Johannes Ylipiha

Request Forgery -haavoittuvuuden paikkaus pitkälle kehitetyssä järjestelmässä

Metropolia Ammattikorkeakoulu

Insinööri (AMK)

Ohjelmistotekniikka

Insinöörityö

13.5.2015

Metropolia

Tekijä(t)	Johannes Ylipiha
Otsikko	Request Forgery -haavoittuvuuden paikkaus pitkälle kehitetyssä järjestelmässä
Sivumäärä	31 sivua
Aika	13. toukokuuta 2015
Tutkinto	Insinööri (AMK)
Koulutusohjelma	Tietotekniikan koulutusohjelma
Suuntautumisvaihtoehto	Ohjelmistotekniikka
Ohjaaja(t)	Järjestelmäasiantuntija Janne Kaavi Lehtori Simo Silander

Insinöörityön tarkoitus oli poistaa tilaajayrityksen pitkään kehitetystä järjestelmästä CSRF-haavoittuvuus. Jatkuvasti käytössä oleva ja pitkään kehitetty järjestelmä asettaa omat haasteensa paikkauksen toteutukselle. Tämän työn avulla lukija saa käsityksen siitä, minkälaisia asioita tulee ottaa huomioon, kun CSRF-suojaus toteutetaan järjestelmään jälkeenpäin.

Työssä käydään lyhyesti läpi CSRF-hyökkäyksen kannalta olennainen HTTP-protokolla ja kerrotaan, kuinka istunnot yleensä muodostetaan protokollan päälle, joka itsessään on tilaton. Tämän jälkeen esitellään CSRF-hyökkäyksen toimintaperiaate ja käydään läpi yleisimmät suojautumistavat.

Suojauksen toteutus on kuvattu tässä työssä vaihe vaiheelta. Tämän avulla lukija saa käsityksen niistä haasteista, joita esiintyy, kun pitkälle kehitettyyn järjestelmään toteutetaan suojaus jälkeenpäin. Työn perusteella voidaan todeta, että järjestelmässä esiintyvän CSRF-haavoittuvuuden paikkaus on jälkikäteen tehtynä paljon työläämpää, kuin järjestelmän suunnittelu alunperin tietoturvalliseksi.

Avainsanat	CSRF, verkkopalvelun tietoturva, jatkokehittäminen

Authors(s)	Johannes Ylipiha
Title	Patching Cross-site Request Forgery Vulnerabilities in Late Development Phase
Number of Pages Date	31 pages 13. toukokuuta 2015
Degree	Bachelor of Engineering
Degree Programme	Degree Programme in Information Technology
Specialisation option	Software Engineering
Instructor(s)	Janne Kaavi, Systems specialist Simo Silander, Senior lecturer

The aim of this study was to patch CSRF-vulnerabilities in a customer organization's web-application that was at a late stage of development. Patching of a web-application that is constantly in use and has a long development history is a challenging task. With the present study, the reader can get an idea of the different things that need to be taken into account when developing CSRF-defense afterwards.

In the study, the HTTP-protocol that is relevant to a CSRF-attack is explained briefly. Establishing sessions over otherwise stateless HTTP-protocol are also explained. After this, the principle of a CSRF-attack is explained and different methods of defending against it are evaluated.

Development of the defense in this case is explained step by step. After reading the paper the reader should have an idea of what kind of challenges can be expected when developing CSRF-defense to a system that is at a late stage of development. Based on the study, it can be concluded that developing CSRF-defense after the initial deployment of a system is much more time consuming than developing an initially secure system.

Keywords	CSRF, Web-application security, further developing

Sisältö

Lyhenteet ja käsitteet

AJAX	Asynchronous JavaScript And XML. Joukko web-sovelluskehitystekniikoita, joilla web-sovelluksista voi tehdä vuorovaikutteisempia.
CAPTCHA	Completely Automated Public Turing test to tell Computers and Humans Apart. Kuvavarmennus, jolla voidaan varmistua, että palvelun käyttäjä on ihminen.
CSRF	Cross-Site Request Forgery. Hyökkäystyyppi, jossa hyökkääjä lähettää käyttäjältä pyyntöjä palvelimelle.
JSP	JavaServer Pages. Sun Microsystemsin kehittämä palvelinpuolen ohjelmointitapa dynaamisten verkkosivujen toteutukseen.
JSP-sivu	JSP-page. Sivu, joka sisältää JSP-merkintää. Voidaan tulkata Java-koodiksi.
JSP-osanen	JSP-fragment. JSP-koodia sisältävä tiedosto, joka ei yksinään ole tulkattava JSP-sivu.
kaksoislähetys	CSRF-tunnisteen välitystapa, jossa sama tunniste välietään palvelimelle sekä evästeenä, että pyynnön parametrina tai otsakkeena.
käsittelijäluokka	Handler. Servletin laajennus, joka on tehty jotain normaalista poikkeavaa tarkoitusta varten.
SaaS	Software as a Service. Ohjelmisto, joka tarjotaan palveluna.
sanitointi	Prosessi, jossa käyttäjältä saadusta syötteestä poistetaan palvelun kannalta mahdollisesti haitallinen koodi.
Servlet	Java-luokka, joka vastaanottaa pyyntöjä ja luo niihin vastauksia.
XSS	Cross-Site Scripting. Haavoittuvuustyyppi, jossa hyökkääjä lisää haitallista koodia, yleensä JavaScriptiä, kohdesivustolle.

1 Johdanto

Netissä tehdyt rikokset ovat yleistyneet ja muuttuneet ajan saatossa entistä vakavammiksi. Tietoa ja työkaluja, joiden avulla hyökkäyksiä voi tehdä, on helpommin saatavilla kuin aikaisemmin, eikä hyökkääminen enää aina vaadi taitoja. Yrityksiin kohdistuvissa tietomurroissa mootivit vaihtelevat rahasta maineeseen. Myös niin sanottu haktivismi, eli tietoverkossa tapahtuva aktivismi on suuressa kasvussa. [1.]

Samalla, kun verkossa tehtyjen tietomurtojen määrä on kasvanut, tietotekniikka ja tietoturvallisuus ovat monimutkaistuneet, eikä normaalin käyttäjän voi enää olettaa ottavan kauheasti vastuuta omasta tietoturvasta [1]. Vastuu palveluiden turvallisuudesta on ennen kaikkea palveluiden tarjoajilla. Suomessa henkilötietolaki edellyttää, että henkilötietojen käsittelyssä noudatetaan huolellisuutta ja hyvää tietojenkäsittelytapaa [2]. Tällä hetkellä EU:n komission asetus 611/2013 edellyttää, että henkilötietoja käsittelevät teleoperaattorit tai televiestintäpalveluita tarjoavat tahot ilmoittavat tietoturvaloukkauksista toimivalle kansalliselle viranomaiselle. Tapauksissa, joissa tietoturvaloukkauksesta on haittaa tilaajan tai henkilön yksityisyydelle tai henkilötiedoille, ilmoitus on tehtävä myös tilaajalle tai henkilölle. [3.] Euroopan komissiossa on käsittelyssä tietosuojasääntöjen uudistus, joka laajentaisi toteutuessaan ilmoitusvelvollisuuden koskemaan kaikkia rekisterinpitäjiä [4]. Käytännössä tämä tarkoittaa kaikkia yrityksiä, jotka käsittelevät henkilötietoja. Verkossa toimivat sivustot ja palvelut ovat aina alttiita hyökkäyksille, eikä mikään järjestelmä ole murtamaton. Palvelun tarjoajan on syytä pitää palvelun suojaustaso niin korkealla, ettei heikosta suojaustasosta tule hyökkääjälle motiivia.

Tämän insinöörityön tilaaja on Likeit Solutions Oy, suomalainen ohjelmistoyritys, joka kehittää ja myy vaativiin henkilöstöhallinnon tehtäviin kehitettyä järjestelmää asiakkailleen SaaS-palveluna. Suurimman osan asiakasryhmistä muodostavat erilaiset henkilöstöpalveluyritykset, joiden tarpeet vaihtelevat rekrytoinnista ja työtuntien välityksesta palkanlaskentaan ja kirjanpitoon. Järjestelmä on kriittinen osa asiakkaiden liiketoimintaa, eikä sen toimintavarmuudessa tai tietoturvassa saa olla puutteita. Lokakuussa 2013 julkaistussa diplomityössään Janne Kaavi kartoitti tämän järjestelmän tietoturvaa. Työn tuloksena järjestelmästä löydettiin useita haavoittuvuuksia. Tämä insinöörityö keskittyy

yhden haavoittuvuustyypin, Cross-Site Request Forgeryn (CSRF) toimintaperjaatteisiin ja siltä suojautumiseen. CSRF on hyökkäystyyppi, jossa hyökkääjä hyödyntää kohteena olevan palvelun luottamusta käyttäjään. Hyökkääjä suorittaa täysin laillisia toimintoja hyödyntäen kirjautuneen käyttäjän käyttöoikeuksia. Luvussa 2.2 kuvaillaan tarkemmin CSRF-hyökkäyksen toimintaperiaate ja esitellään muutamia tapoja, joilla hyökkäys voidaan käytännössä toteuttaa.

CSRF-haavoittuvuuden kaltainen verkkopalvelun haavoittuvuus on kuvattu ensimmäisen kerran vuonna 2000 [5]. Selainten kehittyessä myös CSRF-suojaukset ovat muuttuneet, eivätkä kaikki ennen toimineet hyökkäykset toimi nykyisissä selaimissa. Eri menetelmien kriittinen arviointi on tämän työn keskeinen ja välttämätön osa hyvän ja luotettavan lopputuloksen aikaansaamiseksi.

Tämän insinöörityön päätavoite on suojata kohdejärjestelmä CSRF-hyökkäyksiä vastaan. Samalla, kun järjestelmään tehdään suojauksen edellyttämiä muutoksia, pyritään järjestelmän koodia yhtenäistämään ja tekemään siitä paremmin hallittavaa. Tämän saavuttamiseksi muutoksien yhteydessä kaikkiin käyttöliittymiin otetaan käyttöön järjestelmässä oleva ohjelmistokehys siltä osin, kuin se ei ole vielä käytössä. Ohjelmistokehystä tullaan laajentamaan CSRF-suojauksen vaatimalla tavalla. Järjestelmässä mahdollisesti piilevät CSRF-suojauksen vahvuuteen vaikuttavat tietoturva-aukot eivät kuulu tämän työn alueeseen, eikä niihin tulla tämän työn aikana puuttumaan.

2 CSRF-hyökkäyksen toimintaperiaate

2.1 HTTP-protokolla ja istunnot

Yksi CSRF-suojausten kulmakivi on verkkopalvelimen ja käyttäjän välisen tiedon salaus. Suojaukset olettavat, että käyttäjälle välitetty tieto on vain palvelimen ja käyttäjän tiedossa ja siten luotettava tapa tunnistaa käyttäjä. Liikenteen suojaukseen käytettävät protokollat ovat kriittinen osa internetin toimintaa. Sen vuoksi salausprotokollat, kuten Transport Layer Security, ovat jatkuvasti äärimmäisen kriittisen tarkastelun alaisena. Protokollissa havaittavat puutteet huomataan ja korjataan nopeasti. Tästä syystä oikein toteutetun salausprotokollan käytön voidaan katsoa luotettavasti suojaavan käyttäjän selaimen ja verkkopalvelun välisen tietoliikenteen.

CSRF-hyökkäyksen toiminnan ymmärtämiseksi on ensin ymmärrettävä, miten selain ja verkkopalvelin keskustelevat keskenään. Selainten ja palvelimien väliseen kommunikointiin käytettävä protokolla on HTTP [6]. Protokolla toimii yksinkertaistettuna siten, että asiakasohjelma, esimerkiksi internetselain, avaa TCP-yhteyden palvelimelle ja lähettää pyynnön, johon palvelin lähettää vastauksen. Pyyntö ja vastaus ovat rakenteeltaan samanlaisia siten, että kumpikin koostuu otsakeosasta ja viestin rungosta. Viestin runko voi sisältää esimerkiksi lomakkeen arvoja tai tiedoston dataosuuden. Otsakkeet muodostuvat avain-arvo-pareista, jotka kertovat tietoja esimerkiksi viestistä, sen sisällöstä tai sen käsittelystä.

```
GET /wiki/HTTP HTTP/1.1
Host: fi.wikipedia.org
Accept:text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,*/*
    ;q=0.8
Accept-Encoding:gzip, deflate, sdch
Accept-Language:en-US,en;q=0.8
Cache-Control:max-age=0
Connection:keep-alive
User-Agent:Mozilla/5.0 (Windows NT 6.1; WOW64) Chrome/39.0.2171.65
Referer: http://fi.wikipedia.org/w/wiki.phtml?title=HTTP
```

Koodiesimerkki 2.1: Wikipedian HTTP-artikkelin lataamiseen käytetyn HTTP-pyynnön otsakeosa.

