Antti Juvonen

Poikkeavuuksien havaitseminen WWW-palvelinlokidatasta

Tietotekniikan
(Mobiilijärjestelmät)
pro gradu -tutkielma
12. toukokuuta 2011

JYVÄSKYLÄN YLIOPISTO
TIETOTEKNIIKAN LAITOS

Jyväskylä

Tekijä: Antti Juvonen
Yhteystiedot: antti.k.a.juvonen@jyu.fi
Työn nimi: Poikkeavuuksien havaitseminen WWW-palvelinlokidatasta
Title in English: Anomaly Detection from WWW Server Log Data
Työ: Tietotekniikan (Mobiilijärjestelmät) pro gradu -tutkielma
Sivumäärä: 85
Tiivistelmä: Nykyajan web-palvelut ovat dynaamisia ja avoimia. Tämä antaa suurelle joukolle käyttäjiä mahdollisuuden päästä käsiksi palveluun ja sen sisältämään tietoon. Samalla avautuu uusia mahdollisuuksia toteuttaa hyökkäys. Tietoturvan pitäminen riittävällä tasolla on kilpailua aikaa vastaan. Poikkeavuuksien havaitsemisjärjestelmillä pystytään kuitenkin havaitsemaan ennestään tuntemattomat hyökkäykset ja muu epänormaali toiminta ja siten pitämään tietoturva hyvällä tasolla. Tutkimuksessa sovellettiin n-grammianalyysia, tukivektorikonetta ja diffuusiokarttoja esikäsitellyn verkkodatan analysointiin. Kaikilla menetelmillä saatiin lupaavia tuloksia, mutta reaaliaikainen järjestelmä vaatii vielä jatkokehitystä.
English abstract: Modern web services are very dynamic and open to all. This means that a big group of users have access to the service and the information included. Therefore, there are also new possibilities for attacking the system. Keeping the information security at a good level is a race against time. However, using anomaly detection it is possible to detect previously unknown attacks as well as other abnormal activities. Because of that the security will not be compromised. In this study n-gram analysis, support vector machine and diffusion map were used in finding anomalies from preprocessed network data. All the methods gave promising results, but more research is needed for an efficient real-time system.
Avainsanat: poikkeavuus, n-grammi, tukivektorikone, diffuusiokartta, koneoppiminen, tietoturva, hyökkääjän havaitseminen
Keywords: anomaly, n-gram, support vector machine, diffusion map, machine learning, information security, intrusion detection

Kiitokset

Haluan kiittää perhettäni saadusta tuesta työni aikana, erityisesti isääni, joka tuntui välillä olevan enemmän innostunut tutkimuksesta kuin minä itse. Siitä oli helppo saada motivaatiota.

Lisäksi kiitokset Joel Lehtoselle suuresta avusta, esikäsittelijän toteuttamisesta, tyhmiin kysymyksiin vastaamisesta ja yleisestä johdatuksesta aiheeseen ja työhön. Koko tutkimus perustuu Joelin ja Kristian Siljanderin tekemään pohjatyöhön, ja ilman niitä ei olisi ollut mahdollista hypätä mukaan työhön niin nopeasti. Kiitokset myös Tuomo Sipolalle diffuusiokarttasovelluksen toteuttamisesta ja myös muusta avusta ja opastuksesta, jota olen häneltä saanut. On ollut erittäin mukavaa työskennellä yhteistyössä, ja toivottavasti yhteistyö myös jatkuu.

Haluan myös suuresti kiittää ohjaajaani Timo Hämäläistä kaikesta ohjauksesta ja tuesta. Hänen avullaan sain myös tutkittavaksi oikeaa verkkodataa, jonka analysoiminen on ollut mielenkiintoista ja jatkuu edelleen.

Jyväskylässä toukokuussa 2011

Antti Juvonen

Lyhenteet

AIS Artificial immune system. Keinoimmuunijärjestelmä.

CLF Combined log format. Apache-palvelinohjelmiston lokiformaatti.

DoS Denial of Service. Palvelunestohyökkäys.

DDoS Distributed Denial of Service. Hajautettu palvelunestohyökkäys.

FP False positive. Hyökkäykseksi tulkittu normaali liikenne tai toiminta.

FN False negative. Normaaliksi tulkittu hyökkäävä liikenne tai toiminta.

GA Genetic algorithm. Geneettinen algoritmi.

IDS Intrusion detection system. Tunkeilijan havaitsemisjärjestelmä.

IPS Intrusion prevention system. Tunkeilijan estojärjestelmä.

IRC Internet Relay Chat. Eräs internetin pikaviestipalvelu.

PCA Principal component analysis. Pääkomponenttianalyysi.

R2L Remote to Local. Tietoturvahyökkäystyyppi.

SOM Self-organizing map. Itseorganisoituva kartta.

SVD Singular value decomposition. Singulaarihajotelma.

SVM Support Vector Machine. Tukivektorikone.

TN True negative. Oikein tunnistettu normaali liikenne.

TP True positive. IDS-järjestelmässä oikein tunnistettu hyökkäys.

U2R User to Root. Tietoturvahyökkäystyyppi.

Sisältö

Kuvat

1 Johdanto

Verkko ja verkkopalvelut ovat yhteiskunnassamme suuressa ja yhä kasvavassa roolissa. Yhtiöt ja hallitukset tarvitsevat yhä suurempia järjestelmiä. Samalle ne muuttuvat monimutkaisemmiksi. Tiedon varastointi ja salaaminen sekä käyttäjien luotettava autentikointi ovat muuttuneet todella haastaviksi. Internetin ja verkkopalvelujen käytön lisääntyessä myös hyökkäysten määrä on lisääntynyt räjähdysmäisesti. Tämä asettaa uusia haasteita, ja tavanomaiset tietoturvaratkaisut eivät enää riitä.

Periaatteessa mikä tahansa tietokone tai järjestelmä, oli se sitten kytketty johonkin verkkoon tai ei, on haavoittuvainen. Pelkkä tietokoneen jättäminen valvomatta voi riittää siihen, että hyökkääjä pystyy sitä käyttämään. Ongelmasta tekee kuitenkin paljon suuremman se, että tietokoneet ja palvelut ovat yhä enemmän verkossa ja täten kaikkien saatavilla. Niinpä potentiaalisten hyökkääjien määrä nousee, ja samalla yhä uusia haavoittuvuuksia ilmenee. Järjestelmien käyttäjäkunta ei ole enää kenenkään hallinnassa, joten tarvitaan yhä kehittyneempiä menetelmiä ja järjestelmiä tietoturvan ylläpitämiseksi.

Perinteisesti palomuurit ja autentikointi on riittänyt takaamaan hyvän tietoturvan. Nyt verkkopalvelut ovat kuitenkin jokaisen saatavilla, ja potentiaalisten hyökkäyskohteiden ja niiden suorittajien määrä on kasvanut suuresti. Niinpä palomuurilla ei enää voida suojata kaikkea, eikä kaikkia käyttäjiä voida välttämättä aukottomasti tunnistaa ja pääsyä tarvittaessa estää. Nämä menetelmät pyrkivät estämään hyökkäyksen tapahtumisen. Lisäksi tietoja voidaan salata, jolla pyritään välttämään hyökkäyksiä. Salattu tieto ei ole hyökkääjälle hyödyllinen, joten jo tämä tieto saattaa estää hyökkäyksen [1].

Huolimatta monista järjestelmistämmme, hyökkäyksiä ei koskaan voida estää täydellisesti [2]. Joskus haitallinen toiminta voi tapahtua lokaalisti tietyllä koneella, jolloin palomuuri ei auta sen havaitsemisessa. Aina on olemassa uudentyyppisiä hyökkäyksiä, joita ei pystytä vielä tunnistamaan. Lisäksi monet hyökkäykset muistuttavat käyttäjien normaalia toimintaa, joten niiden estäminen saattaa olla mahdotonta. Siksi edellä mainittujen menetelmien lisäksi on tärkeää pystyä myös havaitsemaan tapahtuneet tai tapahtumassa olevat hyökkäykset [1]. Siihen tarvitaan tunkeilijan havaitsemisjärjestelmiä, jotka pystyvät havaitsemaan tunnettuja ja myös

ennestään tuntemattomia hyökkäyksiä ja muuta haitallista tai poikkeavaa käyttäytymistä. Näiden järjestelmien tarkoituksena ei ole korvata muita järjestelmiä, vaan toimia niiden tukena yhteistyössä parantaen näin turvallisuutta. Muut järjestelmät kuten palomuurit eivät pysty tekemään samoja tehtäviä.

Analysoitavan tiedon määrä on kasvanut niin suureksi, että tavanomaiset lähestymistavat eivät enää ole tarpeeksi tehokkaita. Hyökkäyksiä on jo pitkän aikaa pystytty havaitsemaan vertaamalla verkon toimintaa tiettyihin ennalta määriteltyihin sääntöihin, mutta se ei tarjoa ratkaisua uusien hyökkäysten löytymiseen. Suurten tietomassojen ja monimutkaisten ja hyvin piilotettujen hyökkäysten löytämiseksi on käytetty kehittyneempiä tekniikoita, kuten tekoälyä, koneoppimista ja tiedonlouhintaa. Hyviä esiprosessointitekniikoita ja analysointialgoritmeja käyttäen suurestakin datamassasta voidaan löytää mielenkiintoisia asioita, kuten hyökkäyksiä tai muita poikkeavuuksia. Esimerkiksi neuroverkot, geneettiset algoritmit ja tukivektorikoneet ovat saaneet näin tietoturvasta uuden alueen johon niitä voidaan soveltaa. Kyky oppia normaalia käyttäytymistä ja löytää poikkeamia, joita ei voida muuten havaita ovat oleellisen tärkeitä yhä haastavammaksi muuttuneessa tietoturvamaailmassa.

Näitä algoritmeja ja menetelmiä on tutkittu paljon. Useimmat niistä eivät ole sovellettavissa pelkästään tietoturvaan ja hyökkäyksien havaitsemiseen, vaan niillä on useita käyttökohteita, kuten kuvantunnistus tai dokumenttien luokittelu. Ne soveltuvat usein erittäin hyvin suurien datamassojen käsittelyyn. Tämän on erittäin tärkeää lokidatan määrien kasvaessa. Usein on käytetty valmiita datakokoelmia, joiden pohjalta tunnistustarkkuuksia on testattu. Tähän sisältyy kuitenkin useita ongelmia. Data ei välttämättä vastaa oikeaa verkkoliikennettä. Jos menetelmät opetetaan käyttäen tiettyä datasettiä ja niitä testataan samalla datalla, tulokset ovat usein hyviä. Toisaalta oikealla datalla tulokset saattaisivat olla toisenlaisia. Lisäksi eri tutkimusryhmien tulokset vaihtelevat suuresti osittain siksi, että datasta on analysoitu pieni osajoukko, ja jokainen ryhmä valitsee tämän joukon eri tavalla.

Yksi tämän tutkimuksen motivaatioista on se, että käytettävissä on suuri määrä oikeasta verkkopalvelusta kerättyä lokidataa. Vaikeutena on se, että minkäänlaista ennakkotietoa liikenteestä ei ole, vaan menetelmien pitää tässä tapauksessa suoriutua analysoinnista ohjaamattomasti. Toisaalta oikea verkkodata antaa hyvän kuvan siitä, minkälaista liikennettä verkkopalvelussa todella on. Niinpä sen analysointi on erittäin mielenkiintoista, vaikka ongelmia ja haasteita on paljon. Suuria ennakkooletuksia ei voi olla, koska data on ennestään tuntematonta. Lisäksi käytössä ei ole

kovin paljon tietoa. Tarkka lokitusaste saattaisi parantaa tunnistustarkkuutta mutta samalla nostaa lokidatan määrän liian suureksi niin, että sen kerääminen, varastointi ja analysointi ei olisi enää mielekästä. Tämän tutkimuksen tarkoituksena on testata oikealla lokidatalla esikäsittelyn ja erilaisten analysointimenetelmien ja algoritmien toimivuutta käytännön tilanteessa. Koska lokidataa kerättäisiin joka tapauksessa, tutkimustulosten pohjalta voidaan luoda järjestelmä, joka täyttää sille edellä määritellyn tehtävän, eli toimii lisäturvana koko järjestelmälle viemättä kuitenkaan mitään muuta ominaisuutta pois. Tarkoituksena on siirtyä askel kerrallaan kohti itsenäistä ja reaaliaikaista järjestelmää.

2 Verkkopalveluihin kohdistuvat hyökkäykset

Tietoturvan voidan ajatella rakentuvan kolmen peruskäsitteen ympärille. Näitä ovat tiedon saatavuus (engl. *availability*), luottamuksellisuus (*confidentiality*) ja eheys (*integrity*) [3]. Tiedon saatavuus tarkoittaa sitä, että tarvittavaan tietoon on mahdollista päästä käsiksi. Luottamuksellisuus puolestaan kuvaa sitä, että tieto ei päädy sellaisten tahojen käsiin, joille se ei kuulu. Eheys tarkoittaa tiedon muuttumattomuutta. Näitä perusperiaatteita pyritään horjuttamaan tietoturvahyökkäyksillä. Esimerkiksi palvelunestohyökkäyksellä voidaan haitata tiedon saatavuutta tai estää se kokonaan (ks. 2.3). Hyökkäyksiä on erilaisia, ja ne usein koostuvat useammasta vaiheesta. Niillä voidaan aiheuttaa joskus suurtakin vahinkoa, ja tietoturva onkin nykyajan monimutkaisissa verkoissa ja järjestelmissä yksi tärkeimmistä asioista ja samalla suurimmista ongelmista.

2.1 Hyökkäyksen vaiheet

Tietoturvahyökkäyksissä hyökkääjä saavuttaa tavoitteensa usean eri vaiheen kautta. Asaka ym. ovat jakaneet tyypillisen hyökkäyksen neljään eri vaiheeseen [4]:

- Tarkkailuvaihe (engl. *surveillance stage* tai *probing stage*)
- Toimintavaihe (*activity stage* tai *exploitation stage*)
- Tavoitevaihe (*mark stage*)
- Naamiointivaihe (*masquerade stage*)

Tarkkailuvaiheessa hyökkääjä tarkkailee hyökkäyksen kohteena olevaa järjestelmää yrittäen löytää tietoturva-aukkoja varsinaisen hyökkäyksen toteuttamiseksi. Tämä voi tarkoittaa esimerkiksi salasanojen murtamisyritystä [5]. Apuna voidaan käyttää tarkoitukseen sopivia ohjelmia tai komentoja [4]. Tämä vaihe voidaan myös käsittää omana hyökkäystyyppinään (ks. 2.2).

Kun haavoittuvuus on löydetty, voidaan sitä hyödyntäen päästä käsiksi järjestelmään toimintavaiheessa [5]. Tässä vaiheessa hyökkääjä saa pääsyn ja oikeudet joilla

voidaan toteuttaa itse hyökkäys. Varsinainen hyökkäys voi olla esimerkiksi DDoS-hyökkäys [5] (ks. 2.3) tai haitallisen shell-skriptin ajaminen kohdekoneessa [4]. Tämä vaihe voidaan edelleen tarkemmin jaotella erilaisiin hyökkäystyyppeihin, joista on kerrottu alla tarkemmin.

Edelliseen vaiheen jälkeen hyökkääjällä on mahdollisuus aiheuttaa haittaa järjestelmälle tai hankkia käsiinsä hänelle kuulumatonta tietoa. Tämän jälkeen hyökkääjä on siis päässyt varsinaiseen tavoitteeseensa. Toisaalta järjestelmään on voinut jäädä jälkiä hyökkäyksestä [4]. Tällaisia jälkiä voivat olla esimerkiksi lokitiedosto tai muuttunut salasanatiedosto. Tästä johtuu nimi *mark stage*.

Viimeisessä vaiheessa hyökkääjä pyrkii peittämään jälkensä, jotta hyökkäyksen tapahtumista ei huomata tai hyökkääjää ei voida jäljittää. Tämä voi tarkoittaa esimerkiksi sellaisen lokitiedoston poistamista tai muokkaamista, mikä sisältää tietoa hyökkääjästä tai hyökkäyksestä [5]. Varsinainen hyökkäykseen liittyvä toiminta ja siitä aiheutuva vahinko on siis jo tässä vaiheessa tapahtunut. Kuitenkin hyökkääjän kannalta on edullista estää hyökkäyksen paljastuminen tai ainakin viivästyttää sitä.

2.2 Tiedon urkinta

Tiedon urkinta (engl. *probing*) tarkoittaa samaa kuin tavallisen hyökkäyksen ensimmäinen vaihe eli tarkkailuvaihe (ks. 2.1). Hyökkääjä tutkii järjestelmässä olevia koneita tai palveluita löytääkseen haavoittuvuuden tai aukon, jota käyttäen varsinainen hyökkäys toteutetaan. Tiedusteluhyökkäyksillä voidaan kerätä tietoa esimerkiksi käyttöjärjestelmästä, verkon koneiden tai laitteiden versionumeroista tai verkon rakenteesta. Varsinkin vanhentuneet ohjelmistoversiot voivat avata hyökkääjälle mahdollisuuden päästä käsiksi järjestelmään tai vahingoittaa sitä. Tiedusteluhyökkäykset ovat hyökkäystyypeistä helpoimpia toteuttaa, koska ne eivät vaadi syvällistä tuntemusta tietotekniikasta tai kohteena olevasta järjestelmästä [6]. Siksi se onkin yleisin hyökkäystyyppi.

Urkinta voidaan toteuttaa käyttäen jotain ohjelmaa, joka tutkii verkon koneita etsien avoimia portteja tai muita haavoittuvuuksia. Myös taitamattoman hyökkääjän on mahdollista tutkia jopa satojan tuhansien koneiden verkkoa käyttäen valmiita ohjelmia [7]. Osaa näistä ohjelmista ei ole edes tarkoitettu hyökkäyksien tekoon. Ne kuitenkin keräävät tietoa esimerkiksi väärin konfiguroiduista verkkolaitteista, joten ne ovat hyödyllisiä hyökkäystä suunnittelevalle taholle.

Toisaalta tiedusteluhyökkäys voidaan tehdä hyväksikäyttäen ns. sosiaalista ma-

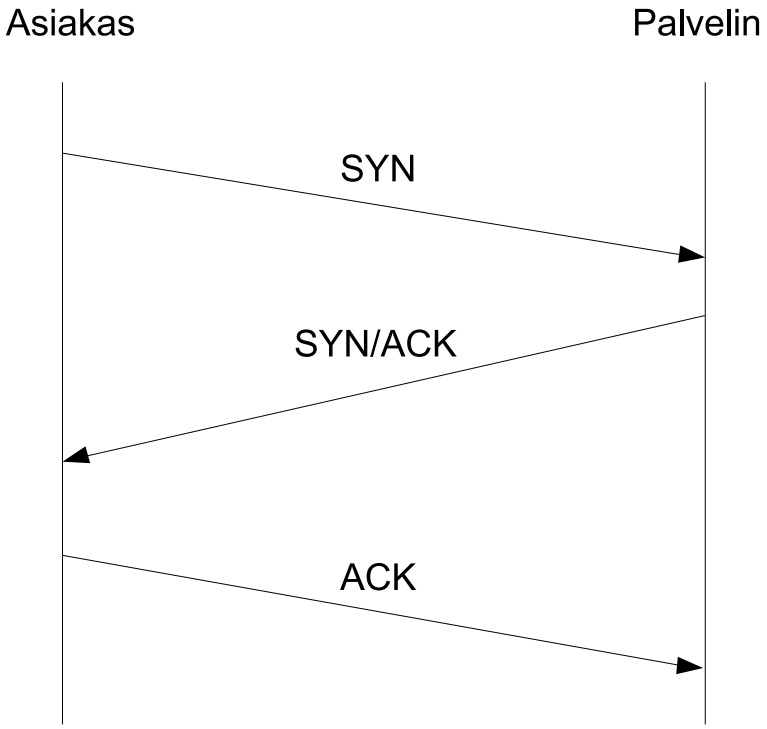
nipulointia (engl. *social engineering*) [6]. Sosiaalinen manipulointi on suuri riski tietoturvalle, sillä usein järjestelmän heikoin lenkki on sitä käyttävä tai hallinnoiva ihminen, ei niinkään itse käytetty teknologia [8]. Pelkkä tietoturvaohjelmisto ei riitä verkon tai järjestelmän tietoturvan varmistamiseksi, vaan esimerkiksi henkilöstön koulutus on tärkeää. Viime vuosina paljon huomiota on saanut ns. tietojen kalastelu (engl. *phishing*). Siinä hyökkääjä tekeytyy käyttäjälle luotettavaksi tahoksi ja yrittää näin saada itselleen luottamuksellista tietoa, esimerkiksi salasanoja [9]. Tällainen hyökkäys voi olla esimerkiksi sähköposti, joka uskottelee sen olevan järjestelmän ylläpitäjän lähettämä. Käyttäjän täytyy vastata siihen käyttäjätunnuksella ja salasanalla, esimerkiksi henkilöllisyyden varmistamiseksi tai käyttäjätilin toiminnan jatkamiseksi. Näin hyökkääjä pääsee käsiksi käyttäjätiliin. Suomessa useat käyttäjät menettivät tiedon kalastelijoille yhteensä yli 60 000 euroa vuonna 2005 [10]. Viimeistään silloin havahduttiin tiedon kalastelun riskeihin myös Suomessa. Sosiaalinen manipulointi on vakava uhka niin kauan, kuin osa käyttäjistä ei tunnista riskejä ja hyökkääjien toimintatapoja.

