Joel Lehtonen
Kristian Siljander

Web-palvelut ja niihin kohdistuvan poikkeavan tietoliikenteen analyysi käyttäen diffuusiokuvausmenetelmää

Tietotekniikan
(Mobiilijärjestelmät)
pro gradu -tutkielma
18. lokakuuta 2010

JYVÄSKYLÄN YLIOPISTO
TIETOTEKNIIKAN LAITOS

Jyväskylä

Tekijät: Joel Lehtonen ja Kristian Siljander
Yhteystiedot: joel.lehtonen@iki.fi, kristian.siljander@jyu.fi
Työn nimi: Web-palvelut ja niihin kohdistuvan poikkeavan tietoliikenteen analyysi käyttäen diffuusiokuvausmenetelmää
Title in English: Web Services and Anomalous Web Traffic Detection Using Diffusion Maps
Työ: Tietotekniikan (Mobiilijärjestelmät) pro gradu -tutkielma
Sivumäärä: 109
Tiivistelmä: Viime vuosina Web-sovellusten ja -palveluiden suosio on ollut voimakkaassa kasvussa, ja monet palveluista ovat nykyisin kriittisiä osia yhteiskuntamme toimivuuden kannalta. Tämän johdosta palveluiden suojaaminen on noussut tärkeään roolin. Hyökkäysten myös muuttuessa yhä hienostuneimmiksi, eivät perinteiset tietoturvamenetelmät tarjoa enää riittävää suojaa. Tämän vuoksi uusia menetelmiä kehitetään jatkuvasti. Diffuusiokuvaukset ovat yksi lupaavimmista menetelmistä, ja sen soveltuvuutta HTTP-liikenteen poikkeavuuksien havaitsemiseen tutkittiin tässä työssä. Saadut tulokset osoittivat, että menetelmää käyttäen pystytään havaitsemaan sellaiset HTTP-kyselyt, jotka eroavat tyypillisestä liikenteestä. Menetelmä vaatii kuitenkin jatkokehittelyä, jotta se soveltuisi tietoturvahyökkäysten rutiininomaiseen tunnistamiseen.
English abstract: In recent years the popularity and usage of Web applications and services have grown rapidly and nowadays many of those services play an important role in our information society. Because of this, securing Web services is essential. The complexity of security attacks has renderd traditional security methods almost useless. Therefore new methods are being developed and diffusion maps are one of the most promising ones. In this thesis, the suitability of diffusion maps in evaluation of HTTP traffic is researched. The results show that the used methods are capable of detecting anomalous HTTP traffic from large data sets. Still, for active intrusion detection further research is needed.
Avainsanat: anomalia, diffuusiokuvaus, N-grammianalyysi, tietoturva, Web-palvelu
Keywords: anomaly, data security, diffusion maps, N-gram analysis, Web service

---

Kiitokset

Haluamme kiittää läheisiämme tuesta ja kannustuksesta tämän työn aikana. Kotoa saatu tuki on ollut korvaamatonta.

Lisäksi kiitämme Tuomo Sipolaa hänen kommenteistaan ja suuresta avusta diffuusiokuvausmenetelmän ymmärtämisessä. Hänen laatimaansa toteutus diffuusiokuvausmenetelmästä on ollut korvaamaton apu analyysiä tehdessä.

Lukuisat vapaat ohjelmistot ovat osoittautuneet korvaamattomiksi työkaluiksi tutkielmaa tehtäessä. Eritysmaininnan ansaitsevat vektorigrafiikkaohjelmisto Inkscape, tilastotieteellinen sovellus R sekä funktionaalinen ohjelmointiympäristö GHC. Mainitsematta ei myöskään voi jättää ladonnassa käytettyä LaTeX-ohjelmistoa, joka kryptisyydestään huolimatta on varsin mainio työkalu.

Lopuksi vielä kiitos Timo Hämäläiselle työmme ohjauksesta. Ohjaamisen lisäksi Timo auttoi meitä saamaan käyttöömme aineiston, joka oli peräisin aidosta tuotantoympäristöstä. Tällaisen aineiston saaminen olisi muutoin ollut lähes mahdotonta.

Jyväskylässä lokakuussa 2010

Joel Lehtonen ja Kristian Siljander

Lyhenteet

AJAX Asynchronous JavaScript And XML. Joukko Web-sovelluskehityksessä käytettyjä tekniikoita.

ARP Address Resolution Protocol. Ethernet-verkoissa käytettävä protokolla, jolla selvitetään IP-osoitetta vastaava MAC-osoite.

CSRF Cross Site Request Forgery. Web-sovelluksissa esiintyvä tietoturva-aukko, joka väärinkäyttää selaimen ja sivuston välistä luottamussuhdetta.

CSV Comma-Separated Values. Tiedostomuoto, jolla tallennetaan yksinkertaista taulukkodataa.

DDoS Distributed Denial of Service. Hajautettu palvelunestohyökkäys, jossa hyökkäys tapahtuu samanaikaisesti useasta eri kohteesta.

DOM Document Object Model. Ohjelmointirajapinta, joka mahdollistaa HTML- ja XHTML-dokumenttien sisällön muokkauksen.

DoS Denial of Service. Palvelunestohyökkäys, jolla pyritään lamauttamaan tietty verkkopalvelu.

HTML Hypertext Markup Language. Standardoitu kuvauskieli, jolla kuvataan hyperlinkkejä sisältävää tekstiä.

HTTP Hypertext Transfer Protocol. Selainten ja Web-palvelimien käyttämä tiedonsiirtoprotokolla.

ICMP Internet Control Message Protocol. TCP/IP-pinon kontrolliprotokolla, jolla lähetetään nopeasti viestejä koneelta toiselle.

IDS Intrusion Detection System. Järjestelmä, joka on ohjelmoitu tunnistamaan verkkoon suuntautuvat hyökkäysyritykset.

IIS Internet Information Services. Microsoftin kehittämä Web-palvelinalusta.

IP Internet Protocol. Verkkokerroksen protokolla, joka huolehtii tietoliikennepakettien välittämisestä pakettikytkentäisessä Internet-verkossa.

IPS Intrusion Prevention System. Järjestelmä, jolla pyritään ennakoivasti estämään tietomurtoyritykset.

IRC Internet Relay Chat. Pikaviestintäpalvelu, joka mahdollistaa reaaliaikaisen keskustelun Internet-käyttäjien välillä.

JSON JavaScript Object Notation. JavaScript-ohjelmissa käytetty yksinkertainen tiedonsiirtomuoto.

LDAP Lightweight Directory Access Protocol. Hakemistopalveluiden käyttöön tarkoitettu verkkoprotokolla.

MAC Media Access Control. Ethernet-verkossa verkkosovittimen yksilöivä osoite.

OSI Open Systems Interconnection Reference Model. Eri tiedonsiirtoprotokollista muodostuva kuvaus.

R2L Remote-to-Local. Hyökkäystyyppi, jossa hyökkääjä pyrkii saamaan koneelle laajemmat käyttöoikeudet, kuin hänellä muuten olisi.

SOA Service Oriented Architecture. Joustava arkkitehtuuriratkaisu, jolla pyritään helpottamaan eri palveluiden välistä kommunikointia.

SQL Structured Query Language. Standardoitu kyselykieli, jolla relaatiotietokantaan voi tehdä hakuja, muutoksia ja lisäyksiä.

TCP Transmission Control Protocol. Kuljetuskerroksen protokolla, joka rakentuu IP-protokollan päälle. TCP:tä käytetään lukuisissa sovelluskerroksen protokollissa, esimerkiksi HTTP:ssä.

U2R User-to-Root. Hyökkäys, jossa hyökkääjä pyrkii hankkimaan koneelle pääkäyttäjän oikeudet.

UDP User Datagram Protocol. Yhteyskäytäntö, jolla sovellus voi lähettää viestejä toiselle tietokoneelle.

VPN Virtual Private Network. Menetelmä, jolla useampi verkko voidaan yhdistää näennäisesti samaan yksityiseen verkkoon käyttäen julkista verkkoa.

XHR XMLHttpRequest. Ohjelmointirajapinta, jota käytetään esimerkiksi JavaScriptissä.

XHTML eXtensible Hypertext Markup Language. HTML:stä kehitetty kuvauskieli, joka täyttää XML:n muotovaatimukset.

XML eXtensible Markup Language. Merkintäkieli, jota käytetään sekä formaattina tiedonvälitykseen järjestelmien välillä että dokumenttien tallentamiseen.

XSS Cross Site Scripting. Web-sovelluksissa esiintyvä tietoturva-aukko, joka mahdollistaa käyttäjän koodin syöttämisen sovellukselle.

Xpath XML Path Language. XML-dokumenttien osien osoittamiseen tarkoitettu kieli.

Sisältö

Kuvat

1 Johdanto

Viime vuosina Web-sovellusten ja -palveluiden suosio on ollut voimakkaassa kasvussa, ja monet palveluista ovat nykyisin kriittisiä osia yhteiskuntamme toimivuuden kannalta. Internetin välityksellä käytettävien palveluiden tietoturvan ja saatavuuden takaaminen on tämän johdosta noussut monessa yrityksessä tärkeään rooliin. Tietojärjestelmien siirtyminen julkishallinnon ja yritysten erillisverkoista julkiseen Internetiin on vaikuttanut omalta osaltaan myös siihen, mihin tietoturvahyökkäykset nykyisin kohdistuvat ja kuinka ne pyritään toteuttamaan.

Hyökkäysten muuttuessa yhä hienostuneimmiksi, eivät perinteiset tietoturvamenetelmät enää riitä suojaamaan loppukäyttäjiä tai palvelun ylläpitäjiä. Tästä syystä erilaisten tietoturvaratkaisuiden ympärillä käy kova kuhina, ja aihepiiri on herättänyt suurta kiinnostusta tutkijoiden keskuudessa. Erilaisia ratkaisuja, joissa on pyritty selvittämään tietoturvahyökkäyksiä ja näiden mukana tulevia haasteita, on lukematon määrä. Haaste on löytää ne menetelmät, jotka oikeasti toimivat riittävällä tarkkuudella ja nopeudella.

Tietoturvahyökkäysten tunnistaminen perustuu siihen, että analysoimalla yhtä tai useampaa tapahtumaa pyritään löytämään viitteitä tapahtuvista hyökkäyksistä. Menetelmät jaetaan kahteen eri tyyppiin: anomalioiden eli poikkeavuuksien tunnistamiseen (engl. *anomaly detection*) ja väärinkäytösten tunnistamiseen (engl. *misuse detection*). Näistä anomalioiden tunnistaminen perustuu malleihin, joita luodaan järjestelmän, käyttäjän tai verkon normaalista käyttäytymisestä. Tulevaa liikennettä sitten verrataan näin luotuihin malleihin, jolloin opituista malleista poikkeava liikenne voidaan tunnistaa. Väärinkäytösten tunnistamiseen tarkoitetut järjestelmät puolestaan sisältävät joukon kuvauksia (engl. *signature*) tunnetuista hyökkäyksistä. Tuleva liikenne tarkistetaan näitä kuvauksia vastaan, jolloin näitä vastaavat hyökkäykset tunnistetaan. Hyökkäysten tunnistusjärjestelmät jaetaan joissakin tapauksissa myös sen mukaan, mistä tutkittava liikenne on kerätty. Tällöin järjestelmät on jaettu verkkoon pohjautuviksi (engl. *network based*) ja asiakaspohjaisiksi (engl. *host based*).

Tässä tutkielmassa tietoturvahyökkäykset pyritään tunnistamaan analysoimalla Web-palvelimien tuottamaa tapahtumalokia. Lokista yritetään etsiä ja tunnistaa

poikkeavuuksia normaalin liikenteen joukosta käyttäen diffuusiokuvauksia, joita käyttäen pystytään helpommin esittämään ja kuvaamaan sellaista materiaalia, joka koostuu suuresta määrästä parametreja.

Tutkielman rakenne on seuraavanlainen. Luvussa 2 esitellään yleisellä tasolla Web-palvelinalustoja, ja käydään läpi Web-palveluiden erilaisia toteutustapoja. Luku 3 käsittelee perinteisiä tietoturvahyökkäyksiä. Luvussa 4 perehdytään uudenlaisiin Web-ratkaisuihin ja tietoturvahyökkäyksiin, joiden kohteeksi Web-palvelut nykyisin joutuvat. Luku 5 sisältää kuvauksen dynaamisista Web-teknologioista, jotka mahdollistavat suurimman osan nykyisistä tietoturvahyökkäyksistä. Luku 6 sisältää katsauksen tutkimuskenttään liittyvään tutkimukseen.

Luvussa 7 kuvaillaan analyysissa käytetyt menetelmät. Luvussa 8 esitellään tutkimuksen lähtökohta ja analysoitavan materiaalin rakenne. Samassa luvussa myös käydään läpi esikäsittelijää, joka on on suunniteltu ja toteutettu tätä tutkimusta varten. Esimerkkien avulla esitellään myös sovelluksen toimintaa.

Luvussa 9 esitellään saatuja tuloksia kuvaajien avulla, ja analysoidaan näitä tarkemmin sekä esitetään mahdollisia syitä saatuihin tuloksiin. Jatkotutkimuksia ajatellen esitetään myös parannusehdotuksia. Viimeinen luku sitten sisältää yhteenvedon koko työstä ja ajatuksia sen onnistumisesta.

Siljander on vastannut pääasiassa tietoturvahyökkäystapojen ja Web-teknologioiden kartoittamisesta sekä perehtynyt aiheeseen liittyvään tutkimukseen. Lehtonen on toteuttanut tutkimuksessa käytetyn sovelluksen ja vastannut Web-palvelinratkaisujen esittelystä. Yhdessä tehtyä on tutkimuksessa käytetytyjen menetelmien selvittäminen ja tulosten analysointi. Työ on tehty näistä vastuualueista huolimatta tiiviissä yhteistyössä.

2 Web-palvelinratkaisut

Web-palveluilla tarkoitetaan järjestelmiä, jotka kommunikoivat keskenään käyttäen vakiintuneita Web-tekniikoita. Web-palvelut eivät ole riippuvaisia mistään tietystä laitteisto- tai käyttöjärjestelmäarkkitehtuurista. Web-palveluiden käyttämät teknologiat eivät sinänsä ole erityisen vallankumouksellisia, mutta tiedonvaihdon helpottuminen standardien protokollien ja Webin hajautetun rakenteen ansiosta on tehnyt Web-palveluista mielenkiintoisia niin kehittäjien kuin käyttäjienkin näkökulmasta [1].

IBM määrittelee Web-palvelut seuraavasti: *Web-palvelut ovat itsenäisiä ja modulaarisia sovelluksia, jotka voidaan julkistaa, määritellä, paikallistaa ja suorittaa verkon ylitse, yleensä Webin välityksellä* [2].

Tietoturvan kannalta Web-palvelut ovat haastavia, koska niitä käytetään Internetin välityksellä eivätkä ne ole rajoittuneita esimerkiksi tietyn organisaation lähiverkkoon. Lisäksi Web-palveluiden osia sijoitetaan usein fyysisesti eri paikkoihin ja ne kommunikoivat keskenään Internetin välityksellä.

Tässä luvussa käsitellään komponentteja, joista tämänhetkiset Web-palvelut koostuvat. Erityistä huomiota on kiinnitetty seikkoihin, jotka vaikuttavat palvelun tietoturvaan.

2.1 Palvelinohjelmistot

Web-palveluiden toteuttajan näkökulmasta lähin komponentti on palvelinohjelmisto. Se käsittelee HTTP-kyselyt ja lähettää vastauksina käyttäjän selaimessa näytettäviä ja prosessoitavia osia kuten hypertekstiä, kuvia ja selainohjelmassa suoritettavia komentosarjoja.

Web-palvelun käyttäjä ei kuitenkaan välttämättä kommunikoi suoraan Web-palvelimen kanssa. Suuret Web-palvelut koostuvat usein palvelinohjelmiston lisäksi välimuistipalvelusta, joka vähentää palvelimelle kohdistuvaa rasitusta säilyttämällä harvoin muuttuvaa sisältöä välimuistissa ja välittämällä tätä suoraan käyttäjille. Tällainen konfiguraatio esitellään tarkemmin luvussa 2.2.

Tietoturvapalveluita tarjoavan Netcraftin raportista huhtikuulta 2010 [3] käy il-

mi, että kaksi suurinta Web-palvelinalustan kehittäjää ovat *Apache Software Foundation* (53,93 % suosituimmista sivuista), *Microsoft* (24,97 %). Näiden jälkeen tulevat *Google* ja *nginx*-projekti kumpikin 6 % osuudella. Tuloksia voi vääristää jonkin verran se, että joissakin tapauksissa edustalla suoritetaan välitys- tai välimuistipalveluna toista palvelinohjelmistoa esimerkiksi Apachea tai nginx:ä, joka välittää kyselyt eteenpäin varsinaiselle Web-palvelinohjelmistolle. Tämä selittää esimerkiksi sen, miksi Zope puuttuu kokonaan suosituimpien palvelinten joukosta. Esimerkiksi Jyväskylän yliopiston palvelinohjelmiston nimenä Netcraftin tilastoissa näkyy Apache [4], vaikka palvelu on toteutettu Zope-palvelinohjelmiston päällä suoritettavalla Plonella. Samoin on käynyt Tomcat-palvelimella ajettavalle Korppi-opintotietojärjestelmälle.

Palvelinohjelmiston suosiolla voidaan olettaa olevan vaikutusta siihen, kuinka paljon mielenkiintoa ohjelmistoa kohtaan on tietomurtoja tekevien kräkkereiden keskuudessa. Toisaalta suuren suosion saaneita Web-palvelinalustoja kohtaan on myös positiivista kiinnostusta tietoturvapiireissä, jolloin etenkin vapaan lähdekoodin palvelinalustojen tietoturvan taso edistynee suosion kasvaessa.

2.1.1 Apache

Apache HTTP Server on Apache Software Foundationin kehittämä vapaan lähdekoodin Web-palvelinalusta. Apache pohjautuu NCSA:ssa kirjoitettuun httpd-ohjelmistoon. Ensimmäinen versio Apache-palvelimesta julkaistiin keväällä 1995 ja alle vuodessa se saavutti suosiossa edeltäjänsä NCSA httpd:n. Tästä lähtien Apache on pitänyt ykköspaikkaa maailman suosituimpana Web-palvelinalustana [5].

Suosituimpana palvelinalustana Apache on luonnollisesti yleinen kohde myös tietomurtojen yrityksille. Apachen tietoturva-aukkoja hyödyntämällä on kuitenkin harvoin onnistuttu saamaan laajamittaista vahinkoa. Suurimmassa osassa tietomurtoja keskitytäänkin murtamaan varsinaista Web-palvelua palvelinalustan sijaan.

Apache pystyy suorittamaan komentosarjoja perinteisen CGI-rajapinnan lisäksi myös palvelinlaajennosten avulla, jolloin saavutetaan tiiviimpi integraatio ja enemmän suorituskykyä verrattuna CGI-rajapintaan [6]. Yksi yleisimmistä palvelinlaajennoksista on PHP-kielen tuki.

Apachea voidaan suorittaa lukuisissa eri käyttöjärjestelmissä mukaan lukien Linux ja Windows. Suosittuja Apache-alustalla käytettäviä ohjelmointikieliä ovat PHP ja Python. Tunnettuja tässä ympäristössä käytettäviä Web-palveluita ovat esimerkiksi Wikipedian käyttämä MediaWiki ja lukuisilla selainkäyttöisillä keskustelu-

palstoilla käytössä oleva phpBB. Palvelinympäristöä, jossa on käytössä Linux, Apache, MySQL ja PHP, kutsutaan usein nimellä LAMP.

Tietoturvan näkökulmasta Apachen vahvuutena on sen tiivis integraatio Linux-jakeluihin. Mikäli vakavia tietoturvaan liittyviä haavoittuvaisuuksia paljastuu, ohjelmisto on vaivatonta ja nopeaa päivittää uudempaan versioon. Tällöin palvelinohjelmistoa vastaan kehitetyt murtautumiskomentosarjat eivät pysy pitkiä aikoja toimintakykyisinä. Tämä on luultavasti syynä siihen, miksi Apache on säilyttänyt ykköspaikkansa Web-palveluntarjoajien ja kehittäjien suosiossa mitattuna.

2.1.2 Microsoft IIS

Apachen jälkeen toiseksi suurinta markkinaosuutta Web-palvelinalustoista pitää Microsoftin kehittämä Internet Information Services (lyh. *IIS*). Ensimmäisen versio sovelluksesta julkaistiin Windows NT-käyttöjärjestelmälle vuona 1996, ja vuonna 2008 julkaistiin IIS 7 Windows Server 2008 palvelinjakelun yhteydessä. Uusin virallinen versio sovelluksesta on tällä hetkellä IIS 7.5 ja siitä on ladattavissa 180-päivän ilmainen kokeiluversio Windows Server 2008 alustalle Microsoftin sivuilta [7].

Aikaisemmista IIS-versioista on löydetty useita eri tasoisia tietoturvariskejä joista tunnetuin on Code Red Wormiksi nimetty hyökkäys, joka saastutti yli kolmesataatuhatta Web-sivustoa. Kyseinen hyökkäys pohjautui puskuriylivuotoon, johon Microsoft oli julkaissut päivityksen kuukausi takaperin, mutta jota osa palveluita ylläpitävistä tahoista ei ollut asentanut. Tämän hyökkäyksen johdosta yleinen käsitys IIS-sovellusten tietoturvasta on vähintäänkin kyseenalainen. Uusimmista versioista ei ole kuitenkaan löydetty vastaavanlaisia tietoturvariskejä, ja Secunian ylläpitämän listan mukaan uusimmasta versiosta ei löydy yhtään paikkaamatonta tietoturvariskiä [8].

Suurin muutos uusimmissa versioissa verrattaessa vanhoihin on siirtyminen modulaariseen arkkitehtuuriin. Tämä mahdollistaa uusien komponenttien nopean lisäämisen ja poistamisen, jonka lisäksi monipuolinen API-rajapinta tarjoaa mahdollisuuden tehdä yksilöllisiä komponentteja omiin tarpeisiin. Modulaarinen rakenne parantaa myös sovelluksen tietoturvaa pienentämällä asennettavien osien määrää ainoastaan niihin, joita ylläpitäjä tarvitsee. Edut modulaarisuudesta vastaavat paljolti niitä etuja, jotka saavutettiin Apache-palvelimen version 1.3 yhteydessä.

Tietoturvaan ja skaalautuvuuteen on muutenkin kiinnitetty enemmän huomiota alusta asti, joiden lisäksi monet Apachen ominaisuuksista kuten URL-osoitteiden uudelleenkirjoitus on otettu käyttöön [7]. IIS:n suurimpana haittapuolena nykyisin

on sen sidonnaisuus Windows-maailmaan, sillä muille alustoille sitä ei ole saatavilla. Rajoituksistaan huolimatta IIS on nykyisin oikeanlaisessa ympäristössä varteenotettava vaihtoehto Apachelle.

2.1.3 Zope

Vaikka Apachen ja Microsoft IIS:n markkinaosuudet kattavat suurimman osan käytetyistä Web-alustoista, on markkinoilla suuri määrä pienempiä ratkaisuja, joille löytyy oma käyttäjäkuntansa. Yksi näistä on vapaaseen lähdekoodiin pohjautuva Zope [9]. Zope tulee sanoista *Z Object Publishing Environment* ja se on kirjoitettu pääosin käyttäen Pythonia. Zope on ensimmäinen Web-sovellusten suunnitteluun tarkoitettu oliopohjainen julkaisujärjestelmä ja sen mukana tulee oma tietokantasovellus ja Web-palvelinalusta. Zopesta on olemassa useita eri jakeluita joista käytetyin on Zope2.

Vaikka Zope itsessään sisältää Web-palvelimen, voidaan sitä käyttää myös rinnakkain muiden palvelimien kanssa. Tällaiseen ratkaisuun voidaan päätyä, jos halutaan esimerkiksi ylläpitää samalla palvelimella sekä Zopella että muilla työkaluilla tuotettua sisältöä, tai jos halutaan käyttää palveluita kuten Apachen tarjoamaa SSL-salausta. Monissa tehtävissä muut alustat toimivat myös nopeammin ja turvallisemmin kuin Zope. Zopen käyttö ei myöskään sulje perinteisen relaatiotietokannan käyttöä ja sen tukemiin tietokantoihin kuuluu muun muassa MySQL, Oracle ja PostgreSQL. Usein eri tietokantoja käytetään myös rinnakkain, jolloin Zopen objektit voidaan tallentaa ZODB-oliotietokantaan ja muu data perinteisiin relaatiotietokantoihin.

Zopen yksi suurimmista vahvuuksista on sille tehdyt lisäosat, joita löytyy runsaasti. Näistä tunnetuin on Plone [10], joka on pitkälle kehitetty sisällönhallintajärjestelmä. Sitä voidaan käyttää kaikenlaisen Web-sisällön kuten blogien ja sisäisten ja ulkoisten Web-sivustojen hallitsemiseen. Se on helppo asentaa ja ottaa käyttöön, jonka lisäksi se on käännetty yli 40 kielelle. Helppokäyttöisyys, joustavuus ja laajennettavuus ovat niitä tekijöitä, jotka ovat tehneet Plonesta suositun. Plonelle löytyvä ohjeistus on myös hyvin kattavaa ja se on yksi tuetuimmista avoimen lähdekoodin projekteista. Vapaata Web-sovellusalustaa etsivälle Zope ja Plone tarjoavatkin erinomaiset työkalut.

2.2 Käänteinen välityspalvelin

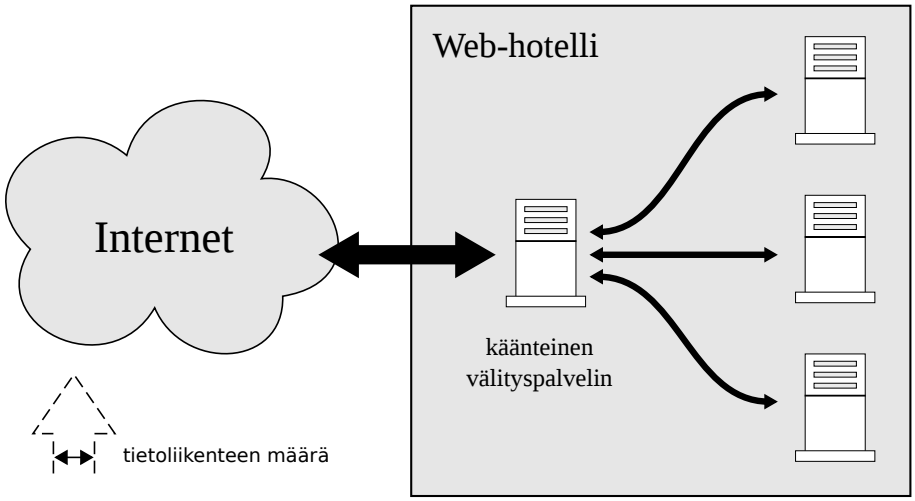
Suositut Web-palvelut ovat usein rakenteeltaan hajautettuja, jolloin palvelimeen kohdistuva kuormitus saadaan jaettua useammalle laitteelle. Tämä vähentää palvelujen kustannuksia, koska tällöin ei tarvita yhtä tehokasta laitetta, vaan palvelu voidaan toteuttaa useammalla pienempitehoisella palvelimella. Kustannussäästöjen lisäksi hajautettu rakenne on vikasietoisempi, koska yhden palvelimen vikaantuessa käyttäjät voidaan automaattisesti ohjata muille palvelimille.

Käytettäessä hajautettua Web-palvelua tarvitaan kuormantasaaja, joka ohjaa kyselyt sellaisenaan toisille palvelimille joko kiertovuorottelulla (engl. *round robin*) tai ohjaamalla sisään tuleva kysely aina sellaiselle palvelimelle, jonka kuormitus on alhaisin. Tällaista järjestelmää kutsutaan käänteiseksi välityspalvelimeksi (engl. *reverse proxy*). Kuvassa 2.1 havainnollistetaan kuormantasauksen toimintaa.

Välimuistina toimiessaan palvelimelle tallennetaan muuttumaton sisältö, jolloin tällaisia kyselyitä ei tarvitse välittää varsinaisille Web-palvelimelle saakka, koska ne löytyvät välityspalvelimen muistista. Ainoastaan muuttuva tai välimuistille uusi sisältö pyydetään palvelinohjelmistolta. Tällöin palvelinohjelmiston suorituskyky ei ole niin keskeisessä roolissa Web-palvelun suorituskyvyssä.

Kun käytetään käänteistä välityspalvelinta, varsinaiset Web-palvelimet voidaan sijoittaa palomuurin taakse, koska näihin laitteisiin ei käyttäjä ole suoraan yhteydessä. Välityspalvelimeen voidaan myös sijoittaa ohjelmistopohjainen palomuuri, joka suodattaa yleisimmät tietomurtoyritykset. Tällöin nämä kyselyt eivät pääse kuormittamaan tai haittaamaan muutoin palvelinten toimintaa.

Käänteisellä välityspalvelimella voidaan muillakin tavoin lisätä palvelinten suorituskykyä ja luotettavuutta. Välityspalvelin voi esimerkiksi ottaa vastuulleen tiedon salaamisen tai toimia välimuistina. Käänteinen välityspalvelin tarjoaa usein myös riittävät työkalut kuormituksen hallintaan. Järjestelmä riittääkin hyvin sellaisten Web-palveluiden tarpeisiin, joiden yleisö on pienellä maantieteellisellä alueella.



Kuva 2.1: Käänteisen välityspalvelimen toiminta.

2.3 Pilvivälimuisti

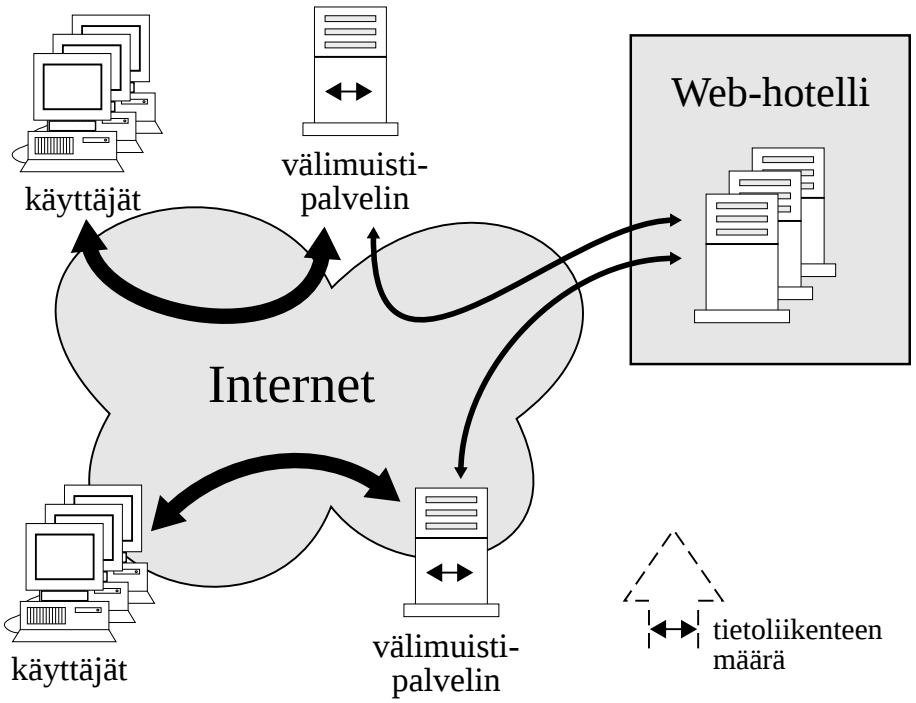
Edellä kuvattu käänteinen välityspalvelin ei ole riittävä ratkaisu kuormituksen tasaamiseen kun kyse on laajamittaisesta, globaalin mittakaavan Web-palveluntarjonnasta. Käänteinen välityspalvelin vähentää Web-palvelinten kuormitusta, mutta ei vähennä eikä hajauta palvelun verkkoliikennettä. Tietoliikenne kohdistuu kuitenkin yhteen pisteeseen verkkoa. Jotta maailmanlaajuiset palvelut voitaisiin toteuttaa ilman maakohtaisia erillissivustoja, tarvitaan keinoja kuormituksen jakamiseen verkkotopologisesti edullisemmin.

Eräs ratkaisu ongelmaan on sijoittaa välimuistipalvelimia eri maantieteellisille alueille. Nämä palvelimet säilövät välimuistiinsa harvoin muuttuvia sivuja ja muita staattisia resursseja. Sivulle tulevat kyselyt voidaan ohjata palveluntarjoajan näkökulmasta edullisimmille palvelimille esimerkiksi aluekohtaisten nimipalveluiden avulla [11]. Kuvassa 2.2 havainnollistetaan järjestelmän toimintaa.

Käytännössä tällaisen järjestelmän rakentaminen ja ylläpito on kallista, koska se vaatii liiketoimintaa useiden valtioiden alueilla ja lukuisien palveluntarjoajien kanssa. Ratkaisuna tähän ongelmaan on kehitetty pilvivälimuistipalveluita. Yksi suosituimmista tällaisista palveluista on Akamai Technologiesin *Web Application Accelerator* [12]. Järjestelmä koostui kesäkuussa 2010 yhteensä 65 000 palvelimesta, jotka sijaitsevat 70 eri valtion alueella. Käyttäjän tekemät kyselyt ohjataan automaattisesti sopivimpaan Akamain palvelimeen. Sopivin palvelin valitaan yhteyden luotettavuuden ja nopeuden perusteella. Palvelu voi noutaa ennakolta kaiken sivulla sijaitsevan staattisen sisällön, jolloin näiden osien välittäminen ei kuormita varsinaisia

Web-palvelimia lainkaan.

Pilvivälimuistin haittapuolena on se, että Web-palveluntarjoaja ei näe suoraan sivustolle tulevia kyselyitä, koska monet kyselyt eivät etene välimuistipalvelua syvemmälle. Tällöin tilastointi ja erilaisten tietomurtojen tunnistaminen on vaikeampaa. Monet välimuistipalveluntarjoajat, Akamai mukaan lukien, tarjoavat kuitenkin asiakkailleen monipuolisia tilastointipalveluita.



Kuva 2.2: Pilvivälimuistin toiminta.

3 Perinteiset tietoturvahyökkäykset

Tietoyhteiskunnassa yksikään verkossa oleva laite ei ole suojassa tietoturvahyökkäyksiltä ja niiden mukana tulevilta ongelmilta. Hyökkäykset ovat nykyisin myös hyvin monimuotoisia ja ne jaetaan eri kategorioihin niiden tavoitteiden ja toteutustapojen mukaan. Uusia hyökkäystapoja kehitetään jatkuvasti lisää ja tunnetut hyökkäystavat saavat uusia ominaisuuksia. Vaikka suurimpaan osaan hyökkäyksistä on olemassa erilaisia suojautumistapoja, on näiltä kaikilta suojautuminen ylläpitäjän kannalta mahdoton tehtävä. Verkon turvallisuuden kannalta on kuitenkin tärkeää, että hyökkäysten riskit tiedostetaan ja uhkien toteutumisen todennäköisyyksiä pienennetään mahdollisuuksien ja resurssien mukaan.

3.1 Tiedon urkinta ja väärentäminen

Ennen kuin hyökkääjä pystyy murtautumaan verkkoon tai verkossa sijaitsevaan tietokoneeseen, tulee hänen selvittää järjestelmän mahdolliset heikkoudet. Tällaisia tietoja ovat esimerkiksi asennetut käyttöjärjestelmät, ohjelmistojen versiotiedot sekä verkon yleinen rakenne ja tietoturvataso. Näiden tietojen selvittämiseksi hyökkääjä aloittaa verkkoa kohtaan joukon hyökkäyksiä, joita kutsutaan tiedusteluhyökkäyksiksi (engl. *Probe Attacks*). Tällaiset tiedustelut ovat varsinaisia hyökkäyksiä yleisempiä, sillä niiden toteuttaminen ei vaadi hyökkääjältä kovinkaan syvällistä osaamista. Verkosta löytyy paljon valmiita työkaluja, joiden avulla hyökkääjä pystyy urkkimaan tarvittavat tiedot ja mahdollisesti jo saman tien hyödyntämään havaittuja heikkouksia. Hyökkääjä, jolla on tietämys verkossa käytettävistä laitteista ja palveluista, voi hyödyntää näitä tietoja haavoittuvuuksien etsimiseen ja järjestelmiin murtautumiseen [13].

