LOKITUKSEN JA MONITOROINNIN HYÖDYLLISYYS VERKKOSOVELLUKSIIN KOHDISTUVIEN HYÖKKÄYKSIEN TUNNISTAMISESSA

Panu Parviainen

Pro Gradu -tutkielma

ITÄ-SUOMEN YLIOPISTO

Tietojenkäsittelytieteen laitos

Tietojenkäsittelytiede

Lokakuu 2018

ITÄ-SUOMEN YLIOPISTO, Luonnontieteiden ja metsätieteiden tiedekunta, Joensuu
Tietojenkäsittelytieteen laitos
Tietojenkäsittelytiede

Opiskelija, Panu Parviainen: Lokituksen ja monitoroinnin hyödyllisyys verkkosovelluksiin kohdistuvien hyökkäyksien tunnistamisessa
Pro Gradu -tutkielma, 51s., ei liitteitä
Pro Gradu -tutkielman ohjaaja: Professori Pasi Fränti
Lokakuu 2018

Verkkosovellukseen kohdistuvan hyökkäyksen havaitseminen ajoissa on elintärkeää, etenkin kun vaakakupissa on sovelluksen käyttäjien henkilökohtaisten tietojen paljastuminen. Tässä Pro Gradu -tutkielmassa on tarkoitus selvittää, ovatko lokitus ja monitorointi käyttökelpoisia keinoja verkkosovelluksiin kohdistuvien verkkohyökkäysten tunnistamisessa ja voidaanko niiden avulla lyhentää hyökkäysten havaitsemiseen kuluvaa aikaa. Kysymykseen pureudutaan pääasiassa *Open Web Application Security Project* -säätiön ylläpitämän *Top 10 Kriittisintä Verkkosovellusten Tietoturvariskiä* -listauksen ja sen ympärille rakennettujen materiaalien pohjalta ja innoittamana. Tutkimuksessa testipalvelimelle pystytetään valvonta, joka monitoroi palvelimelle asennettuihin verkkosovelluksiin kohdistuvaa verkkoliikennettä, lokittaa liikenteen ja laukaisee hälytyksen, mikäli hyökkäys havaitaan. Valvonnan toimivuutta testataan suorittamalla palvelimelle asennettujen sovellusten tietoturvatestaus, joka simuloi sovelluksiin kohdistuvaa verkkohyökkäystä. Tutkielmassa kerätyn näytön perusteella voidaan lokitusta ja monitorointia pitää tehokkaana keinona hyökkäyksien havaitsemisessa ja havaitsemiseen kuluvan ajan lyhentämisessä. Testauksen aikana suoritetuista 29 hyökkäyksestä monitorointi havaitsi 93,1 %, ja hyökkäyksistä 89,7 % johti hälytyksen laukaisemiseen. Tulosten perusteella suosittelen lokitusta ja monitorointia osaksi verkkosovelluksen tietoturvakerrosta.

Avainsanat: verkkosovellus, tietoturva, lokitus, monitorointi, haavoittuvuus

ACM-luokat (ACM Computing Classification System, 1998 version):

C2.3, H.3.5, H.5.4, K.6.5

UNIVERSITY OF EASTERN FINLAND, Faculty of Science and Forestry, Joensuu
School of Computing
Computer Science

Student, Panu Parviainen: Usefulness of logging and monitoring in detecting web application attacks
Master's Thesis, 51p., no appendices
Supervisors of the Master's Thesis: Professor Pasi Fränti
October 2018

Detecting attacks targeting web applications is vital when the objective is to protect the private data of the users. This thesis aims at determining if logging and monitoring are applicable methods for detecting web application attacks and if they can be used to reduce the time it takes to detect these attacks. This study uses the *Open Web Application Security Project* foundation's *Top 10 Most Critical Web Application Security Risks* listing, and the materials built around it, as primary sources and inspiration. In course of this thesis logging and monitoring is set up to a test server hosting two web applications. Logging and monitoring are tested by performing a security testing which simulates a real-world web application attack. Based on the data collected during the tests logging and monitoring can be deemed applicable for detecting web application attacks and reducing the time it takes to detect the attacks. 93.1 % of the 29 individual attacks performed during the testing were detected whereas 89.7 % of the attacks triggered an alarm. Based on these results implementing logging and monitoring as part of the web applications security layer can be recommended.

Keywords: web application, security, logging, monitoring, vulnerability

CR Categories (ACM Computing Classification System, 1998 version):

C2.3, H.3.5, H.5.4, K.6.5

Lyhenneluettelo

CAPTCHA	Completely Automated Public Turing test to tell Computers and Humans Apart; kuvavarmenne ihmisen ja tietokoneen erottamiseksi
CRS	Core Rule Set; palomuurisäännöstö
CSP	Content Security Policy; verkkosivun sisällön turvallisuusmääritykset
CSRF	Cross-site Request Forgery; verkkosovelluksien tietoturva-aukko
DAST	Dynamic Application Security Testing; dynaaminen sovelluksen tietoturvatestaus
DVWA	Damn Vulnerable Web Application; testauksessa käytetty verkkosovellus
DOM	Document Object Model; dokumenttioliomalli
DSS	Data Security Standard; tiedon turvallisuusstandardi
FTP	File Transfer Protocol; tiedostojen siirtämiseen tarkoitettu protokolla
HTTP	HyperText Transfer Protocol; hypertekstin siirtoprotokolla
HQL	Hibernate Query Language; Hibernate -ohjelmistokehyksen kyselykieli
OWASP	Open Web Application Security Project; avoin verkkosovelluksien tietoturvaprojekti
PCI	Payment Card Industry; maksukorttiteollisuus
PVI	Passive Vulnerability Identification; passiivinen uhkien tunnistaminen
SSH	Secure Shell; salattuun tiedonsiirtoon tarkoitettu protokolla
URL	Uniform Resource Locator; internetosoite
WAF	Web Application Firewall; verkkosovelluspalomuuri
WASC	Web Application Security Consortium; verkkosovelluksien tietoturvayhteisö
W3C	World Wide Web Consortium; WWW:n standardeja ja suosituksia kehittävä ja ylläpitämä yhteisö
XSS	Cross-Site Scripting; verkkosovellusten tietoturva-aukko
ZAP	Zed Attack Proxy; Tietoturvan testaustyökalu

Sisällysluettelo

1 Johdanto

Arkielämän asioiden hoitaminen internetissä on yleistynyt kuluneen vuosikymmenen aikana. Ihmiset maksavat laskunsa, varaavat matkansa ja kommunikoivat toistensa kanssa verkossa. Kaikki tämä tapahtuu *verkkosovellusten* avulla. Koska verkkosovellukset ovat jatkuvasti saatavilla internetissä ja niissä liikkuu paljon mitä erilaisinta käyttäjien dataa, ovat ne houkuttelevia kohteita hakkereille ja verkkorikollisille.

Viime vuosina on uutisoitu useista tietomurroista, joissa käyttäjien tietoja on päätynyt rikollisten käsiin. Tyypillistä näissä tapauksissa on se, että murtoa ei ole huomattu kovinkaan nopeasti, vaan pahimmassa tapauksessa vasta kuukausien kuluttua, kun varastettuja tietoja alkaa ilmestyä internetin laittomille kauppapaikoille. Hyvä esimerkki tällaisesta verkkosovelluksen kautta tapahtuneesta laajasta tietomurrosta on vuonna 2017 tapahtunut luottorekisteriyhtiö Equifaxin tietomurto, jossa hakkereiden käsiin päätyi 143 miljoonan ihmisen tietoja (HS, 2017). Yhtiön ilmoituksen mukaan hakkerit murtautuivat sen järjestelmiin toukokuun puolivälissä 2017, mutta murto havaittiin vasta kesäkuun lopussa (Equifax, 2017).

Tämä trendi on huomattu myös verkkosovellusten tietoturvan parantamiseen pyrkivässä *Open Web Application Security Project* -säätiössä (OWASP). Säätiö ylläpitää listausta kymmenestä yleisimmästä verkkosovellusten tietoturvahaavoittuvuudesta. Vuoden 2017 listauksen ensimmäiseen versioon lisättiin maininta riittämättömästä hyökkäyksiltä suojautumisesta. Tällä tarkoitetaan sitä, että verkkosovelluksen ylläpitäjillä ei ole keinoja havaita verkkosovellukseen kohdistuvia hyökkäyksiä ja mahdollisia onnistuneita hyökkäyksiä ajoissa tai ollenkaan. Lisäys sai kuitenkin julkisessa keskustelussa kriittisen vastaanoton ja sitä pidettiin enemmän kaupallisten toimijoiden lobbauksen seurauksena kuin listaukselle todella kuuluvana kohteena (Ragan, 2017). Kritiikin seurauksena OWASP muokkasi listauksen lopulliseen julkaisuun maininnan muotoon ”*riittämätön lokitus ja monitorointi*” (OWASP, 2017). Lokituksella tarkoitetaan tässä verkkosovellukseen kohdistuvien HTTP- ja HTTPS-kutsujen kirjaamista loki- eli tekstitiedostoon. OWASP:in julkaisujen ja sitä seuranneen keskustelun innoittamana tässä Pro Gradu -tutkielmassa on tarkoituksena selvittää:

a) ovatko lokitus ja monitorointi tehokas keino verkkosovelluksiin kohdistuvien automaattisten hyökkäysten tunnistamisessa

b) voidaanko niiden avulla lyhentää hyökkäyksen havaitsemiseen kuluvaa aikaa

Tutkimuksessa keskitytään sellaisten hyökkäystyökaluilla suoritettavien hyökkäysten tunnistamiseen, jotka hyödyntävät verkkosovelluksessa olevia ohjelmointi- tai logiikkavirheitä. Tutkimuksessa ei huomioida esimerkiksi palvelimelle asennettavia haittaohjelmia, palvelunestohyökkäyksiä tai käyttäjän manipulointiin perustuvia tietojenkalasteluyrityksiä. Tutkimuksen ulkopuolelle rajataan myös sellaiset hyökkäykset ja haavoittuvuudet, joita ei voida havaita tai hyödyntää suoraan verkkosovelluksen avulla. Tällainen uhka on esimerkiksi verkkosovelluksen palvelimelle murtautuminen FTP-protokollaa hyödyntäen.

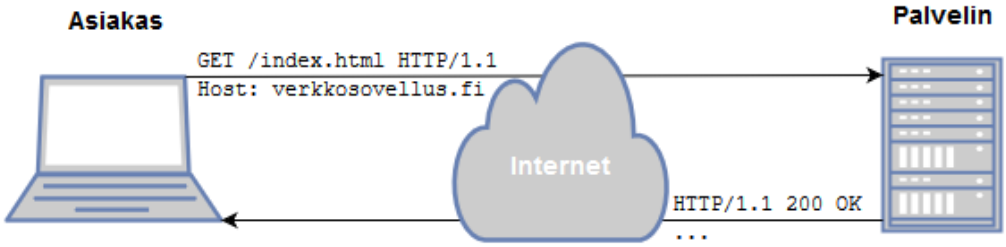
Tutkielma jakautuu kuuteen lukuun. Toisessa luvussa esitellään verkkosovellusten toimintaa ja toteutusta yleisellä tasolla. Tarkoituksena on pohjustaa lukua kolme, jossa esitellään OWASP:in Top 10 -listauksen pohjalta verkkosovellusten yleisimpiä tietoturvahaavoittuvuuksia, sekä antaa lukijalle riittävät pohjatiedot ymmärtää, missä osassa verkkosovellusten toteutusta esiteltävät haavoittuvuudet esiintyvät ja miten niiden hyödyntäminen verkkosovelluksen avulla on mahdollista. Luvussa kolme esiteltävät haavoittuvuudet pohjautuvat OWASP:in vuoden 2013 Top 10 -listaukseen. Vuoden 2013 listaus valikoitui rungoksi haavoittuvuuksien esittelylle, koska se sisältää enemmän ohjelmointi- ja logiikkavirheiden seurauksena syntyviä haavoittuvuuksia kuin uusin vuoden 2017 listaus, jossa on mukana myös enemmän aatteellisia ohjeita, kuten tutkielman motivaattorina toimiva *riittämätön lokitus ja monitorointi*.

Luvussa neljä esitellään kirjallisuuden pohjalta verkkosovelluksen tietoturvakerros sekä tapoja ja työkaluja sen toteuttamiseen. Luvussa ei mennä yksityiskohtaisiin ohjeisiin tietoturvakerroksen toteuttamisesta, sillä nämä ohjeet löytyvät luvun lähteinä käytetystä kirjallisuudesta. Luvussa viisi esitellään tutkielman käytännönosuuden toteutus sekä testaus ja luvussa kuusi saadut tulokset. Luku seitsemän tarjoaa vielä yhteenvedon tutkielmasta ja sen tuloksista.

2 Verkkosovellukset

Tässä tutkielmassa verkkosovelluksella tarkoitetaan mitä tahansa vuorovaikutteista verkkosivua, jonka sisältöön käyttäjä voi vaikuttaa syöttämillään tiedoilla. Verkkosovellusten kirjo on erittäin laaja ja sovellusten monimutkaisuus vaihtelee esimerkiksi YouTube-palvelusta kiinalaisen ravintolan verkkosivuun, jossa ainoa interaktio on pöytävarauksen tekeminen.

Verkkosovellukset toteuttavat asiakas–palvelin -arkkitehtuuria, jossa asiakasohjelma on verkkoselain käyttäjän päätelaitteella. Kaikki asiakkaan ja palvelimen välinen kommunikaatio tapahtuu *HyperText Transfer Protocol*:n (*HTTP)* tai sen salatun version *HyperText Transfer Protocol Secure:n (HTTPS)* avulla. Asiakas lähettää palvelimelle kutsun, jossa määritellään mitä resurssia tai sivua palvelimelta halutaan katsella. Palvelin käsittelee asiakkaan pyynnön ja lähettää tilanteeseen sopivan vastauksen.



Kuva 1. Asiakkaan ja palvelimen välinen kommunikaatio.

Kuvassa 1 asiakas lähettää palvelimelle `verkkosovellus.fi` GET-kutsun, jossa pyytää palvelimelta sivua `index.html`. Palvelin vastaa kutsuun pyydetyllä sivulla ja verkkoselain näyttää käyttäjälle sivun palvelimen lähettämän HTML-tiedoston perusteella. HTML:n avulla ei voida luoda vuorovaikutteisia ominaisuuksia, vaan sen avulla kuvaillaan vain sivun rakenne eli tekstin, kuvien ja painikkeiden sijainnit verkkosivulla.

Verkkosovellusten vuorovaikutteisuus voidaan toteuttaa dynaamisesti joko asiakasohjelmassa tai palvelimella. Käytännössä selain voi luoda sivun itse esimerkiksi JavaScript-koodista tai palvelin voi luoda selaimelle valmiin sivun. Asiakaspään ohjelmointikielenä verkkosovelluksissa toimii tällä hetkellä pääsääntöisesti JavaScript ja palvelimella PHP, ASP.NET tai Java (W3Tech, 2018).


```
<!DOCTYPE html>
<html>
  <body>
    <p>Hello World!</p>
  </body>
</html>
```

Kuva 2. Yksinkertainen verkkosivu ja sen lähdekoodi.

Kuvan 2 esittämä verkkosivu voidaan toteuttaa dynaamisesti JavaScript:lla asiakaspäässä tai muodostaa valmis staattinen sivu palvelimella esimerkiksi PHP:llä.

```
<!DOCTYPE html>
<html>
  <body>
    <?php echo '<p>Hello World!</p>'; ?>
  </body>
</html>
```

Esimerkki 1. Kuvan 2 mukaisen sivun luova PHP-koodi.

Esimerkin 1 esittämä PHP-koodi luo palvelimella valmiiksi kuvan 2 mukaisen staattisen HTML:n, jonka perusteella selain piirtää sivun. Samanlainen verkkosivu voidaan tuottaa myös lähettämällä selaimelle esimerkin 2 mukainen HTML-sivu, jossa sivun sisältämä koodi suoritetaan vasta selaimessa JavaScript-komennon `document.write("<P>Hello World!</p>");` avulla.

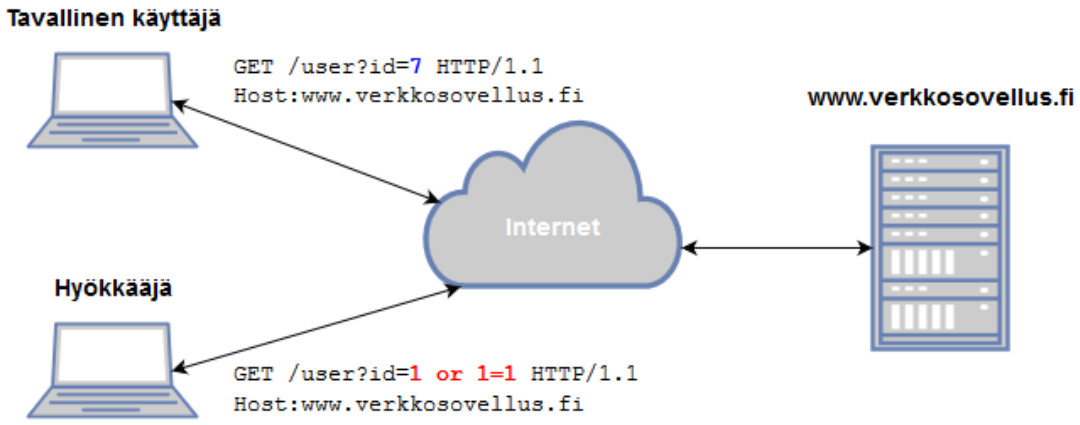
```
<!DOCTYPE html>
<html>
  <body>
    <script>
      document.write("<P>Hello World!</p>");
    </script>
  </body>
</html>
```

Esimerkki 2. Kuvan 2 mukaisen sivun luova HTML- ja JavaScript-koodi.