2.3 Palvelunestohyökkäykset

Palvelunestohyökkäyksessä (engl. *Denial of Service*, lyh. *DoS*) jokin verkkopalvelu tai resurssi käytetään loppuun niin, että pääsy kyseiseen palveluun estyy normaaleilta käyttäjiltä [6]. On myös mahdollista vain estää yhteys tiettyyn palveluun tai koneeseen kokonaan [7]. Apuna voidaan käyttää järjestelmän normaaleja toimintoja tai siitä löytyviä virheitä [6]. Varsinkin järjestelmän normaalia toimintaa hyväksikäyttävät hyökkäykset ovat ongelmallisia, koska niitä on hyvin vaikeaa havaita. Palvelunestohyökkäykset ovat yleisiä, koska on usein helpompaa tehdä jokin järjestelmä käyttökelvottomaksi kuin saada siihen käyttöoikeudet [11]. Syynä hyökkäykseen voi olla turhautuminen, pilailu tai kiristäminen sekä joskus myös esimerkiksi poliittiset syyt [12].

Esimerkkejä palvelunestohyökkäyksistä ovat SYN-hukuttaminen ja Smurf-hyökkäys [11]. SYN-hukuttaminen (engl. *SYN-flooding*) perustuu TCP-protokollan yhteydenmuodostuksen ns. kolmitiekättelyyn. Kolmitiekättelyn vaiheet ilmenevät kuvasta 2.1. Normaalisti asiakas lähettää palvelimelle SYN-paketin, johon palvelin vastaa SYN/ACK-paketilla. Kun asiakas vielä lähettää ACK-kuittauksen, on yhteys muodostettu. SYN-hukuttamisessa hyökkääjä kuitenkin väärentää asiakaskoneen osoitteen. Niinpä palvelin lähettää SYN/ACK-viestin väärään osoitteeseen. Mikä-

li väärennetyssä osoitteessa olisi toimiva asiakaskone, se vastaisi palvelimelle RTS-paketilla ilmoittaakseen, että se ei aloittanut yhteydenmuodostusta ja näin yhteydenmuodostus keskeytyisi. Hyökkääjä on kuitenkin valinnut osoitteen joka ei ole tavoitettavissa. Näin keskeneräinen yhteydenmuodostus laitetaan jonoon, josta se tyhjennetään vasta tietyn ajan kuluttua. Koska tämä jono on tyypillisesti pieni, pystytään varsin pienellä vaivalla tekemään palvelinkone täysin saavuttamattomaksi. Smurf-hyökkäyksessä puolestaan hyökkääjä lähettää ICMP-protokollan ECHO-viestin yleislähetyksenä kaikille verkon koneille. Osoite on kuitenkin taas väärennetty kohdekoneeksi, joten kaikki koneet lähettävät vastauksen tähän viestiin hyökkääjän sijasta kohdekoneelle. Suurissa verkoissa näin aiheutettu liikenne on valtava, ja se kuluttaa kohdekoneen resurssit loppuun. Oikein konfiguroiduilla laitteilla voidaan kuitenkin estää tämänkaltaiset hyökkäykset.



Kuva 2.1: Kolmitiekättely.

2.3.1 Hajautettu palvelunestohyökkäys

Nykyään paljon ongelmia aiheuttavat ns. hajautetut palvelunestohyökkäykset (engl. *Distributed Denial of Service*, lyh. *DDoS*). Periaatteessa hajautettu palvelunestohyökkäys tarkoittaa ainoastaan koordinoitua toimintaa, jossa useat koneet aiheuttavat

yhdessä suuren määrän liikennettä jollekin palvelulle [12]. Näin resurssit kuluvat loppuun ja pääsy palveluun estyy.

DDoS-hyökkäyksessä hyökkääjä tekee esimerkiksi käyttäjien koneelle asentuvan haittaohjelman avulla verkon, joka koostuu ns. orjakoneista eli *zombeista*. Jokainen kone aiheuttaa itsessään pienen määrän liikennettä esimerkiksi tietylle web-sivulle, usein käyttäjän tietämättä [11]. Tämä voi tarkoittaa yksinkertaisesti tietyn sivun lataamista uudelleen ja uudelleen [12]. Nykyään viestien välittämiseen orjaverkossa käytetään yleisimmin IRC-verkkoa [11] (engl. *Internet Relay Chat*). Tyypillinen hyökkäys sisältää siis kolme vaihetta [13]:

Orjaverkon kokoaminen
: Toisin kuin DoS-hyökkäys, DDoS-hyökkäys vaatii aikaa ja valmisteluja. Usein tämä tehdään täysin automaattisesti. Esimerkiksi jokin haittaohjelma saastuttaa automaattisesti suuren määrän koneita, jolloin ne ovat valmiita hyökkääjän käyttöön.

Käskyttäminen ja ohjaus
: Orjakoneille annetaan ohjeita ja käskyjä hyökkäyksen toteuttamiseksi. Usein tämä tehdään IRC-kanavalla, jolle koneet liittyvät automaattisesti.

Varsinainen hyökkäys
: Hyökkäys alkaa vasta, kun orjaverkko on valmis ja käsky hyökkäyksen aloittamiseksi on annettu. Joskus hyökkäys myös kestää vain tietyn ajan. Orjakoneet eivät siis aloita hyökkäystä omin päin.

Satojen tuhansien koneiden verkolla on mahdollisuus ajaa alas suurikin verkkosivusto tai -palvelu. Lisäksi varsinaisen hyökkääjän jäljittäminen on erittäin vaikeaa. Sitä voidaan kuitenkin yrittää esimerkiksi teeskentelemällä yhtä orjaverkon konetta ja olemalla näin yhteydessä ohjeita antavaan koneeseen [11]. Tämä voi kuitenkin olla mahdotonta orjaverkon monimutkaisen rakenteen vuoksi. Orjakoneet eivät siis ole suorassa yhteydessä hyökkääjään, vaan viestintä tapahtuu väliaskelien kautta.

Periaatteessa hajautettuja palvelunestohyökkäyksiä ei voida koskaan täysin estää [11]. Hajautetut palvelunestohyökkäykset ovat mahdollisia, koska verkon resurssit ovat aina rajalliset ja hyökkäyksen koordinointi on hajautettua. Tämän vuoksi myös hyökkäyksen torjunnan pitäisi olla hajautettua. Tämä kuitenkin tarkoittaa sitä, että hyökkäyksen estäminen jäisi sellaisten tahojen vastuulle, jotka eivät itse

suoranaisesti kärsi hyökkäyksestä ainakaan suurta vahinkoa [14]. Esimerkiksi orjakoneet eivät usein lakkaa toimimasta, vaan niiden toiminta voi olla lähes normaalia. Niinpä hyökkäysten torjumisen kustannukset jakaantuvat eri tavalla kuin hyökkäyksistä johtuva haitta. Toisin sanoen yksittäisten käyttäjien pitäisi yrittää estää hajautettujen palvelunestohyökkäysten toteuttaminen pitämällä oman koneensa tietoturva kunnossa, vaikka tällaisesta hyökkäyksestä ei välttämättä ole heille itselleen minkäänlaista haittaa.

2.4 User to Root

User to Root (lyh. *U2R*) tai User to super-user (lyh. *U2Su*) tarkoittaa hyökkäystä, jossa hyökkääjällä on ensin käyttöoikeus normaalit käyttäjäoikeudet omaavaan käyttäjätiliin. Tämä voidaan toteuttaa esimerkiksi hankkimalla käyttäjätunnus ja salasana hyödyntämällä tiedon kalastelua (ks. 2.2). Hyökkääjä hankkii sitten itselleen ylläpitäjän oikeudet hyödyntämällä jotain haavoittuvuutta [6].

Yleisin tapa toteuttaa U2R-hyökkäys on puskuriylivuodon hyväksikäyttäminen. Puskuriylivuoto tapahtuu, kun ohjelma yrittää kopioida liian paljon tietoa puskuriin tarkistamatta, että puskurissa riittää tilaa [7]. Näin osa datasta kopioidaan sille varatun muistualueen ulkopuolelle. Tarkasti puskurista ylivuotavaa dataa manipuloimalla hyökkääjä voi ajaa omia käskyjään kohdekoneen käyttöjärjestelmässä. Yleisin syy tämän haavoittuvuuden olemassaoloon ovat ohjelmointivirheet [6].

Puskuriylivuotohaavoittuvuuden lisäksi hyökkäys voidaan tehdä käyttäen oletuksia tai ennakkotietoa kohteena olevasta ympäristöstä, ja näin esimerkiksi ladata joitain moduuleja käyttöjärjestelmän ytimeen [7]. Lisäksi jotkut ohjelmat käsittelevät väliaikaistiedostoja huolimattomasti, avaten näin uusia haavoittuvuuksia.

User to Root -hyökkäykset ovat erittäin haasteellisia siksi, että niiden tunnistaminen tunkeilijan havaitsemisjärjestelmillä on todella vaikeaa. Osa U2R-hyökkäyksistä muistuttaa paljolti normaalia liikennettä, joten niiden löytäminen ei ole kovin luotettavaa [15].

2.5 Remote to Local

Remote to Local -hyökkäyksessä (lyh. *R2L*) hyökkääjä, joka pystyy lähettämään paketteja tiettyyn verkkoon mutta jolla ei ole käyttäjätiliä verkon koneeseen hankkii itselleen käyttäjäoikeudet jollain keinolla [7]. Joskus käytetään myös nimeä Remote

to User. Useimmin tämä tehdään hyväksikäyttämällä jo edellä mainittua puskuriylivuotoa tai sosiaalista manipulointia [6] (ks. kappaleet 2.4 ja 2.2).

Lisäksi esimerkiksi salasanan arvailu voi antaa oikeudet käyttäjätiliin. Tämä on mahdollista, koska monet käyttäjät käyttävät heikkoa ja helposti arvattavaa salasanaa. Mikäli hyökkääjä saa selville järjestelmän käyttäjätunnukset, hän voi yrittää erilaisia yleisiä salasanavaihtoehtoja. Tällaiset hyökkäysyritykset on kuitenkin helppo havaita tutkimalla epäonnistuneita kirjautumisyrityksiä tietyn ajan sisällä [7]. Kirjautumisten määrää voidaan rajoittaa, jolloin salasanan arvailua ei voida toteuttaa ollenkaan.

Salasana on mahdollista saada haltuunsa myös tutkimalla kohdekoneen näppäimistön painalluksia [7]. Tämä tosin vaatii jonkun ohjelman tai komponentin käyttämistä kohdekoneessa. Monissa käyttöjärjestelmissä on myös oletuksena päällä ns. vierastili, jolloin käyttöjärjestelmään pääsee kirjautumaan rajoitetuin oikeuksin ilman salasanaa. Tämän jälkeen hyökkääjä voi toteuttaa jatkotoimia saadakseen lisää oikeuksia järjestelmään.

3 Tunkeilijan havaitsemisjärjestelmät

Tunkeilijan havaitsemisjärjestelmä eli IDS (engl. *intrusion detection system*) kerää tietoa verkon tai tietokoneen eri osista ja analysoi sitä löytääkseen mahdollisia tietoturvahyökkäyksiä ja poikkeavuuksia [16]. IDS-järjestelmiä on kehitetty jo kymmeniä vuosia sen jälkeen kun ihmisen oma toiminta verkon valvomiseksi ei ollut enää tarpeeksi tehokasta. Järjestelmä koostuu eri osista ja se voi toimia monella tavalla. Sillä voidaan yrittää havaita tunnettuja hyökkäyksiä, tuntemattomia poikkeavuuksia tai molempia. IDS-järjestelmiä tutkitaan ahkerasti, sillä pelkillä palomuureilla ja muilla tietoturvaratkaisuilla tietoturvaa ei saada enää varmistettua.

3.1 Historiaa

Ennen varsinaisia tunkeilijan havaitsemisjärjestelmiä niiden tehtäviä hoitivat järjestelmän ylläpitäjät yksinkertaisesti valvomalla järjestelmän toimintaa tietokoneeltaan. Esimerkiksi epänormaali aktiivisuus harvoin käytetyssä tulostimessa saattoi aiheuttaa epäilyksiä. Valvonta oli manuaalista työtä, mutta se oli aikaansa nähden tarpeeksi tehokasta. 70- ja 80-luvuilla alettiin tallentaa verkon tietoja lokitiedostoihin. Nämä tiedostot tulostettiin ja niistä voitiin jälkeenpäin käydä läpi verkon toimintaa. Tulosteita oli kuitenkin usein niin paljon, että niistä tutkittiin vain tietoa jo toteutuneesta hyökkäyksestä. Toisin sanoen kun tiedettiin tapahtuneen hyökkäyksen ajankohta, voitiin siitä etsiä tarkempaa tietoa lokitiedostoista jälkeenpäin. Käynnissä olevan hyökkäyksen estäminen tai havaitseminen ei ollut mahdollista. Tallennustilan muuttuessa edullisemmaksi lokitiedostot voitiin tallentaa verkkoon, jossan niitä pystyttiin nyt analysoimaan ohjelmallisesti. Analysointi oli raskasta, joten prosessointi tehtiin yleensä öisin kun järjestelmän kuorma oli vähäinen. Tästä johtuen hyökkäykset voitiin havaita korkeintaan kerran päivässä. 90-luvulla pystyttiin jo analysoimaan dataa jossain määrin reaaliaikaisesti ja joissain tapauksissa jopa estämään hyökkäyksen toteutuminen. Ainakin sääntöpohjaisia suodatusjärjestelmiä pystyttiin käyttämään reaaliajassa. Tulevaisuuden haasteeksi jää järjestelmien toteuttaminen suurissa verkoissa [17].

Vuonna 1987 Dorothy Denning julkaisi kuuluisan mallin IDS-järjestelmästä. Jul-

kaisu sisälsi vain mallin, ei käytännön toteutusta. Järjestelmän nimi on IDES (engl. *Intrusion detection expert system*). Mallin perustana on kuusi eri komponenttia [18]:

Subjekti
Järjestelmässä varsinaisen toiminnan suorittava taho. Yleensä käyttäjä.

Objekti
Toiminnan kohteena oleva resurssi, kuten tiedosto, komento, laite tms.

Toimintarekisteri
Kaikesta järjestelmässä tapahtuvasta toiminnasta tallennetaan tieto. Toiminta voi olla esimerkiksi sisäänkirjautuminen, komennon suorittaminen tai tiedoston käyttäminen.

Profiili
Profiili on rakenne joka kuvaa subjektien objekteja kohtaan suorittamia toimintoja. Profiilit luodaan aluksi valmiista malleista. Tämän jälkeen malli luodaan automaattisesti esimerkiksi tilastollisin menetelmin käyttäjien toimintojen pohjalta.

Poikkeusrekisteri
Kun poikkeavaa toimintaa havaitaan, luodaan siitä tieto poikkeusrekisteriin.

Toimintasäännöt
Määrittää toiminnan tietyn ehdon täyttyessä. Toiminta voi tarkoittaa profiilien päivitystä, epänormaalin toiminnan havaitsemista, poikkeavuuksien yhdistämistä mahdollisiin hyökkäyksiin tai raporttien luomista.

Järjestelmässä toimintaa verrataan olemassa oleviin profiileihin. Analysointi perustuu sääntöihin. Perusideana on tarkkailla verkon toimintaa ja havaita poikkeuksia käyttäjien toiminnassa [18]. Mallissa ei määritetä sen tarkemmin käytännön toteutusta. Se on kuitenkin toiminut pohjana useille oikeille IDS-järjestelmille [19]. Mallia pidetään IDS-järjestelmien saralla uraauurtavana [20].

Vuonna 1998 Axelsson julkaisi tutkimuksen, jossa vertailemalla olemassa olevia IDS-järjestelmiä voitiin saada yleinen malli joka koostuu useista moduuleista [20]. Järjestelmästä kerätään dataa liittyen sen käyttäjiin ja toimintoihin. Nämä tallennetaan toimintalokiin. Lisäksi on tallennettu referenssitietoa. Tämä voi olla joko tietoa normaaleista käyttäjäprofiileista tai hyökkäyksistä. Tästä riippuen järjestelmä voi

joko tutkia väärinkäytöksiä tai poikkeuksia (ks. 3.6 ja 3.7). Näiden tietojen pohjalta tehdään poikkeuksien tai hyökkäysten havaitsemista analysointimoduulissa. Tarvittaessa tehdään hälytys, jota varten on oma moduulinsa. Lisäksi yhdellä moduulilla voidaan konfiguroida järjestelmän toimintaa.

IDS-järjestelmiä on pelkästään 80- ja 90-luvuilla toteutettu kymmeniä. Esimerkkejä näistä ovat vuoden 1988 hybridijärjestelmät Haystack [21] ja MIDAS [22] sekä uudempi vuoden 1997 EMERALD [23].

Vuosien saatossa kehityksen painopiste on siirtynyt lokaaleista tietokoneessa toimivista järjestelmistä verkkopohjaisiin ratkaisuihin. Lisäksi järjestelmistä pyritään tekemään enemmän hajautettuja. Niiden on myös tuettava yhä useampia käyttöjärjestelmiä. Samoina ovat kuitenkin pysyneet tavoitteet tehdä toteutuksia, jotka ovat reaaliaikaisia ja noudattavat hybridijärjestelmän periaatetta (ks. 3.8) [20].

3.2 Järjestelmän osat ja rakenne

Täysiverinen tunkeilijan havaitsemisjärjestelmä on iso kokonaisuus, joka sisältää useampia pienempiä komponentteja, joilla on omat tehtävänsä järjestelmässä. Yleisesti voidaan nämä osat jakaa neljään eri tyyppiin [24]:

Sensori

Sensorit ovat yksinkertaisimpia komponentteja, jotka yksinkertaisesti keräävät tietoa järjestelmästä ja sen verkkoliikenteestä, tallentavat sen lokeihin ja muuntavat tämän tiedon sellaiseen muotoon, että monitorit voivat käsitellä sitä.

Monitori

Monitorit ottavat vastaan esikäsiteltyä tietoa sensoreilta. Sen jälkeen esimerkiksi hyökkäyssääntöjä tai tekoälyä hyväksi käyttäen tutkitaan, onko hyökkäyksiä tai poikkeuksia olemassa. Nämä voidaan ohjata edelleen korkeamman tason monitoreille, tai niistä voidaan tehdä ilmoitus resolverille.

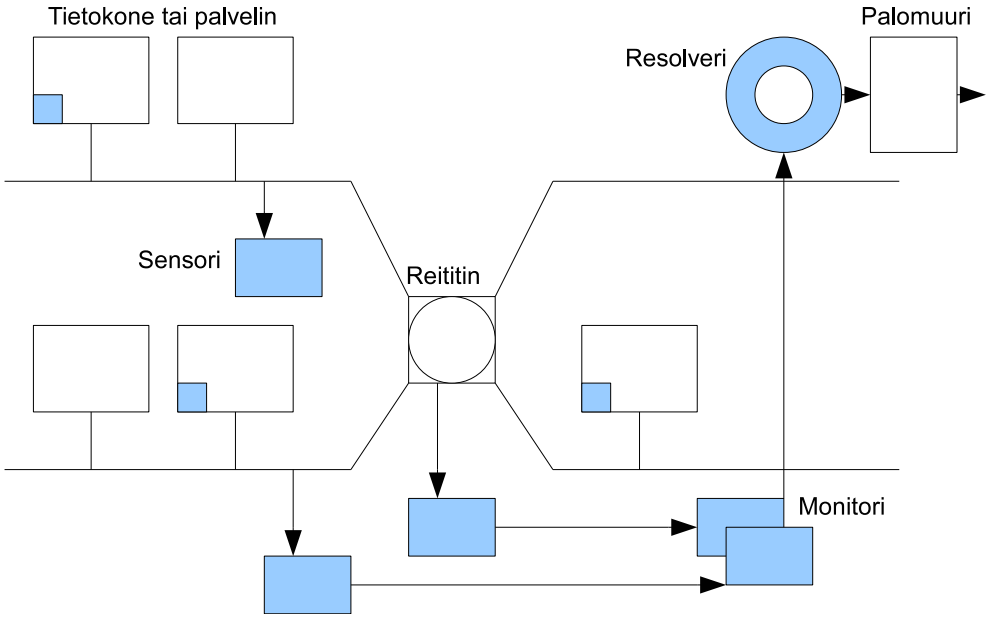
Resolveri

Päättää, mitä monitoreilta tuleville hälytyksille pitää tehdä. Ne voidaan ottaa talteen lokeihin, muiden järjestelmän osien toimintaa voidaan muuttaa, järjestelmään voidaan tehdä konfigurointimuutoksia hyökkäyksen haitan rajoittamiseksi ja ylläpitäjiä voidaan varoittaa mahdollisesta hyökkäyksestä.

Kontrolleri

Kontrolleri ohjaa järjestelmän toimintaa. Varsinkin suurissa hajautetuissa järjestelmissä on vaikeaa tai mahdotonta konfiguroida ja hallita manuaalisesti jokaisen muun komponentin toimintaa. Kontrolleri hoitaa tämän automaattisesti, ja lisäksi se tarjoaa keskitetyn rajapinnan järjestelmän ylläpitoon ja hallinnointiin.

Kuvassa 3.1 on esimerkkijärjestelmän rakenne [24]. Sensoreita voi olla verkossa tai reitittimen yhteydessä, mutta ne voivat sijaita myös lokaalisti yksittäisillä tietokoneilla. Monitori kerää tiedon ja resolveri päättää jatkotoimenpiteistä tarvittaessa. Kontrolleria ei ole kuvattu, eikä sen fyysinen paikka ole oleellinen.