Kun hyökkääjällä on riittävä tietämys verkon eri komponenteista, voi hän aloittaa hyökkäyksen suunnittelun. Yksi tapa murtautua tietoverkkoon on hyödyntää verkkokerroksen protokollissa, kuten TCP ja IP, olevia heikkouksia. Tällaisia hyökkäyksiä kutsutaan *spoofing*- eli *petkutushyökkäyksiksi*. Niissä murtautuja pyrkii naamioimaan oman koneensa siten, että se näyttää kuuluvan kohteena olevan laitteen kanssa samaan lähiverkkoon. Tällöin hyökkääjä pystyy helpommin huijaamaan koh-

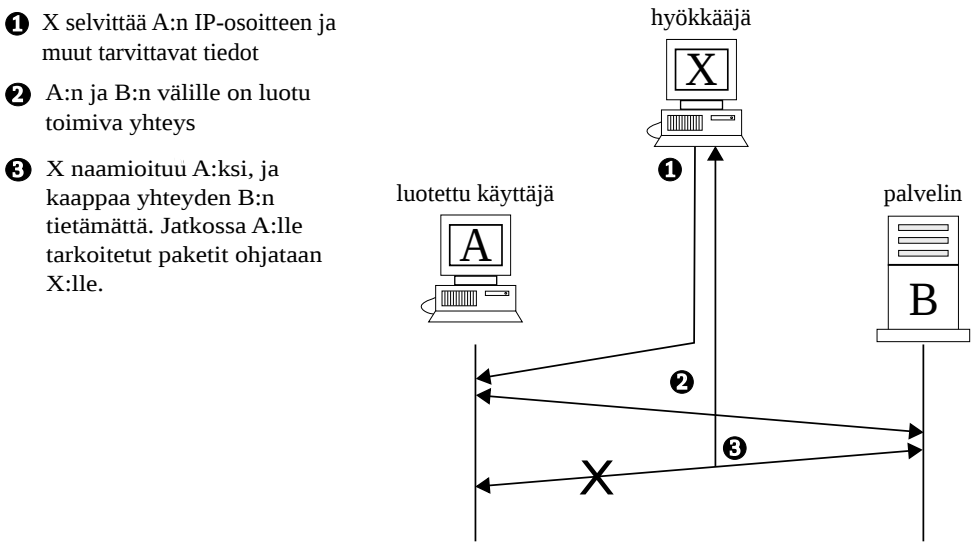
dekonetta jakamaan ja lähettämään arkaluontoista tietoa, jota jaettaisiin muuten ainoastaan luotettujen osapuolien kesken. Tällaiset hyökkäykset jaotellaan edelleen sokeisiin ja aktiivisiin hyökkäyksiin sen mukaan, kuinka paljon hyökkääjällä on tietoa verkon rakenteesta. Sokeassa hyökkäyksessä kohdekoneen lähettämiä vastauksia ei pystytä seuraamaan, koska murtautujan oikeudet ovat puutteellisia ja IP-osoite, jonka haltijaksi hyökkääjä haluaa naamioitua, on harvoin tiedossa. Aktiivisessa hyökkäyksessä murtautujalla on käytettävissään tietoa tietokoneiden välisistä käyttöoikeuksista, jolloin tiedon korruptoiminen, muokkaaminen ja välittäminen toiseen verkkoon, on helpompi toteuttaa [14].

3.1.1 IP Spoofing

IP Spoofing on hyökkäys, jossa pyritään murtautumaan verkkoon ja hankkimaan arkaluontoista tietoa väärentämällä lähetettäviin paketteihin luotetun koneen IP-osoite. Hyökkäykset jaotellaan sokeisiin ja aktiivisiin hyökkäyksiin sen mukaan, mistä hyökkäys toteutetaan. Hyökkäykset toimivat siten, että kolmivaiheisen yhteydenmuodostuksen aikana hyökkääjä tekeytyy tietokoneeksi, johon hyökkäyksen kohteella on luottamussuhde. Tämä tapahtuu muokkaamalla lähetettävien pakettien otsaketietoja ja kuittausnumeroita. Tässä onnistuakseen hyökkääjän tulee pystyä vastaamaan oikeilla kuittausnumeroilla kohteen lähettämiin paketteihin. Nämä kuittausnumerot hyökkääjä joutuu aluksi usein arvaamaan, jonka vuoksi tämä hyökkäystapa on usein hankala toteuttaa [14]. Niin kutsuttu Man in The Middle -hyökkäys, jossa hyökkääjä kaappaa kahden koneen välisen yhteyden, perustuu myös IP-osoitteen väärinkäyttöön. Kuva 3.1 havainnollistaa hyökkäyksen toteuttamista.

Vaikka IP-osoitteen väärentämiseen perustuvat hyökkäykset ovat vaikeita toteuttaa, ovat ne melko yleisiä, sillä jokainen TCP/IP-protokollaa käyttävä järjestelmä on niille alttiina. Tämä johtuu siitä, että TCP/IP-verkossa on sallittua muokata paketteja tiedonsiirron aikana. Näin on, koska eräät IP-pohjaiset palvelut edellyttävät pakettien sisällön muuttamista lähettämisen yhteydessä. Tällaisia palveluita ovat esimerkiksi mobiili-IP- ja VPN-järjestelmät [15].

IP Spoofing -hyökkäyksiä voidaan torjua monin eri keinoin. Tehokkain tapa on määritellä verkon reitittimet estämään sellaisten pakettien kulku, jotka on merkitty lähetetyiksi sisäverkon laitteilta, mutta jotka todellisuudessa saapuvat reitittimeen sen ulkopuolisista liitännöistä. Jos tämä kuitenkin halutaan sallia, niin jokainen sessio tulisi salata [14].



Kuva 3.1: Man in The Middle-hyökkäys.

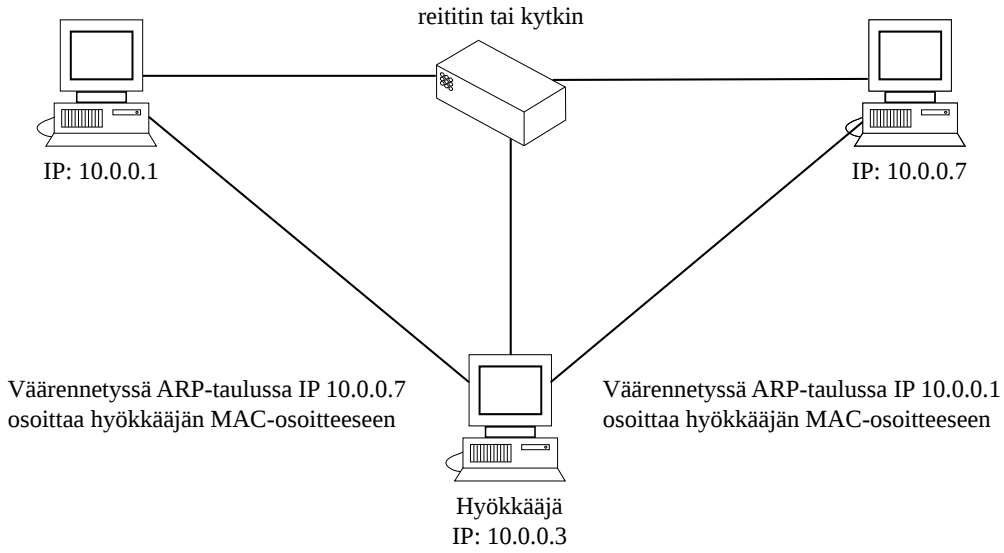
3.1.2 ARP Spoofing

Nykyisin lähes jokainen verkko pohjautuu TCP/IP-protokollan ja Ethernet-verkon väliseen yhteistoimintaan. Tätä varten tarvitaan ARP-protokollaa, jonka tehtävänä on selvittää IP-osoitteita vastaavat Ethernet- eli MAC-osoitteet. Ethernet-osoite on verkkokorttikohtainen ja sen avulla yksilöidään lähiverkossa sijaitsevat tietokoneet ja muut aktiivilaitteet. Sitä tarvitaan IP-protokollalla liikennöitäessä, koska ilman tätä tietoa lähiverkossa sijaitseva reititin ei pysty välittämään Internetistä saapuvia paketteja eteenpäin oikealle laitteelle eikä myöskään lähiverkossa sijaitseva tietokone pysty siirtämään tietoa IP-protokollalla muille lähiverkossa sijaitseville laitteille.

ARP Spoofing -hyökkäyksessä, jota kutsutaan myös ARP-myrkyttämiseksi, pyritään syöttämään reitittimien ARP-tauluihin virheellistä tietoa (kuva 3.2). Tämä tapahtuu kaappaamalla verkon yleislähetysviestejä, ja muokkaamalla kuittausviestien sisältöä siten, että kuittausviesteihin hyökkääjä laittaa kohdeosoitteeksi oman IP-osoitteensa. Verkon aktiivilaitteilta saapuviin, muille tarkoitettuihin kyselyihin hyökkääjä vastaa omalla MAC-osoitteella. Tämän jälkeen kohdelaitteelle tarkoitetut paketit ohjautuvat hyökkääjän koneelle [14].

ARP-protokolla on hyvin haavoittuvainen hyökkäyksille, koska oletuksena se ei sisällä minkäänlaista suojautumiskeinoa ARP-myrkyttämiselle. Siksi paras keino suojautua tällaisilta hyökkäyksiltä on varmistaa, että ennen ARP-taulun muokkausta tunnistetaan käyttäjät jollakin keinolla. Toinen mahdollisuus on käyttää verkkolaitteissa staattisia ARP-tauluja [14]. Jakamalla lähiverkko pienempiin osiin virtuaa-

listen lähiverkkojen (VLAN) avulla voidaan myös rajata ARP-myrkyttämisen vaikutuksia.



Kuva 3.2: ARP-taulun myrkyttäminen käyttäen Man In the Middle -hyökkäystä.

3.2 Denial Of Service

CERT [16], joka on yksi tietoturva-alan johtavista tutkimuskeskuksista, määrittelee palvelunestohyökkäyksen (engl. *Denial of Service*, DoS) sellaiseksi teoksi, jossa hyökkääjän tavoitteena on estää palvelun käyttäjiä käyttämästä heille kuuluvia tai heidän käytettävissään olevia palveluita. Tällaisia palveluita ovat esimerkiksi tietoliikenneresurssit tai verkossa käytettävät Web-sovellukset. Hyökkäyksillä pyritään lamauttamaan palvelua tarjoava verkko tai palvelin siten, että asiaankuuluvien pyyntöjen vastaanottamiseen tai käsittelyyn ei riitä resursseja. Keinoja hyökkäysten toteuttamiseen on monia. Koska valtaosa palvelunestohyökkäyksistä käyttää hyväkseen eri protokollien sallittuja toimintoja, on hyökkäyksiltä suojautuminen hyvin vaikeaa [17].

Palvelunestohyökkäykset voidaan jakaa kolmeen ryhmään näiden toteutustapojen ja tavoitteiden mukaan. Ensimmäiseen ryhmään kuuluvat hyökkäykset, joiden tarkoituksena on kuormittaa palvelimia ja kuluttaa loppuun niiden rajoitetut resurssit. Tämä voidaan toteuttaa esimerkiksi kuormittamalla verkkoa tai palvelinta turhilla pyynnöillä. Toiseen ryhmään kuuluvat hyökkäykset, jotka pyrkivät joko tuhoamaan tai muuttamaan konfigurointitietoja siten, että kone tai verkko lakkaa

toimimasta. Viimeiseen ryhmään kuuluvat verkon komponenttien muokkaamiseen tai tuhoamiseen tähtäävät hyökkäykset [16]. Vaikka seuraavaksi keskitytään vain resursseihin kohdistuviin hyökkäyksiin, niin palvelun kokonaisturvallisuuden kannalta ylläpitäjän tulee kiinnittää jokaiseen ryhmään huomiota.

Viime vuosina palvelunestohyökkäysten kirjo on voimakkaasti laajentunut. Alkuperäisten, pelkästään verkkokerroksen raa'an voiman hyökkäysten lisäksi, on alkanut esiintyä sovelluskerroksen hyökkäyksiä, joiden toteuttaminen vaatii hyökkääjältä usein varsin vähän omia resursseja [17]. Nämä sovelluskerroksen hyökkäykset ovat toteutukseltaan hyvin hienostuneita, jolloin ne jäävät yleensä huomaamatta tavanomaisilta tietoturvajärjestelmiltä, koska tietoliikenteen sisältö ei juurikaan poikkea normaalitilanteesta. Hyökkääjä saattaa esimerkiksi pyytää sellaista resurssia palvelulta, jonka kysely vie vain vähän hyökkääjän omia resursseja, mutta joka aiheuttaa palvelimelle suuren kuormituksen [18].

Sovelluskerroksen hyökkäysten teho on nähty esimerkiksi vuonna 2004, kun MyDoom-viruksella saastuneet koneet kuormittivat yleisimpiä hakukoneita etsimällä näiden avulla uusia sähköpostiosoitteita, joihin lähettää saastunut sähköposti [17]. Sovelluskerroksen hyökkäyksen vaikutus voidaan kohdistaa myös yksittäisiä palveluita kohti, jolloin vaikutukset ovat vieläkin suuremmat. Näin tapahtui vuonna 2003, kun Yhdysvalloissa yrittäjä palkkasi niin sanotun DDoS-mafian kaatamaan kilpailijoidensa verkkosivut HTTP-kyselyillä, jotka pyysivät ladattavaksi isoa kuvatiedostoa [18]. Tämä aiheutti kolmelle kilpailijalle arviolta miljoonan dollarin tappiot ja pysäytti näiden liiketoiminnan lähes kahdeksi viikoksi [19].

3.2.1 SYN-hukuttaminen

TCP/IP-protokollan yksi suunnittelulähtökodista oli, että sitä käytettäisiin avoimessa ja luotetussa ympäristössä. Tästä syystä sen suunnittelussa ei osattu ottaa huomioon mahdollisia vihamielisiä käyttäjiä, jotka pyrkisivät häiritsemään muiden käyttäjien tietoliikennettä. TCP/IP-protokollan käyttö julkisissa verkoissa kuitenkin yleistyi arvaamattomasti, jonka vuoksi suunnitteluvaiheessa tehdyt virheet periytyivät nyt käytössä olevaan IPv4-verkkojen rakenteisiin [17].

Yhteydenmuodostus TCP-protokollalla tapahtuu kolmivaiheisesti. Aluksi yhdistävä tietokone lähettää palvelimelle SYN-paketin, jonka seurauksena palvelin varaa tulevalle yhteydelle resursseja ja lähettää yhdistävälle tietokoneelle SYN/ACK-paketin. Tämän jälkeen palvelin jää odottamaan yhteyden muodostamista. Tähän yhdistävän tietokoneen tulee vastata ACK-paketilla, jonka jälkeen yhteys on muo-

dostunut ja tiedonsiirto voi alkaa [17]. Tunnetuin TCP:tä käyttävä protokolla on Internetissä käytetty HTTP. Sen avulla Web-palvelimet pystyvät nopeasti palvelemaan useita yhtäaikaisia käyttäjiä.

TCP/IP-protokollan perityistä heikkouksista tunnetuin ja käytetyin on SYN-hukuttamiseksi kutsuttu hyökkäys, jossa hyökkääjä pyrkii kuluttamaan kohteen tietoliikennekapasiteetin loppuun tekaistuilla yhteydenmuodostuspyynnöillä. Hyökkäys on hyvin yksinkertainen ja helppo toteuttaa, sillä se käyttää hyväkseen TCP-protokollaan määritettyjä toimintoja.

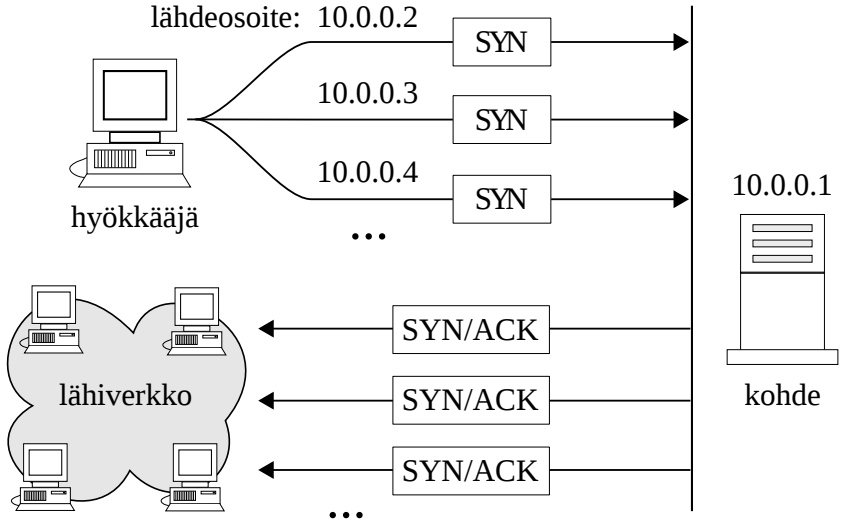
Hyökkäyksessä TCP-protokollan kolmiosaista yhteydenmuodostusta ei viedä loppuun saakka vaan jätetään palvelin odottamaan ACK-paketti yhdistävältä tietokoneelta. Koska TCP-protokolla pyrkii aina varmistamaan yhteyden muodostumisen, lähetetään SYN/ACK-paketti tarvittaessa uudestaan, kunnes yhdistävä tietokone vastaa ACK-paketilla tai määritelty aikaraja tulee vastaan.

Tätä ominaisuutta hyökkääjä pystyy käyttämään hyväkseen. Riittää, että hyökkääjä nuuskii selville käyttämättömän IP-osoitteen, joka on vielä mieluiten samasta aliverkosta kuin missä palvelin sijaitsee. Tämän jälkeen hyökkääjä luo SYN-paketin, jossa on tämä tekaistu IP-osoite. Koska palvelimen lähettämä SYN/ACK-viesti ei koskaan saavu oikealla koneelle, ei palvelin saa ACK-kuittausta. Tällöin TCP-protokolla alkaa lähettämään pakettia uudestaan niin kauan kunnes määritetty raja yhteyden aikakatkaisulle tulee vastaan [20].

Hyökkääjän on vaivatonta automatisoida nämä vaiheet ja hyökkäys voidaan toteuttaa esimerkiksi saastuneista koneista muodostetun verkoston avulla. Näin saadaan aikaiseksi kuvan 3.3 mukainen tilanne, jossa useita IP-osoitteita käyttämällä hyökkääjä häiritsee kohteen toimintaa turhilla yhteydenottopyynnöillä.

SYN-hukuttamisen mahdollistavan mekanismin avulla voidaan myös toteuttaa heijastettu hyökkäys (engl. *reflective attack*), joka on muunnos SYN-hukuttamisesta. Tässä hyökkäyksessä väärennetyissä SYN-viesteissä on lähettäjäksi merkitty haluttu kohde. Lähettämällä suuri määrä näitä SYN-viestejä esimerkiksi Web-palvelimelle, aiheutuu vastaustulvasta ongelmia kohteelle [20].

Koska ylimääräistä palvelinkapasiteettia ei useinkaan ole järkevää pitää SYN-hukuttamisen varalta, joudutaan ratkaisua hakemaan muualta. Alkeellisin keino on rajoittaa puoliavonaisten yhteyksien määrää, jolloin rajan ylittyessä yhteyksiä aletaan pudottamaan [21]. Toinen käytetty keino on reitittimiltä verkkoon päin tulevan liikennemäärän seuraaminen ja liikennepiikkeihin reagoiminen. Hyökkäyksiä vastaan voidaan myös suojautua liikenteen seuraamiseen tarkoitetuilla sovelluksil-



Kuva 3.3: SYN-hyökkäys käyttäen useita lähiverkon IP-osoitteita.

la sekä pääsylistoilla [20].

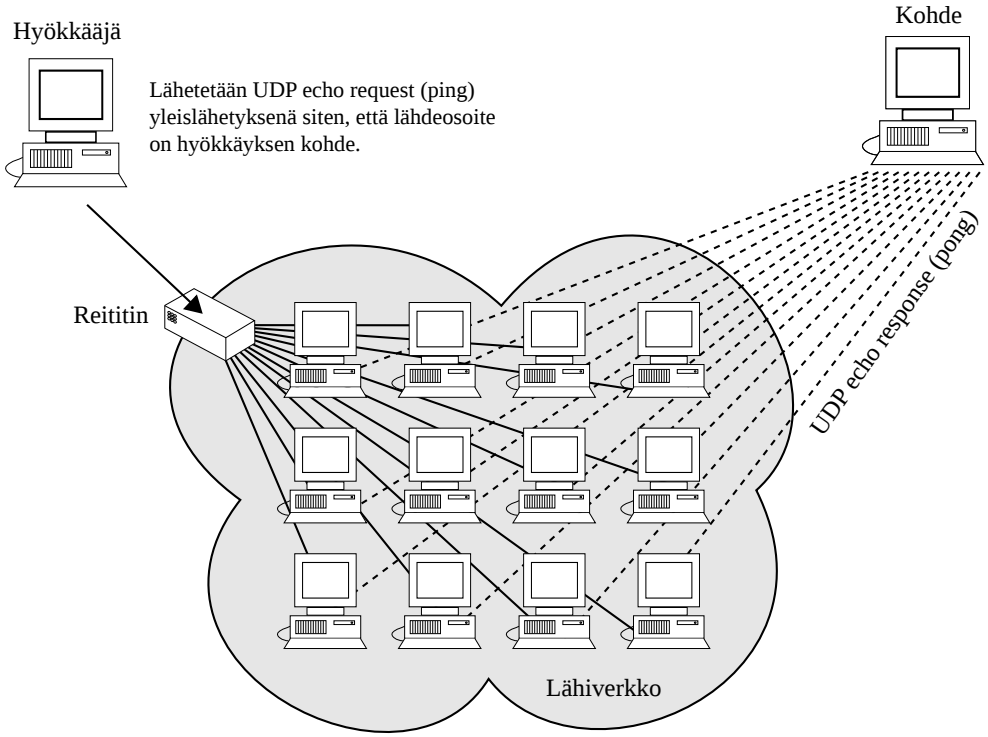
Kehittyneempi suojautumiskeino on muuttaa TCP-protokollan kättelyä siten, että yhteyden tarvitsemat resurssit varataan vasta ACK-viestin vastaanottamisen yhteydessä. Tätä menetelmää kutsutaan SYN-evästykseksi (engl. *SYN Cookies*) ja se on tuettu muun muassa Linuxissa ja FreeBSD:ssä [22]. Yhteydenmuodostuksessa tarvittavat parametrit kätketään tällöin SYN-paketin järjestysnumeroon. Tämän menetelmän käyttö kuitenkin rajaa pois runsaasti TCP-protokollan ominaisuuksista, jonka vuoksi SYN-evästys kytketään yleensä päälle vain hyökkäysten ajaksi. Täydellistä ratkaisua SYN-hukuttamista vastaan ei mikään näistä tavoista kuitenkaan tarjoa, sillä huolellisesti toteutettua hyökkäystä on vaikea torjua.

3.2.2 UDP Echo

Myös UDP-protokollan käyttö mahdollistaa hyökkäysten tekemisen. Nämä hyökkäykset käyttävät hyväkseen UDP Echo -protokollaa, mikäli sen käyttö on sallittu verkossa. Näistä hyökkäyksistä tunnetuin on Fraggleksi nimetty hyökkäys, joka nykyisinkin aiheuttaa väärin konfiguroidussa verkossa isoja ongelmia. Fraggle toimii siten, että hyökkääjä lähettää UDP Echo -viestin yleislähetyksenä, johon on merkitty lähettäjäksi hyökkäyksen kohde. Tähän viestiin kaikki verkon koneet pyrkivät vastaamaan, jolloin kohdetietokoneen tietoliikennekapasiteetti ja resurssit käytetään viestien vastaanottamiseen (kuva 3.4).

Fraggle on hyvä esimerkki vahvistetusta hyökkäyksestä, jossa verkon laitteiden

määrä vaikuttaa siihen, kuinka vakava hyökkäys on [14]. Vastaavanlainen hyökkäys voidaan myös toteuttaa kahden koneen välillä, jos kummassakin on sallittuna UDP Echo -viestit. Tällöin hyökkääjä väärentää viestiin lähettäjän osoitteen ja halutun kohdeportin. Vastaanottaja vastaa tähän viestiin omalla Echo-viestillä, ja näin kahden koneen välille muodostuu ikuinen silmukka [21].



Kuva 3.4: Vahvistettu UDP Echo -hyökkäys.

3.2.3 Smurf

Smurf on yksi ensimmäisistä vahvistetuista DoS-hyökkäyksistä ja se toimii lähes identtisesti Fragglen kanssa sillä erolla, että UDP:n sijasta käytetään ICMP-protokollaa. Hyökkääjä lähettää kohteen puolesta yleislähetyksenä verkolle ICMP Echo -paketin, johon verkon laitteet vastaavat, jos Echo-viestit on sallittuja verkossa. Jo sadan tietokoneen lähiverkko pystyy aiheuttamaan 14 Mb/s haittaliikenteen kohdekoneelle, joten pienessäkin väärin konfiguroidussa verkossa on mahdollista muodostaa riittävästi haittaliikennettä tietoliikenteen häiritsemiseen tai estämiseen [17].

Jos hyökkäys on päässyt käyntiin, ei tälle ole paljoa muuta tehtävissä kuin poistaa kohteena olevat laitteet pois verkosta. Paras vastatoimi Fragglen ja Smurffin kaltaisille hyökkäyksille on konfiguroida verkon laitteet alusta asti oikein. ICMP Echo

-yleislähetysten estämisellä pääsee jo myös pitkälle. ICMP-protokollan käyttöä verkossa kannattaa muutenkin tarkkailla ja tarvittaessa rajoittaa [17].

3.3 Distributed Denial of Service

Vielä viime vuosikymmenellä palvelunestohyökkäykset perustuivat yleensä käyttöjärjestelmistä löydettyjen heikkouksien hyödyntämiseen. Näiden hyökkäysten aikaansaama tuho riippui paljolti käytettävistä järjestelmistä ja vahingot olivat melko rajattua. Palvelunestohyökkäyksiä ei nähty tällöin kovinkaan vakavana uhkana, mutta suhtautuminen muuttui 2000-luvulle tultaessa, kun hajautetut palvelunestohyökkäykset (engl. *Distributed Denial of Service*) kehitettiin. Tällaisessa hyökkäyksessä palvelua saattaa olla lamauttamassa jopa yli 140 000 saastunutta konetta, joita kutsutaan zombeiksi (engl. *zombie*).

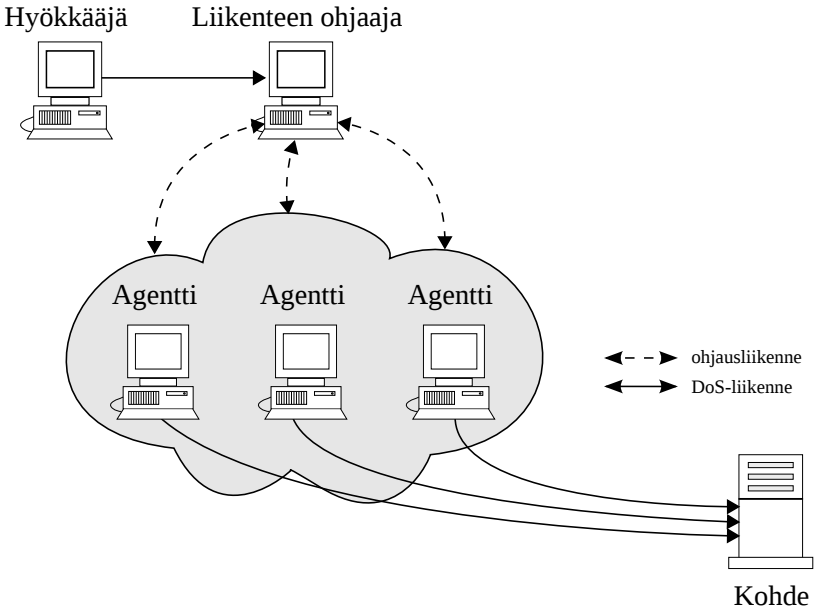
Tietokone voidaan muuttaa zombiksi murtautumalla ja asentamalla siihen ohjelmisto, jonka avulla murtautuja pystyy hallitsemaan laitetta käyttäjän tietämättä. Zombiverkoston ei tarvitse edes olla kovinkaan suuri aiheuttaakseen tuhoa, sillä jo 3 000 koneen verkosto, jossa jokainen kone tuottaa 25 kb/s liikennettä, aiheuttaa yhteensä 75 Mb/s kuormituksen verkolle [17]. Palvelunestohyökkäyksen seuraamukset ovat varautumattomalle taholle usein katastrofaaliset ja pahimmillaan hyökkäys saattaa pysäyttää organisaation toiminnan useiksi päiviksi [16].

Hyökkäysten hajauttaminen useammalla koneelle tuo useita hyötyjä verrattuna perinteiseen palvelunestohyökkäykseen, jossa hyökkäys toteutetaan käyttäen yhtä konetta. Otetaan esimerkiksi vaikka Web-palveluita tarjoava palvelin, jota vastaan halutaan tehdä palvelunestohyökkäys. Tällaisella palvelimella on käytössä huomattavasti suurempi määrä resursseja (tietoliikennekapasiteetti, muisti, laskentateho) kuin yksittäisellä zombitietokoneella. Kun hyökkäys suoritetaan yhtäaikaisesti useammalla koneella, pystyy hyökkääjä kuluttamaan suuretkin resurssit loppuun lyhyessä ajassa. Yksittäiseltä koneelta tuleva palvelunestohyökkäys on helppo tunnistaa ja estää verkon ylläpitäjän toimesta. Hyökkääjien määrän kasvaessa tehtävä vaikeutuu huomattavasti, koska haittaliikennettä aiheuttavat tietokoneet tai itse haittaliikenne pitää tunnistaa hyötyliikenteen joukosta. Mikäli zombitietokoneet ovat sijoittuneet ympäri Internetiä, on hyökkäysten pysäyttäminen ilman palvelun laadun heikkenemistä mahdoton tehtävä ilman automatisointia. Haittaliikennettä on tällöin myös vaikea erottaa tavallisesta liikenteestä, koska se tulee palvelimelle useita reittejä pitkin [15].

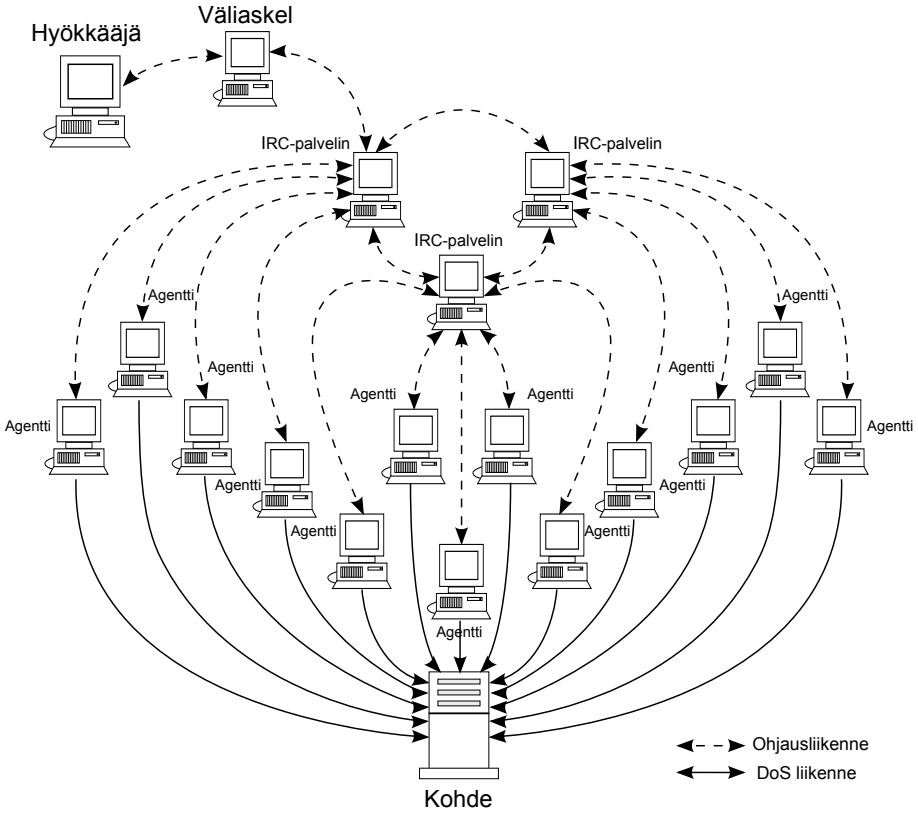
Nopeiden yhteyksien ja verkossa olevien tietokoneiden räjähdysmäinen kasvu on tehnyt hajautetuista palvelunestohyökkäyksistä hyvin tehokkaita ja tuhoisia. Suuri määrä verkkoon liitetyistä laitteista on huonosti suojattu ja päivitetty, ja ihmisten tietoisuus verkon vaaroista on usein puutteellista. Tämä on luonut otollisen maaperän suurten zombiverkostojen muodostamiselle virusten ja matojen avustuksella. Tarjolla on myös tätä tarkoitusta varten suunniteltuja työkaluja ja ohjelmistoja, jotka tekevät tarvittavat toimenpiteet automaattisesti. Jo ensimmäiset, vapaasti saatavilla olevat työkalut kuten Trinoo ja Shaft, mahdollistivat tuhansien suojaamattomien koneiden haltuunoton ilman suurempaa aiheeseen perehtymistä [15].

Hyökkäyksessä käytetyt koneet jaetaan agentteihin (engl. *agents*) ja liikenteen ohjaajiin (engl. *handlers*) riippuen näiden rooleista. Näistä koneista hyökkääjä käskyttää suoraan liikenteen ohjaajia, jotka ohjaavat edelleen annetut käskyt agenteille hyökkäyksen alettua [14] [15]. Ensimmäisissä DDoS-verkoissa nämä ohjausviestit olivat salaamattomia, jolloin kuka tahansa pystyi kaappaamaan näitä. Tämä oli tietomurtojen tekijöiden näkökulmasta ongelmallista, koska liikenteen ohjaajat joutuivat ylläpitämään listaa agenttien käyttämistä IP-osoitteista. Mikäli liikenteen välittäjä joutui vastahyökkäyksen kohteeksi, pystyttiin selvittämään zombiverkon jokaisen laitteen osoitteet. Tätä suoran käskyttämisen heikkoutta pyrittiin korjaamaan muun muassa salaamalla käytettävä viestiliikenne. Ajan saatossa liikenteen ohjaajat pystyttiin kuitenkin tunnistamaan ja poistamaan verkosta, jolloin hyökkäyksessä käytetty zombiverkko hajosi [15].

Nykyisin enää harva DDoS-hyökkäyksessä käytetty verkko käyttää kuvan 3.5 hierarkiaa sen heikon ohjattavuuden takia. Sen tilalle on tullut epäsuoraan käskyttämiseen perustuvat verkot, jotka käyttävät viestien välittämiseen IRC-verkkoja (kuva 3.6). IRC on reaaliaikaiseen viestittämiseen tarkoitettu palvelu, jonka kautta ihmiset pystyvät reaaliajassa kirjoittamaan ja lukemaan viestejä. IRC-verkot koostuvat käyttäjistä ja IRC-kanavista, joille käyttäjät voivat halutessaan liittyä. Epäsuoran käskyttämisen periaatteella toimivat verkot hyödyntävät näitä samoja ominaisuuksia. Jokainen kone liittyy IRC-kanavalle, joka on yleensä salasanalla suojattu. Kanavan kautta hyökkääjä voi antaa käskyjä koneille. Tällä tavalla saadaan monia etuja verrattaessa suoraan käskyttämiseen. Ensinnäkin liikennettä ei pystytä enää tunnistamaan poikkeavaksi, koska se on tavanomaista IRC-liikennettä. Toisekseen käytettyä kanavaa voidaan vaihtaa lennosta, jolloin yhden kanavan sulkeminen ei pysäytä verkon toimintaa. Näiden syiden takia DDoS-hyökkäysten pysäyttäminen ennen niiden käynnistymistä on erittäin vaikeaa [15].



Kuva 3.5: Perinteinen DDoS-verkosto.



Kuva 3.6: Nykyisin käytetty DDoS-verkosto.