Koodiesimerkki 2.1 kuvaa palvelimelle lähetettävän HTTP-pyynnön otsakeosaa. Ensimmäisellä rivillä on kerrottu pyynnön HTTP-metodi, esimerkissä GET, jonka jälkeen tulee pyydettävä resurssi /wiki/HTTP ja käytettävä protokollan versio, tässä tapauksessa HTTP:n versio 1.1. CSRF-suojauksen kannalta viimeisen rivin Referer-otsake on mielenkiintoinen. Referer-otsake kertoo palvelimelle, miltä sivulta pyyntö on lähtöisin, eli mikä oli edellinen sivu, josta tälle sivulle linkattiin. Selaimet luovat pyyntöihin otsakkeet automaattisesti, mutta niitä on mahdollista myös määritellä itse, mikäli pyynnöt lähetetään AJAX-tekniikkaa käyttäen. Selainten itse luomia otsakekenttiä, kuten Referer-kenttää, ei ole mahdollista muokata JavaScriptiä apuna käyttäen. Mikäli ladattavalla sivulla on muita resursseja kuten kuvia, selain lataa ne automaattisesti erillisillä pyynnöillä.

HTTP-protokolla on tilaton, [6, s. 1] eikä palvelin pysty tunnistamaan käyttäjiä pyyntöjen välillä pelkän protokollan avulla. Jotta esimerkiksi sisäänkirjautumista ei tarvitisi suorittaa uudestaan jokaisen pyynnön välissä, ensimmäisen sisäänkirjautumisen yhteydessä luodaan käyttäjälle istunto hyödyntäen sovelluskerroksen ominaisuuksia. Yleisin tapa luoda istunto on tallentaa käyttäjän selaimeen sisäänkirjautumisen yhteydessä eväste. Evästeeseen on tallennettu tunniste, jonka perusteella käyttäjä voidaan jatkossa tunnistaa. Evästeet sisältävät avain-arvo-pareja ja metadataa. Metadatan perusteella selain päättää, lähetetäänkö evästeessä olevat avain-arvo-parit takaisin palvelimelle pyynnön yhteyessä. [7, s. 3.] Selain suorittaa tämän toiminnon automaattisesti.

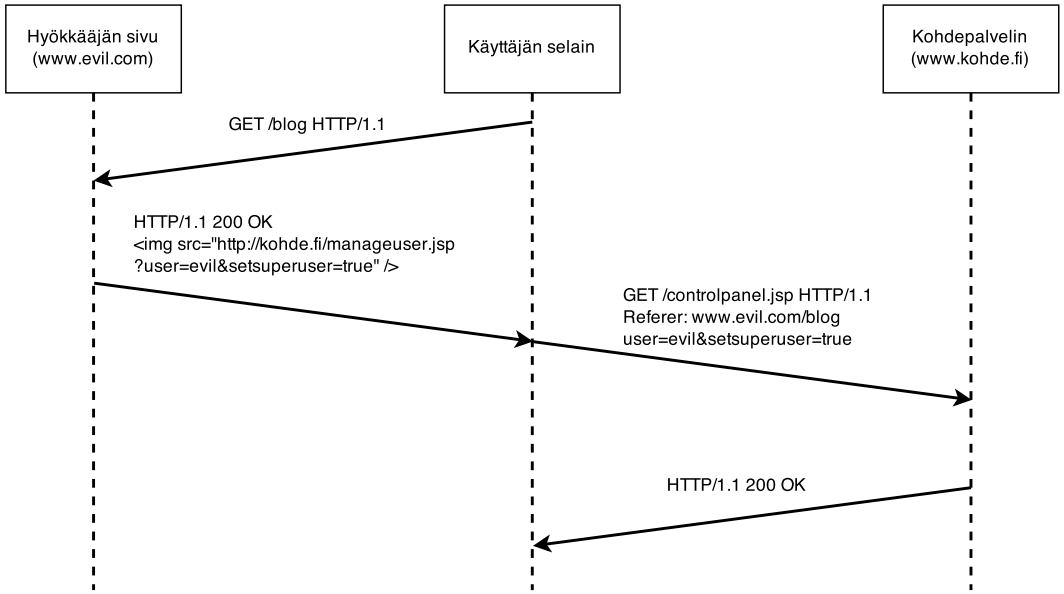
2.2 Hyökkäyksen kuvaus

CSRF-hyökkäyksessä on kyse palvelimen käyttäjään kohdistaman luottamuksen hyödyntämisestä. Yleensä hyökkääjä huijaa käyttäjän selaimen lähettämään pyynnön hyökkääjän parametreilla kohteena olevalle palvelimelle. Palvelimelle pyyntö näyttää tulevan suoraan käyttäjältä. Hyökkääjällä voi olla muun muassa seuraavia tavoitteita:

- Kirjautuneen käyttäjän istunnon hyödyntäminen. Mikäli istunnon tunniste on toteutettu sellaisella tavalla, jossa selain lähettää tunnisteet automaattisesti, ne lähetetään myös CSRF-hyökkäyksen aikaansaaman pyynnön mukana. Tälläisiä tapoja ovat esimerkiksi evästeeseen tallennettu istunnontunniste tai Basic Auth -tunnukset.
- Palomuurin kiertäminen. CSRF-hyökkäyksen aikaansaama pyyntö kohdepalvelimelle tulee suojatussa verkossa olevalta uhrilta, eikä pyyntö siten kulje julkisen internetin ja suojatun verkon välissä olevan palomuurin läpi.
- Intranetissä tai muussa internettiin näkymättömässä sijaitsevaan kohteeseen hyökkäys. Vaikka kohde ei olisi julkiseen internettiin nähtävissä, CSRF-hyökkäyksen avulla on mahdollista lähettää kohteeseen pyyntöjä, mikäli uhrina toimiva käyttäjä näkee kohteen. [8.]

CSRF-hyökkäys ei missään nimessä ole pelkästään istunnonkaappaus, vaikka se yleensä esiintyy nimenomaan siinä yhteydessä. Se ei myöskään ole rajoitettu toimimaan vain internetselaimilla. Hyökkääjä voi hyödyntää hyökkäyksen käynnistämiseen mitä tahansa formaattia, joka mahdollistaa pyyntöjen lähettämisen automaattisesti, esimerkiksi skriptien avulla. Tällaisia formaatteja voivat olla esimerkiksi Word- tai Excel-dokumentit. On myös mahdollista, että ohjelma, joka hyödyntää XML-dokumentteja, lähettää automaattisesti tiettyjä elementtejä käsiteltäessä pyynnön ulkopuolisten dokumenttien lataamiseksi ja sisällyttämiseksi osaksi alkuperäistä dokumenttia. [5.] Tässä työssä keskitytään kuitenkin pelkästään selaimen ja palvelimen väliseen kanssakäymiseen. Verkkopalvelun suojauksen kannalta ei ole merkitystä sillä, mistä pyyntö on lähtöisin, koska käytettävät suojaustavat ovat pyynnön lähteestä riippumattomia.

Hyökkäys käynnistyy, kun käyttäjä vierailee sivulla, jossa on CSRF-hyökkäyksen käynnistävä elementti. Käyttäjän selain suorittaa elementin, joka aiheuttaa automaattisesti pyynnön hyökkäyksen kohteena olevalle palvelimelle.



Kuva 1: Cross-Site Request Forgery -hyökkäyksen kulku

CSRF-hyökkäyksen sisältävä pyyntö on käyttäjän selaimen kohdepalvelimelle lähettämä. Palvelin ei näe, mikä toiminto on aikaansaanut tämän pyynnön. Tästä syystä suojaamattomaan järjestelmään kohdistettujen CSRF-hyökkäysten havaitseminen on vaikeaa. CSRF-hyökkäys on käytännössä yksisuuntainen, eikä hyökkääjä pysty normaalisti lukemaan palvelimen vastausta CSRF-hyökkäyksen aikaansaamaan pyyntöön. Hyökkäyksen käynnistäminen on teknisesti yksinkertaista ja helpoimmillaan se onnistuu esimerkiksi kuva-elementin avulla.

```
<img src="https://kohdesivusto.net/toiminto.jsp?hyokkays=parametri">
```

Koodiesimerkki 2.2: Hyökkäyksen käynnistys kuvan avulla.

Esimerkissä on HTML-kielen kuvan esittämiseen käytettävä merkintä. Merkinnän src-attribuutti kertoo selaimelle, mistä kuva on tarkoitus ladata. [9] Selain lähettää automaattisesti pyynnön src-attribuutin osoittamalle palvelimelle määritellyillä parametreilla, jotta kuva saadaan ladattua ja esitettyä käyttäjälle. Sisällyttämällä hyökkäyksen src-attribuuttiin hyökkääjä huijaa käyttäjän selaimen lähettämään pyynnön kohteena olevalle palvelimelle sivun latauksen yhteydessä. Kuvan lataamiseen käytettävissä pyynnöissä käytetään GET-metodia, joka on HTTP:n määrityksissä niin sanottu turvallinen metodi. Käytännössä GET-metdoilla lähetetty pyyntö ei saisi aiheuttaa muutoksia palvelimella. [6, s. 51.] Hyökkäys määritysten mukaan konfiguroitua palvelinta vastaan ei onnistu pelkän kuva-elementin avulla, sillä väärä pyynnön metodi aiheuttaa pyynnön hylkäämisen.

Oikea palvelinkonfiguraatio ei kuitenkaan käytännössä hankaloita hyökkäyksen tekemistä, koska hyökkäyksessä on mahdollista käyttää myös muita HTTP-metodeja. Palvelimella muutoksen aiheuttaviin pyyntöihin käytettävä metodi on määritysten mukaan nimeltään POST [6, s. 54]. CSRF-hyökkäyksen tapauksessa sivustolle voidaan upottaa esimerkiksi automaattisesti lähetettävä lomake, jonka selain lähettää automaattisesti sivunlatauksen yhteydessä.

```
<html>
  <body onload="document.forms[0].submit();">
    <form action="https://kohdesivusto.net/toiminto.jsp" method="POST">
      <input type="hidden" name="hyokkays" value="pyynto">
    </form>
  </body>
</html>
```

Koodiesimerkki 2.3: Hyökkäys automaattisesti lähetettävän lomakkeen avulla

Koodiesimerkissä on yksinkertainen HTML-lomake, jonka lähetys on automatisoitu body-elementin onload-attribuutissa. Käytännössä CSRF-hyökkäyksen käynnistämiseen on lukuisia erilaisia tapoja, joista edellä kuvatut ovat vain kaksi esimerkkiä.

