Santeri Kaasalainen

VERKKOSOVELLUSTEN YLEISIMMÄT HAAVOITTUVUUDET JA KÄYTÄNTEET NIIDEN EHKÄISEMISEKSI

JYVÄSKYLÄN YLIOPISTO
TIETOJENKÄSITTELYTIETEIDEN LAITOS
2018

TIIVISTELMÄ

Kaasalainen, Santeri
Verkkosovellusten yleisimmät haavoittuvuudet ja käytänteet niiden ehkäisemiseksi
Jyväskylä: Jyväskylän yliopisto, 2018, 43 s.
Tietojärjestelmätiede, kandidaatintutkielma
Ohjaaja: Siponen, Mikko

Verkkosovellukset ovat houkutteleva tapa tarjota kuluttajille palveluita, koska selainpohjainen sovellus takaa sen, että yksi ja sama sovellus toimii usealla eri alustalla riippumatta laitteesta tai käyttöjärjestelmästä. Verkkosovellukset ovat kuitenkin alttiimpia kyberhyökkäyksille, koska niiden verkkokäyttöliittymärajapinta on julkisesti avoin. Lisäksi verkkosovelluksien koostuminen suuresta määrästä vuorovaikutuksessa keskenään olevia teknologioita on johtanut tietoturvan toteutumisen hankaloitumiseen, koska kehittäjät joutuvat pitämään silmällä kunkin teknologian haavoittuvuuksille altistavia tekijöitä. Tämän vuoksi on tärkeää tutkia verkkosovellusten mahdollisia haavoittuvuuksia, syitä näihin sekä kuinka ne voidaan ehkäistä. Tutkielmassa käsiteltiin verkkosovellusten rakennetta siltä pohjalta, että kuinka eri teknologioiden läsnäolo johtaa haavoittuvuuksiin. Tutkielmassa käsiteltiin myös verkkosovellusten yleisimpiä haavoittuvuuksia ja kuinka näitä voidaan ehkäistä. Koska verkkosovelluksiin kohdistuvia haavoittuvuuksia on lukuisia, niitä kaikkia on mahdotonta käsitellä tämän tutkielman puitteissa. Siksi tässä tutkielmassa käsitellyt haavoittuvuudet pohjattiin OWASP:n vuoden 2013 top 10 listaukseen verkkosovelluksiin kohdistuvista haavoittuvuuksista soveltuvilta osin. Kirjallisuuskatsauksen perusteella löydettiin verkkosovellusten haavoittuvuuksien syille neljä yläkategoriaa: puutteellinen syötteiden varmistus, puutteellinen istuntotunnisteen hallinta, puutteet verkkosovelluksen loogisessa rakenteessa ja puutteellinen verkkosovelluksen alustan konfiguraatio. Kirjallisuuskatsauksen perusteella, tutkielmassa esitettiin myös kehittämisvaiheen käytänteet, joiden avulla edellä mainituista syistä johtuvat haavoittuvuudet vältetään.

Asiasanat: tietoturva, verkkosovellus, haavoittuvuus

ABSTRACT

Kaasalainen, Santeri
The Most Common Web Application Vulnerabilities and How to Prevent Them
Jyväskylä: University of Jyväskylä, 2018, 43 s.
Information Systems, Bachelor's Thesis
Supervisor: Siponen, Mikko

Web applications have become a tempting way to provide services for customers because using an application via web browser provides a way to run it no matter what device or operating system user is using. However, web applications are more prone to cyber-attacks because they accessible through their web user interface. In addition, web applications constitute of many different technologies that are in interaction with each other meaning that developers need to keep an eye on vulnerabilities that are due to using each of technologies. That is a reason for doing research of web application vulnerabilities: what are the reasons that lead to their existence and what are the ways to prevent them. In this paper, web application architecture and how structure of large amount of different technologies leads to existence of vulnerabilities were surveyed. Also the most common web application vulnerabilities and prevention technics were surveyed. Because there exists large amount of different web application vulnerabilities it is impossible to have them all present in this research. That's why vulnerability chapter in this paper have been limited to base on of applicable part of OWASP's top 10 list of the most dangerous web application vulnerabilities from 2013. Four main categories were found to be causes of web application vulnerabilities: lack of input validation, poor session management, shortcoming in application's logical structure and security misconfiguration. Based on literature review, practices to prevent vulnerabilities that occur because those reasons were also presented.

Keywords: information security, web application, vulnerability

KUVIOT

TAULUKOT

SISÄLLYS

1 JOHDANTO

World Wide Webin alkuaikoina Internet oli hyvin erilainen paikka, kuin mitä se on nykyään. Se koostui kokonaan staattisista HTML- dokumenteista ja yksittäinen URL-osoite tarjosi saman näkymän jokaiselle verkkoselaajalle. Nykyään tilanne on kuitenkin aivan toisenlainen. Suurin osa nykyajan verkkosivustoista on enemmänkin verkkoselaimella käytettäviä verkkosovelluksia, jotka tarjoavat dynaamisesti mukautuvaa sisältöä käyttäjästä ja käyttäjän toimenpiteistä riippuen. Verkkosovellukset ovat houkutteleva tapa tarjota kuluttajille palveluita, koska selainpohjainen sovellus takaa sen, että yksi ja sama sovellus toimii usealla eri alustalla riippumatta laitteesta tai käyttöjärjestelmästä. Tämä vähentää palveluiden tarjoamisen kustannuksia verrattaessa perinteisiin työpöytäsovelluksiin. Verkkosovellukset ovat kuitenkin alttiimpia kyberhyökkäyksille niiden kaikkien saavutettavissa olevan julkisen verkkokäyttöliittymärajapinnan vuoksi. (Rafique ym., 2015.) Lisäksi dynaamisten sovellusten rakentaminen selaimella käytettäväksi on tietoturvan osalta haastava tehtävä, koska käyttäjät joutuvat kiinnittämään huomioita useiden eri teknologioiden käytöstä aiheutuviin haavoittuvuuksiin. (Stuttard & Pinto, 2011.) Vaadittu tietoturvan tason voidaan kuitenkin saavuttaa käyttämällä sovelluskehityksessä tietoturvan kannalta oikeaksi todettuja käytänteitä. Tämän tutkielman tarkoituksena on tarjota yleiskuva verkkosovelluksia kohtaavista haavoittuvuuksista ja näiden syistä sekä esittää keinot näiden ehkäisemiseksi.

Verkkosovellusten tietoturvaa ja haavoittuvuuksia on tutkittu paljon. Tutkielmien aihepiirit käsittelevät yleisesti jotakin tiettyä haavoittuvuutta ja esittävät metodin tämän haavoittuvuuden ehkäisemiseksi. Tämä tutkielma kokoaa yhteen havaintoja useista tieteellisistä artikkeleista.

Ensimmäinen käsittelyluku *verkkosovelluksen rakenne* avaa verkkosovelluksen rakennetta siltä osin, että haavoittuvuuksien syntymiseen johtavat tekijät on helpompi ymmärtää. Verkkosovellusten rakenne koostuu suuresta määrästä eri teknologioita ja tämä tekee sovelluksista tietoturvanäkökulmasta katsottuna haastavia, koska kehittäjät joutuvat ottamaan huomioon kunkin teknologian haavoittuvuuksille altistavat tekijät. Eri teknologioiden määrä on kasvanut suureksi, koska WWW:tä ei suunniteltu alun perin tukemaan nykyaikaisia ominaisuuksiltaan rikkaita verkkosovelluksia ja nykyaikaisiin vaatimuksiin vastaami-

nen on johtanut uusien teknologioiden lisäämiseen vanhaan WWW-standardiin. Lisäksi World Wide Webin alkuajan vaatimuksiin kehitettyjä teknologioita ei suunniteltu toimimaan osana nykyaikaista dynaamista verkkosovellusta mikä on johtanut siihen, että niihin on jouduttu kehittämään lisäominaisuuksia. (Stuttard & Pinto, 2011.) Suuri lukumäärä vuorovaikutuksessa keskenään olevia teknologioita on johtanut tietoturvan toteutumisen hankaloitumiseen, koska kehittäjät joutuvat pitämään silmällä kunkin teknologian haavoittuvuuksille altistavia tekijöitä.

Toinen käsittelyluku, *verkkosovellusten yleisimmät haavoittuvuudet,* selostaa verkkosovelluksia koskevia haavoittuvuuksia ja hyökkäysmenetelmiä. Lin ja Xuen (2014) mukaan verkkosovellusten haavoittuvuudet voidaan jakaa kolmeen tekijään: puutteisiin syötteiden ja istuntotunnisteen käsittelyssä sekä loogiseen virheeseen verkkosovelluksen rakenteessa. Lisäksi OWASP:n vuoden 2013 listauksessa (2013) verkkosovelluksia uhkaavista kymmenestä vaarallisimmista haavoittuvuuksista voidaan erottaa vielä yksi tekijä: verkkosovelluksen alustan puutteellinen konfigurointi. Tässä tutkielmassa käydään läpi jokaista haavoittuvuuskategoriaa siten, että esitetään tapa kuinka haavoittuvuudet realisoituvat ja keinot joidenka avulla hyökkääjä kykenee käyttämään näitä hyväksi.

Kolmas käsittelyluku, *käytänteet haavoittuvuuksien ehkäisemiseksi*, esittää keinot verkkosovellusten turvallisuuden takaamiseen. Kolmannessa luvussa esitellyt tietoturvan varmistavat toimenpiteet pyrkivät ehkäisemään toisessa luvussa esiteltyjä haavoittuvuuksia. Turvallisuuden varmistavat keinoja on käsitelty puhtaasti ohjelmoinnin ja teknisen kehittämisen näkökulmasta ja tutkielmassa ei oteta kantaa esimerkiksi projektivaiheen riskianalyysiin tai vaatimusmäärittelyyn. Tässä tutkielmassa ei myöskään perehdytä verkkosovelluksiin jälkeenpäin liitettäviin ohjelmistoihin joiden tarkoitus on parantaa sovelluksen tietoturvaa.

Tämä tutkielma on luotu kirjallisuuskatsauksena ja materiaalin etsinnässä on käytetty hyväksi verkosta löytyviä tieteellisten artikkelien arkistoja. Lisäksi materiaalia on etsitty myös ohjelmistoteknologiaan ja tietoturvaan liittyviltä verkkosivustoilta. Yleisimpiä hakusanoja ovat olleet *web application vulnerabilities*, *input validation*, *session management*, *access control*, *security misconfiguration* ja *web application testing*.

Tämän tutkielman tarkoituksena on vastata kirjallisuuskatsauksen perusteella seuraaviin tutkimuskysymyksiin: *Mitkä ovat verkkosovelluksia kohtaavat yleisimmät haavoittuvuudet ja mitkä tekijät niihin johtavat? Kuinka nämä haavoittuvuudet kyetään estämään?* Kirjallisuuskatsauksen perusteella verkkosovellusten tekniset haavoittuvuudet kyettiin kategorisoimaan neljään eri ydin tekijään: syötteiden puutteellinen käsittely, istuntotunnisteen puutteellinen hallinta, puutteet verkkosovelluksen loogisessa rakenteessa sekä verkkosovelluksen alustan puutteellinen konfiguraatio. Kirjallisuuskatsauksen perusteella löydettiin myös keinot ja ohjeet ehkäistä edellä mainituista tekijöistä johtuvia haavoittuvuuksia. Tähän tutkielmaan on kirjallisuuskatsauksen perusteella kerätty käytänteet, joita noudattamalla yleisimpiä haavoittuvuuksia voidaan ehkäistä. Ohjeistuksen avulla eritoten aloittelevat verkkosovelluskehittäjät kykenevät vähentämään haavoittuvuuksien riskiä ollessaan osana verkkosovellusprojektia.

2 VERKKOSOVELLUKSEN RAKENNE

Shklarin ja Rosen (2003) määritelmä verkkosovelluksista on, että ne ovat enemmän kuin pelkkiä verkkosivuja. Ne ovat asiakas-palvelin sovelluksia, jotka tarjoavat interaktiivisia palveluita. Tietoa siirretään käyttäjältä palvelimelle ja päinvastoin. Yksinkertaistettuna voidaan ajatella, että verkkosivustot tarjoavat staattisista ja muokkautumatonta sisältöä, kun taas verkkosovellusten tarjoama sisältö muokkautuu dynaamisesti käyttäjän ja hänen toimintansa mukaan.

Tässä luvussa käsitellään verkkosovellusten rakennetta siltä osin, että myöhemmin esitettävät haavoittuvuudet ja oikeanlaiset käytänteet on helpompi ymmärtää. Verkkosovellukset ovat hajautettuja järjestelmiä ja ne koostuvat suuresta määrästä erilaisia teknologioita. Teknologioiden suuri lukumäärä on johtanut sovellusten kompleksisuuteen ja täten niiden turvallinen toteutus on monimutkaista, koska kehittäjät joutuvat ottamaan huomioon useita eri tekijöitä, jotka saattavat johtaa sovelluksen turvallisuuden vaarantumiseen. Suuri osasyyllinen haavoittuvuuksien ilmentymiseen on se, että nykyaikaisissakin verkkosovelluksissa käytetään teknologioita, joita ei alun perin suunniteltu verkkosovellusten nykyaikaiseen käyttöympäristöön. Esimerkiksi nykyään verkkomaailmassa suuressa osassa olevat evästeet ovat vain tapa kiertää http-protokollan tilattomuus. (Stuttard & Pinto, 2008.) Toinen esimerkki WWW:n käyttövaatimusten kasvamisesta ulos sen suunnitellusta käyttötarkoituksesta on selainnäkymän dynaaminen toiminnallisuus. Verkkosovellusten dynaaminen sisältö tuotetaan JavaScriptin avulla ja tämä on johtanut perinteistä työpöytäsovelluksista poikkeaviin hyökkäystapoihin, kuten esimerkiksi 3. luvussa esiteltävään XSS:ään. (Shar & Tan, 2012.)

2.1 Verkkosovelluksen asiakas-palvelin-arkkitehtuuri

Verkkosovellukset ovat hajautettuja järjestelmiä. Hajautettu järjestelmä voidaan nähdä järjestelmäkokonaisuudeksi, joka koostuu useista eri laitteista ja sovelluksista, joilla jokaisella on oma tehtävänsä. Yksinkertaistettuna verkkosovellus tarvitsee toimiakseen kaksi eri järjestelmää: asiakas- ja palvelinjärjestelmän.

Verkkosovelluksissa asiakasjärjestelmä on laite, joka verkkoselaimen avulla suorittaa verkkosovelluksen asiakasohjelmaa ja tämän avulla käyttäjä saa käyttöönsä verkkosovelluksen käyttöliittymän. Palvelinjärjestelmä, on alusta, jolla ajetaan verkkosovelluksen palvelimella suoritettavaa osaa. (Shklarin & Rosen, 2003.) Alusta voi olla fyysinen tai virtuaalinen tietokone tai pohjautua konttiteknologiaan.

