelimella olevissa ohjelmistoissa, jotka ovat rajattu tämän opinnäytetyön ulkopuolelle. Kun tämänkaltaiseen staattiseen web-sivuun aletaan lisäämään toiminnallisuuksia esimerkiksi PHP-ohjelmointikielellä, voidaan

huolimattomuuden myötävaikutuksella aiheuttaa tietoturvariskejä, joissa hyödynnetään jotain HTML:n tai CSS:n ominaisuuksia.

4 YLEISIÄ TIETOTURVAN KANNALTA HUOMIOITAVIA ASIOITA

Tässä kappaleessa esitellään sellaisia asioita, jotka eivät varsinaisesti ole haavoittuvuuksia, mutta niiden noudattamatta jättäminen vaikuttaa sovelluksen turvallisuuteen negatiivisesti.

4.1 Tietoturvaa yleisellä tasolla

Web-sovellusta ohjelmoidessaan, on ohjelmoijan syytä pitää mielessään muutamia perusajatuksia, jotka auttavat tietoturvallisen sovelluksen rakentamisessa. Ensinnäkin on mietittävä sitä, minkälaiseen käyttötarkoitukseen web-sovellus tulee. Erityisen arkaluontoista dataa, kuten luottokorttitietoja käsittelevä sovellus tarvitsee huomattavasti suurempaa panostusta tietoturvaan, kuin sovellus, jonka ainoa salassapidettävä data on käyttäjän kirjautumistieto.

Suuria käyttäjämääriä hallitseva web-palvelu saa todennäköisesti enemmän huomiota vahingontekijöiltä, kuin marginaalisia käyttäjämääriä hallitseva palvelu. On kuitenkin virheellistä ajatella, että pienelle piirille tarkoitettu web-palvelu olisi automaattisesti turvassa vahingonteoilta. Web-palvelusta riippumatta, tietoturvan ylenkatsominen voi kostautua ennemmin tai myöhemmin.

Tietoturva tulee ottaa huomioon jo web-sovelluksen suunnitteluvaiheessa. Ylimitoitettujen tietoturvaratkaisujen toteuttaminen voi viedä paljon aikaa sekä rahaa. Lisäksi ylimääräisten tietoturvaratkaisujen lisääminen sovellukseen voi vaikuttaa sen käytettävyyteen (Ben-Asher, Mayer, Möller, & Englert 2009. 1b). Tietoturvaa rakentaessa tuleekin punnita riskit sekä niiden vaikutus palveluun ja erityisesti palvelun käyttäjiin.

Web-sovellusta kehittäessä yksi turvallisuuden lähtökodista on se, että jokainen sovelluksen käyttäjä on mahdollinen vahingontekijä. Siispä jokainen käyttäjän antama syöte ja suorittama toiminto on tarkistettava ja tarvittaessa estettävä ennen suorittamista. Huolimattomasti tehty syötteiden tarkistus onkin yleinen tietoturvahaavoittuvuuksien lähde.

Käyttäjän syöte ei sellaisenaan voi tehdä mitään pahaa sovellukselle. Vasta, kun syötettä käytetään sovelluksessa väärällä tavalla, voidaan aiheuttaa vahinkoa. Tietokantaan päässyt injektiokoodi voi tyhjentää tietokannan datasta. Käyttäjälle tulosteen mukana näytettävällä haittakoodilla voi esimerkiksi varastaa toisen käyttäjän identiteetin.

Koska käyttäjiin ei voida koskaan täysin luottaa, ei heille tule näyttää sellaisia asioita, mitä heidän ei kuuluisi tai tarvitsisi nähdä. Esimerkiksi keskeneräisen HTML-koodin jättäminen web-sovellukseen kommentteina ei ole suotavaa. Käyttäjälle ei myöskään ole järkevää jättää näkyville sellaisia ominaisuuksia, johon hänen oikeutensa eivät riitä. Kärjistettynä esimerkkinä voidaan pitää ylläpitäjän paneelin tulostusta normaalille käyttäjälle, joka ei käyttäjätasostaan johtuen pystyisi ylläpitäjälle tarkoitettuja ominaisuuksia hyödyntämään. Tämänkaltainen virhe antaa vahingontekijöille vinkkejä siitä, miten kyseisiä toimintoja pystyisi väärinkäyttämään. Tästä syystä käyttäjälle näytettävät virheilmoituksetkin tulee olla itsemääriteltyjä ohjelman antamien virheilmoitusten sijaan.

Inhimilliset virheet ovat yleisiä syitä web-sovellusten haavoittuvuuksille (Cheng, Wang & Xu. 2006. 1). Yksi hyvä tapa vähentää inhimillisen erehdyksen aiheuttamaa tietoturvariskiä on selkeät nimeämiskäytännöt. Esimerkiksi nimeämällä muuttujat sen mukaan, onko ohjelmassa oleva data turvallista tulostaa käyttäjälle tai tietokantaan vai ei, voi estää ohjelmoijaa vahingolta, jossa validoimaton data siirretään sellaisenaan eteenpäin. (PHP Security Consortium, 2005.) Omien funktioiden tekeminen ei yleensä kannata siinä tapauksessa, jos ohjelmointikielestä löytyy halutun toiminnon tekevä funtio valmiina. Ohjelmointikielestä löytyvät funtiot ovat oletusarvoisesti useiden kehittäjien ja käyttäjien testaamia, jolloin voidaa olettaa niiden olevan käyttökelpoisia. Tiedot PHP:n valmiiden funktioiden toiminnasta ja esimerkit niiden käyttämisestä löytyy muun muassa PHP:n manuaalista.

4.2 Tietokantaliittymät

PHP:sta löytyy kolme ohjelmointirajapintaa, jolla tietokantayhteyden luominen voidaan toteuttaa. Ne ovat MySQL-lisäosa, MySQLi-lisäosa, sekä PHP Data Objects (PDO). MySQL on vanhin edellämainituista toteutustavoista, eivätkä PHP:n kehittäjät enää sitä käytettävän. Uusimmissa PHP versioissa tämä MySQL-lisäosa ei ole oletusarvoisesti käytettävissä ja sen aktiivinen kehittäminen on loppunut. Kyseinen rajapinta tullaankin poistamaan kokonaan lähitulevaisuudessa. (PHP Group 2013. a)

MySQLi ja PDO ovat uudempia tekniikoita tietokantaliittymien toteutukselle. Molemmissa on omat vahvuutensa ja heikkoutensa, mutta tärkeintä on se, että molemmilla voidaan tehdä tietoturvallinen tietokantaliittymä. valinta näiden kahden välillä voidaan tehdä web-sovelluksen erikoistarpeet tai sovelluskehittäjän omat mieltymykset huomioon ottaen.