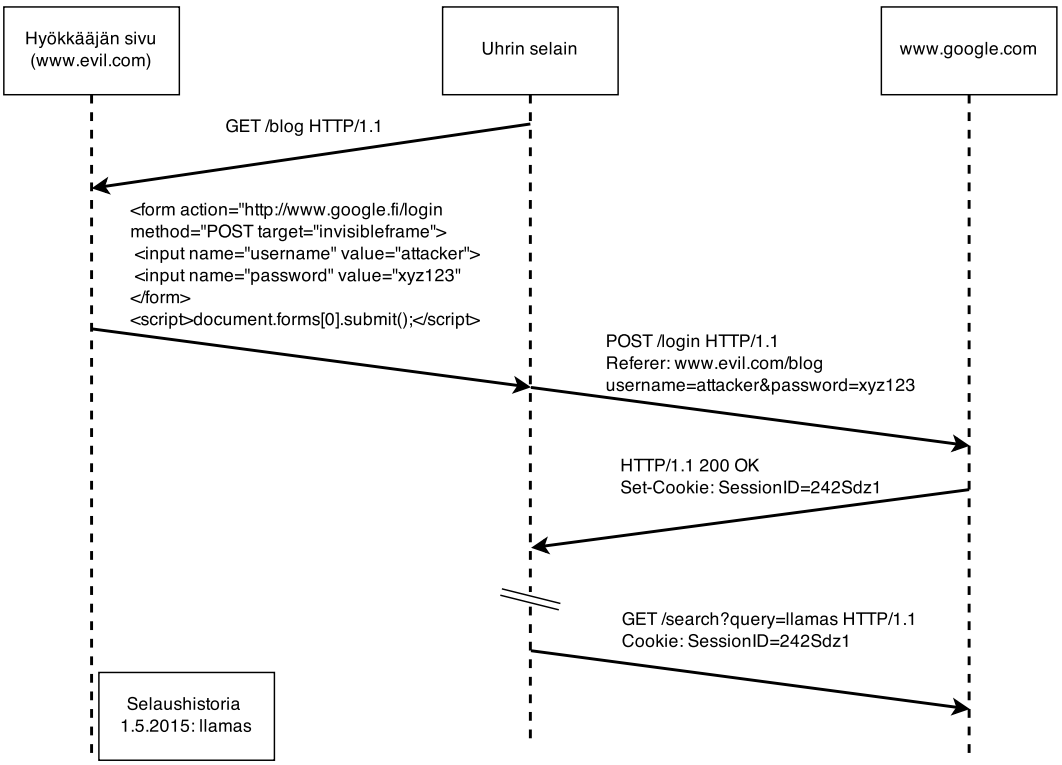
Kuvien sisällyttäminen esimerkiksi keskustelufoorumeille onnistuu yleensä todennäköisemmin, kuin automaattisesti lähettävän lomakkeen sisällytys. Tämä johtuu siitä, että monet sivustot kuten keskustelufoorumit sanitoivat [10] käyttäjiltä otettavat syötteet ennen niiden näyttämistä sivulla. Sanitoinnilla pyritään estämään sivustolle kohdistuvia Cross-Site Scripting (XSS)-hyökkäyksiä. XSS-hyökkäyksessä hyökkääjä sisällyttää sivustolle JavaScriptiä, joka ajetaan myöhemmin automaattisesti kaikkien sivustolla vierailevien käyttäjien selaimissa. Sanitointiprosessi on vaikeaa toteuttaa luotettavasti, koska erilaisia tapoja XSS-hyökkäyksen toteutukselle on lukuisia. Tavallinen kuvaelementti pelkällä src-attribuutilla ei ole uhka forumjärjestelmälle, joten ne on usein sallittu.

Yksi parhaiten dokumentoitu esimerkki XSS-haavoittuvuuden hyödyntämisestä CSRF-suojauksen kierrossa on MySpace-sivustolla aggressiivisesti levinnyt Samy-worm-mato. Madon kirjoittanut Samy Kamkar hyödynsi koodissaan tiettyjen selainten ominaisuutta tulkata väärin kirjoitettu JavaScript-koodi suorituskelvolliseksi koodiksi. Mato levisi saastuneen käyttäjän MySpace-sivulta, sivulla vierailevien, MySpace-palveluun kirjautuneiden käyttäjien MySpace-sivuille. Leviäminen oli mahdollista, koska mato kykeni saa-

maan haltuunsa sivustolla olevan CSRF-tunnuksen ja käyttämään sitä omissa pyynnöissään. [11.] Hyökkäys on hyvä esimerkki käyttäjien syötteiden sanitoinnin vaikeudesta sekä siitä, kuinka CSRF-suojaus pystytään kiertämään XSS-haavoittuvuuden avulla.

2.3 Sisäänkirjautumiseen kohdistuva hyökkäys

CSRF-hyökkäys mielletään yleensä hyökkäykseksi, jonka tavoitteena on lähettää tilaa muuttavia pyyntöjä kohteena olevalle palvelimelle. Vähemmän tunnettu käyttötapa on aiheuttaa tilan muutoksia käyttäjän selaimeen. Yleisin käyttäjän selaimeen vaikuttava hyökkäyksen muoto on käyttäjän sisäänkirjaaminen palveluun hyökkääjän nimissä. Sisäänkirjautumiseen kohdistuva CSRF-hyökkäys toimii samalla tavalla kuin normaali CSRF-hyökkäys, mutta sen tavoitteena on kirjata käyttäjä sisään hyökkääjän tunnuksilla.



Kuva 2: Sisäänkirjautumiseen kohdistuvan CSRF-hyökkäyksen kulku [12, s. 4].

Hyökkääjä huijaa käyttäjän selaimen lähettämään sisäänkirjautumispyynnön kohdepalvelimelle hyökkääjän käyttäjätilin tunnuksilla. Palvelin yleensä vastaa onnistuneeseen pyyntöön vastauksella, jossa on Set-Cookie-otsake, joka asettaa käyttäjän selaimeen evästeen, jonka avulla palvelin jatkossa tunnistaa käyttäjän. Sisäänkirjaus suoritettiin hyökkääjän

tunnuksilla, joten kaikki käyttäjän jatkossa tekemät toimenpiteet sivustolla suoritetaan hyökkääjän nimissä.

Sisäänkirjautumisen toteuttaminen hyökkääjän nimissä saattaa äkkiseltään kuulostaa vaarattomalta toimenpiteeltä, mutta palvelusta riippuen sillä saattaa olla vakavia seurauksia. Esimerkiksi hakukoneet kuten Google ja Bing mahdollistavat käyttäjän hakuhistorian tallentamisen myöhempää tarkastelua varten. Sisäänkirjaukseen kohdistetun CSRF-hyökkäyksen jälkeen kaikki tehdyt haut kirjautuvat hyökkääjän tilille. Tämän jälkeen hyökkääjällä on pääsy käyttäjän hakuhistoriaan, jota voidaan käyttää esimerkiksi kiristykseen tai vakoiluun.

Sisäänkirjautuminen ei ole yleensä normaalin CSRF-suojauksen piirissä, sillä suurin osa CSRF-suojauksista on suoraan tai epäsuorasti käyttäjän istuntoon liittyviä. Monet palvelut käyttävät sisäänkirjautumisen suojaukseen Referer-otsakkeen tarkistusta. Se on helppo ja kohtuullisen luotettava tapa, sillä sisäänkirjaus toteutetaan yleensä salatun yhteyden yli. Selaimet lähettävät Referer-otsakkeen todennäköisemmin, kun pyyntö tapahtuu suojatun yhteyden yli.

Hyökkäyksistä, joissa sisäänkirjautumiseen kohdistuvaa CSRF-hyökkäystä on hyödynnetty, ei ole dokumentoitu paljoa. Yksi dokumentoitu CSRF-haavoittuvuus on löydetty vuonna 2008 YouTube-palvelusta. Hyökkäys hyödynsi YouTuben flash-soittimen huonosti määriteltyä sallittujen verkko-osoitteiden käytäntöä. Flash-selain määrittelee verkkoalueet, joista ladattujen tiedostojen toisto on sallittua crossdomain.xml tiedostossa. YouTuben Flash-soitin salli kaikista google.com aliverkoissa sijaitsevien tiedostojen toistamisen. Hyökkäyksessä liitetiedosto tallennettiin Gmailin liitetiedostoksi, jolloin tiedoston latauslinkki sijaitsi osoitteessa, joka oli muotoa *.google.com. Gmail-palvelun liitetiedostoja voi ladata vain palveluun kirjautunut käyttäjä. Hyökkääjä hyödynsi Gmail-palvelun sisäänkirjautumisessa olevaa CSRF-haavoittuvuutta ja kirjasi kohteena olevan käyttäjän haluamalleen Gmail-tilille, jonne oli tallennettu hyökkääjän haluama flash-tiedosto. Seuraavaksi hyökkääjä sisällytti Gmail-palveluun tallentamansa flash-tiedoston latauslinkin haluamalleen sivulle ja huijasi käyttäjän vierailemaan tällä sivulla. Kun YouTube-palveluun sisäänkirjautunut käyttäjä vieraili sivulla, hyökkääjä sai luku- ja kirjoitusoikeudet hyökkääjän YouTube-tilin tietoihin. [13.]

3 Järjestelmän suojausmenetelmät

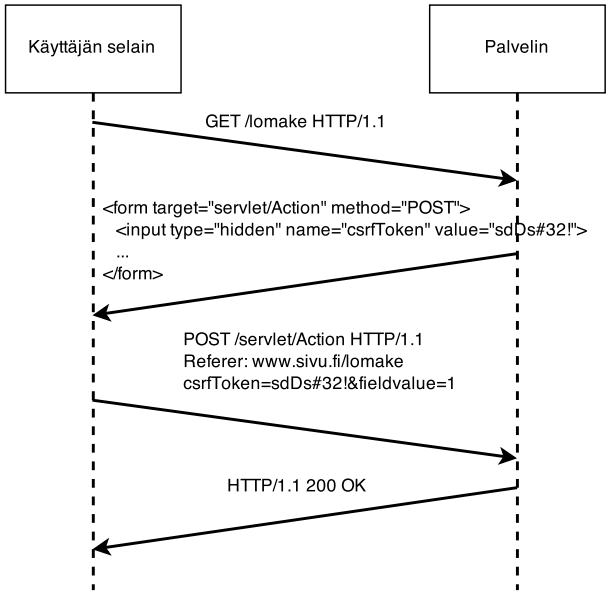
Tässä luvussa esitellään yleisimmät tavat suojata verkkopalvelu CSRF-hyökkäystä vastaan. Ohjelmoinnissa moniin tilanteisiin löytyy valmis suunnittelumalli ja niiden hyödyntäminen on usein kannattavaa. Suunnittelumallit tarjoavat yleensä ratkaisumallin, joka perustuu käytännön kokemukseen ratkaisun toimivuudesta. Tämä on erityisen hyödyllistä tietoturvaan liittyvissä ratkaisuissa. Ohjelmoijalla ei yleensä ole tarpeeksi syvää ymmärrystä järjestelmään kohdistuvista uhista, jolloin omaan ratkaisumalliin perustuvan suojauksen heikkouksia on vaikea huomata.

CSRF-hyökkäykseltä suojautumisen perusajatus on tehdä palvelimelle lähetettävistä pyynnöistä sellaisia, että palvelin pystyy luotettavasti erottamaan sellaiset pyynnöt, jotka käyttäjä on halunnut lähettää, pyynnöistä, jotka ovat hyökkääjän aikaansaamia.

3.1 Synchronizer Token Pattern

Käytännössä kaikki yksinään luotettavat suojaustavat ovat erilaisia variaatioita Synchronizer Token Patternista. Näiden lisäksi on muutamia suojaustapoja, jotka eivät yksinään ole riittäviä suojaamaan CSRF-hyökkäystä vastaan, mutta jotka tarjoavat yhdessä luotettavamman suojauksen kanssa lisäsuojaa. Synchronizer Token Pattern toimii yksinkertaistettuna seuraavasti:

1. Kirjautumisen yhteydessä palvelin tallentaa käyttäjän istuntoon satunnaisen merkkijonon (CSRF-tunniste).
2. Palvelin välittää CSRF-tunnisteen käyttäjälle sivun latauksen yhteydessä.
3. Käyttäjän selain välittää CSRF-tunnuksen takaisin palvelimelle muutoksia tekevän pyynnön yhteydessä.
4. Palvelin tarkistaa CSRF-tunnisteen ja edellyttää, että se vastaa istuntoon tallennettua CSRF-tunnistetta ennen pyynnön käsittelyä.