2.1.1 Asiakasohjelma (Client-side)

Asiakasohjelma on verkkosovelluksen osa, joka suoritetaan käyttäjän verkkoselaimessa. Asiakassovelluksen osuus kokonaissovelluksesta on tuottaa verkkosovelluksen käyttöliittymä sekä siihen liittyvä toiminnollisuus. Asiakassovellus koostuu pääpiirteittäin kolmesta eri teknologiasta:

- HTML
- JavaScript
- CSS

HyperText Markup Language (HTML) on verkkosivustojen käyttämä merkkauskieli. HTML:n avulla merkitään verkkosivuston sisältö ja sen avulla opastetaan verkkoselainta näyttämään sisältö oikeassa muodossa ja järjestyksessä. HTML-dokumentit koostuvat elementeistä, jotka määrittävät sivustolla esitettävän sisällön merkityksen. Esimerkiksi ”<p>” elementti kertoo verkkoselaimelle, että sen sisällä oleva sisältö on merkitykseltään tekstikappale. (Mozilla Foundation, 2016c.) Kaikkiaan HTML:n uusimassa HTML5-versiossa on käytössä 142 erilaista merkitystä edustavaa HTML-elementtiä. (W3C, 2016.) Tämän tutkielman kannalta ja ylipäätään verkkosovellusten tietoturvan osalta, tärkeimmät elementit ovat script, input ja form. Script-elementti ohjeistaa verkkoselainta käsittelemään sen sisällä olevaa tekstiä JavaScriptinä. Input-elementti tarkoittaa, että verkkoselaimen tulee piirtää syötekenttä. Form-elementti puolestaan tarkoittaa, että kaikkien sen sisällä olevien syötekenttien data tullaan lähettämään palvelinsovellukselle yhdessä. Lisäksi seuraavan verkkosovellusten yleisimpiä haavoittuvuuksia käsittelevän luvun pohjalta voidaan huomata, että verkkosovellusten käyttöliittymien rakenteen pohjautuminen HTML:ään ja HTML-dokumentin mukautuvuus käyttäjän syötteiden pohjalta on haastavaa tietoturvan kannalta.

JavaScript on verkkoselaimen ymmärtämä komentosarjakieli, jonka avulla verkkosivustoille voidaan luoda dynaamista sisältöä. Dynaaminen sisältö tarkoittaa, että kun HTML-dokumentin muodostaman näkymän halutaan muokkautuvan, verkkoselain ei nouda palvelimelta uutta muutettua HTML-dokumenttia vaan JavaScript muokkaa jo näkyvillä olevaa. JavaScriptin avulla kehittäjät voivat määrittää verkkosivun dynaamisen mukautuvuuden käyttäjän toimenpiteiden mukaan. JavaScriptin avulla voidaan esimerkiksi tehdä verkkosivustolle toiminnallisuus, joka tarkistaa käyttäjän antaman syötteen oikeellisuutta jo kirjoitusvaiheessa. JavaScript linkitetään esillä olevaan HTML-dokumenttiin, joko erikseen JavaScript-tiedostona tai se merkitään HTML-dokumenttiin script-elementein. (Mozilla Foundation, 2016a.) Lisäksi JavaScript

mahdollistaa lähes reaaliaikaisen kommunikaation palvelimen ja selaimen välillä. JavaScriptin avulla verkkosivustoihin kyetään liittämään toiminnallisuutta ja sen avulla niistä voidaan tehdä interaktiivisia hyötysovelluksia. Seuraavassa luvun pohjalta, joka käsittelee verkkosovellusten haavoittuvuuksia, voidaan kuitenkin todeta, että dynaamisuuden lisääminen verkkosovelluksiin ja niiden vahva pohjautuminen JavaScriptiin on tehnyt ne myös alttiiksi erilaisille haavoittuvuuksille.

Cascading Style Sheets (CSS) on World Wide Webin käyttämä tyylikieli. Se määrittää, millaisena esitettävänä oleva HTML-dokumentti näytetään. Tyylitiedoston mukaan voidaan määrittää HTML-dokumentin elementtien ulkoinen esitystapa, kuten esimerkiksi käyttöliittymän fontit ja värit. (Lie, Bos & Lilley, 1998). Koska verkkoselain lähettää CSS:n määritysten mukaisesti http-pyyntöjä myös CSS:ää voidaan käyttää hyödyksi verkkosovelluksiin hyökätessä (OWASP, 2016).

2.1.2 Palvelinpuolenjärjestelmä (Server-side)

Verkkosovelluksen toinen ydin osa palvelinpuolenjärjestelmä ja se on keskitetysti kaikkien sitä käyttävien asiakasohjelmien yhteisessä käytössä ja tarjoaa asiakasohjelmille verkkosovelluksen tarkoituksen mukaisia palveluita. Useimmiten palvelinpuolella toimii useita sovelluksia, jotka tarjoavat kokonaisuudelle omia palveluitaan, kuten esimerkiksi autentikaatiota, tiedon tallennusta tai välimuistinhallintaa. Haavoittuvuuksien käsittelyn helpottamiseksi, tässä tutkielmassa palvelinpuolenjärjestelmä käsitetään kuitenkin yhdeksi ohjelmistokokonaisuudeksi, joka käsittää kaikki verkkosovelluksen ylläpitäjän palvelimilla toimivat keskenään verkottuneet ohjelmat, jotka ovat osana verkkosovellusta.

Palvelinpuolenjärjestelmällä on lukuisia tehtäviä, jotka vaihtelevat sen mukaan mikä on verkkosovelluksen tarkoitus. Esimerkiksi verkkokaupassa tehtäviä olisi tallentaa tietokantaan tieto siitä, että asiakas on tilannut tuotteen tietyllä hinnalla samalla varmistaen, että asiakkaat eivät voi ostaa tuotteita, jos niitä ei ole varastossa, kun taas esimerkiksi suoratoistopalvelun kohdalla palvelinpuolenjärjestelmän tehtävänä on tarjota käyttäjän verkkoselaimelle haluttu videotiedosto samalla varmistaen, että käyttäjä on kirjautunut sisään ja maksanut kuukausimaksun. Molemmat yllä olevat tapaukset ovat yksinkertaisia esimerkkejä palvelinpuolenjärjestelmän sovelluskohtaisista tehtävistä, jotka ovat järjestelmän tarkoituksenmukaista toimintaa. Tietoturvanäkökulmasta palvelinpuolenjärjestelmän tehtävänä on tarjota asiakasohjelmalle verkkosovelluksen tarkoituksen mukaisia palveluita samalla varmistaen, että käyttäjä ei pääse käsiksi sisältöön, joka ei hänelle ole tarkoitettu. (Shklar & Rosen, 2003.)

2.2 Asiakkaan ja palvelimen välinen kommunikointi

Tässä alaluvussa käsitellään asiakas- ja palvelinsovelluksen välistä tiedonsiirtomenettelyä siltä osin, kuin se on merkityksellistä verkkosovellusten yleisimpien tietoturvahaavoittuvuuksien osalta.

2.2.1 Http-protokolla

Http (Hypertext Transfer Protocol) on WWW:ssä yleisesti käytetty protokolla, jonka avulla verkkoselain ja palvelin kommunikoivat toistensa kanssa. Protokolla on ollut käytössä WWW:n alusta asti ja se kehitettiin alun perin staattisten dokumenttien noutamiseen. Tämän vuoksi siihen on joutunut rakentamaan lisäominaisuuksia, jotta sitä voitaisiin käyttää nykyaikaisissa verkkosovelluksissa. Verkkosovellusturvallisuuden näkökulmasta tärkein myöhemmin lisätty ominaisuus on evästeet ja ne on jouduttu ottamaan käyttöön, koska http on ns. tilaton protokolla tarkoittaen, että jokaista lähetettyä pyyntöä ja vastausta kohdellaan itsenäisenä ja pyynnöt eivät ole kytköksissä toisiinsa. Tämä tarkoittaa, että palvelin ei kykene käyttäjää tunnistamaan pelkästään http-pyyntöjen perusteella, vaan käyttäjän yksilöivää tietoa joudutaan välittämään evästeiden avulla. (Stuttard & Pinto, 2008.)

Http käyttää viestipohjaista mallia, jossa asiakas lähettää palvelimelle pyynnön (request) ja palvelin vastaa tähän palauttamalla http-vastauksen (response). Sekä pyynnöt että vastaukset ovat http-protokollan mukaisia viestejä, jotka koostuvat viestikentistä (message headers) jotka sisältävät tietoa kommunikoivien osapuolien välillä. (Fielding.ym, 1999.)

Http-pyynnöt ovat protokollan mukaisia viestejä, joidenka avulla asiakasohjelma välittää tietoa palvelinpuolenjärjestelmälle. Asiakasohjelman lähettämät pyynnöt sisältävät aina jonkin protokollan mukaisen metodin, joka määrittää asiakkaan lähettämän http-pyynnön merkityksen. Verkkosovelluksien haavoittuvuuksien näkökulmasta merkittävimmät metodit ovat GET ja POST. (Stuttard & Pinto, 2008.)

GET-pyynnöllä tarkoitetaan menettelyä, jossa asiakassovellus siirtää parametrit palvelinsovellukselle, niin että parametrit kulkeutuvat palvelimelle URL-osoitteen mukana. Ohjesääntö GET-pyynnölle on se, että sitä tulisi käyttää silloin, kun halutaan palvelimelta noutaa yksittäinen HTML-dokumentti ja tarkoitus ei ole kirjoittaa tietokantaan. (Fielding.ym, 1999.) Esimerkiksi linkin klikkaaminen saa aikaan http-pyynnön käyttäen GET-metodia.

POST-pyynnössä asiakassovelluksen lähettämät parametrit siirretään palvelimelle lähetettävän http-pyynnön viestikentissä. Ohjesääntönä on, että POST-kutsua tulisi kehittäjien käyttää silloin, kun tarkoituksena on kirjoittaa uutta tietoa tietokantaan, päivittää vanhaa tietoa tai lähetettävä syöte sisältää arkaluontoista tietoa, kuten esimerkiksi salasanan. (Fielding.ym, 1999.)

Alla olevasta kuvasta voidaan havainnoida POST- ja GET-metodin olennainen ero. Kyseiset http-kutsut lähettävät samat parametrit palvelinpuolenjärjestelmälle, mutta eri metodia käyttäen. GET-metodissa käyttäjän antamat syötteet lähetetään osana haettua URL-osoitetta. POST-metodissa puolestaan parametrit välitetään palvelinsovellukselle http-kutsun viestikentässä.

Kuvio 1 POST- ja GET- metodin eroavaisuus.

2.2.2 Evästeet ja istunnon käsittely

Http-protokollan tilattomuuden vuoksi kehitettiin evästeet (engl. cookies), jotka ovat palvelimen http-vastauksessaan lähettämiä avain-arvo pareja, jotka tallennetaan käyttäjän verkkoselaimeen. Kun eväste on tallennettu käyttäjän verkkoselaimeen, jokainen kutsu, joka lähetetään samalle palvelimelle, josta eväste on saatu, sisältää myöskin kyseiset evästeet. (Shklar & Rosen, 2003.)

Verkkosovelluksissa käyttäjän tunnistamiseen käytetään usein evästettä, joka sisältää käyttäjän yksilöivän istuntotunnisteen (engl. session identifier). Istuntotunniste on palvelimen luoma vaikeasti sattumanvaraisesti löydettävä merkkijono, jonka avulla palvelinpuolenjärjestelmä ymmärtää, että kuka http-pyynnön lähettäjänä on. (Shklar & Rosen, 2003.) Tämän tutkielman seuraavan luvun pohjalta voidaan kuitenkin huomata, että käyttäjien yksilöimen istuntotunnisteella tuottaa verkkosovelluskehittäjille haasteita tietoturvan näkökulmasta. Istuntotunnisteen päätyminen hyökkääjän käsiin johtaa siihen, että hyökkääjä kykenee esiintymään käyttäjänä, jolta hän sai istuntotunnisteen käyttöönsä.

Kuvio 2 Istuntotunnisteen sisältävän evästeen asettaminen

2.3 Verkkosovelluksen alustakomponentit

Tämän tutkielman aihepiirin osalta tärkeitä verkkosovellusten alustakomponentteja ovat verkkopalvelin ja siinä toimiva palvelinohjelmisto sekä sovelluskehys.

Verkkopalvelin on Internettiin kytketty tietokone, jossa suoritetaan verkkosovelluksen palvelinpuolenohjelmia. Verkkopalvelimen käyttöjärjestelmä suorittaa laitetekniset operaatiot, kuten esimerkiksi verkkosovelluksen käyttämän datan kirjoittamisen pysyvästi säilyväksi kiintolevylle. Se myös tarjoaa palvelinpuolenohjelman käyttöön yleisesti käyttöjärjestelmien tarjoamia palveluita, kuten hakemistorakenteen jonka avulla verkkosovellus saa käyttöönsä tarvitsemiaan tiedostoja. (Mills, 2016.)

Verkkopalvelimella toimivan *verkkopalvelinohjelmiston* tehtävänä on tulkita saapuneita http-pyyntöjä, ohjata http-pyynnön mukana tuleva data, kuten evästetiedot ja käyttäjän syötteet, prosessoitavaksi oikealle resurssille verkkopalvelimella ja tämän jälkeen palauttaa http-vastaus pyynnön lähettäneelle käyttäjälle. Lisäksi verkkopalvelinohjelmistossa toimii verkkosovelluksien käyttämiä lisäpalveluita, kuten esimerkiksi salaus. (Mills, 2016.)

Sovelluskehykset ovat yleisesti verkkosovelluskehittämisessä käytettäviä apuvälineitä. Ne koostuvat ohjelmakirjastoista, joihin on jo valmiiksi ohjelmoitu useita verkkosovelluksen käyttämiä toiminnollisuuksia. Sovelluskehykset helpottavat verkkosovelluskehittäjän työtä, koska hänen ei tarvitse ohjelmoida verkkosovellusta alusta asti vaan yleisesti käytetyt toiminnollisuudet ovat jo valmiina. (DocForge, 2014.)

Tämän tutkielman seuraavan luvun pohjalta voidaan todeta, että verkkosovelluksien käyttämien alustakomponenttien puutteellinen konfigurointi verkkosovelluksen ollessa tuotantokäytössä johtaa haavoittuvuuksien ilmenemiseen verkkosovelluksessa.

3 VERKKOSOVELLUSTEN YLEISIMMÄT HAAVOITTUVUUDET

Verkkosovellusten yleisuus on kasvanut suuresti muutaman vuosikymmenen aikana. Ne antavat miljardeille ihmisille mahdollisuuden käyttää esimerkiksi verkkopankkipalveluita tai kommunikoida toisten ihmisten kanssa verkon välityksellä käytössä olevasta päätelaitteesta välittämättä. Kuitenkin nämä mahdollisuudet ovat tuoneet mukanaan tietoturvaongelmia ja haasteita. Verkkosovellukset ovat alttiimpia kyberhyökkäyksille niiden kaikkien saavutettavissa olevan julkisen verkkokäyttöliittymärajapinnan vuoksi. (Rafique ym., 2015.) Lisäksi suuri lukumäärä erilaisia teknologioita tekevät verkkosovelluksesta monimutkaisen tuottaen kehittäjille haasteita tietoturvan osalta.

Tietoturvan osalta on tärkeää, että verkkosovellukset sisältävät mahdollisimman vähän haavoittuvuuksia. Verkkosovelluksien tapauksessa haavoittuvuus tarkoittaa järjestelmän kykenemättömyyttä vastata hyökkääjän pahantahtoisiin toimenpiteisiin. Järjestelmässä on haavoittuvuus, kun järjestelmä ei kykene estämään ei haluttuja toimenpiteitä. (Grobauer, Walloschek & Stocker, 2011.)

Lin ja Xuen (2014) mukaan verkkosovellusten haavoittuvuudet voidaan jakaa kolmeen tekijään: puutteisiin syötteiden ja istuntotunnisteen käsittelyssä sekä loogiseen virheeseen verkkosovelluksen rakenteessa. Lisäksi OWASP:n vuoden 2013 listauksessa verkkosovelluksia uhkaavista kymmenestä vaarallisimmista haavoittuvuuksista voidaan erottaa vielä yksi tekijä: verkkosovelluksen alustan puutteellinen konfigurointi.