3 Verkkosovellusten yleisimmät tietoturvahaavoittuvuudet

Tässä luvussa esitellään kymmenen yleisintä verkkosovellusten tietoturvahaavoittuvuutta ja menetelmiä, joilla haavoittuvuuksia voidaan käyttää hyväksi verkkosovellusten tietoturvan murtamiseksi. Esiteltävät haavoittuvuudet perustuvat OWASP:in laatimaan listaukseen vuodelta 2013. Tietoja haavoittuvuuksien hyödyntämiseksi on täydennetty Mike Sheman (2010, 2012) sekä Dafydd Stuttardin ja Marcus Pinton (2011) teosten pohjalta.

Haavoittuvuuksilla tarkoitetaan sellaisia ohjelmointi- ja logiikkavirheitä verkkosovellusten toteutuksessa, joiden seurauksena sovellus saadaan suorittamaan toimintoja, joita sovelluksen tekijät eivät ole tarkoittaneet sen suorittavan. Tällainen virhe on esimerkiksi se, että käyttäjän syötettä ei tarkasteta ennen kuin sitä käytetään parametrina SQL-kyselyssä.



Kuva 3. Verkkosovelluksen haavoittuvuuden hyödyntäminen.

Tämän luvun esimerkeissä käytetään hyökkäyksien havainnollistamiseksi esimerkkeinä kuvitteellisen verkkosovelluksen *verkkosovellus.fi* toimintoja. Kuvan 3 mukaisesti sovellusten haavoittuvuuksien hyödyntäminen tapahtuu samalla tavalla kuin sovelluksen normaali käyttö. Hyökkääjä kutsuu samoja sovelluksen metodeja kuin tavallinen käyttäjä, mutta sisällyttää kutsuihin liitettävään syötteeseen haavoittuvuuksia hyödyntäviä komentoja. Hyökkääjät voivat myös luoda linkkejä haavoittuville sivuille ja varastaa muiden sovellusten käyttäjien tietoja huijaamalla käyttäjiä avaamaan linkkejä (ks. luvut 3.2 ja 3.8).

3.1 Injektio

Injektio on palvelinpään haavoittuvuus, jossa käyttäjän syöte välitetään sellaisenaan kyselyn tai komentosarjan mukana kääntäjälle. Injektiohaavoittuvuuteen kohdistuvassa hyökkäyksessä hyödynnetään ohjelmiston käyttämän kääntäjän syntaksia ja pyritään vaikuttamaan sen toimintaan. Haavoittuvuuden avulla voidaan pyrkiä varastamaan tai tuhoamaan sovelluksen sisältämää dataa tai saamaan pääsy muutoin suojattuun dataan. Yleisiä injektion kohteita ovat SQL-kyselyt, komentorivikomennot ja LDAP -komennot (*Lightweight Directory Access Protocol*) (OWASP, 2013). Injektion avulla voidaan myös toteuttaa palvelunestohyökkäys syöttämällä sovellukseen niin raskaita tietokantakutsuja, että ne hidastavat tai täysin lamauttavat sovelluksen toiminnan (Shema, 2012).

```
public String createQuery(HttpServeletRequest request){
    return ”SELECT * FROM users WHERE id = ”
           + request.getParameter("id");
}
```

Esimerkki 3. Javalla toteutettu tietokantakyselyn luonti.

Sovelluksessa on injektiohaavoittuvuus, kun sovellukseen tulevaa dataa ei tarkasteta ennen sen lähettämistä kääntäjälle (OWASP, 2013). Esimerkki 3 on kuvassa 3 esitetyn `GET user?id=tunniste` metodin palvelinpään toteutus Javalla. Esimerkissä kutsun parametri `id` lisätään SQL-kyselyyn suoraan + -operaation avulla ilman minkäänlaista tarkastusta, jolloin metodi jää alttiiksi injektiolle.

Esimerkin 3 mukainen kyselyn luonti luo kuvan 3 tavalliselle käyttäjälle kyselyn

```
SELECT * FROM users WHERE id = 7;
```

Kysely on sovelluksen toimintalogiikan mukainen ja palauttaa tietokannasta yhden käyttäjän tiedot. Esimerkin 3 hyökkääjälle luotava kysely

```
SELECT * FROM users WHERE id = 1 or 1 = 1
```

on kuitenkin sovelluksen toimintalogiikan vastainen, sillä sen avulla käyttäjä saa valittua tietokannasta useita käyttäjiä. Koska ehto `1 = 1` on aina tosi, valitaan taulusta SQL-syntaksin mukaisesti kaikki rivit yhden rivin sijaan.

Esimerkin 3 mukainen hyökkäys on yksinkertainen selkokielinen SQL-komento. Sivistyneemmät hyökkäykset pyrkivät monimutkaistamaan injektiokoodia esimerkiksi piilottamalla osia siitä heksadesimaalimerkkijonon sisälle, jotta sen automaattinen havaitseminen olisi vaikeampaa (Shema, 2012).

Injektiohaavoittuvuuksia voidaan helposti estää tarkastamalla kaikki käyttäjältä tuleva syöte ennen kuin syötettä käytetään käskyn tai kyselyn osana (OWASP, 2013). Sheman (2012) mukaan käyttäjän syötteestä saadaan turvallinen noudattamalla seuraavia sääntöjä syötteen tarkastamisessa:

- Muunna kaikki data samaan merkistökoodaukseen, esimerkiksi UTF-8:aan
- Toteuta datamuunnokset, kuten Uniform Resource Identifier-koodaus (URI) ja -tulkinta aina yhdenmukaisesti
- Tarkasta, että annettu syöte on oikeantyyppinen (numero, merkkijono, PDF-tiedosto)
- Tarkasta, että syöte on oikeanmuotoinen (sähköpostiosoite, puhelinnumero, rekisteritunnus)
- Älä poista syötteestä kiellettyjä arvoja vaan hylkää koko syöte

Sheman ohjeistusta noudattamalla kuvan 3 metodin toteuttavassa verkkosovelluksessa tulisi siis tarkastaa, että parametrista `id` saatava syöte on positiivinen kokonaisluku. Sovelluksen käyttäjämäärästä riippuen voitaisiin luvun pituus rajoittaa myös esimerkiksi viiteen merkkiin. Jos syöte ei vastaa annettuja määreitä, sitä ei tule käyttää kyselyssä vaan se on hylättävä.

Injektiohaavoittuvuuksia voidaan estää myös käyttämällä valmiita ohjelmistokehyksiä, kuten esimerkiksi *Hibernatea*, joissa on sisäänrakennettu käyttäjän syötteen validointi. Validointi varmistaa, että syötettä käsitellään datana eikä ohjelmakoodina. Ohjelmistokehyksiä käytettäessä on kuitenkin syytä kiinnittää huomiota siihen, että mikäli ohjelmoijat luottavat ohjelmistokehykseen liikaa tai eivät ymmärrä sen toimintaa kunnolla, ovat injektiot edelleen mahdollisia. (OWASP, 2013)

```
session.createQuery(
 "FROM User WHERE id="+ request.getParameter("id")
);
```

Esimerkki 4. Injektion mahdollistava HQL-kysely.

Esimerkki 4 on Hibernatella toteutettu kysely, jossa haetaan käyttäjiä tunnisteen perusteella. Sovelluskehyksen käytöstä huolimatta kysely mahdollistaa edelleen injektion, sillä createQuery-metodi ei pysty tarkastamaan parametreja. Esimerkissä 5 esitetään oikeaoppinen tapa käyttää Hibernate-ohjelmistokehystä lisäämällä parametrit syötteeseen setParameter-metodin avulla.

```
session.createQuery("FROM User WHERE id=:userId")
  .setParameter("userId",request.getParameter("id"));
```

Esimerkki 5. Injektion estävä HQL-kysely.

3.2 Rikkinäinen todennus ja istunnonhallinta

Todennus ja istunnonhallinta ovat menetelmiä tunnistaa verkkosovelluksen käyttäjät, erottaa heidät tosistaan ja liittää käyttäjien selaimessa tekemät toiminnot oikeaan istuntoon palvelimella. Tämä tapahtuu tallentamalla palvelimen luoma istuntotunniste käyttäjän asiakasohjelman muistiin, esimerkiksi selaimen evästeisiin, ja lähettämällä istuntotunniste palvelimelle jokaisen kutsun mukana (Shema, 2012). Todennus ja istunnonhallinta voidaan tietoisesti rikkoa esimerkiksi XSS- tai CSRF-hyökkäyksien avulla (Shema, 2012) tai ne voivat olla huonosti toteutettuja, jolloin ne altistavat verkkosovelluksen usealle erilaiselle hyökkäykselle (OWASP, 2013).

Kuva 4. Verkkosovelluksen tunnistautuminen

OWASP:in (2013) mukaan käyttäjien todennus ja istunnonhallinta voidaan katsoa rikkoutuneeksi esimerkiksi kun:

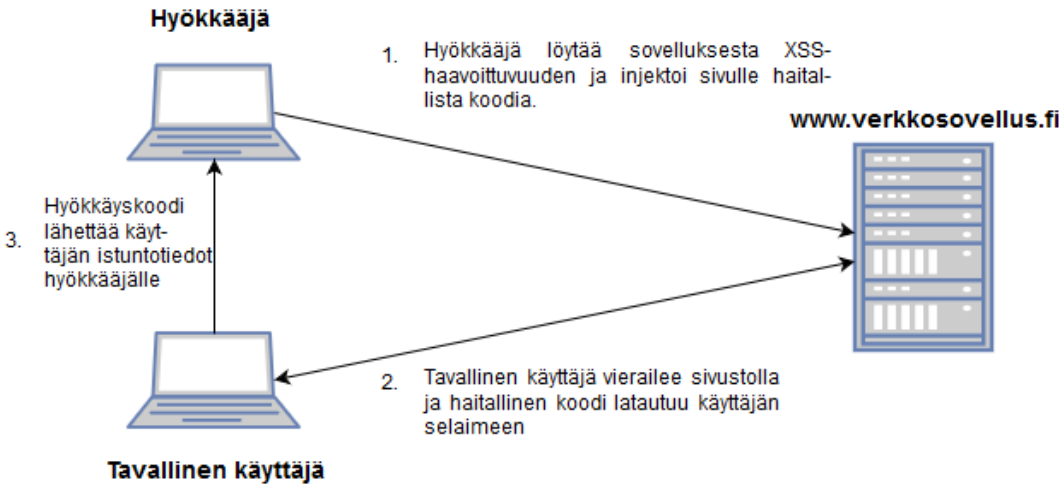
- Salasanat on tallennettu selkokielisinä
- Käyttäjätietoja voidaan arvata tai muuttaa heikkojen tilinhallintatoimintojen vuoksi (esimerkiksi tilin luominen, salasanan vaihto, salasanan palautus, heikot istuntotunnisteet)
- Istuntotunnisteet näkyvät URL:ssa
- Istuntotunnisteet ovat alttiita istuntoonkiinnittymishyökkäyksille
- Istuntotunnisteet eivät vanhene tai käyttäjän istunto-, todennus- ja erityisesti kertakirjautumistunnisteita ei mitätöidä kunnolla uloskirjautumisen yhteydessä
- Istuntotunnisteita ei vaihdeta onnistuneen kirjautumisen jälkeen, jolloin oikean tunnisteen arvaaminen on helpompaa
- Salasanoja, istuntotunnisteita tai muita valtuustietoja ei lähetetä TLS/SSL-salausprotokollaa käyttävän yhteyden yli vaan ne lähetetään selkokielisinä

Rikkinäistä todennusta ja istunnonhallintaa hyödyntäen hyökkääjät yrittävät saada haltuunsa muiden käyttäjien tilejä ja peittää omaa toimintaansa sovelluksessa. Saatuaan käyttäjän istunnon haltuunsa hyökkääjällä on samat oikeudet kuin varastetulla tilillä ja hän voi käyttää tiliä aivan kuin tilin oikea haltija. Rikkinäistä todennusta ja istunnonhallintaa voivat hyödyntää tuntemattomat ulkopuoliset henkilöt tai järjestelmän varsinaiset käyttäjät, jotka haluavat peittää omaa haitallista toimintaansa järjestelmässä. (OWASP, 2013)

Toimivan todennuksen ja istunnonhallinnan saavuttamiseen ei ole olemassa yhtä kattavaa ratkaisua, vaan se vaatii jatkuvaa ylläpitoa, testausta ja havaittujen puutteiden nopeaa korjaamista. Sovelluksen todennusta ja istunnonhallintaa voi parantaa käyttämällä ohjelmistokehystä, jossa on toteutettu istunnonhallinta ja todennus. Tällaisia ovat esimerkiksi *Spring Security*, *WordPress* ja *Express JS*. Istunnonhallinnan ja todennuksen suunnitteluun ja tarkastamiseen on olemassa myös erilaisia ohjeita ja standardeja, kuten OWASP:in tuottama *Application Security Verification Standard Project* (OWASP, 2018a). Ohjeet ja standardit auttavat kehittäjiä ja ylläpitäjiä kiinnittämään huomioita todennuksen ja istunnonhallinnan kannalta olennaisiin asioihin. (OWASP, 2013)

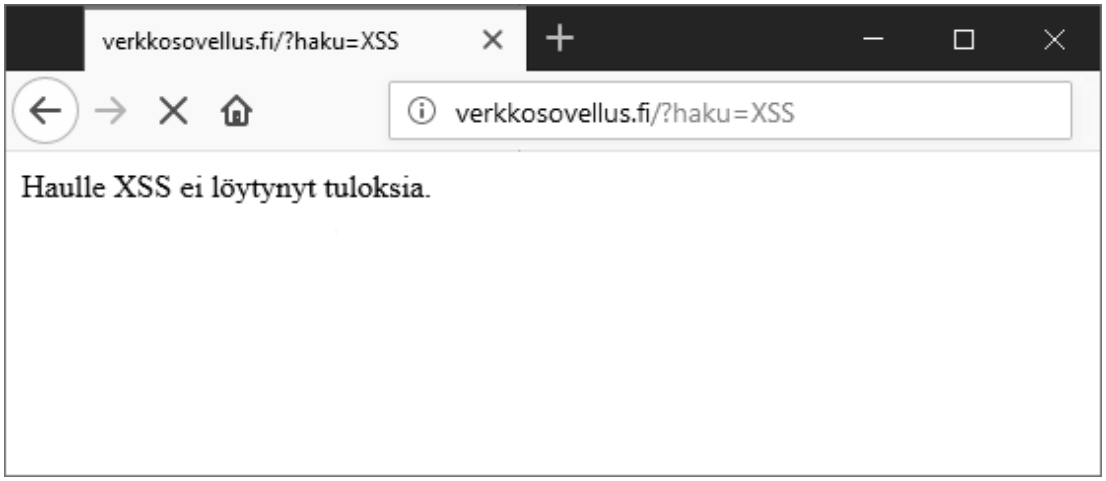
3.3 Cross-Site Scripting

Cross-Site Scripting, eli XSS-haavoittuvuus on injektiohaavoittuvuus, joka mahdollistaa ulkopuolisen koodin injektoimisen osaksi verkkosovelluksen asiakaspään ohjelmakoodia (OWASP, 2013). Kuvan 5 mukaisesti XSS-haavoittuvuuden avulla pyritään suorittamaan injektoitu koodi muiden verkkosovelluksen käyttäjien selaimissa. XSS-haavoittuvuuden avulla voidaan kaapata muiden sovelluksen käyttäjien istuntoja, varastaa käyttäjien tietoja tai ladata sivuston käyttäjien selaimiin haittaohjelmia (OWASP, 2013).



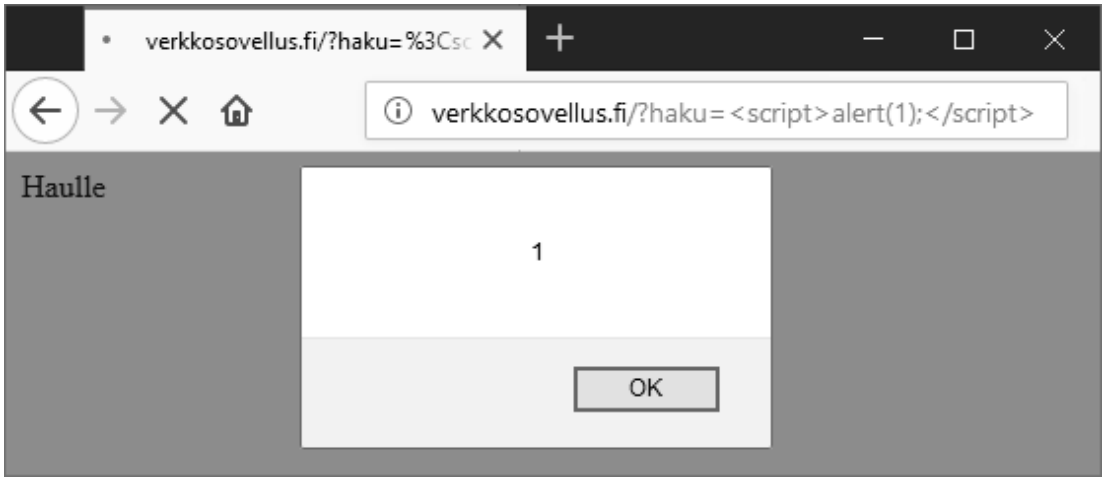
Kuva 5. XSS-haavoittuvuuden hyödyntäminen.

