Haaga-Helia
ammattikorkeakoulu Oy

Web-sovellusten haavoittuvuudet ja penetraatiotestaus

Kristian Harju

Opinnäytetyö
Tietojenkäsittelyn koulutusohjelma
2016

Haaga-Helia
ammattikorkeakoulu Oy

Tiivistelmä

Tekijä
Kristian Harju

Opinnäytetyön otsikko
Web-sovellusten haavoittuvuudet ja penetraatiotestaus

Opinnäytetyön otsikko englanniksi	**Sivu- ja liitesivumäärä**
Web application vulnerabilities and penetration testing	38 + 0

Ohjaaja
Sirpa Marttila

Penetraatiotestaus ja tietoturva ovat olennainen osa ohjelmistokehitystä. Luottamuksellisen tiedon ja identiteettivarkauksien määrä on kasvava ongelma tällä hetkellä. Näistä syistä web-sovellusten haavoittuvuudet ja tietoturvallisuus ovat ajankohtaisia ja kiinnostavia aiheita monille ohjelmoijille.

Tämän tutkimuksen tarkoituksena oli tarjota tietoa ohjelmoijille, miten web-sovellukset ovat alttiina hyökkäyksille sekä antaa muutamia perusohjeita, miten estää näitä uhkia. Toisena tavoitteena oli selvittää, mitä on eettinen hakkerointi ja tutkia, miten penetraatiotestaus suoritetaan asianmukaisella tavalla.

Tässä tutkimuksessa keskityttiin ohjelmistokoodiin. Palvelin- ja asiakasohjelmiston konfiguraatiot jätettiin tutkimuksen ulkopuolelle. OWASP Top Ten -listausta käytettiin lähteenä web-sovellusten tämän hetken kaikkein yleisimmille haavoittuvuuksille.

Web-sovellusten tietoturvallisuuden perusteorian ja joidenkin monimutkaisempien yksityiskohtien selittämiseen käytettiin laadukkaita kirjallisia lähteitä internetistä löytyvien lähteiden ohella. Teorian lisäksi tämä opinnäytetyö tarjoaa toimivia esimerkkejä hyökkäyksistä ja koodiesimerkkejä, kuinka ohjelmoijat voivat estää näiden haavoittuvuuksien hyödyntämistä.

Tutkimuksen tärkein kohderyhmä on web-ohjelmoijat ja erityisesti ne, jotka käyttävät ohjelmointikielenä Javaa.

Asiasanat
Java, tietoturva, haavoittuvuus, hakkerointi, verkko-ohjelmointi

Haaga-Helia
University of Applied Sciences Ltd.

Abstract

Author
Kristian Harju

Thesis title
Web application vulnerabilities and penetration testing

Advisor	**Number of pages and appendix pages**
Sirpa Marttila	38 + 0

Penetration testing and security are an essential part of software development. Stealing confidential information and identity thefts are a growing problem at the moment. For these reasons the vulnerabilities and security of web applications are timely and interesting topics for many programmers.

The purpose of this study was to offer information for programmers on how web applications are exposed to attacks, and provide some basic guidelines for how to prevent these threats. The second objective was to find out what ethnical hacking is and examine how penetration testing is carried out in an appropriate manner.

In this study the focus was at the programming level. Server and client configurations were excluded from the study. OWASP top ten listing was used as the source of the most common vulnerabilities found at the moment in the web applications.

To explain the basic theory about the web application security and some of the more complex details, high quality written sources were used in addition to the resources available on the Internet. In addition to the theory this study provides working examples of attacks and code examples of how programmers can prevent exploitation of these vulnerabilities.

The main target group for this study is web programmers and especially those who use Java as a programming language.

Keywords
Java, security, vulnerability, hacking, web programming

Sisällys

1 Johdanto ja toteutus

Vastuu web-sovelluksen lähdekoodin tietoturvallisuudesta kuuluu pohjimmiltaan sovelluksen kehittäjälle. Sovelluksen kehittäjä ei voi kuitenkaan luoda tietoturvallista lähdekoodia, mikäli asianmukaista tietoturvatestausta ei suoriteta. Penetraatiotestauksen tarkoitus on löytää web-sovelluksesta mahdolliset haavoittuvuudet, jotka saattavat heikentää sovelluksen tietoturvaa. (OWASP 2016s.)

Tämän opinnäytetyönä tehtävän tutkimuksen aihealueena ovat web-sovellusten haavoittuvuudet ja tietoturvatestaus. Kohdeilmiönä käsitellään nykyisin tapahtuvaa web-sovellusten hakkerointia ja sitä vastaan suojautumista ohjelmistokehityksen keinoin. Web-sovellusten haavoittuvuudet ja tietoturva on ajankohtainen sekä monia ohjelmoijia kiinnostava aihe, josta ei käytännössä löydy suomenkielistä kirjallisuutta. Tietoturvatestaus kuuluu olennaisena osana ohjelmistokehitykseen. Tämän tutkimuksen taustalla onkin halu selvittää, miten web-sovellusten tietoturvatestaus käytännössä tapahtuu.

Tässä työssä selvitetään:

- Miten web-sovellusten penetraatiotestaus suoritetaan ohjelmoijan näkökulmasta?
- Millä tavalla OWASP Top Ten -listauksen ohjelmistokoodiin liittyviä haavoittuvuuksia hyödynnetään?
- Millä tavalla ohjelmistokoodia muuttamalla voidaan estää tai hankaloittaa haavoittuvuuksien hyväksikäyttöä?

Opinnäytetyön tavoitteina on selvittää, miten tietoturvatestaus suoritetaan, kuvata mahdollisimman selkeästi ja käytännön esimerkein, kuinka hakkerointi tapahtuu sekä selvittää, kuinka käytännössä voidaan estää tai vaikeuttaa hakkerointia. Opinnäytetyö keskittyy ohjelmistokoodin haavoittuvuuksien tutkimiseen ja ratkaisuihin. Palvelinten ja web-selaimien haavoittuvuudet rajataan tutkimuksen ulkopuolelle.

Tutkimuksen kohdeyleisönä ovat web-sovellusten ohjelmoijat. Esimerkit on tehty käyttäen Java-ohjelmointikieltä sekä Javaan liittyviä tekniikoita, mutta muitakin kieliä käyttävät ohjelmoijat voivat hyödyntää teoriaosuutta. Tutkimuksessa keskitytään viiteen OWASP Top 10 -listalta valittuun haavoittuvuuteen. Muut listan haavoittuvuudet esitellään lyhyesti.

Tutkimuksen luonne on empiirinen eli kokemusperäinen, ja se on menetelmäsuuntaukseltaan kvalitatiivinen eli laadullinen tutkimus. Tutkimusstrategiana sovelletaan tapaustutkimusta sekä kokeellista tutkimusta. Tutkimusmenetelminä käytetään kontrolloitua koetta.

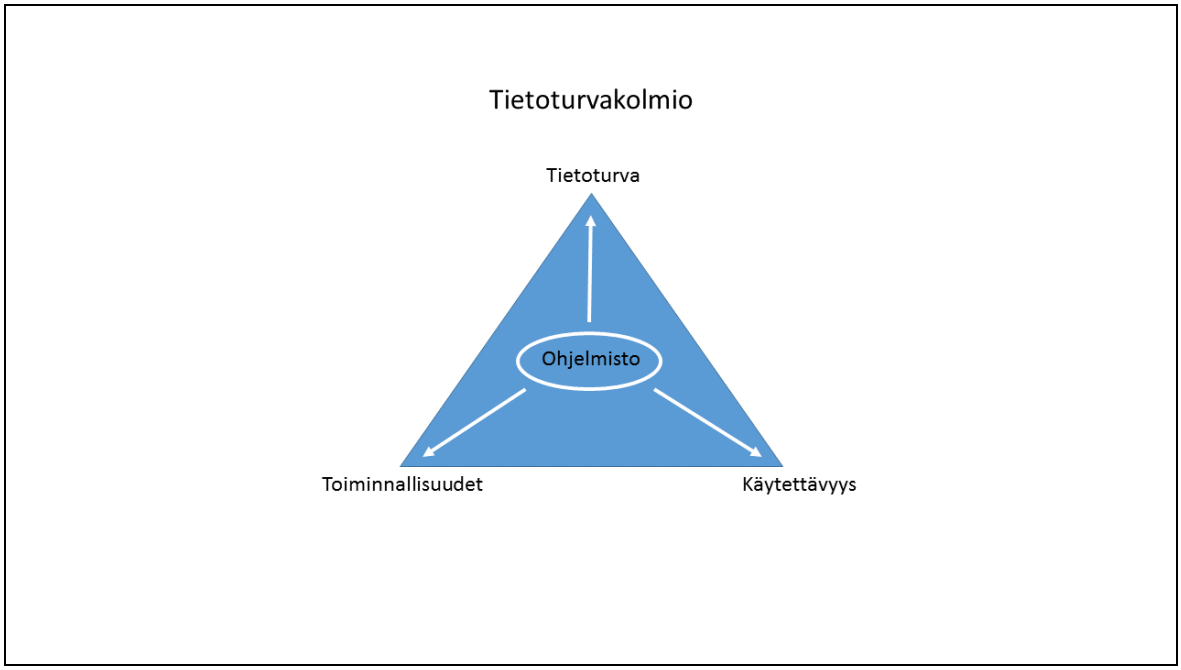
Opinnäytetyön lopputuloksesta, eli käytännön esimerkeistä ja ohjeista hyötyvät erityisesti Java-kieltä ohjelmoinnissa käyttävät web-ohjelmoijat sekä tietoturvatestauksesta kiinnostuneet. Opinnäytetyö tarjoaa konkreettisia ohjeita, miten web-sovellusten haavoittuvuuksien hyödyntämistä voidaan vaikeuttaa.

2 Web-sovellusten eettinen hakkerointi ja penetraatiotestaus

Luvussa esitellään web-sovellusten penetraatiotestaukseen liittyvät kaikkein olennaisimmat käsitteet, sekä miten eettinen hakkerointi eroaa laittomasti tapahtuvasta hakkeroinnista. Lisäksi esitellään tietoturvatestaukseen liittyviä peruskäsitteitä, kuten haavoittuvuus.

2.1 Tietoturva ohjelmistokehityksessä

Ohjelmistokehityksessä tietoturva käsitteenä voidaan ymmärtää monella tapaa. Yksi tapa kuvata tietoturvan, käytettävyyden ja toiminnallisuuksien suhdetta on perinteinen triangeli. Mitä tietoturvallisempaan suuntaan ohjelmistoa kehitetään, sitä vähemmän toiminnallisuuksia siihen voidaan sisällyttää ja sitä huonommaksi kyseisen ohjelmiston käytettävyys muuttuu (Walker 2014, 9). Ala olevassa kuvassa pyritään selventämään tätä suhdetta.



Kuvio 1. Tietoturvan suhde ohjelmiston käytettävyyteen ja toiminnallisuuksien laajuuteen (Walker 2014, 9)

2.2 Haavoittuvuus

Haavoittuvuus on suunnitteluvirhe, bugi tai muu heikko kohta sovelluksessa, joka mahdollistaa hakkerin hyökkäyksen (OWASP 2016a). Web-sovelluksissa olevat haavoittuvuudet ovat usein seurausta nopeatahtisesta ohjelmistokehityksestä (Erickson 2007, 451). Walkerin mukaan (2014, 12) haavoittuvuus on mikä tahansa heikkous, kuten ohjelmistovirhe

tai virheellisesti toimiva sovelluslogiikka, jota hyödyntämällä hakkeri voi aiheuttaa vahinkoa omaisuudelle.

Haavoittuvuuden olemassaoloa ei kuitenkaan voi suoraan rinnastaa tietoturvariskiin. Esimerkiksi fyysinen pääsy palvelimelle voidaan ajatella haavoittuvuudeksi, mutta mikäli palvelin sijaitsee lukitussa ja valvotussa tilassa, todennäköisyys haavoittuvuuden hyödyntämiselle voi olla sijainnin perusteella huomattavasti pienempi. (Walker 2014, 12.)

2.3 Hyökkäys

Hyökkäyksellä tarkoitetaan erilaisia tekniikoita, joita hakkerit käyttävät hyödyntääkseen web-sovellusten haavoittuvuuksia (OWASP 2016b.) Ennen varsinaista hyökkäystä hakkeri suorittaa kuitenkin erilaisia edeltäviä vaiheita. Ensimmäisessä tiedusteluvaiheessa kerätään tietoa kohteesta. Toisessa vaiheessa tapahtuvalla skannauksella etsitään tarkempaa tietoa kohteesta käyttäen tarkoitukseen sopivia työkaluja ja tiedusteluvaiheessa kerättyjä tietoja. Kolmannessa vaiheessa pyritään saamaan pääsy järjestelmään ja neljännessä pyritään varmistamaan pääsy järjestelmään jatkossakin. Viidennessä ja viimeisessä vaiheessa hakkeri yrittää peittää jälkensä esimerkiksi tyhjentämällä lokitiedostoja ja piilottamalla tiedostoja tai hakemistoja. (Walker 2014, 23.)