PDO tukee MySQL-tietokannan lisäksi lukuisia muita tietokantasovelluksia, jonka ansiosta tietokantasovelluksen vaihtaminen käy tarpeen vaatiessa helposti. MySQLi:tä voi käyttää vain ja ainoastaa MySQL-tietokannan kanssa.

Oma valintani tietokantaliittymän tekemiseen on PDO. Syitä tähän on useita. Ensinnäkin PDO on edellämainituista tekniikoista uusin. PDO-tuki lisättiin PHP:n versiossa 5.1. Koska kyseessä on uusin tietokantaliittymien tekemiseen käytettävä tekniikka, voidaan sen elinikä tuen ja kehityksen puolesta olettaa kaikista pisimmäksi. Lisäksi PDO tukee ohjelmoinnin oliopohjaista lähestymistapaa joidenkin ominaisuuksiensa puolesta hieman paremmin kuin MySQLi.

Se, kuinka PDO:n avulla tehdään tietoturvallisia tietokantaliittymiä tullaan käymään läpi tutustuttaessa SQL-injektio hyökkäykseen.

4.3 Käyttäjän syötteiden validointi

Kuten aikaisemmin on mainittu, täytyy kaikkiin käyttäjän antamiin syötteisiin suhtautua sillä varauksella, että se voi sisältää haitallista materiaalia. Tämän takia kaikki käyttäjän syötteiden oikeellisuus varmistaa. Käyttäjän syötteiden

validoinnin ideana on tarkistaa, että käyttäjän antama syöte vastaa sovelluksen odottamaa syötettä. Syötteille heikosti tehty validointi mahdollistaa suuren määrän erilaisia tietoturva-ongelmia. (Hauzar & Kofron 2012, 1a)

Yleisesti ottaen syötteiden oikeellisuutta tutkiessa kannattaa hyödyntää niinsanottua whitelist-menetelmää. Tämän menetelmän perusajatuksena on se, että kaikki syötteessä olevat merkit ovat kiellettyjä, poislukien erikseen sallitut merkit. Whitelist-menetelmän vastakohtaa, eli blacklistia ei ole yleisesti suositeltua käytettäväksi datan validoinnissa. Tämä johtuu siitä, että on todennäköisempää, että ohjelmoija unohtaa lisätä blacklistiin jotain sinne kuuluvaa, kun whitelistissä se on jo lähtökohtaisesti kielletty. (Hauzar & Kofron 2012, 1b)

Käyttäjän syötteen validoinnin voi aloittaa miettimällä sitä, millaisia syötteitä missäkin vaiheessa käyttäjältä odottaa. Käyttäjän syöttämää sähköpostiosoitetta ei ole syytä hyväksyä muussa, kuin sähköpostiosoitteen virallisessa muodossa. Käyttäjätunnukselle, tai mille tahansa muulle syötteelle varatun kentän hyväksymät merkit voidaan edellämainitun whitelist-menetelmän avulla helposti rajoittaa tilanteen vaatimalla tavalla. Se, millä tavalla syöte rajataan riippuu täysin kontekstista, mutta rajaus on syytä tehdä aina niin tarkasti kuin mahdollista.

```
2    //käyttäjänimen tarkistus säännöllisellä lausekkeella
3    //sallitut merkit: isot ja pienet kirjaimet a-z,
4    //numerot sekä alaviiva. Pituus 4-15 merkkiä
5    if(preg_match('/^[a-zA-Z\d_]{4,15}$/i', $kayttajanimi))
6    {
7        //käyttäjänimi vastaa vaatimuksia
8    }   else    {
9        //käyttäjänimi ei vastaa vaatimuksia
10   }
11
12   //sähköpostiosoitteen oikeellisuuden varmistus
13   if(filter_var($email, FILTER_VALIDATE_EMAIL))
14   {
15       //salasana on oikeassa muodossa
16   }   else    {
17       //salasana ei ole oikeassa muodossa
18   }
19
20   //salasanan vahvuuden tarkistus säännöllisellä lausekkeella
21   //sisältää vähintään yhden numeron, kirjaimen ja erikoismerkin,
22   //pituus 8-30 merkkiä
23   if(preg_match('/^(?=.*\d)(?=.*[A-Za-z])[0-9A-Za-z!@#$%_.,]{8,30}$/', $salasana))
24   {
25       //salasana on riittävän vahva
26   }   else    {
27       //salasana ei vastaa vaatimuksia
28   }
```

KUVIO 2: Datan validointia.

Edellä näkyvässä kuvassa on esitelty kaksi eri tapaa hoitaa syötteiden validointi. Preg_match-funktion säännöllisellä lausekkeella voidaan rajata käyttäjän syöte niin tarkasti kuin vain on tarvetta. Esimerkissä käyttäjänimi on rajattu kirjaimiin, numeroihin sekä alaviivaan ja pituusrajoitukseksi on annettu 4-15 merkkiä. Tämänkaltainen syötteen rajaus estäisi hyökkääjän toimintaa erittäin paljon.

Myös käyttäjän antama salasana on myös validoitu säännöllisellä lausekkeella. Tässä tapauksessa säännöllisen lausekkeen etu on se, että sen avulla voidaan helposti pakottaa käyttäjä valitsemaan riittävän vahva salasana.

Sähköpostiosoitteen validoinnissa on käytettä filter_var-funktiota, jonka avulla voidaan validoida muitakin asioita. Tässä tapauksessa filter_var-funktio palauttaa totuusarvon sen mukaan, onko käyttäjän syöte oikeaa sähköpostiosoitetta vastaavassa muodossa. Jos halutaan varmistua sähköpostiosoitteen olemassaolosta, on sitä varten tehtävä erillinen toiminto.

Käyttäjän syötteiden validointi voidaan tehdä sekä selainpuolella että palvelunpuolella. Syötteitä validoidessa on muistettava se tärkeä seikka, ettei

selainpuolella tehty validointi ole koskaan riittävä tietoturvan kannalta. Pienellä vaivalla selainpuolen validoinnin voi saada poistettua käytöstä kokonaan. Jos tähän yhtälöön lisätään heikosti toteutettu tai jopa kokonaan puuttuva palvelinpuolen validointi, on sovellus täysin avoin hyökkäyksille. Selainpuolella suoritettu validointi kuitenkin kannattaa, koska se antaa välitöntä palautetta käyttäjälle ja parantaa sovelluksen käytettävyyttä.