XSS-haavoittuvuudet voidaan jakaa ei-pysyviin ja pysyviin haavoittuvuuksiin. XSS-haavoittuvuus voi olla sekä asiakas- että palvelinpään koodissa (ks. luku 2). Ei-pysyvässä haavoittuvuudessa hyökkäyskoodia ei tallenneta pysyvästi, vaan haavoittuvuus on mahdollinen, kun käyttäjän syötettä käytetään välittömästi ja sellaisenaan sivun sisällön luomiseen. Pysyvässä haavoittuvuudessa hyökkäyskoodi taas tallennetaan pysyvästi esimerkiksi palvelimen tietokantaan ja hyökkäyskoodia palautetaan palvelimen normaalien vastausten mukana. (OWASP, 2013)



Kuva 6. Verkkosovellus.fi sivuston hakusivu.

Ei-pysyvässä XSS-haavoittuvuudessa käyttäjän syötettä käytetään sivun sisällön luomiseen ilman syötteen tarkastamista. Kuvan 6 verkkosovelluksessa on hakutoiminto, jossa hakusana lisätään hakutulosten mukana luotavalle sivulle. Jos parametrin `haku` arvoa ei tarkasteta ennen kuin se lisätään sivulle, on metodi altis ei-pysyvälle XSS-haavoittuvuudelle. Haavoittuvuutta voidaan todentaa esimerkiksi kutsumalla metodia arvolla `<script>alert(1);</script>`, jolloin hakusivulle aukeaa kuvan 7 mukainen ponnahdusikkuna.



Kuva 7. Hakusivulle tehty XSS-injektio.

Ei-pysyvää haavoittuvuutta hyödyntävät hyökkäykset käyttävät hyväkseen käyttäjien eri sivuja kohtaan tuntemaa luottamusta. Hyökkääjä voi esimerkiksi jakaa sähköpostilla sekä muiden sivustojen avulla linkkiä kuvan 7 osoittamalle sivulle ja toivoa, että muut käyttäjät avaisivat linkin, jolloin hyökkäyskoodi suoritettaisiin heidän selaimissaan. (OWASP, 2013)

Oikeissa hyökkäyksissä pyritään usein aiheuttamaan suurempaa vahinkoa kuin häiritsemään käyttäjiä ponnahdusikkunoilla. `Verkkosovellus.fi` sovelluksen hakutoiminnon avulla hyökkääjä voisi varastaa sovelluksen käyttäjän tietoja korvaamalla `haku` parametrin arvon esimerkin 6 script-tagilla. Tällöin linkin avanneiden käyttäjien selaimet lataisivat `haitallinen.js` tiedoston hyökkääjän palvelimelta. Latauksen jälkeen tiedoston sisältämä JavaScript-koodi suoritetaan käyttäjän selaimessa ja se lähettää käyttäjän istunnon tiedot hyökkääjän palvelimelle.

```
(function(){
   var image = new Image();
   image.src='http://hakkeri.fi/?c='+document.cookie
})();
```

```
<script src=”http://hakkeri.fi/haitallinen.js”>
```

Esimerkki 6. haitallinen.js-tiedoston sisältö ja tiedoston lataamiseen käytettävä script-tagi.

Pysyvässä XSS-haavoittuvuudessa hyökkäyskoodi tallennetaan pysyvästi palvelimelle ja se palautetaan aina osana palvelimen normaaleja vastauksia. Pysyvä hyökkäys on ei-pysyvää hyökkäystä tehokkaampi, sillä se vaikuttaa kaikkiin haavoittuvuuden sisältämällä sivulla vieraileviin käyttäjiin. Tyypillinen esimerkki ovat internetin keskustelupalstat, jonne käyttäjät voivat kirjoittaa viestejä. Jos palstalla on XSS-haavoittuvuus, voi hyökkääjä lisätä viestinsä loppuun esimerkissä 6 esitetyn script-tagin. Script-tagin sisältöä ei piirretä selaimessa, joten tavallisilla käyttäjillä ei ole keinoja havaita, että heidän selaimensa lataavat ja suorittavat `haitallinen.js` tiedoston, joka lähettää heidän istuntotunnisteensa hyökkääjän palvelimelle GET-kutsun avulla. (OWASP, 2013)

XSS-hyökkäyksiltä voi suojautua tarkastamalla kaiken käyttäjältä tulevan syötteen ja estämällä syötteen käyttämisen ohjelmakoodina. Syötteen käyttäminen ohjelmakoodina voidaan estää koodaamalla syötteen erikoismerkit, jolloin selain osaa käsitellä syötettä pelkkänä merkkijonona. Monissa tapauksissa sivustot kuitenkin haluavat antaa käyttäjilleen mahdollisuuden jakaa esimerkiksi foorumikirjoituksiin upotettuja YouTube-videoita tai toimivia linkkejä. Rikastetun sisällön tapauksessa XSS-hyökkäyksiä voidaan torjua käyttäen apuna erilaisia kirjastoja, jotka seulovat syötteistä haitallista koodia erilaisten algoritmien avulla. Tällaisia kirjastoja ovat esimerkiksi OWASP-säätiön AntiSamy ja Java HTML Sanitizer –projekti. (OWASP, 2013)

XSS-hyökkäyksiltä voidaan puolustautua myös Content Security Policy:n (CSP) eli sisällön turvallisuuspolitiikan avulla. CSP on W3C:n (2015) määrittelemä standardi, jonka mukaisesti verkkosivu määrittelee lähettämiensä vastausten HTTP-otsikoissa sivustolla käytetyt ja hyväksytyt sisältötyypit. Modernit verkkoselaimet noudattavat standardia ja jättävät lataamatta ja suorittamatta sisällön, joka ei vastaa sivun ylläpitäjän määritelmää. Tämä tarkoittaa sitä, että vaikka hyökkääjä saisi injektoitua verkkosovellus.fi-sivustolle esimerkin 6 mukaisen komentosarjan, ei tiedostoa `haitallinen.js` ladata ja suoriteta, ellei hakkeri.fi ole listattu luotettavaksi lähteeksi verkkosovellus.fi-ylläpitäjien toimesta. (OWASP, 2013)

3.4 Varmentamaton olioviite

Varmentamaton olioviite on palvelinpään haavoittuvuus, jonka avulla käyttäjät voivat selata sovelluksen dataa, jonka muutoin ei kuuluisi olla heidän saatavillaan. Olioviitteellä tarkoitetaan tilannetta, jossa käyttäjä käsittelee järjestelmän resursseja suoraan olion tunnisteen, eli esimerkiksi tietokantataulun id-kentän, perusteella. Varmentamattomalla olioviitteellä tarkoitetaan tilannetta, jossa käyttäjällä on käyttöoikeudet olioviitteen avulla suoritettavaan toimintoon, mutta toimintoa suoritettaessa ei varmisteta, että käyttäjällä on oikeus käsitellä täsmälleen sitä oliota, johon hän viittaa. Varmentamattoman olioviitteen tapauksessa hyökkääjät ovat yleensä järjestelmän varsinaisia käyttäjiä, jotka voivat nähdä tai muokata resursseja, joihin heillä ei tulisi olla oikeuksia. (OWASP, 2013)

Kuvan 3 (ks. sivu 5) mukainen URL mahdollistaa verkkosovelluksen käyttäjän tietojen hakemisen suoraan URL-parametrina annettavan tietokantatunnisteen perusteella. Toiminto on altis varmentamattomalle olioviitteelle, jos palvelin ei varmista, että järjestelmään kirjautuneella käyttäjällä on oikeudet juuri siihen käyttäjään, johon hän parametrilla viittaa. Kuvan osoittamassa tilanteessa verkkosovelluksen käyttäjät voisivat päästä katselemaan tai muokkaamaan muiden käyttäjien tilejä pelkästään tietämällä tai arvaamalla heidän käyttäjänimensä.

```
http://www.verkkosovellus.fi/user?id=89Te29eenu98Wa
```

Esimerkki 7. Tilinhallinta epäsuoran olioviitteen avulla

Suoria ja varmentamattomia olioviitteitä tulee välttää verkkosovelluksissa. Yksi vaihtoehto on korvata suorat olioviitteet istunto- tai käyttäjäkohtaisilla viitteillä esimerkin 7 osoittamalla tavalla. Generoitujen tunnisteiden käyttäminen vaatii tunnisteiden luomisen sekä tunnisteiden ja olioiden välisten yhteyksien ylläpitämistä (OWASP, 2013). Generoituja tunnisteita käytettäessä tulee myös huolehtia siitä, että generoidut arvot ovat todella sattumanvaraisia, eivätkä vain esimerkiksi kasvavassa järjestyksessä olevia numeroita tai heikosti generoituja satunnaislukuja, jolloin hyökkääjä voi helposti arvata seuraavia arvoja (Stuttard & Pinto, 2011). Olioviitteitä käytettäessä tulee kehittäjien siis aina huolehtia, että käyttäjän oikeudet kyseiseen olioon tarkastetaan ennen olion palauttamista (OWASP, 2013).

3.5 Väärin asetetut turvallisuusasetukset

Väärin asetetut turvallisuusasetukset voivat altistaa verkkosovelluksen lähestulkoon mille tahansa tietoturvahaavoittuvuudelle. Verkkosovelluksen turvallisuusasetuksilla tarkoitetaan asetuksia, jotka vaikuttavat sovelluksen tietoturvaan. Niitä voivat olla esimerkiksi muokkausoikeus käyttäjän käyttäjänimeen sovelluksen käyttöliittymässä tai sovelluksen käyttämän palvelimen kansiorakenteen luku- ja kirjoitusoikeudet. Oikeanlaisilla turvallisuusasetuksilla pystytään estämään tai hankaloittamaan merkittävästi kaikkien tässä luvussa esiteltyjen haavoittuvuuksien hyödyntämistä (Shema, 2010).

Turvallisuusasetukset voivat olla väärin millä tahansa verkkosovelluksen sovelluspinon tasolla palvelimen käyttöjärjestelmästä verkkosovelluksen käyttöliittymäkoodiin. Turvallisuusasetukset ovat väärin asetetut kun (OWASP, 2013):

- Verkkosovelluksen arkkitehtuurissa käytetään vanhentuneita ohjelmistoja. Vanhentuneita ohjelmistoja voivat olla esimerkiksi palvelimen käyttöjärjestelmä, HTTP- ja sovelluspalvelin, tietokantaohjelmisto ja kaikki käytetyt ohjelmistokirjastot
- Käytössä tai asennettuna on ylimääräisiä toimintoja kuten portteja, palveluja, sivuja, käyttäjätilejä tai käyttöoikeuksia
- Oletustilit ovat käytössä ja salasanoja ei ole vaihdettu

- Virheilmoitukset paljastavat pinovedoksen tai muuta liian informatiivista sisältöä käyttäjälle
- Kehitykseen käytettävän ohjelmistokehyksen (Esim. ASP.NET, Spring, WordPress) tai kirjastojen tietoturva-asetukset on asetettu väärin.

Turvallisuusasetusten ylläpito vaatii tiivistä yhteistyötä kehittäjien ja järjestelmän ylläpitäjien välillä. Turvallisuusasetusten ylläpitämiseksi on suositeltavaa rakentaa erilliset prosessit ohjelmiston asennuksia ja käytettävien komponenttien päivittämistä varten. Myös vahva ohjelmistoarkkitehtuuri, joka mahdollistaa komponenttien tehokkaan eriytymisen sekä säännölliset testaukset ja auditoinnit, edesauttaa turvallisuusasetusten oikeaa asettamista. (OWASP, 2013)

3.6 Arkaluontoisen datan paljastuminen

Arkaluontoisen datan paljastumisella tarkoitetaan tilannetta, jossa käyttäjien tietoja, esimerkiksi oikeita nimiä ja salasanoja, paljastuu henkilöille, joilla ei ole oikeutta dataan. Arkaluontoisen datan paljastuminen on yleensä seurausta heikosta sovellusarkkitehtuurista ja altistaa sovelluksen laajalle skaalalle erilaisia hyökkäyksiä. (OWASP, 2013.) Datan paljastumisella on yleensä myös kiusallisia seurauksia koko verkkosovelluksen takana olevalle yritykselle tai organisaatiolle, sillä tapaukset, joissa järjestelmän käyttäjien tietoja päätyy ulkopuolisten käsiin, saavat usein paljon julkisuutta.

Arkaluontoisia tietoja pääsee paljastumaan, kun (OWASP, 2013):

- Tietoja tai varmuuskopioita säilytetään selkokielisenä
- Tietoja lähetetään sisäisesti tai ulkoisesti selkokielisenä
- Käytetään vanhoja tai heikkoja salausalgoritmeja
- Generoidaan heikkoja avaimia tai avainten hallinta ja vaihtaminen on vajavaista
- Turvallisuusasetukset tai otsikot ovat puutteelliset, kun lähetetään tai vastaanotetaan arkaluontoista dataa.

Vähintä mitä datan suojelemiseksi voi tehdä on käyttää salausta tietoja välitettäessä ja tallennettaessa. Tehokkain tapa suojautua arkaluontoisen datan paljastumiselta on

säilyttää dataa mahdollisimman vähän ja tuhota se niin pian kuin mahdollista. Näin ollen tietovuodon tapahtuessa paljastuu pienin mahdollinen määrä arkaluontoista dataa. (OWASP, 2013)

3.7 Puuttuva toimintotason pääsynvalvonta

Toimintotason pääsynvalvonnalla tarkoitetaan palvelinpään mekanismeja, joilla tarkastetaan, onko käyttäjällä A oikeus suorittaa operaatio B. Puuttuva toimintotason pääsynvalvonta antaa käyttäjälle mahdollisuuden suorittaa järjestelmässä toimintoja, joihin hänellä ei tulisi olla oikeuksia. Toimintotason pääsynvalvonta voi puuttua järjestelmästä kokonaan tai se voi olla puutteellinen joidenkin toimintojen osalta. Jos pääsynvalvonta puuttuu järjestelmästä kokonaan, tarkoittaa se sitä, että kuka tahansa verkon käyttäjä voi suorittaa toimintoja sovelluksessa. Jos pääsynvalvonta puuttuu osittain voi esimerkiksi järjestelmän peruskäyttäjällä olla oikeudet suorittaa järjestelmänvalvojan toimintoja. (OWASP, 2013)

Puuttuva toimintotason pääsynvalvonta paljastuu helposti testaamalla tai koodin katselmoinnin avulla. Pääsynvalvonta on todennäköisesti puutteellista, jos a) käyttöliittymässä näkyy luvattomia toimintoja, esimerkiksi ylläpitosivu on näkyvissä kaikille käyttäjille, b) palvelimelta puuttuu oikeuksien tarkistus, esimerkiksi ylläpitosivu ei ole näkyvissä, mutta sen metodeihin tehdyt kutsut suoritetaan silti palvelimella tai c) palvelimen tarkastukset perustuvat pelkästään käyttäjän lähettämiin tietoihin eli käyttäjä saa esimerkiksi järjestelmänvalvojan oikeudet arvaamalla ylläpitosivun osoitteen. (OWASP, 2013)

```
http://verkkosovellus.fi/hallinta
http://verkkosovellus.fi/admin-hallinta
```

Esimerkki 8. Tilinhallinnan ja järjestelmänvalvojan hallintasivun osoitteet verkkosovellus.fi järjestelmässä