Kuva 3: CSRF-tunnisteen välitys pyyntöjen yhteydessä

Kuvassa 3 on kuvattu, kuinka CSRF-tunniste välitetään käyttäjän selaimelle ensimmäisen pyynnön yhteydessä ja kuinka tunniste lähetetään palvelimelle osana toista pyyntöä. Palvelin voi välittää CSRF-tunnisteen käyttäjälle esimerkiksi tallennettuna suoraan sivulle, vastausviestin HTTP-otsakkeena tai ylimääräisenä evästeenä. Suojauksen kannalta olennainen seikka on, että tunnisteen välitystapa takaisin palvelimelle pyynnön yhteydessä on jokin muu kuin eväste. Tämä on tärkeää, sillä selaimet välittävät evästeet palvelimelle myös hyökkääjän käynnistämien pyyntöjen yhteydessä. Tunnistetta ei saa sisällyttää kolmansille osapuolille lähetettäviin pyyntöihin, koska sen pysyminen salassa on CSRF-suojauksen kannalta välttämätöntä. [8, s. 10.] Samasta syystä CSRF-tunniste pitää olla pituudeltaan ja ennustettavuudeltaan sellainen, ettei sen murtaminen tai arvaaminen ole käytännössä mahdollista. Nämä ehdot täyttävä merkkijono voidaan muodostaa luotettavalla näennäissatunnaislukugeneraattorilla. Kun CSRF-tunniste on yhtä hankala murtaa kuin istunnon tunniste, on istunnon tunniste hyökkääjälle yleensä houkuttelevampi kohde. Saamalla haltuunsa istunnon tunnisteen, hyökkääjä saa lähes kaikissa tapauksissa paremman kontrollin itse sivustoon. CSRF-tunnisteen vaihtamisella useammin, kuin istunnon tunniste vaihdetaan ei paranneta merkittävästi palvelun suojaa. Kertakäyttöisellä CSRF-tunnisteella voidaan kuitenkin esimerkiksi estää saman lomakkeen lähetys useaan kertaan.

CSRF-suojaukset voidaan jakaa tilattomiin ja tilallisiin suojauksiin. Näiden kahden suojaustavan ero on siinä, onko CSRF-tunniste palvelimen tiedossa vai ei. Tilallisissa suojauksissa CSRF-tunniste on suoraan tallennettuna palvelimelle ja tarkistus suoritetaan

vertaamalla parametrina saatua arvoa palvelimella olevaan arvoon. Tilattomissa suojauksissa palvelin ei tiedä käyttäjän CSRF-tunnisteen arvoa, vaan oikean arvon määrittely suoritetaan jollain muulla tavalla. Tilallinen suojaus on klassisin CSRF-suojauksen toteutus ja se on muun muassa OWASP:n suosittelema tapa suojauksen toteutukseen [14]. Tilallisten suojaustapojen etu on, että ne ovat yksinkertaisia toteuttaa ja ovat tarjonneet vuosien ajan luotettavaa suojaa. Haittapuolena on, että CSRF-tunnisteen tallennus palvelimelle ei ole aina toivottavaa.

Tilallisten suojaustapojen rinnalla on tilattomat suojaustavat, joissa palvelin ei tallenna käyttäjän CSRF-tunnistetta muistiin. Tällaisille suojauksille on muutama erilainen toteutustapa, joita käsitellään seuraavaksi. Yksi tapa tilattoman suojauksen toteutukselle on muodostaa CSRF-tunniste istunnon tunnisteesta symmetrisen salausalgoritmin avulla. Symmetrisessä salauksessa viesti salataan ja puretaan samalla salausavaimella. Tälläinen suojaus on käytössä esimerkiksi Ruby on Rails -ohjelmistokehyksessä [15]. Palvelin muodostaa käyttäjän istunnon tunnisteesta salausalgoritmin avulla CSRF-tunnisteen ja välittää sen käyttäjälle. Tunnisteen tarkistus suoritetaan purkamalla salaus käyttäen palvelimelle tallennettua salausavainta. Tämän jälkeen tarkastetaan, että salauksen purkamisen lopputuloksena saatu merkkijono vastaa käyttäjän istunnon tunnistetta. Mikäli näin ei ole, palvelin ei käsittele pyyntöä. Perjaatteessa istunnon tunnistetta voitaisiin käyttää myös suoraan CSRF-tunnisteena, mutta salaamalla tunniste estetään istunnon tunnisteen vuotaminen, mikäli CSRF-tunniste joutuu vääriin käsiin. Ilman alkuperäistä salausavainta, hyökkääjä ei pysty purkamaan CSRF-tunnisteen salausta, eikä saa siten käsiinsä istunnon tunnistetta.

3.2 Kaksoislähetys

Kaksoislähetys (Double submit) on CSRF-suojauksen muoto, jossa CSRF-tunnus on käyttäjän palvelimelle lähettämässä pyynnössä sekä evästeenä että pyynnön parametrina. Suojaus toteutetaan siten, että palvelin tallentaa CSRF-tunnisteen käyttäjän selaimeen. Jos käyttäjän lähettämän pyynnön lähetys toteutetaan JavaScriptin avulla, CSRF-tunnistetta ei tarvitse välittää käyttäjälle muilla tavoin, vaan se voidaan poimia evästeestä käyttäen JavaScriptiä. Muussa tapauksessa palvelin tallentaa CSRF-tunnisteen myös piilotetuksi lomakkeen kentäksi. Palvelin vertaa evästeenä saatua arvoa pyynnön parametrina saatuun

arvoon ja edellyttää, että ne vastaavat toisiaan. Kaksoislähetys voidaan toteuttaa myös siten, että kun CSRF-tunniste luodaan käyttäjän selaimessa voidaan suojauksen tasoa parantaa luomalla eväste uudelleen jokaisen lähetyksen yhteydessä ja muuttamalla evästeen arvoksi jokin epäkelpo arvo pyynnön päätteeksi. Suurimman osan ajasta käyttäjän selain on tilassa, jossa kelvollisen pyynnön lähetys ei ole mahdollinen, koska evästeen tallennetulla CSRF-tunnisteella ei ole kelvollista arvoa. Tällä tavalla CSRF-hyökkäyksen aikaikkuna pienennetään hyvin pieneksi. Evästeen muuttaminen epäkelvoksi lisäksi helpottaa CSRF-hyökkäyksen havaitsemista palvelimen puolella, sillä epäkelvollisella evästeen arvolla lähetetty pyyntö on osoitus joko CSRF-hyökkäyksen yrityksestä tai ohjelmavirheestä.

Evästeisiin perustuvat suojaukset ovat haavoittuvia hyökkäykselle, jossa hyökkääjä pystyy manipuloimaan käyttäjän kohdepalveluun liittyviä evästeitä. Hyökkääjä voi toteuttaa evästeen asettamisen muutamalla vaihtoehtoisella tavalla. Kontrolloimalla rinnakkaista aliverkkotunnusta hyökkääjä pystyy asettamaan evästeitä rinnakkaisille aliverkkotunnuksille sekä ylemmän tason verkkotunnuksille. [16, s. 3.] Esimerkiksi kontrolloimalla osoitetta xyz.tunnus.com hyökkääjä voi asettaa evästeitä kaikille rinnakkaisille aliverkontunnuksille abc.tunnus.com ja ylemmän tason osoitteelle, kuten tunnus.com. Poikkeuksena ovat ne osoitteet, jotka löytyvät Mozilla Foundationin hallinnoimalta julkisten osoitepäätteiden listalta [17]. Tällä listalla on määriteltynä osoitepäätteet, jotka eivät ole osana yksittäistä osoiterekisteröintiä, eikä niille siten tule voida asettaa evästeitä. Listalla on kaikki ensimmäisen tason osoitteet, kuten .com, .org ja .fi, mutta myös suuri määrä alemman tason osoitteita. Kaikki yleiset selaimet ja monet HTTP-yhteyksien muodostamiseen käytettävät ohjelmakirjastot estävät evästeiden asettamisen osoitteille, jotka on listalla kielletty.

Harvinaisempi tilanne, jossa hyökkääjä voi asettaa käyttäjän selaimeen evästeitä on aktiivinen verkkohyökkäys. Tässä tilanteessa hyökkääjällä on pääsy käyttäjän verkkoon siten, että hyökkääjä pystyy kaappaamaan ja väärentämään käyttäjälle menevää liikennettä. Vaikka liikenne kohdesivuston ja käyttäjän välillä olisi salattu eikä hyökkääjä pystyisi suoraan lukemaan sitä, evästeiden asettaminen paketteja väärentämällä on silti mahdollista. Palvelin voi asettaa evästeelle secure-attribuutin, joka kertoo selaimelle, että evästeen saa lähettää vain suojatun yhteyden yli. Tämä ei kuitenkaan rajoita evästeen vastaanottamista tai ylikirjoittamista. Eväste, jonka palvelin on alunperin asettanut suojatun yhteyden yli, voidaan asettaa uudelleen ilman suojattua yhteyttä. Tämän vuoksi hyökkääjä voi ylikirjoittaa evästeen paketteja väärentämällä, vaikka käyttäjän selaimen ja palvelimen välinen

tietoliikenne olisi salattu.

3.3 Referer- ja Origin-otsakkeet

Pyyntöjen lähde voidaan selvittää toisen kahdessa yleensä pyynnöissä esiintyvän otsakkeen avulla. Luvussa 2.1 esiteltiin HTTP-pyynnön rakenne ja kuvattiin, kuinka pyyntö koostuu otsakkeista sekä pyynnön varsinaisesta runko-osuudesta. Suurin osa näistä otsakkeista on selainten itse asettamia. CSRF-suojauksen kannalta hyödylliset otsakkeet ovat nimeltään Referer ja Origin. Kumpikin otsake kertoo palvelimelle, miltä sivulta pyyntö on lähtöisin. Käytännössä otsake kertoo, millä sivulla linkkiä on klikattu, jonka seurauksena palvelimelle lähetettiin pyyntö. Referer-otsake välittää palvelimelle koko sivun osoitteen, jolta pyyntö on lähtöisin. Tämä osoite sisältää esimerkiksi osoitekenttään sisällytetyt parametrit ja saattaa sen vuoksi paljastaa luottamuksellista tietoa. Tästä syystä osa käyttäjistä saattaa estää selainta lähettämästä Referer-otsaketta pyyntöjen yhteydessä.

Origin-otsake on nimenomaan CSRF-hyökkäyksiä silmällä pitäen kehitetty otsake, joka lähettää palvelimelle vain pyynnön lähteen tunnistamisen kannalta olennaisen tiedon. Origin-otsakke huomioi käyttäjien yksityisyyden suojan tarjoten samalla palvelimelle suojauksen kannalta olennaisen tiedon. Selaimet voivat sisällyttää Origin-otsakkeen kaikkiin HTTP-pyyntöihin [18, s.13]. Origin-otsakkeen välitys ei ole samalla tavalla vakiintunut kuin Referer-otsakkeen välitys, vaan siinä on eroa selainten välillä. Tästä syystä Origin-otsake ei ole vielä käyttökelpoinen CSRF-suojauksen toteutukseen.

Referer- tai Origin-otsakkeen perusteella palvelin voi varmistua pyynnön lähteestä. Selaimet sallivat Referer-otsakkeen väärentämisen, mutta se ei usein ole suoraan mahdollista CSRF-hyökkäyksen yhteydessä [19]. Barth, Jackson ja Mitchell havaitsivat CSRF-suojauksia käsittelevässä katsauksessaan, että suurin osa selaimista sisällyttää Referer-otsakkeen pyyntöihin. Tämä tapahtuu suurimmalla todennäköisyydellä silloin, kun pyyntö tapahtuu suojatun yhteyden yli. [12, s. 6-7.] Referer-otsakkeeseen perustuvaa CSRF-suojausta ei suositella käytettäväksi yksinään suojauksena, mutta sitä voidaan hyödyntää ylimääräisenä suojauskerroksena. Referer-otsakkeen tarkistukseen perustuvaa suojausta voidaan hyödyntää myös CSRF-hyökkäyksen torjumiseen sisäänkirjautumisen yhteydessä. Tämä on hyödyllistä, sillä ennen sisäänkirjausta käyttäjillä ei ole yleensä istun-

toa eivätkä istunnon yhteydessä luotavat CSRF-suojaukset ole siten aktiivisena. Referer-tarkistuksen toteutus on teknisesti yksinkertaista. Palvelimelle saapuvasta pyynnöstä poimitaan Referer-otsakkeen sisältö, jonka muoto tarkistetaan. Pyynnön käsittelyä ei jatketa pidemmälle, mikäli pyynnöstä puuttuu Referer-otsake tai sen muoto on virheellinen. Muodon määrittely voidaan tehdä esimerkiksi säännöllisen lausekkeen avulla. Määrittelyssä on syytä olla tarkkana, sillä huonosti tehtyä tarkistusta on mahdollista huijata.