4.4 Datan salaaminen

Riittävän suurella panostuksella mikä tahansa web-sovellus on murrettavissa. Tämän takia murtautumiseen täytyy varautua siitä huolimatta, että sovellukseen ei näennäisesti kohdistuisi mitään ulkoisia riskejä. Esimerkiksi tietokantaan tallennettavat salasanat on syytä salata jotakin salausmenetelmää käyttäen. Jos arkaluontoinen data koskaan päätyy vääriin käsiin, on sen hyödyntäminen huomattavasti vaikeampaa, kun kyseinen data ei ole tallennettuna selkokielisenä. Lisäksi on hyvä miettiä sitä, mitä dataa käyttäjästä ylipäätään tallennetaan tietokantaan.

Tietomurron sattuessa palvelun ylläpitäjä saa lisää aikaa reagoida jos arkaluontoinen data on salattuna. Ylläpitäjä voi ehtiä tiedottamaan käyttäjilleen sattuneesta tietovuodosta ja sen mahdollisista seurauksista. Näin voidaan välttyä suuremmilta vahingoilta.

Datan salaamiseksi PHP:sta löytyy joitakin valmiita funktioita, joiden avulla tietokantaan tallennettava data, tai mikä tahansa muu salausta vaativa data on salattavissa. Osa salausalgoritmeistä on jo vanhentuneita ja liian heikkoja arkaluontoisen datan salaamista varten. Tällaisia salausalgoritmejä ovat MD5, sha1 ja sha256. Kyseiset algoritmit kehitettiin nopeuden ehdolla murrettavuuden sijaan. Nykyaikaisella tietotekniikan tarjoamalla laskentateholla MD5 salatun salasanan murtaminen on varsin triviaali tehtävä.(PHP Group 2013. b)

Suositeltava salausalgoritmi tallennettavan datan salaamiseen on crypt-funktio ja Blowfish-menetelmä. Blowfish on käytettävissä PHP 5.4 versiosta lähtien ja se on vahvin PHP:sta löytyvä salausalgoritmi. (PHP Group 2013. b)

```
2    function salaus($salasana, $kierros = 10)
3    {
4    $merkkijono = 'abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ0123456789';
5    $salt = substr( str_shuffle($merkkijono), 0, 22 );
6    return crypt( $salasana, sprintf('$2y$%02d$',$kierros).$salt );
7    }
```

Kuvio 3: Datan salausta.

Tässä esimerkissä on tehty yksinkertainen salausfunktio, joka ottaa parametrinä käyttäjän lomakkeelta syöttämän salasanan ja palauttaa salasanan salattuna. Funktion alussa riveillä neljä ja viisi luodaan satunnainen merkkijono, jolla salausta tullaan vahvistamaan. Tarpeen vaatiessa tätä satunnaisen merkkijonon luontia voidaan vahvistaa, mutta esimerkin osoittama tapa ajaa asiansa.

Salaus-funktion viimeisellä rivillä käyttäjän antama salasana muutetaan selkokielisestä salatuksi ja saatu arvo palautetaan ohjelmalle. Salaus tapahtuu käyttämällä crypt-funktiota. Funktion ensimmäinen parametri on käyttäjän syöte. Toinen parametri sisältää tiedon käytetystä salausalgoritmistä, salauksen toistojen määrän ja niinsanotun saltin. Salt on mielellään jokaiselle salasanalle satunnaisesti luotu merkkijono, jonka tarkoituksena on tehdä salauksesta vahvempi (PHP Group 2013. b).

Kyseinen funktio siis luo annetusta salasanasta salatun version, joka voidaan tallentaa tietokantaan. Arkaluontoinen data ei siis enää ole selkokielisenä ja sen salaamiseen on käytetty vahvaksi todettua salausmenetelmää. Näin ollen voidaan todeta, että tietoturva tämän asian osalta on hoidossa.

4.5 Autentikaatio

Käyttäjän autentikaatio eli tunnistaminen on kriittinen osa web-sovellusta, jossa käyttäjällä on mahdollisuus henkilökohtaiseen käyttäjätiliin. Käyttäjän tunnistamisen tehtävänä on varmistua käyttäjän henkilöllisyydestä. Äärimmäisen kriittisissä sovelluksissa voidaan käyttäjän tunnistamisessa hyödyntää esimerkiksi vaihtuvia avainlukulistoja tai vaikka matkapuhelinta. Normaalissa sovelluksessa käyttäjän tunnistaminen hoidetaan sovelluksesta löytyvän kirjautumiselementin avustuksella.

Se, missä vaiheessa autentikaatio suoritetaan, riippuu sovelluksen luonteesta. Autentikaatio voi olla vapaaehtoinen ja sisäänkirjautumalla käyttäjä saa käyttöönsä joitakin lisäominaisuuksia. Toinen ääripää on se, että sovelluksen käyttö vaatii aina sisäänkirjautumisen. Kaiken sisällön piilottaminen sisäänkirjautumisen taakse on tietoturvan kannalta hyvä ratkaisu. Tämä pienentää niinsanottua hyökkäyspinta-alaa. Ilman autentikaatiota, mahdollisen hyökkääjän vaikutusalue rajoittuu kirjautumiselementtiin. Mitä vähemmän autentikoimaton käyttäjä näkee web-sovelluksesta, sitä vähemmän mahdollisia tietoturva-aukkoja sisältävistä elementeistä hän pystyy hyödyntämään. Tämänkaltainen ratkaisu voi olla käytettävyyden kannalta huono ratkaisu, eikä se välttämättä sovellu palveluihin, joiden sisällön on oltava näkyvissä myös autentikoimattomille käyttäjille.

Autentikaation yhteydessä annettavat virheilmoitukset eivät missään nimessä saa antaa yksilöityä palautetta siitä, että salasana on väärin, tai käyttäjänimi on väärin. Tällainen virheilmoitus paljastaa käytössä olevat käyttäjänimet. Oikeanlainen tapa ilmoittaa virheestä autentikaatiossa on sanoa, että joko käyttäjänimi tai salasana on väärin.