2.4 Hakkerointi

Myös hakkerille on olemassa erilaisia määritelmiä. Ericksonin (2007, 1) mukaan hakkeri on alun perin tarkoittanut henkilöä, joka keksii uusia tapoja hyödyntää olemassa olevia ratkaisuja ja kulloinkin voimassa olevia lakeja sekä soveltaa näitä ratkaisuja luovilla tavoilla. Uudenlaisen ohjelmiston menestys houkuttelee usein myös rikollisia etsimään tapoja hyödyntää ohjelmiston heikkouksia. Kaikki hakkerit eivät kuitenkaan tavoittele rahallista hyötyä vaan heitä saattaa motivoida uteliaisuus. Osa hakkereista auttaa sovellusten toimittajia korjaamaan haavoittuvuuksia ja heille maksetaan tehdystä työstä. (Erickson 2007, 451.)

Eettinen hakkeri on henkilö, joka suorittaa tietojärjestelmän omistajan luvalla penetraatiotestejä ja tietoturva-auditointeja (Walker 2014, 394). Eettinen hakkeri tai penetraatiotestaaja vastaa siis ohjelmiston tietoturvatestauksesta ja korjausehdotuksista, joilla voidaan mahdollisesti estää hyökkäykset (Engebretson 2013, 1). Termiä eettinen hakkerointi voidaan siis käyttää myös erottamaan laiton ja laillinen hakkerointi toisistaan (Engebretson 2013, 1).

Hakkereita luokitellessa käytetään joskus erivärisiä hattuja kuvaamaan hakkerin tarkoitusperiä. Mustahattu tarkoittaa hakkeria tai krakkeria, joka laittomasti ilman järjestelmän omistajan suostumusta pyrkii saavuttamaan henkilökohtaista hyötyä tai tekemään ilkivaltaa kohteena olevalle järjestelmälle. Valkohattu taas tarkoittaa eettistä hakkeria, jonka tehtävänä on varmistaa järjestelmän tietoturvallisuus suorittamalla erilaisia hakkerointiyrityksiä. Harmaahatut eivät kuulu varsinaisesti kumpaankaan edellä mainituista ryhmistä vaan he ovat kiinnostuneita hakkerointiin liittyvistä työkaluista ja tekniikoista sekä tietoturvauhkien esittelystä, joka omistajan luvalla tai laittomasti. (Walker 2014, 29.)

2.5 Penetraatiotestaus

Penetraatiotestauksen tarkoitus on tehdä ohjelmistoista tietoturvallisempia. Penetraatiotestauksessa pyritään käyttämään samoja työkaluja kuin hakkeri käyttäisi hyökkäyksessään. Kaikkein olennaisin ero hakkeroinnissa ja penetraatiotestauksessa on toiminnan laillisuus. Penetraatiotestaus on aina laillista ja valtuutettua toimintaa, kun taas hakkeroinnilla tarkoitetaan useimmiten rikollista toimintaa. Testauksen tuloksena syntyy aina suosituksia ja korjausehdotuksia, miten löytyneet ongelmakohdat voidaan korjata. Penetraatiotestausta voidaan kutsua myös eettiseksi hakkeroinniksi. Joitakin käytettyjä englanninkielisiä termejä ovat lisäksi Pen testing (PT), White hat hacking, Offensive security ja Red teaming. (Engebretson 2013, 1.)

Engebretson (2013, 2) erottaa vielä haavoittuvuuksien kartoituksen kokonaan erilliseksi prosessiksi penetraatiotestauksesta. Engebretsonin (2013, 2) mukaan haavoittuvuuksien kartoituksessa ohjelmistoista ja järjestelmistä etsitään potentiaalisia tietoturvauhkia, kun taas penetraatiotestauksessa pyritään demonstroimaan ja todistamaan konkreettisilla hyökkäyksillä haavoittuvuuksien olemassaolo.

Penetraatiotestaus voidaan jakaa samankaltaisiin vaiheisiin, kuten hakkerin suorittama hyökkäys. Engebretson (2013, 14) jättää kuitenkin pois viimeisen, jälkien peittelyvaiheen.

Ensimmäinen vaihe, tiedustelu (Reconnaissance), on kaikkein tärkein vaihe penetraatiotestauksessa. Tämän vaiheen avulla löydetään erilaisia kohteita, joihin hyökkäyksiä on mahdollista kohdistaa myöhemmissä vaiheissa. Tästä huolimatta se on kuitenkin usein väheksytty ja puutteellisesti suoritettu vaihe. Tätä vaihetta hankaloittaa selkeiden voimankäyttösääntöjen puute. Hyvä tiedustelija on osittain hakkeri, sosiaalinen hakkeri sekä yksityisetsivä. Tiedusteluvaihe voidaan vielä jakaa aktiiviseen ja passiiviseen tiedusteluun. Passiivinen tiedustelu rajoittuu internetin eri lähteiden käyttöön. Aktiiviseen tiedusteluun liittyy aina yhteyden ottaminen suoraan kohteeseen. Tästä syystä aktiivinen tiedustelu

vaatii usein myös kohteen valtuutuksen, muussa tapauksessa toiminta saattaa olla laitonta. (Engebretson 2013, 20.)

Tiedusteluvaiheessa koko web-sivusto ladataan kuvineen, sivuineen, linkkeineen ja koodeineen tarkoitukseen sopivalla ohjelmistolla. Saatua kopiota voidaan tutkia tarkemmin ilman, että ollaan yhteydessä itse kohteeseen. Kokonaisen web-sivuston lataamiseen tarvitaan aina lupa. Tiedustelussa käytettyjä muita työkaluja ovat esimerkiksi internetin hakukoneet ja näistä erityisesti Google. Google Hacking on tekniikka, jossa web-sivuston hakutuloksia rajataan Googlen hakuoperaattorien avulla. Tällöin web-sivustolta saatetaan löytää arkaluonteista tietoa. (Engebretson 2013, 25.)

Osa tiedustelua on myös sähköpostiosoitteiden keräys (Engebretson 2013, 31). Lisäksi kerätään domain ja dns-tietoja (Engebretson 2013, 34). Näiden tietojen keräämiseen on olemassa suuri määrä työkaluja, mutta näihin ei tässä opinnäytetyössä perehdytä tarkemmin.

Sosiaalisen hakkeroinnin osuus hyödyntää inhimillisiä heikkouksia ja sen tarkoituksena on saada ihmisiä paljastamaan arkaluonteista tietoa (Engebretson 2013, 48). Tiedusteluvaiheen osana kerätään lista IP-osoitteista, jota käytetään lähteenä seuraavaksi tapahtuvalle skannausvaiheelle (Engebretson 2013, 49).

Tiedusteluvaihe voi sisältää esimerkiksi yritysten työpaikkailmoitusten tutkimisen, joka voi tarjota tietoa yrityksen käyttämistä teknologioista. Myös yrityksen julkistamat uutiset ja tiedotteet voivat sisältää hyödyllistä tietoa. (Engebretson 2013, 22.)

Tiedusteluvaihetta siis seuraa skannausvaihe (Scanning). Vaihe voidaan jakaa neljään osaan, joista ensimmäinen on ping-pakettien lähetys kerättyihin IP-osoitteisiin. Ping-paketteihin saadut vastaukset paljastavat, että IP-osoitteessa on kommunikointiin kykenevän verkkolaite. On syytä huomata, että vaikka vastauksia ei saataisikaan, jatketaan seuraavaan vaiheeseen. Seuraavaksi suoritetaan porttiskannaus, jolla pyritään selvittämään kaikki palvelimen avoimet portit. Tämän jälkeen pyritään saamaan lisää tietoa kohdistamalla tarkempi skannaus avoimiin portteihin. Varsinainen haavoittuvuusskannaus on viimeinen vaihe, jossa voidaan löytää haavoittuvuuksia, joita kohtaan hyökätään seuraavassa vaiheessa. (Engebretson 2013, 54.)

Kolmantena vaiheena penetraatiotestauksessa tulee hyväksikäyttövaihe (Exploitation). Tämä on se vaihe, jossa varsinainen hakkerointi tapahtuu. Tavoitteena on saada kohde hallintaan. (Engebretson 2013, 79.)

Web-sovellusten osalta hyväksikäyttövaihe voidaan suorittaa tarkoitukseen soveltuvilla ohjelmistoilla. Tarvittavat ominaisuudet voidaan jakaa kolmeen osaan. Ensimmäinen tarvittava ominaisuus on proxy, jonka avulla voidaan muokata http-pyyntöjä ennen kuin ne lähtevät selaimesta. Toinen tarvittava ominaisuus on web-sivuston kopiointi, ja kolmantena tulee vielä kyky analysoida palvelimen vastauksia ja pyrkiä löytämään niistä haavoittuvuuksia. (Engebretson 2013, 143.)

Viimeisenä vaiheena on jälkihyväksikäyttö ja tunkeutumiseen käytetyn yhteyden säilyttäminen (Engebretson 2013, 168.) Kokonaisuutena suoritettuna nämä neljä vaihetta muodostavat web-sovelluksen täydellisen penetraatiotestauksen (Engebretson 2013, 14.)

3 Web-sovellusten haavoittuvuudet

Luvussa esitellään web-sovellusten yleisimpiä haavoittuvuuksia sekä selvitetään, miten kyseiset haavoittuvuudet voidaan korjata ohjelmistokoodissa. Haavoittuvuudet on valittu OWASP Top 10, 2013 -listalta sen perusteella liittyvätkö haavoittuvuudet itse ohjelmistokoodiin.

OWASP on voittoa tavoittelematon kansainvälinen avoimenlähdekoodin järjestö. OWASP:in tarkoitus on auttaa organisaatioita hankkimaan, käyttämään, kehittämään ja ylläpitämään ohjelmistoja tietoturvallisesti. Järjestö tarjoaa ilmaiseksi erilaisia avoimenlähdekoodin ohjelmistotestaustyökaluja sekä erilaisia ohjeita tietoturvalliseen ohjelmointiin sekä tietoturvatestaukseen liittyen. Järjestön ylläpitämä web-sivusto avattiin 2001. (OWASP 2016j.)

OWASP:in Top Ten -projektin tarkoitus on listata ajankohtaisimmat ja kaikkein kriittisimmät web-sovelluksiin kohdistuvat tietoturvauhkat ja riskit. Lista ei ole kuitenkaan tarkoitettu standardiksi, vaan järjestön periaatteena on jakaa tietoa. Lista on tuotettu vuosina 2003, 2004, 2007, 2010 ja 2013. Projektiin osallistuu maailmanlaajuisesti erilaisia tietoturva-alan ammattilaisia, jotka yhteistyössä kokoavat listan ja siihen liittyvän dokumentoinnin. Dokumentointi sisältää kuvauksen kyseisistä tietoturvauhkista ja riskeistä sekä erilaisia koodiesimerkkejä sisältäen hyökkäyksissä käytetyt koodit ja koodiin vaadittavat korjaukset. Järjestö tuottaa myös ohjeistuksen, miten välttää web-sovellusten haavoittuvuuksiin kohdistuvia hyökkäyksiä, sekä miten web-ohjelmistoja testataan oikealla tavalla. (OWASP 2016k.)

3.1 HTTP-metodit

Web-sovellusta käytettäessä käyttäjän Web-selain lähettää kutsuja palvelimelle käyttäen http-pyyntöjä. Http-metodit ovat erityyppisiä toimintoja, joita voidaan suorittaa pyynnön osoittamassa kohteessa eli halutulla web-palvelimella. Näistä käytetyimmät ovat post- ja get-metodit, jotka voivat molemmat lähettää tietoa palvelimelle. Erona näillä metodeilla on tiedon lähettämistapa. Get-metodi lisää lähetettävät tiedot osaksi URI-osoitetta ja post-metodi taas sijoittaa tiedot http-pyynnön sisään, jolloin tiedot eivät ole suoraan luettavissa selaimen osoiteriviltä. (Scambray, Liu & Sima 2010, 4.)

Seuraavassa esimerkissä kuvataan get- ja post-metodien eroavaisuuksia. Alla olevassa kuvassa käyttäjä kirjautuu järjestelmään tunnuksillaan esimerkkisovelluksen kirjautumissivulla.

Kuvio 2. Sisäänkirjautuminen

Sisäänkirjautumisen jälkeen sovellus näyttää käyttäjän tiedot. Alla olevassa kuvassa näkyy, miten selaimen osoiteriviltä voidaan lukea käyttäjän syöttämät tiedot eli käyttäjätunnus ja salasana parametreina ”user” ja ”password”. Tämän kaltaisen tiedon näkyminen eri yhteyksissä saattaa paljastaa arkaluonteista tietoa.