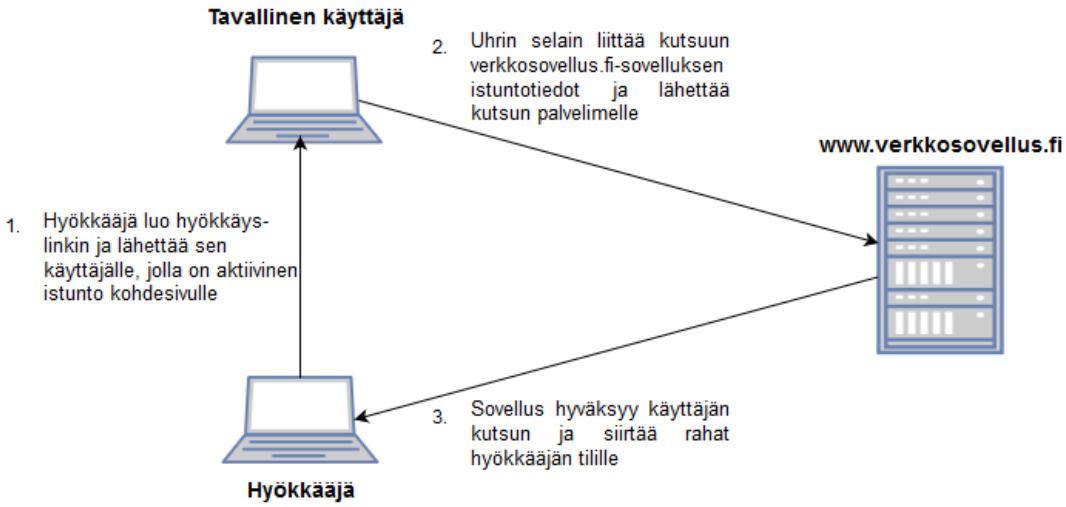
Esimerkissä 8 on kuvitteellisen verkkosovellus.fi järjestelmän hallintasivun ja järjestelmänvalvojasivun osoitteet. Jos palvelin ei tarkasta onko sivulle saapuvalla käyttäjällä todella järjestelmänvalvojan oikeudet, voi kuka tahansa saada käyttöönsä järjes-

telmänvalvojan oikeudet. Tieto käyttäjän oikeuksista tulisi siis tallentaa esimerkiksi tietokantaan siten, että käyttäjä ei pysty itse muokkaamaan tallennettuja oikeuksiaan. (OWASP, 2013)

Pääsynvalvonnan parantamiseksi se kannattaa toteuttaa omana moduulinaan, jota käytetään kaikkialla sovelluksessa. Käyttöoikeudet tulisi tallentaa sovellukseen toimintokohtaisesti, esimerkiksi `lisää_tili`, `muokkaa_tiliä`, `poista_tili` ja oikeuksien lisääminen ja poistaminen käyttäjäkohtaisesti tulisi olla mahdollista. Monet ohjelmistokehykset sisältävät pääsynvalvontaan liittyvää toiminnallisuutta. Ohjelmistokehyksiä käytettäessä on kuitenkin syytä muistaa, että pelkkä kehyksen käyttöönotto ei tee sovelluksesta turvallista, vaan verkkosovelluksen turvaamiseksi sovelluskehyksiä tulee käyttää niiden kehittäjän tarkoittamalla tavalla. (OWASP, 2013)

3.8 Istunnolla ratsastaminen

Istunnolla ratsastamis- eli CSRF-haavoittuvuus mahdollistaa verkkosovelluksen toimintojen suorittamisen toisen sovelluksen käyttäjän nimissä. Hyökkäys tapahtuu, kun käyttäjän selain saadaan tekemään kutsu kohdesovellukseen samalla, kun käyttäjällä on aktiivinen istunto kohdesovellukseen.



Kuva 8. CSRF-hyökkäyksen suorittaminen.

CSRF-hyökkäys voidaan suorittaa jakamalla hyökkäyslinkkiä sähköpostitse tai muiden sivustojen välityksellä. Esimerkissä 9 esitetään img-tagi, jonka lataaminen toteuttaa CSRF-hyökkäyksen verkkosovellus.fi/siirraRahaa-metodiin. Hyökkäys tapah-

tuu, kun käyttäjän selain lähtee lataamaan tagin määrittämää kuvaa src-attribuutin osoittamasta osoitteesta. Osoitteesta ei löydykään valokuvaa vaan selaimen tekemä GET-kutsu suorittaa verkkosovellus.fi/siirraRahaa-metodin ja siirtää käyttäjän tililtä 1500 yksikköä rahaa tilille numero FI4673243243496728.

```
<img>src=”www.verkkosovellus.fi/siirraRahaa?summa=150
0&kohdeTili=FI4673243243496728”</img>
```

Esimerkki 9. Rahan siirtämisen toteuttava toiminto img-tagin lähteenä.

CSRF-hyökkäyksen vaikutukset voivat olla laajat: hyökkäyksen avulla voidaan suorittaa mikä tahansa toiminto, jonka käyttäjän käyttöoikeudet sallivat. Tällaisia toimintoja voivat olla esimerkiksi rahan siirtäminen verkkopankissa, käyttäjän oman tai muiden käyttäjätilien poistaminen tai käyttäjätietojen varastaminen. (OWASP, 2013)

```
<input type="hidden" name="csrf" value="Hfg4-74BN-wwi2-
394a">
```

Esimerkki 10. CSRF-haavoittuvuuden estämiseen tarkoitettu piilotettu input-kenttä

CSRF-hyökkäykset voidaan estää sisällyttämällä sovelluksen tilaa muuttaviin kutsuihin erillinen satunnainen ja vaikeasti arvattava CSRF-tunniste. Tunniste voidaan sisällyttää kutsun otsikoihin tai esimerkin 10 mukaisesti piilotettuna kenttänä kutsun runkoon, jolloin sen huomaaminen ja arvaaminen hankaloituvat eikä se häiritse käyttäjän toimintaa. CSRF-hyökkäyksiä voidaan myös estää vaatimalla käyttäjää tunnistautumaan uudelleen tai vahvistamaan kutsun suorittamisen esimerkiksi salasanan tai CAPTCHA;n eli kuvavarmenteen avulla. (OWASP, 2018b)

3.9 Haavoittuvuuksia sisältävien ulkoisten komponenttien käyttäminen

Kaikki internetsivut käyttävät ulkoisia komponentteja, kuten esimerkiksi palvelinta, tietokantaa ja erilaisia ohjelmistokehyksiä. Ulkoisia komponentteja hyödyntävä hyökkääjä tutkii verkkosivustolla käytössä olevia komponentteja joko koneellisesti tai manuaalisesti ja selvittää löytyykö komponenteista tunnettuja haavoittuvuuksia. Haavoittuvuuksia sisältävien komponenttien käyttäminen verkkosovelluksessa jättää sovelluksen alttiiksi käytännössä kaikille hyökkäystyypeille komponentin haavoittuvuudesta riippuen. (OWASP, 2013)

Haavoittuvuuksia sisältävien komponenttien käyttämiseltä on teoriassa helppo välttyä valitsemalla luotettavia komponentteja ja pitämällä ne päivitettyinä, mutta käytännössä tämä ei aina ole mahdollista. Monessa tapauksessa uuden uhan löydyttyä komponentin valmistajat eivät tarkenna mitä versiota uhka koskee. Erillisiä tietoturvapäivityksiä ei useinkaan julkaista vaan aukot korjataan seuraavaan versioon. Kaikkien komponenttien pitäminen ajan tasalla olisi turvallisin vaihtoehto, mutta komponenttien keskinäisien riippuvuussuhteiden vuoksi uusimpien versioiden asentaminen ei ole aina välttämättä mahdollista. (OWASP, 2013)

Haavoittuneiden komponenttien käytön estämiseksi kehittäjien ja ylläpitäjien on päivitettävä listaa, jolta käyvät ilmi kaikki sovelluksessa käytettävät komponentit ja niiden versiot. Listaa on verrattava säännöllisesti erilaisiin kanaviin, joilla välitetään tietoa löydetyistä haavoittuvuuksista. Tällaisia kanavia ovat komponenttien kehittäjien päivityksiä koskevat postituslistat, yleiset tietoturvaa koskevat postituslistat ja julkiset tietokannat löydetyistä haavoittuvuuksista. (OWASP, 2013)

3.10 Vahvistamattomat uudelleenohjaukset

Vahvistamattomien uudelleenohjauksien avulla hyökkääjät saavat ohjattua käyttäjät haitallisille sivuille. Vahvistamatonta uudelleenohjausta voidaan hyödyntää silloin, kun verkkosovelluksessa on toiminto, jonka avulla käyttäjiä ohjataan sovelluksen eri sivuille. Jos ohjaukseen käytettävää parametria ei tarkasteta, hyökkääjä voi lähettää käyttäjille aidon näköisiä linkkejä näiden mielestä luotettavalle sivustolle, vaikka linkit tosiasiassa uudelleenohjaavat käyttäjän hyökkääjän haluamalle sivulle. (OWASP, 2013)

Vahvistamattomien uudelleenohjauksien välttämiseksi uudelleenohjauksien käyttämistä verkkosovelluksien arkkitehtuurissa kannattaa pyrkiä välttämään kokonaan. Jos uudelleenohjauksia kuitenkin käytetään, tulisi toimintalogiikasta poistaa mahdollisuus käyttäjän antamille parametreille. Jos parametreja ei voida kokonaan poistaa, tulee ne tarkastaa sekä varmistaa käyttäjän käyttöoikeudet kyseiselle sivulle ja kyseiseen dataan. Selkokielisten parametrien sijaan voidaan myös käyttää luotuja arvoja, jotka yhdistetään palvelimella oikeaan osoitteeseen. (OWASP, 2013)

4 Verkkosovelluksen suojaaminen

Kolmannessa luvussa esitellyt verkkosovellusten kymmenen yleisintä tietoturvauhkaa ovat valitettavasti vain jäävuoren huippu verkkosovellukseen kohdistuvista uhista. Pelkkä yleisten haavoittuvuuksien listaaminen ja verkkosovelluksen suojaaminen niitä vastaan eivät riitä takaamaan verkkosovelluksen turvallisuutta. Täysin turvallista verkkosovellusta tuskin on edes olemassa, ja vaikka tällä hetkellä sovellus olisikin turvallinen, tilanne muuttuu, sillä uusia hyökkäysmenetelmiä kehitellään koko ajan (Barnett & Grossman, 2013).

Tässä luvussa esitellään keinoja verkkosovelluksen *tietoturvakerroksen* toteuttamiseen. Tietoturvakerros muodostuu kolmesta tasosta, jotka ovat *hyökkäyksien havaitseminen, hyökkäyksiin reagoiminen* ja *löydettyjen haavoittuvuuksien paikkaaminen.* Tietoturvakerros voidaan toteuttaa hyödyntämällä yhtä tai useampaa tekniikkaa.

4.1 Hyökkäyksien havaitseminen

Jotta verkkosovellukselle voidaan luoda suojauksia uhkia vastaan tai paikata olemassa olevia haavoittuvuuksia, haavoittuvuudet ja heikot kohdat tulee voida tunnistaa. Tämä tarkoittaa sitä, että verkkosovelluksen tietoturvasta vastaavien henkilöiden tulee kerätä ja pitää yllä erilaisia tilastoja verkkosovelluksen internetliikenteestä. Tilastojen pohjalta voidaan muodostaa malli verkkosovelluksen normaalista verkkoliikenteestä ja tunnistaa epänormaali liikenne sekä mahdolliset hyökkäysyritykset. Myös havaituista hyökkäyksistä ja niihin kohdistuneista vastatoimista on hyvä pitää tilastoa, jotta saadaan ylläpidettyä realistinen kuva verkkosovelluksen turvallisuustilanteesta. (Barnett & Grossman, 2013)

Realistisen kuvan muodostamiseksi, hyökkäyksien tunnistamiseksi ja hyökkäystilastojen keräämiseksi tulee verkkosovellukseen ja sen alustana toimivaan palvelimeen kohdistuvaa verkkoliikennettä seurata ja tallentaa lokeihin. Ilman historiatietoja on hankalaa tai lähes mahdotonta selvittää kuinka hyökkäys on toteutettu ja mitä tietoja on mahdollisesti varastettu. Palvelimelle kohdistuvasta liikenteestä tulisi tallentaa ainakin pyyntömenetelmä (*request method*), pyynnön parametrit, parametrien nimet ja pituudet sekä parametrien tyypit. (Barnett & Grossman, 2013)

Historiatietoja voidaan käyttää apuna myös koostettaessa tilastoja tietoturvatoimien toimivuudesta. Tietoturvatoimien toimivuuden seuraamiseksi ja takaamiseksi Barnettin ja Grossmanin (2013) mukaan tulee pitää yllä tilastoja ainakin päivittäisten transaktioiden määrästä, havaituista hyökkäyksistä, havaitsematta jääneistä hyökkäyksistä, virheellisesti estetystä liikenteestä ja hyökkäyksien havaitsemisen epäonnistumisprosentista.

4.2 Pohjatietojen analysointi ja tietoturvauhkien tunnistaminen

Kun tietoturvakerroksen ensimmäinen taso eli verkkoliikenteen seuranta ja tallentaminen lokeihin on saatu toteutettua, on aika siirtyä kerättyjen tietojen analysointiin. Tietojen analysointiin käytetään *aktiivista* ja *passiivista* tietoturvauhkien havaitsemista. Aktiivinen ja passiivinen havaitseminen ovat eri metodeja uhkien tunnistamiseen ja molempia lähestymistapoja toteuttavia työkaluja on useita. Tässä luvussa esitellään aktiivinen ja passiivinen uhkien tunnistaminen yleisellä tasolla.

Aktiivista uhkien tunnistamista suoritetaan *dynaamisella sovelluksen tietoturvatestauksella (Dynamic Application Security Testing*, DAST) ja sitä voidaan suorittaa jo verkkosovelluksen kehitysvaiheessa. Kuten tavallisen ohjelmiston testaukseen myös verkkosovelluksen tietoturvan dynaamiseen testaamiseen käytetään ulkoista ohjelmistoa, joka suorittaa testejä ohjelmistolle. Koska verkkosovellusten kirjo on hyvin laaja, myös turvallisuustestaukseen käytettyjen ohjelmistojen määrä on suuri ja eri ohjelmat soveltuvat erilaisiin käyttötarkoituksiin. Ohjeita sopivan sovelluksen valitsemiseen löytyy esimerkiksi *Web Application Security Consortium*:in *Web Application Security Scanner Evaluation Criteria* dokumentaatiosta. (WASC, 2009; Barnett & Grossman, 2013)

Kaikkia tietoturvauhkia ei todennäköisesti tulla tunnistamaan ennen ohjelmiston julkaisua. Tuotantoympäristössä verkkosovelluksen tietoturvauhkien tunnistamista voidaan jatkaa passiivisella menetelmällä analysoimalla kerättäviä lokitietoja sekä verkkosovelluspalomuurin (Web Application Firewall) avulla. *Passiivisella haavoittuvuuksien tunnistamisella* (*Passive Vulnerability Identification,* PVI) tarkoitetaan menetelmää, jossa seurataan tuotantoympäristössä olevan verkkosovelluksen sisään tulevaa ja ulos lähtevää verkkoliikennettä ja etsitään sieltä viitteitä haavoittuvuuksis-

ta vertaamalla dataa tunnettujen haavoittuvuuksien tai hyökkäyksien "sormenjälkiin". Tunnettuja haavoittuvuuksia kerääviä tietokantoja on useita, esimerkiksi Yhdysvaltain valtion ylläpitämä *National Vulnerability Database* (NIST, 2016). Passiivisen uhkien tunnistamisen avulla ei nimensä mukaisesti ole tarkoitus pysäyttää käynnissä olevaa verkkohyökkäystä, vaan tunnistaa järjestelmän heikkoja kohtia jo ennen kuin hyökkäys pääsee tapahtumaan ja ilmoittaa hälyttävästä liikenteestä verkkosovelluksen ylläpitäjille. Hälyttävä liikenne tunnistetaan pisteyttämällä käyttäjien istuntoja ja IP-osoitteita epänormaalien kutsujen ja toimintojen perusteella. (Barnett & Grossman, 2013)

Parhaaseen tulokseen verkkosovelluksen tietoturvauhkien tunnistamisessa päästään, kun molempia tekniikoita hyödynnetään yhdessä. Samoin kuin ohjelmiston yleinen testaaminen myös aktiivinen tietoturvan testaaminen vähentää tuotantoon pääsevien virheiden määrää. Kaikkia virheitä ei kuitenkaan todennäköisesti tulla havaitsemaan ennen tuotantoon siirtymistä, joten passiivisen uhkien tunnistamisen avulla voidaan havaita mahdollisia haavoittuvuuksia ja saada tieto hälyttävästä liikenteestä.