Lisäksi autentikaatio on syytä hoitaa salatun yhteyden avulla. Tämä on mahdollista, jos palvelimelle on asennettu kyseinen ominaisuus. Joissain tapauksissa haitallinen käyttäjä voi lukea selaimen palvelimelle lähettämät tiedot ja näinollen saada selville sovelluksen käyttäjänä kirjautumistiedot. Salatun yhteyden pakottaminen voidaan tehdä esimerkiksi seuraavanlaisella koodilla. Siinä koodi tarkastaa web-osoitteesta, onko käyttäjän yhteys on salattu. SSL-salauksen käyttö estää selaimen ja palvelimen välillä kulkevan datan luvatonta käyttöä

```
2    if(!isset($_SERVER['HTTPS']) || !$_SERVER['HTTPS'])
3    {
4        $url = 'https://'.$_SERVER['HTTP_HOST'].$_SERVER['REQUEST_URI'];
5        header('Location: ' . $url);
6        exit();
7    }
```

KUVIO 4: Käyttäjän ohjaaminen käyttämään SSL-salattua yhteyttä.

4.6 Istunnot (session)

Istunto on PHP:sta löytyvä ominaisuus, jonka avulla voidaan hoitaa tiedonvälitys selaimen ja palvelimen välillä. Istunnolla on aina elinkaari. Kun kyseessä on web-sovellus, voi elinkaari alkaa joko käyttäjän alkaessa käyttäämään sovellusta tai kirjautuessa sisään. Istunto aloitetaan käyttäämällä funktiota session_start. Elinkaaren tulisi pättyä viimeistään siihen, kun käyttäjä lopettaa sovelluksen käyttämisen. Istunnon poistaminenkin tulee tehdä manuaalisesti. Käyttäjän painaessa uloskirjautumispainiketta, voidaan istunto turvallisesti tuhota käyttämällä seuraavaa koodia (W3Schools 2013.)

```
// Initialize the session.
// If you are using session_name("something"), don't forget it now!
session_start();

// Unset all of the session variables.
$_SESSION = array();

// If it's desired to kill the session, also delete the session cookie.
// Note: This will destroy the session, and not just the session data!
if (ini_get("session.use_cookies")) {
    $params = session_get_cookie_params();
    setcookie(session_name(), '', time() - 42000,
        $params["path"], $params["domain"],
        $params["secure"], $params["httponly"]
    );
}

// Finally, destroy the session.
session_destroy();
```

kUVIO 5: istunnon tuhoaminen (PHP Group 2013. d)

Edellämainitussa esimerkkikoodissa, kaikki istunnon sisältämä tieto tyhjennetään, palvelimelle tallennettu istuntotiedosto poistetaan ja lopuksi selaimessa oleva istunto tuhotaan.

Istuntojen käyttämisestä voi tehdä turvallisempaa joillakin konsteilla. Esimerkiksi asettamalla istunnon elinajalle enimmäispituuden. Tämä tehdään siitä syystä, jos istuntoa ei pystytäkkään tyhjentämään ja poistamaan kunnolla. Tällöin elinaikansa ylittänyttä istuntoa ei voida enää käyttää haitallisiin tarkoituksiin.

4.7 Kulunvalvonta (access control)

Autentikaation antaessa käyttäjälle erilaisia oikeuksia, kulunvalvonnan tehtävä on rajoittaa käyttäjän toimintaa näiden oikeuksien puitteissa. Yksi eniten käytetyistä tavoista on käyttäjätasoihin perustuva kulunvalvonta. Tämänkaltaisessa kulunvalvonnassa käyttäjälle annetaan hänelle sopiva käyttäjätaso tai rooli. Käyttäjätasoille puolestaan annetaan oikeuksia eri toimintojen suorittamiselle. Toinen vaihtoehto olisi määrittää erikseen se, kuka käyttäjä saa kyseistä toimintoa käyttää. Tämä voidaan tehdä esimerkiksi asettamalla toiminnon sisään jonkinlainen lista sallituista käyttäjistä. Tällaista ratkaisua voisi käyttää jossain äärimmäisen kriittisessä toiminnossa. (Bai Zheng Y 2011. 1)

Käyttäjätasojen hallinta perustuu siihen, että autentikoidun käyttäjän käyttäjätasoa verrataan toiminnon suorittamisen vaatimaan käyttäjätasoon. Käyttäjätaso voidaan tallettaa esimerkiksi istuntoon. Käyttäjätason riittävyyden tarkistus toiminnon suorittamiseksi on tehtävä siitä huolimatta, ettei käyttäjä olisikaan edes tietoinen kyseisen toiminnon olemassaolosta.

Esimerkkinä kulunvalvonnasta voidaan esitellä keskustelualue, jossa normaalilla käyttäjällä on oikeus aloittaa uusia keskusteluja ja lähettää uusia viestejä. Ylläpitäjä-tason käyttäjällä voi näiden ominaisuuksien lisäksi olla mahdollisuus poistaa ja lukita kokonaisia keskusteluja. Kulunvalvonnan tehtävänä on estää normaalia käyttäjää suorittamasta ylläpitäjätason toimintoja.

Käyttäjien kulunhallinta voidaan suorittaa seuraavalla tavalla. Käyttäjän autentikoinnin onnistuessa, käyttäjän istuntoon tallennetaan tietokannasta noudettu käyttäjätaso. Tätä käyttäjätasoa verrataan toiminnon vaatimaan tasoon aina, kun ollaan suorittamassa toimintoa, joka vaatii jotain käyttäjätasoa.

```
2   $kysely->execute($kayttajatiedot);
3
4   if($kysely->rowCount() == 1)
5   {
6       //kirjautuminen onnistui,
7       // asetetaan tietokannasta tullut käyttäjän rooli istuntoon
8       $_SESSION['rooli'] = $kysely['rooli'];
9   }   else    {
10
11      echo 'vaara kayttajanimi tai salasana';
12  }
```

KUVIO 5: Käyttäjän käyttäjätason asettaminen.

Kuten aiemmin on mainittu, ei käyttäjälle kannata edes näyttää sellaisia toimintoja, joihin hänen oikeutensa eivät riitä. Käyttäjätasoa ei myöskään saa tarkistaa selaimen puolella, vaan tämä on tehtävä palvelimen puolella tietoturvan takaamiseksi. Kulunvalvonta toimii parhaiten, kun käyttäjän oikeus toiminnon suorittamiseen tarkastetaan aina, kun suoritus tehdään autentikaatiota vaativalla alueella. Hyvänä tapana on lisätä johdonmukaisesti kulunvalvonta aivan kaikkiin toimintoihin

```
2   if(isset($_SESSION['rooli']) && $_SESSION['rooli'] == $vaadittu_rooli )
3   {
4       //Jatka ohjelman suorittamista
5   }   else    {
6       //Estä toiminnon suorittaminen
7   }
```

KUVIO 6: Käyttäjätason tarkistaminen

Kyseisessä esimerkissä tarkistetaan, onko käyttäjälle annettu rooli autentikoinnin yhteydessä ja sitä, riittävätkö roolin oikeudet kyseisen toiminnon suorittamiseksi. Kulunvalvonnan toimimisen voi testata kokeilemalla kulunvalvonnan alaisuudessa olevaa toimintoa riittämättömillä oikeuksilla. Jos käyttäjä pyrkii esimerkiksi web-sivulle, jolle hänen oikeutensa eivät riitä, pitäsi sovelluksen antaa siitä jonkinlainen virheilmoitus tai ohjata käyttäjä takaisin ohjelman pariin.