Kuvio 3. Sisäänkirjautunut käyttäjä

Tällöin on mahdollista myös muokata tietoja suoraan osoiterivillä, jonka jälkeen muuttuneet tiedot lähetetään uudelleen sovellukselle painamalla enteriä. Alla olevassa kuvassa käyttäjä on vaihdettu ”Maijaksi” ilman uudelleenkirjautumista.

Kuvio 4. Osoiterivillä muokattu user-parametri

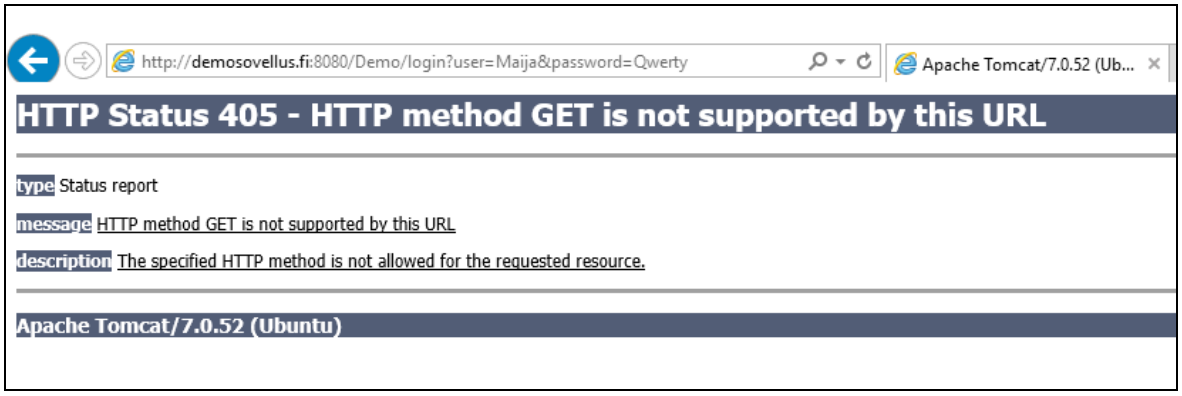
Tämä ongelma on mahdollista korjata vaihtamalla sovelluksen web-lomakkeen lähetyksen käytetyn get-metodin tilalle post-metodi. Lisäksi java-koodissa täytyy olla olemassa vastaava post-metodi. Muutoksen jälkeen get-metodi voidaan poistaa kokonaan java-koodista. Alla olevassa koodiesimerkissä on sovelluksen käyttämä lomake, joka käyttää tietojen lähetyksessä post-metodia.

```
<form action="login" method="post">
<label>Käyttäjätunnus:</label>
<input type="text" name="user"/>
<br><br>
<label>Salasana:</label>
<input type="password" name="password"/>
<br><br>
<button type="submit">Kirjaudu</button>
</form>
```

Kun lomakkeen ”method” -kohtaan on vaihdettu ”post” aikaisemman ”get”-metodin sijaan voidaan havaita, että parametreja ei enää paljasteta selaimen osoiterivillä vaan selain lähettää ne IP-paketin sisällä. Selaimen osoiterivillä ei enää näy yhtään parametria. Alla olevassa kuvassa ei ole enää nähtävissä parametreja.

Kuvio 5. Osoiteriviltä ei ole enää luettavissa parametreja

Voimme nyt myös havaita get-metodin puuttumisen kokeilemalla aikaisemman esimerkin osoiterivillä muokattua user-parametria. Alla olevan kuvassa näkyy virheilmoitus, joka on seurausta poistetusta get-metodista. Tällaisen virheilmoituksen näyttäminen käyttäjälle paljastaa palvelimen tietoja. Tästä syystä on suositeltua käyttää kustomoituja virheilmoituksia.



Kuvio 6. Get-metodin testaus

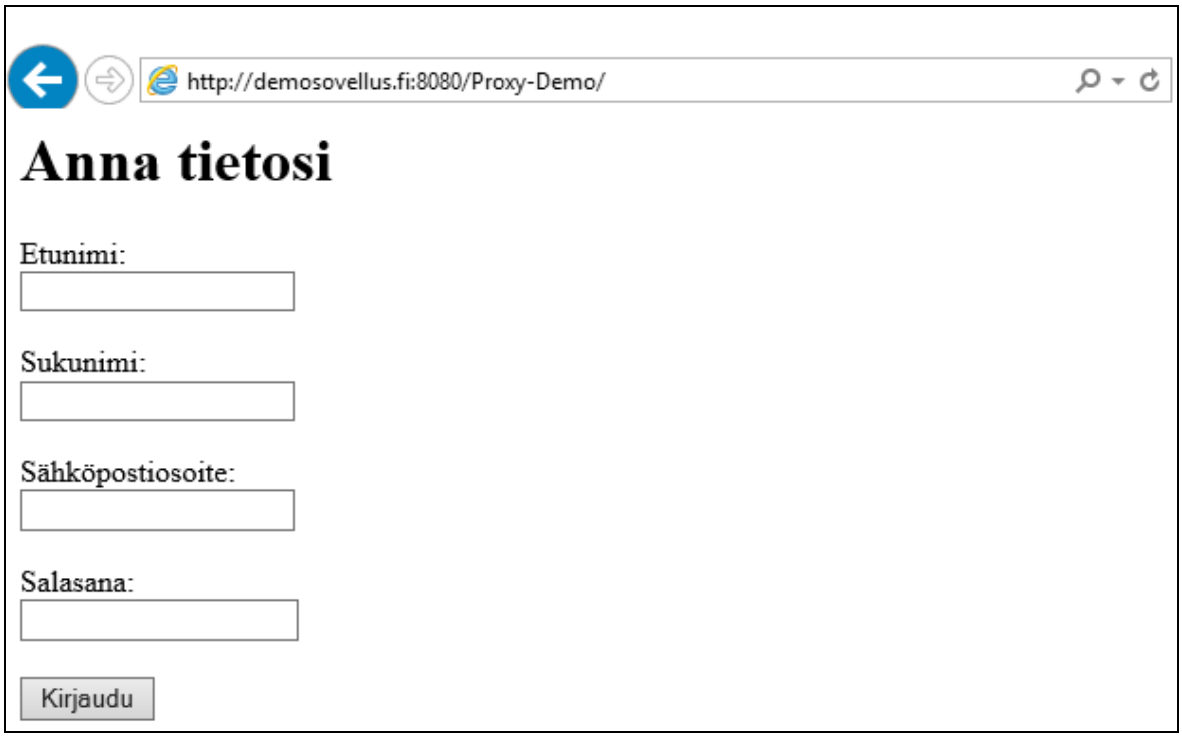
On tärkeää huomata, että post-metodi ei kuitenkaan suojaa lähetettäviä tietoja millään tavalla paremmin. Parametrien muokkaaminen on edelleen mahdollista, joskin hieman vaikeampaa, kuten seuraavassa kappaleessa selvennetään. Siirtyminen post-metodin käyttöön kuitenkin vähentää tietojen väärinkäytön mahdollisuuksia, esimerkiksi selaimen välimuistin tai palvelimen lokia tarkasteltaessa. (Scambray, Liu & Sima 2010, 5.)

3.2 Käyttöliittymän ohitus

Edellisessä luvussa kuvattiin, miten web-sovelluksessa olevan lomakkeen tietojen lähetys tapahtuu web-palvelimelle get- ja post-metodien avulla. Käyttäjä siis antaa syötteet ja ne lähetetään parametreina web-sovelluksen käsiteltäviksi.

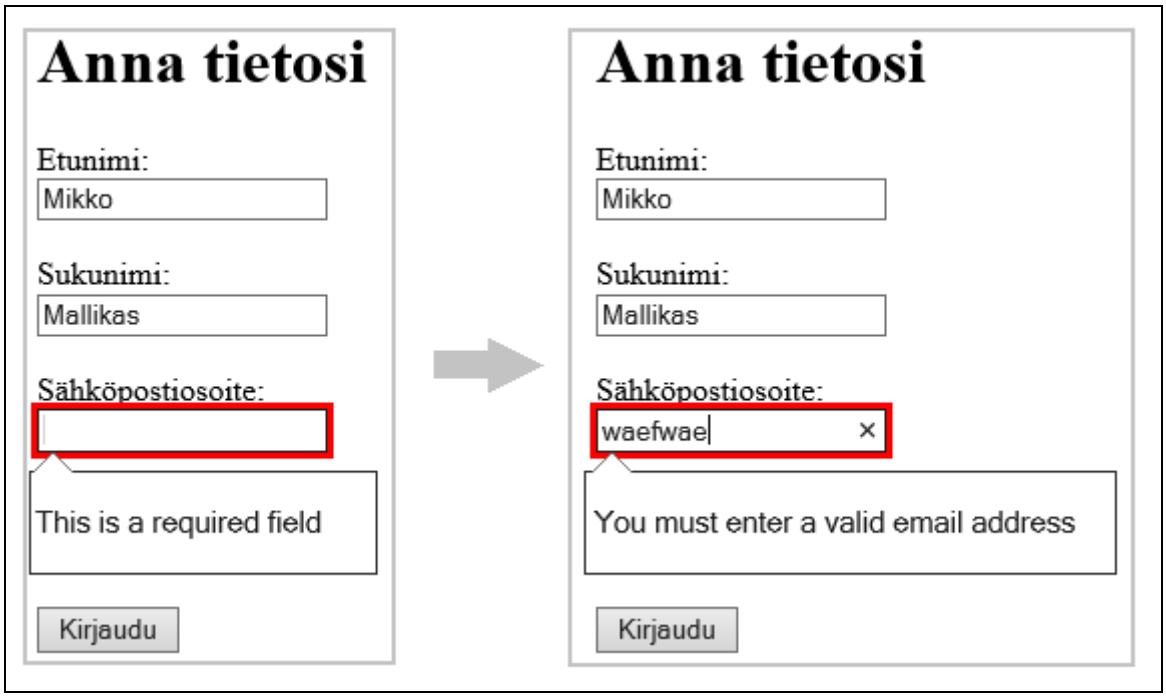
Käyttäjän lomakkeelle syöttämiä tietoja on kuitenkin mahdollista muokata ennen niiden lähetystä web-sovellukselle. Käyttämällä tarkoitukseen soveltuvaa proxy-palvelinta, lomakkeen parametreja voidaan muokata ennen kuin lähetys tapahtuu. Proxy keskeyttää lomakkeen lähetyksen ja näyttää kaikki lomakkeella olevat parametrit ja niihin syötetyt tiedot. Tietoja on tämän jälkeen mahdollista vapaasti muokata, jonka jälkeen proxy jatkaa lomakkeen tietojen lähetystä web-sovellukselle. Hyökkääjä voi siis näin täysin kontrolloida, mitä web-sovellukselle lähetetään. Tästä syystä web-sovelluksen tulisi aina käsitellä käyttäjältä tulevat syötteet ei luotettavana tietona. (Engebretson 2013, 161.)

Alla olevassa esimerkissä on kuvattuna web-sovelluksen rekisteröitymisprosessin normaali kulku. Web-sovellus näyttää lomakkeen, jossa kysytään käyttäjän tietoja. Web-sovelluksen sähköpostiosoitekentässä käytetään HTML5 input -tyyppiä ”email” ja attribuuttia ”required”.



Kuvio 7. Rekisteröityminen

Mikäli käyttäjä jättää sähköpostiosoitteen antamatta, annetaan alla olevassa kuvassa näkyvä virheilmoitus. Lisäksi, jos sähköpostiosoite ei noudata oikeaa muotoa, annetaan vastaavasti siitä kertova virheilmoitus.