3.4 Remote-to-Local

Remote-to-Local (lyh. *R2L*) on hyökkäystyyppi, jossa hyökkääjä pyrkii saamaan koneelle laajemmat oikeudet, kuin mitä hänellä muuten olisi. Tämä tapahtuu useimmiten käyttäen hyväksi järjestelmässä olevia heikkouksia, joiden avulla hyökkääjä pääsee verkon yli murtautumaan koneelle [13]. Pahimmassa tapauksessa hyökkääjä saa hankittua koneelle pääkäyttäjän oikeudet, jolloin koneen ja verkon resurssit ovat täysin hyökkääjän käytettävissä.

Onnistuneet R2L-hyökkäykset ovat verkon ylläpitäjien kannalta pahimpia mahdollisia, sillä niiden mahdollistamat tuhot ja aiheuttamat kustannukset ovat huomattavasti suurempia verrattuna muihin hyökkäystyyppeihin [23]. Onnistunut R2L-hyökkäys saattaa myös muut lähiverkon koneet vaaraan, sillä usein hyökkääjä pyrkii asentamaan koneisiin ohjelmistoja, joiden avulla hyökkääjä pystyy ottamaan koneet haltuunsa. Tällä tavoin osa aikaisemmin mainituista zombiverkostoista saa alkunsa.

Käytetyimmät Web-palvelinohjelmistot Apache ja IIS vastaavat noin 85 % kaikista käytetyistä palvelinsovelluksista. Näiden kahden lisäksi BIND-ohjelmistoa käytetään valtaosassa nimipalvelimista. Ohjelmistojen yleisyydestä johtuen yli puolet R2L-hyökkäyksen mahdollistavista heikkouksista onkin löydetty näille alustoille [13]. Useat haavoittuvaisuudet johtuvat ohjelmointivirheistä, joiden johdosta hyökkääjä pystyy aiheuttamaan sovellukseen muistin ylivuodon (engl. *Buffer Overflow*). Tämä usein kaataa sovelluksen tai saattaa sen sellaiseen tilaan, jossa hyökkääjä pystyy ajamaan omia komentoja koneella. Kattava listaus löydetyistä haavoittuvuuksista ja näiden korjauksista löytyy osoitteesta `www.cve.mitre.org/cve` [24].

Tunnetuilta R2L-hyökkäyksiltä suojautuminen on usein hyvin yksinkertaista, sillä nykyinen trendi on, että haavoittuvuuden löytänyt taho ilmoittaa siitä yleensä ensin sovelluksen kehittäjille ennen julkista ilmoitusta. Siksi korjaus haavoittuvuuteen on usein olemassa ennen kuin sitä on mahdollista hyödyntää [23]. Vastuu jääkin verkon ylläpitäjälle, että käytetyt sovellukset ja suojausjärjestelmät pidetään ajan tasalla. Tästä huolimatta suurin osa onnistuneista hyökkäyksistä johtuu edelleen siitä, että tunnettuja tietoturva-aukkoja ei ole korjattu.

3.5 User-to-Root

User-to-Root (lyh. *U2R*) hyökkäyksessä murtautuja pyrkii hankkimaan koneelle pääkäyttäjän oikeudet. Tämä tapahtuu käyttäen järjestelmässä olevia haavoittuvaisuuksia, joita ei ole paikattu. Useimmiten hyökkäykset pohjautuvat koodausvirheisiin, jotka mahdollistavat ylivuodon aiheuttamisen sekä odottamattomien syötteiden antamisen [13]. Käyttäjästä pääkäyttäjäksi hyökkäys eroaa R2L hyökkäyksestä siten, että hyökkääjällä on jo valmiiksi pääsy koneelle tavallisen käyttäjän oikeuksilla.

U2R-hyökkäykset ovat L2R-hyökkäysten ohella vaikeimpia torjua, jos kohteena oleva järjestelmä on altis U2R-hyökkäyksille. Vaikea torjuttavuus johtuu siitä, että usein hyökkäyksestä aiheutuva liikenne muistuttaa hyvin paljon normaalia liikennettä, jonka vuoksi itsestään oppivat puolustusjärjestelmät eivät pysty riittävän tarkasti erottamaan haitallista liikennettä normaalista [25]. Kehittyneimmilläkin järjestelmillä näiden hyökkäysten tunnistaminen on todella heikkoa. Tämän ovat osoittaneet useat eri tutkimukset, jotka ovat käyttäneet järjestelmiensä testaamiseen DARPAn simuloimaa verkkoliikennettä, jossa olevat tietomurrot ja muu poikkeava toiminta on tiedossa. Parhaimmillaankin tunnistaminen on jäänyt 20 prosentin paikkeille.

4 Web-palvelut ja niiden tietoturva

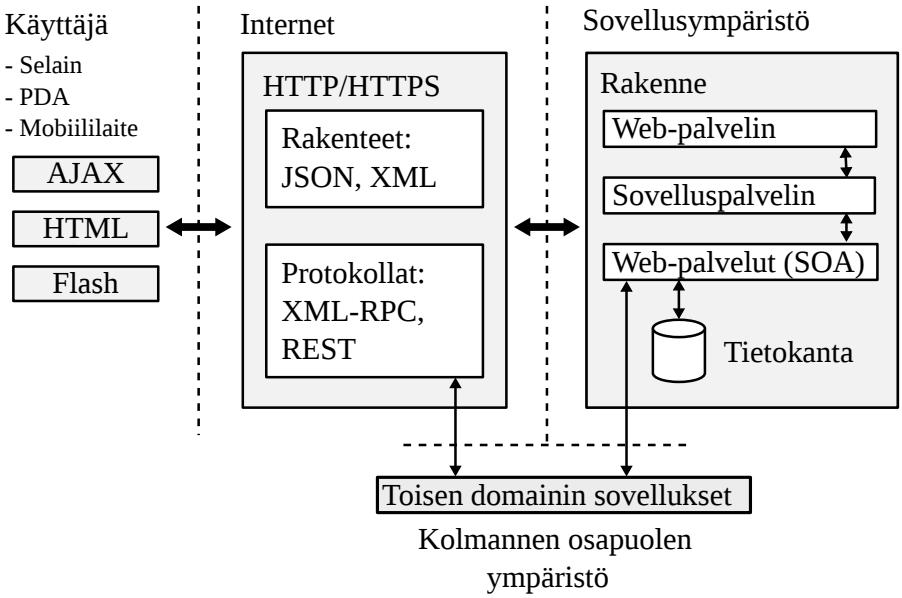
Toimivan ja turvallisen Web-pohjaisen palvelun tarjoaminen vaatii nykyisin todella paljon aikaa ja huolellisuutta sekä palvelun kehittäjältä että palvelun tarjoajalta ja ylläpitäjältä. Ajat ovat muuttuneet siitä, jolloin käyttäjät selailivat pääasiassa staattisia Web-sivuja, ja käyttivät tarjotuista palveluista korkeintaan sähköpostia. Kehitys kulkee kovaa vauhtia eteenpäin, ja tämän päivän suurimpia trendejä ovat interaktiivisuus, sosiaalisuus ja yksilöllisyys, joiden avustuksella verkon käytöstä on pyritty tekemään käyttäjille entistä henkilökohtaisempi kokemus. Palvelut kuten MySpace, Facebook ja YouTube ovat vahvistaneet näitä käyttäjätottumuksia, ja markkinoille on syntynyt kova kilpailu siitä, kuka kehittää seuraavan menestyspalvelun. Nykyisin puhutaankin Internetin seuraavasta evoluutiosta Web 2.0:n muodossa, joita myös edellä mainitut palvelut edustavat. Uudet teknologiat ja kiire tuovat kuitenkin aina mukanaan joukon uusia heikkouksia, joita hyökkääjät pyrkivät hyödyntämään.

4.1 Dynaaminen Web

Web 2.0 on termi, jota käytetään monessa eri merkityksessä ja asiayhteydessä. Se yhdistetään usein uusiin dynaamisiin Web-teknologioihin ja Internet- aikakauden tuotteiden ja palveluiden kehityskaarien kuvaamiseen [26]. Yhteistä näille on se, että ne pyrkivät esittämään sitä muutosta ja kasvua, jota Internet pitää tällä hetkellä sisällään. Tämä muutos on lähtöisin siitä, että kuluttajatottumukset ovat kehittyneet kohti interaktiivisia palvelumalleja, joissa käyttäjillä on entistä suurempi mahdollisuus vaikuttaa siihen, miten haluttu informaatio esitetään. Sosiaalisuus ja sen luoma yhteisöllisyys ovat luoneet tarpeen palveluille, joissa käyttäjät pystyvät tekemään useita asioita saman aikaisesti saumattomasti yhdestä paikasta käsin. Toinen kantava voima muutokselle on ollut markkinoiden tuoma paine, johon yritykset ovat pyrkineet mukautumaan sekä kuluttaja- että yrityspuolella. Tätä varten yritykset ovat kehittäneet entistä suurempia ja monimutkaisempia palvelukokonaisuuksia uusilla teknologioilla, joiden käyttöä ei aina riittävästi hallita. Tähän kun lisätään kiire päästä markkinoilla ensimmäisten joukossa, niin tietoturva-asiat jäävät usein taka-

alalle [27].

Tietoturvan kannalta teknologiat, jotka luetaan kuuluvan osaksi Web 2.0 teknologiaperhettä, ovat jatkuvan huomion ja kehityksen kohteena. Nämä teknologiat muodostavat sen voiman, joka mahdollistaa siirtymisen dynaamisiin sovelluksiin, kuten Google Maps ja yritysten toimintojen ohjaamisen verkkoon. Teknologiat kuten Asynchronous JavaScript And XML (AJAX), Cascading Style Sheet (CSS), Flash, JSON ja XML voidaan kaikki laskea kuuluvan osaksi Web 2.0-perhettä. Osa näistä teknologioista on ollut jo pidemmän aikaa käytössä, kun taas osaa on vasta nyt alettu hyödyntämään siihen, mihin ne on alun perin suunniteltu [26]. Kuvassa 4.1 on nähtävillä näiden yleisimpien teknologioiden ja protokollien väliset suhteet.



Kuva 4.1: Web 2.0 -teknologiaperhe.

Tietoturvan kannalta dynaamiset Web-teknologiat tuovat mukanaan joukon uusia haasteita. Ensinnäkin samat vanhat tietoturvariskit, jotka vaivasivat jo Web 1.0 aikoihin, ovat edelleen voimassa. Näiden lisäksi hyökkäykset kuten Cross-Site Scripting (XSS) ja Cross-Site Request Forgery (CSRF) ovat aikaisempaa vaarallisempia, koska käytetyt teknologiat tarjoavat monipuolisemman ympäristön, jonka kautta murtautua järjestelmiin [26]. Dynaamiset Web-sovellukset antavat myös loppukäyttäjille ja takana oleville ohjelmille enemmän valtaa, jonka johdosta loppukäyttäjä ei välttämättä edes huomaa joutuessaan tietomurron kohteeksi. Tästä hyvänä esimerkkinä Ajax-teknologia, joka mahdollistaa sivujen tietojen päivittämisen käyttäen asynkronisia kutsuja. Tekniikan ansiosta käyttäjä pystyy tekemään kutsuja pal-

velimille ja päivittämään osan sivun tiedoista niin, ettei koko sivua tarvitse päivittää. Tästä suurin osa tapahtuu käyttäjältä piilossa, joten hän ei todennäköisesti huomaa, jos selain lataa haitallisia skriptejä koneelle käyttäen jotain tunnettua tietoturva-aukkoa. Yhden arvion mukaan jopa 70 prosenttia kaikista haitallisista koodeista ladataankin käyttäen Ajaxia [28].

Palvelinpuolella muutokset eivät rajoitu vain dynaamisten Web-teknologioiden tuomiin tietoturvariskeihin, sillä uudenlainen ajattelu vaatii myös uudenlaista palveluarkkitehtuuria. Uusi arkkitehtuuriratkaisu tuo aina mukanaan suuren joukon muutoksia, jotka tulee ottaa huomioon tietoturvan suunnittelussa. Palvelukeskeinen arkkitehtuuri (engl. Service Oriented Architecture, SOA) on yksi näistä kehysmalleista, joka on kasvattanut suosiotaan Web 2.0:n vanavedessä. SOA-arkkitehtuurilla onkin nykyisin tärkeä rooli palvelujen välisen kommunikoinnin kehittämisessä. Siksi on tärkeää ymmärtää, mistä palasista SOA-arkkitehtuuriin perustuvat palvelut koostuvat, ja mitä tämä merkitsee tietoturvan kannalta.

4.2 Palvelukeskeinen arkkitehtuuri

Web-pohjaiset sovellukset ovat saaneet yhä suurempaa huomiota yritysten tuotekehityksessä. Muutoksen taustalla on teknologian kehittyminen siihen pisteeseen, missä toimittajat pystyvät luotettavasti ja nopeasti tarjoamaan aikaisempien palvelin-asiakas -sovelluksien sijaan verkon välityksellä käytettäviä sovelluksia. Tämä ratkaisu on tuonut mukanaan taloudellisia säästöjä yrityksille, jotka ovat ennen joutuneet itse huolehtimaan muun muassa sovellusten ajan tasalla pitämisestä ja palvelimien ylläpitämisestä [26]. Muutos on luonut myös tarpeen löytää yhä tehokkaampia kehitysmalleja ja tapoja toteuttaa entistä monimutkaisempia ja vaativimpia sovelluksia suuryritysten tarpeisiin. Yksi tapa hallita tätä muutosta on perustaa tehdyt ohjelmistosuunnittelun ratkaisut palvelukeskeisen arkkitehtuurin malliin.

Puhuttaessa dynaamisista Web-palveluista, palvelukeskeinen arkkitehtuuri lähtee siitä ajatuksesta, että palveluiden toiminnot ja prosessit pyritään suunnittelemaan toimimaan mahdollisimman itsenäisesti ja avoimesti siten, että niitä pystytään kutsumaan joustavasti eri sovellusten välillä käyttäen standardoitua rajapintaa. Tähän ratkaisuun ovat ajaneet pyrkimys laskea IT-kustannuksia ja uudelleenkäytön maksimointi jo käytössä olevissa ratkaisuissa. Aikaisemmin tämän toteuttaminen on ollut vaikeaa sovellusten heterogeenisyyden takia, jolloin eri alustojen ja toimittajien väliset yhteistoiminnot ovat olleet usein mahdottomia toteuttaa. Yhä

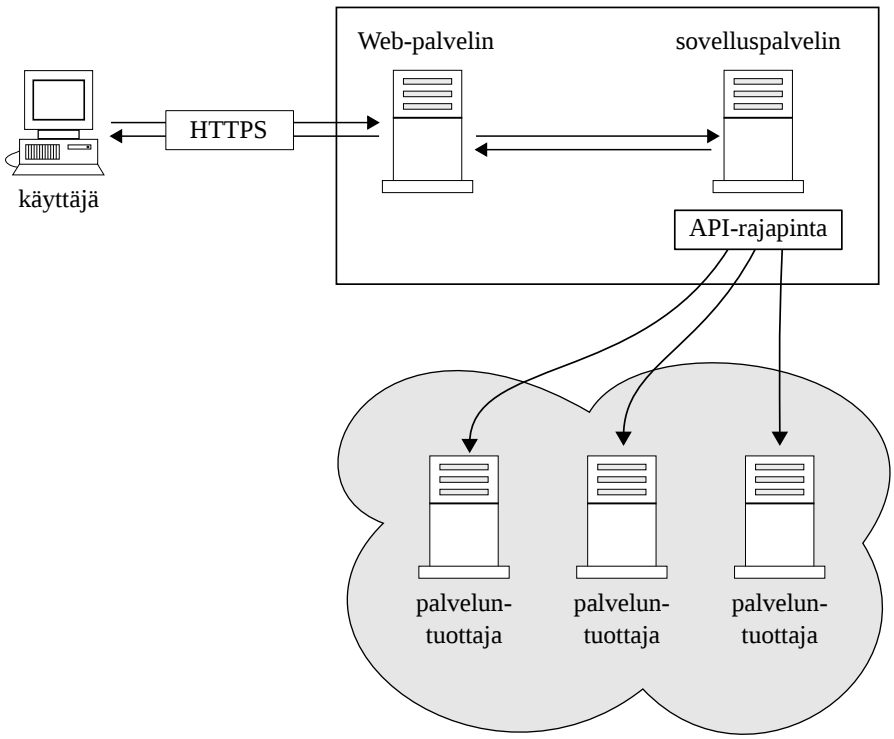
useamman sovelluksen ja palvelun siirtyessä verkkoon tästä rajoitteesta pyritään nyt pääsemään eroon, ja tähän ongelmaan palvelukeskeinen arkkitehtuuri on suunniteltu tuomaan ratkaisu.

Käytännössä siirtyminen palvelukeskeiseen arkkitehtuuriin tarkoittaa sitä, että arkkitehtuurin tulee tarjota sovelluksille mahdollisimman joustava sekä sijainnista ja protokollasta riippumattoman alusta [29]. Tämän avulla loppukäyttäjälle voidaan tarjota palveluita monesta eri paikasta yhtä aikaa siten, ettei hän ole edes tästä tietoinen. Kyseessä on sama ajattelumalli, johon dynaamiset Web-sovellukset pohjautuvat, ja siksi palvelukeskeinen arkkitehtuuri on saanut paljon nostetta Web 2.0:n myötä.

Palvelukeskeisen arkkitehtuurin tuomat edut eivät rajoitu pelkästään kustannussäästöihin. Sen tuomiin etuihin kuuluu myös muun muassa uusien toimintojen helpompi integrointi jo käytössä oleviin järjestelmiin sekä monimutkaisten kokonaisuuksien sujuvampi hallinta. Organisaatioille on myös tärkeää, että tällä tavalla suunniteltuihin palvelumalleihin on nopeampi tehdä haluttuja muutoksia, jolloin reagointi markkinamuutoksiin voidaan toteuttaa nopeassa aikataulussa [28]. Tämä on erityisen tärkeää, koska markkinoille on ilmestynyt monia uusia tekniikoita, jotka mahdollistavat Web-palveluiden integroinnin jo käytössä oleviin palveluihin.

Vuonna 2008 Web-pohjaisten palveluiden kehittämiseen käytettiin arviolta yli 11 miljardia dollaria ja osa fokuksesta on jo siirtynyt kohti keskikokoisia ja pieniä yrityksiä. Web-pohjaisten palveluiden merkityksen uskotaan entisestään kasvavan ja yritykset, jotka reagoivat hitaasti tähän muutokseen, huomaavat tulevaisuudessa olevansa epäsuotuisassa asemassa [27].

Palvelimen arkkitehtuurissa tapahtuvat muutokset tarkoittavat sopeutumista uudenlaiseen ajattelumalliin, jossa palvelut on hajautettu ympäri verkkoa. Näistä palveluista kaikki eivät välttämättä ole yhteisen tietohallinnon alaisuudessa. Kuvan 4.2 mukainen ratkaisu tulee olemaan tulevaisuudessa arkipäivää ja se tuo mukanaan uusia rajapintoja, joiden kautta hyökkääjä voi pyrkiä murtautumaan järjestelmään tai jopa yrityksen sisäiseen verkkoon. Koska palveluiden välinen kommunikointi perustuu tässä mallissa luottamussuhteen luomiseen, tulee erityistä kiinnittää huomiota siihen, kuinka salaus ja tunnistautuminen hoidetaan Web-palveluita tarjoavien osapuolien sekä käyttäjien kesken [27].



Kuva 4.2: Palvelukeskeinen arkkitehtuuri.

4.3 Web-palveluihin kohdistuvia hyökkäyksiä

Internetin kehittyessä myös Web-palveluihin kohdistuvat hyökkäykset ovat saaneet uusia ilmenemismuotoja. Aikaisemmin "Web-hakkeroinnin" nimellä kutsuttiin hyökkäyksiä, jotka kohdistuivat palveluita tarjoaviin alustoihin kuten Apache ja Microsoft IIS. Nämä hyökkäykset perustuivat tunnettujen haavoittuvaisuuksien hyödyntämiseen, ja oikeille työkaluilla varustautunut hyökkääjä pystyi kaatamaan haavoittuvan palvelun muutamassa minuutissa. Esimerkiksi tunnetuimpien Internet-matojen Code Redin ja Nimdan toiminta perustui Microsoft IIS:sä olevan haavoittuvuuden hyödyntämiseen [17].

Tämänlaiset hyökkäykset ovat kuitenkin menettäneet suurimman tehonsa, koska tehdyistä virheistä on opittu. Löydettyihin haavoittuvaisuuksiin reagoidaan entistä nopeammin, yhä useampi alusta on konfiguroitu oikein ja saatavilla on työkaluja, joiden avulla pystytään nopeasti ja helposti havaitsemaan yleisimmät tietoturvariskit [17]. Näistä seikoista johtuen hyökkääjät ovat alkaneet kiinnittämään entistä enemmän huomiota itse alustalla pyöritettäviin palveluihin pyrkien murtautumaan järjestelmiin näiden kautta. Dynaamisten Web-teknologioiden myötä tämän

tyyppiset hyökkäykset ovat saaneet enemmän huomiota, sillä se tarjoaa hyökkääjille entistä monipuolisemman hyökkäyspinnan. Tietoturvan kannalta onkin tärkeää, että kumpaankin hyökkäystapaan kiinnitetään huomiota.

4.3.1 SQL-injektio

Virheelliseen syöttötietoon perustuvat hyökkäykset ovat jo pidemmän aikaa vaivanneet Web-pohjaisia sovelluksia. Dynaamisten Web-sovellusten myötä hyökkäykset ovat entisestään yleistyneet, sillä monimutkaisten sovellusten takana käytetään entistä enemmän erilaisia tietokantapohjaisia ratkaisuja kuten MySQL. Nämä hyökkäykset hyödyntävät sitä seikkaa, että suurin osa sovelluksista ei tee selkeää eroa käyttäjän antaman syötteen ja ohjelmalle annettavien käskyjen välillä. Tämä mahdollistaa sen, että hyökkääjä pystyy piilottamaan annettuun hakuun ohjelmatason käskyjä, jotka muokkaavat sovelluksen toimintaa hyökkääjän haluamaan suuntaan [26]. Tavanomaisilla menetelmillä, kuten palomuureilla, tällaisen hyökkäysten tunnistaminen on hankalaa, sillä monien organisaatioiden tietoturvasta vastaavat palomuurit toimivat OSI-mallin kolmannella, neljännellä ja viidennellä kerroksella. Tämän vuoksi niiltä jäävät huomaamatta ylimmän kerroksen eli ohjelmistokerroksen kautta tulevat hyökkäykset. Tämän lisäksi useimmat palomuurit eivät ymmärrä sovellusprotokollien, kuten HTTP:n, tarkkaa sisältöä [30].

Onnistunut injektiohyökkäys koostuu kolmesta erillisestä vaiheesta. Ensimmäisessä vaiheessa hyökkääjän tulee tunnistaa Web-sovelluksessa käytetyt teknologiat. Tämä onnistuu muun muassa tulkitsemalla sivujen antamia virheilmoituksia, tutkimalla sivun lähdekoodia ja käyttämällä tätä tarkoitusta varten tehtyjä työkaluja kuten Nessusta ja Nmapia. Osaavalla hyökkääjälle tämä vaihe on melko triviaali, mutta hyökkäyksen kannalta tiedot ovat elintärkeitä, sillä injektiohyökkäyksen onnistuminen on täysin riippuvainen käytetystä alustasta.

Kun tarvittavat tiedot on saatu kerättyä, voidaan varsinaista hyökkäystä alkaa suunnittelemaan. Tämä tapahtuu tunnistamalla ne syötteet, jotka käyttäjä pystyy sovellukselle antamaan. Tämä käsittää käytännössä kaiken sen tiedon, joka voidaan välittää HTTP:n GET- ja POST-pyynnöissä. Viimeisessä vaiheessa hyökkääjän tehtäväksi jää niiden syötteiden tunnistaminen, joita käyttämällä sovelluksen toimintaa saadaan muokattua. Tähänkin tehtävään löytyy useita valmiita työkaluja, ja monessa tapauksessa samat toimintaperiaatteet toimivat monilla eri alustoilla [26].

Structured Query Language (lyh. SQL), joka on alan vakiintunut standardi tietokantojen käsittelyyn, on käytössä lähes jokaisessa Web-sovelluksessa, joka käyttää

jonkinlaista tietokantaratkaisua. Sen syntaksi on sekoitus ohjelmakäskyjä ja käyttäjän antamia syötteitä, ja huonosti määritettynä käyttäjän antama syöte voidaan tulkita virheellisesti ohjelmatason käskyksi. Tämän syötteen hyökkääjä joutuu usein etsimään sokeasti, mutta syötteet kuten

- `' OR 1=1 --`
- `') OR 1=1 --`

toimivat hyvin usein [26]. Virheellisestä hausta aiheutuvat virheilmoitukset antavat myös erittäin paljon hyödyllistä tietoa hyökkääjälle [30]. Koska suurin osa SQL-tietokannoista mahdollistaa useamman perättäisen syötteen antamisen, kunhan syntaksi pysyy oikeana, voidaan näiden avulla katkaista ohjelman normaali toiminta. Otetaan esimerkiksi seuraava Java-kielellä toteutettu SQL-kyselyn muodostaminen:

```
String query = "SELECT␣id␣FROM␣user␣WHERE␣" +
               "username='" + username + "'␣AND␣" +
               "password=PASSWORD('" + password + "')';
```

Mikäli muuttujan `username` arvo on

```
'␣OR␣1=1;DROP␣TABLE␣user;--
```

tällöin tietokannalle ohjattava kysely on seuraava:

```
String query = "SELECT␣id␣FROM␣user␣WHERE␣" +
               "username='" + "'␣OR␣1=1;DROP␣TABLE␣user;--␣"+
               "'␣AND␣password␣=␣PASSWORD␣('x')";
```

SQL-kielessä kaksi perättäistä viivaa tarkoittaa, että kaikki siitä oikealla puolella oleva on kommenttia. Näin ollen lopullinen SQL-haku on:

```
SELECT id FROM user WHERE username='' OR 1=1;
DROP TABLE user;
```

Näin muodostettu haku on syntaktisesti täysin oikea. Kyseinen lause poistaa tietokannassa olevan user-taulun, joka tässä tapauksessa sisältää järjestelmään tallennetut käyttäjät [26].

Vastaavanlaisia tekniikoita käyttäen hyökkääjä pystyy tekemään kaiken sen, joka pystytään esittämään SQL-kyselynä, ja johon ajettavalla sovelluksella on riittävät

oikeudet. Mielivaltaisten käskyjen antaminen, DLL-tiedostojen luominen ja ajaminen sekä tietokannan sisällön lähettäminen jollekin toiselle Internetiin liitetylle palvelimelle ovat vain muutamia esimerkkejä toiminnoista, joita onnistunut hyökkäys voi aiheuttaa [30].

Injektiohyökkäykset eivät rajoitu pelkästään SQL-kieleen, sillä useat eri tekniikat kuten XPath ja LDAP sisältävät vastaavanlaisia heikkouksia [26]. SQL-kieleen pohjautuvat hyökkäykset ovat kuitenkin kaikista yleisimpiä johtuen sen levinneisyydestä. SQL-kieleen pohjautuvat ratkaisut sisältävät usein myös osan seuraavista ominaisuuksista, jotka mahdollistavat hyökkäysten tekemisen:

- Kommentoinnin mahdollistava merkkijono: $--$
- Mahdollisuus suorittaa useita hakuja yhdellä kertaa käyttäen puolipistettä
- SQL-palvelimen antamat tarkat virheilmoitukset
- Dynaaminen tyypitys – kokonaisluvut käännetään mahdollisuuksien mukaan merkkijonoksi, jolloin hyökkääjän on helpompi arvata oikea syöte [30]

Esille tulleiden asioiden valossa on selvää, että SQL-kieleen pohjautuvat haavoittuvuudet ovat todellinen uhka tietoturvalle. Tästä johtuen on tärkeää, että sovellusten turvallisuus pyritään takaamaan kaikin mahdollisin tavoin. Täysin turvallisen sovelluksen suunnitteleminen on kuitenkin mahdoton tehtävä, mutta noudattamalla muutamia perusperiaatteita voidaan riskiä vähentää huomattavasti.

Verkon tietoturvaratkaisuja valittaessa ja suunniteltaessa SQL-injektiohyökkäykset tulee ottaa huomioon. Koska injektiohyökkäykset pyrkivät hyödyntämään syötteiden virheellistä tulkintaa, voi hyväksyttävien syötteiden rajoittaminen tuntua helpolta ratkaisulta ongelmaan. Osittain tämä ratkaisee asian, mutta ongelmaksi muodostuu se, kuinka valita mikä syöte on hyväksyttävä ja mikä virheellinen. Yksi keino on sallia ainoastaan syötteet, jotka tiedetään etukäteen kelvollisiksi. Tällöin huono syöte voidaan jättää kokonaan käsittelemättä. Tämä keino on kaikista rajoittavin, koska tällöin syötteen jokainen merkki tarkistetaan ja jos se ei ole sallittujen merkkien joukossa, niin koko haku hylätään.

Toinen lähestymistapa suojaamiseen on riisua syötteestä ongelmalliset merkit kuten merkkijonon päättävä symboli. Usein tämäkin on vaikea toteuttaa käytännössä, koska joskus voi olla vaikeaa määritellä, mikä haku on huono ja mikä hyvä. Yksi hyväksi todettu tapa onkin rajoittaa tiettyjen merkkien käyttöä siten, että merkin ollessa haussa mukana tämä muutetaan toiseen muotoon. Tämäkään ratkaisu

ei ole täysin ongelmaton, ja siksi usein vain hylätään koko syöte, jos se ei vastaa odotettua syötettä [30].

Palvelinohjelmistoissa tulee myös ottaa huomioon muutamia asioita, joita seuraamalla voidaan huomattavasti parantaa yrityksen tietoturvaa. Tietoturvan kannalta huolellisesti suunnitellun palvelinohjelmiston avulla voidaan myös pienentää huonosti suunnitellun sovelluksen riskiä joutua injektiohyökkäyksen kohteeksi. Seuraava lista ei ole millään tavalla täydellinen ratkaisu ongelmaan, mutta se sisältää tärkeimpiä keinoja injektiohyökkäysten estämiseen ja niiden vaikutusten minimointiin.

- SQL-palvelimen ajaminen pienillä käyttöoikeuksilla
- Sallitaan tallennettujen proseduurien ajaminen ainoastaan järjestelmän ylläpitäjille
- Linkitettyjen palvelimien yksinkertaisten tunnistusten poistaminen
- Määrittää palomuuri siten, että ainoastaan luotetut käyttäjät saavat ottaa yhteyden SQL-palvelimeen. Usein ainoat, joiden yhteydet tulee sallia, ovat ylläpitäjät ja Web-palvelut, joita tietokanta palvelee [30].

4.3.2 Cross-Site Scripting

Nykyisin yhä useampi yritys on tietoinen verkon tarjoamista mahdollisuuksista ja sen mukana tulevista vaaroista. Yritykset ovat tämän johdosta alkaneet panostamaan entistä enemmän tietoturvaan ja niistä harva toimii enää ilman kunnollista suojausta. Palomuurit ja hyökkäysten tunnistamiseen ja estämiseen erikoistuneet sovellukset ovat myös kehittyneet viime vuosina niin paljon, että enää harva perinteinen hyökkäys toimii ilman suuria muutoksia sen toteutustapaan. Tästä johtuen hyökkääjät ovat omaksuneet yhä salamyhkäisempiä murtautumistekniikoita, jotka pyrkivät kokonaan ohittamaan perinteiset palomuurit. Web-pohjaiset sovellukset tarjoavat tähän erinomaisen alustan, ja dynaamisten Web-tekniikoiden myötä mahdollisuudet ovat vain entisestään kasvaneet.

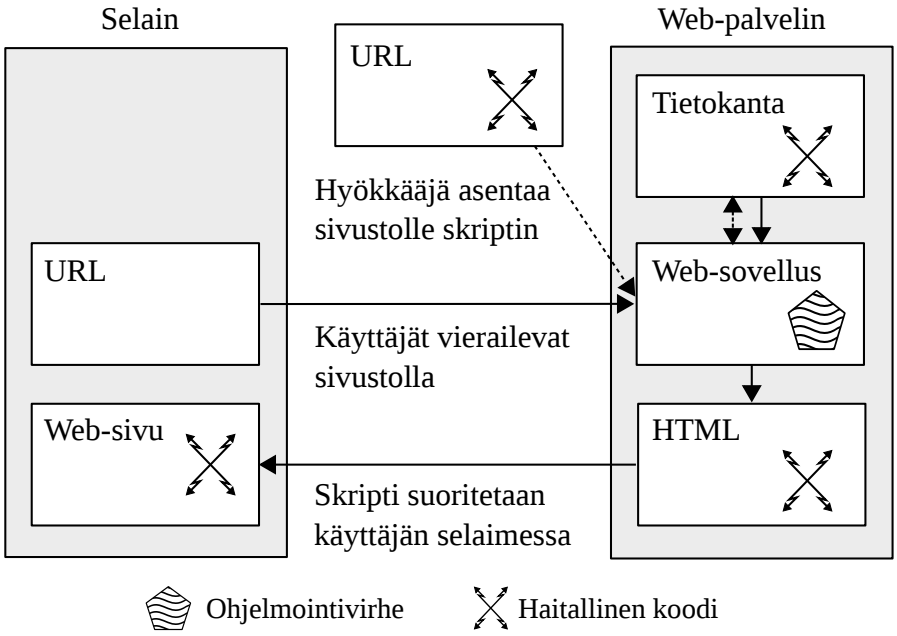
Cross-Site Scripting eli XSS-hyökkäykset eivät ole millään tapaa uusi hyökkäystyyppi, ja joidenkin arvioiden mukaan kahdeksan kymmenestä sivustosta on niille jollakin asteella alttiina. XSS-hyökkäykset ovat myös viime vuosina keränneet suosiota hyökkääjien keskuudessa, sillä dynaamisten Web-sovellusten myötä yhä useampi näistä käyttää erilaisia tiedonlähteitä sisällön keräämiseen. RSS-syötteet,

widgetit, mashupit ja erilaiset JavaScript-pohjaiset komponentit tarjoavat kaikki hyökkääjille monta mahdollisuutta, johon iskeä. Tähän kun lisätään se, että sovelluksista monet päivittävät tietoja käyttäjän tietämättä, niin tilanteesta muodostuu entistä vaarallisempi [27]. Tästä johtuen XSS-hyökkäykset muodostavat nyt ja tulevaisuudessa erittäin suuren riskin tietoturvalle.

Painotuksen siirtyminen kohti Web-pohjaisia hyökkäyksiä on ollut selkeästi havaittavissa jo pidemmän aikaa. Vuonna 2007 johtava tietoturva-alan yritys Symantec julkisti raportin, jonka mukaan vuoden 2007 viimeisen kuuden kuukauden aikana raportoiduista haavoittuvaisuuksista 11,253 oli sivustokohtaisia XSS tietoturva-aukkoja. Tätä lukua kun vertaa muihin löydettyihin riskeihin, joita oli 2,134, niin yli 80 prosenttia löydetyistä tietoturva-aukoista oli Web-pohjaisia [31]. Vuonna 2009 julkistetussa raportissa tilanne oli lähes yhtä paha, sillä 63 prosenttia haavoittuvaisuuksista kohdistui Web-sovelluksiin [32].

Kuten aikaisemmin on tullut esille, niin injektiohyökkäykset eivät rajoitu pelkästään tietokantasovelluksiin. Myös XSS-hyökkäykset ovat eräänlaisia injektiohyökkäyksiä sillä erotuksella, että hyökkääjä ei suoraan toteuta injektiota vaan uhri tekee sen itse. Tähän hyökkääjällä on useita eri keinoja, ja harvoin käyttäjä edes huomaa joutuneensa hyökkäyksen kohteeksi. Hyökkäyksen onnistuessa Web-sivujen käyttämä Same Origin -periaate voidaan kiertää, jolloin hyökkääjän on mahdollista ajaa kohteen selaimessa haluamaansa koodia ja skriptejä. Tämä saman alkuperän periaate on Web-selainten yksi tärkeimmistä suojausmenetelmistä, ja se estää muun muassa skriptien lataamisen yhdestä lähteestä hakemalla tai asettamalla määrityksiä toisesta lähteestä [26].