5 YLEISIMMÄT HAAVOITTUVUUDET

Tässä kappaleessa käydään läpi yleisimmät tietoturvahaavoittuvuudet ja se, kuinka ne voidaan estää. Tietoturvahaavoittuvuus on sovelluksesta löytyvä heikkous, jota hyväksikäyttäen hyökkääjä pystyy suorittamaan haitallisia toimia.

Tietoturva haavoittuvuuksia voi etsiä valmiista web-sovelluksesta joko käymällä koodia manuaalisesti läpi, tai käyttämällä haavoittuvuuksien etsimiseen tarkoitettuja skannereita. Skannerin käyttö on nopea tapa varmistaa, onko sovellukseen jäänyt tietoturva-aukkoja

5.1 Haavoittuvuus: SQL-injektio

Yksi yleisimmistä tietoturva-aukkoa hyödyntävistä tekniikoista on SQL-injektio. Owasp:n mukaan vuonna 2013 SQL-injektio on kaikkein kriittisin riski web-sovellukselle.

SQL-injektion kohteena on aina palvelimella oleva tietokanta. Tässä hyökkäyksessä tarkoituksena syöttää web-sovelluksen käyttämälle tietokantasovellukselle haitallista koodia. Haitallisen koodin tarkoitus on muokata ohjelmoijan tekemää tietokantakyselyä halutulla tavalla. Tämän tarkoituksena voi olla esimerkiksi sisäänkirjautumisen ohittaminen tai tietokannan tuhoaminen. SQL-injektio on helposti toteutettavissa, mutta se on myös helposti estettävissä. (Shahariar & Zulkernine 2012.)

5.2 SQL-injektion estäminen

SQL-injektion estäminen on varsin yksinkertainen toimenpide. Tietoturva-aukon poistamiseksi täytyy vain pitää huoli siitä, ettei käyttäjän antaman syötteen mukana tuleva mahdollinen haitallinen koodi pääse tietokantakyselyyn asti. Tämä voidaan toteuttaa esimerkiksi käyttämällä sellaista funktiota, joka poistaa syötteestä SQL-injektioon käytettävät erikoismerkit.

Parametrisoitujen kyselyjen (parameterized query) käyttäminen on paras tapa estää SQL-injektio. Seuraavassa esimerkissä näytetään kuinka parametrisoitu

kysely tehdään. Epäselvyyksien estämiseksi on mainittava, että kyseisestä esimerkistä on jätetty käyttäjän syöte validoimatta, jotta esimerkin oleellinen anti tulisi selkeämmin ja tiiviimmin esitetyksi. Normaalitilanteessa epäluotettava data on syytä validoitava asianmukaisella tavalla ennen käyttämistä.

```
2    //tietokantayhteyteen tarvittavat tiedot $pdo-olioon
3    $pdo = new PDO( 'mysql:host=localhost;dbname=userdb;charset=utf8',
4                $dbkayttaja, $dbsalasana );
5
6    //mahdollistetaan virheilmiotusten tulostaminen
7    $pdo->setAttribute( PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION );
8    //valmistellun lausekkeen emulointi pois päältä
9    $pdo->setAttribute(PDO::ATTR_EMULATE_PREPARES, false);
10
11   //luodaan valmisteltu lause
12   $kysely = $pdo->prepare( 'SELECT * FROM user WHERE
13                             username = :kayttajanimi
14                             and password = :salasana' );
15
16   //Käyttäjän syötteet assosiatiiviseen taulukkoon
17   $parametrit = array(':kayttajanimi' => $_POST['nimi'],
18                       ':salasana' => $_POST['salasana']);
19
20   //suoritetaan tietokantakysely
21   //kyselyn parametrinä taulukko sisällään käyttäjän syötteet
22   $kysely->execute($parametrit);
23
24   if($kysely->rowCount() == 1)
25   echo 'Kirjautuminen onnistui';
26
27   else
28   echo 'Vaara kayttajanimi tai salasana';
```

KUVIO 7: Koodiesimerkki SQL-injektion ehkäisevästä tietokantaliitymästä

SQL-injektion estämisen kannalta oleellisimpia kohtia ovat koodin rivit 12-14, sekä 17-18 ja rivi 22. Riveillä 12-14 näkyy aivan tavallista SQL-syntaksia. Ainoana erona normaaliin SQL-syntaksiin on kyselyn parametrien kohdalla olevat nimety paikkamerkit. Parametrisoidun kyselyn turvallisuus perustuu siihen, ettei käyttäjän syötettä käytetä kyselyä valmistellessa. Tämä tarkoittaa sitä, ettei injektiokoodia suoriteta tietokantahaun mukana missään vaiheessa. Käyttäjän syöte lisätään kyselyyn siinä vaiheessa, kun tietokantasovellus tietää miltä kysely tulee näyttämään. Näinollen vältytään haitallisen koodin ajamiselta tietokannassa. PDO käsittelee syötettä täysin normaalina tekstisyötteenä ajettavan koodin sijasta.

Riveillä 17-18 käyttäjän antamat arvot asetetaan assosiatiiviseen taulukkoon, joka annetaan parametrinä kyselylle rivillä 22. Muita tietoturvan kannalta oleellisia kohtia ovat rivit kolme ja yhdeksän. Rivillä kolme oleva parametri charset=”utf8” kertoo tietokantasovellukselle sen, ettei syötteen mukana tule hyväksyä merkkejä UTF-8 merkistökoodauksen ulkopuolelta. Rivillä yhdeksän, parametrisoidun kyselyn emulointi otetaan pois päältä. Emuloinnin päälläolo altistaa sovelluksen sql-injektiolle joissain erikoistapauksissa, joten emulointi on syytä asettaa pois käytöstä.