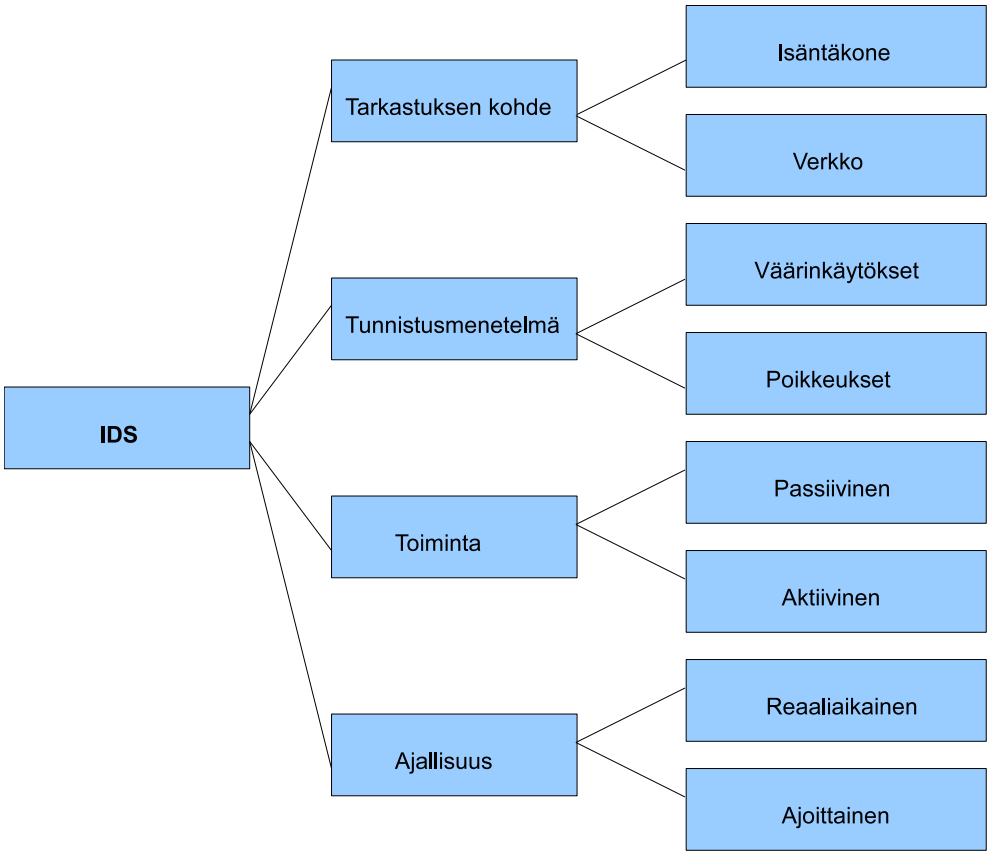


Kuva 3.1: IDS-järjestelmän rakenne.

3.3 IDS-järjestelmien tyyppejä

IDS-järjestelmiä voidaan jaotella eri tavoin. Kuvassa 3.2 on esitetty jaottelu neljän eri tekijän perusteella [5]. On olemassa myös muita kriteerejä joiden perusteella ryhmittely voidaan tehdä, mutta tähän on valittu niistä yleisimmät.

Isäntäkonetta tutkiva järjestelmä (*host-based*) tutkii toimintaa yhden koneen sisällä. Mikäli koneen käyttäjän toiminta poikkeaa normaalista, tapahtuu hälytys. Tärkeämmässä roolissa ovat nykyään kuitenkin verkkoa tutkivat järjestelmät (*network-*



Kuva 3.2: IDS-järjestelmien jaottelu.

based), jotka nimensä mukaisesti tutkivat verkossa tapahtuvaa liikennettä vaikuttamatta sen toimintaan [25]. Verkkopohjaisissa IDS-järjestelmissä on kuitenkin ongelmia. Osa hyökkäyksistä ei aiheuta ollenkaan verkkoliikennettä, ja lisäksi osa liikenteestä voi olla salattua [5]. Lisäksi erilaiset käyttöjärjestelmät monimutkaistavat hyökkäyksien havaitsemista. Edelle mainittujen lisäksi on myös olemassa järjestelmiä, jotka tutkivat sekä yksittäistä konetta että verkkoa [5].

Väärinkäytöksien ja poikkeuksien havaitsemisjärjestelmät sekä näitä molempia käyttävät hybridijärjestelmät on kuvattu kappaleissa 3.6 ja 3.7. Yleisesti ottaen väärinkäytöksien havaitsemismenetelmä tarvitsee etukäteistietoa hyökkäyksistä, mutta poikkeuksien havaitsemisessa sitä ei tarvita.

Hälytyksen sattuessa järjestelmä voi toimia ja passiivisesti tai aktiivisesti. Passiivinen järjestelmä vain kirjaa hyökkäyksen, mutta aktiivinen myös yrittää estää tai pysäyttää sen. Aktiivisia järjestelmiä kutsutaan myös IPS-järjestelmiksi (*intrusion prevention system*), jotka on kuvattu kappaleessa 3.9. Useimmat järjestelmät ovat kuitenkin passiivisia, koska väärien hälytysten aiheuttamat aktiiviset toiminnot haittaavat verkon toimintaa tarpeettomasti [25].

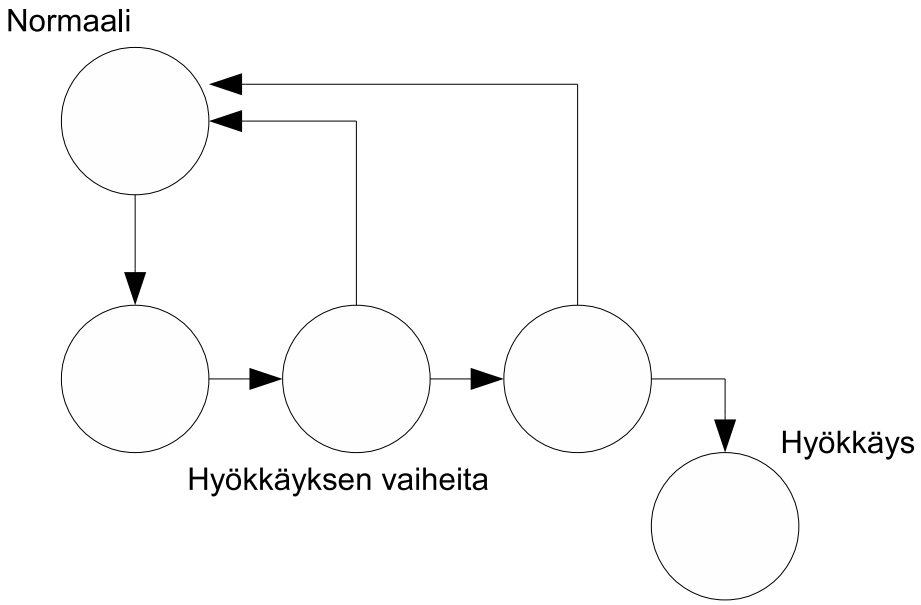
Jotkut järjestelmät analysoivat liikennettä vain tietyin väliajoin. Tällöin dataa kerätään tietyn aikaa, jonka jälkeen järjestelmä tutkii mahdollisia hyökkäyksiä. Näin ei kuitenkaan voida havaita hyökkäyksiä, jotka ovat vielä kesken. Lisäksi viive saattaa jättää hyökkääjälle aikaa piilottaa hyökkäyksensä ennen kuin järjestelmä ehtii havaita sitä [5]. Yksi suurimmista tavoitteista on toteuttaa reaaliaikainen järjestelmä, joka tutkii verkkoa jatkuvasti ja suorittaa hälytyksen heti hyökkäyksen tapahtuessa. Tämä on kuitenkin erittäin vaikeaa erityisesti suurissa verkoissa [5].

3.4 Tilallinen ja tilaton järjestelmä

IDS-järjestelmät voidaan edellisten jaotteluiden lisäksi jakaa kahtia tilattomiin ja tilallisiin järjestelmiin. Molemmat sisältävät hyötyjä ja haittoja.

Tilallisessa järjestelmässä pyritään yhdistämään alemman tason pieniä toimintoja suuremmiksi kokonaisuuksiksi esimerkiksi ajallisen riippuvuuden perusteella. Jos useita pieniä toimintoja tapahtuu lyhyellä aikavälillä, niiden merkityskin korostuu. Yksittäisten tapahtumien yhteyttä toisiinsa pyritään tutkimaan. Usein hyödynnetään tekoälyä ja koneoppimista, esimerkiksi Bayesverkkojen tai tilakoneiden kautta. Useista eri lähteistä tehdyt toiminnat otetaan huomioon. Tapahtumista voidaan rakentaa malleja, joissa tapahtumien suhteet toisiinsa havainnollistuvat [5].

Esimerkki tilakoneesta tässä yhteydessä on kuvassa 3.3 [24]. Tietyt toiminnot johtavat tilasiirtymiin. Jos tietyt tapahtumat sattuvat oikeassa järjestyksessä, tulkitaan toiminta hyökkäykseksi. Esimerkissä yksittäiset tapahtumat eivät siis yksin johda hälytykseen.



Kuva 3.3: Tilakone IDS-järjestelmässä.

Toinen vaihtoehto on ns. tilaton järjestelmä. Siinä tarkastellaan ainoastaan yksittäisiä tapahtumia ja yritellään luokitella niitä joko normaaleiksi tai poikkeaviksi. Poikkeuksien havaitsemisesta tehdään siis luokittelu- tai ryhmittelyongelma, jota voidaan yrittää ratkaista esimerkiksi matemaattisin menetelmin. Tätä varten kerättyä dataa tarvitaan riittävä määrä. Lisäksi tapahtumista pitää kerätä tarpeeksi piirteitä, jotta poikkeavuudet voidaan havaita. Tämä johtaa yleensä suuriin vektoreihin eli laskennallisesti ulottuvuuksien määrä kasvaa. Tämä puolestaan kasvattaa suuresti tarvittavan prosessoinnin määrää. Kuitenkin tilattomista järjestelmistä odotetaan parasta vaihtoehtoa reaaliaikaisen järjestelmän toteuttamiseksi [5].

Tilattomien järjestelmien suurin heikkous on se, että monet hyökkäykset sisältävät useita vaiheita, jotka eivät ole itsessään poikkeavia. Niinpä niiden havaitseminen tutkimalla vain yksittäisiä toimintoja voi olla mahdotonta, koska niiden välisiä riippuvuuksia ei oteta huomioon. Toisaalta hyökkäyksistä ei vaadita etukäteistietoa, sillä normaali liikenne voidaan oppia opetusdatasta ja poikkeamat havaita matemaattisesti koneoppimista hyödyntäen. Tilallinen järjestelmä on yleensä sopivampi väärinkäytön havaitsemisjärjestelmäksi, kun taas tilaton soveltuu paremmin poik-

keavuuksien havaitsemiseen (ks. 3.6 ja 3.7). Molemmat menetelmät sopivat tiettyyn tarkoitukseen, ja toistaiseksi molempia ominaisuuksia yhdistävää yksittäistä menetelmää ei ole onnistuttu kehittämään [5]. Näitä voidaan kuitenkin yrittää yhdistää hybridijärjestelmässä (ks. 3.8).

3.5 Suorituskyky ja tehokkuus

Tutkimuksessa on tärkeää arvioida, kuinka hyvin IDS-järjestelmä toimii. Näin järjestelmiä voidaan verrata keskenään ja niiden sopivuutta tehtäväänsä voidaan mitata.

Yleisesti käytetään seuraavia käsitteitä [5], jotka on myös kuvattu taulukossa 3.1:

True positive (TP): Oikein havaittu hyökkäys.

True negative (TN): Oikein tulkittu normaali liikenne tai toiminta.

False positive (FP): Hyökkäykseksi tulkittu liikenne, joka on kuitenkin normaalia.

False negative (FN): Normaaliksi tulkittu liikenne, joka on kuitenkin hyökkäys.

		Havaittu	
		Normaali	Poikkeava
Todellinen	Normaali	True negative	False positive
	Poikkeava	False negative	True positive

Taulukko 3.1: Arvioinnin peruskäsitteet.

Näistä voidaan johtaa useita yleisesti käytettyjä mittareita IDS-järjestelmän tehokkuudelle [5]:

$$\textit{True positive rate (TPR)} = \frac{TP}{TP + FN} = \frac{\textit{oikein havaitut hyökkäykset}}{\textit{kaikki hyökkäykset}} \tag{3.1}$$

$$\textit{False positive rate (FPR)} = \frac{FP}{TN + FP} = \frac{\textit{hyökkäyksiksi tulkitut normaalit}}{\textit{kaikki normaalit}} \tag{3.2}$$

$$\textit{True negative rate (TNR)} = \frac{TN}{TN + FP} = \frac{\textit{oikein tulkitut normaalit}}{\textit{kaikki normaalit}} \tag{3.3}$$

$$\textit{False negative rate (FNR)} = \frac{FN}{TP + FN} = \frac{\textit{normaaleiksi tulkitut hyökkäykset}}{\textit{kaikki hyökkäykset}} \tag{3.4}$$

Näistä voidaan edelleen johtaa:

$$\textit{Tarkkuus (accuracy)} = \frac{TP + TN}{TP + TN + FP + FN} = \frac{\textit{oikein tulkitut havainnot}}{\textit{kaikki havainnot}} \tag{3.5}$$

$$\textit{Täsmällisyys (precision)} = \frac{TP}{TP + FP} = \frac{\textit{oikein havaitut hyökkäykset}}{\textit{kaikki hyökkäyksiksi tulkitut havainnot}} \tag{3.6}$$

Näistä tarkkuus ilmaisee, kuinka hyvin järjestelmä on onnistunut luokittelemaan havainnot oikeisiin kategorioihin. Jokainen väärä tunnistus alentaa tarkkuutta [26]. Korkea täsmällisyys tarkoittaa sitä, että järjestelmä ei ole tulkinnut suurta määrää normaalia liikennettä hyökkäyksiksi. Tämä on tärkeää siksi, että väärin positiivisten joukosta voi olla vaikea löytää oikeita hyökkäyksiä.

Muita tärkeitä asioita järjestelmässä ovat esimerkiksi suorituskyky (*performance*) ja täydellisyys (*completeness*) [27]. Suorituskyky tarkoittaa yksinkertaisesti järjestelmän nopeutta. Liian hidas järjestelmä ei sovi reaaliaikaiseen käyttöön. Järjestelmän täydellisyyttä eli sitä, paljonko hyökkäyksiä jää löytymättä on vaikeaa mitata, koska oikeassa reaalimaailman datassa ei ole etukäteistietoa hyökkäyksistä [26]. Lisäksi tärkeitä ominaisuuksia ovat vikasietoisuus (*fault tolerance*) ja ajantasaisuus (*timeliness*) [26]. Myös itse IDS-järjestelmä voi olla altis ulkopuolisille hyökkäyksille, joten vikasietoisuus on tärkeää. Ajantasaisuus ei tarkoita vain järjestelmän nopeutta vaan sitä, kuinka nopeasti havaittuihin ongelmiin reagoidaan ja kuinka järjestelmä toimii datamäärän kasvaessa. Joskus tästä käytetään myös termiä skaalautuvuus (*scalability*) [5]. Erityisesti suuria tietomääriä käsiteltäessä on tärkeää, että tieto havaituista hyökkäyksistä liikku verkossa tarpeeksi nopeasti.

3.6 Väärinkäytön havaitseminen

Tässä väärinkäyttö tarkoittaa tunnettua hyökkäystä joka hyödyntää tunnettuja haavoittuvuuksia järjestelmässä [19]. Väärinkäytön havaitseminen (*misuse detection*) tarkoittaa sitä, että käytetään olemassa olevaa tietoa mahdollisista hyökkäyksistä. Liikennettä verrataan tämän tiedon pohjalta luotuihin sääntöihin, ja mikäli käyttäjän

toiminta täyttää jonkun hyökkäyksen tunnusmerkeistä, tehdään hälytys [6,28]. Tällainen järjestelmä havaitsee tehokkaasti hyökkäyksiä, mutta vain siinä tapauksessa, että poikkeukset tunnetaan. Malleja ei siis tehdä automaattisesti [29]. Tällöin hyökkäystiedon luomiseen tarvitaan ihmisten asiantuntemusta. Viime vuosiin asti väärinkäytön havaitsemisjärjestelmät ovat olleet kaupallisesti menestyneimpiä järjestelmiä [5]. Ne ovat yksinkertaisia toteuttaa, eivät vaadi kovin paljoa prosessoritehoa ja antavat vähän vääriä hälytyksiä, joten ne tuovat helposti lisäarvoa järjestelmän turvallisuudelle haittaamatta sen toimintaa juuri mitenkään.

Tilastollisiin menetelmiin ja koneoppimiseen perustuvat menetelmät voidaan opettaa vähitellen hyväksymään käyttäjän toiminta normaaliksi, mutta sääntöpohjaisissa järjestelmissä tätä vaaraa ei ole. Lisäksi väärinkäytön havaitseminen on yksinkertaisempaa kuin poikkeavuuksien havainnointi [28]. Koska käytetään tunnettuja sääntöjä, väärien positiivisten tunnistusten määrä on alhainen. Hyökkäyksen tapahtuessa on helppoa saada selville minkä tyyppinen hyökkäys on tapahtunut, koska jokainen hyökkäystyyppi on tarkasti määritelty säännöissä. Väärinkäytön havaitsemismenetelmä myös alkaa suojaamaan järjestelmää heti käyttöönoton alettua, mikäli oppimisprosessia ei ole [16].

Toisaalta näin toimien voidaan havaita ainoastaan tunnetut hyökkäykset [28]. Kaikkien mahdollisten hyökkäysten ennustaminen ei ole käytännössä mahdollista, ja pelkkä sen yrittäminen vaatisi valtavasti työtä [19]. Lisäksi joka tapauksessa tunnettujen hyökkäysten listaa pitää jatkuvasti päivittää, mikä vaatii yleensä ihmisen asiantuntemusta [6]. Tämä on erityisen vaikeaa jos hyökkäyssäännöt sisältävät useita hyökkäyksen eri vaiheita [16]. Tutkiessaan yhtä toimintaa väärinkäytön havaitsemisjärjestelmät eivät muista edellisiä toimintoja, joten monivaiheisten hyökkäysten havaitseminen ei onnistu, jos yksittäiset toiminnat eivät täytä hyökkäyksen tunnusmerkkejä [30].

Esimerkki säännöstä voi olla esimerkiksi telnet-kirjautuminen käyttäjätunnuksella *root*, joka ei ole tietoturvasäännöissä sallittua [30]. Toisaalta epätavallisen suuri määrä yhteyksiä tietyn ajan sisällä tai liian suuri prosessorikuormitus voivat myös aiheuttaa hälytyksen [28].

On tärkeää huomata, että väärinkäytön havaitsemismenetelmät ovat kehittyneet viime vuosina paljon. Niinpä edellä kuvatut rajoitteet ovat osittain poistuneet. Menetelmät ovat paljon joustavampia kuin ennen, ja myös tekoälyä (esimerkiksi neuroverkkoja) voidaan käyttää apuna havaitsemisessa [5]. Perusperiaate kuitenkin säilyy, eli järjestelmät vaativat ennakkoon tietoa järjestelmästä ja hyökkäyksistä. Vaik-

ka pelkkä informaatio erilaisista hyökkäyksistä riittää periaatteessa niiden tunnistamiseen, mikäli jonkinlaista oppimisalgoritmia käytetään, tarvitaan lisäksi tunnettua normaalia liikennettä [31]. Edelleenkään tämä lähestymistapa ei tarjoa ratkaisua aiemmin tuntemattomien hyökkäysten tunnistamiseen. Väärinkäytön ja poikkeavuuksien havaitsemisten välinen raja on kuitenkin kaventunut. Poikkeavuuksien havaitsemista käsitellään kappaleessa 3.7.

3.6.1 SNORT

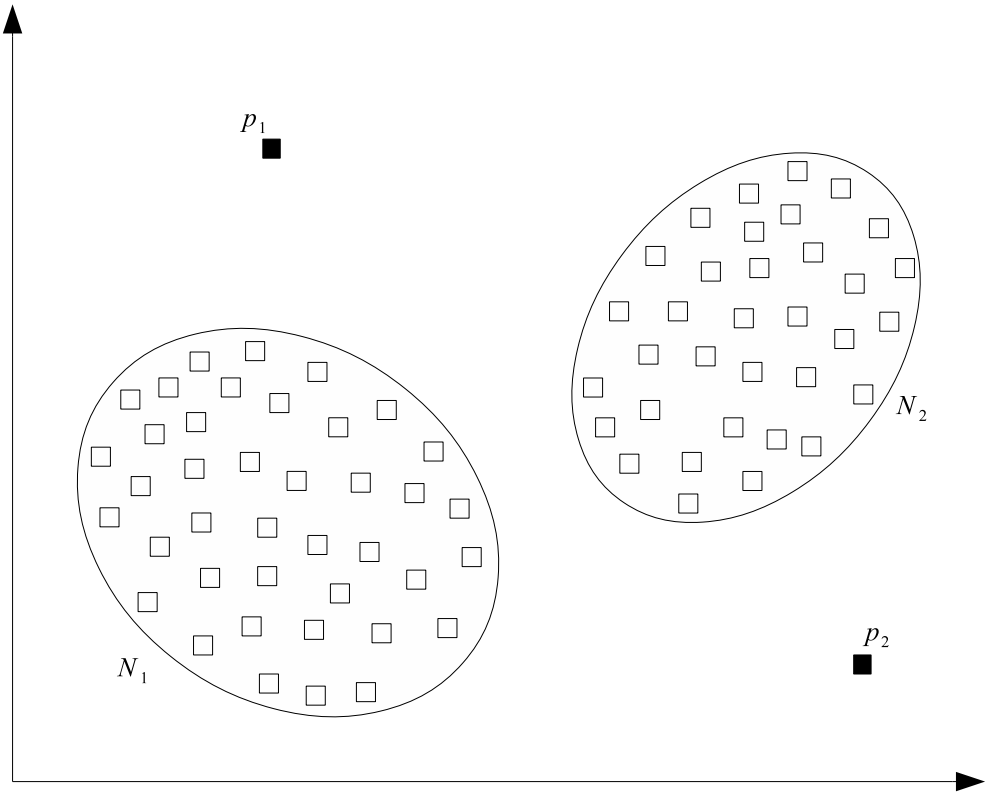
Tunnetuin väärinkäytön havaitsemisjärjestelmä lienee SNORT. Se on käytetyin IDS-järjestelmä, jota on kirjoitushetkellä ladattu lähes neljä miljoonaa kertaa, ja sillä on yli 300 000 rekisteröitynyttä käyttäjää. SNORT perustuu avoimeen lähdekoodiin ja se julkaistiin ensimmäisen kerran vuonna 1998 [32].