Verkkosovelluksia kohtaavien tietoturvahaavoittuvuuksien suuren lukumäärän vuoksi kaikkia haavoittuvuuksia ei tämän tutkielman puitteissa voida käydä läpi. Rajoitan täten haavoittuvuusosion kattamaan The Open Web Application Security Project-organisaation (OWASP) vuoden 2013-listauksessa (2013) olleisiin. OWASP:n lista sisältää maailman kymmenen vaarallisinta verkkosovelluksia uhkaavaa tietoturvariskiä ja lista edustaa laajaa yhteisymmärrystä asiantuntijoiden kesken siitä, että mitkä ovat kymmenen vaarallisinta verkkosovelluksia uhkaavaa tietoturvahaavoittuvuutta. The Open Web Application Security Project on maailman laajuinen voittoa tavoittelematon organisaatio, jonka pyrkimyksenä on parantaa verkkosovellusten tietoturvaa levittäen tietoa

sovelluksia kohtaavista haavoittuvuuksista sekä keinoista ehkäistä niitä. OWASP julkaisee tasaisin väliajoin top-10 listauksen kyseisen hetken vaarallisimmista verkkosovelluksia kohtaavista tietoturvahaavoittuvuuksista.

Pohjaan tutkielmassani seuraavaksi esitettävät haavoittuvuudet OWASP:n vuoden 2013 top 10 listaan.

TAULUKKO 1 OWASP:n vuoden 2013 top 10 lista vaarallisimmista verkkosovelluksia uhkaavista haavoittuvuuksista. Suomennettu lähteestä OWASP (2013)

Sija	Haavoittuvuuden nimi
1.	Injektio
2.	Puutteellinen autentikointi ja istuntotunnisteen käsittely
3.	Cross-site Scripting
4.	Varmistamattomat suorat viittaukset
5.	Puutteellinen turvallisuuskonfigurointi
6.	Arkaluontoisen tiedon paljastuminen
7.	Puuttumaton sallittujen toimintojen tarkistus
8.	Cross-site Request Forgery (CSRF)
9.	Tunnettujen haavoittuvaisten komponenttien käyttö
10.	Varmistamattomat uudelleen ohjaukset

3.1 Syötteiden hyödyntäminen hyökkäyksessä

Käyttäjät ovat usein dynaamisessa vuorovaikutuksessa verkkosovelluksen kanssa. Verkkosovellus käyttää esimerkiksi käyttäjän syötteitä osana tiedon etsintää tietokannasta liittäen parametreja osaksi SQL-lausekkeita sekä rakentaa syötteiden pohjalta käyttöliittymänäkymiä liittäen parametreja osaksi HTML-dokumentteja. Verkkosovellusten käyttäjien syötteiden mukaan ohjautuva dynaaminen toiminta avaa hyökkääjälle väylän hyökätä verkkosovellukseen. (Hydara, Sultan, Zulzali & Admodisastro, 2015.) Stuttardin ja Pinton (2008) mukaan verkkosovellusten ydin tietoturvaongelma on juuri se, että käyttäjä voi antaa täysin mielivaltaisen syötteen sovellukselle ja kehittäjät eivät voi tehdä olettamuksia siitä, että minkälaisia syötteitä sovellukselle annetaan. Suurin osa verkkosovelluksia kohtaavista haavoittuvuuksista johtuu heikosta syötteiden validionnista. Haavoittuvuus syntyy, kun verkkosovellus käyttää käyttäjän syötettä osana jotakin tietoturvan kannalta herkkää operaatiota ilman että syöte tarkastetaan kunnolla ja tietynlaisella syötteellä hyökkääjä kykenee saamaan verkkosovelluksen tekemään vaarallisia toimenpiteitä. (Balzarotti ym., 2008.)

OWASP:n vuoden 2013 top 10 listauksessa (2013) syötteiden kautta syntyvät hyökkäykset ovat suuresti edustettuna ja tähän kategoriaan voidaan lukea kaksi eri riskiä, koska nämä suoranaisesti käyttävät hyväkseen verkkosovelluksen ohjelmointivirhettä:

- Injektio (engl. injection)

- Cross-site scripting

Injektio tyypin haavoittuvuus voi liittyä useaan eri teknologiaan, mutta yleisin on SQL injektio. SQL injektio ja Cross-site Scripting ovat yleisesti paljon tutkittuja niiden yleisyyden ja suurien vahinkojen vuoksi (Buja, Jalil, Ali & Rahman, 2014).

3.1.1 SQL injektio

SQL-injektio on injektiotyypin hyökkäys, jossa hyökkääjä kykenee saamaan tietokannan suorittamaan SQL-lausekkeen, jota kehittäjät eivät ole tarkoittaneet sallituksi tietokantakomennoksi. Hyökkäysmenetelmä luo vakavan tietoturvauhan, koska se mahdollistaa hyökkääjälle rajoittamattoman pääsyn verkkosovelluksen käyttämään tietokantaan ja täten pääsyn sovelluksen tietokannan tietoihin. (Halfond, Viegas & Orso, 2006.)

Seuraavassa kuvasarjassa esitetään esimerkki SQL-injektion syntymisestä tietokannassa, jossa on tietokantataulu nimeltä ”accounts” ja siinä sarakkeet nimeltä ”name” ja ”password”. SQL-injektion mahdollisuus syntyy, kun verkkopalvelun sovelluslogiikka luo dynaamisesti SQL-lausekkeet käyttäjän http-pyynnön mukana tulleiden parametrien mukaan käyttäen normaaleja merkkijono-operaatioita.

```
query = "SELECT * FROM accounts WHERE name='"
+ request.getParameter("name")
+ "' AND password='"
+ request.getParameter("pass") + "'";
```

Kuvio 3 Esimerkki ohjelmakoodista, joka mahdollistaa SQL-injektion

Yllä olevassa ohjelmistokoodissa query-nimiseen muuttujaan sijoitetaan SQL-lauseke, jonka tarkoitus on noutaa tietokannasta accounts-taulusta rivi, jonka name- ja password-sarakkeiden arvot täsmäävät parametrina tulleisiin vastaaviin. Lauseketta käytetään tunnistamaan käyttäjä, joka on aikeissa kirjautua verkkosovellukseen. Ylläolevan autentikointilogiikka voidaan kiertää yksinkertaisella toimenpiteellä. Jos käyttäjä kirjoittaa password-kenttään esimerkiksi ” ’OR’ a’=’a” muodostuu seuraavan kuvan kaltainen SQL-lauseke. Tuloksena syntyneen SQL-lausekkeen avulla verkkosovellukseen voidaan kirjautua ilman, että salasanaa tarkastetaan ollenkaan.

```
SELECT * FROM accounts WHERE
name='hyokkaaja' AND password='' OR 'a'='a'
```

Kuvio 4 Tuloksena syntynyt SQL-lauseke

SQL-injektion mahdollistavia mekanismeja on löydetty useita. Ne poikkeavat toisistaan väylän suhteen, mitä pitkin injektiossa käytettävä parametri välittyy

tietokantaohjelmalle sekä missä vaiheessa itse hyökkäys realisoituu. (Halfond, Viegas & Orso, 2006.)
Hyökkääjä voi toteuttaa SQL-injektion käyttöliittymän *syötteiden kautta*. Tässä hyökkäysmenetelmässä hyökkääjä lähettää http-pyynnössään palvelimelle merkkijonon, joka kääntyy palvelimen sovelluslogiikassa SQL-lausekkeeksi, jota kehittäjät eivät olleet tarkoittaneet lailliseksi SQL-lausekkeeksi. Hyökkääjä voi lähettää vaarallisen merkkijonosyötteen sisältävän http-pyynnön palvelinpuolenjärjestelmälle tutkien verkkosovelluksen käyttöliittymän muodostavan HTML-dokumentin input-kenttien nimiä ja tämän jälkeen lähettämällä palvelinpuolenjärjestelmälle http-pyynnön, jonka parametrien nimet ovat samoja, kuin käyttöliittymässäkin ja parametrien arvot mahdollisesti muodostaisi SQL-injektion. (Halfond, Viegas & Orso, 2006.)

SQL-injektion voi toteuttaa myös *evästeiden välityksellä*. Tämän kaltainen injektiomekanismi mahdollistuu, jos SQL-lausekkeiden muodostamiseen käytetään, evästeiden arvoja. Evästeiden avulla tapahtuva injektiomekanismi on hyökkääjän näkökulmasta yksinkertainen toteuttaa, koska käyttäjien on helppo muokata palvelimen heille tallentamia evästeitä ja täten muokata myös niiden avulla muodostuvaa SQL-lauseketta. (Halfond, Viegas & Orso, 2006.)

Anleyn mukaan (2002) SQL-injektio voidaan toteuttaa myös niin, että itse injektio ja hyökkäys tapahtuvat erillä toisistaan. Tämänkaltaisessa hyökkäysmekanismissa hyökkääjä kykenee toimittamaan sovelluksen tietokantaan merkkijonon, joka vasta myöhemmässä vaiheessa, palvelinpuolenjärjestelmän sitä prosessoidessa, realisoituu vaaralliseksi tietokantakomennoksi. Myöhemmin realisoituvat SQL-hyökkäykset ovat haasteellisia havaita, koska hyökkäyksen vaikutus ja injektiotapahtuma ovat erillä toisistaan.

SQL-injektio on verkkosovellusmaailmassa yleisin injektiotyypin hyökkäys. OWASP:in top 10 listalla se on arvostettu maailman vaarallisimmaksi verkkosovellushaavoittuvuudeksi sen yleisyyden ja suurien vahinkojen johdosta. On hyvä myös muistaa, että SQL:n lisäksi injektiohaavoittuvuuksia voi esiintyä myös muissa verkkosovellusten käyttämissä teknologioissa, kuten esimerkiksi XML-dokumenttien tulkitsemiseen käytetyssä Xpathissä. Hyökkääjä voi kyetä myös oikeanlaisella syötteellä muodostamaan ei-toivotun käyttöjärjestelmäkomennon. (OWASP, 2013.)

3.1.2 Cross-site Scripting (XSS)

Cross-site scripting (XSS) on hyökkäysmenetelmä, jossa hyökkääjä onnistuu liittämään verkkosovelluksen esittämään HTML-dokumenttiin oman haitallisen JavaScript-elementin, jonka uhrin verkkoselain suorittaa. Hyökkäyksen tekee vaaralliseksi se, että HTML-dokumenttiin liitetyn JavaScript-elementin avulla hyökkääjä voi suorittaa käyttäjän verkkoselaimessa useita haitallisia toimenpiteitä, kuten esimerkiksi uudelleenohjata hänet hyökkääjän haluamalle verkkosivulle tai lähettää hyökkääjälle uhrin evästeet. (Shar & Tan, 2012.) Cross-site Scripting voidaan nähdä suoranaisesti johtuvat sovelluksen puutteellisesta syötteiden tarkistuksesta, koska verkkosovelluksen ei tulisi koskaan sallia tilannetta, jossa käyttäjä kykenee itse sisällyttämään verkkosovellusten näkymiin omia HTML-merkkauksiaan. (Wassermann & Su, 2008.) On havaittu, että vaik-

ka kehittäjien käyttäessä uusimpia tekniikoita XSS:n ehkäisemiseksi, Shar ja Tanin (2012) mukaan haavoittuvuus on silti erittäin yleinen. Yleisyyden voidaan nähdä johtuvan hyvin pitkälti siitä, että varmistustekniikoita käytetään väärin sekä XSS:ään johtavien tekijöiden ymmärrys kehittäjien keskuudessa on puutteellista. Vaikka yksittäinen XSS:n mahdollistava haavoittuvuus on helppo paikata, jokaisen haavoittuvuuden korjaaminen verkkosivustolle on haasteellista ja moni verkkosivusto ei ole tähän kyennyt. Lisäksi internetistä on löydettävissä palveluita (esim. xssed.com), jotka listaavat verkkopalveluista jo löydettyjä XSS-haavoittuvuuksia ja täten ne houkuttelevat hyökkääjiä käyttämään näitä hyödyksi. (Bates, Barth & Jackson, 2010.)

Hydaran, Sultan, Zulzalin ja Admodisastron (2005) mukaan kirjallisuudessa XSS on yleisesti jaoteltu kolmeen erilaiseen hyökkäysmenetelmään:

- Heijastettu *(engl. Reflected XSS)*
- Tallennettuun *(engl. Stored XSS)*
- DOM-pohjaiseen *(engl. DOM-based XSS)*

Heijastetussa XSS-hyökkäyksessä hyökkääjän käyttämää haitallista JavaScript-komentosarjaa ei tallenneta tietokantaan vaan hyökkääjä pyrkii saattamaan käyttäjän tilanteeseen, jossa hän itse lähettää haitallisen JavaScript-elementin palvelinpuolenjärjestelmälle, joka puolestaan liittää sen osaksi käyttäjälle esitettyä HTML-dokumenttia. Usein tämän kaltaisia haavoittuvuuksia löydetään esimerkiksi verkkosivustojen hakutoiminnoista. Kun hakukenttään kirjoitetaan hakusana ja haku toteutetaan, käyttäjälle näytetään hakutulosten lisäksi käytetty hakusana. Jos palvelimelle lähetetyn hakusanaa edustavan parametrin turvallisuutta ei tarkasteta, voi tämä sisältää esimerkiksi hyökkääjän laatiman haitallisen JavaScript-komentosarjan, jonka palvelin http-vastauksessa liitetään suoraan osaksi käyttäjälle annettua HTML-dokumenttia. Hyökkääjä voi hyödyntää tämänkaltaista haavoittuvuutta esimerkiksi lähettämällä käyttäjälle sähköpostilla hakuparametrin sisältävän URL-linkin, joka johtaa verkkosivuston haavoittuvuuden sisältävälle hakuosiolle. (Vogt .ym, 2007.)

Tallennetussa XSS-injektiossa hyökkääjä kykenee tallentamaan haitallisen JavaScript-komentosarjan verkkosovelluksen tietokantaan. Itse hyökkäys tapahtuu myöhemmin, kun hyökkäyksen uhri pyytää palvelimelta HTML-dokumenttia, johon sovellus tulee liittäneeksi hyökkääjän tallentaman JavaScript-komentosarjan. Tämän kaltaisia XSS-haavoittuvuuksia on havaittu useasti esimerkiksi Internetin keskustelupalstoilta, joissa käyttäjän jättämät viestit tallennetaan tietokantaan ja esitetään muille verkkoselaajille. Haavoittuvuuden sisältävissä verkkosovelluksissa hyökkääjä voi viestikenttään kirjoittaa JavaScript-elementin sisään komentosarjan ja lähettää sen. Jos viestin sisältöä ei tarkasteta kunnolla JavaScript-komentosarja suoritetaan muiden käyttäjien verkkoselaimessa. (Galán, Alcaide, Orfila & Blasco, 2010.)

Kolmas havaittu XSS-haavoittuvuuden tyyppi on *DOM-pohjainen XSS-haavoittuvuus*. DOM (Document Object Model) on HTML-dokumentista koostettu ohjelmointirajapinta, jota erityisesti JavaScript hyödyntää muokatessaan dynaamisesti esillä olevaa HTML-dokumenttia. DOM on jäsennelty puumalli esillä olevasta HTML-dokumentista, jossa HTML-elementit ovat käsiteltäviä

olioita, joilla on kutsuttavia metodeja ja attribuutteja. (Mozilla Foundation 2016b.) DOM-pohjainen haavoittuvuus syntyy, kun JavaScriptin tekemät ajonaikaiset muutokset DOM-malliin johtavat JavaScript-komentosarjan liittämiseen esillä olevaan HTML-dokumenttiin. Poikkeavan DOM-pohjaisesta haavoittuvuudesta tekee se, että hyökkääjän kirjoittama komentosarja ei ole käyttäjän nähtävillä olevassa HTML-dokumentissa dokumentin lataushetkellä, vaan se liitetään siihen myöhemmin JavaScriptin DOM-malliin tekemien muutosten tuloksena. (Klein, 2005.) DOM-pohjaisten haavoittuvuuksien voidaan nähdä yleistyneen, koska verkkosovellukset ovat siirtäneet paljon toiminnallisuuttaan asiakasohjelman puolelle. Tämän kaltaisen kehityksen voidaan nähdä johtuvan asiakaspuolen teknologioiden, kuten HTML5 ja JavaScript-kirjasto JQueryn suorituskyvyn parantumisesta, joka on tehnyt toiminnollisuuksien suorittamisesta asiakassovelluksessa nopeampaa ja käytännöllisempää. (Stock, Pfistner, Kaiser, Lekies, & Johns, 2015.) Tutkimuksen mukaan jopa 10 % maailman kymmenestätuhannesta suosituimmasta verkkosivustosta kärsii DOM-pohjaisesta haavoittuvuudesta (Stock, Lekies, Mueller, Spiegel & Johns, 2014).