5.3 Haavoittuvuus: Cross-Site Scripting (XSS)

Cross-Site-Scripting on SQL-injektion tapaan injektio-tyyppinen hyökkäys, joka voidaan toteuttaa hyödyntäen kaikkia niitä web-sovelluksen elementtejä, joissa http-pyynnön mukana lähetettävä syöte tulostetaan web-palveluun. Kyseessä on hyvin yleinen haavoittuvuus ja sitä on löytynyt muun muassa Facebookin, Twitterin ja jopa tietoturvayhtiö F-Securen nettisivuilta. Tämä haavoittuvuus ei rajoitu vain PHP-kieleen, vaan sitä esiintyy monilla muillakin ohjelmointikielillä toteutetuissa web-sovelluksissa.

XSS-hyökkäys ei varsinaisesti kohdistu palveluun itseensä, vaan sen käyttäjiin. Palvelun käyttäjälle voidaan XSS-haavoittuvuutta hyväksikäyttäen syöttää selainpuolella suoritettavia haitallisia ohjelmaskriptejä. Nämä ohjelmaskriptit ajetaan käyttäjän selaimessa näkymättömissä, joten käyttäjä ei välttämättä edes tiedä mitä on tapahtunut. Näiden skriptien avulla voidaan päästä käsiksi palvelun käyttäjän istunto- tai evästetietoihin, joita hyväksikäyttäen voidaan muun muassa käyttää palvelua toisen käyttäjän oikeuksilla ilman sisäänkirjautumista. (Iha, G & Doi, H 2009.)

Xss-hyökkäys on jaettavissa kahteen pääkategoriaan: pysyvään ja väliaikaiseen hyökkäykseen. Pysyvä hyökkäys tarkoittaa sitä, että haitallinen koodin on saatu tallennettua esimerkiksi web-sovelluksen käyttämään tietokantaan. Pysyvä hyökkäys on sikäli erityisen vaarallinen, koska se kohdistuu kaikkiin niihin käyttäjiin, jotka tietokantakyselyn avulla tietämättään lataavat haitallista koodia selaimeensa.

Väliaikainen hyökkäys puolestaan kohdistuu yksittäiseen käyttäjään kerrallaan. Tässä hyökkäyksessä palvelun käyttäjä yritetään saada ajamaan esimerkiksi haittakoodilla varustettu linkki, joka johtaa haavoittuvuuden omaavaan web-palveluun. Tällainen linkki voidaan jakaa vaikka sähköpostin välityksellä tai jotain muuta keinoa käyttämällä.

5.4 XSS:n estäminen

XSS:n estäminen ei ole välttämättä helppoa. Tämän haavoittuvuuden kohdalla on huomioitava se seikka, että hyökkäys voi tapahtua minkä tahansa syötteen avustuksella. Kyseisen haavoittuvuuden estäminen riippuu täysin kontekstista. Käyttämällä oikeita keinoja oikeissa paikoissa XSS-haavoittuvuus on kuitenkin täysin estettävissä. Käyttäjän syötteen validointi sekä tulostuksen enkoodaus ovat ne lähtökohdat, joilla XSS-haavoittuvuus estetään.

Syöttäessä dataa html-elementtien sisään voidaan epäluotettava data tulostaa enkoodaamalla se käyttäen php:n omaa htmlspecialchars-funktiota. Tämä funktio muuntaa haitalliseen koodiin tarvittavat merkit sellaiseen muotoon, ettei selain suorita haitallista koodia, vaan tulostaa sen sellaisenaan.

```
2    <?php $tuloste = htmlspecialchars( $data );  ?>
3    <div id="content"><?php echo $tuloste ?> </div>
```

KUVIO 8: Tuloste HTML tagien sisään.

Syöttäessä epäluotettavaa dataa web-osoitteeseen, on syytä käyttää urlencode-funktiota ja htmlspecialchars-funktiota syötettävän datan varmistamiseen. Huomionarvoinen seikka on kuitenki se, että tämän tavan käyttäminen kokonaisen web-osoitteen tulostamiseen saattaa aiheuttaa toimimattomia linkkejä. Kokonaisen linkin tulostamiseen on tällöin syytä käyttää edellämainittua htmlentities- tai htmlspecialchats-funktiota. Seuraavaksi tulevassa esimerkissä on web-osoitteen alkuosa valmiiksi kirjoitettuna.

```
2    <?php $tuloste = htmlspecialchars( urlencode($data) );  ?>
3    <a href="http://esimerkki.fi/<?php echo $tuloste ?> ">linkin teksti</a>
```

KUVIO 9: Tuloste web-osoitteen sisään.

Jos käyttäjän on pystyttävä antamaan syötteitä css-määrittelyyn, se kannattaa tehdä täysin numeerisena suorituksena validoimalla data kunnolla ja asettamalla se suoraan halutun määrittelyn eteen.

```
2  <div style="width:<?php if(is_numeric($tuloste)){echo $tuloste;} ?>;">
```

KUVIO 10: Tuloste CSS määrittelyyn.

Jos web-sovelluksen luonteeseen kuuluu se, että käyttäjän on voitava käyttää syötteenään html-elementtejä, on tällöin suositeltavaa käyttää jotakin kolmannen osapuolen kirjastoa. Esimerkiksi HTMLpurifier kirjasto on rakennettu juuri tällaista tilannetta varten. Kolmannen osapuolen kirjaston käyttäminen tulostettavan syötteen varmistamiseksi on muutenkin suositeltavaa. Kyseiset kirjastot on yleensä huolella tehty, testattu ja dokumentoitu. XSS:ssä on lukuisia erilaisia keinoja validoinnin ja puhdistamisen ohittamiseksi. Niiden kaikkien huomioon ottaminen ja torjuminen omatekemän kirjaston avulla on äärimmäisen riskialtista.