SNORT on erittäin pienikokoinen ja kevyt ohjelma, joka on helppo ja nopea ottaa käyttöön. Se ei vaadi erillisten sensoreiden asettamista järjestelmään, mikä olisi kallista. Ohjelma tutkii paketteja sovelluskerroksella, ja poikkeava liikenne voidaan tallentaa lokiin tai siitä voidaan tehdä hälytys. Ohjelmalle voidaan tehdä helposti erilaisia sääntöjä, joiden perusteella mahdolliset hyökkäykset havaitaan [33]. Valmiita sääntöjä on saatavilla runsaasti [34]. SNORT on parhaimmillaan silloin, kun raskaiden ja monimutkaisten kaupallisten järjestelmien käyttäminen ei ole järkevää tai kustannustehokasta [33]. Se toimii myös pelkkänä *snifferinä*, jolloin ohjelmalla voidaan yksinkertaisesti monitoroida kaikkea pakettiliikennettä [34]. Ohjelman nopean päivitysrytmin ansiosta se pysyy ajan tasalla suhteellisen hyvin, joten tunnetut hyökkäykset pystytään havaitsemaan tehokkaasti.

3.7 Poikkeavuuksien havaitseminen

Määritellään ensin poikkeavuus (engl. *anomaly*) tässä yhteydessä. Poikkeavuudeksi voidaan ajatella toiminta joka on jotenkin erilaista normaaliin nähden. On siis yksi tai useampia joukkoja, joihin normaali toiminta voidaan laskea. Poikkeavuudet eivät sijoitu näihin joukkoihin vaan niiden ulkopuolelle. Kuvassa 3.4 on yksinkertainen esimerkkitilanne kaksiulotteisen datan tapauksessa. Klusterit N_1 ja N_2 sisältävät normaaleja datapisteitä. Pisteet p_1 ja p_2 sijoittuvat nyt näiden ulkopuolelle ja ovat siis poikkeavuuksia. Periaatteessa myös poikkeavuudet voivat muodostaa omia klustereitaan, jotka usein ovat kuitenkin pienempiä kuin normaalit klus-

terit. Tietoliikenteessä poikkeavuuksia voivat aiheuttaa esimerkiksi erilaiset hyökkäykset. Poikkeavuuksien havaitsemisessa tärkeää onkin se, että havaitut poikkeavat ovat jollain tapaa kiinnostavia ja että ne antavat jotain oikeasti hyödyllistä tietoa [35].



Kuva 3.4: Normaalit klusterit ja poikkeamia.

Poikkeavuuksien havaitseminen (*anomaly detection*) tarkoittaa sitä, että ensin luodaan jonkinlainen profiili käyttäjän, verkon tai ohjelman normaalista toiminnasta. Tämän jälkeen käyttäjän toimintaa tai verkon liikennettä verrataan luotuun profiiliin [5,16,31]. Poikkeuksien havaitseminen jakautuu siis yleensä kahteen vaiheeseen, joissa ensin luodaan profiilit opetusvaiheessa (*training phase*) ja sitten luotua profiilia käytetään hyväksi verrattaessa uutta dataa normaaliin liikenteeseen testausvaiheessa (*testing phase*) [16]. Kaikki normaalista liikenteestä tai toiminnasta poikkeava tulkitaan poikkeavuudeksi ja se aiheuttaa hälytyksen. Poikkeavuudet tulkitaan mahdollisiksi hyökkäyksiksi [16]. Mallin luomiseen kuluva aika vaihtelee, ja se voi olla jopa viikkoja. Kun profiili on luotu, se pysyy samana ellei sitä luoda uudelleen. Tämän takia profiili on luotava uudelleen säännöllisin väliajoin. Toisaalta on mahdollista käyttää dynaamisesti luotavaa mallia [30].

Poikkeavuuksien havaitsemisjärjestelmien suurin etu on se, että ne voivat havaita aiemmin tuntemattomia sekä ns. nollapäivähyökkäyksiä (*zero day attack*) [5, 31, 36]. Lisäksi on mahdollista havaita järjestelmän sisältä tulevia hyökkäyksiä. Esimerkiksi järjestelmän käyttäjätilin kaapannut hyökkääjä saattaa hyökkäystä yrittäessään toimia normaalista poikkeavalla tavalla, ja tämä toiminta voidaan näin havaita [16]. Hyökkääjän on myös vaikeaa tai mahdotonta tietää, kuinka paljon normaalista poikkeavaa hänen toimintansa saa olla ennen kuin toiminta havaitaan [16]. Niinpä hyökkääjä ei voi koskaan luottaa siihen, että tietyt toimet jäisivät järjestelmältä huomaamatta, vaikke ne poikkeaisivat normaalista toiminnasta vain vähän.

Poikkeavuuksien havaitsemisessa on kuitenkin paljon heikkouksia ja ongelmia. Ensimmäinen niistä on opetusvaiheen tarve [16]. Normaalin profiilin luominen ei ole helppoa, sillä usein siihen tarvitaan täysin normaalia liikennettä [29]. Usein kuitenkin opetusdatan joukossa on myös poikkeavaa liikennettä, joka vääristää normaalia profiilia [30]. Lisäksi profiilin ylläpito voi olla vaikeaa ja vaatia raskasta laskentaa [16]. Poikkeuksien havaitseminen perustuu oletukseen, että hyökkääjän toiminta eroaa selkeästi verkon normaalista toiminnasta [37]. Näin ei kuitenkaan aina ole, ja toisaalta sinänsä normaali tai ei-haitallinen toiminta voi olla järjestelmän mielestä poikkeavaa. Tämän vuoksi suurin ongelma onkin väärät positiiviset tunnistukset (*false positive*), jotka on tunnistettu poikkeuksiksi mutta jotka ovat oikeasti normaaliin liikenteeseen kuuluvaa toimintaa [5, 16]. Tällaista toimintaa voi olla esimerkiksi järjestelmän ylläpidon tekemä toimenpide, joka tehdään erittäin harvoin ja jota ei esiintynyt opetusvaiheessa [30]. Samanaikaisesti tapahtuva normaali sekä poikkeava liikenne tekevät poikkeavuuden havaitsemisen usein erittäin vaikeaksi [31]. Väärien positiivisten suuri määrä voi myös aiheuttaa sen, että niiden joukosta ei löydetä oikeasti haitallista liikennettä, ja tämä aiheuttaa vääriä negatiivisia tunnistuksia (*false negative*) [16]. Lisäksi hälytyksen sattuessa voi olla vaikeaa yhdistää tiettyä hälytystä sen aiheuttaneeseen toimintaan [16]. Tämä on erityisen vaikeaa monimutkaisissa järjestelmissä, joissa useat eri toiminnot voivat yhdessä aiheuttaa hälytyksen [30]. Erittäin vaikea heikkous on se, että hyökkääjä voi opettaa järjestelmän tulkitsemaan hyökkäykseen liittyvään liikenteen normaaliksi [16]. Hyökkääjä voi esimerkiksi aloittaa pienellä määrällä liikennettä, ja lisätä toiminnan määrää vähitellen, jolloin järjestelmä ei havaitse sitä enää poikkeavaksi [30]. Lisäksi se, mikä loppuen lopuksi on poikkeavaa toimintaa, vaihtelee suuresti eri tilanteissa. Joskus normaali toiminta sisältää paljonkin vaihtelua, mutta toisessa tilanteessa jo vähänkin normaalista poikkeava toiminta on heti oikeasti epänormaalia [35].

Poikkeavuuksien havaitsemisjärjestelmä voidaan toteuttaa monella tavalla, jotka eroavat toisistaan yleensä siinä, miten profiilit luodaan ja miten niitä verrataan uuteen liikenteeseen [31]. Useita erilaisia menetelmiä voidaan käyttää, ja niistä on kerrottu tarkemmin luvussa 4.

3.8 Hybridijärjestelmät

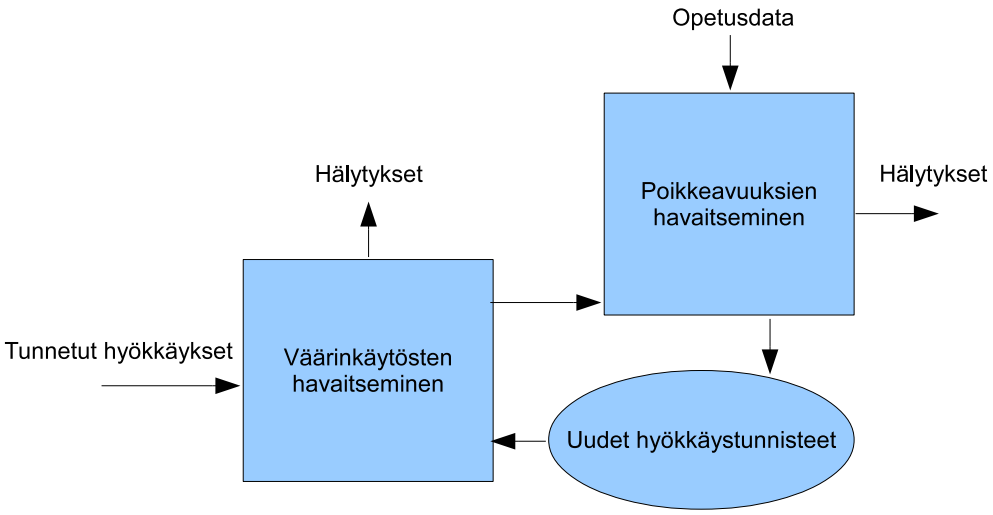
Luvuissa 3.6 ja 3.7 esiteltyjä menetelmiä voidaan myös käyttää yhdessä. Tätä sanotaan HIDS-järjestelmäksi (*hybrid intrusion detection system*) [38,39]. HIDS-järjestelmässä käytetään sekä olemassa olevaa tietoa hyökkäyksistä, että yritetään havaita poikkeavaa liikennettä normaalin joukosta. On mahdollista käyttää ns. hybridimallia, joka luodaan käyttäen hyökkäystietoa sekä normaalia liikennettä [16]. Toisaalta voidaan käyttää kahta erillistä järjestelmän osaa, joista toinen toimii kuten väärinkäytösten havaitsemisjärjestelmä ja toinen kuten poikkeuksien havaitsemisjärjestelmä. Tämän jälkeen jonkinlainen päätöksentekojärjestelmä yhdistää tulokset ja päättää, milloin tehdään hälytys [38]. Esimerkiksi voidaan tehdä hälytys, jos edes toinen järjestelmistä havaitsee poikkeuksen tai hyökkäyksen.

Hybridijärjestelmä voidaan toteuttaa myös siten, että eri järjestelmän osat tukevat toisiaan. Esimerkiksi Hwangin ym. [39] toteuttamassa HIDS-järjestelmässä ensin ajetaan perinteinen väärinkäytösten havaitseminen käyttäen SNORT-ohjelmaa (ks. 3.6.1), ja havaituista hyökkäyksistä tehdään hälytykset normaalisti. Tuntematon liikenne ohjataan sen jälkeen poikkeavuuksien havaitsemisjärjestelmälle, joka on opetusdataa hyödyntäen opetettu tunnistamaan järjestelmän normaali liikenne. Poikkeuksista tehdään hälytykset. Uusista hyökkäyksistä tehdään uudet hyökkäystunnisteet, jotka viedään SNORT-ohjelman tunnisteisiin. Näin uudella hyökkäystiedolla voidaan täydentää olemassa olevia hyökkäystunnisteita ja näin parantaa järjestelmän tarkkuutta. Parhaimmillaan järjestelmä pystyi tämän ansiosta havaitsemaan 60 prosenttia enemmän hyökkäyksiä. Yksinkertaistettu malli on kuvassa 3.5.

Hybridijärjestelmä yhdistää kahden eri lähestymistavan hyvät puolet, mutta toisaalta monimutkaistaa järjestelmää entisestään ja lisää prosessoinnin tarvetta.

3.9 Intrusion prevention system

Tunkeilijan estojärjestelmä eli IPS (*intrusion prevention system*) hoitaa pääosin samoja tehtäviä kuin tunkeilijan havaitsemisjärjestelmä. Havaitsemisen lisäksi se kuitenkin



Kuva 3.5: Hwangin ym. HIDS-järjestelmä.

myös pyrkii pysäyttämään hyökkäyksen tai ainakin rajoittamaan tapahtuvaa vahinkoa. Muilta osin IPS vastaa IDS-järjestelmää, ja termejä käytetään osittain sekaisin.

Hyökkäyksen pysäyttäminen voidaan toteuttaa monella eri tavalla [30]:

- Hyökkäykseen käytetyn yhteyden katkaiseminen tai istunnon lopettaminen
- Yhteyden estäminen hyökkääjän osoitteesta tai käyttäjätililtä hyökkäyksen kohteena olevaan resurssiin
- Yhteyden estäminen kokonaan hyökkäyksen kohteena olevaan resurssiin

Lisäksi tunkeilijan estojärjestelmä voi tehdä muutoksia verkkopalveluun hyökkäyksen vaikeuttamiseksi. Myös verkossa liikkuvan sisällön muokkaaminen on mahdollista, esimerkiksi poistamalla saastunut liitetiedosto sähköpostista [30].

IPS-järjestelmä pystyy toimimaan ainakin neljällä eri kerroksella. Linkkikerroksella voidaan sulkea tiettyjä portteja. Verkkokerroksella on mahdollista lisätä reitittimeen tai palomuuriin sääntöjä joilla estetään liikenne tietyistä IP-osoitteista. Siirtokerroksella voidaan generoida TCP RST -paketteja (*TCP reset*) joilla haitallinen TCP-yhteys voidaan katkaista. Sovelluskerroksella dataa voidaan muokata niin, että siitä ei enää koidu haittaa järjestelmälle [40].

IPS-järjestelmä voi aiheuttaa myös ongelmia. Esimerkiksi voidaan toteuttaa valehyökkäys, jossa hyökkäyksen verkkoliikenteen pakettien lähteeksi väärennetään jokin toinen taho, vaikkapa yrityksen kauppakumppani. Hyökkäyksen kohteena oleva järjestelmä estää yhteydet kyseiseen tahoon, vaikka hyökkäys tulisi tosiasiassa

aivan muualta [40]. Tätä voidaan ajatella eräänlaisena palvelunestohyökkäyksenä (ks. 2.3).

Esimerkkejä IPS-toteutuksista ovat SNORT-ohjelman (ks. 3.6.1) lisäosat Snortsam ja fwsnort. Snortsam mahdollistaa tiettyjen IP-osoitteiden estämisen palomuurista automaattisesti [41]. Fwsnort puolestaan muokkaa iptables-palomuurisääntöjä SNORT-ohjelman hyökkäyssääntöjen pohjalta, eli se toimii yhdessä Linuxin palomuurin kanssa [42]. Käytännön toteutuksia on muitakin, mutta edellä mainittujen ongelmien johdosta ne eivät ole niin yleisiä kuin normaalit IDS-järjestelmät.

4 Hyökkäysten ja poikkeusten havaitsemisessa käytettyjä menetelmiä

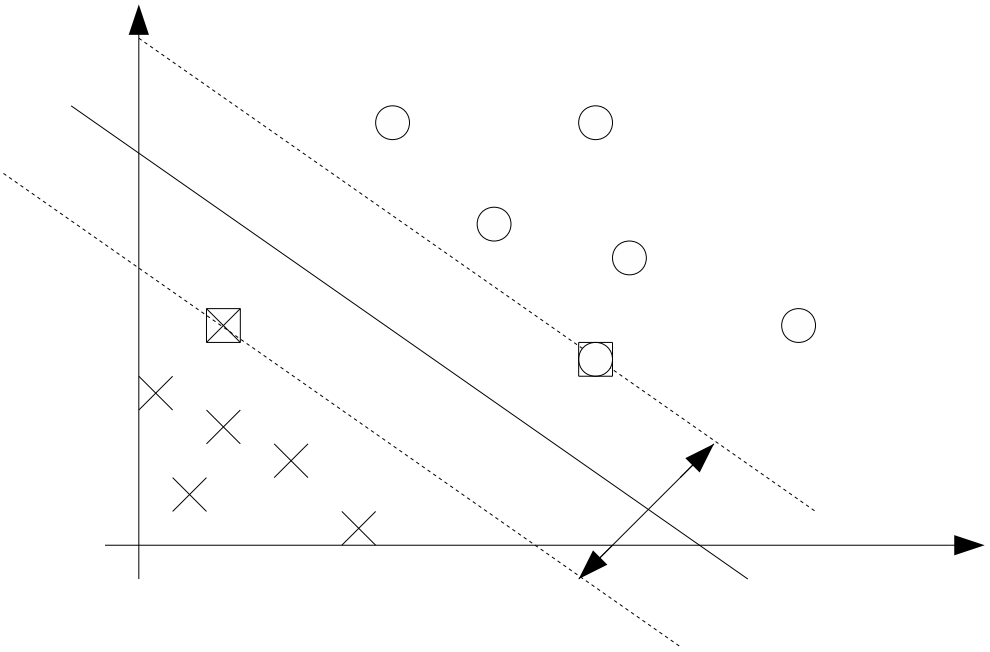
IDS-järjestelmissä varsinainen analyysivaihe voidaan suorittaa käyttäen monia eri menetelmiä. Hyökkäysten tai poikkeavuuksien havaitseminen voi perustua yksinkertaiseen sääntöpohjaiseen suodatukseen, mutta viime aikoina on myös tutkittu paljon erilaisia koneoppimiseen pohjautuvia algoritmeja. Jotkut menetelmät vaativat jonkinlaista ennakkotietoa, jotta ne saataisiin opetettua tunnistamaan haluttua liikennettä, mutta usein riittää normaalin liikenteen opettaminen. Itse menetelmien toiminta ja suorituskyky vaihtelevat suuresti. Useimmat menetelmistä ovat sellaisia, että niitä voidaan soveltaa paljon yleisemminkin useisiin eri sovelluskohteisiin tietoturvan ollessa vain yksi sovellusalue lisää. Tietoliikenteen analysoiminen tarkoittaa usein vain matriisimuodossa olevan datan tutkimista, joten esimerkiksi kuvantunnistus on periaatteeltaan hyvin samanlaista. Lisäksi erityisen mielenkiintoista on se, että monet menetelmistä pohjautuvat biologiaan tai muihin todellisiin ilmiöihin, esimerkiksi neuroverkot aivojen neuronien toimintaan ja keinoimmuunijärjestelmät ihmisen immuunijärjestelmään. Muuttuvassa tietoliikennemaailmassa äly ja oppimiskyky on tärkeää uusien uhkien tunnistamiseksi. Tietoliikenteen analysoinnissa on käytetty vuosien saatossa lukuisia eri menetelmiä, joista muutamia mielenkiintoisimpia on kuvattu tässä.

4.1 Tukivektorikoneet

Tukivektorikone (engl. *support vector machine*) on oppiva kahden luokan luokitinmalli. Sen toiminta perustuu siihen, että kahden luokan väliin luodaan päätöstaso, joka jakaa luokat kahteen osaan moniulotteisessa piirreavaruudessa mahdollisimman suurella marginaalilla [43]. Marginaalitasoille jääviä eli päätöstasoa lähimpänä olevia pisteitä sanotaan tukivektoreiksi. Piirreavaruudessa on yleensä suuri määrä ulottuvuuksia, ehkä jopa äärettömän monta [44].

Kuvassa 4.1 on havainnollistettu tätä tilannetta. Kuvan yksinkertaistamisen takia käytössä on vain kaksi ulottuvuutta. Kuvassa näkyvät kaksi eri joukkoa, joihin kuuluu useita pisteitä. Piirreavaruudessa on lisäksi taso, joka jakaa pisteet kahteen

eri luokkaan selvästi. Tämän päätöstason molemmilla puolilla ovat katkoviivalla kuvatut marginaalitasot. Näille tasoille osuvat pisteet ovat tukivektoreita, ja ne on kuvattu laatikon sisään piirrettyinä. Nämä pisteet ovat lähimpänä päätöstasoa, ja ne ovat siis molemmat saman etäisyyden eri marginaalin päässä päätöstasosta. Mitä suurempi marginaali, sitä paremmin tukivektorikone erottelee luokat toisistaan.



Kuva 4.1: Tukivektorit, marginaalitasot ja päätöstaso.

Toisin kuin monet luokittelumenetelmät, analysoitavan datan ulottuvuuksien määrä ei vaikuta vaan tukivektorikoneen toimintaan, vaan siihen vaikuttavat ainoastaan datan geometriset ominaisuudet [6].