3.2 Käyttäjän istuntotunnisteen hyväksikäyttö

Istunnon hallinta on elintärkeä osa-alue moderneissa verkkosovelluksissa. Sen avulla verkkosovellus kykenee linkittämään saman käyttäjän verkkoselailun tuottamat http-pyynnöt toisiinsa kyeten tuottamaan interaktiivisen ja käyttäjälle yksilöllisen sovelluksen käytön. Valitettavasti evästepohjainen käyttäjän verkkoistunnon hallintajärjestelmä on helposti virhealtis ja tämän takia OWASP:in top 10 listauksessa (2013) on kaksi verkkosovellusriskiä, joidenka voidaan nähdä johtuvat puutteellisesta istuntotunnisteen käsittelystä:

- Sijalla 2. Puutteellinen autentikointi ja istuntotunnisteen käsittely *(engl. Broken Authentication and Session Management)*
- Sijalla 8. Cross-site Request Forgery (CSRF)

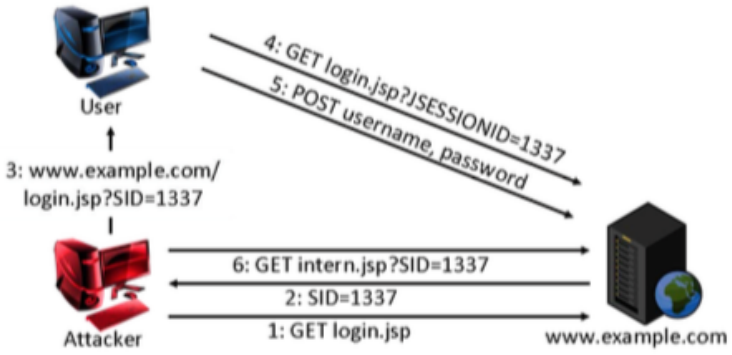
Verkkosovellus tunnistaa yksittäisen käyttäjän hänen käyttämällään yksilöllisellä istuntotunnisteella ja tämän istuntotunnisteen tulisi olla ainoastaan käyttäjän hallussa. Jos hyökkääjä kykenee kuitenkin saamaan haltuunsa käyttäjän istuntotunnisteen, hän kykenee esiintymään verkkosovellukselle käyttäjänä, jolta hän sai istuntotunnisteen tai pakottamaan käyttäjän tekemään toimenpiteitä käyttäjätilillään tahtomattaan. (Kolšek, 2002.)

Puutteellista istuntotunnisteen käsittelyä on havaittu esiintyvän jopa maailman sadan käyttäjämäärältään suosituimman verkkosivuston joukossa (Calzavara, Tolomei, Bugliesi, Orlando, 2014).

Yleisimmät istuntotunnisteen puutteellista turvaamista hyödyntävät verkkosovelluksiin kohdistuvat hyökkäystavat ovat session fixation, joka OWASP:n listauksessa asettuu sijalla 2 olevaan puutteellisen autentikoinnin ja istuntotunnisteen käsittelyn riskiin, ja cross-site request forgery, joka on listalla sijalla 8 (Takamatsu, Kosuga & Kono, 2012).

3.2.1 Session fixation

Session fixation- hyökkäysmenetelmässä hyökkääjä onnistuu pakottamaan käyttäjän istuntotunnisteen arvon, jonka avulla verkkosovellus tunnistaa käyttäjän. Hyökkäys etenee siten, että hyökkääjä vierailee ensiksi verkkosovelluksessa, jolloin sovellus kirjaa hänelle oman istuntotunnisteen. Tämän jälkeen hän harhauttaa käyttäjän verkkoselaimen käyttämään hyökkääjälle annettua istuntotunnistetta. Tämä voi tapahtua, esimerkiksi URL-linkin välityksellä tai hyödyntämällä XSS-haavoittuvuutta, joka mahdollistaa käyttäjän evästeiden muokkaamisen JavaScriptillä. (Johns, Braun, Schrank, Posegga, 2011.) Useimmat verkkosovellukset eivät enää nykyisin hyväksy istuntotunnisteita välitettäväksi URL-osoitteen parametrien välityksellä, joten realistinen hyökkäysskenario on evästeiden muokkaus (Schrank, Braun, Johns & Posegga, 2010). Kun käyttäjä kirjautuu verkkosovellukseen istunnossa, joka on tunnistettu hyökkääjän istuntotunnisteen avulla, hyökkääjällä on pääsy kyseisessä verkkosovelluksessa käyttäjän tilille, koska käyttäjä on täten kirjautunut istuntotunnisteella, joka on hyökkääjällä hallussa.



Kuvio 5 Esimerkki Session fixation-haavoittuvuuden hyväksikäytöstä

Session fixation- haavoittuvuuden syyn on nähty johtuvan istunnon käsittelyn hallinnan ja käyttäjien eriävien sallittujen toimenpiteiden välisestä epäsuhdasta. Verkkosovelluksessa käytetty sovelluskehys pitää huolta istuntotunnisteen luomisesta ja validien istuntotunnisteiden kirjanpidosta, kun taas sallituista toimenpiteistä kirjautumisen jälkeen huolehtii kehittäjä itse. (Schrank, Braun, Johns & Posegga, 2010.)

3.2.2 Cross-site request forgery (CSRF)

Cross-site request forgery on hyökkäystyyppi, jossa hyökkääjä harhauttaa sisään kirjautunutta käyttäjää siten, että hänen verkkoselaimensa lähettää haavoittuvaiselle verkkosovellukselle hyökkääjän tahdon mukaisen http-pyynnön. Koska uhri on kirjautunut verkkosovellukseen aiemmin, hänen verkkoselaimellaan on hallussa validi istuntotunniste. Tällöin kaikki hänen kauttaan tulevat http-pyynnöt palvelin käsittää tulevan kirjautuneelta käyttäjältä. Hyökkääjä voi käyttää tätä tilannetta hyväksi siten, että hän voi harhauttaa käyttäjän verk-

koselaimen lähettämään hyökkääjän haluaman http-pyynnön ja tällöin hyökkääjä kykenee tekemään verkkosovelluksessa toimenpiteitä käyttäjänä, koska käyttäjän verkkoselain automaattisesti liittää http-pyyntöihin verkkosovellukselta saadun istuntotunnisteen. (Scambray, Shema & Sima, 2006.)

Scambrayn, Sheman ja Siman mukaan (2006) CSRF-hyökkäystä on olemassa kahden tyyppistä:

- Tallennettu CSRF
- Heijastettu CSRF

Tallennettu CSRF-hyökkäys on kyseessä, kun hyökkääjä onnistuu tallentamaan sovellukselle merkkijonon, joka myöhemmässä käsittelyssä realisoituu CSRF-hyökkäykseksi. Täten tallennetussa CSRF-hyökkäyksen tapauksessa käyttäjän näkökulmasta tahtomaton http-pyyntö syntyy verkkosovelluksen toiminnan tuloksena. Heijastettu CSRF-hyökkäys tapahtuu, kun hyökkääjä kykenee harhauttamaan sovellukseen kirjautuneen käyttäjän esimerkiksi klikkaamaan linkkiä, joka johtaa siihen, että käyttäjän verkkoselain lähettää hyökkääjän tahdon mukaisen http-pyynnön verkkosovellukselle. Hyökkääjä voi myös johdattaa käyttäjän vierailemaan hyökkääjän hallinnoimalla verkkosivustolla, joka sisältää http-pyynnön lähettämiseen johtavan JavaScript-komentosarjan. Heijastetun CSRF-hyökkäyksen tapauksessa haitallisen http-pyynnön lähetys syntyy toiminnasta jollakin muulla verkkosivustolla tai –sovelluksessa, kuten esimerkiksi klikkaamalla linkkiä hyökkääjän hallinnoimalla verkkosivustolla. (Scambray, Shema & Sima, 2006.)

Hyökkääjän näkökulmasta yleisimpiä käyttötarkoituksia CSRF-hyökkäykselle ovat olleet toimenpiteiden tekeminen käyttäjänä, kuten esimerkiksi keskustelusivustolle kirjoittaminen sekä muutoksien tekeminen käyttäjätiliin, kuten esimerkiksi salasanan vaihtaminen (Lin, Zavarsky, Ruhl & Lindskog, 2009).

Palvelinpuolenjärjestelmän kannalta CSRF-hyökkäyksen tunnistaminen on vaikeaa, koska palvelimelle saapunut http-pyyntö näyttää tulevan tunnistetulta käyttäjältä ja täten se on erittäin vaikea todeta hyökkäykseksi (Scambray, Shema & Sima, 2006).

3.3 Haavoittuvuudet loogisessa rakenteessa

Verkkosovelluksista on tullut vallitsevin tapa toimittaa palveluita verkon välityksellä. Koska niillä suoritetaan monimutkaisia toiminnollisuuksia, ne ovat myös monimutkaisia rakenteeltaan. Verkkosovelluksen loogisella rakenteella tarkoitetaan sen tarkoitettua toimintatapaa, kuten esimerkiksi toimintajärjestystä verkkosovellusten operaatioiden välillä. Verkkosovelluksissa toiminnot tulee usein tietoturvan kannalta suorittaa juuri oikeassa tarkoituksellisessa järjestyksessä. Esimerkiksi autentikointi tulee suorittaa ennen, kuin voidaan suorittaa operaatioita, jotka vaativat tunnistetun käyttäjän. Se, että käyttäjä kykenisi kiertämään vaaditun kirjautumisen ja suorittamaan operaatioita, jotka ovat luvalli-

sia ainoastaan tunnistetuille käyttäjille tarkoittasi, että sovelluksessa on looginen haavoittuvuus. Autentikointi on yleinen haavoittuvuuskohta verkkosovelluksissa, koska se on yleinen toimenpide niissä. Haavoittuvuuskohdat voivat olla myös enemmän sovelluskohtaisia. Esimerkiksi looginen haavoittuvuus verkkokaupassa voi johtaa siihen, että käyttäjä kykenee käyttämään tarjouslipukkeensa useaan kertaan, vaikka se tulisi sallia vain kerran. (Li & Xue, 2011.)

OWASP:n top 10 listalta (2013) voidaan tulkita kolme haavoittuvuutta, jotka voidaan nähdä osallisiksi verkkosovelluksen toimimattomaan loogiseen rakenteeseen. Haavoittuvuudet ovat:

- Varmistamattomat suorat viittaukset (engl. Insecure Direct Object References)
- Puuttumaton sallittujen toimintojen tarkistus (engl. Missing Function Level Access Control)
- Varmistamattomat udelleenohjaukset (Unvalidated Redirects and Forwards)

Verkkosovelluksen tarkoitettu toimintajärjestys voidaan ohittaa erilaisin keinoin. Ensimmäinen esimerkki tällaisesta on niin sanottu pakotettu selaus (engl. forced browsing). Pakotetussa selailussa hyökkääjä kykenee rikkomaan sovelluksen loogisen rakenteen siirtymällä verkkoselaimella suoraan kohdesivulle sen sijaan, että sinne päädyttäisiin käyttämällä verkkosovellusta siten, kuin sitä olisi tarkoitettu käytettävän. Esimerkiksi, jos kirjautumaton käyttäjä kykenee näkemään kirjautuneille tarkoitettuun sivun siirtymällä verkkoselaimella kyseiselle sivulle suoraan kirjoittamalla sivun URL-osoitteen selaimeen, kyseessä on looginen haavoittuvuus verkkosovelluksessa. Oikea menettely olisi, että jos kirjautumaton käyttäjä yrittää siirtyä kirjautuneille tarkoitetulle sivulle, hänet uudelleen ohjataan kirjautumissivulle niin kauan, kunnes hän on kirjautunut palveluun. (Sun, Xu & Su, 2011.) Alttius pakotetulle selaukselle voidaan nähdä kuuluvan osaksi OWASP:in listauksessa olleeseen puuttumattomien sallittujen toimintojen tarkastukseen.

Toinen tapa yrittää kiertää verkkosovelluksen tarkoituksellinen toiminnollisuus on manipuloida http-kutsun parametreja (engl. http-tamper). Usein asiakasohjelma lähettää palvelinsovellukselle valmiiksi asetettuja parametreja, joiden mukaan määräytyy esimerkiksi seuraava suoritettava toimenpide. Esimerkiksi parametrina voi olla seuraavaksi suoritettavan funktion nimi. Hyökkääjä voi kuitenkin halunsa mukaan lähettää http-kutsussa mitä tahansa parametreja palvelinpuolenjärjestelmälle, joten se ei voi varmistamatta suorittaa funktioita käyttäjien lähettämien parametrien mukaan. (Bisht, Hinrichs, Skrupsky, Bobrowicz & Venkatakrishnan, 2010.) Verkkosovelluksen suorittamiin funktioihin ja operaatoihin vaikuttaminen http-kutsun parametreja muokkaamalla voidaan nähdä kuuluvan osaksi OWASP:n listauksessa (2013) olevaa varmistamattomia suoria viittauksia.

Kolmas tapa yrittää kiertää verkkosovelluksen loogista rakennetta on muuttaa verkkosovelluksen mahdollisia uudelleenohjauksia siten, että ne osoittavat jollekin muulle sivulle, kuin oli alun perin tarkoitettu ja täten kierretään ohjelmiston tarkoitettu looginen rakenne. Tämä on mahdollista, jos uudel-

leenohjauksen osoitteita on esimerkiksi kovakoodattu selainpuolen lähdekoodiin ja osoitteita ei validoida ennen kuin niihin uudelleenohjataan. Uudelleenohjauksien osoitteisiin vaikuttaminen voidaan tulkita kuuluvan osaksi varmistamattomia uudelleenohjauksia. (OWASP,2013.)

3.4 Puutteet alustan päivityksessä ja konfiguraatiossa

Verkkopalvelimen turvallinen konfigurointi näyttelee suurta osaa verkkosovellusten tietoturvassa. Palvelimen, palvelinohjelmiston ja sovelluskehyksen sopimaton konfigurointi verkkosovelluksen toimiessa lopullisessa tuotantoympäristössä saattaa johtaa tietoturvahaavoittuvuuksiin. Verkkosovelluksen käyttämien alustakomponenttien sopimattomat asetukset mahdollistavat sen, että hyökkääjä voi etsiä tehokkaammin verkkosovelluksen sisältämiä haavoittuvuuksia. (Mendes, Neto, Dura es, Vieira & Madeira, 2011). Lisäksi erittäin puutteellinen konfiguraatio voi johtaa myös verkkosovelluksen täydelliseen haltuunottoon (OWASP, 2013). Riskin suuruutta lisää myös se, että yksittäistä palvelinalustaa saatetaan käyttää usean sovelluksen alustana ja tämä saattaa johtaa tilanteeseen, jossa yksittäinen haavoittuvainen sovellus vaarantaa muidenkin ympäristössä toimivien verkkosovelluksien turvallisuuden. (Eshete, Villafiorita & Weldemariam, 2011.)