4.3 Verkkosovelluksen ansoittaminen

Passiivinen ja aktiivinen uhkien etsiminen ja tunnistaminen on elintärkeää verkkosovelluksen tietoturvan kannalta, mutta kaiken verkkosovelluksen liikenteen tallentaminen lokeihin, lokien tarkkaileminen ja verkkosovelluksen aktiivinen testaaminen ovat paljon prosessoritehoa ja levytilaa vaativia toimenpiteitä. Lisäksi edellä mainitut menetelmät johtavat yleensä myös vääriin hälytyksiin ja vastatoimiin, jotka voivat häiritä järjestelmän varsinaisten käyttäjien toimia. (Barnett & Grossman, 2013)

Väärien hälytysten ehkäisemiseksi ja oikeiden hyökkäysten huomaamisen nopeuttamiseksi järjestelmään voidaan ohjelmoida ansoja ja syöttejä hakkereiden varalle. *Ansoilla* tarkoitetaan ulospäin näkyvää toiminnallisuutta, jota järjestelmän käyttäjät eivät oikeasti käytä ja jolla ei ole järjestelmän tilaa muuttavia palvelintoteutuksia. Koska ansatoiminnoille ei ole varsinaista käyttöliittymää ja verkkosovelluksen oikeat käyttäjät eivät koskaan käytä ansoiksi rakennettua toiminnallisuutta, verkkosovelluksen ylläpitäjät saavat heti tiedon mahdollisesta hyökkäyksestä, jos ansatoimintoja ak-

tivoidaan. Kun tieto hyökkäyksestä saadaan mahdollisimman aikaisessa vaiheessa jää ylläpitäjille enemmän aikaa reagoida hyökkäykseen ennen kuin mitään todellista vahinkoa pääsee syntymään. (Barnett & Grossman, 2013)

Ansatoiminnallisuutta suunniteltaessa on hyvä tarkastaa, että ansat ovat tarpeeksi houkuttelevia ja samalla järjestelmään sopivia. Hyviä ansoja ovat esimerkiksi toimivan koodin sekaan jätetyt kommentoidut metodikutsut ja "vanhat" versiot tiedostoista tai kirjastoista. Myös tunnistautumista vaativat metodikutsut ovat hyviä ansoja, sillä hyökkääjä joutuu käyttämään resursseja tunnistautumisen yrittämiseen, ja käytetyistä tunnuksista ja salasanoista selviää, onko hyökkääjä onnistunut kalastelemaan järjestelmän käyttäjien tietoja. (Barnett & Grossman, 2013)

4.4 Hyökkäyksiin reagoiminen

Verkkosovelluksen tietoturvaa voidaan testata erittäin kattavasti aktiivisin ja passiivisin menetelmin. Testauksesta huolimatta on kuitenkin todennäköistä, että joku löytää vielä haavoittuvuuksia, joiden avulla voidaan esimerkiksi varastaa tietoja verkkosovelluksesta tai manipuloida sen toimintaa. Tämän vuoksi tarvitaan myös keinoja reagoida käynnissä olevaan hyökkäykseen esimerkiksi suodattamalla verkkosovellukseen saapuvaa ja siitä lähtevää liikennettä tai estämällä kokonaan liikenne tiettyihin verkkosovelluksen toimintoihin. Koska tietoturvauhkien kirjo on erittäin laaja, samoin kuin uhkien tunnistaminen myös niiden torjuminen vaatii useiden työkalujen yhtäaikaista käyttämistä.

Oikean metodin valitseminen verkkohyökkäyksen tai tietoturvauhan torjumiseksi riippuu monesta tekijästä. Hyökkäys voi olla esimerkiksi palvelunestohyökkäys, määrätietoinen yritys murtaa juuri tietty verkkosovellus tai viruksilla saastutettujen koneiden avulla suoritettavaa sattumanvaraista kokeilua aukkojen löytämiseksi. Tässä luvussa esitellään Heiderichin et al. (2011) kuvaamia esimerkkejä siitä, kuinka hyökkäyksiin voidaan reagoida. Heiderich et al. käyttämät esimerkit on toteutettu avoimen lähdekoodin *ModSecurity*-verkkosovelluspalomuurilla ja OWASP:in sitä varten kehittämällä Core Rule Set -säännöstöllä, mutta sama toiminnallisuus voidaan toteuttaa myös muilla työkaluilla.

Ensimmäisen toimenpiteen havainnon jälkeen tulisi olla hyökkäystyypistä riippumatta hyökkäyksestä hälyttäminen asianomaisille tahoille. Hyökkäyshälytykset voidaan lisätä osaksi passiivista uhkien tunnistamista, jolloin järjestelmä seuraa ja pisteyttää epänormaaleja verkkosovellukseen suuntautuvia kutsuja ja lähettää hälytyksen, jos pisteet nousevat tietyn istunnon tai IP-osoitteen osalta liian korkeiksi. (Heiderich et al., 2011)

Kun hälytys epäilyttävästä toiminnasta on saatu, on verkkosovelluksen ylläpitäjillä useita eri keinoja reagoida tilanteeseen. Kevyin vastaus on ohjata käyttäjä verkkosovelluksen virhesivulle. Virhesivulle ohjausta käytettäessä tulisi virhesivun ulkoasun vastata sovelluksen muita sivuja ja virhesivuja, jolloin hyökkääjä ei heti havaitse häneen kohdistuvia vastatoimia. (Heiderich et al., 2011)

Jos pelkkä virhesivuille ohjaaminen ei auta, vaan hyökkääjä jatkaa operaatiotaan, voidaan hyökkäystä yrittää hidastaa ja näin hankkia lisää aikaa verkkosovelluksen ylläpitäjille. Heiderichin et al. (2011) mukaan hyökkäystä voidaan yrittää hidastaa ohjaamalla hyökkääjän verkkoliikenne houkutinverkkosovellukseen, joka ei sisällä oikeaa dataa, hidastaa hyökkäyksen kohteena olevien verkkosovellusmetodien toimintaa ohjelmistollisesti tai antaa hyökkääjän ymmärtää hyökkäyksen onnistuneen.

Koska suurin osa hyökkäyksistä toteutetaan tietokoneohjelmien avulla, tapahtuvat hyökkäykset hyvin nopeasti ja aikaa reagoida hyökkäykseen on vähän. Verkkosovelluksen ylläpitäjät voivat kuitenkin yrittää hyökkäyksen hidastamista ohjelmistollisesti. Ohjelmistollinen hidastaminen tapahtuu pysäyttämällä käskyjä suorittava säie ennalta määritetyksi ajaksi, jolloin käskyjen suorittaminen hidastuu ja hyökkäyksen kesto kasvaa sekunneista minuuteiksi. Toinen tapa hidastaa hyökkäystä on uskotella hyökkääjälle hyökkäyksen onnistuneen. Tämä voidaan toteuttaa palauttamalla hyökkäyskäskyihin onnistumiseen viittaavia 200-alkuisia HTTP-statuksia, valheellisia virheilmoituksia, jotka antavat ymmärtää osan käskystä olevan kelvollinen sekä keksittyä dataa, joka vaikuttaa aidolta verkkosovelluksen datalta. (Heiderich et al., 2011)

Sinnikkäiden hyökkäysten yhteydessä pelkkä hyökkäyksen hidastaminen ei riitä ja palvelunestohyökkäyksen yhteydessä hidastaminen vain pahentaa jo ennestään akuuttia verkkosovelluksen resurssipulaa. Tällöin viimeiseksi oljenkorreksi jää käyt-

täjän istunnon lopettaminen, käyttäjätilin väliaikainen käytöstä poistaminen tai hyökkääjän verkko-osoitteen estäminen ja yhteyden katkaiseminen. (Heiderich et al., 2011)

Käyttäjän istunnon päättäminen ja käyttäjätilin väliaikainen sulkeminen ovat melko lieviä ja turvallisia keinoja, sillä ne koskevat vain tiettyjä ennalta määrättyjä käyttäjiä. Verkko-osoitteiden estäminen ja liikenteen katkaiseminen ovat kuitenkin riskialttiimpia operaatioita, sillä useita käyttäjiä voi yhdistää sovellukseen saman välityspalvelimen kautta, jolloin heillä kaikilla on sama osoite. Jos tällainen osoite estetään, on olemassa mahdollisuus, että samalla estetään suuri määrä asiallisia käyttäjiä. Verkko-osoitteiden estämisessä on myös ongelmana se, että hyökkääjä voi nopeasti ohjata verkkoliikenteensä toisen välityspalvelimen kautta, jolloin hyökkäys pääsee jatkumaan välittömästi. Äärimmäisessä tapauksessa vakavan verkkohyökkäyksen alla voidaankin joutua estämään verkkoliikenne kokonaan tietyistä maista, kun on havaittu hyökkäyksen tulevan tiettyjen maiden verkko-osoitteista. (Heiderich et al., 2011)

4.5 Verkkosovelluspalomuuri

Verkkosovelluspalomuuri (Web Application Firewall, WAF) on yleiskäyttöinen työkalu, jota voidaan käyttää aktiiviseen ja passiiviseen uhkien tunnistamiseen ja torjumiseen. Perinteisen palomuurin tapaan verkkosovelluspalomuurin avulla on tarkoitus suodattaa verkkosovellukseen saapuvaa ja siitä lähtevää liikennettä sekä tarvittaessa estää verkkoliikenne tietyistä osoitteista tai tiettyihin verkkosovelluksen osiin. Perinteisestä palomuurista se eroaa siten, että se monitoroi vain HTTP- ja HTTPS-liikennettä. Ihannetilanteessa palomuuri toimii sallivassa tilassa eli palomuurin sääntöihin on määritetty sallitut osoitteet, pyyntömetodit ja parametrit, ja vain määrityksiä vastaava HTTP-liikenne päästetään palomuurin läpi. Käytännössä verkkosovelluspalomuurin käyttämiseen kuitenkin liittyy ongelmia: palomuurin määrittäminen sallivaan tilaan on erittäin työlästä ja haastavaa. Parhaatkin määritykset voidaan aina kiertää ja myös itse palomuuri on altis hyökkäyksille, joten hyökkäyksille alttiin koodin osuus verkkosovelluksessa kasvaa. (Heiderich et al., 2011)

Koska verkkosovelluspalomuureja on hankala määritellä täysin sallivaan tilaan, käytetään niitä yleensä estävässä tilassa. Estävässä tilassa palomuuri etsii verkkoliiken-

teen seasta tunnettuja hyökkäyskoodeja (Heiderich et al., 2011). Heiderichin et al. (2011) mukaan estävän tilan eli mustan listan käyttäminen hyökkäysten estämiseen on kuitenkin tehotonta, sillä pienillä muutoksilla saadaan hyökkäyskoodia muutettua siten, että palomuuri ei enää tunnista koodia haitalliseksi. Heiderich et al. (2011) esittelevät teoksessaan nimeltä mainitsemattoman verkkosovelluspalomuurin ja listan esimerkkihyökkäyksiä ja muutoksia, joiden avulla palomuuri ei enää tunnista hyökkäystä. Esimerkiksi muuttamalla koodin `' or 1=1 --` muotoon `' or 2=2 --` palomuurin musta lista ei enää tunnistanut hyökkäystä.

Koska verkkosovelluspalomuurin oikeaoppinen määrittäminen on vaikeaa ja parhaatkin määritykset voidaan ohittaa, ei palomuuria suositella ainoaksi ja pitkäkestoiseksi vaihtoehdoksi tunnettujen aukkojen suojaamiseen. Palomuurin avulla voidaan kuitenkin paikata *virtuaalisesti* (virtual patch) löydettyjä haavoittuvuuksia siksi aikaa, kunnes haavoittuvuus voidaan korjata (Barnett & Grossman 2013). Useista haasteistaan huolimatta verkkosovelluspalomuurit ovat hyödyllisiä verkkosovelluksen suojaamisessa ja palomuurien käyttö tuotanto-ohjelmistojen suojaamiseen näyttää kasvavan. Heiderichin et al. (2011) mukaan yksi syy verkkosovelluspalomuurien yleisyyteen on se, että *Payment Card Industry* (PCI) *Data Security Standardin* (DSS) mukaan palomuuri on toinen kahdesta mahdollisesta suojasta julkisessa verkossa näkyville sovelluksille, jotka käsittelevät luottokorttimaksuja. Toinen vaihtoehto ovat vuosittaiset tietoturva-auditoinnit, jotka tulevat pitkällä aikavälillä kalliimmaksi kuin verkkosovelluspalomuurin ylläpitäminen.

5 Tutkimusasetelma

Tässä luvussa kuvataan tutkielman käytännön osuuden tutkimusasetelman tavoitteet ja toteutus. Käytännön osuudessa pyritään lokituksen ja monitoroinnin avulla havaitsemaan kahteen verkkosovellukseen kohdistuva tietoturvatestaus. Tietoturvatestaus simuloi testauksessa verkkosovelluksiin kohdistuvaa verkkohyökkäystä. Testaus suoritetaan kahden lähiverkossa olevan tietokoneen välillä. Koneista toinen toimii verkkosovellusten palvelimena ja toinen hyökkäyksiä suorittavana asiakkaana. Tietoturvatestauksen aikana verkkosovelluksia monitoroidaan ja niihin kohdistuvia hyökkäyksiä pyritään tunnistamaan.

Käytännön osuuden tavoitteena on selvittää, onko palvelimeen kohdistuvan HTTP- ja HTTPS-liikenteen monitorointi ja lokitus toimiva keino verkkosovelluksiin kohdistuvien automatisoitujen hyökkäysten tunnistamisessa. Lisäksi pyritään selvittämään, pystytäänkö näiden valvontatoimien avulla lyhentämään hyökkäysten havaitsemiseen kulunutta aikaa. Ponemon-instituutin (2017) mukaan verkkosovelluksiin kohdistuvien hyökkäyksien huomaaminen kesti vuonna 2016 keskimäärin 191 vuorokautta. Tässä tutkimuksessa kohdesovelluksina käytetään Itä-Suomen yliopiston *Mopsi*-sovellusta ja tietoturvatestaukseen tarkoitettua *Damn Vulnerable Web Application*:ia (DVWA).

Testauksen aikainen monitorointi toteutetaan asentamalla verkkosovelluksia tarjoavalle palvelimelle avoimen lähdekoodin ModSecurity-verkkosovelluspalomuuri, joka monitoroi ja lokittaa verkkosovellukseen kohdistuvaa liikennettä. Monitoroinnin ja lokituksen tehokkuutta testataan hyökkäämällä palvelimelle asennettuja verkkosovelluksia vastaan automaattisilla hyökkäystyökaluilla.

Tämä luku jakautuu neljään alalukuun. Ensimmäisessä alaluvussa esitellään verkkosovellusten monitorointiin käytettävä ohjelmisto ja sen asetukset. Sen jälkeen esitellään palvelimelle asennetut verkkosovellukset sekä nimetään ja esitellään hyökkäyksiin käytettävät työkalut ja yksittäiset hyökkäykset. Viimeisessä eli neljännessä alaluvussa käydään lopuksi läpi testauksen kulku ja tavoitteet.

5.1 Monitorointityökalut

Tutkielman käytännön osuus toteutetaan monitoroimalla ja lokittamalla verkkosovelluksiin kohdistuva liikenne avoimen lähdekoodin ModSecurity-verkkosovelluspalomuurilla. Testipalvelimelle on asennettu ModSecurityn versio 2.9.2 ja siihen on lisätty OWASP:in ylläpitämän Core Rule Set (CRS) -säännöstön versio 3.0.2 hyökkäysten tunnistamista varten. ModSecurity on valittu tutkimukseen, koska OWASP käyttää sitä esimerkkinä monitorointityökaluista 2017 vuoden Top 10 -listauksessaan (OWASP, 2017). ModSecurityn ja CRS:n asennus ja asetukset noudattelevat Christian Folinin (2018) aiheesta tekemää ohjeistusta.

ModSecurity:n avulla on toteutettu tietoturvakerroksen ensimmäinen ja toinen taso. Ensimmäinen taso eli hyökkäyksien havaitseminen on toteutettu asettamalla ModSecurity toimimaan poikkeamapisteytystilassa (*anomaly score*), jossa se ei rajoita verkkosovellukseen kohdistuvaa verkkoliikennettä, mutta lokittaa kutsut ja tunnistamansa poikkeamat. Poikkeamilla tarkoitetaan kutsuissa olevia hyökkäyksiin viittaavia tietoja ja piirteitä, kuten esimerkiksi SQL-injektioon käytettäviä koodinpätkiä. Poikkeamapisteytystilassa ModSecurity jakaa poikkeamat neljään kategoriaan niiden vakavuuden perusteella. Kategoriat on nimetty *kriittiseksi, virheeksi, varoitukseksi* ja *huomautukseksi*. Tätä tutkielmaa varten kategoriat on pisteytetty seuraavalla tavalla: kriittinen 5, virhe 4, varoitus 3 ja huomautus 2.

Tietoturvakerroksen toinen taso, eli hyökkäyksiin reagoiminen, on toteutettu lisäämällä ModSecurityyn oma sääntö, joka laukaisee hälytyksen, kun kutsujen kumulatiivinen poikkeamapisteytys saavuttaa tai ylittää viiden pisteen rajan. Käytännössä hälytys tarkoittaa sitä, että ModSecurity suorittaa *bash-komentosarjan* (bash script), kun hälytysraja ylittyy. Tutkielmaa varten komentosarja kirjoittaa tiedostoon hälytyksen kellonajan. Tuotantoympäristössä hyökkäyksiin voidaan reagoida järeämmin, kuin pelkästään lokittamalla havaitut hyökkäykset. Komentosarja voisi esimerkiksi asettaa ModSecurity:n toimimaan estävässä tilassa, jolloin hyökkäyksiksi havaitut kutsut estetään tai komentosarjan avulla voitaisiin luoda palomuurisääntö, joka estää kaikki yhteydet hyökkäyksiä tekevästä IP-osoitteesta.

5.2 Verkkosovellukset

Testauksessa käytetyn palvelinkoneen käyttöjärjestelmänä toimii Ubuntu 16.06. Verkkosovellusten tarjoamista varten palvelimelle on asennettu Apache httpd -palvelinohjelmiston versio 2.4.29 ja sovellusten relaatiotietokantajärjestelmänä palvelimella toimii MariaDB:n versio 10.0.34. Palvelimelle on lisäksi asennettu kaksi verkkosovellusta, joihin kohdistettavien hyökkäysten avulla monitorointia testataan. Sovellukset ovat tietoturvatestausta varten rakennettu *Damn Vulnerable Web Application* (DVWA) ja Itä-Suomen yliopiston tietojenkäsittelytieteen laitoksen *Mopsi*-sovellus. Mahdollisimman suuren testikattavuuden takaamiseksi palvelimen juurihakemistossa on verkkosovellusten lisäksi indeksisivu, jossa on linkki molempiin verkkosovelluksiin. Indeksisivun tarkoituksena on edesauttaa hyökkäystyökaluja löytämään molemmat sovellukset palvelimelta.