Kahden luokan luokittelu ei kuitenkaan sellaisenaan välttämättä toimi analysoitaessa verkkodataa poikkeavuuksien löytämiseksi. Poikkeuksista ei ole etukäteen tietoa, joten tunnettua on vain yksi luokka eli normaali liikenne. Niinpä menetelmä on pystyttävä opettamaan ilman luokkien nimeämistä. Tähän tarkoitukseen soveltuu yksiluokkainen tukivektorikone [45]. Periaatteena on sijoittaa pisteet moniulotteiseen piirreavaruuteen jonkin kernelifunktion avulla. Erilaisia kernelifunktioita on mahdollista käyttää, esimerkiksi polynomiaalinen tai radiaalinen [44]. Yksiluokkaisessa tukivektorikoneessa on päätösfunktio, josta saadaan tuloksena arvo 1, mikäli piste on joukossa johon useimmat pisteet kuuluvat. Pääjoukkoon kuulumattomat pisteet antavat arvon -1 [37]. Näitä pisteitä voidaan tässä tutkimuksessa ajatella tietoliikenteen poikkeavuuksiksi. Riittää siis opettaa menetelmälle järjestelmän nor-

maali toiminta.

4.1.1 Tukivektorikoneista tehty tutkimus

Tukivektorikoneet ovat olleet suuren mielenkiinnon kohteena jo vuosia. Niitä on tutkittu paljon hyökkäyksien ja poikkeavuuksien löytämisessä. Erityisesti yksiluokkaiset tukivektorikoneet ovat mielenkiintoisia tämän tutkimuksen kannalta.

Mukkamala, Janoski ja Sung tutkivat tukivektorikoneen toimintaa käyttäen kuuluisaa DARPA 1998 -dataa. Data on kerätty siihen tarkoitukseen tehdystä lähiverkosta, joka simuloi tyypillistä Yhdysvaltojen ilmavoimien lähiverkkoa. Verkkoon tehtiin tarkoituksella useita eri tyyppisiä hyökkäyksiä. Data sisältää tiedustelu-, U2R-, R2L- ja palvelunestohyökkäyksiä. Ne käyttävät yhteensä 32 eri haavoittuvuutta. Tukivektorikoneet valittiin tutkimukseen sen takia, että ne ovat nopeita ja skaalautuvat hyvin suurille datamäärille. Järjestelmää opetettiin 7312 datapisteellä, joka sisälsi sekä normaalia liikennettä että hyökkäyksiä. Sen jälkeen menetelmää testattiin 6980 datapisteellä. Testissä saavutettiin 99.5 prosentin tarkkuus, ja varsinainen testausvaihe vei ainoastaan 1.63 sekuntia [46]. Tukivektorikone osoittautui neuroverkkoa nopeammaksi varsinkin opetusvaiheessa. Toisaalta tukivektorikone pystyy tekemään vain kahden luokan luokittelua. DARPA-datasetti on saatavilla vapaasti verkosta [47].

Mukkamala ja Sung toteuttivat samankaltaisen tutkimuksen samalla datasetillä. Koska tukivektorikoneet kykenevät ainoastaan binääriseen luokitteluun, he käyttivät viittä tukivektorikonetta viiden luokan luokitteluun. Luokat olivat samat kuin edellisessä tutkimuksessa, eli normaali liikenne ja neljä eri tyyppistä hyökkäystä. Tulokset olivat samankaltaisia joka luokassa. Tarkkuus oli alin palvelunestohyökkäyksiä tutkittaessa (99.25%) ja korkein U2R-hyökkäysten tunnistuksessa (99.87%) [6]. Edellä mainituissa kahdessa tutkimuksessa myös hyökkäysdataa käytettiin tukivektorikoneen opettamiseen. Koska tämän tutkimuksen palvelinlokissa ei ole mitään tietoa mahdollisista hyökkäyksistä, ei tämä lähtökohta sovellu tähän tutkimukseen.

Myös Lazarevic ym. [29] tutkivat hyökkäysten havaitsemista käyttäen DARPA 1998 -datasettiä. He käyttivät ryhmittelyyn kykenevää tukivektorikonetta, jonka ei tarvitse opetusvaiheessa tietää mihin luokkaan opetusdatan pisteet kuuluvat. Tarkoituksena on tulkita pisteet siten, että suuri osa pisteitä on lähellä toisiaan pääjoukossa ja ne tulkitaan yhdeksi luokaksi. Kauempana olevat pisteet ovat poikkeuksia. Tulokset riippuivat siitä, mihin poikkeukseksi tulkitsemisen raja asetettiin. Kun

vääriä hälytyksiä sallittiin 2 %, löydettiin hyökkäyksistä onnistuneesti vain 44%. Jos vääriä hälytyksiä sallittiin enemmän, useimmat hyökkäykset löytyivät, mutta järjestelmän toiminta ei ollut enää tarkoituksenmukaista. Käytössä oli myös oikeaa verkkodataa, mutta siihen tukivektorikoneita ei sovellettu.

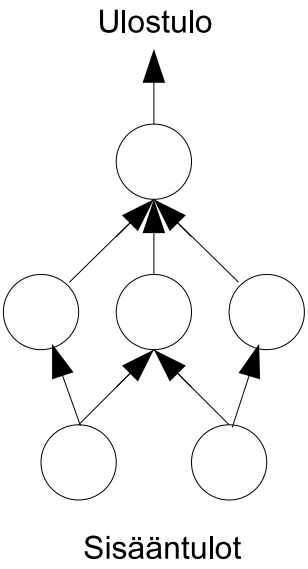
Tran, Duan ja Li kehittivät yksiluokkaiseen tukivektorikoneeseen perustuvan järjestelmän nimeltä OTAD [37]. Se voi toimia kolmessa eri tilassa. Keräystilassa järjestelmä kerää verkosta tietoa normaalista datasta opetusta varten. Data keräämiseen käytetään Tcpstat-ohjelmaa [48]. Opetustilassa tukivektorikone opetetaan käyttäen kerättyä normaalia liikennettä. Havaitsemistilassa järjestelmä yrittää havaita mahdollisia hyökkäyksiä opetetun tiedon pohjalta. Tutkimuksessa käytettiin DARPA 1999 -dataa, jossa ensimmäisen viikon liikenne oli normaalia, kun taas toisen viikon liikenne sisältää hyökkäyksiä. Kun väärien hälytysten määrä pidettiin tarpeeksi pienenä, järjestelmä kykeni parhaimmillaan tunnistamaan 71% hyökkäyksistä. Osaa hyökkäyksistä oli liian vaikeaa havaita [37].

4.2 Neuroverkot

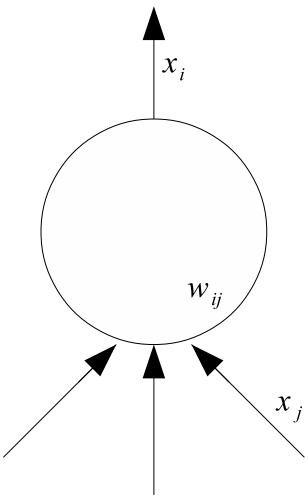
Neuroverkkoja (engl. *neural networks*) on tutkittu kymmeniä vuosia. Ajatus niiden kehittämiseksi lähti siitä, että tietokoneet eivät soveltuneet tietynlaisiin tutkimusongelmiin, esimerkiksi kuvantunnistukseen. Niinpä neuroverkkojen esikuvina toimivat ihmisen aivojen ja aivosolujen toiminta, vaikka tarkoituksena ei ole varsinaisesti kopioida niiden toimintaa. Ideana on se, että neuroverkko toimii opettamisen jälkeen automaattisesti, eikä uutta ohjelmointia tarvita. Mikäli mahdollista, se oppii opetusdatan perusteella arvioimaan jotakin funktiota. Mikäli neuroverkkoa tarvitaan jossain muussa yhteydessä, periaatteessa se voidaan vain opettaa uudelleen [49].

Kuvassa 4.2 on erittäin yksinkertaisen neuroverkon malli. Se koostuu keinotekoisista neuroneista jotka ovat useassa kerroksessa. Jokaiselle neuronille on monta sisääntuloa mutta vain yksi ulostulo, joka tosin voi olla yhteydessä useampaan toiseen neuroniin seuraavalla kerroksella. Neuroneiden välillä olevia yhteyksiä kutsutaan synapseiksi. Kuvassa 4.3 on yksittäinen neuroni, joita voidaan kutsua myös nimellä prosessointiyksikkö (engl. *processing unit*, lyh. *PE*) [49].

Jos on neuroni PE_i ja sen yksittäinen sisääntulo on x_j, tämän sisääntulon painokerroin on w_{ij}. Yhteydet ovat siis painotettuja. Kaikkien sisääntulojen summa on siis



Kuva 4.2: Yksinkertainen neuroverkko.



Kuva 4.3: Yksittäinen neuroni.

$$net_i = \sum_j x_j w_{ij} \tag{4.1}$$

Sisääntulojen summasta voidaan laskea aktivaatioarvo. Yleensä tämä on täysin sama luku, mutta voi joskus myös riippua edellisestä arvosta, jolloin ajassa t aktivaatioarvo on

$$a_i(t) = F_i(a_i(t-1), net_i(t)) \tag{4.2}$$

Tämän jälkeen voidaan laskea varsinainen ulostulo käyttäen ulostulofunktiota (joskus myös aktivaatiofunktio):

$$x_i = f_i(a_i) \tag{4.3}$$

Koska yleensä pätee $a_i = net_i$, ulostulofunktiosta tulee

$$x_i = f_i(net_i) \tag{4.4}$$

Lyhyesti sanottuna neuroverkossa on useita sisääntuloja ja ulostuloja jotka ovat yhteydessä keinotekoisiin neuroneihin. Neuronissa sisääntuloista muodostetaan ulostulo aktivaatiofunktiolla. Viimeisellä kerroksella olevat ulostulot (joita voi olla yksi tai useampia) muodostavat koko verkon ulostulot [49].

4.2.1 Oppiminen ja backpropagation-algoritmi

Neuroverkkojen oppiminen tarkoittaa käytännössä painokerrointen muuttamista opetusdatan mukaan. Yleinen opetusalgoritmi on ns. *backpropagation*-algoritmi [50]. Olkoon E virheen määrä, eli halutun (t_k) ja todellisen (o_k) ulostulon erotus

$$E = \sum_k (t_k - o_k)^2 \tag{4.5}$$

kun on k kappaletta ulostuloja. Oppiminen tarkoittaa nyt sitä, että muutetaan synapsin painokerrointa w_{ij} siten, että virheestä tulee mahdollisimman pieni. Tähän käytetään laskevan gradientin menetelmää:

$$\Delta w_{ij} = -\frac{\delta E}{\delta w_{ij}} = 2\sum (t_k - o_k)^2 \frac{\delta o_k}{\delta w_{ij}} \tag{4.6}$$

Virhe lasketaan jokaisella kerroksella ja se vaikuttaa edellisen kerroksen painokertoimien arvoihin. Jokaisen neuronin painoarvojen korjaaminen on suhteessa siihen, kuinka paljon virheestä kyseinen neuroni aiheuttaa [50]. Kun oppimisprosessia toistetaan, jokainen neuroni alkaa erottamaan opetusdatasta eri piirteitä [49]. Neuroneita voidaan tämän takia ajatella itsenäisiä prosessointiyksikköinä.

4.2.2 Itseorganisoituva kartta

Itseorganisoituva kartta (engl. *self-organizing map*) on Teuvo Kohosen kehittämä neuroverkkomalli [51]. Se on erittäin yleisesti käytetty ja yksi kuuluisimpia tutkimustuloksia neuroverkkojen saralla. Itseorganisoituviin karttoihin liittyen on kirjoitettu jo yli 7700 tieteellistä artikkelia ympäri maailman [52].

Itseorganisoituva kartta perustuu ohjaamattomaan oppimiseen (engl. *unsupervised learning*). Oppiminen tapahtuu siis vain prosessoitavan datan avulla, eikä oppimisprosessissa tarvita tietoa odotetusta ulostulosta. Datan pisteitä ei tarvitse etukäteen jakaa tunnettuihin luokkiin. Menetelmän avulla moniulotteinen data voidaan esittää kaksiulotteisessa avaruudessa siten, että alkuperäisten datavektorien ominaisuudet säilyvät [49]. Tämän takia itseorganisoituvat kartat toimivat hyvin kun halutaan visualisoida moniulotteista dataa. Ohjaamattoman oppimisen takia se on lupaava menetelmä myös poikkeavuuksien havainnointiin.

Oppimisprosessissa tarvitaan menetelmälle syötettäviä datavektoreita. Lisäksi on keinotekoisia neuroneita eli painovektoreita. Määritellään N_c joukoksi painovektoreita, jotka ovat lähellä tiettyä datavektoria. Ne ovat siis datavektorin "naapurijoukko". Aluksi tämän joukon säde voi olla suuri, mutta sitä pienennetään oppimisen edetessä. Joukko N_c keskittyy sen painovektorin ympärille, joka on lähimpänä datavektoria. Tämä lähin painovektori saadaan kaavalla

$$||x - m_c|| = min_i\{||x - m_i||\} \tag{4.7}$$

missä x on syötedatan vektori. Lähin painovektori m_c saadaan siis etsimällä sellainen painovektori m_i, jonka euklidinen etäisyys vektoriin x on mahdollisimman pieni [51].

Painovektorien päivitys tapahtuu kaavalla

$$m_i(t+1) = \begin{cases} m_i(t) + \alpha(t)[x(t) - m_i(t)] & \text{jos } i \in N_c(t), \\ m_i(t) & \text{jos } i \notin N_c(t), \end{cases} \tag{4.8}$$

missä $\alpha(t)$ on mukautumistermi ja $0 < \alpha(t) < 1$. Tästä termistä riippuu, kuinka paljon painovektorin arvoa korjataan ja sen arvon tulisi pienentyä oppimisen edetessä. Vain naapurijoukkoon N_c kuuluvia painovektoreita päivitetään [51]. Oppimisprosessin aikana neuronit lähenevät vähitellen datavektoreita [49]. Naapurijoukon pienentyessä oppiminen muuttuu laajalta alueelta pienelle alueelle, johon oppiminen lopuksi keskittyy. Prosessi suoritetaan jokaiselle datavektorille.

4.2.3 Neuroverkoista tehty tutkimus

Erityisesti itseorganisoituvia karttoja on tutkittu erittäin paljon poikkeavan tietoliikenteen havaitsemisessa. Sen vahvuus perustuu ohjaamattomaan oppimiseen. Poikkeavaa tietoliikennettä ei tunneta etukäteen, joten ohjaamaton oppiminen sopii sen havaitsemiseen hyvin.

Myös perinteisiin neuroverkkoihin pohjautuvaa tutkimusta on jonkun verran. Esimerkiksi Ryan ym. [19] tutkivat tietoliikennettä omassa kymmenen käyttäjän järjestelmässään. He keräsivät tietoa käyttäjien suorittamista komennoista NetBSD-käyttöjärjestelmässä. Tarkasteltavaksi valittiin 100 käytetyintä komentoa. Jokaiselle komennolle laskettiin välillä 0–1 skaalattu arvo sen mukaan, kuinka usein komentoa käytettiin. Opetusdataa kerättiin 12 päivää. Tutkimuksessa käytettiin neuroverkkoa, jossa oli 100 sisääntuloa (komentojen mukaan), 30 neuronia keskimmäisessä piilotetussa kerroksessa ja 10 ulostuloa (käyttäjien määrä). Verkon tehtävä oli tunnistaa käyttäjät oikein. Lisäksi generoitiin satunnaisia käyttäjävektoreita, jotka verkon piti luokitella oikein poikkeavuuksiksi. 96% poikkeavuuksista löydettiin ja normaalien käyttäjien liikenteestä 7% luokiteltiin poikkeavaksi. Järjestelmä toimi melko hyvin, mutta sen toimintaperiaate ja varsinkin ohjatun oppimisen periaate ei toimi kovin hyvin poikkeavuuksien etsimisessä. Lisäksi ongelmia saattaa tuottaa järjestelmän skaalautuvuus.

Mukkamala ja Sung [6] käyttivät myös neuroverkkoa osana tutkimustaan. Testidatana oli DARPA 1998 -data. Opetus- ja testidata olivat samat kuin tukivektorikoneen testaukseen käytetyt datajoukot. Neuroverkossa oli kaksi piilokerrosta joissa oli 20 ja 30 neuronia. Data luokiteltiin viiteen eri luokkaan (normaali, tiedustelu, DoS, U2R ja R2L). Aktivaatiofunktio pidettiin samana, mutta opetuksessa käytettävää funktiota vaihdettiin parhaan mahdollisen löytämiseksi. Parhaan tulokset antoi Rprob-algoritmi (*resilient backpropagation*). Sillä voitiin tunnistaa DoS- ja R2L-hyökkäykset yli 99% tarkkuudella. Toisaalta U2R-hyökkäysten tunnistamistarkkuus oli vain 34.3%. Keskimääräinen tunnistamistarkkuus oli 97.04 % ja väärien positii-

visten määrä 2.76%. Mukkamala ym. [46] olivat jo aiemmin päässeet hyviin tuloksiin neuroverkoilla, mutta kyseisessä tutkimuksessa tukivektorikone tunnisti hyökkäykset hieman paremmin.

Höglund, Hätönen ja Sorvari [53] toteuttivat itseorganisoituvaa karttaa hyödyntävän järjestelmä, joka tutki satojan käyttäjien toimintaa UNIX-käyttöjärjestelmässä. Tarkkoja lukuja tuloksista ei ole koska tietoa poikkeavuuksien määrästä ei oikeassa datassa ole, mutta järjestelmän huomattiin mukautuvan muuttuvaan dataan hyvin. Järjestelmä havaitsi poikkeavuuksia myös silloin, kun kerätystä datasta yksittäiset muuttujat eivät poikenneet normaalista liiaksi, mutta ne yhdessä liittyivät poikkeavaan käyttäytymiseen.

Ramadas ym. [54] tekivät SOM-menetelmää käyttävän poikkeuksien havainnointimoduulin nimeltään ANDSOM. Järjestelmä rakensi ensin kartan opetusdatan pohjalta. Sitten sille syötettiin verkkodatan perusteella yksittäisiä vektoreita. Näille laskettiin lähin naapuri, ja mikäli etäisyys siihen oli suurempi kuin ennalta määritetty raja-arvo, yhteys katsottiin poikkeavuudeksi. Esimerkkinä tehtiin hyökkäys DNS-protokollaa käyttäen. Hyökkäyksestä luodun vektorin etäisyys lähimpään naapuriin oli yli 22, kun raja oli asetettu arvoon 2. Samoin HTTP-protokollaan käyttäen testimielessä luodut hyökkäykset saatiin tunnistettua.

Gonzáles ym. [55] tutkivat poikkeavuuksien havainnointia samantyyppisesti. Opetusvaiheessa data klusteroidaan itseorganisoituvalla kartalla, ja uusia pisteitä verrataan siihen. Liian kaukana normaalista klusterista olevat pisteet tulkitaan poikkeavuuksiksi. Sekä DARPA 1998- että DARPA 1999 -datalla tulokset olivat hyviä. Yli 98% hyökkäyksistä tunnistettiin erittäin pienellä väärien positiivisten määrällä (1-2%).

4.3 Diffuusiokartat

Usein kerätty data on niin moniulotteista, että on jotenkin pystyttävä vähentämään näitä ulottuvuuksia. Diffuusiokarttojen avulla data voidaan esittää yksinkertaisemmassa diffuusioavaruudessa, jolloin myös klusterointi ja poikkeavuuksien havaitseminen on helpompaa [56].

Kerätyn datan piirteet kuvaavat järjestelmän käyttäytymistä. Olkoon tämä datamatriisi $X = \{x_1, x_2, ..., x_N\}, x_i \in \mathbb{R}^n$. N on datarivien lukumäärä ja n alkuperäisen datan ulottuvuuksien eli piirteiden määrä. Saadaan siis matriisi, jonka koko on $N \times n$. Jokaisella rivillä on yksi datan alkio ja sarakkeet kuvaavat jokaisen alkion

piirteitä.

Ensimmäisessä ja eniten prosessointiaikaa vievässä vaiheessa luodaan samankaltaisuusmatriisi W, joka kuvaa pisteiden etäisyyksiä toisistaan. Tässä voidaan käyttää euklidista etäisyyttä kuten yhtälössä 4.9 [56,57].

$$W_{ij} = \exp\left(-\frac{||x_i - x_j||^2}{\epsilon}\right) \tag{4.9}$$

Tässä vaiheessa on tärkeää oikea parametrin ϵ valinta. Se määrittelee, kuinka kaukana olevat pisteet lasketaan pisteen naapurustoon mukaan. Parametri vaikuttaa suuresti analyysin tuloksiin (ks. 7.3).

Samankaltaisuusmatriisin rivit normalisoidaan käyttämällä diagonaalimatriisia D, joka sisältää matriisin W rivien summat.

$$D_{ii} = \sum_{j=1}^{N} W_{ij} \tag{4.10}$$

Normalisaatio P kuvaa todennäköisyyttä siirtyä tilasta toiseen. Nyt jokaisen rivin summaksi tulee 1.

$$P = D^{-1}W \tag{4.11}$$

Seuraavaksi täytyy laskea matriisin P ominaisarvot, jotka ovat samat kuin yhtälössä 4.12 olevan matriisin $\tilde{P}$ ominaisarvot.

$$\tilde{P} = D^{\frac{1}{2}}PD^{-\frac{1}{2}} \tag{4.12}$$

Näistä saadaan nyt laskettua symmetrinen todennäköisyysmatriisi kuten yhtälössä 4.13.

$$\tilde{P} = D^{\frac{1}{2}}PD^{-\frac{1}{2}} = D^{\frac{1}{2}}D^{-1}WD^{-\frac{1}{2}} = D^{-\frac{1}{2}}WD^{-\frac{1}{2}} \tag{4.13}$$

Tätä kutsutaan normalisoidun graafin Laplace-operaatioksi, ja luotu matriisi säilyttää ominaisarvonsa [57].