3.4 Suojaus itse lisättyä HTTP-otsaketta käyttäen

Omien otsakkeiden sisällytys pyyntöihin on mahdollista, mutta se edellyttää JavaScriptin käyttöä. Tarkalleen ottaen lähetys tehdään käyttäen JavaScriptin XmlHttpRequest-olioa. Pyyntöjen lähetyksessä noudatetaan saman alkuperän käytäntöä (same origin policy), eikä yhdestä lähteestä ladattu skripti pysty yleensä lähettämään pyyntöjä oman alkuperänsä ulkopuolelle. Kahdella sivulla on sama alkuperä mikäli protokolla, palvelin ja portti ovat samat. Porttia tarkastellaan vain jos se on määritelty. Taulukko 3.1 selventää saman alkuperän vertailua, kun vertailtava osoite on http://abc.esimerkki.com/kansio/sivu.html

Taulukko 3.1: Saman alkuperän vertailun tulokset ja tulosten perusteet. [22.]

URL	Tulos	Syy
http://abc.esimerkki.com/kansio/sivu.html	Sama alkuperä	
http://abc.esimerkki.com/kansio/kansio2/toinen.html	Sama alkuperä	
https://abc.esimerkki.com/salattu.html	Eri alkuperä	Eri protokolla
http://abc.esimerkki.com:81/kansio/abc.html	Eri alkuperä	Eri portti
http://xyz.esimerkki.com/kansio/sivu.html	Eri alkuperä	Eri palvelin

Omia otsakkeita voidaan tietyin rajoituksin käyttää CSRF-suojaukseen, koska niiden lähetys kahden eri alkuperää olevan sivuston välillä ei ole mahdollista. Otsakkeiden käyttö CSRF-suojauksessa edellyttää, että kaikki sivustolla tehtävät muutoksia aiheuttavat pyynnöt lähetetään JavaScriptin avulla. Suojauksen ideana on sisällyttää pyyntöihin itse luotu otsake, jonka löytyminen pyynnöstä kertoo palvelimelle, että pyyntö on luotettava. Otsakkeen arvolla ei ole tämän suojauksen kannalta merkitystä, pelkkä otsakekentän löytyminen pyynnöstä on riittävä. [12, s. 7.] JavaScript-kirjastoista esimerkiksi Prototype.js käyttää tätä lähestymistapaa ja sisällyttää automaattisesti sen avulla tehtyihin pyyntöihin X-Requested-With-nimisen otsakkeen [20]. Suurin este tämän tyylisen suojauksen toteu-

tukselle etenkin vanhojen sivustojen yhteydessä on tarve käyttää JavaScriptiä kaikkien pyyntöjen toteutukseen.

Referer-kentän tavoin oman otsakekentän käyttö on hyvä lisä CSRF-suojaukseen, mutta se saatetaan nähdä riittämättömänä ainoaksi suojaukseksi [21, s. 3]. Syynä tähän on se, että esimerkiksi Internet Explorer käsittelee JavaScriptien yhteydessä saman alkuperän käytäntöä hieman normaalista poikkeavasti. Näitä poikkeuksia on kaksi:

1. Luotetut osoitealueet (Trust Zones): Mikäli sivusto, jolta pyyntö lähetetään ja pyynnön kohde on molemmat määritelty selaimessa korkean luottotason alueiksi, saman alkuperän käytäntöä ei noudateta.
2. Portin numeroa ei huomioida saman alkuperän käytännön yhteydessä. [22.]

Nämä poikkeukset ovat hyvin rajoittuneita, mutta ne silti mahdollistavat CSRF-hyökkäyksen tietyissä tapauksissa [21]. CSRF-hyökkäyksen toteutus AJAX-tekniikoita käyttäen on poikkeuksellisen vaarallinen, koska se mahdollistaa palvelimelta tulevien vastausviestien lukemisen ja niihin reagoimisen. Hyökkääjä voi esimerkiksi ensin pyytää sivua, jolta CSRF-tunniste löytyy ja sen jälkeen toteuttaa toisen pyynnön käyttäen ensimmäisestä pyynnön avulla saatua CSRF-tunnistetta. [21, s. 6.]

AJAX-tekniikoita käyttävä CSRF-hyökkäys on erittäin harvinainen, koska hyökkäyksen ennakkovaatimukset harvoin toteutuvat. Tästä huolimatta se on hyvä esimerkki eri CSRF-suojausten rinnakkainkäytön hyödyistä. Normaalista CSRF-hyökkäyksestä poiketen AJAX-pohjaisella CSRF-hyökkäyksellä on mahdollista kiertää suurin osa CSRF-avaimeen perustuvista suojauksista, mutta sekään ei pysty kiertämään oikein toteutettua Referer-otsakkeen tarkistusta. Varsinaiseksi suojaukseksi näitä hyökkäyksiä vastaan esitetään kaikkien sivunlatausten suojausta CSRF-avaimen tyyppisellä suojauksella pelkkien tallennuspyyntöjen suojauksen sijaan. [21, s. 15.] Tämä on kuitenkin raskas toteutustapa, joka estää navigoinnin suoraan halutulle sivulle, esimerkiksi kirjainmerkin avulla.

3.5 Challenge-Response-tyyppiset suojaukset

Challenge-Response-suojaukset perustuvat interaktioon käyttäjän ja palvelimen kanssa, eivätkä ole käyttäjälle näkymättömiä. Suojauksessa käyttäjälle esitetään kysymys, johon

vaaditaan oikea vastaus, jotta pyynnön käsittelyä jatketaan. Suojauksien suurin etu on, etteivät ne oikein toteutettuna ole kierrettävissä vaikka sivustolla olisi XSS-haavoittuvuus. Tämä johtuu siitä, ettei Challenge-Response-kysymykseen vaadittava vastaus ole hyökkääjän tiedossa tai skriptin avulla muodostettavissa. [8, s. 10.] Esimerkkejä Challenge-Response-autentikoinneista ovat esimerkiksi verkkopankkien tunnusluvun kysyminen, CAPTCHA-autentikointi tai salasanan uudelleen kysyminen. Challenge-Response -autentikointeja toteutetaan harvoin nimenomaan CSRF-suojauksen takia, vaan ne on otettu sivulla käyttöön palvelemaan jotain muuta käyttötarkoitusta. Salasanan uudelleenkysyminen tai erillisen tunnusluvun kysyminen ovat keinoja, joilla tietyn toiminnon suojauksen tasoa korotetaan. CAPTCHA-autentikointia puolestaan käytetään, kun halutaan varmistua, että käyttäjä on oikeasti ihminen. Tavoitteena on estää palvelun automatisoitua käyttöä, esimerkiksi skriptien avulla. Challenge-Response-autentikointi sopii käytettäväksi rinnakkain näkymättömien CSRF-suojausten kanssa.

3.6 Käyttäjän omat toimet

CSRF-hyökkäys on siitä poikkeuksellinen, että käyttäjä pystyy rajoittamaan sen toimivuutta omilla toimillaan. Kirjautumalla ulos palveluista käytön loputtua ja välttämällä muuta samanaikaista käyttöä selaimella, käyttäjä pienentää sisäänkirjautumista hyödyntävien CSRF-hyökkäyksien toimintamahdollisuutta. Erityistä huomiota kannattaa kiinnittää Basic access authentication -tunnisteita käyttäviin sisäänkirjautumisiin. Basic access authentication on tunnistautumistapa, jossa selain lähettää pyynnön yhteydessä käyttäjätunnuksen ja salasanan. Jotta käyttäjätunnusta ja salasanaa ei tarvitse kysyä udelleen jokaisen pyynnön yhteydessä, selaimet saattavat ehdottaa näiden tunnusten tallennusta myöhempää käyttöä varten. Palveluiden käyttäjätunnuksia ja salasanoja ei vaihdeta usein, jolloin tunnuksia hyödyntävään palveluun kohdistuva CSRF-hyökkäys toimisi, kunnes selainta ohjeistetaan unohtamaan tunnukset tai käyttäjätunnus tai salasana vaihdetaan. [8, s. 11.]

CSRF-hyökkäys on sokea hyökkäys, jossa hyökkääjä arvaa tai ennalta tietää kohteen ja tavan, jolla suorittaa hyökkäys. Reitittimien ja muiden käyttäjän omassa verkossa olevien laitteiden sisäänkirjautumistunnukset tulisi aina vaihtaa pois tehdasasetuksista. CSRF-hyökkäys on hyvä tapa hyöktä esimerkiksi käyttäjän reititintä vastaan ja mahdollistaa

siten laajempi hyökkäys käyttäjän sisäverkkoon. [8, s. 11.] Reitittimien ip-osoitteet noudattavat yleensä ennalta arvattavaa kaavaa, ja tehdasasetukset ovat julkista tietoa.

4 Tapausesimerkki: Haavoittuvuus Googlen Gmail-palvelussa

Ajan saatossa CSRF-haavoittuvuuksia on löytynyt lukuisista eri palveluista, myös tunnetuista palveluista joiden tietoturvaa pidetään yleensä korkeatasoisena. Tässä luvussa käydään läpi vuonna 2007 raportoitu hyökkäys, jossa hyökkääjä hyödynsi Googlen Gmail-palvelun haavoittuvuutta ottaessaan haltuun David Aireyn davidairey.com-verkkotunnuksen.

Vuoden 2007 aikana Gmail-palvelussa esiintyi CSRF-haavoittuvuus, jonka avulla hyökkääjä pystyi saamaan käsiinsä uhrina olevan käyttäjän sähköpostit. Hyökkäys hyödynsi sähköpostisuodattimien muokkaukseen käytetyn rajapinnan CSRF-haavoittuvuutta. CSRF-hyökkäyksen avulla hyökkääjä pystyi lisäämään käyttäjän sähköpostitilille omia sähköpostisuodattimia. Suodattimen avulla hyökkääjä pystyi muun muassa välittämään hyökkääjän valitsemia avainsanoja sisältävät sähköpostit suoraan hyökkääjän hallinnoimaan sähköpostiosoitteeseen ja poistamaan ne automaattisesti uhrin sähköpostilaatikosta. Uhrilla ei ollut mahdollisuutta havaita tätä, ellei uhri tarkastanut oman sähköpostitilinsä suodatinasetuksia.

Hyökkäyksen tietosisältö oli tavallinen multipart/form-data -enkoodauksella lähetetty lomake.

```
<form method="POST" action="https://mail.google.com/mail/h/ewt1jmuj4ddv/?
    v=prf" enctype="multipart/form-data">
  <input type="hidden" name="cf2_emc" value="true"/>
  <input type="hidden" name="cf2_email" value="evilinbox@mailinator.com"/>
  <input type="hidden" name="cf1_from" value=""/>
  <input type="hidden" name="cf1_to" value=""/>
  <input type="hidden" name="cf1_subj" value=""/>
  <input type="hidden" name="cf1_has" value=""/>
  <input type="hidden" name="cf1_hasnot" value=""/>
  <input type="hidden" name="cf1_attach" value="true"/>
  <input type="hidden" name="tfi" value=""/>
  <input type="hidden" name="s" value="z"/>
  <input type="hidden" name="irf" value="on"/>
  <input type="hidden" name="nvp_bu_cftb" value="Create Filter"/>
```

```
</form>
<script>
  document.forms[0].submit();
</script>
```

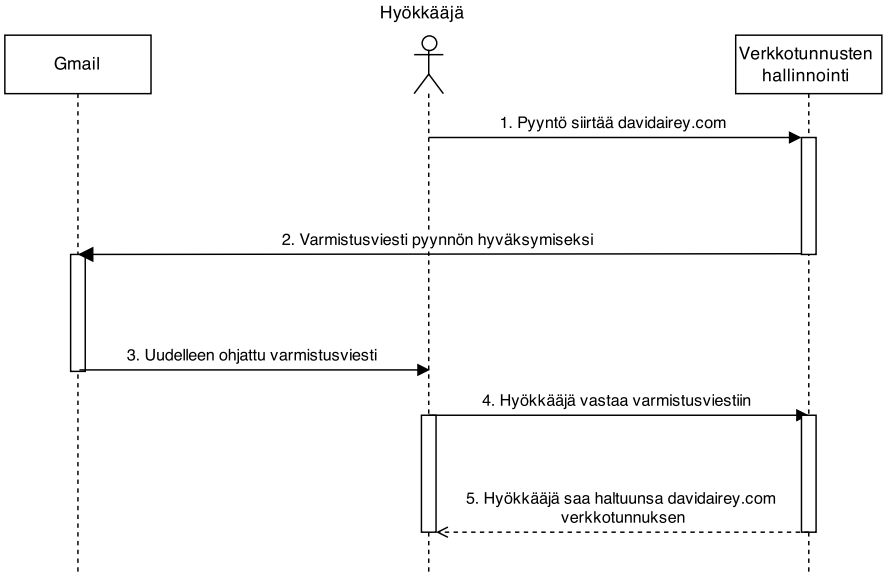
Koodiesimerkki 4.1: Gmail-palveluun kohdistuvan CSRF-hyökkäyksen käynnistykseen käytetty automaattisesti lähettävä lomake

Gmail-palvelussa ei ollut CSRF-suojausta, mikä teki hyökkäyksestä mahdollisen. Tietosisällössä on vain sähköpostifiltterin luomiseen tarvittavat lomakekentän tiedot. Hyökkäyksessä uhrina olevan käyttäjän selain lähettää automaattisesti lomakkeessa määritellyt tiedot lomakkeen action-attribuutissa määritettyyn osoitteeseen, jossa sijaitsee Gmail-palvelun sähköpostisuodattimien lisäämiseen käytetty rajapinta. Pyyntö onnistuu, mikäli uhri on kirjautuneena Gmail-palveluun, sillä pyynnön yhteydessä lähetetään automaattisesti myös istunnon tunnisteen sisältävä eväste.