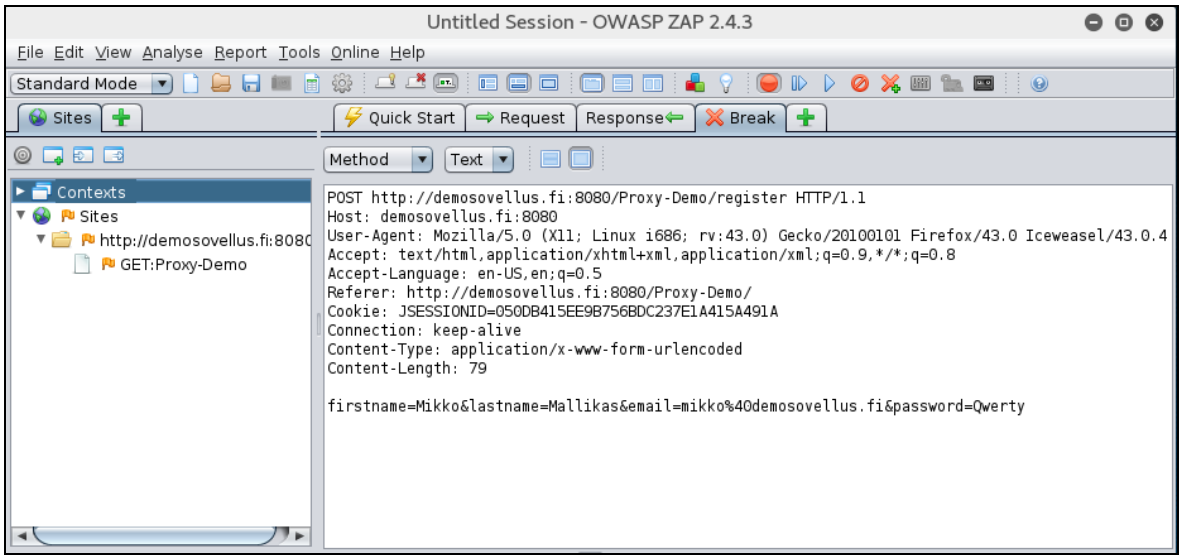


Kuvio 8. HTML5 input atribuutit

Lopuksi, kun tiedot on annettu oikeassa muodossa, käyttäjä saa ilmoituksen onnistuneesta rekisteröitymisestä, sekä tiedon, että käyttäjätunnus on käyttäjän antama sähköpostiosoite. Web-sovellus siis käyttää annettua sähköpostiosoitetta käyttäjätunnuksena. Alla olevassa kuvassa on sovelluksen näyttämät tiedot.

Kuvio 9. Onnistunut rekisteröityminen

Mikäli web-sovellus luottaa ainoastaan HTML5:n tarjoamaan validointiominaisuuteen, voidaan käyttöliittymän tarkistukset ohittaa melko helposti. Alla olevassa esimerkissä käytetään ZAP Attack Proxyä, jonka avulla ohitetaan käyttöliittymän tekemät tarkistukset.



Kuvio 10. ZED attack proxyn käyttöliittymä

Sähköpostiosoite on nyt vapaasti muokattavissa haluttuun muotoon. Alla olevassa kuvassa on valittuna email-parametri.

```
POST http://demosovellus.fi:8080/Proxy-Demo/register HTTP/1.1
Host: demosovellus.fi:8080
User-Agent: Mozilla/5.0 (X11; Linux i686; rv:43.0) Gecko/20100101 Firefox/43.0 Iceweasel/43.0.4
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Referer: http://demosovellus.fi:8080/Proxy-Demo/
Cookie: JSESSIONID=050DB415EE9B756BDC237E1A415A491A
Connection: keep-alive
Content-Type: application/x-www-form-urlencoded
Content-Length: 79

firstname=Mikko&lastname=Mallikas&email=mikko%40demosovellus.fi&password=Qwerty
```

Kuvio 11. Lomakkeelle syötetyt parametrit

Seuraavaksi vaihdetaan oikeanmuotoisen sähköpostiosoitteen tilalle käyttäjän muokkaama teksti. Alla olevassa kuvassa näkyy käyttäjän muokkaama tieto.

```
POST http://demosovellus.fi:8080/Proxy-Demo/register HTTP/1.1
Host: demosovellus.fi:8080
User-Agent: Mozilla/5.0 (X11; Linux i686; rv:43.0) Gecko/20100101 Firefox/43.0 Iceweasel/43.0.4
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Referer: http://demosovellus.fi:8080/Proxy-Demo/
Cookie: JSESSIONID=050DB415EE9B756BDC237E1A415A491A
Connection: keep-alive
Content-Type: application/x-www-form-urlencoded
Content-Length: 79

firstname=Mikko&lastname=Mallikas&email=miksu&password=Qwerty
```

Kuvio 12. Käyttäjän ZAP:ssa muokkaama email-parametri

Lopuksi sovellus näyttää käyttäjän muokkaaman käyttäjätunnuksen, joka ei noudata web-sovelluksen olettamaa sähköpostiosoitteen muotoa. Alla olevassa kuvassa näkyy parametrin muokkauksen lopputulos. Käyttäjä on näin onnistunut ohittamaan lomakkeella olevan tarkistuksen.

Kuvio 13. Onnistunut rekisteröityminen

3.3 Cross-Site Scripting (XSS)

Cross-Site Scripting (XSS) -hyökkäyksessä pyritään sijoittamaan haitallista koodia hyökkäyksen kohteena olevan web-sivuston käyttäjien suoritettavaksi. Haitallinen koodi voi olla esimerkiksi osana linkkiä keskustelufoorumilla, tai se voidaan myös lisätä osaksi sivuston web-osoitetta. Haitallista koodia saatetaan kutsua myös kokonaan web-sivuston ulkopuolisesta osoitteesta. Hyökkäyksen onnistuminen edellyttää, että hakkerin kohteena olevasta web-sovelluksesta löytyy Cross-Site Scripting -haavoittuvuus. Haavoittuvuuden löytyminen web-sovelluksesta taas edellyttää, että kyseisestä sovelluksesta löytyy ominaisuus vastaanottaa käyttäjän syötteitä, sekä näyttää annetut syötteet ilman niille suoritettavaa tarkistusta. Hyökkääjän tavoitteena on usein varastaa käyttäjän eväste, jonka avulla voi olla mahdollista ottaa haltuun käyttäjän identiteetti web-sovelluksessa, tai harhauttaa käyttäjä paljastamaan salasanansa hyökkääjälle. Tällä tekniikalla toteutetuista sosiaalisen hakkeroinnin hyökkäyksistä ovat kärsineet myös jotkin suositut palvelut kuten Gmail ja Hotmail. Hyökkäyksissä käytetään tavallisimmin JavaScriptiä. (Scambray, Liu & Sima 2010, 233.)

Cross-Site Scripting -haavoittuvuudessa on siis kyse skriptien syöttämisestä suoritettavaksi osaksi web-sovellusta. Cross-Site Scripting -haavoittuvuus kohdistuu käyttäjän selaimeen web-palvelimen sijasta. Itse suoritettava skripti voi sijaita palvelimella, tai se voidaan antaa myös osana web-sivulle johtavaa linkkiä. Hyökkääjän skripti suoritetaan siten

osana sovellusta, aivan kuten se kuuluisi web-sovelluksen alkuperäiseen koodiin. Tällä tavalla on myös mahdollista ohittaa suojausrajoite, joka estää web-sovellusta pääsemästä käsiksi käyttäjän toisessa web-sovelluksessa tallentamaan tietoon. Tämä on mahdollista, koska haitallinen koodi on käyttäjän selaimen tulkitsemana osa luotettua web-sovellusta. Cross-Site Scripting -haavoittuvuuden seurausten vakavuus voi vaihdella merkittävästi. XSS:n avulla voidaan luoda erilaisia web-sivuston käyttäjää häiritseviä ponnahdusikkunoita, tai pahimmassa tapauksessa vaarantaa koko web-sivuston tietoturva. (Engebretson 2013, 157.)

Hyökkäyksissä hyödynnetään usein web-sovelluksesta löytyvää hakukenttää. Haitallinen skripti syötetään hakukenttään ja lähetetään sen avulla suoritettavaksi. (Scambray, Liu & Sima 2010, 233.)

Kun skripti annetaan osana käyttäjälle tarjottua linkkiä, on kyse heijastetusta Cross-Site Scriptingistä (reflected XSS). Tällöin skripti suoritetaan, kun käyttäjä klikkaa saamaansa linkkiä. Haitallinen koodi lähetetään käyttäjän selaimesta haavoittuvuuden omaavalle palvelimelle ja palvelin lähettää eli heijastaa koodin takaisin käyttäjän selaimen suoritettavaksi. Tämän tyyppinen XSS voidaan luokitella myös tietojenkalasteluksi. (Engebretson 2013, 159.)

Tallennetussa Cross-Site Scriptingissä (stored XSS) hyökkääjä on onnistunut tallentamaan haitallisen skriptin palvelimelle. Tällöin jokaisen käyttäjän selain suorittaa skriptin käyttäessään web-sovelluksen haitallista koodia sisältämää osaa. XSS-koodi voi olla tallennettuna esimerkiksi web-sovelluksen käyttämään tietokantaan. (Engebretson 2013, 159.)

Lisäksi on vielä olemassa DOM-pohjainen Cross-Site Scriptingin tyyppi. DOM-pohjaisessa XSS:ssä ei hyödynnetä palvelinta vaan hyökkäys tapahtuu kokonaisuudessaan käyttäjän selaimessa. Hyökkääjä voi myös käyttää erityyppisten XSS-hyökkäysten yhdistelmiä. (OWASP 2016n.)

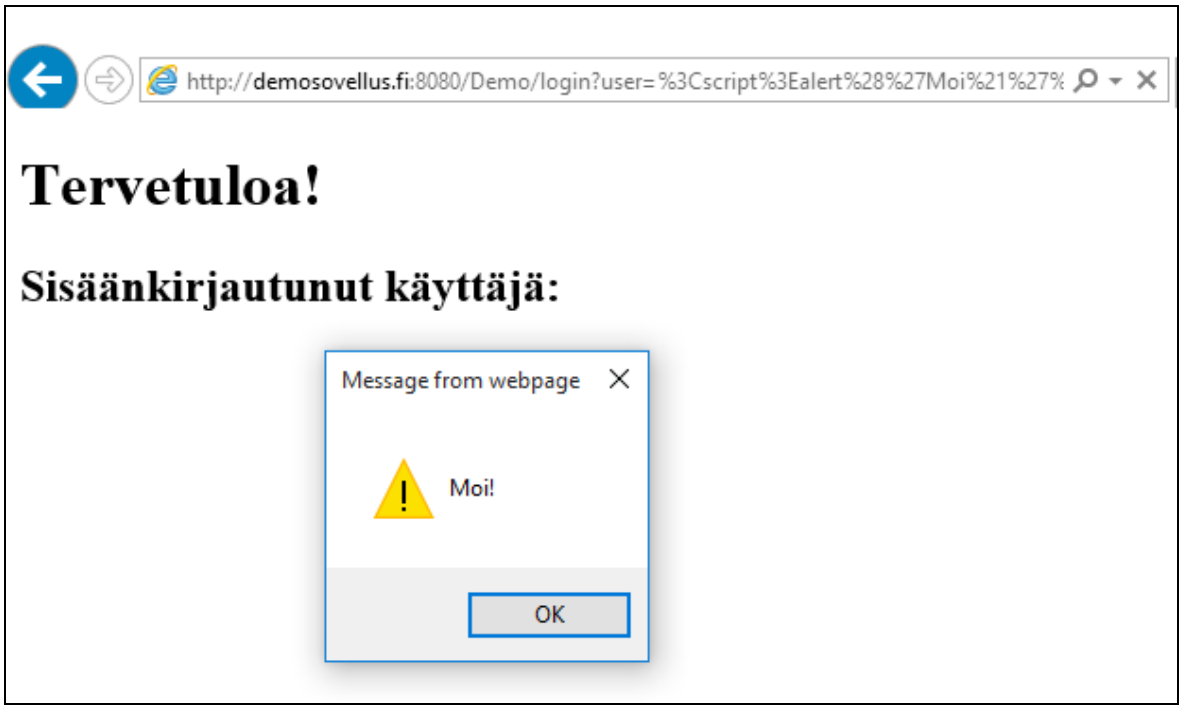
Yksinkertainen Cross-Site Scripting -haavoittuvuuden testaaminen voidaan toteuttaa syöttämällä JavaScriptiä johonkin web-sivulla olevan lomakkeen syöttökentistä. Seuraavassa esimerkissä testataan, miten penetraatiotestaaja voi havaita Cross-Site Scripting -haavoittuvuuden sovelluksessa. Esimerkissä käytetty JavaScript-koodi on kokonaisuudessaan alla.

```
<script>while(1){alert('Moi!')};</script>
```

Alla olevassa kuvassa näkyy käytetty JavaScripti syötettynä käyttäjätunnuskenttään sovelluksen odottaman käyttäjätunnuksen sijasta. JavaScript-koodi kirjoitetaan käyttäjätunnuskenttään ja painetaan kirjaudu-nappia.

Kuvio 14. JavaScriptiä sijoitettuna käyttäjätunnus kenttään

Lomakkeen tiedot lähetetään palvelimelle ja selain suorittaa syötetyn JavaScriptin, joka näyttää käyttäjälle ponnahdusikkunan. Ponnahdusikkunan JavaScriptissä käytetään while-silmukkaa, joten ponnahdusikkuna avautuu sulkemisen jälkeen yhä uudelleen ja uudelleen. Käytetystä selaimesta riippuen, käyttäjän ainoaksi vaihtoehdoksi saattaa jäädä selaimen prosessin lopettaminen tehtävienhallinnan avulla. Alla olevassa kuvassa näkyy JavaScriptin muodostama ponnahdusikkuna. Ponnahdusikkuna paljastaa, että syötetty JavaScript-koodi todella suoritetaan ja sovellus siten sisältää Cross-Site Scripting -haavoittuvuuden.