Tähän symmetriseen matriisiin käytetään sitten singulaarihajotelmaa (engl. *singular value decomposition*, lyh. *SVD*). Koska $\tilde{P}$ on normaalimatriisi, saadaan $\tilde{P} = U\Lambda U^*$. Matriisin Λ diagonaalilla esiintyvät ominaisarvot $\mathrm{diag}([\lambda_1, \lambda_2, \ldots, \lambda_N])$ vastaavat matriisin $\tilde{P}$ symmetrisyyden takia. Matriisi $U = [u_1, u_2, \ldots, u_N]$ sisältää N

kappaletta matriisin $\tilde{P}$ ominaisvektoreita u_k. Koska lisäksi $\tilde{P}$ on matriisin P konjugaatti, niillä on samat ominaisarvot. Nyt voidaan laskea matriisi V kuten yhtälössä 4.14.

$$V = D^{-\frac{1}{2}}U \tag{4.14}$$

Nyt matriisin sarakkeet $V = [v_1, v_2, \ldots, v_N]$ ovat matriisin P ominaisarvot [57].

Käyttämällä ominaisarvoja matriisissa Λ ja ominaisvektoreita matriisissa V saadaan pisteiden koordinaatit uudessa pieniulotteisessa avaruudessa (yhtälö 4.15). Periaatteessa ominaisarvot pitäisi korottaa potenssiin t, joka kuvaa sitä, kuinka monta aikayksikköä otetaan huomioon siirryttäessä pisteestä toiseen. Nyt on valittu kuitenkin $t = 1$ [56].

$$\Psi = V\Lambda \tag{4.15}$$

Kun parametri ϵ on valittu oikein, lähestytään nollaa nopeasti. Ensimmäinen ominaisvektori v_1 on vakio ja se jätetään pois. Jos käytetään sitä seuraavat d komponenttia, saadaan yhtälö 4.16.

$$\Psi_d : x_i \rightarrow [\lambda_2 v_2(x_i), \lambda_3 v_3(x_i), \ldots, \lambda_{d+1} v_{d+1}(x_i)] \tag{4.16}$$

Tämä diffuusiokartta kuvaa tunnetun pisteen x_i uudessa avaruudessa. Alkuperäisen avaruuden ulottuvuuksia oli n kappaletta, kun nyt niitä on vähemmän eli d kappaletta. Yhtälössä $v_k(x_i)$ kuvaa k. ominaisvektorin i. alkiota.

Nyt datan ulottuvuuksien määrää on saatu vähennettyä ja klusterointi on helpompaa. Poikkeavuuksien löytämisessä voidaan käyttää erilaisia menetelmiä, esimerkiksi spektraaliklusterointia. Esimerkiksi voidaan käyttää 2. ominaisvektoria vastaavaa ulottuvuutta jakamaan aineisto kahteen osaan (suurempi kuin 0 ja pienempi kuin 0). Näin saadaan kaksi klusteria, joista toinen sisältää enemmän alkioita ja valitaan näin normaaliksi joukoksi. Toisen klusterin datapisteet tulkitaan poikkeavuuksiksi.

4.4 N-grammianalyysi

N-grammeja on käytetty paljon poikkeavan tietoliikenteen etsinnässä erityisesti silloin, kun on analysoitava tekstiä sisältävää dataa. Tässä tutkimuksessa niitä käytetään juuri HTTP-pyynnön parametrien analysointiin. Alun perin n-grammianalyysin

käytön esitteli Damashek vuonna 1995 [58]. Ideana oli verrata tekstidokumenttien samankaltaisuutta käyttämättä itse sanoja vaan ainoastaan n-grammeja. N-grammien esiintymistiheyksistä voidaan luoda matriisi ja tätä edelleen analysoida matemaattisesti. N-grammeja voidaan käyttää myös esimerkiksi puheentunnistuksessa, tilastollisessa konekääntämisessä ja oikeinkirjoituksen tarkistuksessa. Niiden ansiosta ei tarvitse käyttää staattisia sanalistoja [59]. N-grammien käyttö poikkeavuuksien havaitsemisessa perustuu oletukseen, että aidosti poikkeava liikenne sisältää joitakin n-grammeja, joita ei esiinny muussa datassa [36]. Normaalissa datassa esiintymättömien n-grammien lisäksi voidaan tutkia erittäin harvoin esiintyviä n-grammeja, vaikka ne sisältyisivätkin normaaliin dataan [60]. N-grammit voidaan analysoida esimerkiksi HTTP-pyynnöstä tai itse hyötykuormasta.

N-grammit muodostetaan n:n pituisen liukuvan ikkunan avulla, jota liikutetaan sanan yli. Näin saadaan kaikki n:n pituiset osamerkkijonot eli n-grammit. Tässä tapauksessa käytetään 2-grammeja, mutta n voi olla periaatteessa mikä tahansa kokonaisluku. Esimerkiksi sanan `kopiokone` erilaisten 2-grammien joukko on $\{ko, op, pi, io, ok, on, ne\}$. 2-grammi `ko` esiintyy kaksi kertaa. Tästä voidaan muodostaa matriisi eri n-grammien esiintymistiheyksistä. ASCII-merkkien tapauksessa erilaisia merkkejä on 256 kappaletta, joten erilaisia n-grammeja voi olla 256^n erilaista, 2-grammeja on siis 256^2 kappaletta [59].

4.4.1 N-grammianalyysi poikkeavuuksien etsinnässä

Wang ja Stolfo [61] yrittivät havaita poikkeavuuksia yhteyksien hyötykuormasta. He analysoivat 1-grammeja. Opetusvaiheessa normaalista liikenteestä luotiin malli pohjautuen frekvenssijakaumaan. Tämän jälkeen testausvaiheessa uutta hyötykuormaa verrataan normaaliin käyttäen Mahalanobis-etäisyyttä. Liian kaukana normaalista olevat hyötykuormat aiheuttavat hälytyksen. Liikenne jaotellaan käytetyn portin mukaan, esimerkiksi HTTP-liikenne (portti 80) tai SSH-liikenne (portti 22). Poikkeavaa liikennettä voidaan verrata sensoreista kerättyyn tietoon ja tämän perusteella järjestelmä voi tehdä joitain toimenpiteitä, kuten yhteyden katkaiseminen tiettyyn sivustoon. Järjestelmä on nimeltään PAYL, ja tältä osin se siis muistuttaa IPS-järjestelmää (ks. 3.9). Testidatana käytettiin DARPA 1999 -dataa kuten useassa muussakin tutkimuksessa. Kun sallittiin 1% vääriä hälytyksiä, päästiin parhaimmillaan noin 60% tunnistamistarkkuuteen. Lisäksi järjestelmää testattiin New Yorkilaisesta Columbian yliopistosta kerätyllä CUCS-datasetillä. Etukäteistietoa mahdollisista hyökkäyksistä ei ollut, mutta järjestelmä löysi onnistuneesti useita oikeita

puskuriylivuoto- ja CodeRed II -hyökkäyksiä. CodeRed II on mato, jonka hyökkäys kohdistuu Microsoftin IIS -palvelinohjelmistoon [62].

Wang ym. [63] jatkoivat tutkimusta aiheesta, koska PAYL-järjestelmä huomattiin haavoittuvaiseksi tietyille hyökkäyksille. Tällä kertaa käytettiin korkeamman tason n-grammeja ($n > 2$), mutta esiintymismääriä ei tallennettu. Koska n:n kasvaessa erilaisten n-grammien määrä kasvaa eksponentiaalisesti, he pienensivät datan määrää käyttämällä binääristä tallennusta (esiintyykö tietty n-grammi aineistossa vai ei) ja Bloomin suodatinta (engl. *bloom filter*) [64]. Bloomin suodattimella voidaan tutkia, kuuluuko alkio joukkoon vai ei. Lisäksi oli käytössä SNORT-ohjelman suodatustietoon perustuva toinen suodatin, jossa oli tietoa tunnetuista hyökkäyksistä. Poikkeavuus laskettiin sekä käyttäen tietoa n-grammeista, joita ei esiintynyt opetusdatassa lainkaan että tunnetuista haitallisista n-grammeista. Bloomin suodattimen takia oli mahdollista käyttää eri tasojen n-grammeja yhdessä. Lisäksi menetelmässä hyödynnettiin satunnaistamista siten, että hyökkääjän ei ole mahdollista tietää mihin osaan hyötykuormasta haitallista koodia voisi sisällyttää. Parhaimmillaan käytetystä testidatasta löytyivät kaikki madot ja hyökkäykset erittäin alhaisella väärien positiivisten määrällä.

Hubballi ym. [36] menetelmä perustuu myös korkeamman tason n-grammeihin. Heidän menetelmänsä generoi aluksi 3-grammit. Datassa esiintyneistä n-grammeista tallennetaan esiintymistiheydet. N-grammit jaotellaan eri luokkiin, joista vähiten esiintyneet saavat isoimmat poikkeuslukupisteet. Eri luokkien määrä k saadaan kaavalla $k = [1 + log(\textit{Erilaisten n-grammien määrä})]$. Jokaisella n-grammilla on siis joku pistemäärä. Yhdellä datarivillä esiintyvät n-grammit pisteineen muodostavat koko datarivin poikkeavuusluvun. Mikäli tietty raja-arvo ylitetään, luokitellaan rivi poikkeavuudeksi. Korkeampien n-grammien laskeminen on laskennallisesti raskaampaa, mutta mikäli poikkeavuudet löytyvät jo alemmilla tasoilla, ei vaativaa laskentaa välttämättä tarvita. Menetelmää sovellettiin DARPA 1999 -dataan. Sinänsä hieman keinotekoinen ja opetusdataltaan puhdas datasetti ei tuottanut ongelmia, joten lähes kaikki hyökkäykset löydettiin. Tutkijat olivat kiinnostuneempia menetelmän toimivuudesta oikean datan kanssa. Yleensä oikeassa datassa myös opetusdata sisältää poikkeavuuksia. Tutkimuksissa huomattiin kuitenkin, että korkeamman tason n-grammit sietävät tätä paremmin kuin alemmat. Jos menetelmää sovellettiin alkaen 3-grammeista päätyen aina 8-grammeihin asti, korkeamman tason n-grammit kompensoivat datan epäpuhtautta suhteellisen hyvin.

4.5 Pääkomponenttianalyysi

Kerätty data on usein määrältään suurta ja piirteet moniulotteisia. Tämä tekee datan analysoinnista erittäin aikaa vievää ja haastavaa. Pääkomponenttianalyysi (engl. *principal component analysis*, lyh. *PCA*) pyrkii vähentämään datan ulottuvuuksien määrää jotta analysointi olisi helpompaa [16]. Tässä mielessä pääkomponenttianalyysi muistuttaa diffuusiokarttoja (ks. 4.3). Pääkomponenttianalyysiä on käytetty esimerkiksi kuvanpakkauksessa, hahmontunnistuksessa ja tietoturvahyökkäysten havaitsemisessa.

Tarkoituksena on muuntaa n-ulotteinen data d-ulotteiseksi siten, että $d < n$. Datan piirteet riippuvat toisistaan, mutta uudet saadut piirteet ovat toisistaan riippumattomia. Käytännössä tuloksena saadaan alkuperäisen datan piirteiden lineaarikombinaatioita [16]. Ensimmäinen pääkomponentti on lineaarikombinaatio jonka varianssi on suurin, toisen pääkomponentin varianssi on toiseksi suurin jne. Yleensä muutamat ensimmäiset pääkomponentit kuvaavat suurimman osan varianssista, joten loput voidaan unohtaa menettämättä tarkkuutta oleellisesti [65].

Määritellään ensin datamatriisi $X_{n\times m}$ yhtälössä 4.17 [65]. Se sisältää n kappaletta m-pituisia vektoreita.

$$X_{n\times m} = \begin{bmatrix} x_{11} & \dots & x_{1m} \\ x_{21} & \dots & x_{2m} \\ \dots & \dots & \dots \\ x_{n1} & \dots & x_{nm} \end{bmatrix} = [\mathbf{x}_1, \dots, \mathbf{x}_n] \tag{4.17}$$

Keskimääräinen havainto määritellään

$$\boldsymbol{\mu} = \frac{1}{n}\sum_{i=1}^{n} \mathbf{x_i} \tag{4.18}$$

Keskipoikkeama on

$$\boldsymbol{\Phi_i} = \mathbf{x}_i - \boldsymbol{\mu} \tag{4.19}$$

Nyt kovarianssimatriisi on

$$C = \frac{1}{n} A A^T \tag{4.20}$$

missä $A = [\boldsymbol{\Phi_1}, \boldsymbol{\Phi_2}, \dots, \boldsymbol{\Phi_n}]$.

Nyt matriisista C voidaan laskea ominaisarvot ja vastaavat ominaisvektorit käyttäen singulaarihajotelmaa. Jos ominaisvektoreita on yhteensä m kappaletta, vain k

ensimmäistä valitaan ja loput $(m - k)$ sisältävät vain "kohinaa". k voidaan valita käyttäen yhtälöä 4.21.

$$\frac{\sum_{i=1}^{k} \lambda_i}{\sum_{i=1}^{m} \lambda_i} \geq \alpha \tag{4.21}$$

missä α on aliavaruuden ja koko avaruuden varianssien suhde. Se voidaan määritellä haluttuun arvoon josta sitten lasketaan parametrin k arvo. Nyt voidaan määritellä $m \times k$ -kokoinen matriisi U jonka sarakkeet koostuvat k ominaisvektorista. Data voidaan esittää pääkomponentteina projisoimalla data k-ulotteiseen aliavaruuteen yhtälön 4.22 mukaisesti.

$$\mathbf{y_i} = U^T \mathbf{\Phi_i} \tag{4.22}$$

4.5.1 Pääkomponenttianalyysi poikkeavuuksien havaitsemisessa

Shyu ym. [66] sovelsivat pääkomponenttianalyysia poikkeuksien löytämisessä verkkoliikenteestä. Yleensä käytetään ensimmäisiä pääkomponentteja jotta mahdollisimman suuri osa varianssista saadaan analysoitua. Tutkimusryhmä kuitenkin käytti tämän lisäksi myös viimeisiä komponentteja parantamaan menetelmän tarkkuutta. Poikkeava liikenne on usein hyvin erilaista ja sillä ei välttämättä ole mitään yhteyttä normaaliin liikenteeseen. Analyysejä suoritettiin 5 kappaletta siten, että jokainen ajettiin erikseen tuloksien varmistamiseksi. Jokaisessa ajossa menetelmää opetettiin 5000 satunnaisesti valitulla rivillä. Testidatana toimi KDD CUP 99 -data [67]. Koko opetusdata sisältää noin puoli miljoonaa yhteyttä ja testidata noin 300 000. Tutkimuksessa käytettiin kuitenkin opetusdataa analyysin molemmissa vaiheissa. Data sisältää tietoa TCP-paketeista, jotka on kuvattu 41 piirteellä. Erilaisia hyökkäystyyppejä on kymmeniä, mutta tutkimuksessa hyökkäyksiä ei eroteltu toisistaan. Menetelmää testattiin 92 270 normaalilla yhteydellä ja 39 674 hyökkäyksellä, jotka kaikki oli satunnaisesti valittu datan joukosta. Tunnistamistarkkuudeksi saatiin noin 99% silloinkin, kun väärien hälytysten määrä pidettiin pienenä. Lisäksi huomattiin, että sekä suurien että pienien pääkomponenttien käyttö yhdessä paransi tulosta huomattavasti. Lisäksi menetelmä sisälsi mekanismin epäpuhtaan opetusdatan käsittelyyn, joten hyökkäysten pääseminen opetusdatan joukkoon ei haitannut toimintaa.

Bouzida ym. [68] testasivat lähimmän naapurin menetelmän (engl. *nearest neighbour algorithm*, lyh. *NN*) sekä päätöspuun (engl. *decision tree*, lyh. *DT*) toimintaa poikkeuksien havaitsemisessa. Näihin olemassa oleviin menetelmiin lisättiin pää-

komponenttianalyysin toiminta, ts. datan ulottuvuuksien määrää pienennettiin sen avulla ja tuloksia verrattiin menetelmien itsenäiseen toimintaan. NN-menetelmän tapauksessa pääkomponenttianalyysin käyttö paransi tarkkuutta lievästi. Kovin suurta vaikutusta ei kuitenkaan ollut. Päätöspuiden tarkkuus sen sijaan huononi hieman. Toisaalta ero tarkkuudessa ei ollut suuri ja menetelmän opetus nopeutui pieniulotteisemman datan takia monikertaisesti. Erityishuomiona R2L-hyökkäysten löytyminen oli todella vaikeaa johtuen niiden pienestä määrästä opetusdatan joukossa. Pääkomponenttianalyysi ei tuonut tähän parannusta, vaan tarkkuus itseasiassa pienentyi entisestään molempien menetelmien tapauksessa.

Wang ym. [65] käyttivät myös pääkomponenttianalyysia. Aluksi menetelmä opetettiin, ts. suoritettiin pääkomponenttianalyysi yhtälöjen 4.18–4.21 mukaan. Tämä tehtiin siis normaalilla datalla. Kun nyt halutaan testata tietyn datavektorin poikkeavuutta, se projisoidaan aliavaruuteen yhtälön 4.22 mukaan. Tämän jälkeen voidaan laskea vektorin ja sen projektion euklidinen etäisyys. Jos testivektori on normaali, se on hyvin samankaltainen sen projektion kanssa ja niiden välinen etäisyyskin on pieni. Etäisyydelle voidaan asettaa jokin raja-arvo jota suuremmat arvot luokitellaan poikkeavuuksiksi. Lisäksi jokaiselle hyökkäystyypille on luotu omat aliavaruutensa. Mikäli poikkeavuus löydetään, se voidaan projisoida vastaavasti jokaisen hyökkäystyypin aliavaruuteen. Pienin etäisyys vektorin ja projektion välillä tarkoittaa, että hyökkäys kuuluu todennäköisimmin kyseiseen hyökkäystyyppiin. Menetelmän testauksessa käytettiin DARPA 1998 -dataa. Keskimäärin tunnistamistarkkuus oli 98.8% väärin hälytysten ollessa 0.4%. Tiedusteluhyökkäykset löytyivät huonoimmin, sillä vain 80.7% niistä löytyi. Menetelmä siis löytää uudet hyökkäykset, mutta pystyy lisäksi tunnistamaan hyökkäystyypin.

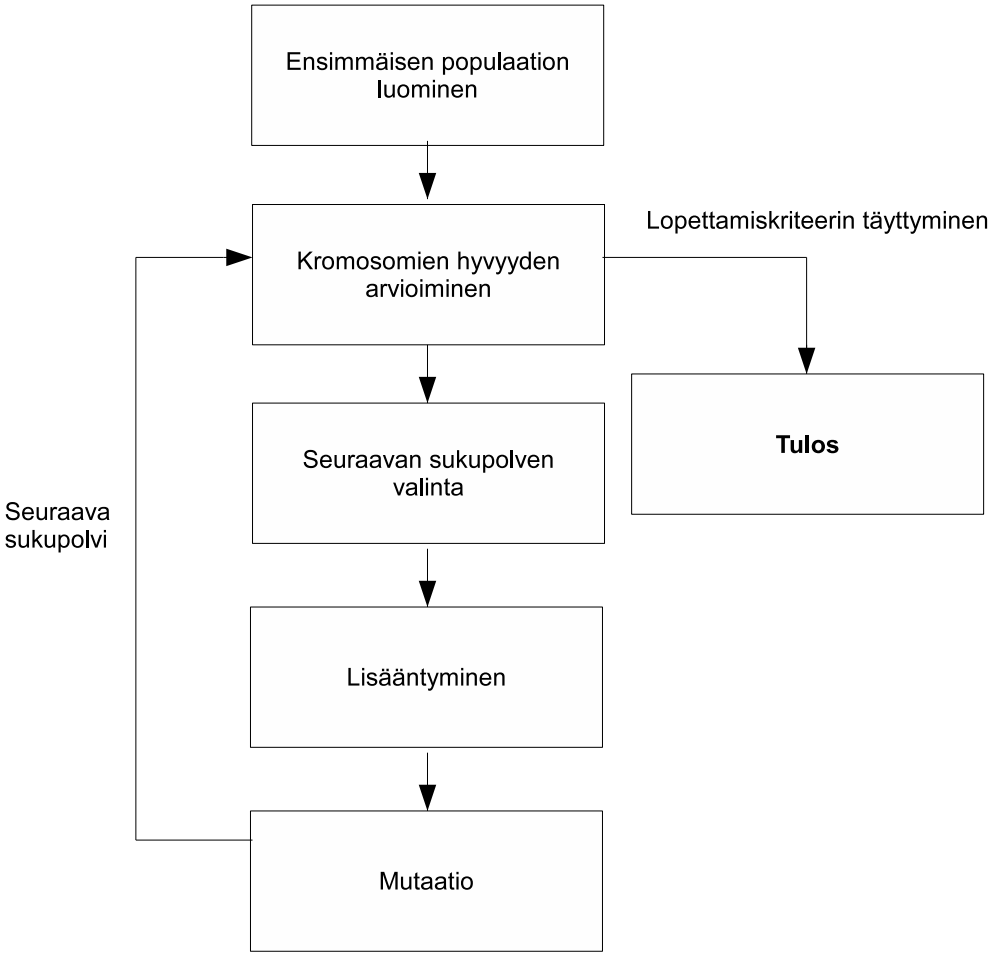
4.6 Geneettiset algoritmit

Geneettiset algoritmit (engl. *genetic algorithm*, lyh. *GA*) ovat algoritmeja, jotka perustuvat genetiikan ja luonnonvalinnan periaatteisiin [69]. Niitä on käytetty ratkaisemaan ongelmia monella alalla, kuten tietotekniikassa, matematiikassa ja taloustieteessä [15].