Hyökkääjä aloitti hyökkäyksen huijaamalla Aireyn vierailemaan hyökkäyksen sisältävällä sivustolla Aireyn ollessa sisäänkirjautuneena Gmail-palveluun. Aireyn selain suoritti CSRF-hyökkäyksen sisältämän elementin, jolloin Aireyn sähköpostitilille lisättiin seuraavat sähköpostisuodattimet:

- “Matches: transfer-approval.com “Do this: Forward to ba_marame_pooli@yahoo.com, Skip Inbox, Delete it”
- “Matches: from:(transfer-approval.com) “Do this: Forward to ba_marame_pooli@yahoo.com, Skip Inbox, Delete it”

Suodattimet lähettivät viestit, joista löytyy teksti “transfer-approval.com” suoraan hyökkääjän hallinnoimaan sähköpostiosoitteeseen laittamatta niitä uhrin vastaanotettujen viestien joukkoon.



Kuva 4: Sekvenssikaavio hyökkäyksen kulusta sen jälkeen, kun hyökkääjä on saanut lisättyä sähköpostisuodattimet Aireyn sähköpostitilille.

Kuvassa 4 on kuvattu hyökkäyksen vaiheet sen jälkeen, kun Aireyn sähköpostitilille oli lisätty hyökkääjän haluamat sähköpostisuodattimet. Sähköpostisuodattimien lisäämisen jälkeen, hyökkääjä on väärentänyt verkkotunnuksen rekisteröintipalveluun sähköpostiviestin, jossa pyydetään davidairey.com-verkkotunnuksen siirtämistä toiselle verkkotunnusten hallinnointipalvelulle. Verkkotunnusta alunperin hallinnoinut taho on lähettänyt varmistusviestin Aireyn sähköpostiosoitteeseen, joka on ollut merkittynä yhteysosoitteeksi verkkotunnuksen hallinnointia varten. Sähköpostisuodattimien avulla hyökkääjä pystyi lukemaan varmistusviestit ja vastaamaan niihin. Lopuksi hyökkääjä sai haltuunsa davidairey.com verkkotunnuksen. [23.]

5 Suojattavan järjestelmän arkkitehtuuri

Järjestelmän arkkitehtuuri määrittää hyvin paljon CSRF-suojauksen toteutusta. Perinteiseen usean sivun verkkopalveluun valittava suojaustapa ei ole välttämättä paras modernimpaa yhden sivun verkkopalvelua suojattaessa.

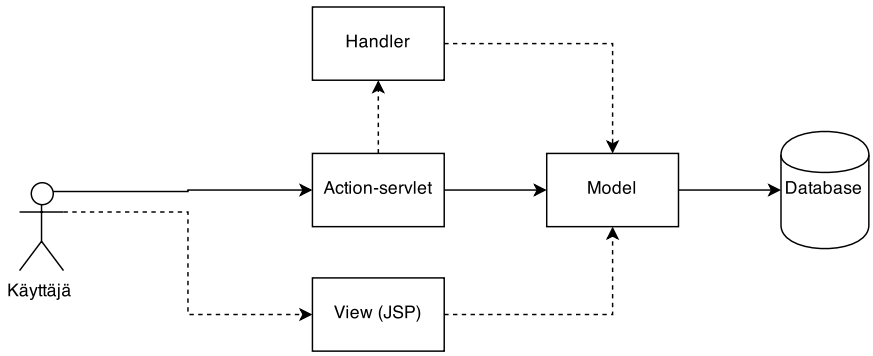
Tässä työssä suojaus toteutetaan pitkään kehityksen alla olleeseen järjestelmään, jossa ei ole käytössä järjestelmänlaajuista ohjelmistokehystä. Järjestelmä on toteutettu käyttäen Sun Microsystemsin kehittämää JavaServer Pages (JSP) -teknologiaa, joka mahdollistaa dynaamisten verkkosivujen luomisen Javan avulla. Ohjelmistokehyksen osittainen puuttuminen tarkoittaa käytännössä, ettei tarvittavia muutoksia ole mahdollista toteuttaa keskitetysti koko järjestelmänlaajuisesti ainakaan ennen, kuin ohjelmistokehys on otettu käyttöön koko järjestelmässä.

JSP-sivut sisältävät usein sekaisin HTML-merkintää, pätkiä Java-koodia ja JSP-merkintöjä. Javan avulla on yleensä toteutettu loogisia rakenteita, kuten silmukoita. Kun palvelimelta pyydetään JSP-sivua, palvelin tarkastaa ensin, onko kyseisestä sivusta olemassa ajantasainen käännetty versio. Mikäli käännetty versio puuttuu tai sivua on muutettu kääntämisen jälkeen, palvelin tulkkaa JSP-sivun ensin Java Servlet -luokaksi, jonka jälkeen luokka käännetään ja ajetaan. Ajon tulos välitetään vastauksena pyyntöön. JSP-sivut voivat sisältää JSP-osasia (JSP-fragment). JSP-osaset eroavat JSP-sivuista siten, että ne eivät ole yksinään tulkattavia.

JSP-merkinnät ovat XML-merkintätapaa noudattavia merkintöjä, jotka käsitellään JSP-sivun tulkkauksen yhteydessä. JSP:ssä on vakiona muutamia JSP-merkintöjä, joilla voi esimerkiksi sisällyttää sivuille toisia JSP-sivuja. Näiden lisäksi on mahdollista määrittää omia JSP-merkintöjä. JSP-sivun tulkkauksen yhteydessä tulkki kutsuu JSP-merkinnälle määriteltyä käsittelijää. Käytännössä JSP-merkinnällä siis kutsutaan taustalla olevaa Java-luokkaa ja välitetään sille merkinnässä määriteltyjä parametreja. Määriteltyjä merkintöjä käyttäen koodin kirjoitus nopeutuu, ja samalla JSP-sivuille kirjotettavan Java-koodin määrä vähenee. Seurauksena järjestelmän sivujen koodista tulee yhtenäisempi ja käyttöliittymäsivujen koodista helpommin luettavaa. JSP-merkintöjen ero tavalliseen sivulle

upotettuun Java-koodin metodikutsuun nähden on mahdollisuus määrittää käsittelyjä, jotka suoritetaan vain alku- tai loppumerkinnän kohdalla. Tämän työn kohteena olevassa järjestelmässä omia JSP-merkintöjä on määritelty laajasti. Kaikki järjestelmän käyttöliittymät on toteutettu JSP-sivuina ja niitä pyydetään suoraan samaan tapaan kuin normaaleja HTML-sivuja.

Tallennettaessa tietoa järjestelmään suurin osa pyynnöistä käsitellään keskitetysti Action-nimisessä Java-servletissä.



Kuva 5: Tiedon tallennusreitit järjestelmässä

Kuvassa 5 on kuvattu paikat, joissa tietoa tallennetaan järjestelmään. Tiedon käsittelyä varten on määritelty malli-luokat (model), joiden kautta kaikki keskustelu tietokannan kanssa tapahtuu. Action-servlet käsittelee itse yleisimmät tiedon tallennukseen, muokkaamiseen ja poistamiseen liittyvät pyynnöt. Action-servletin lisäksi on mahdollista tehdä omia käsittelijäluokkia (Handler), joissa voi määritellä erikoisempia vain tiettyihin käyttötapauksiin liittyviä toimintoja. Action-servletille kerrotaan parametrilla, mikäli pyyntö on tarkoitus välittää myös käsittelijäluokalle. Pyyntö käsitellään ensin määritellyssä käsittelijäluokassa, jonka jälkeen se välitetään normaalisti Action-servletille. Seuraavaksi Action-servlet käsittelee pyynnön kokonaisuudessaan samaan tapaan, kuin jos käsittelijäluokkaa ei olisi määritelty. Action-servletin lisäksi järjestelmässä on yksittäisiä JSP-sivuja, jotka tekevät kutsuttaessa järjestelmään muutoksia. Niitä käytetään hieman samaan tapaan, kuin kutsuja Action-servletiin. Suurin osa näistä sivuista on vanhoja toteutuksia, jotka on parempi toteuttaa Action-servletin käsittelijäluokkien avulla. Mikäli kaikki tallennuksia tekevät pyynnöt kulkisivat yhtä reittiä pitkin, CSRF-suojauksen tarvitsemat pyyntöjen käsittelyjen muutokset olisi selkeämpi toteuttaa. Nykyisessä tilanteessa muutokset tulee toteuttaa useampaan kuin yhteen paikkaan.

6 Suojaustavan valinta ja toteutus

6.1 Suojaustavan valinta

Varsinainen työ aloitettiin valitsemalla käytettävä suojaustapa. Ennen työn varsinaista aloitusta oli selvää, että helpoin tapa toteuttaa CSRF-suojaus on tehdä se käyttäen perinteistä istuntoriippumatonta Synchronizer Token Pattern -mallia. Suojaustavan valintaa ohjasi työn aloitushetkellä vahva tarve istuntoriippumattomalle suojaukselle. Järjestelmän käyttäjien keskuudessa on normaalia, että järjestelmää käytetään samanaikaisesti usealla selaimen välilehdellä. Istuntoriippuvaisella suojauksella sisäänkirjautuminen toisella selaimen välilehdellä olisi automaattisesti aiheuttanut aikaisemmin ladatulla sivulla olevan CSRF-avaimen vanhenemisen. Tämän jälkeen aikasemmin ladatuilla sivuilla mahdollisesti kesken olevaa työtä ei olisi mahdollista jatkaa. Suurin osa järjestelmän pyynnöistä lähetetään tavallisilla lomakkeilla, joten CSRF-tunnisteen päivitys ennen pyynnön lähetystä ei käytännössä onnistu ilman suuritöistä muutosta.

Jokainen käyttäjä keskustelee kerrallaan vain yhden palvelimen kanssa, joten tilan tallentamiseen ja jakamiseen liittyviä ongelmia palvelinten välillä ei ilmene. Järjestelmissä ei myöskään ole sellaisia tallennuskapasiteettia rajoittavia tekijöitä, joiden takia tallennettua tilamuuttujaa ei kannattaisi käyttää. Myös nämä seikat tukivat päätöstä toteuttaa suojaus istuntoriippumattomana tilamuuttujan avulla.