XSS-hyökkäykset jaetaan heijastettuihin ja tallennettuihin hyökkäyksiin sen mukaan, kuinka hyökkäys on toteutettu. Tallennetussa hyökkäyksessä haavoittuvalle Web-sivustolle asennetaan pysyvästi vihamielinen skripti, joka ajetaan sivun vierailun yhteydessä (kuva 4.3). Tällaiset skriptit ovat yleensä kirjoitettu JavaScriptillä ja ne voidaan tallentaa esimerkiksi Web-palvelimen tietokantaan, kommenttikenttiin tai foorumin viesteihin. Koska selain luottaa siihen, että sivuston sisältö on luotettavasta lähteestä, pystyy hyökkääjä tämän jälkeen lukemaan ja muokkaamaan selaimen arkaluontoista sisältöä. Evästeet, jotka ovat Web-sovellusten toiminnan kannalta erittäin tärkeitä, ovat ensimmäisiä kohteita joihin yritetään yleensä päästä käsiksi, sillä ne ylläpitävät yhteyksien tiloja eri sivustoille ja oikean käyttäjän tunnistamiseen tarvittavia tietoja [27]. Näitä kaappaamalla hyökkääjä pystyy esiintymään verkossa toisena käyttäjänä ja väärinkäyttämään palveluita kuten verkkopankkia

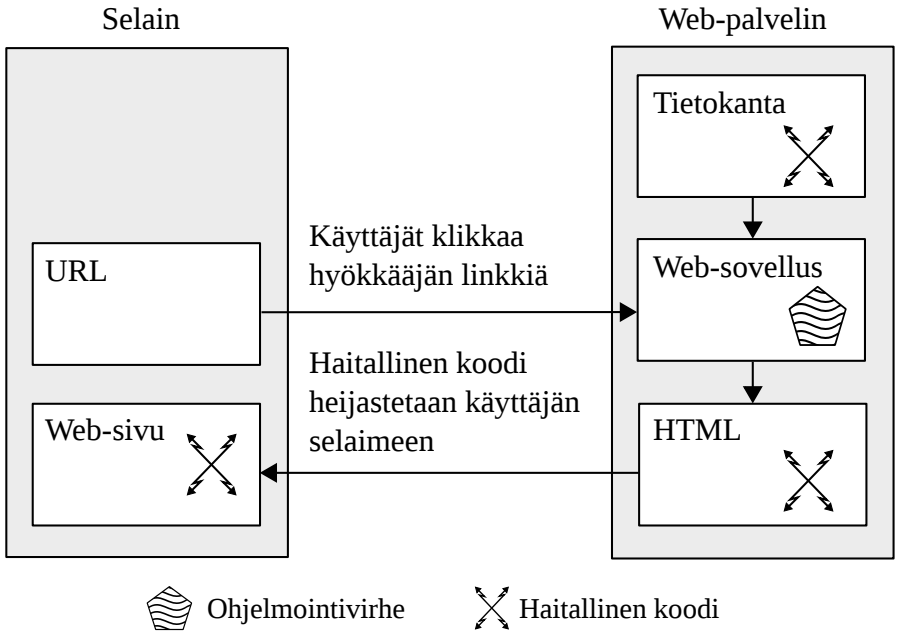


Kuva 4.3: Palvelimelle tallennettu XSS-hyökkäys.

tai webmailia.

Heijastettu hyökkäys (kuva 4.4) eroaa tallennetusta hyökkäyksestä siten, että haitallista skriptiä ei tallenneta minnekään pystyvästä. Hyökkäyksen kohteeksi käyttäjä joutuu, kun hänet huijataan esimerkiksi painamaan vihamielistä linkkiä, joka tekee käyttäjän puolesta kyselyn haavoittuvalle sivustolle. Haavoittuva sivusto heijastaa kyselyn tuloksen takaisin käyttäjälle esimerkiksi virheviestin tai hakutuloksen muodossa, ja näin toimitettu koodi suoritetaan käyttäjän selaimessa, koska se tulee selaimen luottamalta osapuolelta [26].

Hyökkääjän mahdollisuudet eivät rajoitu pelkästään evästeiden varastamiseen. Onnistuneen XSS-hyökkäyksen jälkeen hyökkääjä voi esimerkiksi ohjata käyttäjän osoitteeseen, joka jäljittelee haluttua sivustoa. Käyttäjän yrittäessä kirjautua tähän palveluun, saa hyökkääjä selville käyttäjän tunnuksen ja salasanan. XSS-hyökkäykset tarjoavatkin erinomaisen mahdollisuuden toteuttaa *phishing* eli tiedon kalastelu-hyökkäyksiä. XSS-madot ovat myös viime aikoina yleistyneet, sillä yhä useampi palvelu on niille haavoittuvainen. Esimerkiksi Webmail-sovelluksia varten kirjoitetut madot käyttävät hyödyksi sitä seikkaa, että hyökkääjä pääsee käsiksi uhrin kontaktilistaan. Tällainen mato pyrkii leviämään lähettämällä kontaktilistalla oleviin osoitteisiin sähköpostia, ja koska viesti tulee luotettavasta osoitteesta, ei käyttäjät osaa usein varoa viestissä olevaa linkkiä. Hyökkääjä vielä usein muokkaa lin-



Kuva 4.4: Palvelimelta heijastettu XSS-hyökkäys.

kistä sen näköisen, että vastaanottaja ei osaa epäillä linkin aitoutta [26]. XSS-madot leviävätkin usein todella nopeasti, ja ne kuljettavat usein myös muita hyökkäyksiä mukanaan. Tästä hyvänä esimerkkinä Sammy-madoksi nimetty XSS-hyökkäys, joka vuonna 2005 iski MySpace-sivustolle, joka on yksi tunnetuimmista sosiaaliseen verkostoistumiseen erikoistuneista sivustoista. Käyttäen sivustossa olevaa heikkoutta hyväkseen mato saastutti käyttäjien sivuja, ja jonkun vieraillessa saastuneella sivulla, saastui myös hänen oma MySpace sivunsa. 24 tunnissa mato oli saastuttanut yli miljoona sivua, ja tilanteen korjaamiseksi MySpace joutui sulkemaan väliaikaisesti sivustonsa. Hyökkäyksen vahingot eivät rajoittuneet tähän, vaan seuraavina viikkoina useista tunnetuista sivustoista kuten Googlesta ja Yahoosta löytyi vastaavanlaisia tietoturva-aukkoja [27].

Tärkein asia XSS-hyökkäysten estämiseksi on olla tarkkana sen suhteen, kuinka käyttäjien tarjoama sisältö välitetään toisille käyttäjille. Tärkeää on myös muokata käyttäjän antama syöte aina sellaiseen muotoon, että kun palvelin lähettää esimerkiksi virhesanomassa käyttäjän antaman syötteen takaisin, niin ei tätä voida käyttää sellaisenaan hyökkäyksen toteuttamiseen. Tiettyjen merkkien poistamisella voidaan hyökkäyksiä vähentää, mutta usein merkkien kuten heittomerkin (') poistaminen ei tule kyseeseen. Suunnittelijoiden avuksi on myös olemassa suuri määrä sovelluksia, jotka etsivät sivustoilla olevia heikkouksia [26]. Tämän suhteen tulee kuitenkin

muistaa, että myös hyökkääjillä on käytettävissä samat sovellukset, joten näillä löydetyt tietoturvariskit tulisi korjata mahdollisimman nopeasti. Symantecin raportin mukaan näin kuitenkin tapahtuu todella harvoin, sillä 6,961 sivustosta vain 473 oli korjattu raportin kirjoittamisen aikana, ja tähänkin oli mennyt keskimäärin 52 päivää [31]. XSS-haavoittuvista sivuista kirjaa pitävän sivuston mukaan tilanne ei ole nykyisin yhtään sen parempi, sillä suurin osa löydetyistä sivuista on edelleen korjaamatta [33]. Näyttääkin siltä, että yritykset ymmärtävät asian vakavuuden vasta sitten, kun sivusto on joutunut hyökkäyksen kohteeksi.

4.3.3 Cross-Site Request Forgery

Dynaamisten Web-sovellusten yksi keskeisimmistä selaimessa toimivista ratkaisuista on Document Object Modeliksi (lyh. DOM) nimetty rakenne, joka on vastuussa siitä, kuinka Web-sivun sisältöä käsitellään, ja kuinka eri objektit kommunikoivat keskenään. Tämä rakenne vastaa siitä, että ainoastaan saman domainin sisäinen kommunikointi ja sisällön muokkaaminen on sallittu. Tämä ratkaisu poistaa sen mahdollisuuden, että Web-sivun DOM:ia voitaisiin muokata toisen domainin kautta. Tämä vastaa saman alkuperän periaatetta, jolla pyritään estämään Web-sivuihin ja käyttäjiin kohdistuvia vihamielisiä toimia. Webin muuttuessa entistä interaktiivisemmaksi, ovat sivustot kuitenkin alkaneet sallia yhä enemmän domainien välistä kommunikointia. Tällaisia toimia ovat esimerkiksi hyperlinkkien käyttäminen sisällön näyttämisessä, kuvien ja objektien lataaminen sekä toisessa domainissa olevien JavaScriptien ajaminen. Nämä ja monet muut toimet ovat mahdollistaneet entistä monipuolisempien Web-palveluiden suunnittelun, mutta huonosti toteutettuna ne jättävät palvelut hyvin haavoittuviksi. Tämä johtuu siitä, että käyttäjän tarkoituksella tekemiä toimia ei pystytä erottamaan niistä toimista, jotka tehdään automaattisesti käyttäjän vieraillessa sivustolla [26].

XSS-hyökkäysten tapaan myös Cross-Site Request Forgery (lyh. CSRF) hyökkäykset ovat jo pidemmän aikaan olleet olemassa, mutta vasta viime vuosina ne ovat yleistyneet. Ero näiden kahden hyökkäyksen välillä on siinä, että missä XSS skripti suoritetaan käyttäjän selaimessa, niin CSRF hyökkäyksessä käytetyt käskyt ja skriptit kohdistetaan Web-palveluihin, joiden kanssa selaimella on luottamussuhde. CSRF toteutetaan siten, että hyökkääjä asettaa jollekin sivulle vihamielisiä tageja, joiden avulla voidaan luoda haitallisia HTTP-pyyntöjä. Nämä pyynnöt kohdistetaan johonkin toiseen domainiin ja ne ajetaan käyttäjän vieraillessa kyseisellä sivustolla. Jos käyttäjän selaimella on sitten esimerkiksi tähän domainiin liittyvä

eväste voimassa, voidaan HTTP-pyynnöillä pyytää muun muassa salasanan vaihtoa tai sähköpostin lähettämistä käyttäjän nimissä [27]. Hyökkääjä pystyy tällä tavoin myös ottamaan yhteyttä sellaisiin koneisiin ja palveluihin, joihin hänellä ei muuten olisi oikeuksia johtuen esimerkiksi palomuurista tai IP-osoitteiden suodatuksesta [34].

CSRF hyökkäysten teho perustuu siihen, että monet sivustot ja palvelut käyttävät evästeitä melko huolimattomasti. Evästeitä käytetään käyttäjän yksilöimiseen ja tunnistamiseen, joten riittää, että hyökkääjä pääsee niihin käsiksi tavalla tai toisella. Useat sivustot myös sallivat käyttäjien pysyvän kirjautuneena sisällä jopa useita viikkoja, jolloin sama eväste on pitkään käytössä. Monien sovellusten ja pyyntöjen rakenne on myös hyvin ennalta arvattavissa, jolloin hyökkääjän on helpompi arvata käskyjen ja parametrien oikea muoto [26].

CSRF hyökkäykset eivät rajoitu pelkästään perinteisiin Web-tekniikoihin, sillä suurin osa dynaamisista teknologioista ovat sille jollakin tavalla alttiina. Erityisesti AJAX ja JavaScript ovat tuoneet mukanaan suuren joukon uusia haasteita, jotka suunnittelijoiden tulee ottaa huomioon uusien sovelluksien suunnittelussa [27]. Täydellinen suojautuminen CSRF-hyökkäyksiltä vaatii syvällistä osaamista käytetyistä tekniikoista, ja tietämystä eri komponenttien välisistä toiminnoista. Puutteellisista toteutuksista johtuen monet sivustoista ja palveluista ovat haavoittuvaisia CSRF-hyökkäyksille. Tästä tunnettuna esimerkkinä sähköpostisivusto Gmail, josta löytyi vuonna 2007 CSRF-hyökkäyksen mahdollistava heikkous. Tämän avulla hyökkääjän oli mahdollista luoda Gmailiin sellainen suodatus, joka ohjasi tulevat postit toiseen osoitteeseen [34].

CSRF-hyökkäyksiltä suojautumiseen on käytössä kolme hyvin yleistä tekniikkaa. Näistä käytetyin on sisällyttää jokaiseen palvelinpuolen dataa muokkaavaan GET/POST pyyntöön mukaan salattu tokeni, jonka avulla jokainen pyyntö voidaan yksilöidä ja tunnistaa kuuluvan oikeaan istuntoon. Tämä salattu tokeni tekee käyttäjän syötteestä arvaamattoman muotoisen, jonka johdosta CSRF-hyökkäyksen toteuttaminen on hyvin vaikeaa [26]. Yksinkertaisin suojausmenetelmä on taas tunnistaa HTTP-kyselyn Referer-otsake, ja hyväksyä vain sellaiset pyynnöt, jotka tulevat luotettavista lähteistä. Kolmas menetelmä on tehdä kustomoituja otsakkeita käyttäen XMLHttpRequestia, jonka käyttö on lisääntynyt AJAXin yleistyessä. Kustomoitujen otsakkeiden oikeellisuus sitten varmistetaan, ennen kuin pyynnöt käsitellään palvelinpuolella [34].

Nämä kolme esitettyä tapaa ovat yleisimmin käytössä, mutta nekin sisältävät

joukon heikkouksia. Salatun tokenin ongelma on siinä, että kyseisestä tekniikasta ei ole olemassa yleisesti sovittua toteutustapaa. Tämä aiheuttaa sen, että useat ratkaisut ovat tavalla tai toisella puutteellisia. Referer-otsakkeiden suodatuksen ongelma on siinä kuinka käsitellä pyyntöjä, joista puuttuu Referer-otsake. Otsakkeen piilottaminen ei ole vaikea tehtävä, joten palvelin saattaa joutua päättämään vajailla tiedoilla päästääkö pyyntö läpi vaiko suodattaa se pois. XMLHttpRequestin käyttöä taas rajoittaa sen tuomat lisävaatimukset toteutukselle, joka rajoittaa tiettyjen palveluiden käyttöä [34].

Näiden ja monien muiden syiden takia tutkimustyö CSRF-hyökkäysten parissa on kasvattanut suosiotaan. Yhdessä tutkimuksessa ehdotettiin erillisen Origin otsakkeen lisäämistä pyyntöön, joka korjaisi Referer-kentän heikkoudet [34]. Toisessa tutkimuksessa taas pyrittiin kasvattamaan Web-selaimen turvallisuutta lisäämällä selaimeen mekanismi, joka pyrki arvioimaan vastasiko tehdyt pyynnöt käyttäjän toimia [35]. Näistä kahdesta erilaisesta ratkaisusta on nähtävissä se, että tutkimuskenttä CSRF-hyökkäysten ympärillä on hyvin alkuvaiheessa, ja vastuu riittävän suojauksen hoitamisesta jää vielä tällä hetkellä sovelluksen kehittäjälle ja ylläpitäjälle.

4.4 Web-palveluiden tietoturvan parantaminen

Web-palveluiden ja verkon tietoturvasta vastaaminen on tarkkuutta ja aikaa vaativaa työtä, joka edellyttää omien tietojen jatkuvaa päivittämistä sekä ja liikenteen ja palveluiden seuraamista. Jo pelkästään yleisimpien tietoturvariskien tunnistaminen ja korjaaminen osoittautuu usein käsin tehtynä ylivoimaiseksi tehtäväksi verkkojen ja palveluiden muuttuessa entistä monimutkaisemmiksi. Työn helpottamiseksi on saatavilla useita työkaluja, joiden avulla sekä kehittäjät että ylläpitäjät voivat etsiä sovelluksista ja verkoista tunnetuimpia tietoturva-aukkoja. Osa näistä pohjautuu vapaaseen lähdekoodiin, mutta tarjolla on myös kaupallisia sovelluksia, jotka ainakin paperilla lupaavat parempia ominaisuuksia. Näiden lisäksi palvelinpuolella on tarjolla erilaisia lisäosia, jotka lisäävät Web-palveluiden tietoturvaa. Mikään sovellus ei tietenkään voi löytää jokaista tietoturvariskiä, joten pelkästään näiden varaan ei voida tietoturvaa rakentaa. Näiden avulla saadaan kuitenkin hyvä yleiskuva siitä, minkälaisille heikkouksille sovellus on mahdollisesti alttiina, ja kuinka verkon turvallisuutta voidaan parantaa. Koska samat sovellukset on myös hyökkääjien käytössä, niin ainakin aukot, jotka löytyvät yleisimmin käytössä olevilla työkaluilla, tulisi

korjata mahdollisimman nopeasti.

Web Application Security Consortium (lyh. WASC) on vuonna 2004 perustettu avoin kansainvälinen ryhmittymä, joka koostuu tietoturva-alan asiantuntijoista sekä eri yritysmaailman tahoista, joiden tavoitteena on tuottaa vapaaseen käyttöön best practice -malleja ja standardeja turvallisesta Internetistä. Se julkaisee erilaisia artikkeleita, esitysmateriaaleja ja tietoturvalinjauksia, joita käytetään hyvin laajasti ohjenuorina yritysmaailmassa sekä tuotekehityksessä [36]. WASC:n tämän hetkisiin projekteihin kuuluvat Web-sovellusten turvallisuuden tutkimiseen käytettävien ohjelmistojen arviointikriteerien laatiminen sekä tilastointi Web-sovellusten turvallisuudesta. Viimeisimmän WASC:n tekemän raportin mukaan tutkituista sivustoista, joita oli yhteensä 12186 kappaletta ja joista löytyi 97554 eri asteista tietoturvariskiä, 13 prosenttia pystyttiin murtamaan täysin automaattisesti, ja noin 49 prosenttia Web-sovelluksista sisälsi helposti havaittavia vakavia tietoturvariskejä. Tämä luku kasvoi jopa 96 prosenttiin, kun sivuston tutki tietoturva-alan asiantuntija. Huomattavaa oli myös se, että sivustoista 99 prosenttia ei noudattanut alan standardeja, ja ylläpidon aiheuttamat haavoittuvaisuudet olivat 20 prosenttia yleisempiä kuin sovelluskehittäjien tekemät. Tutkimus myös osoitti sen, että XSS-hyökkäykset olivat SQL-injektiohyökkäysten ohella kaikista yleisimpiä [37]. Tämän tiedon valossa onkin tärkeää, että sekä sovelluksen kehittäjä että ylläpitäjä ovat ajan tasalla palvelun nykyisestä tilasta.

4.4.1 Tietoturvastrategia

Tietoturvan kannalta on tärkeää, että verkon ylläpitäjä tietää tarkoin kuinka verkko toimii ja sen valvonnassa hyödynnetään verkon analysointityökaluja. Näiden avulla verkon heikot kohdat pystytään tunnistamaan ja vahvistamaan. Käytössä tulee olla myös jonkinlainen suojausjärjestelmä, joka kirjaa ylös riittävän pitkältä ajalta ylös verkkoon tulevan ja lähtevän liikenteen, sillä esimerkiksi hajautettu palvelunestohyökkäys ei aina kaada koko verkkoa. Tällöin on tärkeää, että tapahtunut pystytään tarkasti analysoimaan ja löydetyt heikkoudet paikattua parhaalla mahdollisella tavalla [15].

Kun hyökkäys on meneillään, tulee käytetty hyökkäystapa tunnistaa mahdollisimman nopeasti. Useimmat hyökkäykset noudattavat tiettyä kaavaa ja näiden tunnistamiseen on olemassa useita valmiita työkaluja. Tämä on tärkeää, koska tunnistamisen jälkeen voidaan rajata, mistä hyökkäys on peräisin ja tehdä tarvittavat toimenpiteet hyökkäyksen torjumiseksi. Useimmissa tapauksissa tämä tarkoittaa yh-

teistyötä palveluntarjoajan ja viranomaisten kanssa. Viranomaisten mukaan tuominen on muutenkin tärkeää, koska vain tätä kautta voidaan käytettyä hyökkäysverkostoa lähteä tunnistamaan ja toivottavasti myös hajottamaan [15].

Jotta nopea reagoiminen olisi mahdollista, tulee organisaatiolla olla toimintasuunnitelma hyökkäyksen varalta. Ensimmäinen vaihtoehto lienee aina haitallisen liikenteen estäminen, mutta tätä varten tulee olla tehtynä tarkat prosessit kuinka toimitaan hyökkäyksen alettua. Tärkeää on sopia mitä dokumentoidaan ja kuinka asioista raportoidaan oikeille tahoille. Hyökkäyksen jälkeinen analyysi on myös riippuvainen sovituista käytännöistä. Tämä analyysi on erittäin tärkeää, sillä vain sitä kautta toimintasuunnitelmia voidaan kehittää oikeaan suuntaan [15].

4.4.2 Web-alustojen tietoturva

Aikaisemmin esitetyt seikat eivät tarkoita sitä, että Web-palveluita pyörittävät alustat olisivat turvassa hyökkäyksiltä. Onnistuneet hyökkäykset ovat edelleen yhtä tuhoisa, jos niihin ei varauduta ennalta. Erilaiset hyökkäykset pyrkivät yleensä hyödyntämään seuraavissa kategorioissa olevia heikkouksia

- Valmiit esimerkkitiedostot
- Lähdekoodin paljastuminen
- Kanonisointi
- Palvelimiin asennetut lisäosat
- Syötteen tarkistaminen [17].

Kanonisoinnilla tarkoitetaan sääntöjen mukaisen syötteen väärinkäyttöä. Hyökkäys pohjautuu siihen, että useimpia palveluita ja resursseja voidaan kutsua monin eri tavoin. Esimerkiksi tiedostoon `C:\text.txt` voidaan suhteellisesti viitatata syntaksilla `..\text.txt`. Sovellukset, jotka tekevät tietoturvapäätökset käyttäen hyödyksi resurssinimeä, voidaan helposti huijata suorittamaan odottamattomia toimintoja [17].

Listatuilta asioilta suojautuminen on melko yksinkertaista, kunhan noudattaa muutamia perussääntöjä. Ensinnäkin tuotannossa olevilla palvelimilla ei tulisi koskaan olla asennettuna tai käytettynä tiedostoja, joiden turvallisuudesta ei ole takeita. Näihin tiedostoihin lukeutuvat muun muassa paketeissa mukana tulevat esimerkkitiedostot ja palvelimelle asennettavat lisäosat, joiden alkuperästä ei ole varmuutta.

Toisekseen on tärkeä varmistaa, että käytettyihin sovelluksiin on asennettu viimeisimmät päivitykset, sillä ne yleensä korjaavat tunnetut heikkoudet [17]. Jo pelkästään näillä toimilla pystytään suurimmaksi osaksi estämään sellaiset hyökkäykset, jotka kohdistuvat itse käytettyyn palvelinalustaan. Spoofing- ja palvelunestohyökkäyksiltä nämä eivät suojaa, ja tästä syystä resursseihin kohdistuville hyökkäyksille tulee olla erilliset suojausmenetelmät.

4.4.3 Snort

Tietoturvahyökkäysten siirtyessä hiljalleen verkkokerrokselta sovelluskerrokselle, eivät perinteiset palomuurit pysty enää yksin takaamaan riittävää tietoturvatasoa. Tästä syystä näiden rinnalle on kehitetty erilaisia tietoturvasovelluksia, joista yleisimpiä ovat IDS- ja IPS-järjestelmät. Näistä tunnetuin ja käytetyin on vapaaseen lähdekoodiin perustuva Snort [38], jolla on rekisteröityneitä käyttäjiä lähes kolmesataatuhatta. Snortia on ladattua miljoonia kertoja ja sitä käyttävät niin yksityiset tahot kuin myös suuret yritykset ja eri valtioiden virastot. Suuren levinneisyyden takia Snortia kutsutaan usein IPS-järjestelmien alan de facto sovellukseksi.

Snort on pohjimmiltaan sääntöpohjainen IDS-järjestelmä eli se pyrkii tunnistamaan hyökkäykset vertaamalla tulevaa liikennettä ennalta kirjoitettuihin sääntöihin. Tämä tunnistaminen toteutetaan verkkotasolla tutkimalla jokainen tuleva paketti. Suuren käyttäjämääränsä ansiosta Snortin käyttämät säännöstöt ovat hyvin kattavat, ja yleisimpiin hyökkäyksiin kirjoitetaan nopeasti uudet säännöt. Snortiin on myös saatavilla useita lisäosia, joiden avulla on esimerkiksi mahdollista estää tietystä osoitteesta tuleva liikenne sekä tallentaa ja analysoida haluttu liikenne.

Koska Snortin lähdekoodi on vapaasti saatavilla, on tämän pohjalta kirjoitettu uusia järjestelmiä, joiden avulla pystytään muun muassa luomaan uusia sääntöjä automaattisesti verkkoliikenteestä [39] ja tutkimaan verkkoliikennettä erillisinä tapahtumaketjuina [40]. Näitä järjestelmiä kutsutaan usein hybridi IDS-järjestelmiksi, ja tällaisia löytyy useita erilaisia. Näitä ja muita sääntöpohjaisia IDS-järjestelmiä vaivaa kuitenkin sama ongelma, eli ne ovat täysin riippuvaisia oikeanlaisesta säännöstöstä. Jos tehtyyn hyökkäykseen ei löydy suoraan sääntöä, ei järjestelmä pysty tätä havaitsemaan. Hyökkäysten muuttuessa yhä monimutkaisemmiksi, on tästä muodostunut todellinen ongelma sääntöpohjaisille järjestelmille. Toinen ongelma IDS-järjestelmillä on niiden tapa luoda isoilla datamäärillä suhteellisen paljon virheellisiä tunnistuksia. Näistä rajoituksista huolimatta sääntöpohjaiset IDS-järjestelmät ovat tällä hetkellä suosituin tapa suojautua verkkohyökkäyksiltä.

4.4.4 Nikto

Nikto [41] on avoimeen lähdekoodiin perustuva ilmainen Web-skanneri, joka etsii Web-sivustoilta haavoittuvia CGI-skriptejä ja tiedostoja. Sen tietokanta käsittää yli 3500 tunnettua haavoittuvuutta sekä yli 900 eri Web-alustoihin liittyvää heikkoutta. Nikto on koodattu käyttäen Perliä ja se pyrkii skannaamaan palvelimen mahdollisimman lyhyessä ajassa. Tästä syystä sen aiheuttama liikenne on helppo havaita lokeista. Tämän kiertämiseksi Niktoon on lisätty mahdollisuus käyttää Whiskerin tarjoamaa libWhisker kirjastoa, joka pyrkii ohittamaan käytössä olevat suojausjärjestelmät. Nikto on erittäin tehokas ja käytetty työkalu perusturvallisuuden varmistamiseen, ja vuonna 2006 tehdyssä tutkimuksessa se valittiin parhaimmaksi Web–haavoittuvaisuuksia skannaavaksi sovellukseksi [42]. Nikton tietokantoja päivitetään hyvin satunnaisesti, joten nykyisellään se ei tunnista kaikista uusimpia hyökkäyksiä. Tästä huolimatta se on erittäin tehokas työkalu tietoturvan parantamiseen.

4.4.5 Nessus

Nessus [43] oli alunperin vapaaseen lähdekoodiin perustuva ilmainen tietoturvaskanneri, joka muuttui maksulliseksi vuonna 2008. 1200 dollarin vuosimaksua vastaan saa kuitenkin yhden parhaimmista UNIX- ja Windows-alustoilla toimivista tietoturvaskannereista [44]. Se sisältää yli 20000 erillistä lisäosaa, ja sen avulla on mahdollista suorittaa muun muassa reaaliaikaista liikenteen tarkkailua, konfigurointitiedostojen auditointia sekä eri verkko-osien skannausta. Maksullisuuden ansiosta Nessus pystytään pitämään ajan tasalla uusimmista hyökkäyksistä ja se soveltuukin erinomaisesti suurten ja monimutkaisten verkkojen turvaamiseen [43].

4.4.6 Paros Proxy

Paros Proxy [45] on Javalla kirjoitettu HTTP-välityspalvelin, jonka avulla voidaan arvioida Web-sovelluksen turvallisuutta. Se mahdollistaa kokonaisten sivustojen indeksoimisen, jonka lisäksi sen avulla on mahdollista tallentaa ja muokata reaaliajassa käyttäjän ja palvelimen välistä HTTP- ja HTTPS-liikennettä mukaan lukien evästeitä. Sen mukana tulee myös skanneri yleisimpien Web-haavoittuvuuksien tunnistamiseen. Paros pohjautuu vapaaseen lähdekoodiin ja se on hyvin käytetty työväline tietoturva-ammattilaisten parissa, jotka etsivät sivustoilta haavoittuvuuksia.

4.4.7 ModSecurity

Koska nykyisistä hyökkäyksistä yhä useampi kohdistuu verkon laitteiden ja resurssien sijasta Web-palveluihin ja -sovelluksiin [37] [31], eivät perinteiset tietoturvaratkaisut pysty enää tarjoamaan riittävää tietoturvatasoa. Tästä syystä sovelluskehittäjät ja ylläpitäjät ovat joutuneet etsimään uusia ja entistä tehokkaampia tapoja hyökkäysten tunnistamiseen ja torjumiseen. Yksi mahdollinen ratkaisu on lisätä erillinen, Web-sovelluksia varten suunniteltu palomuuri, käyttäjän ja palvelimen välille, jolloin palvelulle tulevat pyynnöt kulkevat tämän palomuurin kautta. Tällä tavoin saapuva liikenne voidaan tutkia ja suodattaa käyttäen haluttuja määrityksiä.

ModSecurity [46] on Apachelle suunniteltu, vapaaseen lähdekoodiin pohjautuva palomuuri, joka tarjoaa kattavan suojan Web-palveluihin kohdistuvilta hyökkäyksiltä. Se mahdollistaa HTTP-liikenteen monitoroinnin ja reaaliaikaisen liikenteen analyysin vaatien korkeintaan hyvin pieniä muutoksia jo olemassa olevaan arkkitehtuuriin. ModSecuritystä on saatavilla kaupallisia lisenssejä sekä tukipalveluita, mutta sovelluksen käyttämät perussäännöt ovat ilmaiseksi ladattavissa, tosin näitä ei ole hetkeen päivitetty. ModSecurityn käyttämä säännöstä mahdollistaa kuitenkin uusien sääntöjen kirjoittamisen omien tarpeiden mukaan, joten käytetyt säännöt voidaan räätälöidä tilannekohtaisesti [46].

ModSecurityn parhaimpiin ominaisuuksiin kuuluu mahdollisuus ylläpitää täydellistä lokia koko HTTP-tapahtuman ajalta, jolloin sekä pyynnöt että vastaukset pystytään analysoimaan. Tämän ansiosta myös tulevien POST-pyyntöjen sisältö ja rakenne voidaan tutkia. Nämä pyynnöt jäävät normaalisti tallentamatta lokiin, jolloin hyökkääjän kiinnijäämisen riski on pienempi. Suurin osa hyökkäyksistä toteutetaankin nykyisin POST-metodia käyttämällä. Se, mitä halutaan lokittaa, voidaan myös tarkkaan määrittää. Salattu ja pakattu liikenne ei myöskään aiheuta ongelmia, sillä lokitus tapahtuu sillä tasolla, jossa pyynnöt ovat valmiiksi puretussa muodossa [46].

Liikenteen kirjaamisen ohella ModSecurity voidaan määritellä estämään tulevia hyökkäyksiä, jonka lisäksi sen avulla pystytään nopeasti paikkaamaan löydetyt haavoittuvaisuudet. Ensinnäkin kaikki tulevat paketit voidaan pisteyttää halutulla tavalla, ja tietyn pisterajan ylittäneet paketit voidaan esimerkiksi pudottaa pois. Toinen mahdollisuus on sallia vain tietynlaiset pyynnöt. ModSecurityn käyttämä säännöstökieli mahdollistaa myös palveluiden nopean suojaamisen uusilta hyökkäyksiltä ilman, että palvelun omaan koodiin tarvitsee koskea. Tämä on erityisen hyödyllistä niissä tilanteissa, joissa päivitysten tuominen tuotantoon vie aikaa.

5 Dynaamiset Web-teknologiat

Internet on aina ollut vihamielinen paikka, jossa sivustolla vierailevan motiiveja sivustoa ja palvelua kohtaan on mahdotonta ennustaa. Tästä syystä useimmat kehittäjät ja ylläpitäjät ovat noudattaneet periaatetta, että kehenkään ei voi täysin luottaa. Sivustojen ja palveluiden kehittyessä entistä monimutkaisemmiksi, on tämän luottamattomuusperiaatteen noudattaminen noussut entistä tärkeämmäksi.

Webin teknisen monimutkaistumisen taustalla on dynaamisuuden lisääntyminen. Paljon sellaisia tehtäviä, jotka aiemmin olivat palvelimen vastuulla, on siirretty käyttäjän selaimessa suoritettavaksi. Lisäksi Web-sivujen sisältö ei ole enää staattista, vaan uutta sisältöä voidaan ladata sivulle käyttäen useitakin eri teknologioita. Vaikka kasvava määrä tietoturvahyökkäyksistä tapahtuu dynaamisten Web-teknologioiden avulla, ei se tarkoita sitä, että ne olisivat tietoturvan kannalta normaalia heikompia tai, että ne sisältäisivät helposti hyödynnettäviä haavoittuvaisuuksia. Dynaamisuus vain mahdollistaa aikaisempaa joustavamman alustan toteuttaa interaktiivisia ja normaalien työpöytäsovelluksien kaltaisia palveluita, joita käyttäjät voivat ajaa suoraan Web-selaimesta. Varsinainen ongelma piilee siinä, että dynaamiset Web-palvelut ovat aikaisempaa monimutkaisempia toteuttaa. Niissä hyödynnetään useita eri teknologioita ja rajapintoja, jonka johdosta mahdollisuus tehdä virheitä kasvaa verrattaessa vanhoihin ratkaisuihin. Käytetyt teknologiat eivät siis ole syynä heikentyneeseen tietoturvatasoon, vaan syynä on kehittäjien huolimattomuus tai tietämättömyys niistä mahdollisuuksista, joita huonosti suunnitellut ja toteutetut palvelut avaavat hyökkääjille.

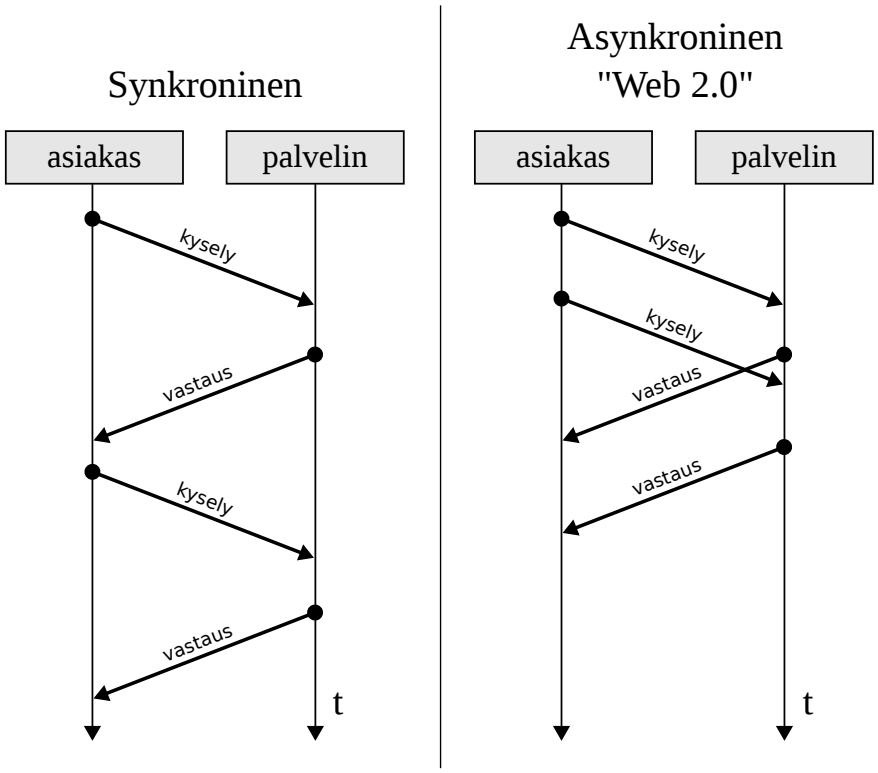
5.1 AJAX

Jos jokin dynaamisista Web-teknologioista halutaan nostaa kehitystä eniten eteenpäin vieväksi voimaksi, niin vahvin ehdokas tähän on Asynchronous JavaScript and XML (lyh. AJAX). Se itsessään ei ole mikään yksi tietty teknologia, vaan se on usean eri teknologian yhdistelmä, joita käyttämällä Web-sivut pystytään muuttamaan nopeasti reagoiviksi ja niiden käyttökokemus saadaan muistuttamaan perinteisiä työpöytäsovelluksia [47]. AJAX koostuu seuraavista teknologioista:

- HTML ja CSS rakentavat Web-selaimella näytettävän sivun.
- DOM mahdollistaa ajonaikaisen sisällön tuottamisen.
- XML ja Extensible Stylesheet Language Transformation (lyh. XSLT) mahdollistavat eri komponenttien välisen tiedonsiirron.
- JavaScript helpottaa eri komponenttien integroimista sekä näiden ohjelmoimista.
- XMLHttpRequest (lyh. XHR) -olio helpottaa kommunikointia palvelinten kanssa [27].