Kuvio 15. Selaimen suorittaman JavaScriptin tuottama ponnahdusikkuna

Myös web-sivun HTML-koodin muokkaus onnistuu JavaScriptin avulla. Seuraava skripti vaihtaa websivun otsikon (H2) tilalle Googlen web-sivulta ladatun kuvan. Taitava hakkeri voi siis näin mahdollisesti muokata web-sivustoa merkittävästi alkuperäisestä poikkeavaksi.

```
<script>document.getElementsByTagName("h2")[0].innerHTML = "<img
src='https://www.google.fi/images/branding/googlelogo/1x/googlelogo_col
or_272x92dp.png'/>";</script>
```

On syytä huomioida, että web-palvelimella sijaitseva alkuperäinen sivusto säilyy koko ajan koskemattomana. Näkymä, joka selaimessa näytetään käyttäjälle, on kuitenkin muuttunut olennaisesti. Toisin sanoen hyökkäys kohdistuu palvelimen sijasta käyttäjään, mutta hyökkäyksen mahdollistaa XSS -haavoittuvuuden sisältävä web-sovellus.

XSS -hyökkäys voidaan toteuttaa lomakkeen sijasta myös osana URL-osoitetta, jolloin JavaScriptillä varustettu linkki voidaan lähettää esimerkiksi sähköpostilla tai lisätä web-sivuston keskustelufoorumille. Tällöin on mahdollista, että käyttäjä luottaa linkkiin sen alkuosan, tässä tapauksessa ”demosovellus.fi” perusteella. Alla olevassa esimerkissä web-sovelluksessa käytetään get-metodia, joka osaltaan mahdollistaa XXS-haavoittuvuuden.

Alla olevassa esimerkissä JavaScript-koodi sijoitettuna web-sovelluksen URL-osoitteen jälkeen tuottaa samankaltaisen ponnahdusikkunan kuin kuviossa yhdeksän.

```
http://demosovellus.fi:8080/Demo/login?user=Mikko<script>while(1){aler
t('Moi!')};</script>
```

Myös käyttäjän web-sovellukseen lisäämä kuva voi sisältää haitallista koodia (OWASP 2016m). Alla on esimerkki HTML-kuvaan upotetusta XSS-hyökkäyksestä. Kuvan lähde (src) on normaalin kuvan sijasta JavaScript-koodia. Lisäksi kuvan korkeudeksi ja leveydeksi on asetettu nolla eli se ei ole normaalisti havaittavissa selaimessa. Lähteenä oleva JavaScript-koodi suoritetaan sivun latautuessa ja käyttäjän huomaamatta.

```
<img src=javascript:alert('Moi!') height="0" width="0">
```

Cross-Site Scripting -haavoittuvuudelta suojaudutaan, tarkistamalla ei luotettu data, ennen kuin se tulostetaan web-sivulle. Eräs keino suojautua heijastettua XSS:ää vastaan on käyttää JSTL core-kirjaston escapeXml-attribuuttia. Tällöin tietojen tulostamiseen JSP-sivulle voidaan käyttää kirjaston tarjoamaa ”<c:out>” tagia, joka muuntaa <, >, &, ', " -merkit vastaaviksi merkkientiteeteiksi. JSTL-kirjasto voidaan ottaa käyttöön JSP-sivulla käyttäen alla olevaa syntaksia. (OWASP 2016c.)

```
<!-- JSTL core-kirjasto -->
<%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c" %>

<!-- Escape XML:n käyttö estää XSS:n -->
<c:out escapeXml="true" value="${user}"/>

<!-- Tietojen tulostustapa, joka mahdollistaa XSS:n -->
<% String username = request.getParameter("user"); %>
```

Escape XML:n käyttöönoton jälkeen lomakkeelle syötetty JavaScript tulostuu suoraan sivulle ilman, että selain suorittaa sitä. Alla olevassa kuvassa näkyy muutoksen vaikutukset eli selain ei enää suorita JavaScriptiä.

Kuvio 16. Escape XML

Näiden esimerkkien lisäksi JavaScriptiä hyödyntäen voidaan suorittaa paljon muitakin tietoturvaa uhkaavia toimintoja, kuten session kaappaus, erilaisten komentojen suorittaminen kohteessa, näppäinpainallusten tallentaminen tai sen avulla voidaan päästä käsiksi web-sovelluksen suojattuihin osiin. (Engebretson 2013, 157.)

3.4 Puutteellinen tunnistusmenettely ja istunnonhallinta

Tunnistusmenettelyllä tarkoitetaan web-sovelluksen käyttäjän kirjautumisen ja tunnistautumisen yhteydessä suoritettavia toimia (OWASP 2016f). Tunnistautumisen tarkoitus on varmistaa kommunikoijan identiteetti (OWASP 2016g).

Istunnonhallinnan tehtävä on tallentaa käyttäjän identiteetti sisäänkirjautumisen jälkeen ja todentaa käyttäjä web-sivustolla ilman käyttäjän jatkuvaa uudelleen kirjautumista. Käyttäjän identiteettitiedot tallennetaan istuntoevästeeseen. Palvelin luo evästeen ja lähettää sen käyttäjän selaimeen. (OWASP 2016e.)

Eväste on tekstipohjainen tiedosto, johon web-palvelin voi tallentaa identiteetin lisäksi myös muita sivuston käyttäjään liittyviä tietoja (Walker 2014, 215). Palvelin luo evästeen ja lähettää sen käyttäjän selaimeen (OWASP 2016e.)

Huonolla tavalla toteutettu Web-sovellus saattaa paljastaa hakkerille olemassa olevan käyttäjätunnuksen. Yksi mahdollinen tapa on kirjautumisen yhteydessä saatava virheilmoitus. Mikäli virheilmoitus kertoo käyttäjän syöttäneen virheellisen salasanan, voi hakkeri tästä päätellä käyttäjätunnuksen olevan olemassa. Parempi tapa onkin kertoa virheilmoituksessa, että käyttäjä on syöttänyt virheellisen käyttäjätunnuksen tai salasanan, jolloin hakkeri ei voi päätellä virheilmoituksesta kumpi tiedoista on annettu väärin. (Scambray, Liu & Sima 2010, 125.)

Toinen mahdollinen tapa on virheellisesti toteutettu salasanan palautustoiminto, jossa käyttäjä voi palauttaa salasanan syöttämällä sähköpostiosoitteensa. Oikein toteutettuna palautustoiminto ei ilmoita käyttäjälle, onko palveluun rekisteröity käyttäjän syöttämää sähköpostiosoitetta. (Scambray, Liu & Sima 2010, 126.)

Muita tapoja päätellä olemassa oleva käyttäjätunnus on esimerkiksi rekisteröitymisprosessi, jossa sovellus ilmoittaa, onko käyttäjän syöttämä käyttäjätunnus jo käytössä palvelussa sekä web-sovelluksen url-osoite, joka sisältää käyttäjätunnuksen, esimerkiksi (http://www.demosovellus.fi/~a1234567). (Scambray, Liu & Sima 2010, 126.)

Koska evästeet sisältävät usein tietoja, joita käytetään käyttäjän todennuksessa, evästeitä myös pyritään kaappaamaan hakkereiden toimesta. Evästeiden kaappaamisessa yleisimmät käytetyt tekniikat ovat Cross-Site Scripting ja tietoliikenteen salakuuntelu. (Erickson 2007, 152.)

Tunnistusmenettely voidaan, joissain tapauksissa ohittaa evästeessä olevan istuntotunnisteen uudelleenlähetyksellä. Tällöin hyökkääjä uudelleen lähettää kaappaamansa evästeen palvelimelle oman evästeensä sijasta. Muita tapoja ohittaa tunnistusmenettely ovat SQL-injektio, josta kerrotaan kappaleessa 3.4, Cross-Site Request Forgery, josta lisää kappaleessa 3.5 sekä suora sivuviittaus, josta tarkemmin kappaleessa 3.6. (Scambray, Liu & Sima 2010, 151.)

Salasana tulisi aina käyttäjän rekisteröitymisprosessin osana tallentaa käyttäen tarkoitukseen soveltuvaa salausalgoritmia ja suolaa. Suola on satunnaisesti generoitu merkkijono, joka lisätään käyttäjän syöttämän salasanan jatkoksi. Esimerkiksi tiedoston tallennukseen palvelimelle liittyvä haavoittuvuus voi mahdollistaa hakkerille koko salasanalistan latauksen palvelimelta. Tästä syystä salasanat tulisi olla salakirjoitettuna ja suolattuna tietokannassa. (OWASP 2016p.)

Salakirjoitukseen ja suolaukseen voidaan käyttää Javan SecretKeyFactory- ja PBEKeySpec-luokkia. Alla on esimerkki metodista ja luokkien käytöstä. Metodille annetaan parametreinä käyttäjän syöttämä salasana, suolana käytetty satunnainen merkkijono sekä salausalgoritmin toistojen lukumäärä. (OWASP 2016p.)

```
public static byte[] hashPassword( final char[] password, final
byte[] salt, final int iterations, final int keyLength ) {

try {
    SecretKeyFactory skf = SecretKeyFactory.getInstance(
    "PBKDF2WithHmacSHA512" );
    PBEKeySpec spec = new PBEKeySpec( password, salt, iterations,
    keyLength );
    SecretKey key = skf.generateSecret( spec );
    byte[] res = key.getEncoded( );
    return res;

} catch( NoSuchAlgorithmException | InvalidKeySpecException e ) {
           throw new RuntimeException( e );
       }
}
```

Hyökkääjä voi kerätä sivustolle kirjautumalla kokoelman evästeitä ja sitten tutkia niissä esiintyviä eroavaisuuksia. Eroavaisuudet saattavat paljastaa mahdollisia heikkouksia tunnistusmenettelyssä, joita hakkeri voi sitten hyödyntää muokkaamalla keräämiään evästeitä. Hakkerin muokkaama eväste voi siten mahdollistaa web-sovelluksen luvattoman käytön tai sovelluksen tietojen manipuloinnin. (OWASP 2016e.)

Istuntotunnisteen ennustaminen on aikaisemmin laajasti käytetty hyökkäystekniikka, jonka mahdollisti web-sovelluksissa käytetty huonosti toteutettu istuntotunnisteen luontitekniikka. Tällöin hyökkääjän oli mahdollista ennustaa seuraavaksi saatava istuntotunniste keräämiensä näytteiden perusteella. Toinen keino hankkia käyttäjän istuntotunniste on Brute-forcing, jossa palvelimelle lähetetään tuhansittain pyyntöjä erilaisilla istuntotunnisteilla varustettuna. Tämän kaltainen haavoittuvuus löytyi esimerkiksi PHP alustasta vuonna 2010, jolloin mahdollisten sessiotunnisteiden määrää voitiin vähentää käyttäen Brute-forcing -hyökkäystä. Nykyisin useimmat web-palvelimista kuitenkin pystyvät luomaan ennustamattomissa olevia istuntotunnisteita. (Scambray, Liu & Sima 2010, 151.)

Session Fixacation on istunnonhallintaan liittyvä hakkerien käyttämä tekniikka, jossa hyökkääjä valitsee käyttäjälle istuntotunnisteen. Tällöin hyökkääjän ei tarvitse yrittää arvata tai varastaa käyttäjän istuntotunnistetta. Hyökkääjä tarjoaa käyttäjälle oman sessiotunnisteensa esimerkiksi osana linkkiä, jolloin kyseinen käyttäjä kirjautuu web-sovellukseen käyttäen hyökkääjän istuntotunnistetta. Käyttäjän kirjauduttua sisään on hyökkääjän mah-

dollista kaapata istunto käyttäen samaa istuntotunnistetta. (Scambray, Liu & Sima 2010, 195.)