6.2 Suojauksen toteutus

Järjestelmä on jatkuvassa käytössä kymmenissä yrityksissä. Tämän takia suojaus oli välttämätöntä toteuttaa siten, ettei järjestelmän normaali toiminta keskeydy missään vaiheessa. Ensin järjestelmän käyttöliittymiä muokattiin, jotta CSRF-tunniste saatiin välitettyä suurimpaan osaan lomakkeista ja linkeistä, joilla tallennuksia tehdään. Tämän jälkeen pyyntöjen käsittelyn yhteyteen lisättiin CSRF-tunnisteen tarkistus. Tässä vaiheessa palvelin ei vielä hylännyt pyyntöjä puuttuvien tai virheellisten CSRF-tunnisteiden perusteella, vaan palvelin teki tapauksista merkinnän lokiin ja lähetti sähköpostin ylläpidolle. Kun

lokien perusteella näyttää siltä, että kaikki käyttöliittymät välittävät CSRF-tunnisteen pyyntöjen yhteydessä oikein, järjestelmä laitetaan tilaan, jossa se hylkää pyynnöt virheellisten CSRF-tunnisteiden perusteella.

6.2.1 Referer-tarkistuksen toteutus

Toteutuksen ensimmäisessä vaiheessa Action-servletin yhteyteen toteutettiin Referer-otsakkeen tarkistus ja virheellisen otsakkeen lokiin kirjaus. Alkuperäisen suunnitelman mukaan Referer-otsaketta oli tarkoitus käyttää vain tiedon keräämiseen. Referer-otsaketta pidettiin ennakkotietojen perusteella turhan epävarmana tapana pyyntöjen hylkäämiseen [5] ja oli pelkona, että Referer-otsakkeen käyttö pyyntöjen hylkäämiseen aiheuttaisi täysin kelvollisten pyyntöjen hylkäyksiä. Kuitenkin lähes kolmen kuukauden seurantajaksolla ainoa lokiin kirjattu pyyntö, jossa Referer-kenttä oli virheellinen, oli Googlen hakurobotin aikaansaama. Tämän perusteella Referer-otsaketta tullaan myöhemmin käyttämään pyyntöjen hylkäämiseen yhdessä CSRF-avaimen tarkistuksen kanssa.

Referer-kentän tarkistus toteutettiin käyttäen Javan Pattern- ja Matcher-luokkia, joiden avulla toteutettiin säännöllisen lausekkeen vertailu.

```
Pattern p = Pattern.compile("^https://\w+\.likeit\.fi/.*$");
p.matcher(input).matches(); // True, jos input vastaa säännöllistä
    lauseketta
```

Koodiesimerkki 6.1: Säännöllisen lausekkeen vertailu Pattern- ja Matcher-luokilla.

Pattern-luokasta luodaan ilmentymä kutsumalla Pattern-luokan compile-metodia ja välittämällä parametriksi haluttu säännöllinen lauseke. Tämän jälkeen Pattern-luokan matcher-metodia kutsutaan käyttäen parametrina vertailtavaa syötettä. Kutsu palauttaa Matcher-luokan ilmentymän, jonka matches-metodia kutsumalla saadaan boolean-arvo, joka kertoo, vastasiko vertailtava syöte säännöllistä lauseketta.

Koodin puolesta vertailu on hyvin yksinkertainen, mutta säännöllisen lausekkeen muotoiluun tulee kiinnittää erityistä huomiota, jottei hyökkääjällä ole mahdollisuutta huijata tarkistusta. Tässä tapauksessa sallittavien osoitteiden muoto noudattaa hyvin toistuvaa kaavaa, jolloin säännöllinen lauseke on muodoltaan melko yksinkertainen.

1. ^https:// rajoittaa käytettäväksi protokollaksi https-protokollan.
2. \w+ määrittää, että osoitteen seuraava osan tulee koostua vähintään yhdestä alfanumeerisesta merkistä. Myös väliviiva ja alaviiva ovat sallittuja
3. (.likeit.fi/){1} edellyttää, että suluissa oleva merkkijono esiintyy tasan yhden kerran.

Kriittisimpiä kohtia lausekkeen kannalta ovat lausekkeen alun ^https:// ja lopun .fi/. Nämä määrittävät tiukasti osoitteen palvelimen ja protokollan rajat. Tämä estää lausekkeen huijaamisen muodostamalla hyökkääjän hallinnoimalle verkko-osoitteelle aliosoitteita, joiden muoto vastaa säännöllisen lausekkeen edellyttämää muotoa. Pyynnöt, joista Referer-kenttä puuttuu kokonaan tai joiden Referer-kenttä ei vastaa säännöllisen lausekkeen määrittelemää muotoa, kirjataan lokiin ja hylätään.

6.2.2 CSRF-tunnisteen tarkistus

CSRF-tunnisteen vertailu on käytännössä yksinkertainen merkkijonovertailu. Parametrina saatu CSRF-tunniste ei saa olla tyhjä ja sen täytyy vastata käyttäjän tietoihin tallennettua avainta. Käyttäjän tietoihin tallennettua avainta varten luotiin getter-metodi, jolla avain saadaan palautettua. CSRF-tunnisteen tarkistus suoritetaan välittömästi kyselyn vastaanottamisen yhteydessä ennen muita toimintoja. Pyynnön käsittely lopetetaan, eikä mitään toimintoja suoriteta, mikäli CSRF-tunnisteen arvo on virheellinen.

6.2.3 Poikkeustapaukset

Järjestelmässä on käytössä yksittäisiä JSP-sivuja, joissa käsitellään muutoksia tekeviä pyyntöjä. Nämä sivut ovat sellaisia, joista päästään tulevaisuudessa eroon laajentamalla Action-servletiä tukemaan kyseisiä erikoistapauksia. Väliaikaisena toimenpiteenä näiden sivujen alkuun lisättiin kutsut Referer-otsakkeen ja CSRF-tunnisteen validoiviin metodeihin. Sivujen toimintoja ei suoriteta, mikäli pyyntö ei läpäise tarkistuksia.

6.3 Käyttöliittymäsivujen muokkaus

Suojausta varten käyttöliittymä sivuja täytyy muokata siten, että CSRF-tunniste välittyy käyttäjän selaimeen sivun latauksen yhteydessä ja se saadaan välitettyä pyyntöjen parametrina takaisin palvelimelle. Käyttöliittymä sivuilla käytetään kolmea eri tapaa lähettää pyyntöjä.

1. tavallisen HTML-lomakkeen lähetys
2. AJAX-pyyntö
3. suora pyyntö Action-servletiin.

Käyttöliittymien muokkausten yhteydessä pyritään samalla yhtenäistämään koodia, jotta muutoksien tekeminen jatkossa olisi helpompaa. Kaikki lomakkeet ja suorat linkit toteutetaan käyttäen JSP-merkintöjä. JSP-merkintöjen käyttö mahdollistaa tulevaisuudessa esimerkiksi keskitettyjen ulkoasumuutosten tekemisen. Vastaavasti kaikki AJAX-pyynnöt toteutetaan yhtä JavaScript-funktiota käyttäen.

6.3.1 HTML-lomakkeiden muokkaus

Suurin osa järjestelmän lomakkeista on toteutettu käyttäen <am:form> JSP -merkintää, joka tulostaa käyttöliittymälle normaalin HTML-lomakkeen ja sen sisälle annettujen parametrien mukaan piilotettuja lomakekenttiä. CSRF-tunnisteen sisällyttäminen suurimmalle osalle sivuista onnistuu yksinkertaisesti muokkaamalla am:form-merkinnän kautta kutsuttavaa Java-luokkaa siten, että CSRF-tunniste on kaikissa lomakkeissa automaattisesti piilotettuna lomakekenttänä. Tämän jälkeen seuraava askel on toteuttaa järjestelmässä olevat tavalliset lomakkeet am:form-merkinnän avulla. Teoriassa tavalliset HTML-form-merkinnät voidaan vain suoraan vaihtaa am:from-merkinnöiksi ja poistaa lisäksi lomakekentät, jotka am:form-merkintä tulostaa automaattisesti. Käytännössä tämä ei kuitenkaan aina onnistu, sillä JSP-merkintöjen alku- ja loppumerkin täytyy sijaita samassa tiedostossa. HTML-merkinnöillä ei ole vastaavaa rajoitetta ja tästä syystä lomakkeiden aloitus- ja lopetusmerkinnät sijaitsevat osassa tapauksista eri tiedostoissa. Tämä on suurin syy sille, ettei lomakkeiden muokkauksen automatisointia lähdetty yrittämään, vaan kaikki noin 100 muokattavaa sivua muokattiin käsin. Oletettavissa oli, ettei käyttöliittymän muokkauksen automatisoinnilla saavutettaisi mitään hyötyä käsin muokkaukseen verrattuna.

Tämä voidaan perustella sillä, että muokkausta ei tarvitse toteuttaa järjestelmässä useita kertoja. Automatisointia ei olisi voinut toteuttaa luotettavasti siinä ajassa, missä sen pystyisi toteuttamaan käsin. Käsin muokkauksen yhteydessä käyttöliittymäsivuja käytiin läpi myös muilta osin, jolloin käyttöliittymien koodillista laatua saatiin parannettua.

6.3.2 AJAX-pyyntöjen muokkaus

Lähes kaikki AJAX-pyynnöt järjestelmässä on toteutettu käyttäen apuna itse toteutettuja JavaScript-funktioita. Tämä mahdollistaa CSRF-tunnisteen lisääminen pyyntöihin helposti. CSRF-tunniste upotettiin kaikkiin järjestelmän sivuihin JavaScript-muuttujaksi, jota voidaan käyttää skripteissä tarpeen mukaan. CSRF-tunnistetta ei saa sisällyttää puhtaasti JavaScriptiä sisältäviin tiedostoihin, koska ne on mahdollista sisällyttää skripteiksi kolmannen osapuolen sivuille. Tämä on mahdollista vain, mikäli käyttäjä on kirjautuneena sisälle. Onnistuessaan se mahdollistaisi CSRF-tunnisteen poimimisen skripteistä ja sisällyttämisen CSRF-hyökkäyksen sisältävään pyyntöön.

Action-servletille lähetystä varten luotiin uusi JavaScript-funktio, joka toteutettiin jQuery-JavaScript-kirjastoa apuna käyttäen. Järjestelmässä olevat vanhemmat, Prototype-kirjastolla toteutetut funktiot muokattiin kutsumaan uutta funktiota, jolloin vältyttiin suurilta käyttöliittymämuutoksilta. Actionille lähettävän JavaScript-funktion uudelleentoteutus jQuery:llä on osa laajempaa siirtymää pois Prototype:stä, sillä kokonaisuudessaan jQuery tarjoaa laajemman valikoiman ominaisuuksia.

6.3.3 Suorien pyyntöjen muokkaus

Suoriin Action-servletiin osoittaviin pyyntöihin liittyy ongelma, joka on GET-metodi. Toistaiseksi järjestelmä vastaanottaa muutoksia tekeviä pyyntöjä myös GET-metodia käyttäen, mutta se on suoraan ristiriidassa HTTP-määrityksen kanssa. GET-metodin salliminen muokkauspyynnöissä helpottaa CSRF-hyökkäyksen käynnistystä, kuten luvussa 2.2 olevasta esimerkistä voi havaita. Tästä syystä järjestelmässä tulisi tulevaisuudessa estää muutoksia tekevien pyyntöjen lähetys GET-metodia käyttäen.

Kuten edellisissäkin tapauksissa, perinteisten HTML-elementtien suorasta käytöstä ha-

luttiin eroon, sillä ne eivät mahdollista keskitettyjen muutoksien tekemistä. Suoran linkin toteutusta varten järjestelmään tehtiin oma am:execute-JSP-merkintä, jonka avulla suorat linkit tulostettiin käyttöliittymään. Muutoksia tekevän pyynnön pitäisi määritysten mukaan käyttää POST-metodia. Tämän käyttö edellyttää joko AJAX-pyynnön tai lomakkeen käyttöä. Tavallisen lomakkeen käyttö tässä kohdassa ei ole mahdollista, koska suora linkki voi sijaita lomakkeen sisällä, eivätkä sisäkkäiset lomakkeet ole HTML:ssä tuettuja.