5.5 Haavoittuvuus: Cross-Site Request Forgery (CSRF)

CSRF on web-sovelluksen käyttäjän web-selaimen kautta suoritettava hyökkäys. Hyökkäyksen tarkoituksena on saada web-palveluun kirjautuneen käyttäjän web-selain suorittamaan hyökkääjän haluama toiminto. Hyökkäys toteutetaan johdattelemalla palveluun sisäänkirjautunut käyttäjä hyökkääjän saastuttaman materiaalin luo. Tällainen materiaali voi olla esimerkiksi sähköposti, johon on upotettu haitallista koodia sisältävä html-elementti. Hyökkäyksen onnistuminen vaatii sen, että käyttäjä on kirjautuneena sisään ja omaa sovelluksen vaatimat oikeudet halutun toiminnon suorittamiseksi. (Boyan, Zavarsky, Ruhl, & Lindskog 2011. 1)

CSRF perustuu siihen faktaan, ettei web-sovellus pysty tunnistamaan sitä, tekeekö pyynnön toiminnon suorittamiseksi käyttäjä vai hyökkääjä. Web-sovellus pystyy tunnistamaan ainoastaan sen, onko pyyntö toiminnon suorittamiseen tullut sisäänkirjautuneen käyttäjän web-selaimesta vai ei. (Boyan, Zavarsky, Ruhl, & Lindskog 2011. 1)

5.6 CSRF:n estäminen

Kyseisen tietoturvaongelman ehkäisemiseksi on useita erilaisia vaihtoehtoja. Yksi vahvimmista varmistuskeinoista on käyttää niinsanottua CSRF tokenia, joka on satunnaisuutta hyödyntäen luotu pitkä merkkijono. Tämä merkkijono tallennetaan palvelinpäähän, sekä lähetetään käyttäjälle tulostettavan näkymän mukana html-koodiin sijoitettuna. Kun käyttäjä lähettää uuden pyynnön, tarkistetaan selaimen takaisinpäin lähettämän merkkijonon ja palvelimelle tallennetun merkkijonon yhteneväisyys. Jos nämä kaksi merkkijonoa eivät täsmää, tai kyseistä merkkijonoa ei ole käyttäjän pyynnön mukana, ei pyyntö ole todennäköisesti tullut käyttäjältä itseltään. Tämän jälkeen uusi näkymä lähetetään käyttäjälle, jonka mukana lähtee uusi satunnaisesti luotu merkkijono. CSRF-tokenin käyttö ei vaadi sovelluksen käyttäjältä ylimääräisiä toimenpiteitä, koska sen käyttö suoritetaan ohjelmalogiikan sisässä käyttäjän näkymättömissä. (Boyan, Zavarsky, Ruhl, & Lindskog 2011. 2)

CSRF-suojaksen tokenin avulla voi hoitaa kahdella eri tavalla. Joko pysyvällä tokenilla, tai joka pyynnön yhteydessä vaihdettavalla tokenilla. Jälleen kerran ylimääräinen tietoturva haittaa sovelluksen käytettävyyttä. Jokaisen pyynnön jälkeen vaihtuvan tokenin takia sovellusta käytettäessä ei voi käyttää selaimen ”takaisin” toimintoa, sillä selaimessa oleva vanha CSRF-token ei vastaisi palvelimella olevaa uutta tokenia. Vain sisäänkirjautumisen yhteydessä annettava CSRF-token ei vaikuta sovelluksen käytettävyyteen. Jos web-palvelussa on XSS-haavoittuvuus, pystyy hyökkääjä kuitenkin selvittämään CSRF-tokenin ja käyttämään CSRF-hyökkästä.

```
2   //Aloitetaan istunto
3    session_start();
4
5    //Jos lomake on lähetetty
6    if(isset( $_POST['submit']) )
7    {   //Vastaako palvelimelle tallennettu tunnus lomakkeelta tulevaa tunnusta?
8        if( isset($_SESSION['token'])&& $_SESSION['token'] == $_POST['token'] )
9        {
10         //jatka ohjelman suorittamista
11       }   else   {
12           //tarkastus epäonnistui
13       }
14   }
15
16   //Luodaan riittävän pitkä uniikki tunnus, asetetaan tunnus istuntoon
17   $token = md5( uniqid() );
18   $_SESSION['token'] = $token;
19
20  ?>
21  <html>
22      <head></head>
23      <body>
24          <form method="POST" action="<?php $_SERVER['PHP_SELF'] ?>" >
25              <input type="text" name="kayttajanimi"></input>
26              <input type="password" name="salasana"></input>
27              <input type="hidden" name='token' value="<?php echo $token ?>">
28              <input type="submit" name='submit'>
29          </form>
30      </body>
31  </html>
```

KUVIO 11: CSRF-suojausken käyttö lomakkeessa.

Ylläolevassa koodiesimerkissä näytetään, kuinka CSRF-tokenia voidaan hyödyntää haavoittuvuuden paikkaamisessa. Sivun latauksen yhteydessä, rivillä 17-18 luodaan uniikki tunnus. Tämä tunnus tulostetaan sivun html-koodiin esimerkiksi hidden-tyyppisen lomakekentän arvoksi, kuten rivillä 27. Toinen vaihtoehto voi olla tokenin asettaminen HTML:n metadatan sekaan. Kun käyttäjä on lähettäny lomakkeen painamalla lähetä-nappia, tarkastetaan rivillä 6-8 se, onko käyttäjä lähettänyt tiedot lomakkeen kautta ja täsmääkö palvelimelle tallennettu CSRF-token lomakkeelta lähetettyyn arvoon. Jos arvot täsmäävät, tarkoittaa se sitä, että pyyntö toiminnon suorittamiselle on tullut käyttäjän toimesta. Tällöin voidaan ohjelman suorittamista jatkaa normaalisti. Jos arvot eivät täsmää, tulee käyttäjän pyyntö evätä virheilmoituksen kanssa.

CSRF:n tapauksessa käytettävyydestä voidaan joutua hieman tinkimään tietoturvan vuoksi. Jos web-sovellukseen on lisätty kriittisiä toimintoja, kuten käyttäjätilin poistaminen, on toimintoja suorittaessa hyvä tehdä jonkinlainen erikoisvarmistus. Käyttäjätiliä poistaessa voidaan käyttäjää pyytää varmistamaan

henkilöllisyytensä esimerkiksi salasanaa pyytämällä. Tämänkaltainen varmistus ei suojaa pelkästään CSRF-hyökkäystä vastaan, vaan toimii muissakin tilanteissa, jossa haitallinen käyttäjä saa toisen käyttäjän tilin haltuunsa tietämättä kuitenkaan salasanaa.

6 YHTEENVETO

Tietoturvallisen web-sovelluksen tekemiseen ei ole olemassa mitään sellaista yleisohjetta, jonka avulla voidaan automaattisesti tehdä murtumaton sovellus. Tietoturva on ennemminkin kuin palapeli, jossa jokaista lisättyä ominaisuutta tai palasta kohden on suoritettava tietyt varotoimenpiteet. Hyvä suunnittelu ja oikein mitoitetut ja käytetyt tietoturvaratkaisut pitävät sovelluksen turvallisena, eikä käytettävyyskään pääse liiaksi kärsimään.