Aluksi luodaan satunnainen joukko *kromosomeja*, jotka ovat ratkaistavan ongelman ilmentymiä. Riippuen ongelmasta, kromosomien paikat on koodattu käyttäen *geenejä*, jotka voivat käytännössä olla esimerkiksi bittejä. Jokaisessa evoluution vaiheessa luodaan uusi populaatio kromosomien pohjalta, ja jokaisen kromosomin hy-

vyys arvioidaan. Uudet kromosomit luodaan siis *mutaatioiden* avulla. Kun tietty kriteeri on täytetty, algoritmin suoritus päättyy ja paras kromosomi valitaan ongelman ratkaisuksi [70]. Lisäksi ennen mutaatiota luodaan uudet kromosomit eli tehdään lisääntyminen *crossover*-operaatiolla, jossa geenejä vaihdetaan kromosomien kesken tietyllä tavalla [15]. Kuvassa 4.4 on geneettisen algoritmin vaiheet.

Geneettisten algoritmien etuna on se, että ne ovat hyvin vikasietoisia ja erittäin mukautuvia [16]. Koska ne luovat useita jälkeläisiä, algoritmeilla voidaan tutkia useita erilaisia ratkaisuvaihtoehtoja rinnakkain. Joka vaiheessa huonot vaihtoehdot voidaan jättää pois. Tämän takia ne soveltuvat hyvin sellaisten ongelmien ratkaisuun, missä mahdollisten ratkaisuvaihtoehtojen määrä on erittäin suuri [15]. Geneettisiin algoritmeihin perustuvia menetelmiä on myös helppo opettaa uudelleen, joten ne mukautuvat hyvin erilaisiin tilanteisiin.



Kuva 4.4: Geneettisen algoritmin toiminta.

4.7 Keinoimmuunijärjestelmät

Keinoimmuunijärjestelmät (engl. *artificial immune system*, lyh. *AIS*) pohjautuvat nimensä mukaisesti ihmisen immuunijärjestelmän toimintaperiaatteille. Immuunijärjestelmä (*human immune system*, lyh. *HIS*) pystyy tunnistamaan ja tuhoamaan haitalliset ja aikaisemmin tuntemattomat hyökkääjät. Tätä samaa periaatetta on yritetty soveltaa tietotekniikassa, sillä periaatteessa IDS-järjestelmä pyrkii tekemään saman. Niinpä keinoimmuunijärjestelmät tarjoavat kiinnostavan lähtökohdan poikkeavuuksien havainnointiin. Kuten neuroverkkojenkin tapauksessa, lähtökohtana on käytetty joitain biologiasta tuttuja ominaisuuksia joita sovelletaan tietojenkäsittelyssä. Tarkoituksena ei ole kopioida esimerkiksi ihmisen biologista toimintaa tarkasti. Keinoimmuunijärjestelmillä on IDS-järjestelmien kannalta monia hyviä ominaisuuksia [71]:

- Hajautettavuus
- Itseorganisoituvuus
- Keveys
- Monikerroksisuus
- Diversiteetti
- Muokattavuus

Keinoimmuunijärjestelmät ovat siis hajautuneita ja sietävät häiriöitä hyvin siksi, että yksi poikkeavuuksien havaitsemisprosessi ei väärin toimiessaan haittaa koko järjestelmän toimintaa. Lisäksi ne toimivat täysin itsenäisesti, eikä valvontaa tai kontrollia tarvita. Ne koostuvat useasta kerroksesta, ja yhden kerroksen ongelmat eivät heijastu muihin. Järjestelmän toiminta ei riipu yksittäisen komponentin toiminnasta, vaan se voidaan tarvittaessa vaihtaa toiseen ja näin saadaan entistä parempi häiriönkestokyky [71].

Kun keinoimmuunijärjestelmiä sovelletaan poikkeavuuksien havaitsemiseen, voidaan käyttää montaa eri lähtökohtaa. Edelleen käytetyin on ns. *negative selection* -menetelmä, jonka Forrest ym. esittelivät vuonna 1994 [72]. Jos data esitetään esimerkiksi merkkijonoina, menetelmässä luodaan aluksi ilmaisimia (*detector*), jotka ovat

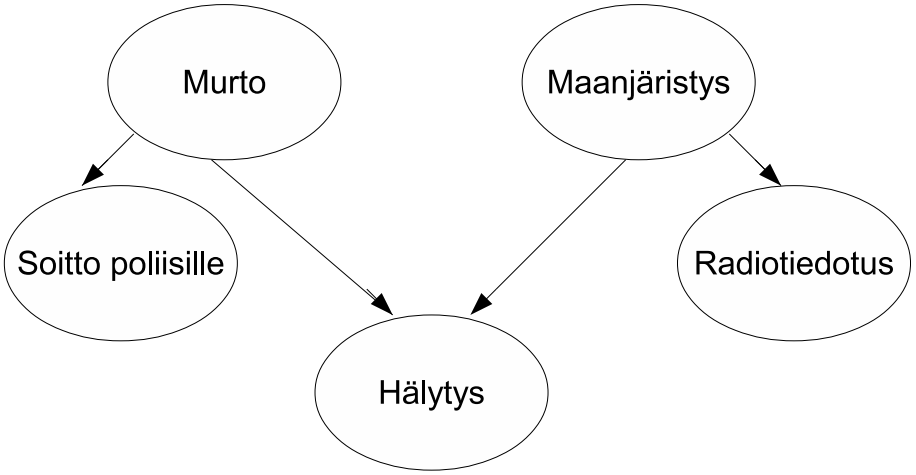
satunnaisia merkkijonoja ja eroavat normaalista datasta, ts. yksikään normaaliin dataan kuuluva merkkijono ei ole sama kuin joku ilmaisimista. Näitä ilmaisimia verrataan uuteen dataan, ja mikäli vastaavuus löytyy, tehdään hälytys. Myös osittain vastaavuus riittää eli esimerkiksi se, että n kappaletta merkkejä ovat samoja ja samoilla paikoilla. Menetelmä tuntuu yksinkertaiselta mutta toimii yllättävän hyvin. Etukäteistietoja poikkeuksista ei tarvita ja menetelmä toimii hajautetusti, eli pienemmillä lokaaleilla alueilla voi olla eri ilmaisimia [73]. On huomattava, että kaikkia mahdollisia ilmaisimia ei tarvitse luoda, vaan niitä luodaan satunnaisesti tietty määrä. Tätä määrää voidaan tarvittaessa muuttaa laskentatehon ja havaitsemistarkkuuden muuttamiseksi. Ilmaisimien määrä voidaan laskea tietylle todennäköisyydelle, joten riittää, että valitaan millä todennäköisyydellä poikkeavuudet täytyy havaita [72].

4.8 Bayesverkot

Bayesverkot (engl. *Bayesian network*) ovat graafisia malleja joilla voidaan kuvata muuttujien välisiä todennäköisyyspohjaisia suhteita [16]. Niillä on datan analysoinnissa monia hyödyllisiä ominaisuuksia [74]. Bayesverkkoja voidaan käyttää tilanteissa, joissa osa datasta puuttuu. Lisäksi niiden avulla voidaan mallintaa kausaalisia suhteita ja ennustaa tekojen seurauksia.

Bayesverkkoja on käytetty jonkun verran hyökkäysten havaitsemisjärjestelmissä [75, 76]. Bayesverkkojen suurimpia ongelmia ovat menetelmän hitaus ja se, että tarvitaan ennakkotietoa [5]. Osa toteutuksista käyttää yksinkertaisempaa *naive Bayes* -verkkoa, jossa oletetaan, että datan piirteet ovat riippumattomia toisistaan [16]. Tästä aiheutuu rajoituksia menetelmän toiminnalle, mutta se ratkaisee hitausongelman. Yksi iso ongelma eli ennakkotiedon tarve kuitenkin säilyy [5]. Niinpä Bayesverkot eivät sovellu ohjaamattomaan oppimiseen ja siten niiden käyttö on jäänyt lähinnä väärinkäytöksien havaitsemiseen.

Kuvassa 4.5 on erittäin yksinkertainen esimerkki Bayesverkosta. Usein kausaaliset suhteet ovat huomattavasti monimutkaisempia. Perusperiaate on kuitenkin aina sama.



Kuva 4.5: Yksinkertainen esimerkki Bayesverkosta.

5 Tiedon arkistointi ja esikäsittely

Järjestelmästä kerätty tietoliikennedata on kerättävä ja arkistoitava tietyssä muodossa. Tämä vaikuttaa suuresti siihen, minkälaista informaatiota on mahdollista kerätä datasta analysoitavaksi. Tässä esitellään palvelimien arkistointiformaatit ja esikäsittelyn toiminta.

5.1 Tiedon arkistointi ja kerääminen

Palvelindata saatiin yritykseltä, joka tarjoaa web-palveluja suurille yrityksille. Lokidata on kerätty Apache-palvelimista noin kuuden kuukauden aikana neljällä eri palvelimella. Jokainen palvelin tarjoaa useita palveluita, joten lokidata on vaihtelevaa. Dataa on kertynyt yhteensä noin 4 gigatavua, ja jokaisessa neljästä tiedostossa on noin 4 miljoonaa riviä. Tiedostoista kuitenkin suodatettiin ennen esikäsittelyä pois sellaiset rivit, joissa ei ollut HTTP-kyselyssä parametreja ollenkaan. Näin tiedon määrä pieneni noin kymmenesosaan. Poikkeavuuksien analysointi kohdistuu nimenomaan GET-kyselyn parametreihin, joten parametrittomien rivien sisällyttäminen ei ollut tässä vaiheessa mielekästä. Näin myös esikäsittelyyn kuluvaa aikaa saatiin lyhennettyä.

5.1.1 Oikea lokidata vai valmiit datasetit

Saatu data on erityisen mielenkiintoista siksi, että se on oikeasti käytössä olevilta web-palvelimilta kerättyä dataa. Usein vastaavanlaisissa tutkimuksissa käytetään valmiiksi luotua datasettiä josta on etukäteistietoa. Näin menetelmiä voidaan opettaa tuon tiedon avulla tunnistamaan erilaisia hyökkäyksiä. Tässä tapauksessa ennakkotietoa ei ollut, joten oppimisen on oltava ohjaamatonta.

Yleisimmät testauksessa käytetyt datasetit ovat DARPA 1998 ja DARPA 1999. Näissä on kuitenkin monia ongelmia, jotka voivat vaikuttaa tuloksiin. McHugh kritisoi näitä voimakkaasti [77]. Normaali liikenne ja hyökkäykset ovat molemmat keinotekoisesti luotuja. Ne eivät välttämättä vastaa oikeaa tietoliikennettä, sillä todellinen liikenne on todella vaihtelevaa. Hyökkäysten ja normaalin liikenteen suhde saattaa olla väärä. Todellista vaikutusta tuloksiin on vaikea määritellä, mutta on-

gelmat kasvavat siinä vaiheessa, kun käytetään ohjattuun oppimiseen perustuvia menetelmiä ja yritetään siirtää menetelmä todelliseen maailmaan. Jos menetelmä on opetettu tunnistamaan hyökkäyksiä esimerkiksi DARPA-datan avulla, se todennäköisesti löytää hyökkäykset saman datan joukosta helposti. Kun näin opetettua järjestelmää sovelletaan oikeaan tietoliikenteeseen, eivät oikeat hyökkäykset välttämättä löydykään, mikäli opetusdata ei vastaa todellisuutta. Myös Mahoney ja Chan löysivät monia ongelmia [78]. Ainakin DARPA 1999 -datassa hyökkäyksissä on piirteitä joita ei esiinny normaalissa datassa. Esimerkiksi hyökkäyksissä saattaa olla poikkeavia TTL-arvoja (engl. *time to live* eli datan elinaika). Todellisessa liikenteessä myös näitä TTL-arvoja esiintyy normaalissakin liikenteessä, joka vaikeuttaa hyökkäysten havaitsemista. Heidän mukaansa hyökkäykset oli tosin suurimmaksi osaksi mallinnettu hyvin, mutta normaalin liikenteen mallintaminen keinotekoisesti ei ole kannattavaa eikä välttämättä edes mahdollista. He ehdottavatkin oikean liikenteen lisäämistä testidatan joukkoon.

On huomattava, että vapaasti saatavilla olevia datasettejä ei ole kovin montaa. Oikean verkkoliikenteen kanssa kohdataan aina yksityisyysongelma, sillä käyttäjiä on usein mahdollista tunnistaa liikenteestä. Niinpä on erittäin onnekasta, että tässä tutkimuksessa voitiin käyttää oikeaa lokidataa. Sitä ei voi edellä mainitusta syystä kuitenkaan julkaista vapaasti saataville.

5.2 CLF-formaatti ja HTTP-kysely

Palvelindata on tallennettu CLF-muodossa [79](*combined log format*), joka on yleisesti käytetty lokiformaatti. Se sisältää paljon tietoa HTTP-kyselystä ja käyttäjästä, joka kyselyn on suorittanut. CLF-rivi sisältää seuraavaa tietoa:

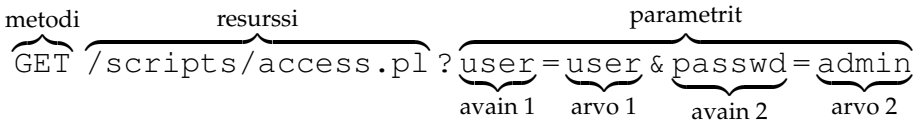
- Käyttäjän IP-osoite
- Aika ja aikavyöhyke
- HTTP-kysely (tyyppi, resurssi ja parametrit)
- Apache-palvelimen vastauskoodi
- Käyttäjälle siirretyn datan määrä
- Web-sivu, josta HTTP-kysely on tehty
- Tietoa käyttäjän selainohjelmistosta

Apache palvelimen vastauskoodi ilmaisee, onko kysely onnistunut. Numerolla 2 alkava koodi tarkoittaa onnistumista. Numero 3 tarkoittaa uudelleenohjausta, 4 ongelmaa asiakasohjelmistossa ja 5 ongelmaa palvelimella [79]. Käyttäjän selain kertoo itsestään tietoa palvelimelle, ja sekin tallennetaan lokitiedostoon. Listauksessa 5.1 on esimerkkirivi lokitiedostosta.

```
123.234.0.1 - - [10/April/2011:12:10:01 +0300]
"GET /scripts/access.pl?user=user&passwd=admin HTTP/1.1"
200 2680 "http://www.jyu.fi/testi.html"
"Mozilla/5.0 (SymbianOS/9.2;...)"
```

Listaus 5.1: Esimerkki CLF-lokirivistä.

Itse HTTP-kysely on poikkeavuuksien havaitsemisen kannalta ehkä tärkein osa. HTTP-kyselyn rakenne on kuvassa 5.1. Aluksi määritellään käytetty HTTP-metodi (esim. GET, POST tai HEAD), sen jälkeen käytetty (*uniform resource identifier*) eli käytetty resurssi [80]. GET-pyynnössä on lisäksi yksi tai useampi parametri ja niiden arvot. Juuri parametriosaan kohdistuu suurin mielenkiinto, koska siihen mahdollista sisällyttää haitallista koodia, esimerkiksi SQL-rivejä. Parametrittomat kyselyt kohdistuvat staattiseen web-sisältöön, jossa ei ole enää merkittäviä tietoturvauhkia ainakaan HTTP-kyselyn kannalta. Niinpä sellaiset rivit voitiin poistaa. Parametrin nimellä ei siis pystytä itsessään hyödyntämään vakavaa haavoittuvuutta, ellei palvelin ole todella vanhentunut tai huonosti konfiguroitu. Sen sijaan olemassa olevien parametrien arvoilla voidaan mahdollisesti tehdä jotain haittaa järjestelmälle. Siksi juuri parametrien arvot ovat kiinnostavimpia tässä yhteydessä.



Kuva 5.1: HTTP GET -kyselyn rakenne.

5.3 Esikäsittely

CLF-lokiformaatti on tekstimuotoinen, joten sitä ei voida sellaisenaan käsitellä ilman jonkinlaista käsittelyä numeeriseen muotoon. Esikäsittelijänä käytetään tässä

Joel Lehtosen toteuttamaa *PhasefulSplitter*-ohjelmaa [81], joka on ohjelmoitu Haskel-kielellä. Se käsittelee edellä kuvattua lokiformaattia numeeriseen matriisimuotoon. Käsittely on toteutettu vaiheittain, ja joka vaiheessa sen hetkinen käsittely tallennetaan väliaikaistiedostoihin. On mahdollista myös käyttää useita säikeitä, jolloin prosessointi voidaan hoitaa useammalla prosessorilla tai prosessoriytimellä. Näin raskaasta laskennasta voidaan selviytyä nopeammin.

Ensimmäisessä vaiheessa ohjelma muodostaa listat käsiteltävänä olevista tiedostoista. Samalla tiedostot ryhmitellään palvelimen ja palveluiden mukaan. Sitten tiedostojen sisältö luetaan ja muutetaan tietorakenteeksi. Tiedostot on ryhmitelty käytetyn resurssin mukaan. Resurssiin kohdistuneiden HTTP-kyselyiden parametrien arvoista lasketaan 2-grammit. Samalla sellaiset sarakkeet, jotka vastaavat tietyssä resurssissa esiintymätöntä 2-grammia, poistetaan. Näin saadaan joukko tiedostoja, jotka kuvaavat tiettyyn resurssiin kohdistuneiden pyyntöjen 2-grammeja. Yksi sarake vastaa tiettyä 2-grammia, ja sarakkeen arvoksi tulee 2-grammin esiintymismäärä kyseisellä rivillä. Lisäksi ohjelma tallentaa mm. aikatiedot ja palvelimen vastauskoodit sekä tiedon siitä, missä lokitiedostossa ja millä rivillä kyseinen rivi esiintyy. Näin poikkeavat rivit voidaan jäljittää alkuperäisestä lokitiedostosta tarkempaa tutkimista varten. Palvelimille annetaan numeeriset tunnisteet, esimerkiksi *Palvelin 1* saa arvon 1 ja niin edelleen. Lisäksi koska jokaiselta palvelimelta voi tulla useampia erillisiä lokitiedostoja, jokainen tiedosto saa oman numeerisen tunnisteensa automaattisesti. Tämän lisäksi tarvitaan tieto lokitiedoston rivistä. Näin esimerkiksi rivi, joka alkaa sarakkeilla `1,2,12345` viittaa palvelimen 1 tiedostoon numero 2 ja edelleen tämän tiedoston riville 12345. Lopuksi nämä tiedot kirjoitetaan matriisimuotoon tiedostoon.

Listauksessa 5.2 on ohjelman tulostamia n-grammirivejä. Muut sarakkeet on jätetty selvyyden vuoksi pois, koska niitä ei otettu mukaan varsinaiseen analyysiin. Tällä hetkellä tärkein tieto liittyy n-grammeihin, joten muut sarakkeet vain sekoittavat matemaattista analyysia. Niinpä ainoastaan n-grammeja käytetään analyysissa. On huomattava, että jokaisessa resurssissa esiintyvien erilaisten n-grammien määrä vaihtelee. Tämän takia sarakkeiden määrä ei ole vakio. Kuitenkin yhden tiedoston sisällä se pysyy samana. Joissain resursseissa n-grammeja ei ole välttämättä kovin montaa, mutta toisissa niitä voi olla tuhansia. Näin osa tiedostoista sisältää tuhansia sarakkeita. Myös rivien määrä vaihtelee suuresti. Käytetyimmät resurssit sisältävät kymmeniä tuhansia rivejä. Tämä saattaa asettaa haasteita analyysimenetelmien skaalautuvuudelle. Jokainen matriisitiedosto analysoidaan erikseen, eli poik-

keavuudet pyritään havaitsemaan tiettyyn resurssiin kohdistuneen liikenteen joukosta.

```
1 ,0 ,1 ,2 ,1 ,0 ,3 ,1 ,2 ,1 ,1 ,0 ,2 ,3 ,1 ,0 ,1 ,2 ,1 ,0 ,1 ,2 ,0 ,1 ,1 ,0 ,2
0 ,1 ,0 ,0 ,1 ,1 ,2 ,0 ,1 ,0 ,0 ,1 ,1 ,0 ,2 ,0 ,1 ,1 ,0 ,0 ,0 ,1 ,1 ,0 ,1 ,0 ,1
1 ,0 ,0 ,1 ,2 ,0 ,6 ,0 ,7 ,7 ,8 ,0 ,0 ,1 ,1 ,0 ,2 ,0 ,1 ,0 ,1 ,0 ,2 ,2 ,0 ,0 ,0
```

Listaus 5.2: PhasefulSplitter-ohjelman tekemän n-grammianalyysin esimerkkirivejä.