Tutkielmassa käytettäviä verkkosovelluksia ei kuvata yksityiskohtaisesti, sillä käytettävillä sovelluksilla ei ole suurta merkitystä testiasetelman kannalta. Monitorointi ei vaadi, että palvelimelle olisi asennettu sovelluksia, vaan se monitoroi ja lokittaa tarvittaessa myös tyhjään palvelimeen kohdistuvaa HTTP- ja HTTPS-liikennettä. Testiasetelmassa käytettävät verkkosovellukset toimivatkin ainoastaan syötteenä ja hyökkäyspinta-alana automaattisille hyökkäystyökaluille.

DVWA on PHP:llä toteutettu, verkkoselaimella käytettäväksi tarkoitettu verkkosovellus, joka sisältää useita tietoturvahaavoittuvuuksia. Sen tarkoituksena on tarjota verkkosovellusten tietoturvasta kiinnostuneille henkilöille laillinen alusta työkalujen ja tekniikoiden kehittämiseen ja opettelemiseen. DVWA toimii tutkielman testiasetelmassa verrokkisovelluksena, jonka haavoittuvuuksien avulla voidaan varmistaa hyökkäystyökalujen toimivuus. Mikäli haavoittuvuuksia löytyy, voidaan todeta hyökkäystyökalujen toimivan.

Mopsi puolestaan on Itä-Suomen yliopiston tietojenkäsittelytieteen laitoksen kehittämä sovellus, joka hyödyntää paikkatietoihin perustuvaa informaatiota. Sovelluksella voi esimerkiksi mitata ja tallentaa käyttäjän kulkemia reittejä, lisätä valokuvia ja suosituksia erilaisista paikoista sekä hakea bussiaikatauluja. Sovellukseen voidaan li-

sätä tietoja mobiililaitteen avulla. Omia ja kavereiden lisäämiä tietoja voidaan selata mobiililaitteella tai verkkoselaimella. Tässä tutkielmassa keskitytään testaamaan Mopsin verkkoselaimella käytettäviä osia.

5.3 Hyökkäystyökalut ja testauksen kulku

Tässä alaluvussa esitellään verkkosovellusten testaamiseen käytetyt työkalut sekä nimetään ja kuvaillaan niillä tehdyt hyökkäykset. Testauksen kulku noudattelee aidon verkkohyökkäyksen kaavaa ja koostuu neljästä eri vaiheesta: tiedonkeräämisestä, käyttäjätunnusten selvittämisestä, automaattisesta haavoittuvuuksien etsinnästä ja kohdennetusta haavoittuvuuksien etsimisestä. (Barnett & Grossman, 2013)

Testauksen aikana molempiin sovelluksiin tehdään eri työkaluilla yhteensä 29 eri hyökkäystä. Hyökkäykset on nimetty käytettävän työkalun nimellä ja järjestysnumerolla riippuen siitä, monesko kyseisellä työkalulla tehty hyökkäys on kyseessä. Lisäksi luvussa kuusi eri sovelluksiin tehdyt hyökkäykset erotetaan lisäämällä hyökkäyksen nimeen kohteena olevan sovelluksen nimi.

5.3.1 Tiedonkeruu

Tiedonkeruuvaiheen tavoitteena on tunnistaa palvelimella oleva hyökkäyspinta-ala, eli paikallistaa palvelimelle asennetut internetin välityksellä käytettävät sovellukset ja palvelut, sekä niiden osoitteet ja avoimet portit. Tiedonkeruu toteutetaan *Nmap*, *Nikto* ja *Metasploit framework* skannaustyökalujen avulla. Työkalut keräävät tietoa palvelimesta ja sille asennetuista ohjelmista lähettämällä palvelimelle kutsuja ja tarkastelemalla palvelimen vastauksia. Vastausten avulla skannaustyökalut pystyvät päättelemään palvelimelle asennetut ohjelmistot ja versiot.

Tiedonkeruuvaihe jakautuu kolmeen eri skannaukseen. Ensimmäinen skannaus suoritetaan Nmap:lla ja se kohdistuu koko palvelimeen. Skannauksen tavoitteena on selvittää kohteena olevaa palvelinarkkitehtuuria. Sen avulla pyritään selvittämään muun muassa palvelimella olevien HTTP-, FTP- ja SSH-palvelimien nimet ja versiot. Kerättyjen tietojen avulla tulevia hyökkäyksiä voidaan kohdentaa esimerkiksi palvelimelta mahdollisesti löytyviin tunnettuja haavoittuvuuksia sisältäviin ohjelmistoversioihin.

Seuraavan skannauksen tavoitteena on kerätä tarkempia tietoja palvelimella olevasta HTTP-palvelinohjelmistosta, ja se kohdistuukin pelkästään palvelimella olevaan HTTP-palvelimeen. Skannaus suoritetaan *Nikto*-skannerilla. Nikton avulla pyritään löytämään palvelinohjelmistosta esimerkiksi virheellisiä tai puutteellisia asetuksia, jotka heikentävät palvelimen tietoturvaa. Tällaisia ovat esimerkiksi palvelimen HTTP-vastauksista puuttuvat turvallisuusotsikot.

Kolmas skannaus kohdistuu palvelimella oleviin verkkosovelluksiin ja se suoritetaan *Metasploit Frameworkin* sisältämän *hakurobotin* (web crawler) avulla. Hakurobotit etsivät HTML-sivuilta linkkejä ja seuraavat niitä. Tämän skannauksen tarkoituksena on löytää kaikki palvelimella olevat verkkosovellukset ja niiden kaikki avoimet osat.

Ensimmäinen palvelinarkkitehtuuriin kohdistuva skannaus suoritetaan *Nmap*-skannerin versiolla 7.60. Nmap:lla suoritetaan skannaukset *Nmap1* ja *Nmap2*. Nmap1 suoritetaan komennolla

```
nmap -sV http://www.verkkosovellus.com
```

Komentoon lisätään parametri `sV`, jolloin työkalu tutkii palvelimella avoimena olevia portteja pyrkien selvittämään niissä olevat palvelut ja niiden versiot. Nmap2-hyökkäys suoritetaan komennolla:

```
nmap -sV http://www.verkkosovellus.com --script-args
http.useragent="Mozilla/5.0 (X11; Linux x86_64;
rv:58.0) Gecko/20100101 Firefox/58.0"
```

Parametrin –script-args avulla muutetaan työkalun tekemien kutsujen User-Agent HTTP-otsikon sisältö vastaamaan Mozillan Firefox-selaimen version 58.0 tekemien kutsujen otsikkotietoa. Koska Nmap on tehty laillista tietoturvatestausta varten, se kertoo palvelimille User-Agent-otsikon avulla, että kutsut on tehty Nmap-työkalulla, jolloin monitorointityökalujen on helpompi havaita tehdyt skannaukset. Parametrin tarkoituksena on siis hämätä palvelimelle asennettuja monitorointityökaluja.

HTTP-palvelinohjelmistoon kohdistuvat skannaukset *Nikto1* ja *Nikto2* suoritetaan *Nikto*-skannerin versiolla 2.1.6. Molemmat skannaukset suoritetaan komennolla:

```
perl nikto.pl -host http://www.verkkosovellus.com
```

Nikto2-skannauksessa on sovelluksen asetustiedostoon muokattu sovelluksen lähettämä User-Agent-HTTP-otsikko samaan arvoon kuin Nmap2-hyökkäyksessä.

Kolmannessa skannauksessa suoritetaan skannaus *Metasploit1*. Skannauksessa käytetään *Metasploit Framework*:in version 4.16.19-dev *Metasploit Web Crawler* hakurobottia.

5.3.2 Käyttäjätunnusten selvittäminen

Kun verkkosovellusten sijainnit ja avoimet osat on löydetty, on aika aloittaa hyökkäyksen seuraava vaihe eli käyttäjätunnusten selvittäminen. Verkkosovelluksen sisäänkirjautuminen on lähes poikkeuksetta toteutettu käyttäjänimen ja salasanan avulla. Sisäänkirjautuminen onnistuu, jos käyttäjän syöttämä käyttäjänimi ja salasana vastaavat sovelluksen tietokantaan tallennettua käyttäjänimeä ja salasanaa. Koska sisäänkirjautumistoiminto on aina julkisesti saatavilla, ottaa vastaan käyttäjän syötettä ja on yhteydessä sovelluksen tietokantaan, sitä vastaan voidaan hyökätä SQL-injektion tai *sanakirjahyökkäyksen* avulla.

SQL-injektion avulla on tarkoitus saada käyttäjätunnukset selville varastamalla ne tietokannasta. Hyökkääjän kannalta parhaassa tilanteessa salasanat ovat tietokannassa selkokielisinä, jolloin sisäänkirjautuminen onnistuu välittömästi injektion onnistuessa. Salasanat tulisi kuitenkin salata ennen tietokantaan tallentamista. Tällöin hyökkääjän käsiin päätyvät vain salasanojen *tiivisteet* (hash).

Kaikki tutkielman SQL-injektioyritykset noudattavat samaa kaavaa. Ensimmäinen vaihe on varmistaa, että kohteena oleva metodi on haavoittuva SQL-injektiolle komennolla:

```
python sqlmap.py -r request.txt
```

Komennossa sqlmap-ohjelmalle annetaan parametrina tiedosto request.txt, joka sisältää ZAP:n avulla tallennetun HTTP-kutsun metodiin, josta ZAP on tunnistanut mahdollisen SQL-injektio -haavoittuvuuden.

Jos sqlmap löytää injektiohaavoittuvuuden, seuraavaksi voidaan sqlmap:n avulla listata palvelimelta löytyvät tietokannat komennolla:

```
python sqlmap.py -r request.txt --dbs
```

Tietokantalistauksen jälkeen voidaan selvittää yksittäisen tietokannan sisältämät taulut komennolla:

```
python sqlmap.py -r request.txt -D <tietokanta> --
tables
```

Tämän jälkeen yksittäisen tietokantataulun sisältö voidaan lukea komennolla

```
python sqlmap.py -r request.txt -D <tietokanta> -T
<taulu> --dump
```

Jos valittu tietokantataulu sisältää salasanatiivisteitä, voidaan tiivisteet yrittää murtaa injektion yhteydessä sanakirjahyökkäyksen avulla.

Sanakirjahyökkäyksessä salasanojen tiivisteitä yritetään arvata ennalta luodun sanalistan perusteella. Sanalistalla voi olla esimerkiksi 10000 yleisimmin käytössä olevaa salasanaa. Sanakirjahyökkäyksessä listalla olevista salasanoista muodostettuja tiivisteitä verrataan tietokannassa oleviin tiivisteisiin. Jos sama tiiviste löytyy listalta ja tietokannasta, hyökkääjä on onnistunut murtamaan käyttäjän salasanan. Sanakirjahyökkäys voidaan toteuttaa myös internetin välityksellä yrittämällä suorittaa sisäänkirjautuminen verkkosovellukseen hyökkäystyökalun avulla. Tällaisessa hyökkäyksessä hyökkäystyökalu pyrkii arvaamaan oikeita käyttäjätunnus–salasana -pareja ennalta luotujen sanalistojen avulla.

Tässä tutkielmassa SQL-injektioiden tekemiseen käytetään *sqlmap*-työkalun versiota 1.1.11.25 ja tässä vaiheessa sillä suoritetaan hyökkäykset *sqlmap1A* ja *sqlmap1B*. Molemmat hyökkäykset kohdistuvat sovelluksien sisäänkirjautumismetodiin. Hyökkäys sqlmap1A suoritetaan ilman lisäparametreja. Hyökkäykseen sqlmap1B lisätään kaikki sqlmap:n mukana tulevat verkkosovelluspalomuurien hämäämiseen tarkoitetut komentosarjat. Injektion onnistuessa tietokannasta saatuja salasanatiivisteitä yritetään murtaa automaattisesti sqlmap:n sisäänrakennetulla sanakirjahyökkäyksellä.

Verkon yli sisäänkirjautumismekanismiin kohdistunut sanakirjahyökkäys *Hydra* toteutetaan *THC Hydra* -työkalun versiolla 8.6. Hyökkäyksessä käytetään Kali Linuxin mukana tulevia sanalistoja. Kaikkien testitapauksien kattamiseksi sanalistoille lisätään yksi oikea käyttäjänimi–salasana -yhdistelmä, jotta saadaan toteutettua ainakin

yksi onnistunut sanakirjahyökkäys ja testattua sen monitorointi. Koska samoja sanalistoja käytettäessä hyökkäys sisältää aina täsmälleen samat kutsut, suoritetaan sanakirjahyökkäys ainoastaan DVWA:ta vastaan.

5.3.3 Automaattinen haavoittuvuuksien etsintä

Kun sovelluksen käyttäjätiedot on selvitetty, voidaan aloittaa automaattinen haavoittuvuuksien etsintä. Automaattisen haavoittuvuuksien etsinnän tarkoituksena on löytää verkkosovelluksesta logiikka- tai ohjelmointivirheiden seurauksena siinä olevia haavoittuvuuksia. Haavoittuvuuksien etsintä on toteutettu OWASP:n kehittämällä *Zed Attack Proxy:*n (ZAP) versiolla 2.7.0 ja tässä vaiheessa toteutetaan hyökkäys *ZAP*.

Hyökkäys tapahtuu asettamalla ZAP verkkoselaimen *välityspalvelimeksi*, jolloin verkkoselaimen verkkoliikenne kulkee hyökkäystyökalun kautta. Tämän jälkeen selaimella käydään läpi mahdollisimman tarkasti kaikki verkkosovelluksen ominaisuuden ja toiminnot. ZAP tallentaa kaikki selaimella tehdyt kutsut ja verkkosovelluksen vastaukset. Kun varsinainen hyökkäys aloitetaan, ZAP toistaa käyttäjän tekemiä kutsuja, muuttaen niiden parametreja sisältämään hyökkäyskoodeja. ZAP tunnistaa mahdollisen haavoittuvuuden sovelluksessa etsimällä hyökkäyskoodeja ja tuloksia sovelluksen vastauksista. ZAP myös vertailee selailun aikana verkkosovellukselta saatuja vastauksia hyökkäyksen aikana saataviin vastauksiin tunnistaakseen hyökkäyksen aiheuttamat muutokset sovelluksen vastauksissa. Nämä tunnistusmetodit voivat aiheuttaa myös vääriä hälytyksiä, joten ZAP:n havainnot on vielä validoitava kohdennetulla haavoittuvuuksien etsinnällä.

5.3.4 Kohdennettu haavoittuvuuksien etsintä

Kohdennetussa haavoittuvuuksien etsinnässä tarkoituksena on varmentaa ZAP:n löytämät haavoittuvuudet. Varmennus voidaan tehdä esimerkiksi viemällä sovelluksen tietokannasta tietoja SQL-injektion avulla tai saada XSS-hyökkäyksen avulla selaimeen näkyviin sivulle injektoitu ponnahdusikkuna (ks. kuva 7).

ZAP:n raportti näyttää havaintoon johtaneen kutsun ja sovelluksen vastauksen, jonka perusteella havainto on tehty. Osassa havainnoista ZAP:n raportista voidaan suoraan

nähdä, että kyseessä on oikea sovelluksesta löytynyt haavoittuvuus. Jos näin ei kuitenkaan ole, haavoittuvuuden olemassaolo varmistetaan erillisellä testillä, joka kohdistuu ainoastaan raportoituun metodiin.

Kaikki ZAP:n raportoimat SQL-injektiohaavoittuvuudet tarkastetaan sqlmap-työkalulla. Havainto todetaan oikeaksi ainoastaan, jos sqlmap onnistuu injektiossa. Muut epävarmat havainnot varmistetaan manuaalisesti toistamalla ZAP:n tekemä kutsu verkkoselaimen avulla.

6 Tulokset

Tässä luvussa esitellään tutkielman tulokset. Tuloksissa esitellään lokituksen ja monitoroinnin tehokkuutta verkkohyökkäyksen havaitsemisessa. Lisäksi tuloksissa todistetaan käytettyjen hyökkäystyökalujen toimivuus esittämällä työkalujen palvelimesta ja sovelluksesta keräämiä tietoja. Taulukossa 1 esitetään monitoroinnin tehokkuus eri hyökkäysvaiheiden havaitsemisessa.

Taulukko 1. Monitoroinnin tehokkuus hyökkäysten tunnistamisessa hyökkäysvaiheittain.