Suurin AJAXin tuoma muutos perinteisiin Web-sivuihin on mahdollisuus päivittää sivun sisältöä asynkronisesti ilman käyttäjän vuorovaikutusta. Yksinkertaistettuna tämä tarkoittaa sitä, että sivun sisällöstä pystytään päivittämään ainoastaan halutut osat käyttäen XHR-kyselyitä. Aikaisemmin tällainen ei ole ollut mahdollista, sillä vanhat Web-sivut toimivat pelkästään synkronisesti, jolloin erillisiä kutsuja ei voitu tehdä. Tällöin koko sivu täytyi päivittää kerralla. Tämä on ollut käytettävyyden kannalta huono asia, sillä sivu ei ole synkronisen latauksen ja käsittelyn aikana käytettävissä. XHR sisältää kaikki perinteiset HTTP-metodit mukaan lukien GET, POST, HEAD ja DELETE, joten sen avulla pystytään suorittamaan kaikki yleisimmät käyttäjän toiminnot [26]. Kuvassa 5.1 on esitetty tilanne, jossa sama palvelu on toteutettu käyttäen synkronisia ja asynkronisia kutsuja. Kuvasta voi havaita, että dynaamisuus mahdollistaa useamman samanaikaisen kutsun tekemisen, jolloin myös palvelun viiveet voidaan pitää lyhyinä.

AJAXiin kohdistuvat hyökkäykset eivät eroa suuresti vanhoista murtautumismenetelmistä. Hyökkääjät pyrkivät edelleen hyödyntämään syötteen puutteellista suodattamista, muokkaamaan ulos tulevaa dataa, murtamaan salauksia ja hankkimaan istuntokohtaisia tietoja esimerkiksi evästeitä varastamalla. Hyökkääjien näkökulmasta AJAXin tekee erityisen kiinnostavaksi se, että aiempaa suurempi osa Web-palvelun käyttämästä laskennasta suoritetaan käyttäjän selaimessa. Käyttäjän selaimeen, eli niin sanottuun asiakaspuoleen, ei voida kuitenkaan koskaan luottaa. JavaScriptien huolimaton käyttö avaakin hyökkääjille lukuisia tapoja murtautua palveluihin. Kyseessä on sinänsä pitkään tunnettu ongelma, mutta kehittäjien kiinnostuksen siirtyessä AJAXiin ovat myös hyökkääjät alkaneet kiinnittämään asiaan enemmän huomiota [47]. Monet käytetyistä AJAX-ympäristöistä sisältävät myös eriasteisia tietoturvariskejä, joita hyökkääjät voivat hyödyntää [48].



Kuva 5.1: Synkroninen ja asynkroninen kommunikointi.

5.1.1 JavaScript

JavaScript on Netscapen suunnittelema skriptikieli, joka tunnettiin aluksi nimellä LiveScript. Sun Microsystemsin kehittämän Java-ohjelmointikielen yleistyessä Netscape päätti muuttaa sen nimen JavaScriptiksi toivoen sen käytön yleistyvän Javan menestyksen myötä. JavaScriptillä kirjoitetut skriptit sijoitetaan HTML:n `<script>`-elementin sisään, josta voi olla myös viite erilliseen lähdekooditiedostoon. JavaScript vaatii tuen selaimelta, jollainen löytyy miltei jokaisesta Web-selaimesta mukaan lukien yleisimmät mobiililaitteet.

JavaScript on tehokas työkalu, joka mahdollistaa monen muun toiminnon ohella muun muassa dynaamisten Web-sivujen tekemisen, viestien kirjoittamisen selaimen tilariville, uusien ikkunoiden avaamisen ja interaktiivisten lomakkeiden luomisen. Skriptien toiminta rajoittuu kuitenkin Web-selaimen toimintoihin, eikä sitä käyttäen voida esimerkiksi kirjoittaa levylle ja levyn lukeminen rajoittuu pelkästään evästeisiin [49]. Näistä rajoituksista huolimatta JavaScript mahdollistaa erittäin monipuolisten toimintojen toteuttamisen Web-ympäristössä.

Suurin osa Web-sivuilla olevista skripteistä toimii siten, että ne ladataan aluksi

käyttäjän koneelle, jonka jälkeen ne suoritetaan selaimessa. Skriptit sijaitsevat ja ne suoritetaan käyttäjien koneilla, joten toimintaperiaatteen selvittäminen ei ole hyökkääjälle vaikeaa. Asiaa pystytään kuitenkin vaikeuttamaan muutamalla eri tavalla. Näistä yksinkertaisin on *obfuskointi*, jonka yhteydessä koodista tehdään vaikeasti luettavaa. Aluksi poistetaan kommentointi ja tyhjät rivivälit, seuraava askel on koodin muuttujien ja objektien uudelleen nimeäminen, jolloin hyökkääjän on vaikeampi päätellä näistä skriptin toimintaperiaate. Nimet voidaan myös muuttaa esimerkiksi binäärikoodiksi. Nämä keinot ovat kuitenkin vain hidasteita osaavalle hyökkääjälle [47]. Palvelu tulisikin suunnitella siten, että kaikki luottamuksellisen tiedon käsittely ja käyttäjän syötteen tarkistukset suoritettaisiin Web-palvelimella. Tällöin käyttäjän selaimessa suoritettaisiin ainoastaan sellainen laskenta, joissa käsitellään käyttäjää itseään koskevaa dataa.

Koneelle ladatut skriptit ajetaan oletuksena hyvin rajatussa tilassa, jossa niillä on pääsy vain kyseistä sivun sisältämään dataan tai siihen hyvin läheisesti kuuluviin dokumentteihin. JavaScript noudattaa myös edellisessä luvussa esitettyä *saman alkuperän* periaatetta, joka estää skriptien suorittamisen muista Internet-domaineista kuin siitä, josta skripti on ladattu. Näin vaikeutetaan hyökkäyksen tekemistä käyttäen toiselta sivulta ladattua haitallista skriptiä, jonka avulla voitaisiin esimerkiksi varastaa evästeitä ja urkkia näppäimistön painallukset. Aikaisemmin selaimet ovat sallineet joitain poikkeuksia tähän periaatteeseen, mutta tietoturvasyistä nämä on kuitenkin poistettu.

Liian tiukat tietoturvasäännöt eivät kuitenkaan toimi nykyaikaisten Web-palveluiden kanssa, ja tästä syystä *saman alkuperän* periaatetta on mahdollista löysentää. Esimerkiksi ulkoiset linkit, jotka osoittavat toisella sivustolla olevaan skriptiin, ohittavat *saman alkuperän*, sillä vaikka itse skripti sijaitsee toisella palvelimella niin se lasketaan kuuluvan siihen sivuun, jossa viittaus lähdekooditiedostoon on. Näin kutsutulla skriptillä on pääsy sellaisiin evästeisiin ja tiedostoihin, joihin sillä ei välttämättä tarvitsisi olla oikeuksia. Sivustojen ja palveluiden käyttäessä resursseja entistä useammasta lähteestä eri palvelimilta, voi tämä aiheuttaa riskitilanteita, jos jonkin lähteen tietoturva on vaarantunut [47].

5.1.2 AJAXiin liittyvät tietoturvariskit

Koska AJAXin toiminta perustuu hyvin pitkälti JavaScriptin varaan, on itsestään selvää, että hyökkääjät pyrkivät väärinkäyttämään ensisijaisesti JavaScriptin haavoittuvaisuuksia. Asiaa ei auta se, että suurin osa sivustoista on jollakin tavalla alt-

tiina XSS-hyökkäyksille [28] johtuen huonosti toteutetusta syötteen tarkistuksesta. Asetettuja suodatuksia pystytään myös kiertämään käyttäen esimerkiksi kuvalinkkejä ja monia HTML-elementtejä, joiden avulla ajettava skripti voidaan hukuttaa muun syötteen joukkoon. Tästä syystä pelkästään tiettyjen merkkijonojen suodattaminen ei takaa suojaa JavaScript-pohjaisilta hyökkäyksiltä. Tehokkain ja yksinkertaisin keino onkin poistaa kaikki HTML-elementteihin kuuluvat merkit, jolloin esimerkiksi <-merkistä tulee `<` ja >-merkistä `>`. Monet ympäristöt mahdollistavat myös suoraan HTML-elementtien poistamisen käyttäjien jättämistä viesteistä [47]. Tällöin esimerkiksi foorumeilla toisten käyttäjien lähettämistä viesteistä saadaan estettyä niiden mahdollisesti sisältämän JavaScript-ohjelmakoodin suorittaminen.

Kasvaneella JavaScriptin käytöllä on suora vaikutus myös XSS-hyökkäysten yleistymiseen. JavaScriptillä pääsee helposti evästeisiin käsiksi käyttäen *document.cookie*-oliota. Vaikka sen käyttö on rajoitettu ainoastaan sen Internet-domainin evästeeseen, josta kutsu on tehty, voidaan tämä rajoitus ohittaa melko helposti. Hyökkääjälle riittää, että käyttäjä vierailee esimerkiksi sellaisessa foorumissa tai blogissa, joissa käyttäjien viesteissä sallitaan XHTML:n käyttö. Tämä mahdollistaa haitallisten skriptien lähettämisen sivustolle ja pahaa aavistamaton käyttäjä, joka vierailee tällä sivulla, joutuu tietämättään hyökkäyksen kohteeksi.

Yksi tapa suojautua evästeiden varastamiselta on käyttää HTTP-only -evästeitä, jolloin käyttäjien evästeitä ei pystytä lukemaan JavaScriptillä. Näiden evästeiden tukeminen on kuitenkin selainkohtaista, joten tämä ratkaisu ei aina välttämättä ole mahdollista toteuttaa. XSS-hyökkäykset eivät myöskään rajoitu pelkästään evästeiden varastamiseen, ja yhtä helposti hyökkääjä voi esimerkiksi luoda skriptin, joka lukee näppäimistön painallukset ja lähettää ne haluttuun osoitteeseen. HTTP-only -evästeiden käyttö ei myöskään täysin suojaa evästeitä, jos käytetään XHR-kutsuja. Tällöin hyökkääjän on mahdollista lukea suoraan otsaketiedoissa asetettuja arvoja. Jotkut XHR-objekteihin liittyvät haavoittuvuudet ovat myös niin syvällä XHMLHttpRequest-objekteissa, että kaikki nykyisin käytössä olevat toteutukset ovat niille jossakin määrin alttiina. Kaapattujen XHR-objektien tunnistaminen ei myöskään vielä ole mahdollista [47]. Näistä syistä johtuen AJAX tarjoaa nyt ja tulevaisuudessa hyökkääjille monia mahdollisuuksia murtaa asetettuja suojauksia.

Yksi yleisimmistä AJAX-ohjelmoinnissa käytetyistä tiedon esitystavoista on JavaScript Object Notation (lyh. JSON). Se on ohjelmointikielestä riippumaton ja laajasti tuettu Web-palvelimella käytetty työkalu. Se on rakenteeltaan hyvin kevyt ja se

koostuu kahdesta eri tietotyypistä: olioista ja taulukoista [50]. Formaatin heikkous piilee siinä, että JSON-olio itsessään on kelvollinen JavaScript-ohjelma, jonka sisältö on mahdollista kaapata [47]. Termi *JavaScript Hijacking* kuvaa tällaista tilannetta, jossa hyökkääjä ohittaa *saman alkuperän* periaatteen silloin, kun JavaScriptiä käytetään arkaluontoisen tiedon lähettämiseen. Aiheesta tehdyn tutkimuksen [48] mukaan lähes jokainen käytetty AJAX-ympäristö on alttiina tällaisilla hyökkäyksille. Käyttäen CSRF-hyökkäystä hyökkääjä pystyy väärinkäyttämään JSON-formaattia, ja varastamaan sekä muokkaamaan toiselle sivustolle lähetettäviä paketteja. Tällainen hyökkäys voidaan toteuttaa useilla eri tavoilla mukaan lukien käyttäen `<script>`-elementtiä, ja koska pyyntö tulee selaimen luottamasta lähteestä, voidaan tällä tavoin ohittaa esimerkiksi SSL-suojaus [47].

Suojautuminen tämän tyyppisiltä hyökkäyksiltä voidaan toteuttaa monella eri tapaa, ja paras tulos saadaan yhdistelemällä näitä. Koska CSRF-hyökkäyksessä hyökkääjä joutuu toimimaan osittain sokeasti, voidaan jokaiseen pyyntöön lisätä parametri, jota hyökkääjän on vaikea arvata. Palvelin voidaan myös asettaa tarkistamaan HTTP Referer -kenttä, jolla voidaan varmistaa, että pyyntö tulee sallitulta taholta, vaikkakin Referer-kentän sisältö voidaan helposti muuttaa. Toinen tapa on muokata vastaanottopäässä paketteja siten, että hyökkääjä ei pysty ajamaan lähettämissään pyynnöissä skriptejä [48].

5.2 Flash

AJAXin ohella Macromedian suunnittelema, ja nykyisin Adoben kehittämä Flash, on ollut yksi niistä merkittävistä tekniikoista, joita käyttämällä YouTuben ja MySpacen kaltaiset menestyspalvelut ovat olleet mahdollisia toteuttaa. Flashia käyttämällä kehittäjät ovat jo pitkään pystyneet luomaan interaktiivista ja rikasta Web-sisältöä yksinkertaisista animaatioista aina monimutkaisiin peleihin ja valikoihin asti. Suurin osa Web-sivustoilla olevista mainoksista on myös toteutettu käyttäen Flashia. Näistä syistä johtuen Flash Player on yksi ohjelmistoalan de facto -tekniikoista, jonka käyttöaste yritys- ja kuluttajapuolella on lähes 100 prosenttia [51].

Flashin käyttämä ActionScript-skriptikieli muistuttaa läheisesti JavaScriptiä ja sitä käyttäen on mahdollista luoda uusia TCP-yhteyksiä, ajaa JavaScriptiä selaimessa ja luoda sallittuihin domaineihin HTTP-pyyntöjä. Flashin käyttämä tietoturvamalli noudattaa paljolti *saman alkuperän* periaatetta, jolla rajoitetaan eri domaineissa sijaitsevien sovellusten kommunikointia. Domainien välinen kommunikointi on

kuitenkin sallittua, ja käytetty tietoturvapolitiikka luetaan tähän tarkoitetusta XML-tiedostosta. Tämä XML-tiedosto sijaitsee usein domainin juuressa ja se sisältää ne domainit, jotka saavat olla siihen yhteydessä. Osa Flashiin kohdistuvista hyökkäyksistä pyrkiikin muokkaamaan tietoturvapolitiikkatiedoston sisältöä siten, että kohde sallisi yhteydet myös hyökkääjän käyttämästä osoitteesta. Tämä voidaan toteuttaa esimerkiksi käyttämällä vihamielistä RSS-syötettä tai luomalla tiedosto, joka sisältää uuden XML-tiedoston. Yksi tällaisista hyökkäyksistä on Stefan Esserin luoma GIF-tiedosto, jonka kommenttiin on piilotettu uusi tietoturvapolitiikka. Hyökkääjälle riittää, että kohdekone vain mahdollistaa tiedoston lataamisen palvelimelle, jonka jälkeen vanha sisältö voidaan korvata GIF-tiedostoon piilotetulla [26].

Perimmäinen syy sille, miksi XSS-hyökkäykset ovat yleistyneet viime vuosina on se, että käyttäjien antamaa syötettä ei usein tarkisteta riittävällä tarkkuudella. Sama pätee myös suureen osaan Flash-pohjaisista toteutuksista, joihin käyttäjien on mahdollista antaa syötteitä. Tämä avaa useita erilaisia Flashia hyödyntäviä XSS-hyökkäyksiä, jotka pyrkivät väärinkäyttämään käyttäjän selaimen ja sivuston välistä luottamussuhdetta. Aina vika ei ole Flash-toteutuksen tekijässä, sillä Flash-tiedostoja luodaan usein automaattisesti käyttämällä valmiita sovelluksia. Aiheesta tehdyn kattavan tutkimuksen mukaan [52] suurimmasta osasta näin luoduista tiedostoista löytyy XSS-haavoittuvaisuuksia, jotka mahdollistavat JavaScriptin ajamisen siinä domainissa, jossa haavoittuvainen SWF-tiedosto sijaitsee. Osa näistä tietoturvaongelmista on myöhemmin korjattu tutkimuksen julkaisun jälkeen, mutta tehty tutkimus osoittaa sen, että edes luotettavina pidettävien tahojen toteutuksiin ei pidä luottaa liikaa.

Yksi käytetyimmistä Flash-pohjaisista toteutuksista ovat erilaiset interaktiiviset mainokset, joita löytyy lähes jokaiselta sivustolta. Flashin levinneisyyden ansiosta ne toimivat miltei jokaisella käyttäjällä ja siksi ne ovat houkutteleva keino toteuttaa hyökkäyksiä. Adobe Flash Playerista on löydetty useita eri haavoittuvaisuuksia, jotka ovat mahdollistaneet haitallisten koodien ajamisen kohdekoneessa. Flash-mainoksia käytetään myös laajalti haittaohjelmien levittämiseen, ja usein tällä tavalla altistuneet koneet kuuluvat käyttäjän tietämättä bottiverkkoihin, joita käytetään roskapostin lähettämiseen ja palvelunestohyökkäyksiin. Haitallista koodia sisältävä mainos voi myös esimerkiksi pyrkiä ohjaamaan selaimen toiselle sivustolle käyttäen ActionScriptia. Tällaiset haitalliset mainokset eivät ole pelkästään vain pienten sivustojen ongelma, sillä vuonna 2009 suuret sivustot kuten *guardian.co.uk* sisälsivät mainoksia, jotka olivat luonteeltaan haitallisia [53].

5.3 Google Web Toolkit

Nykyisin yhä useampi taho pyrkii hyödyntämään tekniikoita kuten AJAXia ja JavaScriptiä omissa Web-palveluissaan, ja tulevaisuudessa tämä kasvu tulee vain jatkumaan. Toimivien ja turvallisten Web-palveluiden suunnitteleminen ja toteuttaminen vaatii kuitenkin sellaista syvällistä tietämystä käytetyistä tekniikoista, jota harvemmin suunnittelijoilta löytyy. Tämä on yksi niistä pääsyistä, jonka takia Web-pohjaiset hyökkäykset ovat kasvaneet määrällisesti suurimmaksi hyökkäystavaksi. Asian helpottamiseksi on kehitetty erilaisia kehitysympäristöjä, jotka pyrkivät helpottamaan suunnittelijoiden taakkaa, ja samalla tarjoamaan jonkinlaista tietoturvaa. Yksi tällaisista on Googlen kehittämä Google Web Toolkit, joka on kasvamassa yhdeksi käytetyimmistä uusista kehitysympäristöistä.

5.3.1 Google Web Toolkit lyhyesti

Google Web Toolkit (lyh. GWT) on avoimeen lähdekoodiin perustuva kehitysympäristö rikkaiden ja dynaamisten Web-sovellusten kehittämiseen. Suunnittelun lähtökohtana on ollut mahdollistaa korkealaatuisten Web-sovellusten tehokas suunnittelu ilman, että kehittäjän tulee olla asiantuntija niin selainten pienissä kummallisuuksissa kuin myös XMLHttpRequest-pyyntöjen tekemisessä ja JavaScriptien kirjoittamisessa. Tämä on mahdollista, koska GWT:llä AJAX-sovellukset kirjoitetaan Javalla, joka mahdollistaa keskittymisen korkeamman tason suunnitteluun ilman, että aikaa tuhlaantuisi DOM- ja XHR-kutsujen säätämiseen. Valmis Java-sovellus käännetään automaattisesti JavaScriptiksi, ja samalla skriptien sisältö optimoidaan muun muassa poistamalla käyttämättömät muuttuja ja parametrit. Tämän ratkaisun etuna on myös se, että kehittäjät voivat käyttää haluamaansa Java-ympäristöä sovellusten kirjoittamiseen ja koodin tarkistamiseen. Käännetyt JavaScriptit myös toimivat suoraan suurimmassa osassa selaimista sekä Androidissa ja iPhonessa [54].

5.3.2 Google Web Toolkitin ominaisuuksia

GWT 2.0 on uusin versio ympäristöstä ja se on tuonut mukanaan suuren joukon uusia ominaisuuksia vanhoihin versioihin verrattuna. Yksi tärkeimmistä on niin kutsutun "hosted moden" korvaaminen *GWT Developer* liitännäisellä. Hosted mode oli yksi GWT:een kantavista ideoista, joka mahdollisti tietyn selainympäristön emuloimisen. Hosted modea käyttäen kehittäjällä oli mahdollisuus kirjoittaa Java-

koodia ja tutkia sen toimivuutta Web-selaimessa ilman, että koodia piti kääntää JavaScriptiksi. Tämä säästi reilusti kehittäjän aikaa, sillä ison koodin kääntäminen on aina hidasta. Tämän ratkaisun ongelmana oli se, että emuloidut ympäristöt edustivat vanhoja selainversioita, eikä selaimille jo valmiiksi kirjoitettuja testausohjelmia kuten Firebugia pystynyt käyttämään. Käyttäen Developer pluginia kehittäjien on nyt mahdollista ajaa ja testata koodia suoraan suosituimmilla selaimilla, jolloin esimerkiksi tyylitiedostojen säätäminen on helpompaa. Koodin testaaminen yhtäaikaisesti useammalla eri selaimella on myös mahdollista [55].

Toinen suuri uudistus GWT 2.0:ssa on mahdollisuus pilkkoa ajettava koodi pienempiin osiin (Code Splitting). Nykyisten AJAX-sovellusten ongelmana on se, että sivun sisältö pitää lähes aina ladata kokonaan selaimeen, ennen kuin palvelua voidaan käyttää. Sivustojen kasvaessa entistä suuremmiksi tämä kasvattaa käyttäjien kokemaa viivettä, ja yksi AJAXin eduista on juuri viipeen vähentäminen. Pilkkomalla ajettava komentosarja pienempiin osiin käyttäen valmista funktiota, voidaan tämä ongelma välttää kokonaan. Riittää, että kehittäjä määrittää ne kohdat, jotka hän haluaa ladattavan pienemmissä osissa. Tällä tavoin voidaan esimerkiksi ladata ensiksi käyttöliittymä ja yleisimmin käytetyt valikot, ja vasta käytön aikana ladata vähemmän käytetyt toiminnot. GWT pitää myös itse huolen siitä, että kaikki tarvittavat riippuvuudet ladataan oikessa järjestyksessä [55]

5.3.3 Google Web Toolkitin tietoturva

Koska GWT:n lopputuotos on JavaScriptiä, on se periaatteessa samalla tavalla haavoittuvainen hyökkäyksille kuten Cross-Site Scriptingille ja Cross-Site Request Forgerylle. Yleisellä tasolla GWT:een tuottama JavaScript-komentosarja noudattaa hyväksi todettuja käytäntöjä, jotka vähentävät riskiä joutua tällaisten hyökkäysten kohteeksi. On kuitenkin asioita, joihin kehittäjien tulee edelleen kiinnittää tarkkaa huomiota.

Ensimmäinen näistä on olla käyttämättä omia ja kolmannen osapuolen skriptejä sekaisin. Tiettyjen toimintojen kuten innerHTML:n ja eval-funktion kanssa tulee myös olla erittäin tarkkana. Esimerkiksi innerHTML-atribuuttia käytetään usein staattisen HTML-sisällön tuomiseen JavaScript-objektin määrittämiin valmiisiin taulukoihin ja ikkunoihin. Tämän tekniikan käyttäminen on kuitenkin riskialtista, sillä se mahdollistaa haitallisen koodin sisällyttämisen suoraan sivustolle. Mihinkään syötteeseen ei pidä myöskään koskaan luottaa suoraan, vaikka se tulisi omalta palvelimelta erityisesti jos se on menossa innerHTML- tai eval-funktioiden käyttöön.

Sama pätee myös JSON-syötteisiin, joiden avulla on mahdollista suoraan ajaa haitallista JavaScriptiä, jos syötettä ei tarkisteta riittävällä tarkkuudella. Tärkeintä onkin aina tarkistaa erittäin tarkasti eri syötteet riippumatta siitä, mikä osapuoli sen on lähettänyt ja mihinkä käyttöön se on menossa [56].

CSRF-hyökkäykset kohdistuvat aina palvelinpuolen huonosti toteutettuun sessionhallintaan, jossa pyynnön tehneen tahon oikeellisuutta ei varmisteta. Määrittämällä JavaScript kopioimaan aina sessiossa käytetyn evästeen arvo jokaiseen kyselyyn mukaan, voi palvelin verrata tätä arvoa luomaansa evästeeseen. Koska *saman alkuperän* periaate estää kolmannen osapuolen sivustoa pääsemästä käsiksi tähän evästeeseen, voi palvelin olla varma, että pyynnön on todellakin tehnyt käyttäjä itse. GWT:stä löytyy valmiiksi tähän tarkoitukseen kirjoitettuja funktioita, joita käyttäen jokaiseen pyyntöön voidaan lisätä oikean evästeen arvo. Näitä käyttäen CSRF-hyökkäykset voidaan lähes kokonaan poistaa, kunhan vain palvelinpuoli osaa verrata evästeiden arvoja ja tehdä näiden pohjalta oikeat johtopäätökset. JSONin tapauksessa voidaan tämän lisäksi käyttää jo aikaisemmin esitettyä tapaa, jossa pakettien sisältöä muokataan siten, ettei niiden sisältöä voida kaapata käyttäen `<script>`-elementtiä [56].

6 Aiheeseen liittyvä tutkimus

Tässä luvussa tutustutaan tietoturvahyökkäyksien tunnistamiseen liittyvään tutkimukseen.

6.1 Väärinkäytösten tunnistaminen

Väärinkäytökseen perustuvat järjestelmät ovat pitkään olleet suosituin lähestymistapa tietoturvahyökkäysten torjumiseen. Nämä järjestelmät voidaan jakaa kahteen erilliseen osaan, jossa tilattomassa järjestelmässä jokaista tulevaa tapahtumaa käsitellään itsenäisesti, kun taas tilallisessa järjestelmässä tutkitaan tapahtumien välisiä suhteita. Web-pohjaisten hyökkäysten tutkimisessa tietolähteinä on käytetty esimerkiksi palvelimien tuottamaa lokia [57], ja IDS-järjestelmien [38] [58] tapauksessa analysoimalla verkkokerroksen liikennettä. Ensimmäisessä ratkaisussa ongelmana on, että itse palvelimiin kohdistuvia hyökkäyksiä ei pystytä havaitsemaan ainoastaan lokitietoa tarkkailemalla. Samoin hyökkäyksiä, jotka koostuvat monista eri vaiheesta, on mahdotonta mallintaa. Verkkokerroksella toimivia järjestelmiä taas pystytään harhauttamaan muokkaamalla hyökkäyksiä, ja näistä harvat mahdollistavat tilallisen analyysin.

Esitettyjä ongelmia on pyritty ratkaisemaan lisäämällä havaittavien tapahtumien määrää, ja keräämällä informaatiota eri lähteistä. Esimerkiksi asentamalla sovellustasolle erillinen tiedonkeruuseen tarkoitettu komponentti [59] on saatu hyviä tuloksia. Tässä tapauksessa tiedonkeruu tapahtui Apache-palvelimelle asennetun komponentin välityksellä, joka tarkkaili lokitiedon lisäksi pyyntöjen tulkitsemiseen ja toteuttamiseen menevää aikaa. Tällaisen ratkaisun hyvänä puolena on se, että tutkittava data on salaamattomassa muodossa, ja sessioiden uudelleenrakentaminen on mahdollista. Tällä tasolla toimiva IDS-järjestelmä voi myös toimia ennaltaehkäisevästi eli haitalliseksi havaittu liikenne voidaan tiputtaa pois ennen kuin se käsitellään palvelimella. Suurimpana haittapuolena on, että tietylle sovellukselle suunniteltua komponenttia ei voida sellaisenaan käyttää muilla alustoilla.

WebSTAT [60] on tilalliseen analyysiin perustuva IDS-järjestelmä, joka pohjautuu STAT-kehitysympäristöön [61]. WebSTAT hyödyntää STATL-ohjelmointikieltä,

joka mahdollistaa hyökkäysten mallintamisen ottamalla huomioon eri tapahtumien välisiä yhteyksiä, verkkohistoriaa sekä palvelimien kuten Apachen ja Microsoftin IIS:n lokia. Koska tietoa kerätään yhtäaikaisesti monesta eri lähteestä, voidaan palvelimien lokitiedon analysointiin yhdistää alempien toimintojen kuten järjestelmä- ja verkkotason tuottamaa tietoa. Näin tehty analyysi kuvaa koko järjestelmä tilaa, jolloin yllättävät muutokset pystytään havaitsemaan nopeasti. Testien mukaan tällainen analyysi voidaan toteuttaa suurissa järjestelmissä reaaliajassa aiheuttamatta suurempaa viivettä palvelimien toimintaan. Menetelmän sovittaminen tiettyyn järjestelmään vaatii jonkin verran manuaalista työtä, ja tätä voidaankin pitää sen suurimpana heikkoutena. Monimutkaisten hyökkäysten kuvaaminen on usein myös hankala toteuttaa ja niiden tulkitsemiseen saattaa tuhlaantua turhaa aikaa.

Väärinkäytösten tunnistamiseen tarkoitettuja järjestelmiä on hyödynnetty myös uudempien tietoturvahyökkäysten tunnistamisessa. Esimerkiksi vihamielisten Flash-mainosten tunnistamiseen tarkoitettu OdoSwiff [53] pyrkii etsimään sivuilla olevista mainoksista hyökkäykseen tarkoitettua koodia käyttäen staattista ja dynaamista analyysia. Palvelinpuolella XSS-hyökkäyksiä vastaan on luotu järjestelmä, josta löytyy yleisimpien hyökkäysten kuvaukset [62]. Kumpikin järjestelmistä toimii erittäin hyvin, kunhan hyökkäys on entuudestaan tuttu.

6.2 Poikkeavuuksien tunnistaminen

Väärinkäytösten tunnistamiseen käytettyjen järjestelmien suurin heikkous piilee siinä, että mallintamattomat hyökkäykset jäävät näiltä huomaamatta. Tämä on erityisesti Web-palveluihin kohdistuvien hyökkäysten tapauksessa iso ongelma, sillä toimintaympäristö muuttuu jatkuvasti. Uusia hyökkäyksiä ja vanhojen hyökkäystyyppien variaatioita ilmestyy tiuhaan tahtiin. Tällöin signatuurien pitäminen ajan tasalla muodostuu mahdottomaksi tehtäväksi. Hyökkäysten monimuotoisuus onkin johtanut siihen, että nykyisin yhä useammassa järjestelmässä pyritään tunnistamaan poikkeavat tapahtumat normaaliin liikenteen seasta ilman tarkkoja signatuureja.

Anomalioiden tunnistamiseen on käytössä useita eri menetelmiä ja ne voidaan jakaa kahteen eri ryhmään [63]. Näistä ensimmäinen sisältää oppimiseen pohjautuvat mallit, jossa normaali käyttäytyminen opetetaan opetusmateriaalin avulla. Normaalia käyttäytymistä esittävät profiilit voidaan mallintaa käyttäen joko sääntöpohjaista-, mallipohjaista- tai tilastopohjaista menetelmää. Näistä sääntöpohjaiset mene-

telmät muistuttavat eniten perinteisiä IDS-järjestelmiä sillä erolla, että luodut säännöt pohjautuvat kerättävään dataan, ja säännöt voivat olla rakenteiltaan hyvin monimutkaisia. Mallipohjaisissa menetelmissä taas luodaan normaalia käyttäytymistä kuvaavat profiilit, jota vastaan tuleva liikenne arvioidaan. Tiedonlouhinta, neuroverkot ja liikenteestä luotujen kuvioiden vertaaminen tulevaan liikenteeseen (engl. *pattern matching*) ovat tekniikoita, joita on käytetty tällaisten mallien luomiseen. Analyysissa voidaan esimerkiksi tutkia verkkopakettien kuormia [64] [65] ja klusteroimalla ja luokittelemalla liikenne protokollien ja palveluiden mukaan [66]. Viimeisen ryhmän muodostavat tilastollisiin menetelmiin pohjautuvat menetelmät [67], jotka ovat jääneet vähemmälle huomiolle johtuen alati muuttuvasta toimintaympäristöstä.

Toisen ryhmä anomalioiden tutkimisessa muodostavat spesifistiset mallit (engl. *specification model*). Nämä menetelmät pohjautuvat enemmän ihmisten huomioihin ja asiantuntijuuteen kuin matemaattisiin kuvauksiin. Menetelmissä käytetään useita eri elementtejä aina sovellustasolta verkkotasolle, ja näitä käyttäen luodaan normaalia käyttäytymistä kuvaavat mallit. Järjestelmät voivat hyödyntää esimerkiksi protokollista kerättävää tietoa anomalioiden tunnistamisessa. Järjestelmien eri tiloja ja tapahtumien välisiä suhteita voidaan myös mallintaa, jolloin poikkeavat tilat ja tapahtumaketjut voidaan tunnistaa.

Anomalioiden tunnistamiseen perustuville järjestelmille löytyy useita eri käyttökohteita. Niillä voidaan esimerkiksi pyrkiä tunnistamaan tietokantoihin kohdistuvia tunnettuja ja tuntemattomia SQL-hyökkäyksiä. Järjestelmälle voidaan opettaa normaali käyttäytyminen esimerkiksi tiedonlouhintamenetelmin [68] tai luomalla profiileja normaalista käyttäytymisestä [69] [70]. Erilaisia menetelmiä voidaan myös yhdistellä, jolloin todennäköisyys poikkeavan liikenteen tunnistamiseen kasvaa. Tutkimalla esimerkiksi yhtä aikaisesti sekä HTTP-pyynnöissä että SQL-kyselyissä ilmeneviä poikkeavuuksia, voidaan tulevat kyselyt pisteyttää tarkasti [71]. Kyselyiden kategorisointi mahdollistaa sen, että haitalliseksikin merkityt kyselyt voidaan ohjata sellaisille palvelimille, joilla ei ole pääsyä arkaluontoiseen tietoon.

Web-palveluihin kohdistuvien hyökkäysten tunnistaminen on hankala ja aikaa vievä prosessi. Aikaisemmin tässä työssä esitetyt hyökkäykset kattavat vain osan hyökkäyksistä, joita vastaan sovellussuunnittelijat ja ylläpitäjät joutuvat suojautumaan. Juuri tämä hyökkäysten monimuotoisuus on se seikka, joka on nostanut anomaliatutkimuksen muiden menetelmien yläpuolelle, ja aihepiiri on viime vuosina noussut yhdeksi puhutuimmista tietoturvan saralla.

Poikkeavan tilan tunnistamiseen on käytetty monia eri menetelmiä, ja päätöksen tekemiseen on käytetty useita eri tietolähteitä. Esimerkiksi Swaddler [72] on Web-sovelluksille suunniteltu menetelmä, joka oppii kriittisten järjestelmäkutsujen ja sovelluksen tilojen väliset suhteen analysoimalla Web-sovelluksen sisäisiä tiloja. Tällä tavoin voidaan tunnistaa hyökkäykset, jotka aiheuttavat poikkeavia tiloja esimerkiksi sovelluksen normaaliin työkulkuun. Järjestelmä koostuu eri malleista ja osista, joille opetetaan harjoittelujakson aikana sitä vastaava normaali käytös. Jokaiselle osalle, jotka käytännössä vastaavat tiettyjä sovelluksen toimintoja, lasketaan myös kynnysarvo, jonka ylittyessä sen aiheuttanut toiminto lasketaan anomaliaksi. Tehdyissä testeissä järjestelmä tunnisti kaikki toteutetut hyökkäykset, ja virheellisten positiivisten hälytysten määrä oli kohtuullisen pieni. Analysointi aiheutti jonkin verran kuormaa palvelimelle, mutta optimoimalla toteutusta tämä voidaan poistaa lähes kokonaan.