Käyttäjän syötteet ja niiden huolimaton käsittely on yksi suurimmista tietoturvaongelmien aiheuttajista. Syötteiden tarkka validointi ja filtteröinti, sekä tulosteiden käsittely tilanteen vaatimalla tavalla lisää sovelluksen turvallisuutta huomattavasti.

Ennen datan tietokantaan syöttämistä täytyy pitää huoli siitä, ettei tietokannan heikkouksia hyväksikäytetä. Riippumatta käytetystä ohjelmointirajapinnasta, tarvittavat toimenpiteet tietokannan heikkouksien hyödyntämisen estämiseksi täytyy tehdä. Tämä tapahtuu joko syötteen manuaalisella puhdistuksella haitallisista merkeistä tai käyttämällä parametrisoituja kyselyitä.

Tulostaessa mitä tahansa tulostetta käyttäjälle tulee siitä puhdistaa mahdolliset haitalliset elementit pois. Puhdistuksessa käytettävät tavat ja funktiot riippuvat täysin konteksista, joten tulosteiden kanssa tulee olla erityisen varovainen. Koska syötteiden kokonaisvaltaista puhdistamista on vaikea suorittaa johtuen suuresta hyökkäyspinta-alasta ja lukuisista erilaisista kiertoteistä, on suositeltavaa käyttää jotain kolmannen osapuolen sovellusta.

Koska täysin murtovarman sovelluksen tekeminen on hyvin haastavaa tai jopa mahdotonta, on murtautimiseen varauduttava kaikin mahdollisin keinoin. Tämä tarkoittaa sitä, että kaikki käyttäjästä sovellukseen tallennettava arkaluontoinen tieto suojataan riittävän tehokkaasti ennen tallentamista. Tätä varten voidaan käyttää jotain vahvaksi luokiteltua salausalgoritmia, jonka salaaman datan voi tallettaa tietokantaan huoletta.

Käyttäjän suorittaessa erilaisia toimintoja web-sovelluksessa pitää varmistua siitä, että toiminnon suorittaja on todellakin kyseinen käyttäjä, eikä pyyntö tule

käyttäjän tahdon vastaisesti kolmannelta osapuolelta. Tämä voidaan suorittaa lisäämällä web-palveluun ylimääräisiä varmennuksia palvelimen ja selaimen välille. Nämä varmennukset voivat olla joko käyttäjältä piilossa tai käyttäjän interaktiota vaativia toimintoja.

Lisäksi pitää varmistua siitä, että kyseisellä käyttäjällä on riittävät oikeudet toiminnon suorittamiseen. Jokaiseen sivuun ja mahdollisesti myös toimintoon on hyvä lisätä käyttäjältä vaadittava käyttäjätaso. Näin tekemällä voidaan päätellä saako käyttäjä suorittaa pyydetyn toiminnon.

Sovelluksen käyttäjälle ei tule näyttää mitään sellaista, mitä hänen ei tarvitse tai kuulu nähdä. Tähän lukeutuu muun muassa käyttäjän käyttöoikeuksien ulkopuolella olevat toiminnot, ohjelman tulostamat virheilmoitukset ja esimerkiksi kommentteihin tallennettu keskeneräinen koodi. Tällä tavoin voidaan vaikeuttaa haitantekijän mahdollisuuksia hyödyntää olemassaolevia tietoturva-aukkoja.

LÄHTEET

Bai, Q-H & Zheng Y. 2013. Study on the Access Control Model in Information Security [viitattu 26.10.2013]. Saatavilla: http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=6037079

Ben-Asher, N. Mayer, J. Möller, S. & Englert, R. 2009. [viitattu 25.10.2013]. Saatavissa: http://ieeexplore.ieee.org/stamp/stamp.jsp?arnumber=5066581

Boyan, C., Zavarsky P., Ruhl, R. & Lindskog, D. 2011. A Study of the Effectiveness of CSRF Guard. [viitattu 28.10.2013]. Saatavilla: http://ieeexplore.ieee.org/stamp/stamp.jsp?arnumber=6113294

Cheng, X-Y. Wang, Y-M. & Xu, L-Z. 2006. RISK ASSESSMENT OF HUMAN ERROR IN INFORMATION SECURITY. [viitattu 28.10.2013]. Saatavissa: http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=4028690

Geerts, G. 2011. a Design Science Research Methodology and its Application to Accounting Information Systems Research. [viitattu 20.10.2013]. Saatavissa: http://ac.els-cdn.com/S1467089511000200/1-s2.0-S1467089511000200-main.pdf?_tid=ae93a524-408d-11e3-a35e-00000aab0f27&acdnat=1383046481_0bce88abb5cc01d21b44417070dd5f91

Hauzar, D & Kofron, J. 2012. On Security Analysis of PHP Web Application. [viitattu 23.10.2013]. Saatavilla: http://www.php.net/manual/en/intro.mysql.php

Iha, G & Doi, H. 2009. An Implementation of the Binding Mechanism in the Web Browser for Preventing XSS Attacks: Introducing the Bind-Value Headers. [viitattu 23.10.2013]. Saatavilla: http://ieeexplore.ieee.org.aineistot.phkk.fi/xpls/icp.jsp?arnumber=5066595

Matbouli, H. & Gao, Q. 2012. An Overview on Web Security Threats and Impact to E-Commerce Success [viitattu 24.10.2013]. Saatavissa: http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=6216645

PHP Security consortium. 2005. PHP Security Guide: Overview. [viitattu 28.10.2013]. Saatavilla: http://phpsec.org/projects/guide/1.html

PHP Group. 2013. Introduction. [viitattu 28.10.2013]. Saatavilla: http://www.php.net/manual/en/intro.mysql.php

PHP Group. 2013. Safe Password Hashing. [viitattu 26.10.2013]. Saatavilla: http://us2.php.net/manual/en/faq.passwords.php

PHP Group 2013. session_destroy. [viitattu 20.10.2013]. Saatavilla: www.php.net/manual/en/function.session-destroy.php

Shahariar, H. & Zulkernine, M. 2012. Information-Theoretic Detection of SQL Injection Attacks. [viitattu 18.10.2013]. Saatavilla: http://ieeexplore.ieee.org.aineistot.phkk.fi/xpl/articleDetails.jsp?tp=&arnumber=6375635&queryText%3Dphp+user+authentication

W3Schools. 2013. PHP Sessions. [viitattu 27.10.2013]. Saatavilla: http://www.w3schools.com/php/php_sessions.asp

W3Techs. 2013. Usage of server-side programming languages for websites. [viitattu 27.10.2013]. Saatavilla: http://w3techs.com/technologies/overview/programming_language/all