Verkkosovellusten ollessa usein räätälöityen tehtyjä ja jokainen erilaisia, niiden haavoittuvuudetkin ovat yksilöllisiä. Verkkosovelluksien usein kuitenkin yhteisesti jakama komponentti on verkkopalvelin. Verkkosovellukset usein rakennetaan toimimaan jonkin yleisesti käytössä olevan palvelinohjelmiston päälle. Esimerkiksi 50 % maailman verkkosivustoista ja –sovelluksista käyttää palvelinohjelmistonaan Apache HTTP-palvelinta (Netcraft, 2015). Tämä mahdollistaa sen, että jos hyökkääjällä on tiedossa sovelluksen alustana toimiva palvelinohjelmisto, hän kykenee kokeilemaan kyseisen komponentin yleisesti tunnettuja haavoittuvuuksia. (Stuttard & Pinto, 2008.)

Usein verkkosovelluksen kehittäjät eivät vastaa verkkopalvelimen ylläpidosta ja tämä kuilu heidän välillään on yksi tekijä verkkosovellusten haavoittuvuuksien muodostumisessa. Verkkosovelluksen turvallinen suoriutuminen palvelimella vaatii molemmilta osapuolista ymmärrystä tietoturvasta. (OWASP, 2015.)

OWASP:in top 10 listauksessa (2013) voidaan nähdä kolme eri haavoittuvuutta kuuluvan osaksi sopimatonta konfiguraatiota:

- Sijalla 5 oleva puutteellinen turvallisuuskonfigurointi (engl. Security Misconfiguration)
- Sijalla 6 oleva arkaluontoisen tiedon paljastuminen (engl. Sensitive Data Exposure)
- Sijalla 9 oleva tunnettujen haavoittuvaisten komponenttien käyttö (engl. Using Known Vulnerable Components)

3.4.1 Verkkopalvelimen vääränlainen konfigurointi

Verkkopalvelinohjelmistojen, että ohjelmistokehysten tiedossa olevia tietoturvahaavoittuvuuksia paikkaillaan ohjelmistopäivityksillä. Palvelimen tai sen lisämoduulien vanhentuneet ohjelmistopäivityksen mahdollistavat hyökkääjälle vanhojen tietoturvahaavoittuvuuksien hyväksikäytön. Esimerkki erittäin laajalti verkkosovelluksien tietoturvaa koskevasta tietoturvauhkasta, joka johtui tietyssä ohjelmistoversiossa olleesta virheestä, oli HeartBleed-haavoittuvuus. OpenSSL-salauskirjastossa oli ohjelmointivirhe, joka mahdollisti sen, että hyökkääjät kykenivät ohjelmointivirhettä hyödyntämällä lukemaan verkkopalvelimelta erittäin arkaluontoista dataa, kuten kirjautumistunnuksia ja salausavaimia. Arvioiden mukaan haavoittuvuus koski 24 – 55 %maailman miljoonasta suosituimmasta salausta käyttävästä verkkosivustosta. (Durumeric. ym, 2014.) Suoranaisesti päivittämättömät ohjelmistot täsmäävät OWASP:n vuoden 2013 top 10 listauksessa (2013) olleeseen tunnetuiden haavoittuvaisten komponenttien käyttöön.

Monet verkkopalvelimet sisältävät julkisen järjestelmähallinnan rajapinnan johon saavutetaan pääsy käyttäjätunnusta ja salasanaa vastaan. Usein järjestelmähallintasovellus saavutetaan palvelimen juurihakemistosta tai erillisestä portista. Yleinen ja yksinkertainen haavoittuvuus on, että järjestelmävalvonnan rajapinnan *oletussalasanaa ei muuteta*, joka mahdollistaa hyökkääjälle helpon pääsyn palvelimen järjestelmänvalvojan rooliin. Riski vaarantaa järjestelmävalvojan oikeudet kasvaa, jos palvelimen järjestelmävalvojan käyttäjätilin oletusarvoja ei muuteta. (OWASP, 2015.) Tämän haavoittuvuuden voidaan nähdä kuuluvan osaksi OWASP:n listauksessa (2013) ollutta puutteellista turvallisuuskonfiguraatiota.

Verkkosovellusta julkaistaessa tuotantokäyttöön tulee verkkopalvelimen hakemistolistaus-ominaisuus ottaa pois käytöstä. Hakemistolistaus helpottaa kehittäjiä kehitysvaiheessa, mutta tuotantokäytössä se ei saa olla sallittuna ominaisuutena, koska hyökkääjä hyötyisi tästä suuresti. Hyökkääjä näkisi verkkosovelluksen hakemistorakenteen ja voisi vapaasti tutkia sovelluksen lähdekoodia ja etsiä sieltä haavoittuvuuksia. (Acunetix, 2012.) Tämänkaltaisen haavoittuvuuden voidaan nähdä myös kuuluvan osaksi puutteellista turvallisuuskonfiguraatiota.

Tiedostojen turvalliset luku- ja kirjoitus- ja suoritusoikeudet ovat tärkeässä roolissa koko verkkosovelluksen turvallisuuden osalta. Vääränlaiset oikeudet voivat johtaa esimerkiksi siihen, että hyökkääjällä on mahdollisuus muokata tiedostoja tai poistaa niitä. Esimerkiksi kirjoitusoikeus verkkosovelluksen lähdekooditiedostossa saattaa antaa hyökkääjälle mahdollisuuden muokata sovelluksen lähdekoodia. (Auger, 2009.) Tiedostojen luku-, suoritus- ja kirjoitusoikeudet määritellään verkkopalvelimen käyttöjärjestelmässä. Tiedostojen oikeuksien sopimaton määrittäminen voidaan nähdä myös puutteeksi turvallisuuskonfiguraatiossa.

Jos käyttäjän ja palvelimen välillä siirretään arkaluontoista dataa, kuten esimerkiksi salasanoja tai pankkikorttinumeroita, tulee tämänkaltainen liikenne aina salata. Vanhoista salausprotokollista on löydetty haavoittuvuuksia, jotka antavat hyökkääjälle mahdollisuuden vakoilla salattua liikennettä (Canvel,

Hiltgen, Vaudenay & Vuagnoux, 2003). Verkkopalvelinta konfiguraatioita tehdessä tuotantokäyttöön tulee varmistaa, että palvelin ei anna käyttäjälle mahdollisuutta salata liikennettä turvattomalla salausprotokollalla. Nykyisin turvalliset salausprotokollat ovat TLS 1.1 ja TLS 1.2. Vanhentuneet salausprotokollat SSLv2, SSLv3 ja TLS1.0 on todettu haavoittuviksi ja hyökkääjä on kykeneväinen kiertämään vanhentunutta salausprotokollaa käyttävän tietoliikenteen salauksen ja vakoilemaan täten sitä (Neohapsis, 2016). Heikko salausmenettely voidaan nähdä kuuluvan osaksi OWASP:n listauksessa ollutta arkaluontoisen tiedon paljastumista. Vanhentuneen salausprotokollan käyttö kuuluu myös osaksi tunnettujen haavoittuvaisten komponenttien käyttöä.

Lisäksi verkkopalvelimet sisältävät usein jonkin demonstraatiotarkoitukseen asennetun näytesovelluksen tai tiedoston, jonka tarkoituksena on osoittaa, että verkkopalvelin toimii odotetusti asennuksen jälkeen. Joidenkin näytesovellusten on todettu kuitenkin sisältävän haavoittuvuuksia ja täten ne tulee poistaa käytöstä. (CVE, 1999.) Tämä haavoittuvuus voidaan nähdä myös kuuluvan osaksi puutteellista turvallisuuskonfiguraatiota.

3.4.2 Sovelluskehyksen vääränlainen konfigurointi

Verkkosovellukset ovat monimutkaisia kokonaisuuksia ja vaikka ne testattaisiin perusteellisesti kehittämisvaiheessa, saattaa niihin silti jäädä ohjelmointivirheitä, jotka nostavat poikkeuksen. Poikkeuksia ovat esimerkiksi nollalla jakaminen, tyhjään muistipaikkaan osoittaminen tai virhe tietokantahaussa. Kehittäjien tulisi toteuttaa sovellukseen oikeanlainen poikkeuksienkäsittely, joka varmistaa, että verkkosovellus ei kaadu ja toiminta jatkuu. (OWASP, 2009.) Kuitenkin sovelluksiin saattaa jäädä huomioimattomia poikkeustilanteita, joidenka johdosta sovelluksen suoritus keskeytetään. Virhetilanteet ovat mielenkiintoisia hyökkääjän näkökulmasta siksi, että jos debug-tarkoituksin käytettyjä informatiivisia virheilmoituksia ei ole kytketty pois päältä ja sovelluskehyksen asetuksia ei ole muutettu tuotantokäyttöön sopiviksi, suorituksen keskeytyessä saatetaan tarjota hyökkääjälle hyödyllistä informaatiota virheviestin muodossa. (Stuttard & Pinto, 2008.)

Hyödyllinen virheviesti hyökkääjän näkökulmasta on, jos palvelin palauttaa virheen johdosta kutsupinon (engl. stack trace). Kutsupinon sisältävä virheilmoitus annetaan, jos verkkosovellus kohtaa ajonaikaisen virheen, joka johtaa sovelluksen kaatumiseen. Kutsupino antaa hyökkääjälle paljon informaatiota järjestelmästä. Sen avulla hyökkääjä pystyy määrittelemään, miten sovellus käsittelee käyttäjien antamia syötteitä palvelinpuolenjärjestelmässä. Kutsupinosta näkee myös sen, että mitä kolmannen osapuolen ohjelmakirjastoja sovelluksessa on käytetty. Tämä on hyödyllinen tieto hyökkääjälle, koska usein ohjelmakirjastot ovat vapaasti jokaisen ladattavissa ja täten hyökkääjä voi tutkia, kuinka ohjelmakirjastot on toteutettu ja tämän avulla päätellä, kuinka hyökättävä verkkosovellus toimii. Virheilmoituksen avulla hyökkääjä voi myöskin päätellä sovelluksessa käytetyt alustakomponentit ja niiden versiot, ja täten käyttää hyväksi esimerkiksi yleisesti tiedossa olevaa tietyn ohjelmistoversion haavoittuvuutta. Tulostetusta kutsupinosta selviää myös esimerkiksi lähdekoo-

din rivinumeroita ja funktioiden nimiä joita hyökkääjä voi myös käyttää hyväksi pääteltäessä ohjelmiston rakennetta. (Stuttard & Pinto, 2008.)

Haavoittuvuudet, jotka johtuvat sovelluskehyksen tuotantokäyttöön sopimattomista asetuksista, voidaan nähdä OWASP:n listauksessa kuuluvan osaksi puutteellista turvallisuuskonfiguraatiota.

4 KÄYTÄNTEET HAAVOITTUVUUKSIEN EHKÄISEMISEKSI

Verkkosovellusten kehittämisvaiheen turvallisuuteen tähtäävät toimenpiteet voidaan jakaa kahteen osioon:

- Ohjelmakoodin ratkaisut
- Testaaminen ja turvallisuusanalyysi

Kehitysvaiheen ratkaisuilla tarkoitetaan, että verkkosovellus pyritään rakentamaan niin, että haavoittuvuuksia ei syntyisi. Sovelluksen ohjelmoinnin aikana on käytettävä käytänteitä, joiden on todettu parantavan sovellusten tietoturvaa ja samalla vaarallisia ja haavoittuvuuksille altistavia käytänteitä tulee välttää. Ohjelmakoodin ratkaisut korostuvat rakennettaessa uutta verkkosovellusta ja jo olemassa olevan verkkosovelluksen turvallisuuden parantamiseen ohjelmakoodin muuttaminen on haasteellista, koska tämä vaatisi paljon sovelluksen uudelleen koodaamista. (Li & Xue, 2014.)

Verkkosovelluksen *testaaminen ja turvallisuusanalyysi* koostuvat tekniikoista, jotka käsittävät verkkosovelluksen riskien tunnistamisen ja sovelluksen testaamisen näiden osalta. Analyysi voidaan jakaa kahteen osaan: staattiseen ja dynaamiseen analyysiin. Staattisella analyysillä tarkoitetaan sovelluksen turvallisuuden analysointia tutkimalla ohjelmakoodia ja löytämällä sieltä haavoittuvuuksia. Dynaamisella analyysillä tarkoitetaan puolestaan sovelluksen tarkkailua suorituksen aikana. (Li & Xue, 2014.) Dynaamiseen analyysiin kuuluu esimerkiksi perinteinen ohjelmiston testaaminen, jolla tarkoitetaan ohjelman suorittamista siten, että tarkoituksellisesti etsitään siitä ohjelmointivirheitä, jotka johtavat ei-toivottuun käyttäytymiseen (Myers, Sandler & Badgett, 2011).

Staattisen analyysin avulla haavoittuvuuksia löydätetään yleensä enemmän kuin dynaamisen avulla, mutta ongelmana on usein se, että se tuottaa myös vääriä hälytyksiä enemmän. Dynaamisen analyysin avulla löydetyt haavoittuvuudet voidaan todeta yksiselitteisesti tietoturvaongelmiksi, koska dynaaminen analyysi edustaa autenttista käyttöympäristöä ja –tilannetta. (Li & Xue, 2014.)

4.1 Syötteiden turvallisuuden takaaminen

Käyttäjien syötteitä voidaan käyttää hyökkäysvektorina, koska sovellus käyttää niitä esimerkiksi muodostaessaan tietokantakomentoja ja rakentaessaan käyttäjän nähtäväksi annettavaa HTML-dokumenttia. Tämän takia verkkosovelluskehittäjien tulee varmistaa, että kaikki käyttäjien antamat syötteet varmistetaan turvallisiksi.

4.1.1 Syötteiden turvallisuuden takaaminen ohjelmointiratkaisuilla

SQL-lausekkeiden muodostamisessa käyttäjän syötteiden pohjalta tulee aina varmistaa, että SQL-lausekkeen rakenne ei pääse muuttumaan ja hyökkääjä ei kykene manipuloimaan suoritettavaa lauseketta. Li ja Xuen (2014) mukaan SQL-lausekkeet voidaan suojata käyttämällä valmisteltua SQL-lauseketta. Valmistellussa SQL-lausekkeessa, lauseketta ja käyttäjän parametrina tulleita syötteitä ei liitetä merkkijono-operaatiolla SQL-lausekkeeksi, vaan SQL-lausekkeen rakenne määritellään ennalta ja käyttäjän parametrina tulleilla syötteillä ei voi enää muuttaa lausekkeen merkitystä (Thomas, Williams & Xie, 2008). Ohjelmointikielten tietokantayhteyksistä vastaaviin kirjastoihin tämä menetelmä on valmiiksi toteutettuna metodina ja nämä metodit varmistavat automaattisesti suoritettavan SQL-lausekkeen turvallisuuden. Esimerkiksi Javassa tietokantayhteyksistä vastaava kirjasto on JDBC ja se sisältää metodit valmisteltujen lausekkeiden tuottamiseksi. (Rooney, 2006.)