Aikaisemmin esitettyä tapaa, jossa hyödynnetään palvelimen tuottamaa lokia, voidaan käyttää myös poikkeavuuksien tunnistamisessa [73]. Tässä tapauksessa analysoitiin Apachen tuottamaa HTTP-lokia, ja huomio kiinnitettiin kyselyihin, joissa parametreja käyttäen välitettiin arvoja palvelinpuolen ohjelmille tai aktivoitiin dokumentteja. Kyselyt purettiin osiin ja niitä analysoitiin käyttäen useita eri malleja muun muassa kyselyjen pituutta ja normaalia järjestystä, merkkien jakaumia ja pyyntöjen tiheyttä. Vastaavaa mallia on sovellettu myös poikkeavien järjestelmäkutsujen tunnistamisessa [74], joten sen käyttö ei rajoitu pelkästään Web-palveluihin. Väärinkäytös- ja anomaliamenetelmien yhdistämistä on myös ehdotettu [75]. Tällainen järjestelmä voi toimia siten, että tuleva liikenne syötetään ensiksi anomalioita tutkivalle järjestelmälle, ja vain tunnistetut positiiviset tapaukset ohjataan väärinkäytösjärjestelmälle. Menetelmän etuna on, että virheellisten positiivisten määrä tippuu suuresti, ja tarkan analyysin vaativat tapahtumat pienenevät.

7 Tutkimuksessa käytetyt menetelmät

Edellisissä luvuissa on esitetty osa niistä hyökkäyksistä, joita vastaan verkot ja Web-palvelut joutuvat nykyisin suojautumaan. Niiden suuresta määrästä johtuen on ilmiselvää, että täysin turvallista ympäristöä on mahdoton rakentaa. Tätä ei edes pidetä tietoturvasuunnittelun lähtökohtana, vaan tärkeämpää on löytää tasapaino palvelun saatavuuden ja käytettyjen tietoturvaratkaisuiden välillä. Liian raskaat menetelmät aiheuttavat ylimääräistä viivettä palvelun tai verkon saatavuuteen, ja liian kevyet ratkaisut jättävät ne avoimiksi tietoturvahyökkäyksille.

Ongelmia lokin analysoimisessa aiheuttaa aineiston suuri määrä, ja siinä esiintyvien parametrien tyypit. Parametreista osa on luokka-asteikollisia ja osa puolestaan numeroasteikollisia, joten tietynlaisten toimintamallien etsiminen näistä suoraan on haastavaa. Parametrien suuri määrä aiheuttaa myös laskennallisia ongelmia, joten niiden määrää tulee pystyä jotenkin vähentämään säilyttäen kuitenkin mahdollisimman tarkkaan alkuperäisten muuttujien piirteet. Seuraavaksi esitellään yleisellä tasolla analysoinnissa käytetyt tekniikat, ja tätä seuraa näiden tarkempi matemaattinen kuvaus.

7.1 Menetelmien yleinen kuvaus

Anomalioiden tunnistamiseen käyttämämme järjestelmä pohjautuu diffuusiokuvausten ja diffuusioetäisyyksien käyttöön. Ne tarjoavat tehokkaan tavan löytää merkittäviä geometrisia rakenteita datasta, ja niiden käyttämistä moniulotteisen datan esittämisessä on esitelty [76] [77]. Menetelmien tehokkuus perustuu siihen, että diffuusiokuvausten avulla pystytään vähentämään analysoitavan datan dimensioita säilyttäen kuitenkin sen rakenne. Periaatteessa dimensioiden vähentäminen tarkoittaa sitä, että datajoukko esitetään toisella datajoukolla, jonka dimensio on pienempi. Tällöin sen klusteroiminen sekä analysoiminen ja esittäminen graafisesti on helpompaa.

Datan luonteesta johtuen pelkkä dimensioiden vähentäminen ei tuo esille poikkeavuuksia, vaan tätä varten tarvitaan parametreja, jotka kuvaavat HTTP-pyynnön sisältöä tarkemmin. Suurimmasta osasta kyselyn sisältämistä parametreista kuten

IP-osoitteesta, ajasta tai käytetystä selaimesta tämä ei käy ilmi. Toki näistä voidaan tunnistaa esimerkiksi hyökkäykset, joissa yritetään kuluttaa palvelimen resurssit loppuun hakemalla samaa resurssia yhä uudestaan tai pommittamalla uusia yhteysyrityksiä. Web-palveluihin kohdistuvat hyökkäykset ovat kuitenkin usein paljon hienovaraisempia. Parhaiten näitä voidaan yrittää tunnistaa tutkimalla tarkemmin GET-parametrin jälkeistä osaa, josta käy ilmi parametrit, joita hyökkääjä välittää palvelimelle.

Dimensioiden vähentäminen tapahtuu laskemalla diffuusioetäisyydet eli keskimääräiset arvot kaikista kahden pisteen välisistä poluista ts. todennäköisyydet kulkea satunnaiskululla pisteestä toiseen kiinteällä askelmäärällä. Ennen tätä analysoitava data tulee muuttaa kategoriseksi, sillä muuten erilaisten parametrityyppien välisiä etäisyyksiä ei pystyä laskemaan. Osa parametreista on jo valmiiksi kategorisessa muodossa, mutta numeerinen data tulee erikseen kategorisoida. Numeerisen datan automaattinen kategorisointi tapahtuu klusteroimalla yhtä ominaisuutta, ja laskemalla klusteroinnin hyvyysarvo. Tätä jatketaan niin pitkään, kunnes optimaalinen klusterointi on saavutettu. Prosessi toistetaan vaihtaen kategorioiden lukumäärää jokaisessa iteraatiossa, ja se luku, joka tuottaa parhaimman arvon, valitaan optimaaliseksi kategorioiden määräksi .

Tietoturvahyökkäyksissä hyökkääjä pyrkii aina ohittamaan jollakin tavalla asetetut suojaukset. Usein tämä tarkoittaa sitä, että palvelimelle välitetyt pyynnöt muodostuvat pitkistä merkkijonoista, ja niissä käytetyt merkit poikkeavat tyypillisesti käytetyistä merkeistä. Useiden peräkkäisten avain-arvo -parien määrä myös saattaa kasvaa reilusti tavallista suuremmaksi. Näiden tunnistamista varten käytämme analyysissa n-gram -analyysiksi kutsuttua menetelmää. Menetelmällä lasketaan datassa esiintyvien peräkkäisten merkkien tai sanasten esiintyvyystiheyksiä. Analyysi voidaan tehdä esimerkiksi koko kyselylle, avain-arvo -pareille tai avainten nimille.

Suurilla tietomassoilla n-gram -analyysi tuottaa isoja matriiseja, joiden käsitteleminen on hidasta. Tehokkuuden takia matriisien ulottuvuuksia tulee pystyä jollakin tavalla vähentämään. Tämä onnistuu satunnaisprojektion avulla, joka on ulottuvuuksien vähentämiseen tarkoitettu menetelmä. Satunnaisprojektiossa moniulotteinen data heijastetaan pienempiulotteiseen aliavaruuteen käyttäen satunnaisesti luotua matriisia. Näin syntynyt uusi matriisi on laskennallisesti tehokas, ja se säilyttää tässä tapauksessa riittävän määrän informaatiota.

7.2 Diffuusiokuvaus

Moniulotteisen datan analysoiminen on aina haasteellinen tehtävä johtuen parametrien suuresta määrästä. Tämän takia käytämme tässä työssä hyödyksi diffuusiokuvauksia, jotka ovat tehokas tapa vähentää analysoitavan datan ulottuvuuksia säilyttäen kuitenkin sen rakenne. Tämän jälkeen esimerkiksi klusterointi ja visualisointi pienempiulotteisessa avaruudessa on helpompaa.

Ensimmäiseksi diffuusiokuvauksiin perustuvalle järjestelmälle opetetaan datan normaali käyttäytyminen. Tämä tapahtuu käyttäen opetusmateriaalia, joka on osa analysoitavaa dataa. Olkoot tämä opetusmateriaali $\Gamma = \{x_1, x_2, \ldots, x_N\}, x_i \in \mathbb{R}^n$, jossa N on kyselyiden määrä ja n alkuperäisen datan ulottuvuuksien määrä. Meidän tapauksessamme data muodostaa $N \times n$ matriisin, jossa yksittäinen rivi kuvaa yhtä HTTP-kyselyä ja sarakkeet vastaavat HTTP-kyselyistä määritettyjä parametreja.

Aluksi luodaan samankaltaisuusmatriisi W, joka kuvaa pisteiden välisiä etäisyyksiä. Matriisi muodostetaan käyttäen painotettua Hammingin etäisyyttä. Tämä tapahtuu valitsemalla tutkittavasta matriisista Γ rivit x_i ja x_j sekä merkitsemällä yhden sarakkeen kategorioiden määrää π_k:lla.

$$W_{ij} = \exp\left(-\frac{\delta_{WH}(x_i, x_j)}{\epsilon}\right)$$

Datapisteiden välinen painotettu Hammingin etäisyys lasketaan summaamalla sarakkeiden väliset Hammingin etäisyydet ja jakamalla se kussakin sarakkeessa esiintyvien kategorioiden määrällä.

$$\delta_{WH}(x_i, x_j) = \sum_{k=1}^{n} \frac{\delta(x_{ik}, x_{jk})}{\pi_k}$$

Kahden alkion välinen Hammingin etäisyys määritellään seuraavasti:

$$\delta(x_{ik}, x_{jk}) = \begin{cases} 0 & \text{, kun } x_{ik} = x_{jk} \\ 1 & \text{, muutoin} \end{cases}$$

Tässä käytetään gaussisen ytimen etäisyysmittana δ_{WH}:aa. Pisteiden samankaltaisuuden toisiinsa nähden määrittelee ϵ.

Parametrin ϵ tulee olla riittävän suuri kattaakseen riittävästi ympäröiviä pisteitä, mutta ei kuitenkaan niin suuri, että se hävittäisi pisteiden väliset etäisyydet. Tässä tutkimuksessa käytetään laskennallisesti tehokasta Lafonin menetelmää, joka

on kuvattu B. Bahin tutkielmassa [78]. Menetelmässä lasketaan keskiarvo funktion $\delta_{WH}(x_i, x_j)$ minimiarvoista, kun $x_i \neq x_j$.

$$\epsilon = \frac{1}{k} \sum_{i=1}^{k} \min_{j:x_j \neq x_i} \delta_{WH}(x_i, x_j)$$

Näin luodun matriisin rivit normalisoidaan diagonaalimatriisin D avulla, joka luodaan yhtälössä 7.1.

$$D_{ii} = \sum_{j=1}^{N} W_{ij} \tag{7.1}$$

Nyt jokaisen rivin summaksi tulee 1. P kuvaa todennäköisyyttä sille, että yksittäinen alkio muuttuu tilasta toiseen.

$$P = D^{-1}W \tag{7.2}$$

Seuraavaksi määritellään muuttumistodennäköisyysmatriisin ominaisarvot. P:n ominaisarvot ovat samat kuin konjugaattimatriisin, joka on esitetty yhtälössä 7.3. Tämän matriisin ominaisarvot voidaan johtaa matriisista $\tilde{P}$, kuten myöhemmin tulee ilmi.

$$\tilde{P} = D^{\frac{1}{2}} P D^{-\frac{1}{2}} \tag{7.3}$$

Jos vaihdamme P:n yhtälöstä 7.3 yhtälössä 7.2 olevan kanssa, saamme yhtälön 7.4 symmetrisen todennäköisyysmatriisin $\tilde{P}$. Näin luotu matriisi säilyttää ominaisarvonsa, ja sitä kutsutaan normalisoidun graafin Laplace-operaatioksi.

$$\tilde{P} = D^{\frac{1}{2}} P D^{-\frac{1}{2}} = D^{\frac{1}{2}} D^{-1} W D^{-\frac{1}{2}} = D^{-\frac{1}{2}} W D^{-\frac{1}{2}} \tag{7.4}$$

Tämän jälkeen symmetrinen matriisi hajotetaan käyttäen singulaarihajotelmaa (engl. *Singular Value Decomposition*, SVD). Koska $\tilde{P}$ on normaalimatriisi, spektraaliteorian (engl. *spectral theorem*) mukaan tällaisen matriisin hajotelma on yhtälön 7.5 mukainen.

$$\tilde{P} = U \Lambda U^* \tag{7.5}$$

Matriisissa Λ diagonaalilla olevat singulaariarvot $diag([\lambda_1, \lambda_2, \ldots, \lambda_N])$ vastaavat matriisissa $\tilde{P}$ olevia ominaisarvoja, koska se on symmetrinen. Edelleen koska $\tilde{P}$ on konjugaatti P:n kanssa, sisältävät nämä kaksi samat ominaisarvot.

Matriisin $U = [u_0, u_1, \ldots, u_k]$ sarakkeet sisältävät matriisin $\tilde{P}$ k ominaisvektoria u_k. Käyttämällä yhtälöä 7.6 voidaan laskea matriisin P oikeat ominaisvektorit v_k, jolloin saamme ne matriisin V sarakkeina $V = [v_0, v_1, \ldots, v_k]$.

$$V = D^{-\frac{1}{2}}U \tag{7.6}$$

Nyt datan koordinaatit pienennetyssä avaruudessa ovat yhtälön 7.7 matriisissa Ψ. Matriisin rivit vastaavat datapisteitä ja sarakkeet näiden uusia pienennetyn avaruuden koordinaatteja.

$$\Psi = V\Lambda \tag{7.7}$$

Käyttämällä sopivaa arvoa ϵ lähestyvät ominaisarvot nopeasti nollaa, jolloin riittävän tarkkaan diffuusiokuvaukseen tarvitaan ainoastaan d komponenttia. Ensimmäinen ominaisvektori v_0 on vakio, joten se voidaan jättää pois. Käyttäen ainoastaan ensimmäiset d komponenttia diffuusiokuvauksen luomiseen on esitetty yhtälössä 7.8.

$$\Psi_d : x_i \rightarrow [\lambda_1 V_{i1}, \lambda_2 V_{i2}, \ldots, \lambda_d V_{id}] \tag{7.8}$$

Tämä diffuusiokuvaus upottaa tunnetut pisteet x_i d-ulotteiseen avaruuteen. Näin n-ulotteisesta datasta on muodostettu d-ulotteinen diffuusioavaruus.

Tarvittaessa diffuusiokuvaus voidaan skaalata jakamalla sen koordinaatit luvulla λ_1.

7.3 N-grammianalyysi

Tässä tutkielmassa on tarpeen analysoida tarkemmin HTTP:n GET-parametrin jälkeistä kyselyosuutta. Tarkoitusta varten tarvitaan menetelmä, joka toimii riittävän nopeasti ja tehokkaasti, mutta tuottaa kuitenkin hyödyllistä tietoa parametrien ominaisuuksista. N-grammianalyysi on hyvin tunnettu ja käytetty menetelmä, jolla tutkitaan peräkkäisten merkkien tai sanasten esiintymistiheyttä. Sitä käytetään laajalti muun muassa tilastollisen kielen analyysissa, jossa esimerkiksi puheentunnistuksessa sillä tutkitaan foneemeja eli kielen äänneyksikköjä.

Tässä tutkimuksessa yksiköt ovat merkkejä, joiden esiintymistiheyttä ja jakaantumista analysoidaan. Merkkien n-grammit muodostetaan asettamalla n:n merkin

pituinen ikkuna merkkijonon alkuun, jonka jälkeen ikkuna liu'utetaan merkki kerrallaan merkkijonon yli, keräten näin muodostuneet osamerkkijonot. Esimerkiksi merkkijonon `automaatti` 2-grammit ovat `au`, `ut`, `to`, `om`, `ma`, `aa`, `at`, `tt` ja `ti`. Näin muodostetusta n-grammien listasta voidaan muodostaa n-grammianalyysin avulla vektori, jonka alkioihin asetetaan kunkin n-grammin esiintymistiheys. Näin muodostettua vektoria voidaan käyttää hyödyksi merkkijonon samankaltaisuuden arvioinnissa.

N-grammianalyysin tuottamien vektoreiden pituus on m^n, jossa m on datassa esiintyvien sanasten määrä ja n on n-grammisarjan pituus. Tutkimuksessa analysoidaan vain kahden peräkkäisen merkin esiintyvyyksiä, jolloin $n = 2$, ja koska tutkittavat sanaset ovat 8-bittisiä merkkejä, niin $m = 2^8 = 256$. Informaation määrää pystytään vähentämään jonkin verran jättämällä huomiotta ne vektorin alkiot, jotka sisältävät nollan jokaisessa tutkittavassa vektorissa. Tästäkin huolimatta alkioita on niin runsaasti, että niistä muodostetun avaruuden ulottuvuuksien määrä tulee pystyä jollakin tavoin vähentämään.

7.4 Satunnaisprojektio

Satunnaisprojektiossa alkuperäinen N-ulotteinen data heijastetaan k-ulotteiseen $(k \ll N)$ aliavaruuteen käyttäen satunnaista $k \times N$ matriisia R [79]. Olkoot meillä esimerkiksi matriisi $X_{m \times N}$, jossa m on havaintojen määrä, ja N on datan alkuperäinen dimensio. Olkoot k sitten uusi haluttu dimensioiden määrä. Uuden matriisin laskemiseksi luodaan satunnainen matriisi $R_{n \times k}$, jossa jokaisen sarakkeen arvot ovat satunnaisesti jakautuneet. Kertomalla nämä keskenään saadaan matriisi $X^{RP}_{m \times k}$, joka on esitys alkuperäisestä datasta X heijastettuna k-ulotteiseen aliavaruuteen:

$$X^{RP}_{m \times k} = X_{m \times N} \cdot R_{n \times k}. \tag{7.9}$$

Satunnaisprojektion idea on lähtöisin Johnson-Lindenstraussin lemmasta: jos vektoriavaruudessa olevat pisteet heijastetaan satunnaisesti valittuun aliavaruuteen jossa on sopiva määrä ulottuvuuksia, säilyvät pisteiden väliset etäisyydet riittävällä tarkkuudella. $X_{m \times N} \cdot R_{n \times k}$. laskemisen aikavaativuus on $O(dkN)$, ja jos matriisi X sisältää pääasiallisesti nollia ja rivissä on keskimääräisesti c kappaletta arvoja $(c \ll N)$, on aikavaativuus $O(ckN)$.

Satunnaisesti luotu matriisi R voidaan valita monella eri tapaa. Useimmiten matriisin R elementit r_{ij} noudattavat Gaussin jakaumaa, mutta se voidaan muodostaa

myös muulla tavoin kuten esimerkiksi

$$r_{ij} = \sqrt{3} \cdot \begin{cases} +1 & \text{todennäköisyydellä } \frac{1}{6} \\ 0 & \text{todennäköisyydellä } \frac{2}{3} \\ -1 & \text{todennäköisyydellä } \frac{1}{6} \end{cases} \tag{7.10}$$

Tällaisen jakauman käyttäminen vähentää entisestään laskenta-aikaa, sillä laskenta voidaan suorittaa käyttäen kokonaislukuja. Yllä olevan jakauman tapauksessa laskenta on vieläkin nopeampaa, sillä operaatioista tarvitaan vain kolmasosa, sillä luotu matriisi sisältää suurimmaksi osaksi nollia [79].

8 Tiedon keruu ja käsittely

Tiedon käsittely jaetaan esikäsittelyyn, menetelmään ja jälkikäsittelyyn. Esikäsittelyn aikana palvelinlokit muutetaan sellaiseen muotoon, jota menetelmä pystyy käsittelemään. Menetelmävaiheessa aineisto ajetaan diffuusiokuvausalgoritmin läpi. Jälkikäsittelyssä sitten luetaan diffuusiokuvauksen tuottamaa dataa, ja selvitetään mielenkiintoisten havaintojen taustalla olevat HTTP-kyselyt.

Tässä luvussa tutustutaan aluksi tutkittavien lokitiedostojen alkuperään ja tiedostomuotoon sekä aineiston arkistointitapaan. Tämän jälkeen käydään läpi, kuinka tietoa on esikäsitelty analyysiä silmällä pitäen. Käytetyt menetelmät soveltuvat pienin muutoksin myös muiden Web-palveluntarjoajien palvelinlokien esikäsittelyyn, mikäli arkistointikäytänteet eivät suuresti poikkea tässä esitellystä.

8.1 Aineiston rakenne

Analysoitavaksi saamamme loki on peräisin yritykseltä, joka toimii Web-palveluntarjoajana suuryrityksille. Lokitiedostot ovat peräisin kolmelta tuotannosta jo poistetuilta palvelimilta, ja ne ovat olleet vastuussa muutaman suuren palvelukokonaisuuden pyörittämisestä.

Analysoitava data on Apache-palvelimien tuottamaa lokia, joka sisältää Web-palveluille kohdistuvia HTTP-pyyntöjä. Lokia on yhteensä 24 gigatavua ja se sisältää yli 1,7 miljardia sivupyyntöä. Lokia on kerätty noin 10 kuukauden ajanjaksolta. Se on tallennettu *Combined Log Format* -muotoon, joka on yleinen Apache-palvelimen käyttämä lokitiedoston muoto [80]. Listauksessa 8.1 on esimerkki onnistuneen HTTP-kyselyn seurauksena muodostuneesta lokirivistä.

```
130.234.49.2 - - [10/May/2009:15:53:01 +0300]
"GET /scripts/access.pl?user=matti&passwd=admin HTTP/1.1"
200 2680 "http://www.jyu.fi/a.html"
"Mozilla/5.0 (SymbianOS/9.2;...)"
```

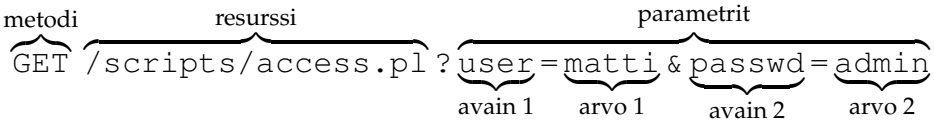
Listaus 8.1: Esimerkki onnistuneesta HTTP-kyselystä.

Web-palvelimien tuottama loki sisältää paljon tietoa muun muassa palveluiden käyttöasteista, ja niihin kohdistuvista kuormista. Lokeja analysoimalla voidaan myös tunnistaa mahdollisia hyökkäysyrityksiä, sekä hyökkäyksen jo tapahduttua tutkia siihen johtaneita vaiheita. Pienillä sivustoilla ihmisen on mahdollista läpikäydä lokitiedostot hyökkäyksien varalta, mutta puhuttaessa palveluista, joilla on miljoonia käyttäjiä kuukaudessa, ei käsin läpikäyminen ole enää käytännössä mahdollista. Tästä syystä lokien analysoiminen tulee automatisoida. Analysoinnille on myös kovat laatuvaatimukset, koska järjestelmän tulisi pystyä tunnistamaan miljoonien kyselyiden joukosta ne, jotka ovat syntyneet hyökkäyksen johdosta.

Yksi mahdollisuus on käyttää sääntöpohjaisia suodattimia, mutta aikaisemmissa luvuissa esitetyistä syistä johtuen ei tämä tarjoa aina riittävää tunnistuskykyä. Siksi tässä tutkimuksessa on päädytty käyttämään diffuusiokuvauksiin perustuvaa anomalioiden tunnistamismenetelmää. Tässä järjestelmälle opetetaan ensiksi normaali käyttäytyminen, jonka jälkeen analysoitavaa lokia verrataan opetettuun malliin. Näin poikkeava liikenne voidaan tunnistaa.

Tietoturvan kannalta kiinnostavin osa lokista on HTTP-kyselyn osoiteosa, jossa viitataan johonkin resurssiin ja välitetään tarvittaessa parametreja. HTTP-kysely sisältää niin staattiset sivunlatauspyynnöt kuin myös palvelimelle välitettävät parametriarvot. Tällaisia voivat olla esimerkiksi kirjautumisessa käytetyt tiedot ja tietokannalle välitetyt kyselyt.

Kysely koostuu metodista, resurssista ja parametriosasta. Metodin jälkeen välilyönnillä erotettuna seuraa osoiteosa. Se jakautuu resurssiin ja parametreihin, jossa erottimena toimii kysymysmerkki. Parametriosa koostuu parametrilistasta, jossa parametrit ovat erotettu toisistaan `&`-merkillä. Yksittäinen parametri on avain-arvopari, jossa avain on yhtäsuuruusmerkin vasemmalla puolella ja arvo oikealla puolella (kuva 8.1). Käytettävien parametrien lukumäärä vaihtelee palvelusta riippuen. Mikäli parametreja ei ole lainkaan, ei parametriosaa eikä sitä edeltävää kysymysmerkkiä esiinny kyselyssä. Näin on esimerkiksi silloin, kun HTTP-kysely kohdistuu staattiseen Web-sivuun.



Kuva 8.1: HTTP GET -kyselyn rakenne.

8.2 Arkistointikäytänteet

Web-hotelli, josta data on peräisin, on toteutettu siten, että yksittäiset palvelut on sijoitettu useammalle palvelimelle. Ratkaisun taustalla on kuormituksen tasaaminen. Erillinen järjestelmä huolehtii siitä, että Internetistä tulevat kyselyt ohjataan tasaisesti eri palvelimille. Tästä johtuen samaan palveluun kohdistuvat kyselyt jakautuvat useamman palvelimen lokitiedostoihin.

Taulukossa 8.1 on esimerkki palveluiden ja palvelinten nimeämisestä. Esimerkeissä käytetyt palvelinten ja palveluiden nimet ovat kuvitteellisia, mutta rakenne vastaa tutkittua ympäristöä.

palvelin	palvelu
dapper	buzz
edgy	rex
feisty	potato
	hamm

Taulukko 8.1: Palvelinten ja palveluiden nimeäminen.

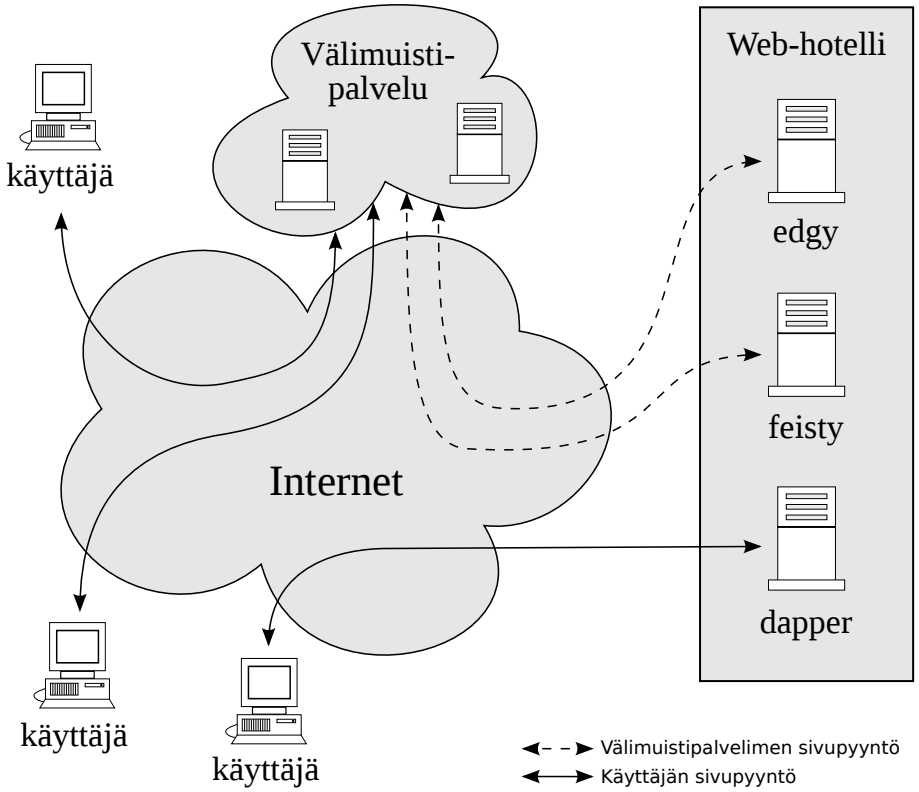
Lokitiedostot on sijoitettu hakemistorakenteeseen, jossa juuressa ovat palvelinten nimien mukaiset hakemistot. Näiden hakemistojen sisällä sijaitsevat lokitiedostot, jotka on nimetään yhdistämällä palvelun nimi päivämääräleimaan. Tiedostonnimessä käytetty päivämäärän muoto noudattaa ISO 8601 -standardia [81], ja tiedostot on pakattu `gzip`-pakkausohjelmalla. Taulukossa 8.2 on havainnollistettu tiedostonnimien muodostuminen.

palvelin	palvelu	päivämäärä	tiedostonnimi
edgy	buzz	14.6.2009	`edgy/buzz.http.2009-06-14.gz`
dapper	potato	2.7.2009	`dapper/potato.http.2009-07-02.gz`
feisty	rex	30.7.2009	`feisty/rex.http.2009-07-30.gz`

Taulukko 8.2: Tiedostojen nimeäminen.

Tässä mainittujen palvelinten lisäksi käytössä on ulkoinen välimuistipalvelu, jonka kautta välitetään harvoin muuttuvia resursseja kuten kuvia. Valtaosa dataliikenteestä välitetään palvelun kautta. Välimuistipalvelu toimii siten, että mikäli pyydettyä resurssia ei löydy sen omasta muistista, se pyytää sitä yhdeltä Web-hotellin palvelimista ja välittää vastauksen edelleen asiakkaalle. Välimuistipalvelun käyttö

ei kuitenkaan muilta osin vaikuta palvelun rakenteeseen. Tämä rakenne käy ilmi kuvasta 8.2.



Kuva 8.2: Web-palvelun rakenne.

8.3 Esikäsittely

Luvussa 8.1 esiteltiin lokitiedostoissa käytetty *Combined Log Format* -muoto. Tiedosto on tekstimuotoinen, joten se ei sellaisenaan sovellu analyysivaiheeseen, jossa käsitellään luokka-asteikollisia muuttujia. Osa palvelinlokin sisällöstä on myös analyysin kannalta tarpeetonta ja nämä osat tulee suodattaa pois. Lisäksi yhteen Web-palveluun liittyvät kyselyt ovat jakautuneet useaan eri tiedostoon eri palvelinten ja vuorokausien mukaisesti.

Tässä työssä käytetty esikäsittelijä on toteutettu osana tätä tutkimusta. Sovelluksen nimi on *PhasefulSplitter* ja se on ohjelmoitu käyttäen Haskell-ohjelmointikieltä [82]. Haskell on valittu, koska se tarjoaa tehokkaat työkalut ohjelmakoodin rinnakkaistamiseen ja datan sarjallistamiseen. *PhasefulSplitter* on vapaa ohjelma ja sitä saa levittää edelleen ja muuttaa Free Software Foundationin julkaiseman GNU General

Public Licensen (GPL-lisenssi) version 3 [83] tai (valinnan mukaan) myöhemmän version ehtojen mukaisesti. Sovellus on ladattavissa Internetistä osoitteesta `http://iki.fi/zouppen/repo/phasefulsplitter.git`.

Sopivaa yksivaiheista parseria käyttämällä olisi mahdollista käsitellä lähtödata suoraan analyysissä käytettävään muotoon. Käytännössä kuitenkin datan esikäsittely kannattaa hoitaa useammassa vaiheessa, jotta datassa olevat puuttuvat tai poikkeavat arvot voidaan huomioida ja esikäsittelijää voidaan korjata suorittamatta koko ajoa uudelleen alusta lähtien. Monivaiheinen esikäsittely helpottaa myös datan käsittelyä jälkikäteen erilaisin menetelmin. Lisäksi sarjallistetut tietorakenteet voidaan tarvittaessa anonymisoida, jolloin dataa voidaan luovuttaa myös ulkopuoliseen käyttöön.

Tiedon käsittelyn helpottamiseksi tässä työssä käytetään esikäsittelyn välivaiheet ja lopputulokset tallennetaan väliaikaistiedostoihin. Relaatiotietokantaa ei käytetä, koska tietokantakyselyiden rinnakkaistaminen osoittautui hyvin vaikeaksi verrattuna suoraan tiedostoja käsittelevään toteutukseen.

Esikäsittely jakaantuu seuraavaan neljään vaiheeseen:

1. Tiedostolistan muodostaminen ja tiedostojen ryhmittely palveluittain.
2. Tiedostojen sisällön lukeminen ja muuntaminen tietorakenteeksi.
3. N-grammien muodostaminen HTTP-kyselyn parametreista.
4. Aineiston muuntaminen matriisimuotoon analyysiä varten.

Seuraavaksi käydään läpi esimerkkien tuella se, kuinka nämä vaiheet suoritetaan.

8.3.1 Tiedostolistan muodostaminen

Tekstipohjainen tiedostolista luetaan *PhasefulSplitter*-ohjelman ymmärtämään muotoon, jolloin tiedostonnimet ryhmitellään palveluiden nimien perusteella. Luokittelun yhteydessä myös palvelinten nimet korvataan numeerisilla viitteillä. Tätä varten asetetaan tiedostoon `server.map` palvelinten nimet ja niitä vastaavat numeeriset viitteet. Viitteitä voidaan hyödyntää myöhemmin, jos halutaan muuntaa analyysivaiheessa kiinnostava havainto takaisin Apache-lokin riviksi. Tässä luvun esimerkissä käytettävän tiedoston sisältö on esitelty listauksessa 8.2.

```
[
      ("dapper",1)
    , ("edgy",2)
    , ("feisty",3)
]
```

Listaus 8.2: Tiedoston server.map sisältö.

Tiedostolista voidaan muodostaa Linux-järjestelmässä listauksen 8.3 mukaisesti. Ajon yhteydessä muodostuu hakemiston `lists` alle palveluiden mukaan nimetyt tiedostot. Tiedoston sisältönä on palveluun kuuluvat tiedostonnimet sekä tiedon siitä, minkä palvelimen lokitiedostosta on kyse. Tiedostot ovat tekstimuotoon sarjallistettuja.

```
mkdir lists
find polku -iname '*.gz' | classifier lists/
```

Listaus 8.3: Tiedostolistan muodostaminen.

8.3.2 Muuntaminen tietorakenteeksi

Toisessa vaiheessa tekstimuotoiset lokitiedostot luetaan ja käsitellään koneellisesti helpommin analysoitavaan muotoon. Tätä työtä varten kehitetyssä tiedonkäsittelijässä lokitiedoston rivin eri kentät palastellaan ja kyselyt sarjallistetaan tiedostoihin. Tietorakenne on esitelty listauksessa 8.4.

Koska jatkokäsittely voidaan hoitaa säikeistettynä useammalle prosessorille tai ytimelle, tässä vaiheessa sovellukselle tulee ilmoittaa säikeiden määrä. Tiedon perusteella sovellus jakaa yhteen palveluun liittyvän datan haluttuun määrään erillisiä tiedostoja. Jako mahdollistaa tehokkaan jälkikäsittelyn, koska tiedostojen sisältöä on mahdollista käsitellä myöhemmin samanaikaisesti.

Linuxin `xargs`-komennolla voidaan myös tämän vaihe suorittaa rinnakkaistettuna, jolloin eri palveluihin kuuluva data voidaan muuntaa samanaikaisesti. Mikäli eri palveluihin kuuluvien lokien datamäärä vaihtelee, ei rinnakkaistamisesta kuitenkaan saavuteta täyttä hyötyä. Listauksessa 8.5 muunnetaan palvelinlokit sovelluksen käyttämään muotoon. Käsiteltävät tiedostot luetaan `lists`-hakemistosta löytyvien tiedostolistauksista. Tietorakenteet kirjoitetaan hakemistoon `data`. Lis-

tauksessa mainittu N korvataan tietokoneen prosessorien tai ytimien määrällä.

```
data Entry = Entry {
      info        :: LineInfo
    , ip          :: ByteString
    , date        :: UTCTime
    , method      :: ByteString
    , url         :: URL
    , protocol    :: ByteString
    , response    :: Integer
    , bytes       :: Integer
    , referer     :: ByteString
    , browser     :: ByteString
} deriving (Show,Eq)
```

Listaus 8.4: Yhden lokirivin säilövä tietorakenne.

```
ls lists/* | xargs -n 1 -I{} -P N apache2data N {} data/
```

Listaus 8.5: Muuntaminen tietorakenteeksi.

Suoritusaika riippuu luonnollisesti koneen suorituskyvystä ja lokitiedostojen määrästä. Tämän tutkimuksen yhteydessä suoritetussa 24 gigatavun ajossa 2,5 gigahertsin Intel Xeon -prosessorilla varustetussa tietokoneessa tämän vaiheen suoritus kesti useita prosessorivuorokausia.