Vaihe	Kutsujen lukumäärä	Havaintoprosentti
1. Tietojenkeräys	15287	69.2%
2. Käyttäjätunnusten selvittäminen	2213	65.2%
3. Automaattinen haavoittuvuuksien etsintä	61619	50.5%
4. Manuaalinen haavoittuvuuksien etsintä	3778	70.2%

Tämä luku jakautuu viiteen alalukuun. Neljässä ensimmäisessä alaluvussa esitellään tietoturvatestauksen eri vaiheiden eli tiedonkeruun, käyttäjätietojen selvittämisen, automaattisen haavoittuvuuksien etsinnän ja kohdennetun haavoittuvuuksien etsinnän aikana hyökkäystyökalujen avulla palvelimesta ja sovelluksista kerättyjä tietoja ja analysoidaan tietojen kriittisyyttä sovellusten tietoturvan näkökulmasta. Lisäksi näissä alaluvuissa pureudutaan tarkemmin monitoroinnin tehokkuuteen eri vaiheiden tunnistamisessa. Viimeinen alaluku esittää tuloksista tehdyt johtopäätökset.

6.1 Tiedonkeruu

Tiedonkeruuvaiheessa suoritettiin koko palvelimeen kohdistuvia skannauksia, joiden tarkoituksena oli kartoittaa palvelimen hyökkäyspinta-ala, eli selvittää palvelimelle asennetut ohjelmistot, niiden versiot ja sijainnit sekä löytää palvelimella olevat verkkosovellukset. Tässä vaiheessa suoritettiin skannauksia Nmpa, Nikto ja Metasploit -skannereilla.

Taulukossa 2 esitetään monitoroinnin tehokkuus tiedonkeruuvaiheen tunnistamisessa. Vastaavanlaiset taulukot löytyvät myös luvuista 6.2, 6.3 ja 6.4. Taulukoissa esitetään suoritetun hyökkäyksen nimi, kesto ja hyökkäyksen aikana palvelimeen kohdistuneiden kutsujen lukumäärä. Monitoroinnin osalta esitetään, kuinka suuri prosentti hyökkäyksen aikana tehdyistä kutsuista tunnistettiin haitalliseksi ja kuinka kauan hyökkäyksen alusta kesti, ennen kuin hyökkäyksestä hälytettiin.

Taulukko 2. Monitoroinnin tehokkuus tiedonkeruuvaiheen havaitsemisessa.

Hyökkäys	Kesto	Kutsujen lukumäärä	Havainto-prosentti	Hälytysviive
Nmap1	15s	12	66,7 %	6s
Nmpa2	14s	12	33,3 %	Ei hälytystä
Nikto1	7min 39s	7442	97,0 %	0s
Nikto2	7min 3s	7445	44,8 %	0s
Metaploit1	2min 2s	376	1,3 %	1min 4

6.1.1 Nmap

Nmap-skannauksella selvitettiin palvelimen arkkitehtuuria ja sen avulla saatiin selville, että kohdepalvelimen käyttöjärjestelmä oli Linux ja HTTP-palvelinohjelmistona Apachen HTTP-palvelin, joka kuunteli HTTP- ja HTTPS-protokollaa. Lisäksi palvelimella oli portissa 22 OpenSSH-palvelimen versio 7.2.

Tulokset ovat erittäin yleisluontoisia, eikä mikään tieto itsessään ole haitallinen palvelimen tietoturvan kannalta. Tuloksia voidaan käyttää hyödyksi räätälöidessä myöhempiä hyökkäyksiä, mutta mitään suoraa hyötyä tai haavoittuvuutta tuloksista ei voida havaita.

Nmap:lla suoritettiin skannaukset Nmap1 ja Nmap2. Molempien skannausten aikana palvelimeen kohdistui 12 HTTP-kutsua. Nmap1-skannauksen kutsuista monitorointi havaitsi haitallisiksi 66,7 % ja laukaisi hälytyksen hyökkäyksestä. Kun skannerin lähettämä User-Agent-HTTP-otsikko muutettiin vastaamaan tavallista verkkoselainta Nmap2-skannauksessa, putosi havaintoprosentti 33,3 prosenttiin, eikä skannaus enää laukaissut hälytystä.

Tuloksista voidaan päätellä, että monitorointi ei välttämättä pysty havaitsemaan Nmap-skannauksia. Vaikka Nmap-skannaukset eivät itsessään ole haitallisia, olisi niiden havaitseminen hyödyllistä, sillä ne voivat olla merkkinä alkavasta hyökkäyksestä sovellusta vastaan.

6.1.2 Nikto

Nikto-skannauksen avulla pyrittiin selvittämään tarkempia tietoja palvelimelle asennetusta HTTP-palvelimesta. Nikto-skannaus vahvisti Nmap-skannausten havainnon siitä, että palvelimella on Apachen HTTP-palvelinohjelmisto. Lisäksi skannauksessa selvisi, että palvelimen vastauksista puuttuu X-Frame-Options, X-XSS-Protection ja X-Content-Type-Options HTTP-otsikot. Otsikoiden puuttuminen ei itsessään ole haavoittuvuus, mutta se helpottaa esimerkiksi mahdollisten XSS-haavoittuvuuksien hyödyntämistä.

Nikto1-skannaus suoritettiin ilman mitään toimenpiteitä hyökkäyksen piilottamiseksi. Tämän skannauksen aikana tehdyistä kutsuista monitorointi havaitsi haitallisiksi 97 prosenttia kutsuista ja laukaisi hälytyksen välittömästi hyökkäyksen alettua. Kun skannerin lähettämä User-Agent-otsikko muutettiin vastaamaan tavallista verkkoselainta, putosi havaintoprosentti 44,8 prosenttiin, mutta hälytys laukesi edelleen välittömästi hyökkäyksen alettua.

Tulosten perusteella voidaan todeta, että monitorointi tunnisti Nikto-skannaukset tehokkaasti hämäysyrityksistä huolimatta. Lokituksesta myös selviää, että hyökkääjät

ovat löytäneet mahdollisen haavoittuvuuden ja lokeista selviää myös havainnon aiheuttanut metodi. Näin haavoittuvuutta voidaan lähteä ratkomaan tarvittaessa välittömästi.

6.2 Käyttäjätunnusten selvittäminen

Käyttäjätunnusten selvittämiseksi yritettiin verkkosovellusten käyttäjätunnuksia selvittää sanakirjahyökkäyksen ja SQL-injektion avulla. Käyttäjätunnusten selvittämisen tarkoituksena on päästä kirjautumaan sisään verkkosovellukseen, sillä useissa sovelluksissa suurin osa toiminnallisuudesta eli hyökkäyspinta-alasta on sisäänkirjautumisen takana. Taulukossa 3 esitetään monitoroinnin tehokkuus käyttäjätunnusten selvittämisen havaitsemisessa.

Taulukko 3. Monitoroinnin tehokkuus käyttäjäntunnusten selvittämisen havaitsemisessa.

Hyökkäys	Kesto	Kutsujen lukumäärä	Havainto-prosentti	Hälytysviive
Hydra	1min 48s	266	0 %	Ei hälytystä
Sqlmap login A DVWA	1min 54s	297	97,6 %	0s
Sqlmap login B DVWA	1min 28s	298	97,0 %	0s
Sqlmap login A Mopsi	1min 44s	614	93,5 %	1s
Sqlmap login B Mopsi	2min 27s	738	39,3 %	9s

6.2.1 THC Hydra

Sanakirjahyökkäys käyttäjätunnusten selvittämiseksi suoritettiin THC Hydra -ohjelmalla. Koska sanakirjahyökkäyksessä lähetettävät kutsut ovat riippuvaisia käytettävistä sanalistoista eivätkä verkkosovelluksen metodeista, suoritettiin sanakirjahyökkäys vain DVWA:ta vastaan.

Sanakirjahyökkäyksessä palvelimelle lähetettiin 266 kutsua. Monitorointi ei tunnistanut yhtään kutsua haitalliseksi, eikä laukaissut hälytystä. Monitoroinnin tehottomuus verkon yli tapahtuvan sanakirjahyökkäyksen tunnistamisessa selittyy sillä, että työkalu lähettää sovellukseen vain sisäänkirjautumiskutsuja eri käyttäjänimi–salasana -kombinaatiolla. Koska tämä on täysin normaalia sovellukseen kohdistuvaa liikennettä, ei monitorointi kykene havaitsemaan sitä. Sanakirjahyökkäyksiltä suojautumisen tulisikin olla rakennettuna sovelluksen sisäänkirjautumislogiikkaan esimerkiksi siten, että viiden virheellisen sisäänkirjautumisyrityksen jälkeen käyttäjän tunnukset lukitaan viideksitoista minuutiksi. Tämä ei estä sanakirjahyökkäystä, mutta hidastaa sen suorittamista huomattavasti.

6.2.2 Sqlmap

Sqlmap-sovelluksella suoritettujen SQL-injektioiden avulla yritettiin selvittää molempien sovellusten käyttäjätunnuksia. Molempiin sovelluksiin kohdistettiin hyökkäykset *Sqlmap login A* ja *Sqlmap login B*. A ja B hyökkäyksissä erona on se, että B-hyökkäyksessä käyttöön otettiin monitoroinnin häirintään tarkoitetut Sqlmap-työkalun komentosarjat. SQL-injektion avulla ei onnistuttu murtautumaan kumpaankaan verkkosovellukseen.

DVWA:han kohdistuneessa häirinnättömässä A hyökkäyksessä palvelimeen kohdistui 622 kutsua. Näistä kutsuista 98,4 % havaittiin haitalliseksi ja hälytys laukaistiin 2 sekuntia hyökkäyksen alkamisesta. Häirintäkomentosarjojen kanssa suoritettu B hyökkäys kohdisti sovellukseen 419 kutsua, joista 95,5 % tunnistettiin haitalliseksi. Hälytys laukesi 4 sekuntia hyökkäyksen alkamisen jälkeen.

Mopsi-sovellukseen kohdistunut A hyökkäys kohdisti palvelimeen 614 kutsua, joista 93,5 % tunnistettiin haitalliseksi ja hälytys annettiin sekunnin kuluttua hyökkäyksen alkamisesta. Häirintäkomentosarjojen kanssa suoritettuna hyökkäys suoritti palvelimelle 629 kutsua, joista haitalliseksi luokiteltiin 48,5 %. Hälytys hyökkäyksestä saatiin 9 sekuntia hyökkäyksen alkamisesta.

Tuloksista nähdään, että monitorointi on erittäin tehokas keino SQL-injektioiden tunnistamisessa. Monitorointi tunnisti kaikki injektioyritykset ja hälytykset annettiin nopeasti hyökkäyksen alkamisen jälkeen.

6.3 Automaattinen haavoittuvuuksien etsintä

Automaattisen haavoittuvuuksien etsinnän tarkoituksena oli käydä sovellusten toiminnallisuus läpi mahdollisimman tarkasti ja etsiä sovelluksista haavoittuvuuksia. Tässä vaiheessa suoritettiin skannaukset ZAP DVWA ja ZAP Mopsi OWASP ZAP -työkalulla. Työkalu antaa aina ilmoituksen mahdollisesta haavoittuvuudesta, joten alaluvuissa esitettävät havainnot ovat vasta mahdollisia haavoittuvuuksia ja niiden olemassaolo ja hyödynnettävyys esitellään tarkemmin luvussa 6.4. Taulukossa 4 esitetään monitoroinnin tehokkuus automaattisen haavoittuvuuksien etsinnän tunnistamisessa.

Taulukko 4. Monitoroinnin tehokkuus automaattisen haavoittuvuuksien etsinnän havaitsemisessa.

Hyökkäys	Kesto	Kutsujen lukumäärä	Havainto-prosentti	Hälytysviive
ZAP DVWA	25min 29s	7181	52.0%	1s
ZAP Mopsi	1h 30min 20s	54438	50.3%	3s

6.3.1 ZAP Damn Vulnerable Web Application

Damn Vulnerable Web Application on tietoturvatestausta varten tehty, nimensä mukaisesti Helkkarin Haavoittuvainen Verkkosovellus, joka toimi tutkielmassa verrokkisovelluksena hyökkäysmetodien toimivuuden varmistamiseksi. ZAP:lla suoritettu skannaus todisti metodit toimiviksi, sillä sen avulla sovelluksesta löytyi neljä matalan tason, yksi keskitason ja kuusi korkean tason ongelmaa tai haavoittuvuutta. Löytyneet haavoittuvuudet ovat:

- Matala taso
    - Puuttuva X-Content-Type-Options HTTP-otsikko
    - Puuttuva X-XSS-Protection HTTP-otsikko
    - Evästeistä puuttuva HttpOnly-turvallisuusasetus
    - Ulkoiselta sivulta ladattu JavaScript-lähdekoodi

- Keskitaso
  - Puuttuva X-Frame-Options HTTP-otsikko
- Korkea taso
  - Pysyvä XSS-haavoittuvuus kahdessa metodissa
  - Ei-pysyvä XSS-haavoittuvuus kahdeksassa metodissa
  - SQL-injektio kolmessa metodissa
  - MySQL-spesifi SQL-injektio
  - *Path Traversal* -haavoittuvuus
  - *Remote OS Command Injection* -haavoittuvuus

Kaikki sovelluksesta löytyneet matalan ja keskitason ongelmat eivät itsessään ole tietoturvahaavoittuvuuksia, mutta helpottavat sovelluksesta löytyvien haavoittuvuuksien hyödyntämistä. Puuttuvat X-Content-Type-Options, X-XSS-protection ja X-Frame-Options HTTP-otsikot altistavat sovelluksen XSS-haavoittuvuuksille. Evästeistä puuttuva HttpOnly-asetus mahdollistaa evästeiden muokkaamisen selaimessa JavaScript:lla, jolloin esimerkiksi onnistuneen XSS-hyökkäyksen avulla voidaan manipuloida käyttäjien evästeitä. Ulkoiselta sivulta ladatussa JavaScript-lähdekoodissa ongelmana on se, että verkkosovelluksen ylläpitäjä ei pysty takaamaan ladatun koodin turvallisuutta.

Korkean tason havainnoista XSS- ja injektiohaavoittuvuudet on esitelty luvussa 3, joten seuraavaksi esitellään lyhyesti vain Path Traversal -haavoittuvuus. Path Traversal -haavoittuvuus mahdollistaa palvelimen hakemistojen selaamisen ja tiedostojen lukemisen verkkoselaimen avulla. Käytännössä tämä tarkoittaa siis sitä, että hyökkääjällä on pääsy palvelimelle lukuoikeuksin. Tällaista haavoittuvuutta voidaan hyödyntää esimerkiksi lukemalla unix-palvelimen /etc/passwd tiedoston sisältö, jolloin saadaan selville kaikki palvelimen käyttäjät. Tätä tietoa voidaan hyödyntää esimerkiksi, kun muodostetaan sanalistoja SSH-tunnusten selvittämiseen tarkoitettua sanakirjahyökkäystä varten.

Monitorointi tunnisti ZAP-skannauksen tehokkaasti. Skannauksen aikana tehtiin 2781 kutsua, joista 52 % tunnistettiin haitallisiksi. Hälytys laukaistiin sekunnin kuluttua hyökkäyksen alkamisesta.

6.3.2 ZAP Mopsi

Mopsi-sovelluksen skannaus tuotti neljä matalan tason, kaksi keskitason ja neljä korkean tason havaintoa:

- Matala taso
  - Puuttuva X-Content-Type-Options HTTP-otsikko
  - Puuttuva X-XSS-Protection HTTP-otsikko
  - Evästeistä puuttuva HttpOnly-turvallisuusasetus
  - Ulkoiselta sivulta ladattu JavaScript-lähdekoodi
- Keskitaso
  - Puuttuva X-Frame-Options HTTP-otsikko
  - Sovellus paljastaa oletusvirheviestejä
- Korkea taso
  - Ei-pysyvä XSS-haavoittuvuus kahdessa metodissa
  - SQL-injektio neljässä metodissa
  - Path Traversal -haavoittuvuus
  - Varmentamaton uudelleen ohjaus

Matalan ja keskitason ongelmat ovat yhtä poikkeusta lukuun ottamatta samoja kuin DVWA:sta löydetyt ongelmat ja johtuvat palvelimen asetuksista. DVWA:sta poiketen Mopsi palauttaa joissakin ongelmatilanteissa käyttäjälle muokkaamattomana oletusvirheviestejä. Riskinä tässä on se, että viestit voivat sisältää esimerkiksi sovelluksen pinovedoksen tai tietokannan virhekoodeja. Näiden tietojen avulla hyökkääjä voi saada selville, miksi hyökkäys ei onnistunut ja parantaa hyökkäyskoodia seuraavaa yritystä varten. Korkean tason haavoittuvuuksia Mopsista raportoitiin neljä kappaletta.

Monitorointi tunnisti Mopsiin kohdistuneen ZAP-skannauksen tehokkaasti. Skannauksen aikana palvelimeen kohdistui 54 438 kutsua, joista 50,3 % tunnistettiin haitalliseksi. Hälytyksen skannaus laukaisi kolmessa sekunnissa.

6.4 Kohdennettu haavoittuvuuksien etsintä

Kohdennetun haavoittuvuuksien etsinnän tarkoituksena oli varmistaa ZAP:n tekemät kriittiset havainnot yrittämällä hyödyntää sen ilmoittamia haavoittuvuuksia. Koska DVWA:n osalta oli tiedossa, että se sisältää kaikki ZAP:n ilmoittamat havainnot, suoritettiin sitä vastaan vain yksi hyökkäys jokaista havaintotyyppiä kohden, jotta käytettyjen metodien toimivuus voidaan todeta. Mopsin osalta testattiin kaikkien havaintojen paikkansa pitävyys. Taulukossa 5 esitetään monitoroinnin tehokkuus hyökkäysvaiheen tunnistamisessa.