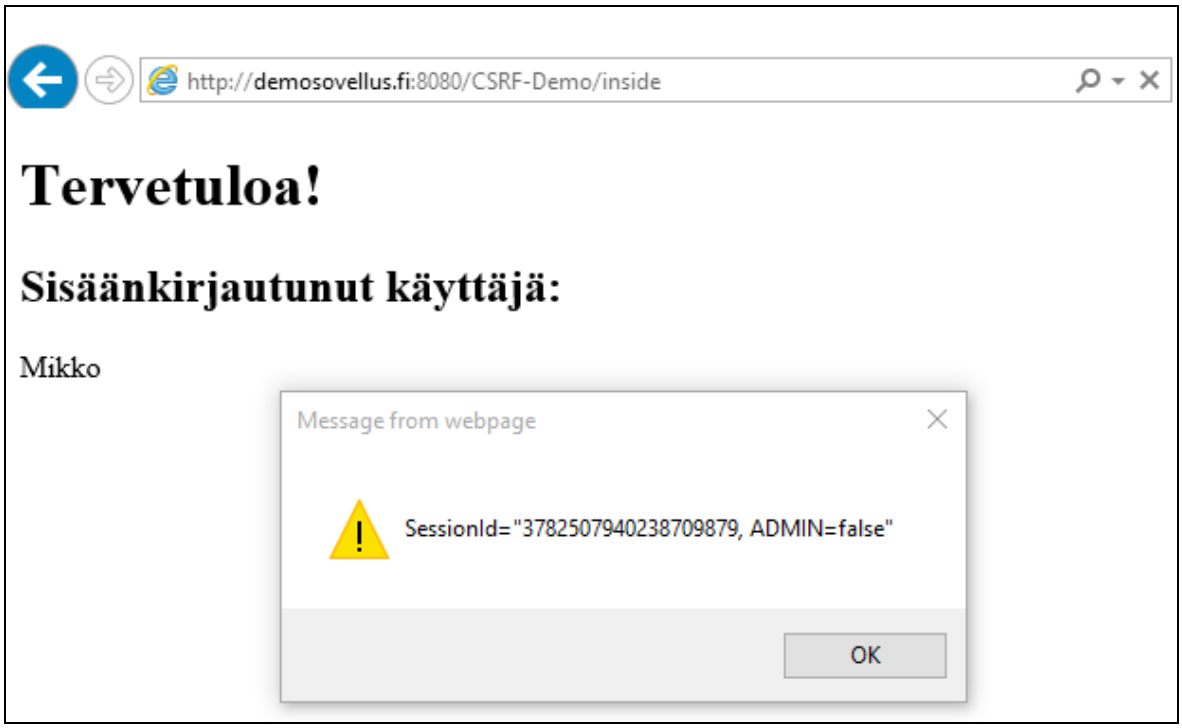
Session Fixacation -hyökkäyksen vastatoimena voidaan käyttää menettelyä, jossa kirjautumisen yhteydessä luodaan joka kerta uusi istuntotunniste. Lisäksi istunnoissa käytetään aikakatkaisua ja istunto asetetaan vanhenemaan määritellyn ajan mentyä umpeen. (Scambray, Liu & Sima 2010, 195.)

Web-sivustolla käytössä oleva ja käyttäjän tietoja sisältävä eväste saadaan esille esimerkiksi alla olevalla yksinkertaisella JavaScriptillä. Käyttäjän web-selaimen suorittama JavaScript näyttää ponnahdusikkunan, jossa näkyvät evästeelle tallennetut tiedot.

```
<script>alert(document.cookie);</script>
```

Esimerkkisovelluksessa JavaScript sijoitetaan käyttäjätunnuskenttään, varsinaisen käyttäjätunnuksen jälkeen. Selain suorittaa JavaScriptin, jonka jälkeen selain näyttää alla olevan kuvan kaltaisen ponnahdusikkunan, josta voidaan lukea kaikki evästeelle tallennetut käyttäjäkohtaiset tiedot. Esimerkissä hyödynnetään aikaisemmin luvussa 3.2 esiteltyä Cross-Site Scripting -tekniikkaa.

Alla olevan kuvan esimerkissä evästeen sisältö näytetään selkeyden vuoksi ponnahdusikkunana. Evästeen sisältämiä tietoja ovat ”SessionID” sekä tieto, onko käyttäjällä admintason oikeuksia web-sovellukseen ”ADMIN=false”.



Kuvio 17. Evästeen sisältämät tiedot ponnahdusikkunassa

Evästeen tiedot on kuitenkin myös mahdollista lähettää JavaScriptin avulla esimerkiksi sähköpostina hyökkääjälle, tai tallentaa ne hyökkääjän ylläpitämälle tai hallussa olevalle palvelimelle (Walker 2014, 215).

3.5 SQL-injektiot

SQL-injektiossa hyökkääjä syöttää web-sovelluksen lomakkeella olevaan kenttään SQL-kielisen kyselyn. SQL-injektiossa hyökätään web-sovelluksen suorittamaa SQL-kyselyä vastaan ja muokataan sovelluksen suorittamaa SQL-lausetta alkuperäisestä poikkeavaksi. Hyökkäys on mahdollista, mikäli sovelluksen suorittamaa SQL-kyselyä täydennetään dynaamisesti käyttäjän antamilla tiedoilla ilman syötteiden tarkastusta. (Scambray, Liu & Sima 2010, 238.)

SQL-injektio on yleisin niin sanotuista injektiohyökkäyksistä (Walker 2014, 217). Se löytyy myös OWASP:n tuottaman haavoittuvuuslistan kärjestä (OWASP 2016d). Sopivasti muotoillulla SQL-injektiolla voi olla mahdollista ohittaa kokonaan web-sovelluksen lomakkeen alkuperäinen käyttötarkoitus ja suorittaa tietokantapalvelimella hakkerin muokkaamia SQL-kyselyitä. (Walker 2014, 217.)

Onnistuneen SQL-injektion avulla voi olla mahdollista muokata ja poistaa web-sovelluksen käyttämän tietokannan tietoja, lukea arkaluonteisia tietoja tai suorittaa tietokantapalvelimen hallintaan liittyviä komentoja. Tietokantapalvelimen kautta voi olla myös mahdollista

antaa erilaisia komentoja tietokantapalvelimen käyttöjärjestelmälle. SQL-injektion vakavuuteen vaikuttaa osaltaan myös hakkerin saavuttamat käyttöoikeudet hyökkäyksen kohteena olevalle tietokantapalvelimelle. (OWASP 2016o.)

Seuraavassa esimerkkisovelluksessa kuvataan web-sovelluksessa oleva SQL-injektiohaavoittuvuus ja kyseisen haavoittuvuuden hyödyntäminen sisään kirjauduttaessa sovellukseen. Sovelluksen suorittama, lomakkeen syötteillä täydennetty SQL-lause on kokonaisuudessaan alla. Alla olevassa esimerkissä lomakkeen käyttäjätunnus- ja salasanakenttiin on syötetty arvot ”mikko” ja ”qwerty”. Kyseiset syötteet lisätään dynaamisesti osaksi suoritettavaa SQL-lausetta.

```
Dynaamisesti täydennettävä Java-koodi:
String sql = "select * from user where username='" + username + "' and
password='"+ password +"'";

Lomakkeen syötteillä täydennetty SQL-lause:
select * from user where username='mikko' and password='Qwerty'
```

Esimerkkisovelluksen kirjautumissivulla on lomake, jossa kysytään käyttäjätunnusta ja salasanaa. Hyökkäyksen toteutuksessa lomakkeen kenttiin syötetään alla olevat SQL-injektiot.

```
käyttäjätunnus: abc' or '1'='1
salasana: 123' or '1'='1
```

Tämän jälkeen painetaan kirjautumispainiketta, jolloin lomakkeen tiedot lähetetään palvelimelle ja edelleen osaksi suoritettavaa SQL-lausetta. Alla on SQL-injektioilla muokattu sovelluksen suorittama lause. Käytetty SQL-injektio lisää lauseeseen kaksi uutta ehtoa (OR 1=1), jolloin suoritettava lause on aina tosi huolimatta annetusta käyttäjätunnuksesta tai salasanasta.

```
Lomakkeen syötteillä täydennetty SQL-lause:
select * from user where username='abc' or '1'='1' and password='123'
or '1'='1'
```

Esimerkkisovellukseen kirjauduttaessa salasanakysely voidaan myös kokonaan kommentoida pois käyttäen käyttäjätunnuskentässä SQL-injektiota ”mikko' – ” (huomaa molemmat välilyönnit). Tällöin lomakkeelle syötetään ainoastaan tiedossa oleva käyttäjätunnus ja salasanakenttä jätetään tyhjäksi. Toisin sanoen salasanaa ei siis tarvitse tietää. Sovelluksen suorittama SQL-kysely on kokonaisuudessaan alla. Heittomerkki päättää SQL-lauseen ja kaksi perättäistä väliviivaa mitätöivät lauseen loppuosan, eikä tietokantapalvelin suorita sitä.

```
Dynaamisesti muodostettu SQL-lause kokonaisuudessaan:
select * from user where username='admin' -- ' and password=''

Tietokantapalvelimen suorittama SQL-lause:
select * from user where username='admin'
```

SQL-injektiohaavoittuvuudelta voidaan suojautua suodattamalla käyttäjän antamat syötteet ennen kuin ne lisätään osaksi SQL-kyselyä (Scambray, Liu & Sima 2010, 250). Syötteiden tarkistus on kuitenkin joissain tapauksissa mahdollista kiertää esimerkiksi käyttämällä merkkientiteettejä kiellettyjen merkkien sijasta. Toinen ja varmempi vaihtoehto on käyttää parametrisoituja kyselyjä. (OWASP 2016n.)

Java tarjoaa tähän tarkoitukseen PreparedStatement-luokan, joka parametrisoi käyttäjän antamat syötteet. Tällöin SQL-kyselyn muokkaaminen ei ole enää mahdollista vaan annetut syötteet lisätään kokonaisuudessaan osaksi SQL-lausetta. Alla olevassa esimerkissä näkyy SQL-injektion käyttämä SQL-lause käytettäessä PreparedStatement -luokkaa.

```
SELECT * FROM user WHERE username='abc\' or \'1\'=\'1' AND pass-
word='123\' or \'1\'=\'1'
```

Alla on kuvattuna haavoittuva Java-koodi sekä vaihtoehtoinen PreparedStatement-luokkaa soveltava versio samasta Java-koodista.

```
//SQL-injektion mahdollistava Java-koodi:
String sql = "SELECT * FROM users WHERE username ='"+ username + "'" +
" AND password='" + password + "'";
Statement stmt = connection.createStatement();
ResultSet rs = stmt.executeQuery(sql);

//SQL-injektion estävä Java-koodi:
PreparedStatement stmt = connection.prepareStatement("SELECT * FROM
users WHERE username=? AND password=?");
stmt.setString(1, username);
stmt.setString(2, password);
ResultSet rs = stmt.executeQuery();
```

3.6 Cross-Site Request Forgery (CSRF)

Cross-Site Request Forgery (CSRF/XSRF) on tekniikka, jota hyödyntämällä hyökkääjä saa haavoittuvaan web-sovellukseen kirjautuneen käyttäjän selaimen lähettämään sopivasti muokatun http-pyynnön kyseiseen web-sovellukseen. Haavoittuva web-sovellus ei osaa erottaa käyttäjältä tulevia http-pyyntöjä ja hakkerin muokkaamia http-pyyntöjä toisistaan. Tällöin hyökkääjän on mahdollista muokata http-pyyntö haluamaansa muotoon ja pakottaa käyttäjän selain lähettämään kyseinen pyyntö web-sovellukselle käyttäjän sitä huomaamatta. (OWASP 2016d.)

Mikäli käyttäjä on sisään kirjautuneena haavoittuvuuden sisältävään web-sovellukseen, riittää, että hakkerin sivustolla viitataan esimerkiksi html-koodin kuvaelementissä haavoittuvalle sivustolle. Toisin sanoen riittää, että käyttäjä samanaikaisesti vierailee hyökkääjän sivustolla. (OWASP 2016h.)

CSRF-hyökkäyksen avulla voidaan suorittaa erilaisia tilanvaihdoksia haavoittuvassa web-sovelluksessa, kuten käyttäjän profiilin muutokset, ostosten teko käyttäjän tunnuksilla, ulos- ja sisäänkirjautumiset. Sisään kirjautuminen voi olla mahdollista, mikäli käyttäjä on tallentanut selaimeen käyttäjätunnuksensa ja salasanansa. (OWASP 2016h.)

CSRF-hyökkäyksen vakavuus vaihtelee myös sen mukaan, mitkä ovat uhriksi joutuneen käyttäjän oikeudet kyseisessä web-sovelluksessa (OWASP 2016i). Kyseisestä CSRF-haavoittuvuudesta on kärsinyt esimerkiksi tilausvideopalvelu Netflix. Netflixin tapauksessa hyökkääjän oli mahdollista CSRF-hyökkäyksen avulla esimerkiksi lisätä elokuvia omaan

tiliinsä, muokata tilien tietoja sekä vaihtaa käyttäjätilin salasanaa. (Scambray, Liu & Sima 2010, 155.)