8.3.3 Parametrien N-grammianalyysi

Palvelinlokissa olevista kyselyistä muodostetaan tietorakenne, jossa yksi osa on kyselyn URL. Tässä vaiheessa tutkitaan URL-osoitetta tarkemmin. Mikäli URL:n osana on parametreja, muodostetaan jokaisesta parametrin arvosta 2-grammilistaus. Näin saadut 2-grammikartat muodostetaan ja lopuksi ne tallennetaan matriisina tekstimuotoon jatkokäsittelyä varten. Tiedostossa `ParameterAnalyzer.hs` on määritelty vakiona, että lasketaan nimenomaisesti 2-grammit. Tätä vakiota voi kuitenkin tarvittaessa muuttaa.

Ensimmäisessä ajossa selvitetään, mitkä mahdollisista 2-grammeista ylipäätään esiintyvät aineistossa. Mahdollisia N-grammeja on yhteensä 2^{8n} kappaletta, koska käsiteltävät merkkijonot koostuvat Word8-tyypeistä (8-bittinen tavu). Säästääksemme muistia analyysivaiheessa, selvitetään aluksi, millaisia 2-grammeja aineistossa esiintyy ja jätetään taulukoimatta sellaiset 2-grammit, jotka eivät esiinny lainkaan.

Jokaiselle resurssille muodostetaan oma N-grammilistansa. Datan määrä luonnollisesti riippuu käytettävästä aineistosta. Käyttämällämme datalla taulukon kooksi muodostui muutamia kymmeniä kilotavuja. Lopputulos on siis vain murto-osa alkuperäisen datamassan suuruudesta.

N-grammianalyysin ensimmäinen vaihe suoritetaan halutun palvelun hakemistossa listauksen 8.6 mukaisesti. Hakemistoon muodostuu ajon seurauksena tiedostot `ngrams.out` ja `grams_raw.txt`. Näistä ensimmäinen on binaarimuodossa ja se sisältää toisessa vaiheessa tarvittavan datan. Toisessa tiedostossa on tekstimuodossa (Haskellin Show-muodossa) eri resurssien ja 2-grammien esiintymistiheydet. Tekstimuotoista tiedostoa voidaan käyttää apuna arvioitaessa, mihin resursseihin ja parametreihin kannattaa kiinnittää jatkossa huomiota.

```
parameter_analyzer *.pf.gz +RTS -N
```

Listaus 8.6: N-grammianalyysin ensimmäinen vaihe.

8.3.4 Matriisien muodostaminen tietorakenteista

Tämän vaiheen yhteydessä aineistossa esiintyvät eri tyyppiset arvot numeroidaan. Nämä arvot sijoitetaan matriisin sarakkeisiin taulukon 8.3 mukaisesti. Näistä kolme ensimmäistä saraketta yksilöivät HTTP-kyselyn. Tietojen avulla on mahdollista hakea matriisin riviä vastaava palvelinlokin rivi. Ominaisuutta hyödynnetään analyysivaiheessa kun poikkeaviksi havaittuja pisteitä tutkitaan.

Sarakkeissa 4 ja 5 on HTTP-kyselyn saapumisajankohdasta laskettu tunti ja viikonpäivä. Tämä ajankohta määräytyy palvelimen kellon mukaan ja aikavyöhykkeenä on UTC. Lokista ei kuitenkaan nähdä miltä aikavyöhykkeeltä kysely on peräisin.

Kyselyssä esiintyvä HTTP-metodi ja HTTP-protokollan versio ovat esitetty sarakkeissa 6 ja 7. Metodit on numeroitu taulukon 8.4 mukaisesti, ja protokollan versio puolestaan taulukon 8.5 mukaisesti. Numeerinen HTTP-vastauskoodi esiintyy sellaisenaan sarakkeessa 8.

sarake	sisältö
1	tiedoston järjestysnumero
2	palvelimen tunnistenumero
3	rivinumero lokitiedostossa
4	tunti (0-23)
5	viikonpäivä (maanantai = 1, ... , sunnuntai = 7)
6	HTTP-metodi
7	HTTP-protokollan versio
8	HTTP-vastauskoodi
9	siirretyn datamäärän suuruusluokka
10	resurssin numero
11 ...	lista n-grammien esiintymistiheyksistä

Taulukko 8.3: Analysoitavan matriisin sarakkeiden sisältö.

metodi	ryhmä
HEAD	1
GET	2
POST	3
PUT	4
DELETE	5
TRACE	6
OPTIONS	7
CONNECT	8
PATCH	9

Taulukko 8.4: HTTP method -kentän numerointi.

versio	ryhmä
HTTP/1.0	1
HTTP/1.1	2

Taulukko 8.5: HTTP:n versiokentän numerointi.

Kyselyn seurauksena siirretyn datamäärän suuruusluokka ilmaistaan sarakkeessa 9. Suuruusluokka lasketaan funktion 8.1 avulla, jossa x on siirretyn datan määrä tavuina.

$$\mathrm{M}(x) = \begin{cases} 0 & , \text{kun } 0 \leq x < 1 \\ \lfloor \log_2 x \rfloor + 1 & , \text{kun } x \geq 1 \end{cases} \tag{8.1}$$

Sarakkeessa 10 ilmaistaan resurssin numero. Koska jokaisesta resurssista muodostetaan erillinen matriisi, tämän sarakkeen arvo on vakio jokaisessa matriisissa.

Käyttämässämme datassa olevat kyselyiden parametrit sisältävät hyvin samantyyppisiä arvoja, joten niistä muodostuu kohtuullisen pieniulotteisia 2-grammitaulukoita. Tässä tutkimuksessa käytetyllä aineistolla suurin esiintynyt 2-grammien lukumäärä oli 189. Käytännössä havaittiin, että käyttämällämme laitteistolla alle 200 saraketta sisältävän matriisin laskenta oli riittävän nopeaa. Tämän vuoksi dimensioita ei tarvitse vähentää, joten matriiseihin ei sovelleta aikaisemmin esitettyä satunnaisprojektiota. Sen sijaan muodostetaan matriisi, jossa luetellaan kussakin HTTP-kyselyssä esiintyvien N-grammien esiintymistiheydet. Näin muodostetun matriisin tiedot liitetään aiemmin kerättyjen parametrien perään.

Analyysissä käytettävät matriisit muodostetaan `parameter2vector`-sovelluksella. Listauksessa 8.7 esiintyvä parametri `ngram` korvataan edellisessä vaiheessa muodostuneella N-grammitiedoston nimellä, `lahde` korvataan tietorakennetiedoston nimellä ja parametrissa `kohde` määritellään muodostettavien matriisitiedostojen nimen alkuosa.

```
parameter2vector ngram lahde kohde
```

Listaus 8.7: Matriisien muodostaminen.

Ajon seurauksena muodostuu jokaisesta resurssista erillinen matriisi. Kukin matriisi tallennetaan resurssin mukaan nimettyyn tiedostoon CSV-muodossa. Siinä matriisin rivit erotetaan toisistaan rivinvaihdolla, ja sarakkeet pilkulla. Listauksessa 8.8 on poiminta CSV-muotoon tallennetusta matriisista.

```
6,3,13619,10,2,2,2,200,12,882,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
7,3,833,21,5,2,2,200,12,882,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
8,2,2324,23,3,2,2,200,12,882,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
8,2,18596,15,4,2,2,200,12,882,1,1,1,1,1,2,1,1,1,1,1,0,0,0,0,0,0
8,2,14500,12,4,2,1,200,12,882,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
```

Listaus 8.8: Esikäsiteltyä dataa CSV-muodossa esitettynä.

8.4 Diffuusiokuvausten laskenta

Tässä tutkimuksessa käytettiin diffuusiokuvaussovellusta, joka on toteutettu Jyväskylän yliopistossa Tuomo Sipolan toimesta. Sovellus toimii Matlab-ympäristössä ja se lukee taulukon 8.3 mukaisesti muotoiltuja tiedostoja ja palauttaa analyysin tulokset CSV-muodossa, jonka rakenne on esitelty taulukossa 8.6.

sarake	sisältö
1	tiedoston järjestysnumero
2	palvelimen tunnistenumero
3	rivinumero lokitiedostossa
4	2. ominaisvektorin arvo
5	3. ominaisvektorin arvo
6	4. ominaisvektorin arvo
7	poikkeavuusluku

Taulukko 8.6: Tulosmatriisin sarakkeiden sisältö.

```
6,3,13619,-66.236,-0.23311,2.3295,0.0001666
7,3,833,-66.236,-0.23311,2.5856,0.0012385
8,2,2324,-66.236,-0.23311,2.3297,0.0001666
8,2,18596,-284.17,-6.6455e-06,1.3067e-12,653.84
8,2,14500,-66.236,-0.23311,-83.326,1.5157
```

Listaus 8.9: Analysoitua dataa CSV-muodossa esitettynä.

Diffuusiokuvausalgoritmin aikavaativuus on eksponentiaalinen suhteessa opetusmateriaalin kokoon. Tämän vuoksi diffuusiokuvausten laskentaan voidaan käytännössä käyttää korkeintaan 5 000 yksittäistä HTTP-kyselyä vastaavaa pistettä.

Ajan säästämiseksi tässä tutkimuksessa käytettiin kuitenkin enintään 2 000 pisteen kokoisia aineistoja. Pisteiden määrän rajoittamiseen käytettiin satunnaisotannan suorittavaa apuohjelmaa, joka sijaitsee PhasefulSplitter-sovelluksen tiedostossa `src/LinePicker.hs`.

8.5 HTTP-kyselyiden yhdistäminen tuloksiin

Esikäsittelyvaiheessa jokaiseen HTTP-kyselyyn liitetään LineInfo-tietorakenne, jota kuljetetaan analyysivaiheen läpi. Analyysin tuloksena saadaan taulukon 8.6 mukaisia CSV-muodossa olevia matriiseja. Tietorakenne tallennetaan matriisin kolmeen ensimmäiseen sarakkeeseen, eli tekstimuotoisessa esityksessä se sijaitsee rivin kolmessa ensimmäisessä pilkulla erotetussa kentässä. Esimerkiksi listauksessa 8.8 ensimmäisen pisteen LineInfo-arvot ovat 6, 3 ja 13619.

Jotta analyysin tulokset olisivat hyödynnettävissä, on tarpeellista yhdistää tulokset alkuperäisiin HTTP-kyselyihin. Tällöin voidaan tarkemmin tutkia, millaisia poikkeavat kyselyt ovat muodostaan. Yhdistämisessä käytettävä apuohjelma sijaitsee tiedostossa `src/LineInfoConvert.hs`. Sen käyttö on esitelty listauksessa 8.10. Parametriin `tulosmatriisi` sijoitetaan analyysin tuloksena saadun matriisin tiedostonnimi. Parametri `lista` korvataan palvelun tiedostolistalla, joka on muodostettu luvussa 8.3.1 esitetyllä tavalla. Yhdistetyt tulokset kirjoitetaan parametrissa `kohde` määriteltyyn tiedostoon, jonka rakenne on kuvattu taulukossa 8.7.

```
a <- readZipolaFile tulosmatriisi
saveLogLines lista kohde a
```

Listaus 8.10: HTTP-kyselyiden yhdistäminen tuloksiin.

sarake	sisältö
1	tiedoston järjestysnumero
2	palvelimen tunnistenumero
3	rivinumero lokitiedostossa
4	HTTP-kysely Combined Log Format -muodossa

Taulukko 8.7: Yhdistetty tulosmatriisi.

9 Analyysi

Tässä luvussa käydään vaiheittain läpi yhden valitun palvelun analysointi sekä esitetään perustelut, joiden pohjalta tehtyihin ratkaisuihin on päädytty. Luvussa myös esitellään analysoinnista saatuja tuloksia sekä pohditaan näihin johtaneita syitä. Lopuksi vielä esitetään jatkotutkimuksen kannalta tärkeitä kehitysideoita, joita syntyi tutkimuksen aikana.

9.1 Tutkimuksen toteutus

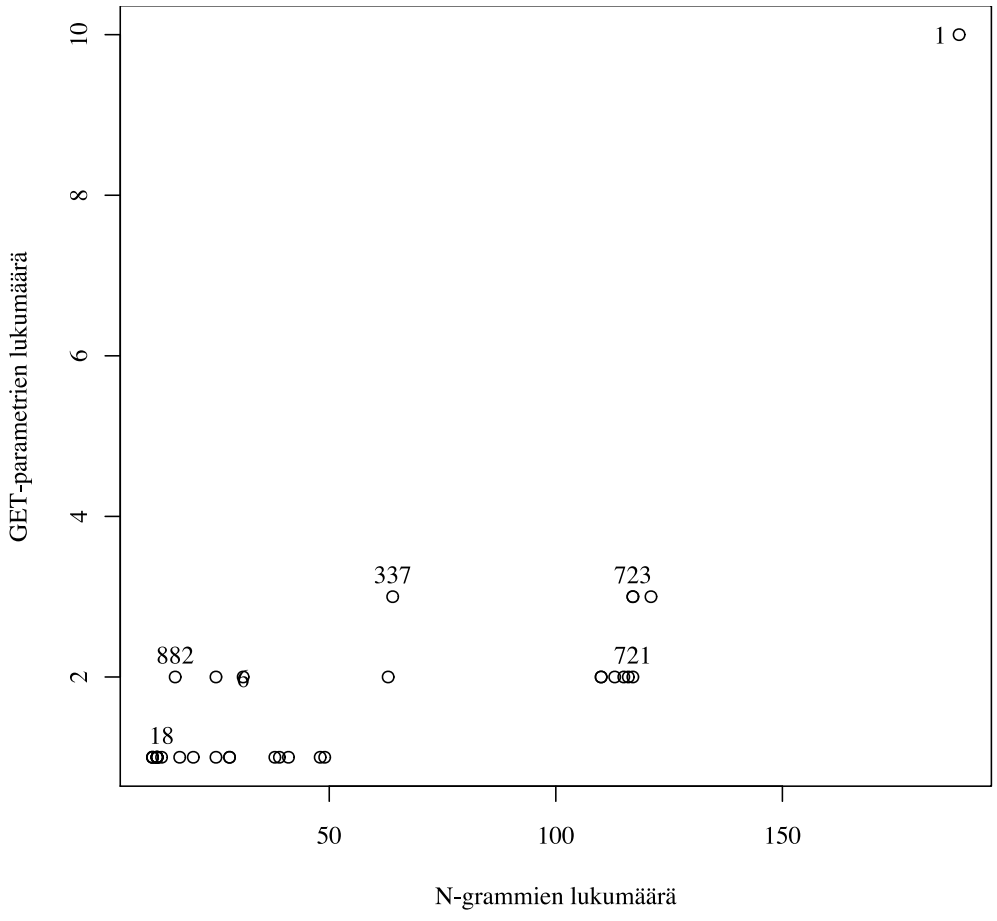
Saamamme materiaali koostui neljästä eri Web-palvelusta, joista analyysiin valitsimme yhden. Muista palveluista saadut lokitiedostot käsiteltiin myös valmiiksi, mutta koska lokitietoja oli määrällisesti niin paljon, päädyimme tarkastelemaan tarkemmin vain yhtä. Valitsemastamme palvelusta analysoimme myös vain viikon mittaisen jakson. Tähän ratkaisuun päädyttiin sen takia, että analysoitavaa dataa oli kerätty noin puolen vuoden ajalta, joten jo yhden palvelun osalta koko datan läpikäymiseen olisi mennyt kohtuuttomasti aikaa. Käytetyn menetelmän rajoitukset, jotka on kerrottu luvussa 8.4, asettivat myös käytännön rajoituksia analysoitavien tiedostojen määrälle. Tämän kokoinen otanta oli kuitenkin riittävän suuri esikäsittelijän toimivuuden testaamiseen, jonka lisäksi käytettyjen menetelmien sopivuutta käytössä olleen materiaalin analysoimiseen voitiin arvioida.

Esikäsittelyvaiheessa HTTP-kyselyt jaetaan palvelun resurssien mukaan omiin tiedostoihin, joissa ne ovat CSV-formaatissa kuvan 8.8 mukaisesti. Resursseihin kohdistuvien kyselyiden lukumäärä vaihtelee suuresti sen mukaan, kuinka käytetty mikäkin resurssi on. Web-sivun ollessa kyseessä lukumäärä vastaa sivun käyntimäärää. Mikäli resurssia käytetään esimerkiksi muun sivun osana, kuten tyylitiedostojen ja kuvien tapauksessa, tämä kasvattaa merkittävästi kyselyiden lukumäärää verrattuna tavanomaiseen Web-sivuun.

Analyysissä käyttämämme viikon pituinen jakso sisälsi yhteensä 913 eri resurssia ja näihin kohdistuvat kyselymäärät vaihtelivat muutamasta kappaleesta aina kymmeniin tuhansiin. Koska Web-palveluihin kohdistuvat tietoturvahyökkäyksissä hyödynnetään erityisesti HTTP-kyselyn parametriosan käsittelyn haavoittuvai-

suuksia, keskityimme vain sellaisiin palveluihin, joissa esiintyi vaihteleva kirjo parametreja. Tämä rajaus tehtiin suodattamalla pois sellaiset resurssit, joihin kohdistui alle sata kyselyä ja joiden GET-parametreista muodostettujen erilaisten n-grammien määrä oli alle kymmenen. Suodatuksen jälkeen analysoitavaksi jäi 36 resurssia. Suodatus suoritettiin esikäsittelyn jälkeen ja suodatusrajat on säädettävissä PhasefulSplitterin tiedostosta `src/InterestingParameters.hs`.

Käytetyimmissä resursseissa HTTP-kyselyitä oli kymmeniä tuhansia, joten jokaisesta resurssista ei pystytty suoraan tekemään diffuusiokuvausta johtuen menetelmän rajoituksista. Ongelma ratkaistiin valitsemalla satunnaisotannalla ilman takaisinpanoa 2 000 HTTP-kyselyä niistä resursseista, joissa oli kyselyitä yli tämän määrän. Näin ollen jokaisesta resurssista muodostettiin lopulta tiedosto, jossa oli enintään 2 000 HTTP-kyselyä.



Kuva 9.1: Palvelun resurssien ominaisuudet.

Analysoitavan palvelun eri resurssit on esitetty kuvassa 9.1. Kuvassa vaaka-akselina on käytetty N-grammien lukumäärää ja pystyakselina GET-parametrien lu-

kumäärää. Resursseista valitsimme kuusi tarkempaan tarkasteluun ja ne on merkitty kuvaajaan niiden resurssinumerolla. Resurssit pyrimme valitsemaan siten, että jokaisesta ryppäästä olisi yksi valittuna.

Palvelun jokaisesta resurssista muodostettiin aluksi diffuusiokuvaus käyttäen edellä kuvatulla tavalla suodatettua aineistoa. Analysoimalla saatua diffuusiokuvausta voitiin määrittää ne pisteet, jotka oli luokiteltu poikkeaviksi siinä joukossa, jossa kyseisiä pisteitä oli verrattu.

Seuraavaksi lasketuista diffuusiokuvauksista muodostettiin diffuusiokartta käyttäen toista ja kolmatta ominaisvektoria. Koska suuri määrä pisteitä sijoittui täsmälleen samoihin koordinaatteihin, kartan luettavuuden parantamiseksi sirontakuvion pisteitä "täristettiin". Täristäminen suoritettiin lisäämällä ominaisvektorien arvoihin normaalijakaumasta satunnaisesti poimittu arvo. Normaalijakauman keskihajontana käytettiin yhtä prosenttia akselin leveydestä. Kuvassa 9.2 nähdään edellä kuvatulla tavalla muodostettu diffuusiokartta resurssista 1. Kuvaajasta voidaan havaita, että muutamaa poikkeusta lukuun ottamatta pisteet sijoittuivat hyvin lähelle toisia.

Diffuusiokartta ei ole sellaisenaan kovinkaan havainnollinen, joten jokaiselle pisteelle laskettiin vielä poikkeavuusluku. Se muodostettiin summaamalla euklidinen etäisyys lähimpään kolmeen pisteeseen diffuusiokuvauksen toisen, kolmannen ja neljännen ominaisvektorin muodostamassa diffuusioavaruudessa. Tämän etäisyystiedon avulla voitiin piste luokitella poikkeavuudeksi sekä piirtää poikkeavuuskartta.

Kuvassa 9.3 on resurssista 1 muodostettu poikkeavuuskartta. Kuvaajassa vaaka-akselille on sijoitettu resurssille kohdistuneet HTTP-kyselyt ja pystyakselin mittana on poikkeavuus. Tässä pystyakseli ilmaisee sen kuinka poikkeava kukin piste on vertailujoukon sisällä. Jokaiselle kuvaajalle on vielä määritetty keskihajonta ja raja-arvo, jonka ylittäneet pisteet merkitään poikkeaviksi. Tämä raja-arvo vaihtelee tapauskohtaisesti ja sen arvo on kolme kertaa keskihajonta. Keskihajonta on merkitty kuvaajaan katkoviivalla ja raja-arvo yhtenäisellä viivalla. Esimerkiksi resurssin 1 poikkeavuuskartassa 9.3 on nähtävissä viisi poikkeavuudeksi merkittyä pistettä.

Jokainen diffuusiokartan luomiseen käytetty piste pystytään palauttamaan siihen HTTP-kyselyyn, josta se on muodostettu. Esitetyissä kuvaajissa palauttamiseen tarvittavat tiedot on merkitty niihin pisteisiin, jotka ylittävät raja-arvon. Otetaan esimerkiksi kuvaajan 9.5 ainoaksi merkitty poikkeama, jolla on arvot ovat 7,3 ja 14594. Näistä ensimmäinen arvo ilmoittaa sen tiedoston järjestysnumeron, josta kyseinen

HTTP-kysely löytyy. Seuraava luku puolestaan kertoo sen palvelimen järjestysnumeron, johon kysely on ohjattu. Näiden kahden luvun avulla voidaan yksilöidä tietty lokitiedosto. Viimeisin luku sisältää rivinumeron kyseisen lokitiedoston sisällä.

9.2 Tulokset

Analysoitavassa materiaalissa resurssilla 1 (kuva 9.3) oli viisi HTTP-kyselyä, jotka merkittiin poikkeaviksi. Tutkimalla diffuusiokuvauksen luomiseen käytettyä lokia, ja palauttamalla HTTP-kyselyt alkuperäiseen muotoon, pystyimme paikallistamaan nämä kyselyt ja vertaamaan niitä lokin muihin kyselyihin. Resurssin 1 tapauksessa neljä poikkeavaksi merkittyä kyselyä sisälsivät sellaisen GET-parametrin, jota ei esiintynyt muissa kyselyissä. Parametri vaikutti olevan jonkinlainen tunniste, jolla käyttäjä oli mahdollista yksilöidä. Jäljelle jääneessä kyselyssä esiintyi useita eri GET-parametreja, joita kyseinen resurssi ei käsittele. Tämä on saattanut yksinkertaisesti johtua siitä, että käyttäjä on kopioinut Web-sivun osoitteen selaimeensa virheellisesti.

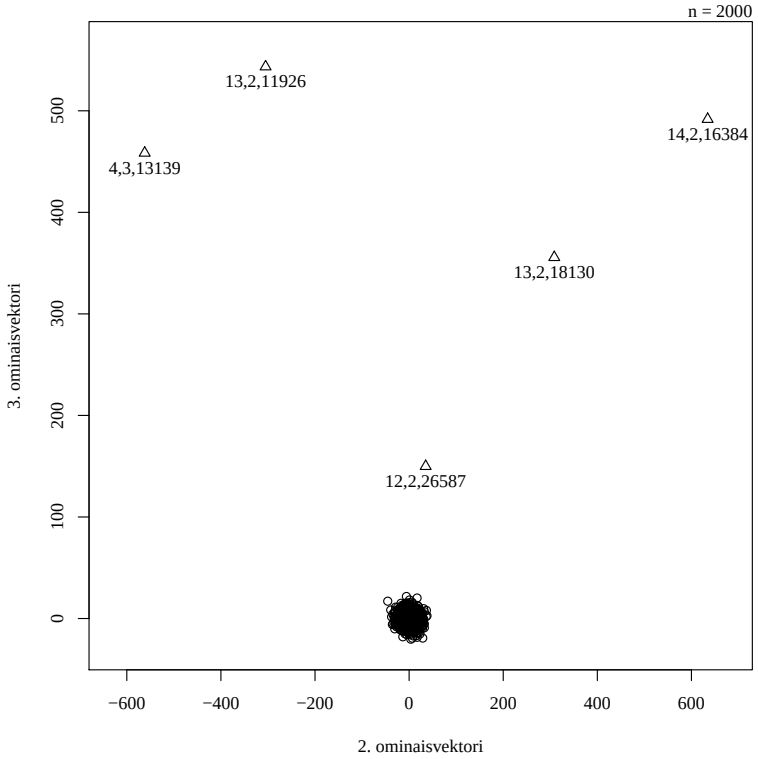
Kuvassa 9.5 on nähtävillä resurssin 8 poikkeavuuskartta. Kyseisessä otannassa oli mukana 181 pistettä ja näistä kaksi todettiin poikkeavaksi. Resurssi 8 tarjosi vain staattista sisältöä, joten GET-parametrin jälkeistä kyselyosuutta ei HTTP-kyselyissä pitänyt olla. Yhdessä poikkeavassa kyselyssä oli GET-parametrin resurssipolun jälkeen kuitenkin kysymysmerkki ja tämän perässä oli pyynnön tehneen alustan verkko-osoite. Tämän poikkeavan kyselyn tunnistaminen oli helppoa, sillä se oli ainoa kysely, jolle n-gram -analyysi tuotti nollasta poikkeavia arvoja. Resurssille tehty kysely oli selkeästi virheellinen, sillä merkitty verkko-osoite oli privaattiverkon osoite. Kyselyssä käytetty alusta oli myös ainoa laatuaan. Toinen poikkeavaksi merkitty kysely oli myös helppo tunnistaa, koska siinä käytetty metodi oli muista poiketen HEAD. Tarkempi analyysi osoitti, että kysely oli peräisin automaattisesta robotista, joka indeksoi Web-sivuja.

Resurssissa 337 (kuva 9.7) poikkeavuuksia havaittiin kuusi kappaletta. Näistä neljässä oli GET-parametrina samanlainen tunnistetieto kuin resurssin 1 kolmessa havaitussa poikkeavuudessa. Yksi poikkeavuus taas oli saman käyttäjän aiheuttama kuin resurssilla 8, sillä kysely tuli samasta verkko-osoitteesta ja näiden välillä ei ollut kulunut kuin muutama minuutti. Käytetty alusta oli myös ainoa laatuaan ja kyselyn yhtenä parametrina oli sama verkko-osoite. Onkin hyvin todennäköistä, että kyselyt kohdistuivat väärään palveluun, koska kyselyiden oikeassa muodostuksessa ilmeni

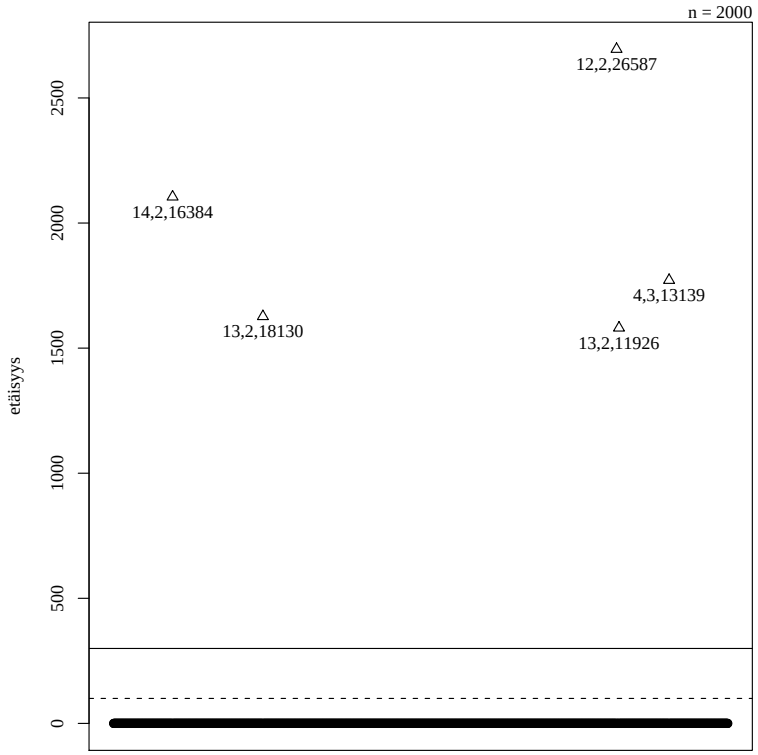
virheitä. Viimeisessä poikkeavassa kyselyssä oli sitten muista eroava määrä avain-arvo pareja.

Molempien kuvien 9.9 ja 9.11 poikkeavuudet yhtä lukuunottamatta aiheutuivat GET-parametrista, jota ei esiintynyt muissa kyselyissä. Ylimääräisen parametrin nimi oli sama kuin resurssissa 1, ja niistä jokainen tuli samasta IP-osoitteesta. Parametrin arvo kuitenkin vaihteli kyselystä riippuen. Tämä viittaa siihen, että kyselyt olivat tulleet välityspalvelimen kautta, joka jostakin syystä lisää kyseisen tunnistetiedon. Tämän takia sen kautta tulevat pyynnöt näyttävät erilaiselta. Palvelin kuitenkin oli osannut käsitellä nämä kyselyt normaalisti, sillä HTTP-vastauskoodi näiden kyselyiden osalta oli 200. Viimeinen poikkeavuus oli sitten samanlainen kuin kuvassa 9.5 eli kyselyssä oli pyynnön tehneen alustan verkko-osoite.

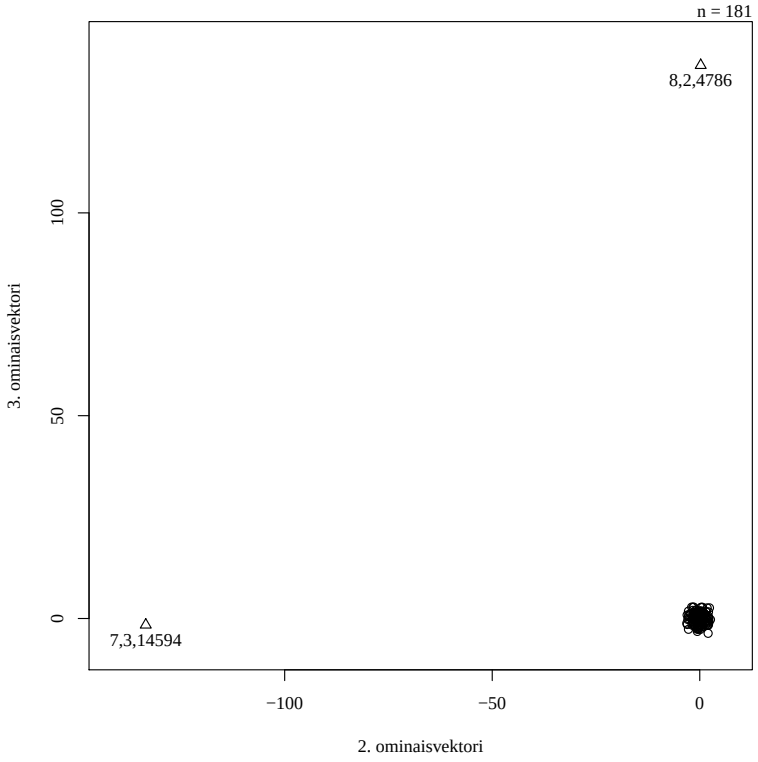
Viimeisen analysoidun resurssin (kuva 9.13) kahdesta havaitusta poikkeavuudesta yksi johtui samanlaisesta tunnistetiedosta, joka esiintyi myös aikaisemmissa kuvissa. Toinen poikkeavuus sisälsi myös ylimääräisen GET-parametrin, mutta aiemmista tapauksista poiketen tämä oli perinteisempi parametri, jota ei kuitenkaan olisi pitänyt välittää palvelimelle. Tästä huolimatta palvelin oli osannut käsitellä kyselyn.



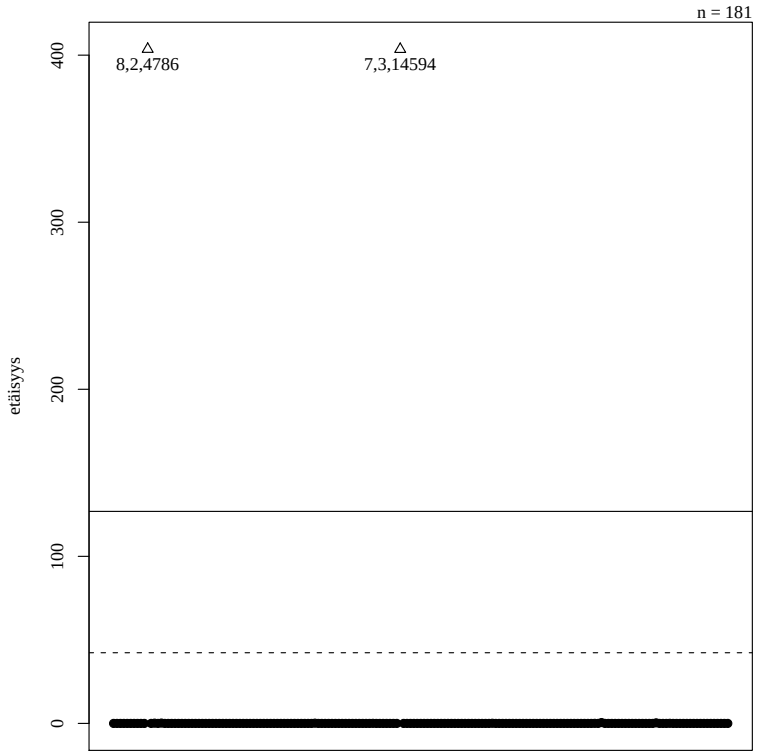
Kuva 9.2: Resurssin 1 täristetty diffuusiokartta.



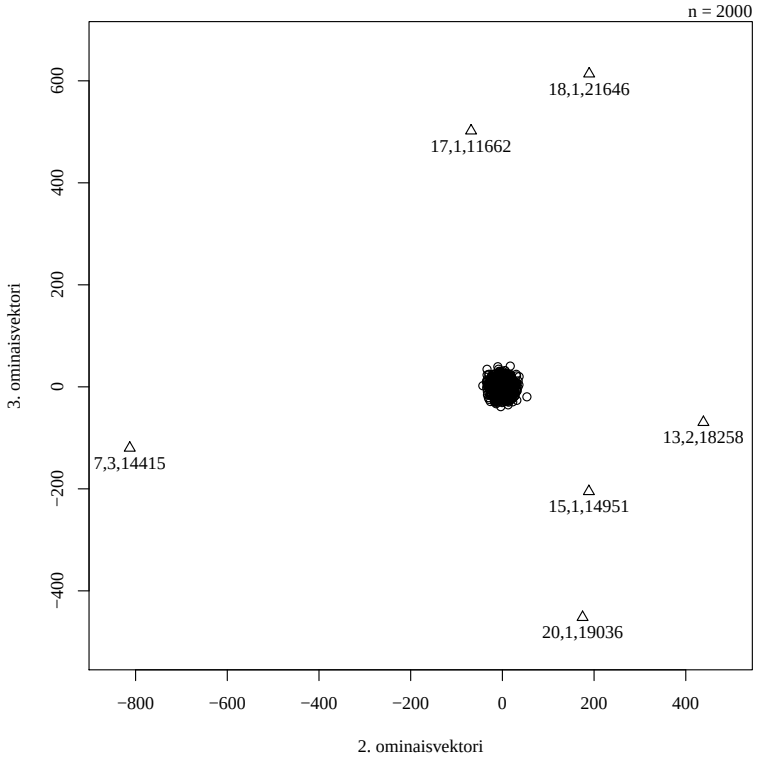
Kuva 9.3: Resurssin 1 poikkeavuuskartta.