XSS-haavoittuvuutta on yleisesti ehkäisty käyttämällä sovelluskehysten automaattisia tarkistuksia XSS-haavoittuvuuden varalta. Weinbergerin, Saxenan, Akhawen, Finifterin, Shinin ja Songin tutkimuksessa (2011) havaittiin, että sovelluskehysten varmistukset eivät kuitenkaan tee verkkosovellusta täysin suojatuksi XSS-haavoittuvuuksilta, koska sovelluskehysten tarjoamien automaattisten varmistuksien kyvyt eivät kohtaa verkkosovelluksia kohtaavien vaatimusten kanssa. Sen sijaan on havaittu, että automaattisten tarkistuksien tarjoama turvallisuuden tunne kehittäjien keskuudessa saattaa jopa olla vaaraksi verkkosovelluksen turvallisuudelle. Samalla tutkijat toteavat kuitenkin, että sovelluskehykset ovat paras mahdollinen tapa suojautua XSS-haavoittuvuuksilta, koska XSS-haavoittuvuuksien estämiseen tarkoitettujen funktioiden manuaalinen kirjoittaminen on erittäin työlästä, aikaa vievää ja virhealtista. (Weinberger ym., 2011.) Automaattisia tarkastuksia kannattaa siis käyttää, mutta kehittäjien on tärkeää ymmärtää XSS:n perimmäinen syy. XSS:n osalta ohjesääntönä voidaan pitää, että jos käyttäjän syöte päätyy käyttäjän nähtäville HTML-dokumenttiin, tulee kehittäjän myös pohtia kohdetta, johon syöte päätyy. Kehittäjän tulee aina saattaa syöte muotoon, jossa se tulkitaan tekstiksi eikä verkkoselaimen toimesta suoritettavaksi koodiksi, mutta ongelmana on se, että eri kohteet vaativat eri toimenpiteet, jotta verkkoselain tulkitsi syötteen ainoastaan dataksi eikä suoritettavaksi koodiksi johtaen HTML-dokumentin manipuloinnin mahdollisuuteen. Esimerkiksi jos syöte liitetään HTML-elementin sisälle, tulee syöte koodata HTML-entiteeteiksi, joka varmistaa sen, että syötteen mukana tulevaa mahdollista JavaScript-komentosarjaa ei suoriteta vaan

sitä käsitellään ainoastaan tekstinä. Jos taas syötteet päätyvät osaksi URL-osoitteen parametreja tulee nämä parametrit koodata parametrien välityksessä käytettävällä prosenttimerkkikoodauksella. Syötteet tulee aina saattaa muotoon, jossa ne eivät lopullisessa ympäristössään pääse kääntymään koodiksi (OWASP, 2016a.)

XSS-haavoittuvuuden ehkäisemisessä ei voida sokeasti luottaa sovelluskehysten kykyyn suoriutua syötteiden turvallisuuden varmistuksesta. Syötteiden tarkastuksen ne hoitavat suurimmaksi osaksi hyvin, mutta kehittäjien tulee ymmärtää riskitekijät, jotka johtavat XSS-haavoittuvuuden realisoitumiseen.

4.1.2 Syötteiden analyysi ja testaaminen

Verkkosovelluksen haavoittuvuutta syötteiden osalta voidaan tarkastella *merkitsevällä analyysimenetelmällä*, jossa lähdekoodin pohjalta kaikki sovelluksen mahdolliset syöteväylät tunnistetaan ja tämän jälkeen mallinnetaan syötteiden kulku sovelluksen sisällä. Mallinnuksessa syötteet merkitään aluksi epäluotettavaksi ja kun ne ovat kulkeutuneet turvallisuuden varmistavan funktion läpi, ne merkitään luotetuiksi. Merkitsevää analyysiä voidaan suorittaa sekä staattisesti, että dynaamisesti. Staattinen analyysimenetelmä tutkii sovelluksen syötevirtoja lähdekoodin pohjalta ja se tehdään ilman, että sovellusta ajetaan. Dynaaminen analyysimenetelmä puolestaan seuraa sovelluksen syötevirtoja ajon aikana. Mahdollisimman kattava syötteiden analyysin ja testaaminen saadaan aikaan, kun prosessissa yhdistetään sekä staattinen että dynaaminen analyysimenetelmä. Tällaista menettelyä kutsutaan hybridianalyysiksi. (Li & Xue, 2014.) Tieteellisessä kirjallisuudessa on esitelty tapoja suorittaa hybridianalyysi. Esimerkiksi Balzarotti ym. (2008) esittivät ratkaisuna syötevarmistusten analysointityökalun Sanerin, joka ensiksi analysoi lähdekoodin pohjalta, kuinka sovellus varmistaa parametrina tulleiden syötteiden turvallisuuden ja tämän jälkeen syöttää sovellukselle suuren määrän erilaisia vaarallisiksi todettuja syötteitä, jotta sovelluksen heikot syötteidenvarmistusmenetelmät tulisivat ilmi.

Verkkosovellusten syötteiden haavoittuvuuksia voidaan testata myös puhtaasti käyttäjän näkökulmasta katsottuna ja tällöin testaajalla ei ole saatavilla sovelluksen lähdekoodia. Verkkosovellusta testataan menetelmällä, jossa sille syötetään suuri määrä erilaisia syötteitä ja tämän jälkeen havainnoidaan kuinka sovellus näihin reagoi. Verkkosovellukselle syötetään tarkoituksellisesti syötteitä, joidenka on tutkittu johtavan usein haavoittuvuuksien realisoitumiseen. Verkkosovelluksen syötteiden testaaminen suoritetaan usein jollakin testaamistarkoitukseen kehitetyllä työkalulla. Nämä työkalut yleensä ensiksi kartoittavat kaikki verkkosovelluksen käyttöliittymien mahdolliset syötekentät kulkemalla läpi sovelluksen kaikki linkit ja etsien näiden alta syötekenttiä. Tämän jälkeen työkalut generoivat niihin syötteitä, jotka on haettu tunnettuja syötehaavoittuvuuksia sisältävistä tietokannoista. Tämän jälkeen testaaja havainnoi, kuinka sovellus reagoi testitapaukseen. Testaamiseen käytetty työkaluja on sekä kaupallisia että open-source-työkaluja**.** (Li & Xue, 2014.) Verkkosovellusten syötehaavoittuvuuksia voidaan testata esimerkiksi tietoturvatestaajien yleisesti käyttämän Linux distibuutio Kali Linuxin mukana tulevalla Vega-testaustyökalulla.

Vega on ilmainen syötehaavoittuvuuksien etsimiseen tarkoitettu open-source-työkalu, joka kykenee löytämään verkkosovelluksesta SQL-injektion ja XSS:n mahdollistavia haavoittuvuuksia. (Kali Tools, 2014.)

4.2 Verkkoistunnon hallinnan takaaminen

Verkkoistunnon suojaamisen puutteet johtavat tilanteeseen, jossa hyökkääjä kykenee varastamaan verkkoistuntotunnisteen ja esiintymään sovellukselle käyttäjänä, jolta hän istuntotunnisteen kaappasi tai muuten suorittamaan toimenpiteitä käyttäjänä hyödyntäen hänen istuntotunnistettaan. (Kolšek, 2002.) De Ryckin, Desmetin, Piessenssenin ja Joosen (2015) mukaan verkkosovellusten tapa tunnistaa käyttäjät pelkästään istuntotunnisteen avulla on tietoturvamielessä suunnitteluvirhe, koska puutteet pelkän istuntotunnisteen käsittelyssä johtavat erittäin vakaviin tietoturvauhkiin. Tämän vuoksi verkkosovellusta kehitettäessä istuntotunnisteen turvallinen käsittely tulee varmistaa hyvin.

4.2.1 Verkkoistunnon hallinnan takaaminen ohjelmointiratkaisuin

Verkkoistunnon turvallinen käsittely vaatii useita muistettavia asioita.

Istuntotunniste tulee muodostaa sellaiseksi, että se ei ole helposti arvattavissa, eikä sitä ole mahdollista löytää valjastamalla tietokonetta kokeilemaan kaikkia mahdollisia tunnisteita (engl. brute-forcing). Istuntotunnisteesta saadaan luotua vahva, kun sen luomiseen käytetään jotakin kryptografiaan perustuvaa salausalgoritmia. (Li & Xue, 2014.)

Verkkoistunto tulisi myös mitätöidä, kun käyttäjä ei ole ollut hetkeen aktiivinen. Tällä mahdollistetaan se, että käyttäjän unohtaessa kirjautua ulos sovelluksesta hänen istuntotunnisteensa ei jää voimaan ja täten hyökkääjä ei voi käyttää hyödyksi käyttäjän aiempaa kirjautumista verkkosovellukseen. Lisäksi verkkosovelluksen istuntotunnisteen arvo tulisi aina muuttua, kun käyttäjän käyttäjäoikeustaso muuttuu. Esimerkiksi uusi istuntotunniste tulee luoda, kun käyttäjä kirjautuu palveluun ja hänelle kirjautumattomana luotu istuntotunniste tulee mitätöidä. Tällä menetelmällä erityisesti estetään session fixation-hyökkäysmenetelmä. (Li & Xue, 2014.)

Istuntotunnisteen kaappausten ehkäisemiseksi http-protokollaan evästeiden asettamiseen on lisätty uusia ominaisuuksia, jotka pyrkivät estämään istuntotunnisteen laittoman kaappauksen. Kehittäjä voi esimerkiksi määrätä, että evästeeseen ei ole mahdollista päästä käsiksi asiakasohjelman puolelta JavaScriptillä asettamalla HttpOnly-ominaisuuden evästeeseen. Toinen tärkeä asetus on asettaa Secure-ominaisuus evästeeseen, joka varmistaa, että evästettä ei saa lähettää ilman, että salattu https-yhteys on käytössä. (De Ryck ym., 2015.)

Sovelluksessa tulee myös aina käyttää ratkaisua, jossa istuntotunnisteen muodostaa palvelinsovellus. Istuntotunnisteen muodostaminen asiakkaan puolella johtaa siihen, että hyökkääjän on mahdollista suorittaa session fixation hyökkäys, koska hän voi pakottaa käyttäjän käyttämään hänen luomaansa istuntotunnistetta. (Li & Xue, 2014.)

Jotta verkkosovelluksen alttius CSRF-haavoittuvuudelle estettäisiin, tulee kaikissa verkkosovelluksen syötelomakkeissa käyttää CSRF-merkintää (engl. CSRF-token). CSFR-merkintä on satunnaisesti muodostettu arvo, jonka palvelinpuolenjärjestelmä muodostaa yksilöllisesti kullekin käyttäjälle annettavalle syötekentälle. Kun käyttäjän antama syöte kyseiseen syötelomakkeeseen palautuu takaisin palvelinsovellukselle, palvelinsovellus tarkastaa, että kyseinen syöte on saapunut sen oikeasti luomasta lomakkeesta. CSRF-merkki muodostetaan oikeaoppisesti siten, että se on käyttäjälle ja istuntotunnisteelle yksilöllinen ja sitä ei ole käytössä kenelläkään muulla käyttäjällä, se on suuri satunnaisluku ja sen muodostamiseen on käytetty turvallista kryptografiaan perustuvaa salausalgoritmia. (OWASP, 2016b) Nykyaikaiset sovelluskehykset yleisesti tukevat automaattisesti CSRF-merkintää.

4.2.2 Istunnonhallinnan toimivuuden testaaminen

Istuntotunnisteen hyväksikäyttöön liittyviä haavoittuvuuksia voidaan myös testata automatisoiduin työkaluin. Esimerkiksi Lukanta, Asnar ja Kistijantoro (2014) ovat esitelleet testaustyökalun, joka on lisäosa verkkosovellusten haavoittuvuuksien etsimiseen käytettyyn Nikto-työkaluun. Heidän menetelmänsä kykenee tunnistamaan sekä CSRF-haavoittuvuuden että alttiuden session fixation- hyökkäykselle. Heidän menetelmänsä on kykeneväinen tunnistamaan session fixation- haavoittuvuuden tutkimalla verkkosovelluksen istuntotunnisteiden arvoa ennen ja jälkeen kirjautumisen. Jos verkkosovellus ei muuta istuntotunnistetta käyttäjäoikeuksien tason muuttumisen jälkeen, tulkitsee testaustyökalu verkkosovelluksen haavoittuvaiseksi. Testaustyökalu tutkii myös sen, että verkkosovellus käyttää varmasti CSRF-merkintöjä ja jos näin ei ole, tulkitsee testaustyökalu verkkosovelluksen alttiiksi CSRF-haavoittuvuudelle. Lisäksi testaustyökalu tutkii http-vastausten evästeiden ominaisuuksia. Jos istuntotunniste lähetetään asiakasohjelmalle ilman HttpOnly- ja Secure-arvoja, testaustyökalu tulkitsee, että käyttäjän istuntotunniste voidaan kaapata, koska tällöin se lähetetään salaamattomana ja siihen voi päästä käsiksi JavaScriptin avulla asiakasohjelman puolelta. (Lukanta ym., 2014.)

4.3 Loogisen toiminnallisuuden takaaminen

Verkkosovelluksissa on usein toiminnollisuuksia, jotka ovat sallittuja vain tietyille käyttäjäryhmille. Esimerkiksi sovellukseen kirjautuneet käyttäjät voivat usein käyttää ominaisuuksia, jotka eivät ole sallittuja kirjautumattomille käyttäjille. Sovelluksen loogisen rakenteen toimivuudella tarkoitetaan, että sovellus toimii loogisesti siten, kuten sen on tarkoitettukin toimivan. Esimerkiksi, kun kirjautumaton käyttäjä kirjoittaa URL-osoitteen, jonka alla oleva sisältö tulisi olla vain kirjautuneiden käyttäjien nähtävillä, hänet uudelleen ohjataan kirjautumissivulle ja vasta onnistuneen kirjautumisen jälkeen sisältö näytetään käyttäjälle.

4.3.1 Loogisen rakenteen takaaminen ohjelmointiratkaisuin

Verkkosovelluksia kehitetään usein jonkin sovelluskehyksen avulla, koska se nopeuttaa verkkosovelluksen rakentamista. Ne pitävät huolen myös siitä, että käyttäjien on mahdollista saada pääsy vain oman käyttäjäoikeuksien mukaiseen sisältöön. Sovelluskehykset pitävät esimerkiksi huolta siitä, että jotakin tiettyä funktiota käytetään vain, jos käyttäjä on kirjautunut sisään. (Li & Xue, 2014.) Esimerkiksi alla oleva kuva on Django-sovelluskehyksen mukainen ja tässä esimerkissä kehittäjän työtä helpotetaan siten, että käyttäjän kirjautumistila kyetään selvittämään helposti ehtolauseella. Kehittäjän vastuulle jää ainoastaan kirjoittaa kyseinen kirjautumisen varmistava ehtolause. Tällöin kyetään helposti määrittämään ehtolausein sallittu sisältö kullekin käyttäjätasolle. (The Django Software Foundation, 2016b.)

```
from django.contrib.auth import authenticate
user = authenticate(username='john', password='secret')
if user is not None:
    # the password verified for the user
    if user.is_active:
        print("User is valid, active and authenticated")
    else:
        print("The password is valid, but the account has been disabled!")
else:
    # the authentication system was unable to verify the username and password
    print("The username and password were incorrect.")
```

Kuvio 6 Sovelluskehyksen hyödyntäminen määriteltäessä käyttäjäoikeuksien mukaisia sallittuja toimintoja

4.3.2 Loogisen rakenteen testaaminen ja analysointi

Koska loogisien haavoittuvuuksien syntymekanismit ovat hyvin yksilöllisiä ja sovelluksen käyttötarkoituksesta riippuvaisia, niihin ei ole olemassa kaiken kattavaa analyysi- tai testausmenetelmää, jonka avulla haavoittuvuudet kyettäisiin löytämään. Kirjallisuudessa on esitetty joitakin testaamistapoja, mutta nämä menetelmät kattavat ainoastaan jonkin yksittäisen loogisen haavoittuvuuden osan, kuten esimerkiksi mahdollisuuden kiertää autentikoinnin (Doupe.ym., 2011). Loogisen haavoittuvuuksien analysoinnin ja testaamisen perusteellinen vaikeus piilee siinä, että ei ole olemassa menetelmää, joka kykenisi tekemään määrittelyn loogisesta rakenteesta riippumatta sovelluksesta ja sen toiminnollisuuksista. (Li & Xue, 2014.) Sovelluksen tarkoituksellisen ja oikeaoppisen käyttäytymisen määrittelyyn pyritään kehittämään menetelmä ja useat tutkimukset ovat havainnoineet, että sovelluksen oikea oppinen tarkoituksellinen käyttäytyminen kyetään määrittelemään jollakin tasolla sovelluksen tarjoamien navigaatiopolkujen pohjalta (Balzarotti, Cova, Felmetsger & Vigna, 2007; Felmetsger, Cavedon, Kruegel & Vigna, 2010; Cova, Balzarotti, Felmetsger & Vigna, 2007; Li & Xue, 2011; Li, Yan & Xue, 2012). Lisäksi varteenotettavaksi määrittelymenetelmäksi on esitetty myös menetelmä, jossa asiakassovelluksen oletetaan oi-

keinkäytettynä lähettävän aina tietynlaisia http-pyyntöjä samankaltaisessa järjestyksessä ja tietyin parametrein (Guha, Krishnamurthi & Jim, 2009).