Hyökkääjä voi pakottaa käyttäjän tekemään http-pyynnön toiselle sivustolle esimerkiksi lataamalla kuvan web-sovellukseen. Alla olevassa esimerkissä käyttäjä pakotetaan yksinkertaisesti kirjautumaan ulos toiselta sivustolta. Hyökkääjän lisäämässä kuvassa on lähteenä ”src” suoritettavan CSRF-hyökkäyksen koodi. Alla on esimerkki CSRF-hyökkäyksen koodista kokonaisuudessaan.

```
<img src="http://esimerkkisovellus.fi/logout" width="0" height="0"
/>
```

Haavoittuvuuden käyttö web-sovelluksessa on mahdollista estää esimerkiksi evästeen uudelleenlähetyksen avulla. Web-sovelluksen tuottaman lomakkeen luonnin yhteydessä siihen lisätään käyttäjän istuntotunniste piilotettuna kenttänä. Web-sovellus vertaa lomakkeen käsittelyn yhteydessä http-pyynnössä olevaa istuntotunnistetta ja lomakkeen parametrina tullutta istuntotunnistetta toisiinsa. (Scambray, Liu & Sima 2010, 155.)

Toinen vaihtoehto estää CSRF-hyökkäykset on käyttää uudelleenkirjautumista esimerkiksi lomakkeen lähetyksen yhteydessä. Käyttäjä syöttää tällöin salasanansa, joka kerta uudelleen, kun tietoja lähetetään palvelimelle. (Scambray, Liu & Sima 2010, 156.)

On myös mahdollista käyttää CAPTCHA-tekniikkaa, jossa käyttäjä yritetään erottaa automatisoidusta hyökkäyksestä käänteisen Turingin testin avulla. CAPTCHA-tekniikassa kone pyritään erottamaan ihmisestä tekstin tai kuvan tunnistuksen avulla. Tämä tekniikka on kuitenkin, jossain tapauksissa mahdollista murtaa esimerkiksi käyttäen PWNtcha-dekooderia. (Scambray, Liu & Sima 2010, 158.)

Yleinen ja suositeltu tapa estää CSRF-hyökkäykset on kuitenkin sisällyttää yksilöllinen salattu tunniste (token) jokaiseen käyttäjän selaimesta lähettävään http-kutsuun. Token voidaan lisätä http-kutsuun esimerkiksi käyttämällä lomakkeella piilotettua kenttää, joka sisältää kyseisen tokenin eli automaattisesti generoidun ja salatun merkkijonon. (Scambray, Liu & Sima 2010, 156.)

Token voidaan luoda esimerkiksi Javan SecureRandom-luokalla, joka luo satunnaisen salakirjoitetun merkkijonon. Kyseinen token tallennetaan käyttäjän sessioon ja se luodaan

uudelleen aina kirjautumisen yhteydessä. Alla olevassa esimerkissä luodaan yksinkertainen token käyttäen SHA1-algoritmia. Luotu token tallennetaan käyttäjän sessioon.

```
//Tokenin luontimetodi
private String generateCSRFToken() throws NoSuchAlgorithmException{
        SecureRandom token = SecureRandom.getInstance("SHA1PRNG");
        return "" + token.nextLong();
}

//Luodaan token käyttäen yllä olevaa metodia ja tallennetaan se ses-
sion attribuutiksi
try {
           String token = generateCSRFToken();
           request.getSession().setAttribute("csrfToken", token);
} catch (NoSuchAlgorithmException e) {
           e.printStackTrace();
}
```

Tämä luotu sessio token lisätään jokaiselle käyttäjän saamalle lomakkeelle piilotettuna (hidden) kenttänä. Alla olevassa esimerkissä lomakkeella olevaan piilotettuun kenttään lisätään istunnon (session) token.

```
<input id="token" type="hidden" value="${sessionScope.csrfToken}"
/>
```

Tätä lomakkeelta http-pyynnön mukana tulevaa parametria verrataan sitten esimerkiksi post-metodissa käyttäjän sessiossa olevaan tokeniin, josta voidaan päätellä, onko kyseessä CSRF-hyökkäys ja tarvittaessa estää kyseinen hyökkäys.

3.7 Suora kohdeviittaus

Suora kohdeviittaus tarkoittaa esimerkiksi hakemistoon, tiedostoon tai tietokannan avaimeen viittaamista ohi web-sivuston pääsynvalvonnan. Hyökkääjä voi esimerkiksi muokkaamalla html-pyynnön parametria tai sivuston URL-osoitetta saada pääsyn web-sovelluksen osaan, jolle kyseisellä käyttäjällä ei ole oikeuksia. (OWASP 2016l.)

SQL-injektion lisäksi toinen mahdollinen tapa ohittaa tunnistusmenettely on suora sivuviittaus. Mikäli tunnistusmenettely on toteutettu siten, että tunnistautuminen vaaditaan aino-

astaan kirjautumissivulla, on mahdollista, että suoralla viittauksella sivulle voidaan ohittaa tunnistusmenettely kokonaan. (OWASP 2016f.)

Seuraavassa esimerkissä kuvataan tunnistusmenettelyn ohittaminen suoralla sivuviittauksella. Esimerkkisovelluksen käyttämä URL-osoite on kokonaisuudessaan alla.

```
http://demosovellus.fi:8080/Authentication-and-Session-Demo/
```

Sisäänkirjautuminen voidaan kuitenkin ohittaa viittaamalla suoraan kirjautumisen jälkeen avautuvalle tervetuloa-sivulle. Esimerkkisovelluksessa osoitteen perään voidaan lisätä ”inside.jsp”, joka on kyseisen yksittäisen sivun nimi. Tällöin sisäänkirjautumista ei vaadita. Alla olevassa kuvassa näkyy käyttäjäkohdassa ”null” eli käyttäjän sessiossa ei ole olemassa käyttäjätietoa.

Kuvio 18. Sisäänkirjautuminen ohitettu suoralla sivuviittauksella

3.8 Muut haavoittuvuudet

OWASP Top Ten -lista sisältää kymmenen haavoittuvuutta, joista osa on käsitelty aikaisemmissa kappaleissa. Seuraavaksi esitellään lyhyesti muut listalta löytyvät haavoittuvuudet.

Tietoturva-asetusten virheellinen määritys (Security Misconfiguration), on listan viides kohta. Siihen sisältyy palvelinten käyttöjärjestelmät, web-palvelimet, sovelluspalvelimet ja käytetyt ohjelmistokehykset. Puuttuvien tietoturvapäivitysten tai väärin määriteltyjen tietoturva-asetusten johdosta voi olla mahdollista päästä käsiksi esimerkiksi web-sovelluksen käyttämiin käyttäjätileihin, toiminnallisuuksiin tai tietoihin. OWASP suosittelee käyttämään haavoittuvuuksien kartoitukseen soveltuvia automatisoituja ohjelmistoja. (OWASP 2016q.)

Arkaluonteisen datan paljastuminen (Sensitive Data Exposure) sisältää virheet, jotka liittyvät arkaluonteisen tiedon tunnistamiseen, tiedon tallennuspaikkojen tuntemiseen, tiedon lähetyspaikkojen tuntemiseen ja virheisiin suojata kyseinen tieto sen kaikissa mahdollisissa sijainneissa. Tähän liittyy esimerkiksi luottokorttitietojen, terveystietojen tai taloudellisten tietojen tallentuminen palvelimen lokitiedostoihin. Palvelimen lokitiedostoihin saattaa päästä käsiksi ei luotettavia tahoja, jolloin arkaluonteista dataa saattaa paljastua. Tästä syystä kaikki web-sovelluksen sisältämä, tallentama ja lähettämä luottamuksellinen tieto täytyy tuntea ja sen säilytyspaikat täytyy olla tarkkaan tiedossa. Luottamuksellinen tieto täytyy suojata käyttäen riittävän vahvaa salausalgoritmia, salausavainten vaihtoon täytyy varautua sekä myös huolehtia avainten tietoturvasta. Lisäksi OWASP suosittelee käyttämään TLS-salausprotokollaa, jonka avulla voidaan suojata arkaluonteisen tiedon lähetys ja vastaanotto sekä HSTS turvallisuuspolitiikan käyttöönottoa, jonka avulla voidaan varmistaa HTTPS-yhteyden säilyminen palvelimen ja web-selaimen välillä. (OWASP 2016q.)

Puuttuva logiikkatason käyttäjäkontrolli (Missing Function Level Access Control) voi mahdollistaa pääsyn sellaisiin web-sovelluksen tietoihin, joihin käyttäjällä ei muuten olisi käyttöoikeuksia. Hyökkääjä voi tällöin suoran sivuviittauksen avulla, tai kutsumalla suoraan tiettyä web-sovelluksen toimintoa ohittaa tunnistusmenettelyn ja sen johdosta päästä käsiksi web-sovelluksen suojattuihin tietoihin. OWASP suosittelee tarkistamaan jokaisen käytössä olevan url-osoitteen ja parametrin tarjoama toiminnallisuus sekä estämään kokonaan luvattoman pääsyn tiettyihin sivutyyppeihin, kuten loki-, asetus- ja lähdekooditiedostoihin sekä tiedostotyyppeihin, jotka eivät ole sovelluksen kannalta välttämättömiä. (OWASP 2016q.)

Tunnettuja haavoittuvuuksia sisältävien komponenttien käyttö (Using Known Vulnerable Components) tarkoittaa sellaisten kehysten ja kirjastojen käyttöä, jotka tunnetusti sisältävät haavoittuvuuksia, joita voidaan hyödyntää käyttäen automatisoituja ohjelmistoja. Ohjelmistokehittäjät eivät aina tiedä kaikkien käyttämiensä komponenttien versiota, mikä osaltaan pahentaa tilannetta. OWASP suosittelee tarkistamaan säännöllisesti käytettyjen komponenttien versiot ja automatisoimaan prosessin. (OWASP 2016q.)

Tarkistamattomat uudelleenohjaukset ja edelleen lähetykset (Unvalidated Redirects and Forwards) tarkoittavat toimintoja, joissa web-sovellus ohjaa käyttäjän uuteen osoitteeseen ja osoitteen oikeellisuutta ei tarkisteta millään tavalla. Tällöin voi olla mahdollista, että hyökkääjä voi lähettää käyttäjälle web-sivustolle viittaavan linkin, joka uudelleenohjaa käyttäjän hakkerin määrittelemälle haitalliselle sivustolle. Lisäksi mikäli parametreja ei tarkisteta, voi olla mahdollista, että web-sovellus edelleen lähettää syötetyn parametrin

perusteella hyökkääjän ohi tunnistusmenettelyn. OWASP suosittelee kokonaan välttämään uudelleenohjauksia (redirects) ja edelleen lähetyksiä (forwards). Mikäli nämä ovat sovelluksen kannalta välttämättömiä, suositellaan käytettäväksi parametrien tarkistuksia tai palvelimessa tapahtuvaa sivujen mappausta. (OWASP 2016q.)

Alla on OWASP Top Ten -lista kokonaisuudessaan (OWASP 2016d).

- A1-Injektiot
- A2-Puutteellinen tunnistusmenettely ja istunnonhallinta
- A3-Cross-Site Scripting (XSS)
- A4-Turvaton suora kohdeviittaus
- A5-Tietoturva-asetusten virheellinen määritys
- A6-Arkaluonteisen datan paljastuminen
- A7-Puuttuva logiikkatason käyttäjäkontrolli
- A8-Cross-Site Request Forgery (CSRF)
- A9-Tunnettuja haavoittuvuuksia sisältävien komponenttien käyttö
- A10-Tarkistamattomat uudelleenohjaukset ja edelleen lähetykset

4 Pohdinta

Tämän tutkimuksen yhtenä tavoitteena oli selvittää, miten web-sovellusten tietoturvatestaus tapahtuu. Tietoturvatestaus voidaan kuitenkin määritellä eri tavoin ja sen varsinainen suoritus on aina riippuvainen testin suorittajasta. Tärkeimpiä huomioita oli mielestäni penetraatiotestauksen ensimmäisen vaiheen tärkeyden huomaaminen varsinaisen hakkerointiosuuden lisäksi. Kartoitus selvästikin tarjoaa pohjan koko penetraatiotestaukselle. Sen avulla saadaan kerättyä kohteen tiedot, jonka ansiosta voidaan seuraavissa vaiheissa valita oikeanlaiset hyökkäystekniikat.

Täysin tietoturvallisen web-sovelluksen toteutus on vaikeaa tai jopa mahdotonta. Tietoturva on jatkuvaa kilpajuoksua hakkereiden ja sovelluksia toteuttavien tahojen välillä. Eettinen hakkeri tai toisin ilmaistuna penetraatiotestaaja pyrkii omistajan pyynnöstä ja valtuutuksesta löytämään web-sovelluksen haavoittuvuudet jo ennen kuin hakkeri pääsee hyödyntämään niitä. Penetraatiotestaaja toimii kuitenkin kuten hakkeri, käyttäen samoja työkaluja. Valmiiden työkalujen lisäksi, penetraatiotestaajan täytyy osata luovasti soveltaa erilaisia tekniikoita ja toimintatapoja. Rikollisen hakkerin ei tarvitse noudattaa minkäänlaisia prosesseja, sääntöjä tai edes lakia, toisin kuin eettisen hakkerin. Tärkein ero rikollisen hakkerin ja eettisen hakkerin välillä on kuitenkin omistajalta saatu lupa.