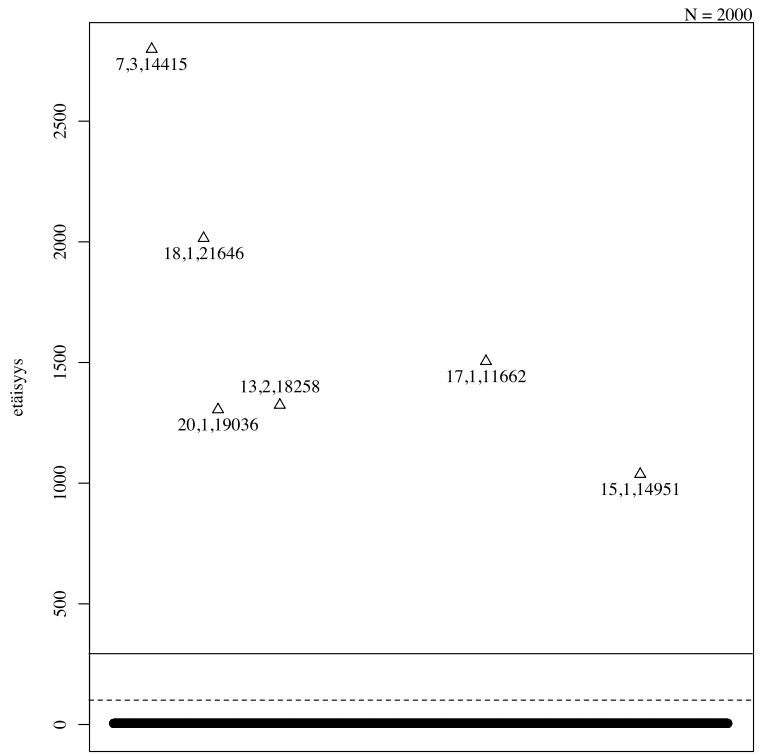
Kuva 9.4: Resurssin 18 täristetty diffuusiokartta.



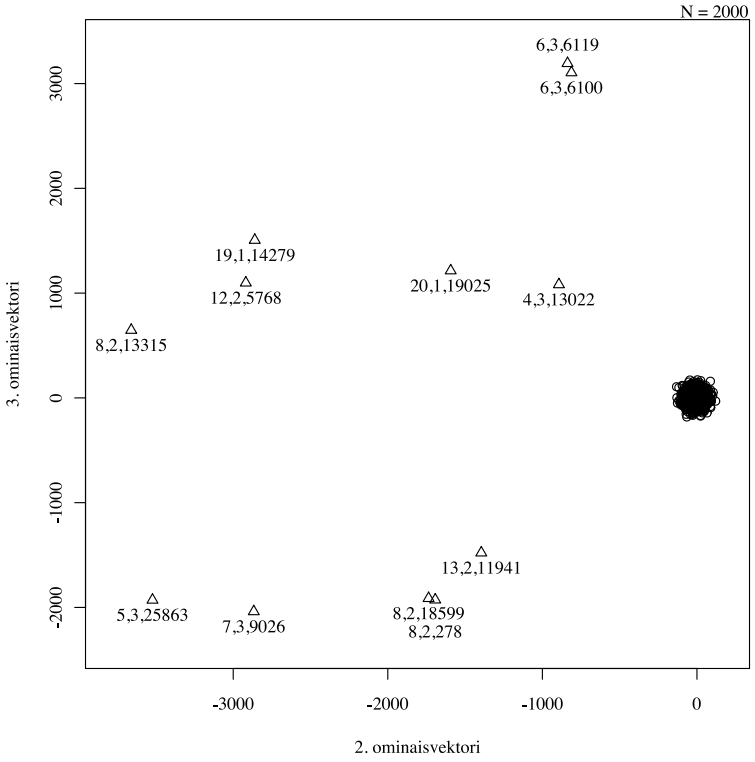
Kuva 9.5: Resurssin 18 poikkeavuuskartta.



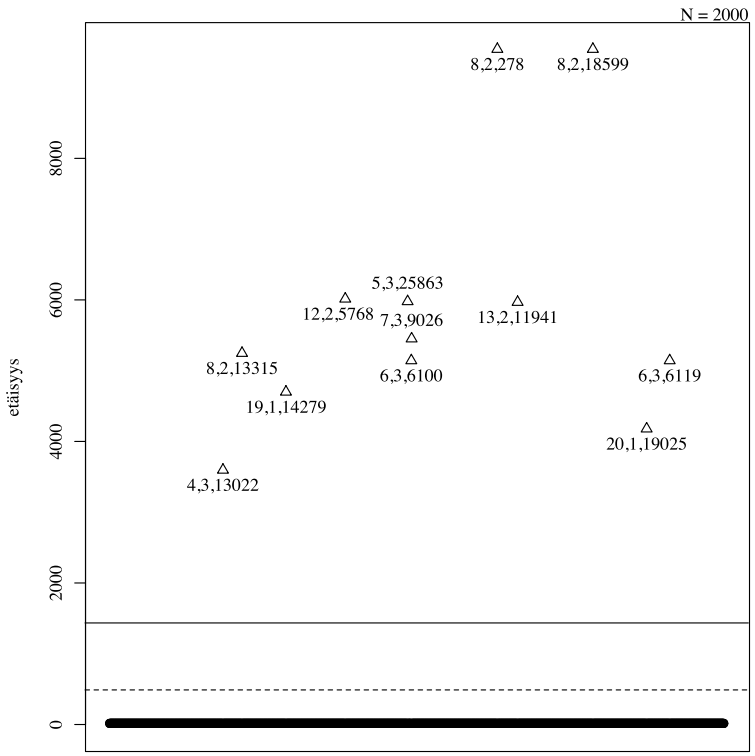
Kuva 9.6: Resurssin 337 täristetty diffuusiokartta.



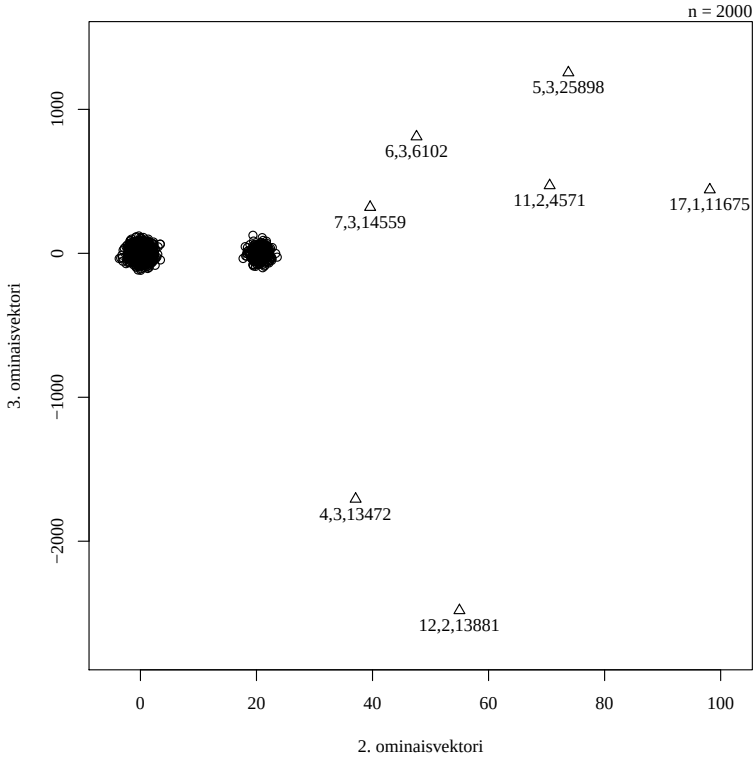
Kuva 9.7: Resurssin 337 poikkeavuuskartta.



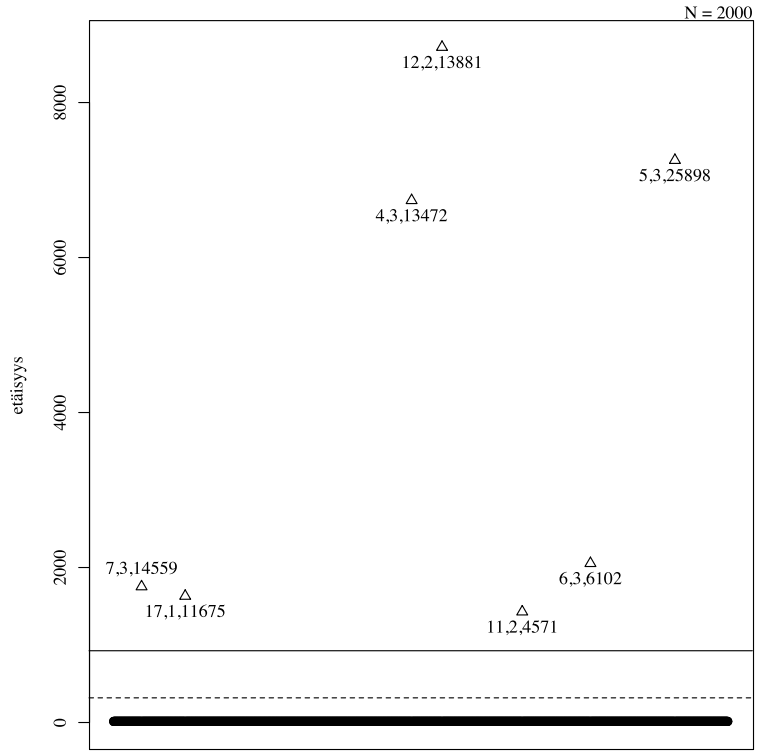
Kuva 9.8: Resurssin 721 täristetty diffuusiokartta.



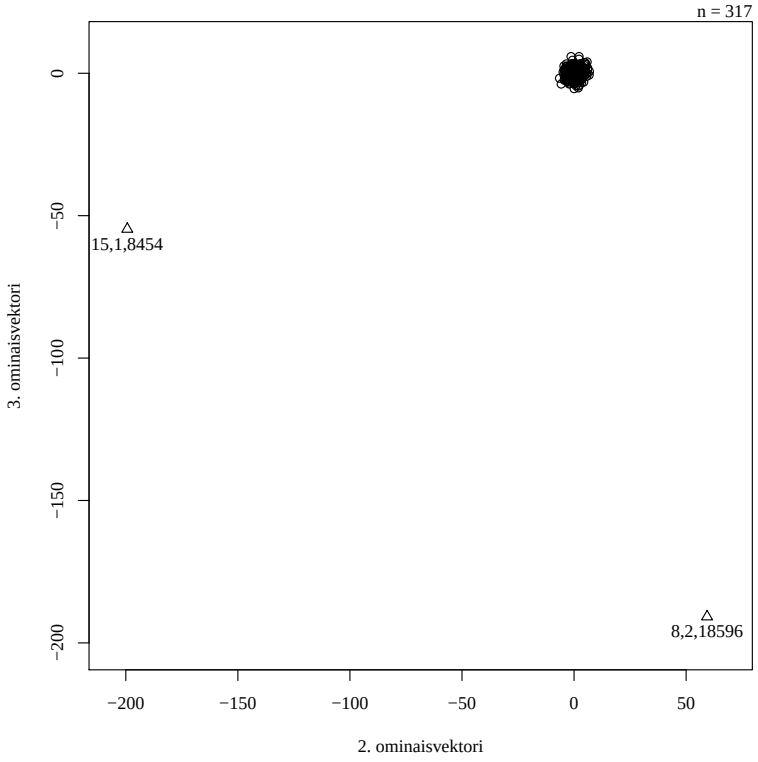
Kuva 9.9: Resurssin 721 poikkeavuuskartta.



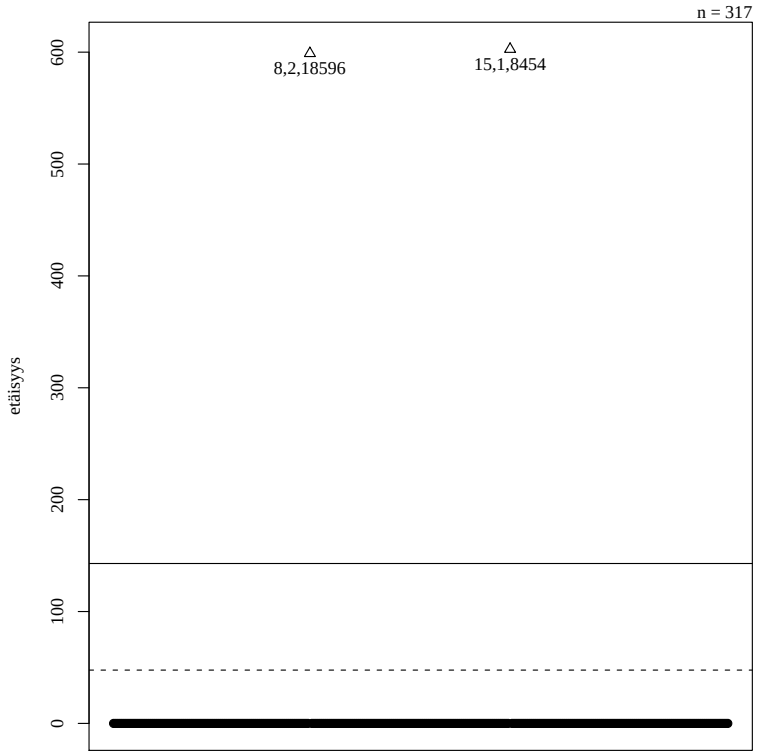
Kuva 9.10: Resurssin 723 täristetty diffuusiokartta.



Kuva 9.11: Resurssin 723 poikkeavuuskartta.



Kuva 9.12: Resurssin 882 täristetty diffuusiokartta.



Kuva 9.13: Resurssin 882 poikkeavuuskartta.

9.3 Johtopäätöksiä ja kehitysideoita

Analyysissä tuli hyvin esille se, että suurin osa liikenteestä oli hyvin samankaltaista. Tutkittavan materiaalin osalta tätä osattiin odottaa, mutta erilaisten pyyntöjen pieni valikoima yllätti siitä huolimatta. Ja vaikka kyseisestä otannasta ei varsinaisia tietoturvahyökkäyksiä löydetty, olimme tyytyväisiä saatuihin tuloksiin. Tulokset osoittivat selkeästi sen, että esikäsittelijä toimii halutulla tavalla ja valittujen menetelmien avulla poikkeavat pyynnöt pystytään paikallistamaan.

Valtavasta tietomassasta johtuen aineistosta pystyttiin käytettävissä olleen ajan puitteissa analysoimaan vain murto-osa. On siksi täysin mahdollista, että palveluun kohdistuneita tietomurtoyrityksiä jäi havaitsematta. Analysoidut palvelut saattoivat myös olla sellaisia, joihin ei kohdistu kiinnostusta hyökkääjien keskuudessa.

Vaikka tässä työssä diffuusiokuvausten laskentaan käytetyn algoritmin kompleksisuus rajoitti jonkin verran analyysin tekemistä, ei tämä käytännön toteutuksessa muodostunut kuitenkaan ongelmaksi. Reaaliaikaisessa toteutuksessa järjestelmälle voidaan opettaa etukäteen normaali käyttäytyminen diffuusiokuvauksen avulla, jonka jälkeen uusien pisteiden paikka vertailujoukossa lasketaan yksitellen. Tällöin algoritmin aikavaativuus riippuu ainoastaan opetusjakson koosta, eikä analysoitavien pisteiden määrästä. Uudet pisteet voidaan projisoida diffuusioavaruuteen esimerkiksi Nyströmin laajennuksen avulla, jota ei tähän työhön enää otettu mukaan.

Opetusjakson tarkempi määrittely tarjoaisi analyysin kannalta hyödyllisempää tietoa. Jatkokehityksessä seuraava askel olisi analysoida tarkemmin valikoitua materiaalia. Tämä voisi tarkoittaa esimerkiksi lokin analysointia sellaiselta ajanhetkeltä, jolloin tiedetään tietomurtoyritysten tapahtuneen. Tätä liikennettä voitaisiin verrata normaalitilanteeseen ja näin tunnistaa piirteitä, jotka erottavat poikkeavan liikenteen normaalista. Tällaisen aineiston käyttö tarjoaisi erinomaiset puitteet testata tässä työssä käytettyjä ratkaisuja.

Toinen kehityssuunta olisi kehittää esikäsittelymenetelmiä. Tämä voisi tarkoittaa esimerkiksi sitä, että palvelinlokista kerättäisiin enemmän piirteitä, joista voitaisiin muodostaa enemmän parametreja diffuusiokuvasta varten. Tällaisia piirteitä olisi esimerkiksi käyttäjän IP-osoite ja Web-selaimen tunnistetieto. IP-osoitteen luokittelussa voitaisiin hyödyntää esimerkiksi tarjolla olevia GeoIP-tietokantoja. Käyttäjän IP-osoitteesta voitaisiin näin selvittää esimerkiksi palveluntarjoaja, jonka kautta kysely tuli.

Käyttämämme esikäsittelijä ei myöskään hyödynnä kyselyiden ajallista riippuvuutta eikä ryhmittele kyselyitä muuten kuin resurssien perusteella. Esikäsittelijää

voisi laajentaa siten, että se mittaisi esimerkiksi samasta osoitteesta tulevien kyselyiden tiheyttä. Näin olisi mahdollista tunnistaa nykyistä paremmin purskeista liikennettä, joka usein liittyy palvelunestohyökkäyksiin.

Poikkeavuuksien tunnistamisessa voitaisiin käyttää Apache-lokin rinnalla myös muita tietolähteitä. Tällaisia olisivat esimerkiksi palvelinten muistinkulutus ja prosessorikuormitus. Lisäksi Web-palvelimen toiminnasta voisi kerätä myös sellaisia parametreja, joita ei yleisesti kerätä lokeihin. Esimerkkinä kyselyihin vastaamiseen kulunut aika ja resurssit. Tietoa voisi kerätä myös verkkokerrokselta, jolloin voitaisiin havaita sellaiset hyökkäykset, joita ei ole mahdollista tunnistaa sovelluskerroksella.

10 Yhteenveto

Monien perinteisten toimintojen siirtyessä kohti digitaalisia palvelumalleja, on erilaisten Web-sovellusten ja -palveluiden merkitys kasvanut huomattavasti. Lisäksi yhä useampi näistä sovelluksista toimii tietoturvan kannalta kriittisillä sektoreilla. Tämän muutoksen ovat havainneet myös tietomurtoja tekevät tahot, jotka pyrkivät kaikin keinoin hyödyntämään järjestelmistä löytyviä heikkouksia rikollisiin tarkoituksiin.

Tietoturva on aina kulkenut alan muuta kehitystä hiukan jäljessä, ja tätä seikkaa hyökkääjät ovat aina käyttäneet hyväksi. Nykytilanne noudattaa myös samaa kaavaa. Perinteisiltä tietoturvahyökkäyksiltä osataan jo kohtuullisesti suojautua ja löydetyt haavoittuvuudet korjataan riittävän nopeasti. Sovellusten tekijät ja palveluiden tarjoavat tuntevat myös sen verran hyvin oman toimintaympäristönsä, että suuria virheitä ei pääse syntymään.

Web-pohjaisissa palveluissa tilanne on toinen. Aihealue on vielä sen verran tuore, että sovellusten kehittäjät ja ylläpitäjät yrittävät vasta sopeutua siirroksen tuomiin muutoksiin. Uusien teknologioiden ja kehitysympäristöjen oppimiseen menee aina oma aikansa ja näiden turvalliseen käyttöön vielä pidempään. Tähän kun lisätään mukaan uudenlaiset arkkitehtuurimallit, joihin kehittäjät yrittävät sopeutua, niin on varmaa, että sovelluksiin eksyy erilaisia haavoittuvaisuuksia.

Tähän muutoksen tuomaan epävarmuuteen tietomurtoja ja -varkauksia tekevät hyökkääjät ovat iskeneet. Erilaisia uusia hyökkäystapoja kehitetään jatkuvasti ja väärinkäytettyjä haavoittuvaisuuksia löydetään lähes päivittäin. Monet näistä hyökkäyksistä eivät edes pyri ohittamaan asetettuja suojauksia, vaan ne käyttävät hyödykseen sovelluksen omia toimintoja, joita ei vain ole osattu riittävästi suojata. Osa näistä heikkouksista on myös niin syvällä käytetyssä tekniikoissa tai alustoissa, että suoraan niitä ei pystytä edes korjaamaan vaan korjaaminen vaatii kokonaan uudenlaista lähestymistapaa. Tämä ajaakin monet suunnittelijoista tekemään omia pikaisia korjauksia, joiden toimivuus on usein kuitenkin hyvin tapauskohtaista.

Uudenlaisten tietoturvahyökkäysten tunnistaminen vanhoilla työkaluilla on usein hyvin hankalaa tai lähes mahdotonta. Yhteistä erilaisille hyökkäyksille on, että ne poikkeavat yleensä jollakin tavoin normaalista tietomassasta. Ongelmana on se, kuin-

ka nämä poikkeavuudet eli anomaliat tunnistetaan suuresta tietomassasta riittävällä nopeudella ja tarkkuudella. Tätä tunnistamista varten onkin kehitetty useita erilaisia menetelmiä, jotka pyrkivät jollakin tavoin erottamaan hyökkäykset muusta liikenteestä.

Tämän tutkielman tavoitteena oli tunnistaa poikkeavat HTTP-kyselyt automaattisesti suuresta tietomassasta. Käytettäväksi menetelmäksi valittiin diffuusiokuvaukset, jotka tarjoavat nopean ja tehokkaan tavan tunnistaa poikkeavuudet järjestelmän normaalista käytöksestä. Materiaalin käsittelyä varten kehitettiin erillinen sovellus, joka suoritti lähdemateriaalille halutut toiminnot. Varsinainen analyysi suoritettiin Matlab-sovelluksessa, jonka jälkeen poikkeaviksi merkityt pisteet käytiin yksitellen läpi.

Saadut tulokset osoittavat, että diffuusiokuvausmenetelmä soveltuu hyvin tämän tyyppisen materiaalin analysointiin. Poikkeaviksi havaitut kyselyt erosivat myös todellisuudessa normaalista tietomassasta. Tulokset eivät kuitenkaan osoita sitä, että diffuusiokuvaus olisi paras valinta käytännön sovelluksiin. Tätä varten tarvitaan vielä paljon jatkotutkimusta ja vertailua muihin menetelmiin, johon ei tässä työssä perehdytty ollenkaan. Alustavat tulokset antavat kuitenkin uskoa siihen, että tulevaisuudessa diffuusiokuvausten käyttö on mahdollista eri sovellusalueilla.

Lähteet

[1] J. McGovern, S. Tyagi, M. Stevens, and S. Mathew, *Java Web Services Architecture*. Morgan Kaufmann Publishers, 2003.

[2] M. O'Neill *et al.*, *Web Services Security*. McGraw-Hill Osborne, 2003.

[3] "April 2010 web server survey." `http://news.netcraft.com/archives/2010/04/15/april_2010_web_server_survey.html`. Viitattu 11.5.2010.

[4] "OS, Web Server and Hosting History for www.jyu.fi." `http://uptime.netcraft.com/up/graph?site=www.jyu.fi`. Viitattu 8.7.2010.

[5] "About the Apache HTTP Server Project – The Apache HTTP Server Project." `http://httpd.apache.org/ABOUT_APACHE.html`. Viitattu 8.7.2010.

[6] "PHP: CGI and command line setups." `http://www.php.net/manual/en/install.unix.commandline.php`. Viitattu 11.5.2010.

[7] "Microsoft Internet Information Services." `http://www.iis.net/`. Viitattu 20.6.2010.

[8] "Vulnerability Report: Microsoft Internet Information Services (IIS) 7.x." `http://secunia.com/advisories/product/17543/`. Viitattu 20.6.2010.

[9] "Zope community." `http://www.zope.org/`. Viitattu 27.6.2010.

[10] "Plone community." `http://plone.org/`. Viitattu 27.6.2010.

[11] J. Hawley, "GeoDNS – Geographically-aware, protocol-agnostic load balancing at the DNS level," in *Proceedings of the Linux Symposium*, pp. 123–130, Linux Symposium Inc., heinäkuu 2009.

[12] "Akamai Technologies – Web Application Accelerator." `http://www.akamai.com/html/technology/products/waa.html`. Viitattu 29.6.2010.

[13] S. Mukkamala and A. H. Sung, "A comparative study of techniques for intrusion detection," *Tools with Artificial Intelligence, IEEE International Conference on*, vol. 0, p. 570, 2003.

[14] S. Bhasin, *Web Security Basics*. Course Technology Ptr, 2002.

[15] J. Mirkovic, S. Dietrich, D. Dittrich, and P. Reiher, *Internet Denial of Service: Attack and Defense Mechanisms (Radia Perlman Computer Networking and Security)*. Upper Saddle River, NJ, USA: Prentice Hall PTR, 2004.

[16] CERT, "Denial of Service." `http://www.cert.org/tech_tips/denial_of_service.html`, 1999. Viitattu 20.10.2009.

[17] S. McClure, J. Scambray, and G. Kurtz, *Hacking Exposed*. McGraw-Hill Osborne, 5th ed., 2005.

[18] S. Ranjan, R. Swaminathan, M. Uysal, and E. Knightly, "DDoS-Resilient Scheduling to Counter Application Layer Attacks Under Imperfect Detection," in *INFOCOM 2006. 25th IEEE International Conference on Computer Communications. Proceedings*, pp. 1–13, toukokuu 2006.

[19] Fermentas Inc., "Wanted by the FBI – Saad Echouafni." `http://www.fbi.gov/wanted/fugitives/cyber/echouafni_s.htm`. Viitattu 27.10.2009.

[20] M. Gregg and S. Watkins, *Hack the Stack: Using Snort and Ethereal to Master the 8 Layers of an Insecure Network*. Syngress Publishing, 2006.

[21] K. Kaario, *TCP/IP-verkot*. Docendon, 2001.

[22] D. J. Bernstein, "SYN Cookies." `http://cr.yp.to/syncookies.html`, Helmikuu 2002. Viitattu 21.7.2010.

[23] S. W. R. Lippmann, D. Stetson, "The Effect of Identifying Vulnerabilities and Patching Software on the Utility of Network Intrusion Detection," in *Recent Advances in Intrusion Detection*, pp. 307–326, Springer Berlin / Heidelberg, 2002.

[24] "Common Vulnerabilities and Exposures." `http://www.cve.mitre.org/cve`. Viitattu 29.10.2009.

[25] Z. Bankovic, S. Bojanic, O. Nieto-Taladriz, and A. Badii, "Increasing detection rate of user-to-root attacks using genetic algorithms," in *SECUREWARE '07:*

Proceedings of the The International Conference on Emerging Security Information, Systems, and Technologies, (Washington, DC, USA), pp. 48–53, IEEE Computer Society, 2007.

[26] R. Cannings, H. Dwidedi, and Z. Lackey, *Hacking Exposed Web 2.0: Web 2.0 Security Secrets and Solutions*. McGraw-Hill Companies, 2008.

[27] S. Shreeraj, *Web 2.0 Security: Defending Ajax, RIA and SOA*. Course Technology, 2007.

[28] G. Lawton, "Web 2.0 creates security challenges," *Computer*, vol. 40, pp. 13–16, Oct. 2007.

[29] M. Endrei, J. Ang, A. Arsanjani, S. Chua, P. Comte, P. Krogdahl, D. M. Luo, and T. Newling, *Patters: Service-Oriented Architecture and Web Services*. IBM Redbooks, 2004.

[30] D. L. C. Andrews, *SQL Server Security*. McGraw-Hill / Osborne, 2003.

[31] Symantec, "Symantec Internet Security Threat report. Trends for July-December 2007." `http://www.symantec.com/business/index.jsp`. Viitattu 1.12.2009.

[32] Symantec, "Symantec Global Internet Security Threat Report. Trends for 2008." `http://www.symantec.com/business/index.jsp`. Viitattu 1.12.2009.

[33] "XSSed." `http://http://www.xssed.com/archive/special=1`. Viitattu 2.12.2009.

[34] A. Barth, C. Jackson, and J. C. Mitchell, "Robust defenses for cross-site request forgery," in *CCS '08: Proceedings of the 15th ACM conference on Computer and communications security*, (New York, NY, USA), pp. 75–88, ACM, 2008.

[35] Z. Mao, N. Li, and I. Molloy, "Defeating cross-site request forgery attacks with browser-enforced authenticity protection," pp. 238–255, 2009.

[36] "Web application security consortium." `http://www.webappsec.org`. Viitattu 9.12.2009.

[37] Web Application Security Consortium, "Web application security statistics." `http://projects.webappsec.org/Web-Application-Security-Statistics`. Viitattu 9.12.2009.

[38] "Snort." http://www.snort.org/. Viitattu 16.03.2010.

[39] K. Hwang, M. Cai, Y. Chen, and M. Qin, "Hybrid intrusion detection with weighted signature generation over anomalous internet episodes," *IEEE Trans. Dependable Secur. Comput.*, vol. 4, no. 1, pp. 41–55, 2007.

[40] L.-C. Wuu, C.-H. Hung, and S.-F. Chen, "Building intrusion pattern miner for snort network intrusion detection system," *J. Syst. Softw.*, vol. 80, no. 10, pp. 1699–1715, 2007.

[41] "Nikto." http://www.cirt.net/nikto2. Viitattu 9.12.2009.

[42] "Insecure.org, Top 10 Web Vulnerability Scanners." http://sectools.org/web-scanners.html. Viitattu 9.12.2009.

[43] "Nessus." http://www.tenablesecurity.com/nessus/. Viitattu 20.01.2010.

[44] "Top 100 network security tools." http://sectools.org/. Viitattu 20.01.2010.

[45] "Paros proxy." http://www.parosproxy.org/index.shtml. Viitattu 08.02.2010.

[46] "Modsecurity." http://www.modsecurity.org/documentation/modsecurity-apache/2.5.11/modsecurity2-apache-reference.html. Viitattu 20.01.2010.

[47] T. Powell, "Ajax: The complete reference." McGraw-Hill, Inc., 2008.

[48] B. Chess, Y. T. O'Neil, and J. West, "JavaScript hijacking." http://www.fortify.com/servlet/download/public/JavaScript_Hijacking.pdf, 2007. Viitattu 03.02.2010.

[49] J. Keogh, "Javascript demystified." McGraw-Hill, Inc., 2005.

[50] "JSON." http://www.json.org/. Viitattu 03.02.2010.

[51] "Adobe Flash Player Penetration." http://www.adobe.com/products/player_census/flashplayer/. Viitattu 18.02.2010.

[52] "XSS vulnerabilities in common Shockwave Flash files." http://docs.google.com/Doc?docid=ajfxntc4dmsq_14dt57ssdw. Viitattu 22.02.2010.

[53] S. Ford, M. Cova, C. Kruegel, and G. Vigna, "Analyzing and detecting malicious Flash advertisements," in *ACSAC*, pp. 363–372, IEEE Computer Society, 2009.

[54] "Google Web Toolkit Overview." http://code.google.com/intl/fi-FI/webtoolkit/overview.html. Viitattu 25.02.2010.

[55] "What's New in GWT 2.0." http://code.google.com/intl/fi-FI/webtoolkit/doc/latest/ReleaseNotes.html. Viitattu 26.02.2010.

[56] "Security for GWT Applications." http://code.google.com/intl/fi-FI/webtoolkit/articles/security_for_gwt_applications.html. Viitattu 26.02.2010.

[57] M. Almgren, H. Debar, and M. Dacier, "A lightweight tool for detecting web server attacks," in *In Proceedings of the ISOC Symposium on Network and Distributed Systems Security*, 2000.

[58] "Bro Intrusion Detection System." http://bro-ids.org/Overview.html. Viitattu 16.03.2010.

[59] M. Almgren and U. Lindqvist, "Application-integrated data collection for security monitoring," in *RAID '00: Proceedings of the 4th International Symposium on Recent Advances in Intrusion Detection*, (London, UK), pp. 22–36, Springer-Verlag, 2001.

[60] G. Vigna, W. Robertson, V. Kher, and R. Kemmerer, "A Stateful Intrusion Detection System for World-Wide Web Servers," in *Proceedings of the Annual Computer Security Applications Conference (ACSAC 2003)*, (Las Vegas, NV), pp. 34–43, December 2003.

[61] "Stat Framework." http://www.cs.ucsb.edu/~seclab/projects/stat/index.html. Viitattu 17.03.2010.

[62] S. Jayamsakthi and M. Ponnavaikko, "Risk mitigation for cross site scripting attacks using signature based model on the server side," in *IMSCCS '07: Proceedings of the Second International Multi-Symposiums on Computer and Computational Sciences*, (Washington, DC, USA), pp. 398–405, IEEE Computer Society, 2007.

[63] S. Y. Lim and A. Jones, "Network anomaly detection system: The state of art of network behaviour analysis," in *ICHIT '08: Proceedings of the 2008 International Conference on Convergence and Hybrid Information Technology*, (Washington, DC, USA), pp. 459–465, IEEE Computer Society, 2008.

[64] K. Wang and S. J. Stolfo, "Anomalous payload-based network intrusion detection," in *RAID*, pp. 203–222, 2004.

[65] S. Zanero, "Ulisse, a network intrusion detection system," in *CSIIRW '08: Proceedings of the 4th annual workshop on Cyber security and information intelligence research*, (New York, NY, USA), pp. 1–4, ACM, 2008.

[66] H. Yang, F. Xie, and Y. Lu, "Network anomalous attack detection based on clustering and classifier," pp. 672–682, 2007.

[67] S. S. Kim and A. L. N. Reddy, "Statistical techniques for detecting traffic anomalies through packet header data," *IEEE/ACM Trans. Netw.*, vol. 16, no. 3, pp. 562–575, 2008.

[68] Y. Hu and B. Panda, "A data mining approach for database intrusion detection," in *SAC '04: Proceedings of the 2004 ACM symposium on Applied computing*, (New York, NY, USA), pp. 711–716, ACM, 2004.

[69] J.-C. Park and B.-N. Noh, "Sql injection attack detection: Profiling of web application parameter using the sequence pairwise alignment," in *WISA*, pp. 74–82, 2006.

[70] F. Valeur, D. Mutz, and G. Vigna, "A Learning-Based Approach to the Detection of SQL Attacks," in *Proceedings of the Conference on Detection of Intrusions and Malware and Vulnerability Assessment (DIMVA)*, (Vienna, Austria), pp. 123–140, July 2005.

[71] G. Vigna, F. Valeur, D. Balzarotti, W. Robertson, C. Kruegel, and E. Kirda, "Reducing errors in the anomaly-based detection of web-based attacks through the

combined analysis of web requests and sql queries," *Journal of Computer Security*, vol. 17, no. 3, 2009.

[72] M. Cova, D. Balzarotti, V. Felmetsger, and G. Vigna, "Swaddler: An approach for the anomaly-based detection of state violations in web applications," in *Proceedings of the 10th International Symposium on RecentAdvances in Intrusion Detection (RAID)*, (Queensland, Australia), pp. 63–86, September 5–7, 2007.

[73] C. Kruegel, G. Vigna, and W. Robertson, "A multi-model approach to the detection of web-based attacks," *Comput. Netw.*, vol. 48, no. 5, pp. 717–738, 2005.

[74] D. Mutz, F. Valeur, G. Vigna, and C. Kruegel, "Anomalous system call detection," *ACM Trans. Inf. Syst. Secur.*, vol. 9, no. 1, pp. 61–93, 2006.

[75] E. Tombini, H. Debar, L. Me, and M. Ducasse, "A serial combination of anomaly and misuse idses applied to http traffic," in *ACSAC '04: Proceedings of the 20th Annual Computer Security Applications Conference*, (Washington, DC, USA), pp. 428–437, IEEE Computer Society, 2004.

[76] R. Coifman and S. Lafon, "Diffusion maps," *Applied and Computational Harmonic*, vol. 21, no. 1, pp. 5–30, 2006.

[77] R. Coifman and S. Lafon, "Geometric harmonics: a novel tool for multiscale out-of-sample extension of empirical functions," *Applied and Computational Harmonic*, vol. 21, no. 1, pp. 31–52, 2006.

[78] B. Bah, "Diffusion Maps: Analysis and Applications," Master's thesis, Wolfson College, University of Oxford, 2008.

[79] E. Bingham and H. Mannila, "Random projection in dimensionality reduction: applications to image and text data," in *KDD '01: Proceedings of the seventh ACM SIGKDD international conference on Knowledge discovery and data mining*, (New York, NY, USA), pp. 245–250, ACM, 2001.

[80] "Log Files – Apache HTTP Server." `http://httpd.apache.org/docs/2.2/logs.html`. Viitattu 25.5.2010.

[81] ISO, *8601:2004(E): Data elements and interchange formats — Information interchange — Representation of dates and times*. International Organization for Standardization, Geneve, Sveitsi, 2004.

[82] S. P. Jones, *Haskell 98 Language and Libraries: the Revised Report*. Helmikuu 1999.

[83] Free Software Foundation, "GNU General Public License, version 3." `http://www.gnu.org/licenses/gpl.html`, Kesäkuu 2007. Viitattu 12.2.2010.