Verkkosovelluksen loogisen toiminnan täydelliseen testaamiseen vaaditaan ainakin toistaiseksi sovelluksen systemaattista toiminnollisuuksien kokeilemista ja kokeilemisen pohjalta johtopäätöksiä toimivuudesta. (Li & Xue, 2014.)

4.4 Oikeanlaiset sovellusympäristön konfiguraatiokäytänteet

Sovelluksen toimintaympäristön puutteellisen konfiguraation johdosta syntyvät haavoittuvuudet eroavat muista tässä tutkielmassa esitetyistä haavoittuvuustekijöistä siinä, että haavoittuvuudet eivät niinkään johdu ohjelmointivirheistä vaan enemminkin tietyn valmiin komponentin oikeaoppisen käytön laiminlyönnistä. Lisäksi tämän kategorian oikeat käytänteet ovat yhteisiä sovelluksesta riippumatta, koska monet suuresti toisistaan toiminnollisuuksin poikkeavat sovellukset jakavat usein saman komponentin, kuten verkkopalvelimen tai sovelluskehyksen.

4.4.1 Oikeaoppinen konfigurointi

Alustakomponenttien oikeanlaisen konfiguroinnin ydin tekijä on seurata kunkin komponentin ohjeistusta eri käyttötilanteisiin. Komponenttien ohjeistuksista selviää eri käyttövaiheisiin, kuten kehittämis- ja tuotantokäyttövaiheeseen, sopivat asetukset. (OWASP, 2015.)

Joitakin yleistettäviä oikeanlaisia menettelyitä voidaan myös havaita olevan. Sekä verkkopalvelimen että sovelluskehyksen kaikki tarpeettomat palvelut ja toiminnollisuudet tulisi poistaa käytöstä. Näihin voidaan lukea mukaan esimerkiksi juuri sovelluksen virheestä annettava virheilmoitus; komponenttien yhteydessä toimivat näyteohjelmat, joiden tarkoitus on ainoastaan demonstroida komponentin toimintaa ja verkkopalvelimen hakemiston listaava ominaisuus. (OWASP, 2015.) Usein nämä ylimääräiset palvelut on tarkoitettu avustamaan kehittämisvaiheessa virheen etsintää, mutta verkkosovellusta tuotaessa tuotantokäyttöön ne tulee ottaa pois käytöstä. Nykyaikaiset sovelluskehykset usein tarjoavat kehittäjille jonkin vaihtoehtoisen tavan analysoida mahdollisia ajon aikaisia virheitä ilman, että järjestelmän tietoturva vaarantuu. Esimerkiksi sovelluskehys Django lähettää automaattisesti virheviestinä kutsupinon kehittäjille sähköpostitse, jos verkkosovellus kohtaa virheen (The Django Software Foundation, 2016a).

Toinen yleisluontoinen ohje komponenttien konfigurointiin on käyttöoikeuksien sopivat määrittelyt. Sovelluksilla ja niiden toiminnollisuuksilla tulisi olla ainoastaan niin vahvat luku-, kirjotus- ja suoritusoikeudet, että ne selviävät niiden tehtävistä. Lisäksi käyttöoikeuksien määrittelyyn kuuluu myös oletus käyttäjätilien ja salasanojen muuttaminen. (OWASP, 2015.)

Komponenttien ohjelmistopäivitykset tulisivat olla aina ajan tasalla. Tämä on yhteydessä myös siihen, että ylläpitäjien tulisi seurata komponenteista jul-

kaistuja uusia haavoittuvuustiedotteita ja haavoittuvuuksiin julkaistuja uusia korjauspäivityksiä. (OWASP, 2015.)

Lisäksi verkkosovelluksen alustan konfiguraatiota tehdessä kannattaa käyttää jotakin automatisoitua työkalua. Automatisoituja työkaluja on tarjolla paljon ja nämä kykenevät asettamaan verkkopalvelimelle ja -palvelinohjelmistolle oikeanlaiset asetukset ohjeistuksen ja käyttötarkoituksen pohjalta. Lisäksi ennalta hyvin konfiguroitu palvelin voidaan suoraan asetuksineen kloonata käytettäväksi uudelleen. (OWASP, 2015.)

Verkkopalvelimen salauspalveluiden osalta on tärkeää muistaa, että palvelimen ei tule antaa mahdollisuutta käyttäjälle käyttää vanhentuneita salausalgoritmeja, kuten SSLv1, SSLv2, SSLv3 tai TLS1.0. Käytettävän salausalgoritmin tulisi olla TLS1.1 tai TLS1.2 sillä näistä salausmenetelmistä ei ole löydetty haavoittuvuuksia. Lisäksi tulee varmistaa, että jos käyttäjän ja verkkosovelluksen välillä vaihdetaan arkaluontoista informaatiota, tulee kytkeä pois mahdollisuus käyttää salaamatonta http-protokollaa kyseisellä sivulla. Verkkopalvelinohjelmistot tarjoavat yleisesti ominaisuuden, joka kytkettynä estää verkkosovelluksen käytön ilman salausta. (Red Hat Customer Portal, 2016.)

4.4.2 Alustakomponenttien ja niiden konfiguraation testaaminen

Verkkosovelluksen alustaohjelmistojen puutteista syntyviä haavoittuvuuksia voidaan testata työkaluilla, jotka on kehitetty löytämään löytämään niitä. Nämä työkalut toimivat siten, että ne lähettävät verkkosovellukselle erilaisia http-pyyntöjä ja tämän jälkeen analysoivat vastineeksi saatua http-vastausta ja päättelevät tästä, että mitä alustaohjelmistoja sovellus käyttää. Alustaohjelmistojen tunnistamisen etuna testaajan tai hyökkääjän näkökulmasta on se, että kun käytetyt ohjelmistot, sovelluskehys ja sovelluskirjastot on tunnistettu, voi näistä komponenteista ennalta löydettyjä haavoittuvuuksia käyttää hyödyksi. (Stuttard & Pinto, 2011.)

Verkkopalvelinohjelmiston testaamiseen puutteellisen konfiguraation ja ylläpidon osalta voidaan käyttää Kali Linuxin mukana tulevaa testaustyökalu Niktoa, joka kykenee testaamaan, onko verkkosovelluksen käyttämä verkkopalvelin altis tunnetuille haavoittuvuuksille. Työkalu kykenee esimerkiksi selvittämään verkkosovelluksen käyttämän verkkopalvelinohjelmiston ohjelmistoversion ja ilmoittamaan testaajalle, jos kyseinen ohjelmisto on vanhentunut ja haavoittuvainen. Ohjelmisto kykenee myös skannauksen tuloksena selvittämään, onko verkkopalvelin asetuksiltaan haavoittuvainen. Työkalu kertoo myös, jos verkkopalvelimella on edelleen toiminnassa verkkopalvelinohjelmiston mukana tulleita oletusohjelmia. (Sullo & Lodge, 2011.)

Verkkosovelluksen käyttämän sovelluskehyksen konfiguroinnin testaamiseen voidaan käyttää esimerkiksi WhatWeb-työkalua, joka tulee myös oletuksena mukana yleisesti tietoturvatestaajien käyttämässä Kali Linux käyttöjärjestelmässä. WhatWeb on kykeneväinen tunnistamaan sovelluskehyksen ja sovelluskirjastot, joista verkkosovellus koostuu. Analyysin tuloksena tietoturvatestaaja voi tutkia, onko sovelluskehyksen ohjelmistoversio esimerkiksi vanhentunut ja altis tunnetuille haavoittuvuuksille. (Horton, 2016.) Tämän tiedon pohjalta testaaja kykenee myös testaaman sovellusta pyrkien löytämään haavoittu-

vuuksia ja puutteita, jotka ovat ominaisia juuri kyseisille sovelluskehykselle ja -kirjastoille.

Alustaohjelmistojen haavoittuvuuksien etsintään käytetään jotakin siihen kehitettyä työkalua, jos testaaja on kehitystiimin ulkopuolinen henkilö eikä tiedä sovelluksen alla olevista ratkaisuista. Jos testaaja on tietoinen alustaratkaisuista, testaaminen voidaan suorittaa myös käymällä systemaattisesti läpi konfiguraatio-ohjeistus kyseiseen käyttötilanteeseen. (OWASP, 2015.)

5 YHTEENVETO

Tämä tutkielma oli kirjallisuuskatsaus, joka yhdisteli aiempia verkkosovellusten haavoittuvuuksia käsitteleviä tutkielmia yhdeksi katsaukseksi. Tarkoituksena oli aikaisempien tutkimuksien pohjalta tunnistaa verkkosovelluksia uhkaavat yleisimmät tietoturvauhat ja kuinka nämä voidaan välttää kehitysvaiheen ratkaisuilla.

Verkkosovellusten yleisuus on kasvanut suuresti muutaman vuosikymmenen aikana niiden antaessa mahdollisuuden tarjota palveluita käyttäjän päätelaitteesta välittämättä. Tarve kehittää ainoastaan yksi sovellus vähentää kustannuksia merkittävästi. Verkkosovellukset ovat kuitenkin alttiimpia kyberhyökkäyksille niiden kaikkien saavutettavissa olevan julkisen verkkokäyttöliittymärajapinnan vuoksi. (Rafique ym., 2015.) Tämän lisäksi tämän tutkielman pohjalta voidaan myös todeta, että verkkosovellusten useita eri teknologioita yhdistelevä rakenne tekee ne alttiiksi monille eri tietoturvahaavoittuvuuksille sekä hankaloittaa sovelluksen tietoturvan toteutusta, koska kehittäjät joutuvat ottamaan huomioon eri teknologioiden käytöstä syntyvät haavoittuvuudet. Verkkosovelluksia uhkaavat tekniset tietoturvahaavoittuvuudet voidaan yleisimpien verkkosovelluksia uhkaavien tietoturvahaavoittuvuuksien pohjalta kategorioida neljään eri haavoittuvuustekijään: syötteiden mukana tulevat haavoittuvuudet, istuntotunnisteen käsittelyyn liittyvät haavoittuvuudet, verkkosovelluksen loogisen rakenteen pettäminen sekä verkkosovelluksen alustan puutteellinen konfiguraatio.

Syötteet ovat yksi väylä hyökätä verkkosovellusta vastaan, koska verkkosovellukset yleisesti käyttävät paljon käyttäjän syötteitä osana toimintaansa. Syötteitä liitetään esimerkiksi osaksi SQL-lausekkeita sekä HTML-dokumentteja. Oikeanlaisella syötteellä hyökkääjä kykenee kiertämään verkkosovelluksen tarkoitetun toiminnan ja muodostamaan esimerkiksi tahtonsa mukaisia SQL-lausekkeita tai liittämään oman JavaScript-komentosarjansa osaksi muille käyttäjille osoitettavaa HTML-dokumenttia. Kehittämisvaiheessa tämän kaltaiset haavoittuvuudet voidaan ehkäistä käyttämällä oikeanlaisia ohjelmointiratkaisuja. SQL-injektiot kyetään ehkäisemään käyttämällä valmisteltua SQL-lauseketta (Li & Xue, 2014.). XSS-haavoittuvuuden ehkäisemisessä hyvä apu on käyttää sovelluskehysten tarjoamia automaattisia tarkistusfunktioita, mutta

samalla kehittäjän tulee ymmärtää myös automaattisten funktioiden rajallisuus suoriutua täydellisesti tehtävistään. Tämän vuoksi kehittäjien on tärkeää pohtia jokaisen käyttäjän antaman syötteen lopullinen määränpää HTML-dokumentissa ja varmistua, että syöte koodataan vaarattomaksi niin, että se ei pysty kääntymään suoritettavaksi koodiksi määränpäässään. (Weinberger. ym, 2011.) Verkkosovellusta testattaessa syötehaavoittuvuuksia löydetään parhaiten käyttämällä merkitsevää hybridianalyysimerkintää, jossa verkkosovelluksen mahdolliset syötevirrat analysoidaan lähdekoodin pohjalta. Syötevirrat merkitään turvallisiksi siinä vaiheessa, kun syötteet ovat kulkeutuneet syötteen turvallisuuden varmistavan funktion läpi. Lisäksi syöteväyliin syötetään paljon erilaisia vaarallisia syötteitä, jotta heikot varmistusmenetelmät tulisivat ilmi. Lisäksi syötetarkastuksen turvallisuutta voidaan testata silloinkin, jos lähdekoodia ei voida tarkastella. Tällöin verkkosovellusta testataan jollakin työkalulla, joka tunnistaa verkkosovelluksen syötepisteet ja syöttää näihin paljon erilaisia syötteitä. Tämän jälkeen testaaja tarkkailee kuinka verkkosovellus syötteisiin reagoi. (Li & Xue, 2014.)

Verkkoistunnon suojaamisen puutteet johtavat tilanteeseen, jossa hyökkääjä kykenee varastamaan verkkoistuntotunnisteen ja esiintymään sovellukselle käyttäjänä, jolta hän istuntotunnisteen kaappasi (session fixation) tai muuten suorittamaan toimenpiteitä käyttäjänä hyödyntäen hänen istuntotunnistettaan (Cross-site Request Forgery). (Kolšek, 2002.) Jotta istuntotunnisteen väärinkäytöt voidaan ehkäistä, tulee istuntotunnistetta muodostaessa käyttö kryptografiaan perustuvaa algoritmia, jonka avulla estetään hyökkääjän toimesta istuntotunnisteen arvaaminen. Istuntotunnisteen tulee mukautua myös tilanteen mukaan: istuntotunniste tulee mitätöidä, kun käyttäjä ei ole ollut aktiivinen vähään aikaan sekä istuntotunnisteen tulee aina muuttua, kun käyttäjän käyttöoikeuksien taso muuttuu. (Li & Xue, 2014.) Istuntotunnistetta asetettaessa tulee aina määrittää myös käyttöön evästeiden secure- ja httpOnly-määritykset (De Ryck ym., 2015.) Lisäksi jotta verkkosovellus ei olisi altis CSRF-hyökkäykselle, tulee sen lomakkeita muodostaessaan käyttää CSRF-merkintää, joka varmistaa, että saapunut http-pyyntö on oikeasti tullut käyttäen verkkosovelluksen luomaa lomaketta (OWASP, 2016b). Verkkosovelluksen istunnonhallinnan toimivuutta voidaan testata työkuilla, jotka analysoivat verkkosovelluksen tapaa käsitellä istuntotunnistetta (Lukanta ym., 2014.).