Penetraatiotestauksessa hakkeroinnin laillisuus nousee usein esille. Penetraatiotestaajan täytyy ottaa huomioon, että käytettävä testausohjelmisto ei saa vahingossakaan skannata väärää kohdetta. Skannausohjelmistoon virheellisesti kirjoitetut IP-osoitteet voivat tehdä penetraatiotestaajasta hetkessä rikollisen hakkerin. Penetraatiotestaajan on tärkeä huomioida käyttäessään tässä opinnäytetyössä mainittuja penetraatiotestaajan tekniikoita ja työkaluja, että niiden kohteena olevan järjestelmän omistajalta on oltava lupa suorittaa kyseistä testausta. Suosittelenkin siis tietoliikenneyhteyksien katkaisua kokonaan esimerkiksi virtuaalikoneilla suoritettavan penetraatiotestauksen ajaksi.

Toinen kysymys oli, miten hakkerit hyödyntävät web-sovellusten haavoittuvuuksia, ja mikä on ylipäätään mahdollista. Tietoturvan kannalta puutteellisesti toteutettujen web-sovellusten hakkerointi voi olla joissakin tapauksissa todella helppoa. Tutkimus osoittaa, että siirtyminen get-metodin käytöstä post-metodin käyttöön ei yksinään estä käyttäjää muokkaamasta web-sovellukselle lomakkeella lähetettäviä parametreja. Proxy-palvelimen käyttö selaimen ja web-palvelimen välillä antaa käyttäjälle rajattomat mahdollisuudet muokata sitä tietoa, mikä selaimesta lopulta lähtee web-palvelimelle. Web-sovellusten ohjelmoijan täytyy ottaa tämä huomioon sovellusta kehittäessään. Käyttöliittymään asetetut tarkistukset voidaan siis ohittaa kokonaisuudessaan, jolloin ei voida käytännössä pu-

hua tietoturvaa lisäävästä ominaisuudesta. Myös Cross-Site Scripting haavoittuvuus osoittaa, että käyttäjän antamiin syötteisiin ei tulisi koskaan luottaa. On tärkeää ymmärtää, että käyttäjän syöttämä tieto ei ole luotettavaa tietoa. Tällaisen tiedon tarkistaminen ennen tiedon uudelleen tulostamista tai tallentamista, on olennainen osa ohjelmiston tietoturvaa. Useat opinnäytetyössä esitellyt haavoittuvuudet liittyvät myös toisiinsa. Esimerkiksi SQL-injektio tai suora kohdeviittaus voivat tarjota tavan ohittaa tunnistusmenettely osittain tai kokonaan. Myös Cross-Site Request Forgery -haavoittuvuus liittyy tunnistusmenettelyyn. Se voi antaa hyökkääjälle mahdollisuuden käyttää tunnistusmenettelyssä toisen käyttäjän tunnistetietoja. Puutteellisesti toteutettu istunnonhallinta voi myös mahdollistaa toisen käyttäjän tietojen hyödyntämisen tunnistusmenettelyssä. Koska hakkerin päämääränä on usein tunkeutua tietojärjestelmään ja ottaa se hallintaansa, monet esitellyistä haavoittuvuuksista liittyvätkin tunnistusmenettelyyn.

Olemassa olevien haavoittuvuuksien lähteenä käytettiin OWASP Top 10 -listaa. Lista tarjosi tietoa siitä, mitkä ovat tällä hetkellä web-sovellusten kaikkein yleisimmät haavoittuvuudet ja riskit. Vaikka tämä lista ei ole kaiken kattava, olisin mielelläni käsitellyt kaikki listalta löytyvät kohdat. Ottaen huomioon opinnäytetyön rajaus ja sen toteuttamiseen varattu aika, lista oli kuitenkin riittävän kattava.

Kolmantena tutkimuskysymyksenä oli, miten Java-koodissa voidaan estää haavoittuvuuksien syntyminen ja korjata löytyviä haavoittuvuuksia. Kysymyksen tutkiminen vaati haavoittuvien web-sovellusten penetraatiotestausta. Tarkoitukseen luodut Java-sovellukset toimivat penetraatiotestauksessa hyökkäysten kohteena. Esimerkkisovellukset täytyi ensin tehdä haavoittuviksi, jolloin oli mahdollista esitellä haavoittuvuuksien hyödyntämistä. Tämän jälkeen sovelluskoodiin tehtiin vaadittavat muutokset, jonka jälkeen vielä testattiin muutosten toimivuus. OWASP:in tarjoamista erilaisista internetlähteistä löytyi hyvin tietoa, jota soveltamalla esimerkkisovellusten haavoittuvuudet saatiin korjattua. Lisäksi käytetyt kirjallisuuslähteet tarjosivat yksityiskohtaista tietoa käytettävistä hyökkäystekniikoista.

Tutkimuksessa käytettyjen esimerkkisovellusten lähdekoodia ei luovuteta tämän opinnäytetyön liitteenä. Esimerkkisovellusten avulla toteutetut kokeet ovat kuitenkin toistettavissa. Tarvittaessa voidaan käyttää tarkoitukseen tehtyjä ilmaisia ohjelmistoja. Tällaisia tarkoituksella haavoittuviksi tehtyjä web-sovelluksia ovat esimerkiksi Damn Vulnerable Web Application (DVWA), joka on toteutettu PHP:llä (DVWA 2016). Lisäksi OWASP:in WebGoat on vastaava, mutta Javaan perustuva web-sovellus (OWASP 2016r).

Tämä opinnäytetyö keskittyy ainoastaan web-sovelluksiin ja sovelluskoodiin. Lisäksi on hyvä tietää, että esimerkiksi web-palvelin- tai sovelluspalvelinalustasta löytyvät haavoittu-

vuudet voivat mahdollistaa tunnistusmenettelyn murtamisen ja arkaluonteistentietojen paljastumisen. Lisäksi nämä haavoittuvuudet voivat tarjota pääsyn sovelluksen tietokantaan kokonaan ohi web-sovelluksen toimintalogiikan. Sosiaalisen hakkeroinnin mahdollisuudet on myös huomioitava. Mitä hyötyä esimerkiksi on rakentaa äärimmäisen tietoturvallinen tunnistusmenettely, jos käyttäjiltä voi soittamalla pyytää käyttäjätunnukset.

Opinnäytetyön teoria ja koodiesimerkit tarjoavat lukijalle perustiedot, joista on hyvä jatkaa web-sovellusten penetraatiotestauksen opiskelua. Tuloksien luotettavuutta arvioitaessa on huomioitava esimerkkien rajallisuus, ne eivät pyri esittämään täysin turvallista Java-kielistä ratkaisua, vaan toimivat ohjaavina esimerkkeinä. Jatkotutkimuksena olisi mielenkiintoista jatkaa listan läpikäyntiä myös palvelinalustojen osalta. Lisäksi työssä läpikäytyjä esimerkkejä voisi laajentaa ja koodiesimerkkejä voisi esitellä kokonaisuutena.

Kokonaisuutena opinnäytetyöprosessi oli onnistunut. Pysyin suunnitellussa aikataulussa ja mielestäni saavutin tavoitteet melko hyvin. Työn toteuttaminen vaati laaja-alaista perehtymistä asiaan. Opinnäytetyön tekeminen sisälsi tietoturvatestauksen opiskelua, web-ohjelmointia ja dokumentointia.

Oma kehittymiseni ohjelmoijana kasvoi tavoitteen mukaisesti tietoturvatestauksen osa-alueella. Lisäksi oli mielenkiintoista ohjelmoida tietoturvattomia sovelluksia, joita sai sitten yrittää hakkeroida. Oli myös mukavaa huomata, että aina löytyivät keinot estää haavoittuvuuden käyttö Java-koodissa.

Lähteet

DVWA 2016. DVWA - Damn Vulnerable Web Application. Luettavissa: http://www.dvwa.co.uk/. Luettu 11.5.2016.

Engebretson, P. 2013. The Basics of Hacking and Penetration Testing. Second Edition. Syngress Media. Waltham.

Erickson, J. 2007. Hacking: The Art of Exploitation. Second Edition. No Starch Press. San Francisco.

Scambray J., Liu V. & Sima C. 2010. Hacking Exposed Web Applications. Third Edition. McGraw-Hill Professional. New York.

OWASP 2016a. Category:Vulnerability - OWASP. Luettavissa: https://www.owasp.org/index.php/Category:Vulnerability. Luettu 9.3.2016.

OWASP 2016b. Category:Attach - OWASP. Luettavissa: https://www.owasp.org/index.php/Category:Attack. Luettu 14.3.2016.

OWASP 2016c. JSP JSTL - OWASP. Luettavissa: https://www.owasp.org/index.php/JSP_JSTL#Example:_Using_standard_actions_to_produce_xss. Luettu 14.3.2016.

OWASP 2016d. Top 10 2013-Top 10 - OWASP. Luettavissa: https://www.owasp.org/index.php/Top_10_2013-Top_10. Luettu 28.3.2016.

OWASP 2016e. Testing for Session Management Schema (OTG-SESS-001) - OWASP. Luettavissa: https://www.owasp.org/index.php/Testing_for_Session_Management_Schema_%28OWASP-SM-001%29. Luettu 12.4.2016.

OWASP 2016f. Testing for Bypassing Authentication Schema (OTG-AUTHN-004) - OWASP. Luettavissa: https://www.owasp.org/index.php/Testing_for_Bypassing_Authentication_Schema_%28OTG-AUTHN-004%29. Luettu 12.4.2016.

OWASP 2016g. OWASP Application Security Verification Standard 3.0 - OWASP. Ladattavissa: https://www.owasp.org/images/6/67/OWASPApplicationSecurityVerificationStandard3.0.pdf. Luettu 12.4.2016.

OWASP 2016h. Top 10 2013-A8-Cross-Site Request Forgery (CSRF) - OWASP. Luettavissa: https://www.owasp.org/index.php/Top_10_2013-A8-Cross-Site_Request_Forgery_%28CSRF%29. Luettu 16.4.2016.

OWASP 2016i. Cross-Site Request Forgery (CSRF) Prevention Cheat Sheet - OWASP. Luettavissa: https://www.owasp.org/index.php/Cross-Site_Request_Forgery_%28CSRF%29_Prevention_Cheat_Sheet#CSRF_Prevention_without_a_Synchronizer_Token. Luettu 16.4.2016.

OWASP 2016j. About The Open Web Application Security Project - OWASP. Luettavissa: https://www.owasp.org/index.php/About_OWASP. Luettu 19.4.2016.

OWASP 2016k. Category:OWASP Top Ten Project - OWASP. Luettavissa: https://www.owasp.org/index.php/Category:OWASP_Top_Ten_Project. Luettu 19.4.2016.

OWASP 2016l. Top 10 2013-A4-Insecure Direct Object References - OWASP. Luettavissa: https://www.owasp.org/index.php/Top_10_2013-A4-Insecure_Direct_Object_References. Luettu 19.4.2016.

OWASP 2016m. XSS Filter Evasion Cheat Sheet - OWASP. Luettavissa: https://www.owasp.org/index.php/Types_of_Cross-Site_Scripting. Luettu 3.5.2016.

OWASP 2016n. Types of Cross-Site Scripting - OWASP. Luettavissa: https://www.owasp.org/index.php/SQL_Injection. Luettu 3.5.2016.

OWASP 2016o. SQL Injection - OWASP. Luettavissa: https://www.owasp.org/index.php/SQL_Injection. Luettu 3.5.2016.

OWASP 2016p. Hashing Java - OWASP. Luettavissa: https://www.owasp.org/index.php/Hashing_Java. Luettu 7.5.2016.

OWASP 2016q. OWASP Top 10 2013 Presentation. Ladattavissa: http://owasptop10.googlecode.com/files/OWASP_Top-10_2013%20-%20Presentation.pptx. Luettu 9.5.2016.

OWASP 2016r. Category:OWASP WebGoat Project - OWASP. Luettavissa: https://www.owasp.org/index.php/Category:OWASP_WebGoat_Project. Luettu 11.5.2016.

OWASP 2016s. OWASP Testing Guide 4.0. Ladattavissa: https://www.owasp.org/images/5/52/OWASP_Testing_Guide_v4.pdf. Luettu 13.5.2016.

Walker, M. 2014. CEH Certified Ethical Hacker All-inOne Exam Guide. Second Edition. McGraw-Hill Professional. New York.