Taulukko 5. Monitoroinnin tehokkuus manuaalisen haavoittuvuuksien etsinnän havaitsemisessa.

Hyökkäys	Kesto	Kutsujen lukumäärä	Havainto-prosentti	Hälytysviive
XSS_R DVWA	0s	1	100 %	0s
SQLMap DVWA A	1min 52s	222	94,1 %	0s
SQLMap DVWA B	1min 19s	230	93,5 %	0s
PT DVWA	0s	1	100 %	0s
ROCI DVWA	0	1	100 %	0s
XSS_R Mopsi 1	0s	1	100 %	0s
XSS_R Mopsi 2	0s	1	100 %	0s
SQLMap Mopsi 1A	5min 35s	802	91,5 %	29s
SQLMap Mopsi 1B	22min 7s	895	41,8 %	24s
SQLMap Mopsi 2A	8min 7s	284	82,7 %	25s

SQLMap Mopsi 2B	2min 5s	172	32,0 %	45s
SQLMap Mopsi 3A	2min 49s	173	89,0 %	1min 32s
SQLMap Mopsi 3B	2min 0s	163	33,7 %	11s
SQLMap Mopsi 4A	30s	160	96,9 %	1s
SQLMap Mopsi 4B	27s	170	41,8 %	2s
PT Mopsi	0s	1	100 %	0s
REDIRECT Mopsi	0s	1	0 %	Ei hälytystä

6.4.1 DVWA

Automaattisessa haavoittuvuuksien etsinnässä DVWA:sta löytyi kuusi erilaista korkean tason haavoittuvuutta ja yhteensä kuusitoista metodia merkittiin mahdollisesti haavoittuvaisiksi. Jokaista haavoittuvuutta kohden yksi metodi testattiin tarkemmin haavoittuvuuden varmistamiseksi. Varmistettavat haavoittuvuudet ovat ei-pysyvä XSS-haavoittuvuus, SQL-injektio, Path Traversal -haavoittuvuus ja Remote OS Command Injection.

Taulukko 6. ZAP:n raportoima XSS-haavoittuvuus

Haavoittuvuus	**Ei-pysyvä XSS**
URL	`/vulnerabilities/xss_s/`
HTTP-metodi	`POST`
Parametri	`txtName`
Hyökkäyskoodi	`</div><script>alert(1);</script><div>`

Taulukossa 6 esitetty XSS-haavoittuvuus varmistettiin testillä DVWA XSS_S. Pysyvän XSS-hyökkäyksen mahdollistava haavoittuvuus saatiin varmistettua ja sivulle saatiin injektoitua taulukon esittämä hyökkäyskoodi. Testin aikana suoritettiin yksi kutsu, jonka monitorointi havaitsi. Testi laukaisi myös hälytyksen.

Taulukko 7. ZAP:n raportoima SQL-injektio haavoittuvuus

Haavoittuvuus	**SQL-injektio**
URL	`/vulnerabilities/sqli/`
HTTP-metodi	`GET`
Parametri	`id`
Hyökkäyskoodi	`1' AND '1'='1' --`

Taulukon 7 esittämä SQL-injektio haavoittuvuus varmistettiin testillä SQLMap DVWA A. Haavoittuvuus saatiin varmistettua ja sqlmap:lla suoritetun injektion avulla pystyttiin lukemaan sovelluksen tietokannasta kaikki sovelluksen käyttäjien nimet ja salasanatiivisteet. Testin aikana suoritettiin 222 kutsua. Haitalliseksi kutsuista tunnistettiin 94,1 % ja hälytys laukaistiin välittömästi testin alettua. A testin lisäksi suoritettiin B testi, jossa hyökkäykseen oli lisätty monitoroinnin häirintään tarkoitettuja komentosarjoja. Myös tämä testi onnistui injektiossa. B testin aikana suoritettiin 230 kutsua, joista 93,5 % tunnistettiin haitallisiksi ja hälytys laukaistiin niin ikään välittömästi testauksen alettua.

Taulukko 8. ZAP:n raportoima polunselaushaavoittuvuus

Haavoittuvuus	**Path Traversal**
URL	`/vulnerabilities/fi/`
HTTP-metodi	`GET`
Parametri	`page`

Hyökkäyskoodi	`/etc/passwd`

Path Traversal -haavoittuvuus varmistettiin testillä PT DVWA ja Remote OS Command Injection-haavoittuvuus testillä ROCI DVWA. Molemmat haavoittuvuudet saatiin varmistettua ja hyökkäysten avulla pystyttiin lukemaan palvelimen `/etc/passwd` tiedoston sisältö. Kummankin testin aikana suoritettiin yksi kutsu, jonka monitorointi tunnisti ja laukaisi siitä hälytyksen.

6.4.2 Mopsi

Automaattisessa haavoittuvuuksien etsinnässä Mopsista löydettiin neljä mahdollista korkean tason haavoittuvuutta yhteensä kahdeksasta eri metodista. Kohdennetussa haavoittuvuuksien etsinnässä tarkoituksena oli varmistaa haavoittuvuuksien olemassaolo.

Taulukko 9. ZAP:n Mopsista raportoimat SQL-injektiohaavoittuvuudet.

Haavoittuvuus	**SQL-injektio**
URL	`/index.php`
HTTP-metodi	`GET`
Parametri	`query`
Hyökkäyskoodi	`query OR 1=1 --`
URL	`/login/login.php`
HTTP-metodi	`POST`
Parametri	`create_username`
Hyökkäyskoodi	`ZAP AND 1=1 --`
URL	`/index.php`

HTTP-metodi	`GET`
Parametri	`mode`
Hyökkäyskoodi	`normal AND 1=1 --`
URL	`/index.php`
HTTP-metodi	`GET`
Parametri	`search`
Hyökkäyskoodi	`" items[i]["name"] " OR 1=1 --`

Taulukossa 9 esitetään ZAP:n Mopsista raportoimat SQL-injektiohaavoittuvuudet. Haavoittuvuuksien olemassaolo tarkastettiin hyökkäyksillä SQLMap Mopsi 1–4. Mikään kohdennettu hyökkäys ei tuottanut tulosta, eikä injektiohaavoittuvuuksien olemassaoloa pystytty varmistamaan.

Monitorointi havaitsi ja hälytti kaikista tehdyistä hyökkäyksistä. Jokainen hyökkäys suoritettiin kahteen kertaan: ensin ilman monitoroinnin häirintää ja sen jälkeen häirinnän kanssa. Häirintä pienensi hyökkäyksien tunnistamisprosenttia, mutta myös kaikki häirinnän kanssa suoritetut hyökkäykset tunnistettiin ja niistä annettiin hälytys.

Taulukko 10. ZAP:n Mopsista raportoimat XSS-haavoittuvuudet.

Haavoittuvuus	**Ei-pysyvä XSS**
URL	`/mopsiMetaSearch/testImage.php`
HTTP-metodi	`GET`
Parametri	`url`

Hyökkäyskoodi	`"><script>alert(1);</script>`
URL	`/index.php`
HTTP-metodi	`GET`
Parametri	`search`
Hyökkäyskoodi	`</script><script>alert(1);</script><script>`

Taulukkoon 10 on listattu ZAP:n Mopsista raportoimat mahdolliset XSS-haavoittuvuudet. Havainnot pyrittiin varmistamaan hyökkäyksillä XSS_R Mopsi 1 ja 2. Index.php-sivun search-parametrista raportoitua haavoittuvuutta ei pystytty hyödyntämään. Palvelin palautti hyökkäyskoodin vastauksensa mukana, mutta koodia ei käytetty osana sivun lähdekoodia ja ponnahdusikkuna ei tullut näkyville. Metodista /mopsiMetaSearch/testImage.php raportoitu XSS-haavoittuvuus pystyttiin varmistamaan ja injektoitu ponnahdusikkuna tuli näkyviin selaimessa.

Testauksen aikana suoritettiin kaksi kutsua. Monitorointi tunnisti molemmat kutsuista haitallisiksi ja laukaisi niistä hälytyksen.

Taulukko 11. ZAP:n Mopsista raportoima polunselaushaavoittuvuus

Haavoittuvuus	**Polunselaus**
URL	`/login/login.php`
HTTP-metodi	`POST`
Parametri	`login_goto`
Hyökkäyskoodi	`login.php`

Taulukossa 11 on ZAP:n raportoima havainto Mopsissa olevasta polunselaushaavoittuvuudesta. Havainnon varmistamiseksi suoritettiin hyökkäys PT Mopsi, jonka aika-

na tehtiin yksi kutsu. Haavoittuvuutta ei pystytty todentamaan ja monitorointi havaitsi hyökkäyksen ja hälytti siitä.

Taulukko 12. ZAP:n Mopsista raportoima varmentamaton uudelleenohjaus.

Haavoittuvuus	**Varmentamaton uudelleenohjaus**
URL	`/login/login.php`
HTTP-metodi	`POST`
Parametri	`login_goto`
Hyökkäyskoodi	`4743201327599272056.owasp.org`

Taulukossa 12 esitetään ZAP:n Mopsista raportoima varmentamaton uudelleenohjaus. Haavoittuvuuden olemassaolo varmistettiin hyökkäyksellä REDIRECT Mopsi. Hyökkäyksessä ZAP:n käyttämä hyökkäyskoodi korvattiin arvolla `https://www.google.fi`. Haavoittuvuus pystyttiin varmentamaan ja onnistuneen sisäänkirjauksen jälkeen Mopsi ohjasi käyttäjän parametrin määrittämälle sivulle, eli osoitteeseen `https://www.google.fi`.

6.5 Johtopäätökset

Suoritettujen testien perusteella voidaan todeta, että verkkosovellukseen kohdistuvan liikenteen monitorointi ja lokitus on tehokas keino sovellukseen kohdistuvien hyökkäyksien havaitsemisessa. Suoritetuista 29 hyökkäyksestä monitorointi tunnisti 27 ja laukaisi hälytyksen kolmea hyökkäystä lukuun ottamatta kaikista.

Havaitsematta jäivät hyökkäykset Nmap2, Hydra ja REDIRECT Mopsi. Tämä johtuu siitä, että kaikki mainitut hyökkäykset käyttävät verkkosovellusten normaaleja toimintoja ilman haitallisia parametriarvoja, jolloin monitorointi ei erota hyökkääjää tavallisesta käyttäjästä.

Tulosten perusteella OWASP:n vuoden 2017 Top 10 listauksen kohta ”riittämätön lokitus ja monitorointi” voidaan todeta listalle kuuluvaksi. Lokituksen ja monitoroin-

nin avulla pystyttiin tunnistamaan verkkosovelluksiin kohdistuvat automaattiset hyökkäykset tehokkaasti. Lisäksi monitorointiin lisätyn hälytyslogiikan avulla pystyttiin havaitsemaan verkkosovelluksiin kohdistuvat hyökkäykset reaaliaikaisesti ja tarvittaessa olisi pystytty myös keskeyttämään tai ainakin hidastamaan käynnissä olevaa hyökkäystä.

7 Yhteenveto

Tässä tutkielmassa on selvitetty lokituksen ja monitoroinnin hyödyllisyyttä verkkosovelluksiin kohdistuvien tietoturvauhkien havaitsemisessa. Tutkielma on rajattu koskemaan automaattisilla työkaluilla suoritettavia hyökkäyksiä, jotka hyödyntävät sovelluksista löytyviä ohjelmointi- ja logiikkavirheistä johtuvia haavoittuvuuksia.

Tutkielman teoriaosuudessa esiteltiin OWASPin *Top 10 Kriittisintä Verkkosovellusten Tietoturvariskiä* -listauksen pohjalta yleisimpiä verkkosovellusten tietoturvahaavoittuvuuksia. Lisäksi teoriaosuudessa esiteltiin verkkosovellusten tietoturvakerros ja tapoja sen toteuttamiseen alan kirjallisuuden pohjalta. Käytännönosuudessa testipalvelimelle rakennettiin tietoturvakerros, joka lokitti ja monitoroi palvelimen HTTP- ja HTTPS-liikenteen. Tietoturvakerroksen käyttökelpoisuutta testattiin simuloimalla palvelimelle asennettuihin sovelluksiin kohdistuvia verkkohyökkäyksiä sovellusten tietoturvatestauksen muodossa.

Testauksen perusteella voidaan todeta, että lokitus ja monitorointi ovat tehokkaita keinoja automaattisilla hyökkäystyökaluilla suoritettavien hyökkäyksien tunnistamisessa ja tunnistamiseen kuluvan ajan lyhentämisessä.

Tutkielmassa käsiteltiin vain sovelluksesta löytyvien ohjelmointi- ja logiikkavirheiden mahdollistamia haavoittuvuuksia ja hyökkäyksiä. Sovelluksen tietoturva koostuu kuitenkin monista muistakin osa-alueista kuin pelkästään sovelluksissa olevista virheistä. Tässä tutkielmassa kerätty näyttö koskettaakin vain pientä osaa kokonaistietoturvasta ja noudattamalla tutkielmassa esiteltyjä keinoja ei voida taata sovelluksen kokonaistietoturvaa. Tutkielman ulkopuolelle jäi myös lokituksen ja monitoroinnin hyödyllisyys onnistuneiden hyökkäysten laajuuden selvittämisessä. Tämä olisi mielenkiintoinen kohde jatkotutkimukselle.

Tutkielmassa saatujen tulosten perusteella OWASP:in vuoden 2017 listauksen kohta A10: *riittämätön lokitus ja monitorointi* on aiheellinen lisä listaukseen. Monitoroinnin avulla mahdollisuudet sovellukseen kohdistuvan hyökkäyksen havaitsemiseen paranevat selvästi. Myös hyökkäysten havaitsemiseen kuluvaa aikaa pystytään lyhentämään hälytysten avulla.

Viitteet

Barnett, R, C., Grossman, J. (2013): *Web Application Defender's Cookbook: Battling Hackers and Protecting Users*, Wiley Pub Inc., Indianapolis, IN.

Equifax (2017): *Equifax Announces Cybersecurity Incident Involving Consumer Information*, https://investor.equifax.com/news-and-events/news/2017/09-07-2017-213000628 (10.06.2018)

Folini, C. (2018) *Apache / ModSecurity Tutorials*, `https://www.netnea.com/cms/apache-tutorials/` (29.10.2018)

Heiderich, M., Nava, E, A, N., Heyes, G., Lindsay, D. (2011): *Web application obfuscation: '-/WAFs..Evasion..Filters//alert(/Obfuscation/)-'*, Elsevier, Boston, MA.

HS (8.9.2017) *Hakkerit varastivat 143 miljoonan ihmisen henkilötiedot Yhdysvalloissa – tietoja epäillään käytettävän identiteettivarkauksiin*. Helsingin Sanomat, `https://www.hs.fi/ulkomaat/art-2000005358582.html` (29.10.2018)

NIST (2016) *National Vulnerability Database*. National Institute of Standards and Technology, `https://nvd.nist.gov/home.cfm` (29.10.2018)

OWASP (2013) *OWASP Top 10 Project*. The Open Web Application Security Project, `https://www.owasp.org/index.php/Top_10_2013-Top_10` (29.10.2018)

OWASP (2017) *OWASP Top 10 - 2017 The Ten Most Critical Web Application Security Risks*. The Open Web Application Security Project, `https://www.owasp.org/images/7/72/OWASP_Top_10-2017_%28en%29.pdf.pdf` (29.10.2018)

OWASP (2018a) *Application Security Verification Standard Project*. The Open Web Application Security Project, `https://www.owasp.org/index.php/Category:OWASP_Application_Security_Verification_Standard_Project` (29.10.2018)

OWASP (2018b) *Cross-Site Request Forgery (CSRF) Prevention Cheat Sheet*. The Open Web Application Security Project, https://www.owasp.org/index.php/Cross-Site_Request_Forgery_(CSRF)_Prevention_Cheat_Sheet (29.10.2018)

Ponemon Instituutti (2017): *Cost of Data Breach Study: Global overview*, https://www-01.ibm.com/common/ssi/cgi-bin/ssialias?htmlfid=SEL03130WWEN& (29.10.2018)

Ragan, S. (26.4.2017): *Contrast Security responds to OWASP Top 10 controversy*, CSO. https://www.csoonline.com/article/3192505/security/contrast-security-responds-to-owasp-top-10-controversy.html (29.10.2018)

Shema, M. (2012): *Hacking web apps: detecting and preventing web application security problems*, Syngress, Waltham, MA.

Shema, M. (2010): *Seven deadliest web application attacks*, Syngress, Waltham, MA.

Stuttard, D., Pinto, M. (2011): *Web Application Hacker's Handbook: Finding and Exploiting Security Flaws*, Wiley Pub Inc., Indianapolis, IN.

WASC (2009) *Web Application Security Scanner Evaluation Criteria.* Web Application Security Consortium, http://www.webappsec.org/ (29.10.2018)

W3C (2015) *Specifications: Content Security Policy Level 2* World Wide Web Consortium, https://www.w3.org/TR/CSP2/ (29.10.2018)

W3Tech (2018): *Technologies Overview,* https://w3techs.com/technologies (29.10.2018)