Kolmas tässä tutkielmassa tunnistettu haavoittuvuuden tyyppi on verkkosovelluksen loogisen rakenteen kiertäminen. Verkkosovelluksissa esimerkiksi autentikoinnin osalta on tärkeää, että toimenpiteet tehdään juuri oikeassa järjestyksessä ja virhe verkkosovelluksen loogisessa rakenteessa mahdollistaisi tämän kiertämisen. Verkkosovelluksien kehittämisessä usein käytetyt sovelluskehykset tarjoavat metodit varmistaa loogisen rakenteen pitävyys ja kehittäjien on hyvä käyttää niitä hyödykseen rakentaessaan verkkosovellusta. Toistaiseksi loogisen rakenteen toimivuuden analysointiin ei ole kehitetty menetelmää, joka kykenisi tekemään määrittelyn loogisesta rakenteesta riippumatta sovelluksesta ja sen toiminnollisuuksista. (Li & Xue, 2014.) Tällä hetkellä kehityksen alla olevat menetelmät lähestyvät ongelmaa näkökulmasta, jossa ne määrittelisivät verkkosovelluksen tarkoitetun rakenteen sen käyttöliittymän navigointipolkujen pohjalta. (Balzarotti ym., 2007; Felmetsger ym., 2010; Cova ym., 2007; Li &

Xue, 2011; Li ym., 2012). Tällä hetkellä verkkosovelluksen loogisen rakenteen testaustavat perustuvat sen toiminnollisuuksien kokeilemiseen. (Li & Xue, 2014.)

Neljäs verkkosovellusten haavoittuvuuksiin johtava tekijä on verkkosovelluksen alustan sopimaton konfiguraatio verkkosovelluksen ollessa tuotantokäytössä. Verkkosovelluksen käyttämien alustakomponenttien sopimattomat asetukset mahdollistavat sen, että hyökkääjä voi etsiä tehokkaammin verkkosovelluksen sisältämiä haavoittuvuuksia. (Mendes, Neto, Dura es, Vieira & Madeira, 2011). Lisäksi erittäin puutteellinen konfiguraatio voi johtaa myös verkkosovelluksen täydelliseen haltuunottoon (OWASP, 2013). Riskin suuruutta lisää myös se, että yksittäistä alustaa saatetaan käyttää usean sovelluksen alustana ja tämä saattaa johtaa tilanteeseen, jossa yksittäinen haavoittuvainen sovellus vaarantaa muidenkin ympäristössä toimivien verkkosovelluksien turvallisuuden. (Eshete, Villafiorita & Weldemariam, 2011.) Tärkeitä tarkistuskohtia tuotaessa verkkosovellusta lopulliseen toimintaympäristöön ovat verkkopalvelimen ja verkkopalvelinohjelmiston osalta oletuskirjautumistunnuksien muuttaminen; hakemistolistauksen pois kytkeminen; tiedostojen sopivien luku-, suoritus- ja kirjoitusoikeuksien varmistaminen; turvallisen salausprotokollan käyttäminen; demonstraatiotarkoitukseen käytettyjen ohjelmien poistaminen käytöstä sekä sovelluskehyksen asetusten määrittäminen tuotantokäyttöön sopivaksi. Lisäksi alustaohjelmistojen turvallisuuspäivitykset tulee aina olla ajan tasalla, koska muutoin hyökkääjä kykenee käyttämään hyväksi vanhentuneen ohjelmistoversion yleisesti tiedossa olevaa haavoittuvuutta. (OWASP, 2015; Red Hat Customer Portal, 2016.) Alustaohjelmien konfiguraatiota voidaan testata käyttämällä kyseiseen tarkoitukseen kehitettyä työkalua, jotka kykenevät tunnistamaan käytetyt ohjelmistot ja niiden konfiguraatiot ja tämän pohjalta tulkitsemaan, että onko verkkosovellus näiltä osin haavoittuvainen. Työkaluja ovat esimerkiksi Nikto ja WhatWeb. (Sullo & Lodge, 2011; Horton, 2016.)

Tässä tutkielmassa verkkosovellusten haavoittuvuuksia käytiin läpi OWASP:n listauksen (2013) kautta. On kuitenkin hyvä muistaa, että kyseinen lista ei edusta kaikkia verkkosovelluksiin kohdistuvia haavoittuvuuksia vaan ainoastaan kymmentä vaarallisinta. Tämän tutkielman avulla ei siis voida täysin ehkäistä kaikkia verkkosovelluksien haavoittuvuuksia, mutta voidaan käyttää luomaan yleiskatsaus yleisimmistä haavoittuvuuksista ja keinoista ehkäistä ne. Lisäksi tässä tutkielmassa koottiin yhteen aikaisempia haavoittuvuuksista tehtyjä tutkimuksia poimien niistä vain ydin kohdat. Syvällisempi perehtyminen aihepiiriin vaatii lukijalta perehtymistä myös tämän tutkielman lähteisiin.

Mielenkiintoisia jatkotutkimusaiheita tämän tutkielman aihepiiriin liittyen on paljon. Esimerkiksi olisi tärkeää kehittää kuvantamismenetelmä, jolla kyettäisiin mallintamaan verkkosovelluksen looginen rakenne ja tämän pohjalta automatisoidusti testaamaan verkkosovellusta mahdollisilta loogiseen rakenteen virheiltä. Tämän tutkielman pohjalta voidaan myös todeta, että sovelluskehykset ovat tärkeässä osassa verkkosovellusten tietoturvan kannalta. Olisi hyvä tehdä esimerkiksi tutkimus, jossa selvitettäisiin ohjelmointikielittäin turvallisimmat sovelluskehykset. Lisäksi olisi mielenkiintoista tutkia verkko- ja työpöytäsovellusten välisiä eroja tietoturvassa.

LÄHTEET

Anley, C. (2002). Advanced SQL Injection in SQL Server Applications,

Auger, R. (2009, 12/30). Improper filesystem permissions. Haettu 4/242016 osoitteesta http://projects.webappsec.org/w/page/13246932/Improper%20Filesystem%20Permissions

Balzarotti, D., Cova, M., Felmetsger, V., Jovanovic, N., Kirda, E., Kruegel, C. & Vigna, G. (2008). Saner: Composing static and dynamic analysis to validate sanitization in web applications. Security and Privacy, 2008. SP 2008. IEEE Symposium on, (387-401). IEEE.

Balzarotti, D., Cova, M., Felmetsger, V. V. & Vigna, G. (2007). Multi-module vulnerability analysis of web-based applications. Proceedings of the 14th ACM Conference on Computer and Communications Security, (25-35). ACM.

Bass, B. M. (1990). From transactional to transformational leadership: Learning to share the vision. Organizational Dynamics, 18(3), 19-31.

Bass, B. M. (1990). From transactional to transformational leadership: Learning to share the vision. Organizational Dynamics, 18(3), 19-31.

Calzavara, S., Tolomei, G., Bugliesi, M. & Orlando, S. (2014). Quite a mess in my cookie jar!: Leveraging machine learning to protect web authentication. Proceedings of the 23rd International Conference on World Wide Web, (189-200). ACM.

Canvel, B., Hiltgen, A., Vaudenay, S. & Vuagnoux, M. (2003). Password interception in a SSL/TLS channel. Advances in cryptology-CRYPTO 2003 (s. 583-599) Springer.

Cova, M., Balzarotti, D., Felmetsger, V. & Vigna, G. (2007). Swaddler: An approach for the anomaly-based detection of state violations in web applications. Recent Advances in Intrusion Detection, (63-86). Springer.

De Ryck, P., Desmet, L., Piessens, F. & Joosen, W. (2015). SecSess: Keeping your session tucked away in your browser. Proceedings of the 30th Annual ACM Symposium on Applied Computing, (2171-2176). ACM.

DocForge. (2014, 06/20). Web application framework. Haettu 05/062016 osoitteesta http://web.archive.org/web/20150622201140/http://docforge.com/wiki/Web_application_framework

Durumeric, Z., Kasten, J., Adrian, D., Halderman, J. A., Bailey, M., Li, F., . . . Payer, M. (2014). The matter of heartbleed. Proceedings of the 2014 Conference on Internet Measurement Conference, (475-488). ACM.

Felmetsger, V., Cavedon, L., Kruegel, C. & Vigna, G. (2010). Toward automated detection of logic vulnerabilities in web applications. USENIX Security Symposium,

Fielding, R., Gettys, J., Mogul, J., Frystyk, H., Masinter, L., Leach, P. & Berners-Lee, T. (1999). Hypertext Transfer Protocol--HTTP/1.1,

Galán, E., Alcaide, A., Orfila, A. & Blasco, J. (2010). A multi-agent scanner to detect stored-XSS vulnerabilities. Internet Technology and Secured Transactions (ICITST), 2010 International Conference for, (1-6). IEEE.

Grobauer, B., Walloschek, T. & Stocker, E. (2011). Understanding cloud computing vulnerabilities. IEEE Security & Privacy, 9(2), 50-57.

Halfond, W. G., Viegas, J. & Orso, A. (2006). A classification of SQL-injection attacks and countermeasures. Proceedings of the IEEE International Symposium on Secure Software Engineering, (13-15). IEEE.

Horton, A. (2011, 04/05). Whatweb. Haettu 06/052016 osoitteesta http://www.morningstarsecurity.com/research/whatweb

Hydara, I., Sultan, A. B. M., Zulzalil, H. & Admodisastro, N. (2015). Current state of research on cross-site scripting (XSS)–A systematic literature review. Information and Software Technology, 58, 170-186.

Johns, M., Braun, B., Schrank, M. & Posegga, J. (2011). Reliable protection against session fixation attacks. Proceedings of the 2011 ACM Symposium on Applied Computing, (1531-1537). ACM.

Kali Tools. (2014, 2/18). Vega. Haettu 4/302016 osoitteesta http://tools.kali.org/web-applications/vega

Li, X. & Xue, Y. (2011). BLOCK: A black-box approach for detection of state violation attacks towards web applications. Proceedings of the 27th Annual Computer Security Applications Conference, (247-256). ACM.

Li, X. & Xue, Y. (2014). A survey on server-side approaches to securing web applications. ACM Computing Surveys (CSUR), 46(4), 54.

Li, X., Yan, W. & Xue, Y. (2012). SENTINEL: Securing database from logic flaws in web applications. Proceedings of the Second ACM Conference on Data and Application Security and Privacy, (25-36). ACM.

Lie, H., Bos, B. & Lilley, C. (1998, 3). Haettu 4/102016 osoitteesta https://tools.ietf.org/pdf/rfc2318.pdf

Lin, X., Zavarsky, P., Ruhl, R. & Lindskog, D. (2009). Threat modeling for CSRF attacks. 2009 International Conference on Computational Science and Engineering, (486-491). IEEE.

Lukanta, R., Asnar, Y. & Kistijantoro, A. I. (2014). A vulnerability scanning tool for session management vulnerabilities. Data and Software Engineering (ICODSE), 2014 International Conference on, (1-6). IEEE.

Mendes, N., Neto, A. A., Durães, J., Vieira, M. & Madeira, H. (2008). Assessing and comparing security of web servers. Dependable Computing, 2008. PRDC'08. 14th IEEE Pacific Rim International Symposium on, (313-322). IEEE.

Mills, C. (2016, 03/02). What is a web server? Haettu 05/062016 osoitteesta https://developer.mozilla.org/en-US/Learn/Common_questions/What_is_a_web_server

Mozilla Foundation. (2016, 2016). About JavaScript. Haettu 4/102016 osoitteesta https://developer.mozilla.org/en-US/docs/Web/JavaScript/About_JavaScript

Mozilla Foundation. (2016, 26.2). Document object model (DOM). Haettu 4/102016 osoitteesta https://developer.mozilla.org/en-US/docs/Web/API/Document_Object_Model

Mozilla Foundation. (2016, 6.4). Html. Haettu 04/102016 osoitteesta https://developer.mozilla.org/en-US/docs/Web/HTML

Myers, G. J., Sandler, C. & Badgett, T. (2011). The art of software testing John Wiley & Sons.

OWASP. (2009, 02/24). Improper error handling. Haettu 05/132016 osoitteesta https://www.owasp.org/index.php/Improper_error_handling

OWASP. (2015, 4/7). Insecure configuration management. Haettu 5/242016 osoitteesta https://www.owasp.org/index.php/Insecure_Configuration_Management

OWASP. (2016, 01/08). Cross-site request forgery (CSRF) prevention cheat sheet. Haettu 05/142016 osoitteesta https://www.owasp.org/index.php/Cross-Site_Request_Forgery_(CSRF)_Prevention_Cheat_Sheet

OWASP. (2016, 03/27). XSS (cross site scripting) prevention cheat sheet. Haettu 05/132016 osoitteesta https://www.owasp.org/index.php/XSS_(Cross_Site_Scripting)_Prevention_Cheat_Sheet

OWASP, T. (2013). 10: Ten most Critical Web Application Security Risks,

Rafique, S., Humayun, M., Hamid, B., Abbas, A., Akhtar, M. & Iqbal, K. (2015). Web application security vulnerabilities detection approaches: A systematic mapping study. Software Engineering, Artificial Intelligence, Networking and Parallel/Distributed Computing (SNPD), 2015 16th IEEE/ACIS International Conference on, (1-6). IEEE.

Rooney, A. (2006). JDBC technology. Foundations of Java for ABAP Programmers, , 97-106.

Scambray, J., Shema, M. & Sima, C. (2006). Hacking exposed web applications.

Schrank, M., Braun, B., Johns, M. & Posegga, J. (2010). Session fixation-the forgotten vulnerability? Sicherheit, (341-352).

Shar, L. K. & Tan, H. B. K. (2012). Auditing the XSS defence features implemented in web application programs. Software, IET, 6(4), 377-390.

Shklar, L. & Rosen, R. (2003). Web application architecture. JohnWilley & Sons, Ltd,

Stock, B., Lekies, S., Mueller, T., Spiegel, P. & Johns, M. (2014). Precise client-side protection against dom-based cross-site scripting. 23rd USENIX Security Symposium (USENIX Security 14), (655-670).

Stock, B., Pfistner, S., Kaiser, B., Lekies, S. & Johns, M. (2015). From facepalm to brain bender: Exploring client-side cross-site scripting. Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security, (1419-1430). ACM.

Stuttard, D. & Pinto, M. (2011). The web application hacker's handbook: Finding and exploiting security flaws John Wiley & Sons.

Sullo, C. & Lodge, D. (2011). Nikto2,

Sun, F., Xu, L. & Su, Z. (2011). Static detection of access control vulnerabilities in web applications. USENIX Security Symposium,

The Django Software Foundation. (2016, 12/01). Error reporting. Haettu 05/062016 osoitteesta https://docs.djangoproject.com/en/1.9/howto/error-reporting/

The Django Software Foundation. (2016, 12/01). Using the django authentication system. Haettu 05/102016 osoitteesta https://docs.djangoproject.com/en/1.9/topics/auth/default/

Thomas, S., Williams, L. & Xie, T. (2009). On automated prepared statement generation to remove SQL injection vulnerabilities. Information and Software Technology, 51(3), 589-598.

Vogt, P., Nentwich, F., Jovanovic, N., Kirda, E., Kruegel, C. & Vigna, G. (2007). Cross site scripting prevention with dynamic data tainting and static analysis. Ndss, (12).

W3C. (2013, 05/28). HTML elements. Haettu 05/062016 osoitteesta http://www.w3.org/TR/html-markup/elements.html

Wassermann, G. & Su, Z. (2008). Static detection of cross-site scripting vulnerabilities. Software Engineering, 2008. ICSE'08. ACM/IEEE 30th International Conference on, (171-180). IEEE.

Weinberger, J., Saxena, P., Akhawe, D., Finifter, M., Shin, R. & Song, D. (2011). A systematic analysis of XSS sanitization in web application frameworks. Computer Security–ESORICS 2011 (s. 150-171) Springer.