Nopean haun perusteella muokkausta tarvitsevia käyttöliittymäsivuja oli yli 200. Suoraa linkkiä varten toteutettu am:execute-JSP-merkintä ei käytä loppumerkkiä, joten sivujen muokkauksen automatisointi oli mahdollista ja siihen päädyttiin muokattavien sivujen suuren lukumäärän takia. Sivujen muokkausta varten toteutettiin yksinkertainen Python-skripti, joka käy käyttöliittymätiedostot läpi ja toteuttaa muokkaukset automaattisesti. Skripti toimii yksinkertaisuudessaan seuraavasti:

1. Skripti etsii .jsp ja .jspf -päätteisiä tiedostoja annetulta polulta ja lukee tiedostot rivi riviltä
2. Jos tiedoston rivillä esiintyy teksti servlet/Action, skripti muokkaa kyseistä riviä
3. Riviltä poimitaan Action-servletille lähetettävän pyynnön parametrit
4. Parametrien perusteella luodaan korvaava am:execute -merkintä ja korvataan sillä koko rivi.

Tietyissä poikkeustapauksissa skripti ei osannut luoda oikeanlaista am:execute-merkintää, joten käyttöliittymiin jouduttiin tekemään pieni määrä muutoksia käsin.

Kun suurin osa käyttöliittymistä oli muokattu käyttämään CSRF-tunnistetta, tarkistuksien yhteyteen lisätty sähköpostin lähetys aktivoitiin. Tämän jälkeen järjestelmä lähettää pyynnöistä, joissa CSRF-avain tai Referer-otsake on virheellinen, tiedon suoraan ylläpidolle sähköpostina. Tässä vaiheessa työ siirtyi kuuden kuukauden seurantavaiheeseen, jossa sähköpostiin tulleiden ilmoitusten perusteella kerättiin ylös sivuja, jotka tarvitsevat vielä muokkausta CSRF-suojausta varten.

6.3.4 CSRF-tunnusten käsittely

CSRF-tunnuksen elinikä on rajoitettu, jolloin aikaikkuna tunnuksen vuotamiselle tai murtamiselle pienenee ja järjestelmän turvallisuus siten kasvaa. CSRF-tunnusten uusimista varten järjestelmän taustaprosesseihin tehtiin tarkistus, joka etsii tietyin väliajoin käyttäjät, jotka ovat olleet tarpeeksi kauan epäaktiivisina ja uusii näiden CSRF-tunnuksen. Tunnusten uusimista ei toteuteta, kun käyttäjä on aktiivisesti kirjautuneena järjestelmään, sillä CSRF-tunnuksen vaihtaminen kesken istunnon haittaisi järjestelmän käyttöä. Esimerkiksi lomakkeet, joiden täyttö on, olisi pakko täyttää uudelleen, mikäli CSRF-tunnus vaihtuu kesken käytön.

7 Johtopäätökset

Työn tavoitteena oli suojata järjestelmä CSRF-hyökkäyksiä vastaan. Samalla päivitettiin selkeästi vanhentunutta koodia ja otettiin järjestelmänlaajuisesti käyttöön osassa järjestelmää oleva ohjelmistokehys. Työn kuluessa muutoksia tehtiin reilusti yli 200:lle käyttöliittymäsivulle, joista osa jouduttiin tekemään alusta asti uusiksi. Lopputuloksena järjestelmä saatiin suojattua CSRF-hyökkäyksiä vastaan ja koodipohjan kunto parani tuntuvasti.

Järjestelmän suojaus CSRF-hyökkäystä vastaan osoittautui huomattavasti työläämmäksi, kuin oli ennalta odotettavissa. Suurin syy tähän oli järjestelmän koodipohjan heikko kunto, joka teki käyttöliittymäsivujen muokkauksesta työlästä ja toisaalta muokkauksen automatisoinnista tehotonta. Käyttöliittymiä, joissa koodi oli vanhaa, jouduttiin muokkaamaan laajasti, mikä lisäsi mahdollisten virheiden määrää ja aiheutti siten suurempaa tarvetta testaukselle. Käyttöliittymien testausta ei ole järjestelmässä automatisoitu, joten se täytyy tehdä käsin. Järjestelmän käyttöliittymäsivujen suuri lukumäärä tekee käsin testauksesta hidasta, koska kunnollisten testitapauksien luonti on hyvin työlästä.

Työn sujumisen kannalta suunnitelmallisuudella on suuri merkitys. Tätä työtä ei suunniteltu kovin yksityiskohtaisesti esimerkiksi käyttöliittymämuutosten osalta. Jälkeenpäin tarkasteltuna tämä olisi voinut olla hyödyllistä, koska silloin suoritetuista ja suoritettavista muutoksista olisi muodostunut parempi kokonaiskuva. Tämä olisi helpottanut työn etenemisen seurantaa ja antanut selkeämmän kokonaiskuvan tehdyistä muutoksista. Muutosten jäljitettävyys ja kokonaiskuva on edelleen rakennettavissa versionhallinnan avulla, mutta se on jälkeenpäin tehtynä työlästä.

Tässä työssä käytettäväksi valikoitunut suojaustapa lyötiin lukkoon hyvin varhaisessa vaiheessa työtä. Järjestelmän ominaisuudet ja käyttötapaukset rajasivat käyttökelpoisten suojausten lukumäärän hyvin pieneksi, jolloin toteutettavaksi valikoitui CSRF-suojauksen klassisin tapa. Järjestelmään toteutettu suojaus on kuitenkin hyväksi todettu, eikä järjestelmässä ole muita rajoittavia tekijöitä, joiden perusteella esimerkiksi istuntoriippuvaista suojaustapaa olisi kannattanut suosia.

Sisäänkirjautumiseen kohdistuvaan CSRF-hyökkäykseen ei otettu tässä työssä lainkaan kantaa. Palvelun luonteesta johtuen on epätodennäköistä, että hyökkääjä saavuttaisi mitään hyötyä huijaamalla käyttäjää kirjautumaan jonkun toisen käyttäjän tunnuksilla. Mikäli tätä hyökkäystä vastaan toteutetaan tulevaisuudessa suojaus, tehdään se todennäköisesti Referer-otsakkeen tarkistuksella.

Tämä insinöörityö rajautui siten, että järjestelmän katsottiin olevan suojattu CSRF-hyökkäystä vastaan, kun järjestelmä edellytti kaikkia tallennuksia tehdessä oikean tunnisteen ja vastaavasti kaikki järjestelmän käyttöliittymät lisäsivät tunnisteen omiin pyyntöihinsä. Järjestelmään luotiin edellytykset CSRF-suojaukselle, mutta suojaukseen vaikuttaviin muihin haavoittuvuuksiin ei otettu kantaa. Käytännössä esimerkiksi XSS-haavoittuvuus mahdollistaa kaikkien muiden paitsi Challenger-response-tyyppisten suojausten kiertämisen.

Järjestelmään aikaisemmin tehdyn arvioinnin perusteella on tiedossa, että sen tietoturvassa on parantamista. Tässä työssä toteutetut parannukset ovat yksi osa laajempaa järjestelmän tietoturvan päivitystä. Lopuksi järjestelmän tietoturvasta pyritään varmistumaan hyödyntäen yrityksen sisäistä tietotaitoa ja erilaisia automatisoituja työkaluja. Järjestelmän tietoturvatason lopullinen auditointi tullaan ostamaan palveluna ulkopuoliselta asiantuntijayritykseltä. Järjestelmän kunnollinen penetraatiotestaus edellyttää tietotaitoa, jota yrityksen sisällä ei ole käytössä. Tavoitteena on myös saada ulkopuolisen tahon myöntämä tietoturvasertifiointi osoituksena tietoturvan korkeasta tasosta. Näiden toimenpiteiden päätteeksi järjestelmän tietoturvan odotetaan olevan korkealla tasolla.

Lähteet

1 Lehtinen L. Haktivismin nousu yllätti tietomurtojen asiantuntijat; 2013. Available from: http://suomenkuvalehti.fi/jutut/kotimaa/haktivismin-nousu-yllatti-tietomurtojen-asiantuntijat/ [cited April 16, 2015].

2 Ajantasainen lainsäädäntö. finlex.fi; 2014. Available from: http://www.finlex.fi/fi/laki/ajantasa/1999/19990523 [cited October 14, 2014].

3 KOMISSION ASETUS (EU) N:o 611/2013. Euroopan komissio; 2013. Available from: http://eur-lex.europa.eu/legal-content/FI/TXT/?uri=uriserv:OJ.L_.2013.173.01.0002.01.FIN [cited April 16, 2015].

4 How does the data protection reform strengthen citizens' rights?. Euroopan komissio; 2012. Available from: http://ec.europa.eu/justice/data-protection/document/review2012/factsheets/2_en.pdf [cited April 16, 2015].

5 Auger R. The Cross Site Request Forgery FAQ; 2010. Available from: http://www.cgisecurity.com/csrf-faq.html [cited February 21, 2015].

6 Hypertext Transfer Protocol (HTTP/1.1): Message Syntax and Routing. Internet Engineering Task Force; 2014. Available from: http://tools.ietf.org/html/rfc7230 [cited January 30, 2015].

7 HTTP State Management Mechanism. Internet Engineering Task Force; 2011. Available from: http://tools.ietf.org/html/rfc6265 [cited January 30, 2015].

8 Blatz J. CSRF: Attack and Defense. McAfee; 2011. Available from: http://www.mcafee.com/us/resources/white-papers/wp-csrf-attack-defense.pdf [cited April 19, 2015].

9 HTML <img> tag. W3 Schools;. Available from: http://www.w3schools.com/tags/tag_img.asp [cited February 21, 2015].

10 XSS (Cross Site Scripting) Prevention Cheat Sheet. Open Web Application Security Project;. Available from: https://www.owasp.org/index.php/XSS_(Cross_Site_Scripting)_Prevention_Cheat_Sheet [cited April 29, 2015].

11 MySpace Worm Explanation. Samy Kamkar; 2005. Available from: http://namb.la/popular/tech.html [cited February 11, 2015].

12 Adam Barth JCM Collin Jackson. Robust Defenses for Cross-Site Request Forgery. Stanford University; 2008. Available from: https://sparrow.ece.cmu.edu/group/731-s11/readings/csrf.pdf [cited October 14, 2014].

13 Grossman J. I used to know what you watched, on YouTube;. Available from: http://jeremiahgrossman.blogspot.ca/2008/09/i-used-to-know-what-you-watched-on.html [cited April 29, 2015].

14 Cross-Site Request Forgery. Open Web Application Security Project; 2014. Available from: https://www.owasp.org/index.php/Cross-Site_Request_Forgery_(CSRF) [cited October 15, 2014].

15 Ruby on Rails Security Guide. Ruby on Rails;. Available from: http://guides.rubyonrails.org/security.html [cited October 16, 2014].

16 Andrew Bortz AC Adam Barth. Origin Cookies: Session Integrity for Web Applications; 2011. Available from: http://www.adambarth.com/papers/2011/bortz-barth-czeskis.pdf [cited April 16, 2015].

17 Public Suffix List. Mozilla Foundation;. Available from: https://publicsuffix.org/list/effective_tld_names.dat [cited April 28, 2015].

18 Barth A. The Web Origin Concept; 2011. Available from: https://tools.ietf.org/html/rfc6454 [cited April 18, 2015].

19 Open Web Application Security Project; 2014. Available from: https://www.owasp.org/index.php/Cross-Site_Request_Forgery_(CSRF)_Prevention_Cheat_Sheet [cited April 19, 2015].

20 Prototype v1.7.2 API documentation. Prototype; 2014. Available from: http://api.prototypejs.org/ajax/ [cited April 19, 2015].

21 Ofer O. AJAX Hammer, Harnessing AJAX for Dynamic CSRF; 2012. Available from: http://hasc-research.googlecode.com/files/AJAX%20Hammer%20-%20Harnessing%20AJAX%20for%20%28Direct%29%20Dynamic%20CSRF.pdf [cited March 18, 2015].

22 Same-origin policy. Mozilla Developer Network; 2015. Available from: https://developer.mozilla.org/en-US/docs/Web/Security/Same-origin_policy [cited April 19, 2015].

23 Airey D. WARNING: Google's Gmail security failure leaves my business sabotaged; 2007. Available from: http://www.davidairey.com/google-gmail-security-hijack/ [cited April 23, 2015].