Ohjelmalla on myös yksi erittäin tärkeä ominaisuus. Esikäsittelyssä samalla anonymisoidaan lokirivit. Ne sisältävät tietoa mm. käyttäjien IP-osoitteista, joten periaatteessa tietyt rivit voidaan yhdistää johonkin tiettyyn käyttäjään. Yksityisyydensuojan takia on tärkeää, että esikäsitellyt lokirivit eivät tätä tietoa sisällä. Tässä vaiheessa tietoa eri käyttäjistä ei sinänsä otettu mukaan analyysiin eivätkä ne esiinny lopullisissa esikäsitellyissä tiedostoissa. Käyttäjätieto olisi kuitenkin mahdollista ottaa ohjelman seuraavaan versioon mukaan siten, että yhden käyttäjän useammat erilliset lokirivit voitaisiin koostaa yhteen ja näin ottaa käyttäjän toiminta huomioon. Tällöinkin anonymisointi pitää säilyttää. Tässä vaiheessa rivejä käsitellään kuitenkin yksittäisinä tapahtumina.

6 Tutkimuksen menetelmät ja toteutus

Esikäsitellystä datasta valittiin pieni osajoukko sen kiinnostavuuden ja selkeyden takia. Analyysivaiheessa testattiin kolmea eri menetelmää ja verrattiin niiden toimintaa toisiinsa. Testidatan koostumus ja koko sekä varsinaiset käytetyt implementaatiot on esitelty tässä.

6.1 Testidata

Alkuperäinen lokidata jaoteltiin siis esikäsittelyvaiheessa eri tiedostoihin HTTP-pyynnössä olevan resurssin mukaan. Näistä yksi resurssi valittiin tarkempaan analyysiin siksi, että se sisältää oikeita hyökkäyksiä ja normaali liikenne erottuu suhteellisen selvästi. Näin lokidatasta oli mahdollista löytää oikeasti poikkeavat rivit ja pystytään laskemaan, kuinka hyvin menetelmät toimivat näiden poikkeavuuksien etsimisessä.

Resurssi sisältää yhteensä 4292 lokiriviä, joista 1293 riviä liittyvät hyökkäyksiin. Hyökkäykset erottuvat HTTP-kyselyn parametriosasta, joten ne pitäisi olla mahdollista löytää n-grammianalyysin avulla. Testidatassa ei alkuperäiseen datamassaan nähden ole kovin montaa riviä, mutta tämä johtuu rivien ryhmittelystä resurssien mukaan. Näitä tiedostoja tulee määrällisesti paljon, mutta ne täytyy kaikki analysoida erikseen poikkeavuuksien havaitsemiseksi mikäli halutaan tutkia koko lokidata. Tämä ei aseta kovin suuria vaatimuksia menetelmän skaalautuvuuden suhteen, mutta koska tiedostoja on monta, tarvitaan myös analysointiajoja yhtä paljon. Niinpä pienikin hidastuminen saattaa pidentää koko prosessia paljonkin. Tässä tutkimuksessa keskityttiin menetelmien tarkkuuden arvioimiseen, jotan vain yksi resurssitiedosto otettiin analysoitavaksi.

6.2 N-grammianalyysi

Tätä tutkimusta varten tehtiin erittäin yksinkertainen pelkästään n-grammianalyysin tuloksiin pohjautuva menetelmä. Toteutus perustuu siihen oletukseen, että poikkeavat lokirivit sisältävät harvoin tai ei ollenkaan esiintyviä n-grammeja. Toteutus teh-

tiin *Python*-ohjelmointikielellä, mutta se voitaisiin toteuttaa lähes millä tahansa kielellä. Myöskään mitään erityisiä optimointeja ei tehty tässä vaiheessa. Joka tapauksessa ohjelman aikavaativuus ei kasva kovin paljoa datamäärän kasvaessa.

Kuvassa 6.1 on kuvattu n-grammirivit. Tiedostossa on siis n riviä ja jokaisella rivillä m saraketta. Yksittäisen rivin tietyn n-grammin esiintymistiheys on nyt x_{ij} rivillä i ja sarakkeessa j. Ensin lasketaan jokaisen sarakkeen (eli jokaisen yksittäisen n-grammin) esiintymistiheydet yhteen. Ne lasketaan siis sarakkeelle j

$$ngrams_j = \sum_{i=1}^{n} x_{ij} \tag{6.1}$$

Nyt voidaan laskea yksittäisen rivin tietyn n-grammin poikkeavuuspistemäärä jakamalla n-grammin esiintymistiheys tietyllä rivillä sen esiintymistiheydellä kaikilla riveillä yhteensä, eli

$$score_{ij} = \frac{x_{ij}}{ngrams_j} \tag{6.2}$$

missä $0 < score_{ij} < 1$.

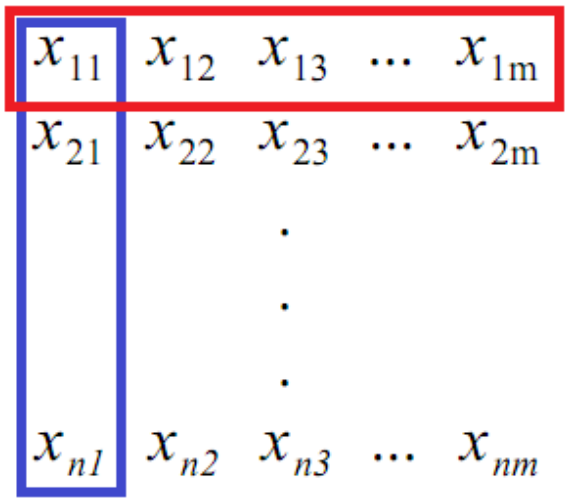
Tästä saadaan lopulta laskettua koko rivin poikkeuspistemäärä summaamalla rivin n-grammien pisteet yhteen, eli

$$totalscore_i = \sum_{j=1}^{m} score_{ij} \tag{6.3}$$

Ohjelma siis antaa harvoin esiintyville n-grammeille korkeamman pistemäärän, kun taas lähes kaikilla riveillä esiintyvien n-grammien pistemäärä on lähellä nollaa. Tämän lisäksi täytyy määritellä joku raja-arvo poikkeavuuspisteille, jonka ylittävät rivit tulkitaan poikkeavuuksiksi. Tätä arvoa voidaan tarvittaessa muuttaa ja siten vaikuttaa ohjelman tarkkuuteen ja väärien hälytysten määrään.

Mitään varsinaista esitietoa datasta ei tarvitse ohjelmalle antaa. Se käy datan läpi aina kahteen kertaan, ensin laskien n-grammien esiintymistiheydet sekä tärkeimmät tunnusluvut (esim. keskiarvo) joita tarvitaan poikkeavuuspisteiden antamiseen sekä tarvittavan raja-arvon määrittämiseen. Toisella ajokerralla voidaan antaa pisteet jokaiselle riville ja luokitella liian suuren pistemäärän saavat rivit poikkeavuuksiksi.

Tämän tutkimuksen testidatalla koko analyysiin meni vain muutama sekunti, mutta suurempikaan datamäärä ei pitäisi koitua ongelmaksi, koska aikavaativuus ei kasva eksponentiaalisesti.



Kuva 6.1: N-grammirivit.

6.3 LIBSVM

LIBSVM [82] on kirjasto, joka sisältää työkaluja tukivektoriluokitteluun, regressioon ja ryhmittelyyn. Myös moniluokkainen luokittelu on mahdollista. Se on ladattavissa vapaasti osoitteesta `http://www.csie.ntu.edu.tw/ cjlin/libsvm/`. LIBSVM on toteutettu eri alustoille ja ohjelmointikielille, esimerkiksi *Python*-ohjelmointikielelle ja *MATLAB*-ohjelmistolle. Sen käyttäminen on varsin yksinkertaista ja peruskäyttö onnistuu muutamalla komennolla.

Luokittelijan käyttö koostuu yleensä seuraavista vaiheista [44]:

- Datan muuntaminen sopivaan formaattiin
- Datan skaalaus
- Kernelifunktion valinta
- Parametrien C ja γ valinta
- Menetelmän opettaminen opetusdatalla ja valituilla parametreilla
- Varsinainen testaus

Aluksi data on oltava oikeassa formaatissa. Listauksessa 6.1 on esimerkkirivejä. Ensimmäisenä jokaisella rivillä on datarivin luokka. Tämä liittyy siis ohjattuun

oppimiseen, jossa jokaisen luokan edustajat pitää erikseen opettaa menetelmälle. Mikäli käytetään yksiluokkaista tukivektorikonetta kuten tässä tutkimuksessa, ei luokkaa tarvitse tietää eikä sitä oteta huomioon. Loput sarakkeista sisältävät tiedot sarakkeen numerosta ja sen arvosta. Nollasarakkeita ei tarvitse merkitä erikseen, sillä ne jätetään vain pois tiedostosta. Esimerkkilistauksen ensimmäisellä rivillä esimerkiksi sarakkeet 1–5 ovat nollia, koska ensimmäisenä on sarake 6. Näin tietoa ei menetetä, mutta ainakin harvan datan tapauksessa näin pystytään säästämään paljon tilaa, mikäli tiedostot olisivat muuten suuria. Tämän tutkimuksen kohteena olevan lokidatan tapauksessa näin onkin. Toisaalta nollasarakkeet voi myös merkitä normaalisti, mikäli se on helpompaa.

```
1 6:1 8:1 15:1 21:1 29:1 33:1 34:1 37:1 42:1 50:1 53:1 57:1
2 6:1 8:1 20:1 21:1 23:1 33:1 34:1 36:1 42:1 50:1 53:1 57:1
2 2:1 8:1 19:1 21:1 27:1 33:1 34:1 36:1 44:1 50:1 53:1 57:1
1 6:1 7:1 19:1 21:1 29:1 33:1 34:1 37:1 44:1 50:1 53:1 57:1
2 6:1 8:1 14:1 22:1 28:1 33:1 35:1 36:1 42:1 51:1 53:1 57:1
```

Listaus 6.1: Esimerkkirivejä LIBSBM:n dataformaatista.

Datan skaalaus on tulosten kannalta erittäin tärkeää. Muuten sarakkeet, joiden arvot vaihtelevat suurella skaalalla saavat liian suuren painoarvon analysoinnissa ja tulokset vääristyvät. Tällä kertaa muuttujat skaalattiin välille $[0, 1]$, mutta usein skaalaus tehdään myös välille $[-1, +1]$. Lisäksi testidata on skaalattava samoin kuin opetusdata on skaalattu. Esimerkiksi jos opetusdatan yhden sarakkeen muuttujien arvot vaihtelevat välillä $[-10, +10]$ ja ne skaalataan välille $[-1, +1]$ ja testidatan vastaavan sarakkeen arvot vaihtelevat välillä $[-11, +8]$, ne on silloin skaalattava välille $[-1.1, +0.8]$ [44].

Tämän jälkeen on valittava käytettävä kernelifunktio. Testaus aloitettiin käyttämällä yleisintä vaihtoehtoa eli radiaalista kernelifunktiota (*RBF-kernelifunktio*), joka on useimmiten hyvä vaihtoehto. Tulokset olivat riittävän hyviä, joten tätä valintaa ei muutettu. Radiaalinen kernelifunktio toimii hyvin erityisesti silloin, kun luokkien ja datarivien arvojen välinen riippuvuus ei ole lineaarista. Muuten voidaan myös käyttää lineaarista kernelifunktiota, joka voi olla joissain tilanteissa nopeampi.

Lisäksi on valittava RBF-kerneliin kuuluvat parametrit C ja γ. LIBSVM sisältää aputyökalun näiden arvojen määrittämiseen, koska mitään ennakkotietoa sopivista arvoista ei ole. Työkalu jakaa opetusdatan pienempiin osiin, joiden lukumäärä

on v. Näistä yksi osa valitaan testattavaksi, ja lopuilla osilla ($v-1$ kappaletta) testataan kuinka hyvin opetus onnistuu. Eri parametrien arvoja kokeillaan, jonka jälkeen parhaat valitaan opetusvaiheeseen. On kuitenkin huomattava, että tämä toimii parhaiten normaalissa luokittelussa. Koska nyt opetusdatana on vain yksinkertaista normaalia liikennettä, menetelmä ei ota huomioon poikkeavuuksia parametrien valinnassa, koska etukäteistietoa poikkeavuuksista ei vielä ole. Lisäksi opetusdatan joukossa saattaa olla poikkeavuuksia mukana. Testissä päästiin hyviin tuloksiin valitsemalla alkuparametrit edellä kuvatulla tavalla. Tämän jälkeen parametria γ pienennettiin niin kauan, että päästiin hyvään tunnistamistarkkuuteen. Aluksi γ oli liian suuri, ja menetelmä tulkitsi liian monta normaalia riviä poikkeavuuksiksi. Liian pieni γ taas jätti monta hyökkäystä havaitsematta. Sopivalla parametrilla päästiin hyvään tulokseen, mutta täysin automaattisesti sopivia parametrien arvoja ei pystytty määrittämään, vaan siihen vaadittiin useita testejä. Tuloksissa raportoidut arvot (ks. 7.2) saatiin siis parhailla parametrien arvoilla.

Kun tiedostoformaatti, skaalaus sekä kernelifunktion ja sen parametrien valinta on tehty, voidaan suorittaa varsinainen opetus yhdellä komennolla. Opetusvaiheessa saatu malli tallennetaan erilliseen tiedostoon. Tämän jälkeen testausvaiheessa myös yhdellä komennolla voidaan edellä saatua mallitiedostoa käyttäen suorittaa varsinainen analyysi. Ohjelma tulostaa analyysin tulokset samassa järjestyksessä kuin testidatan rivit ovat. Normaaliksi luokitellut rivit saavat siis arvon 1 ja poikkeavuudet arvon -1.

LIBSVM toimi testissä riittävän nopeasti niin, että aikavaativuus ei muodostunut ongelmaksi. On huomattava, että eniten aikaa vie skaalaus ja parametrien määrittäminen. Opetus tapahtuu nopeasti, ja itse analyysivaihe saatiin suoritettua tässä muutamassa sekunnissa, vaikka käsiteltiin isompaa datamäärää kuin opetusvaiheessa.

6.4 Diffuusiokarttasovellus

Analyysissä käytettiin Tuomo Sipolan implementoimaa toteutusta diffuusiokartoille. Sovellus toimii MATLAB-ympäristössä ja sille syötetään esikäsittelyvaiheen mukaisia matriiseja datan piirteistä (ks. 5.3). Ulostulona saadaan lasketut matalan ulottuvuuden koordinaatit, joita voidaan sitten analysoida tarkemmin. Lisäksi sovelluksella voidaan klusteroida pisteitä sekä arvioida tulosten tarkkuutta eli laskea esimerkiksi se, kuinka monta prosenttia hyökkäyksistä löydettiin. Myös erilaisten kuvaa-

jien piirtäminen voidaan hoitaa samassa ympäristössä. Sovellus perustuu kohdassa 4.3 esiteltyyn teoriapohjaan [56,57].

Listauksessa 6.2 on ohjelman ulostulon mukaisia esimerkkirivejä. Riveistä nähdään datapisteiden lasketut matalan ulottuvuuden koordinaatit, eli pisteiden paikat uudessa avaruudessa joka on laskettu diffuusiokarttojen avulla. Lisäksi ohjelmalla voidaan suoraan tulostaa taulukon 3.1 mukaisia matriiseja, joista nähdään ohjelman toiminta ja tarkkuus.

```
−0.1695    0.0072   −0.0001
−0.1711    0.0078    0.0008
−0.1710    0.0077    0.0008
−0.1710    0.0078    0.0008
−0.1711    0.0077    0.0008
−0.1695    0.0073   −0.0001
−0.1712    0.0077    0.0007
−0.1711    0.0078    0.0007
−0.1711    0.0078    0.0007
−0.1711    0.0078    0.0008
```

Listaus 6.2: Diffuusiokarttasovelluksen ulostulorivejä.

7 Tulokset

Kaikki menetelmät kykenivät suhteellisen hyviin tuloksiin. Pieniä eroja kuitenkin oli. Alla on kuvattu tarkemmin eri menetelmien tulokset kuten löytyneet hyökkäykset, väärät hälytykset ja tarkkuus.

7.1 N-grammianalyysi

N-grammien esiintymistiheyteen perustuva menetelmä ei tarvitse erillistä opetusdataa ollenkaan. Se käsittelee koko dataa yhtenä kokonaisuutena. Kuitenkin jokin raja-arvo vaaditaan poikkeavuuksien määrittämiseksi. Useimmat poikkeavuusarvot jäivät erittäin pieniksi, ja toisaalta monet hyökkäysrivit saivat erittäin suuria arvoja. Useimmat arvot olivat pienempiä kuin 0.1. Pienimmät arvot olivat noin 0.006 ja suurimmat noin 1.9. Keskihajonta oli 0.393 ja keskiarvo 0.112.

Aluksi laskettiin tulokset raja-arvolla $1.1 * keskiarvo$, koska yleensä on tarkoituksenmukaista käyttää keskiarvoa jonkin verran suurempaa arvoa. Nämä tulokset ovat taulukossa 7.1.

		Havaittu	
		Normaali	Poikkeava
Todellinen	Normaali	2991	8
	Poikkeava	75	1218

Taulukko 7.1: N-grammianalyysi, $1.1 * keskiarvo$.

Koska tässä tapauksessa hyökkäysrivejä on melko paljon rivien kokonaismäärään nähden, laskettiin raja samaksi kuin keskiarvo. Tämä perustuu olettamukseen, että suurin osa normaaleista riveistä on keskiarvon alapuolella, koska suurempia arvoja saaneet hyökkäysrivit nostavat keskiarvoa. Tarkoituksena oli tutkia, nostavatko ne sitä riittävästi, jotta suurin osa normaalista liikenteestä jäisi keskiarvon alapuolelle ja siten niitä ei luokiteltaisi poikkeavuuksiksi. Tulokset ovat taulukossa 7.2. Näistä tuloksista on vielä laskettu tunnusluvut menetelmän arviointia varten.

		Havaittu	
		Normaali	Poikkeava
Todellinen	Normaali	2969	30
	Poikkeava	21	1272

Taulukko 7.2: N-grammianalyysin tulokset keskiarvolla.

$$\textit{True positive rate (TPR)} = \frac{TP}{TP + FN} = \frac{1272}{1293} = 98.38\% \tag{7.1}$$

$$\textit{False positive rate (FPR)} = \frac{FP}{TN + FP} = \frac{30}{2999} = 1.00\% \tag{7.2}$$

$$\textit{True negative rate (TNR)} = \frac{TN}{TN + FP} = \frac{2969}{2999} = 99.00\% \tag{7.3}$$

$$\textit{False negative rate (FNR)} = \frac{FN}{TP + FN} = \frac{21}{1293} = 1.62\% \tag{7.4}$$

$$\textit{Tarkkuus (accuracy)} = \frac{TP + TN}{TP + TN + FP + FN} = \frac{1272 + 2969}{4292} = 98.81\% \tag{7.5}$$

$$\textit{Täsmällisyys (precision)} = \frac{TP}{TP + FP} = \frac{1272}{1293} = 98.38\% \tag{7.6}$$

7.2 Tukivektorikone

Aluksi tukivektorikoneen toimintaa testattiin täysin puhtaalla opetusdatalla. Tämän tarkoituksena oli testata sitä, erottaako tukivektorikone ylipäätään poikkeavuuksia normaalista liikenteestä nykyisellä esikäsittelyllä. Testidatan joukosta valittiin 620 riviä, jotka valittiin ainoastaan normaalin datan joukosta. Koska normaali data on melko yhteneväistä, tulokset olivat hyviä. Ne on kuvattu taulukossa 7.3. Kaikki hyökkäykset havaittiin ja väärien positiivisten määrä saatiin pidettyä niin pienenä, että sillä ei ole suurta merkitystä. Tämä oli kuitenkin odotettua puhtaan opetusdatan ansiosta.

Toisaalta oletus puhtaasta opetusdatasta ei pidä usein paikkaansa. Opetusdatan manuaalinen läpikäyminen ei ole realistista tällä toteutuksella, sillä lokirivit on jaettu resurssien mukaan tuhansiin eri tiedostoihin. Niinpä nämä tiedostot on voitava käydä läpi täysin automaattisesti. Tämän takia opetusdataksi valittiin 500 riviä

		Havaittu	
		Normaali	Poikkeava
Todellinen	Normaali	2996	3
	Poikkeava	0	1293

Taulukko 7.3: Tukivektorikoneen tulokset puhtaalla opetusdatalla.

satunnaisesti kaikkien rivien joukosta. Näin tehtiin sen takia, että yleensä normaalia liikennettä pitäisi olla datassa enemmän kuin poikkeavaa. Näiden rivien joukkoon sekoittui 156 hyökkäysriviä. Normaali ja poikkeava liikenne erottuvat kuitenkin niin hyvin, että tästä huolimatta tulokset olivat lähes yhtä hyviä. Ne on kuvattu taulukossa 7.4.

		Havaittu	
		Normaali	Poikkeava
Todellinen	Normaali	2997	2
	Poikkeava	3	1290

Taulukko 7.4: Tukivektorikoneen tulokset epäpuhtaalla opetusdatalla.

Lisäksi jälkimmäisistä tuloksista laskettiin yleisimmät metriikat menetelmän toiminnan arvioinnille. Ne on esitetty kaavoissa 7.7–7.12.

$$\textit{True positive rate (TPR)} = \frac{TP}{TP + FN} = \frac{1290}{1293} = 99.77\% \tag{7.7}$$

$$\textit{False positive rate (FPR)} = \frac{FP}{TN + FP} = \frac{2}{2999} = 0.07\% \tag{7.8}$$

$$\textit{True negative rate (TNR)} = \frac{TN}{TN + FP} = \frac{2997}{2999} = 99.93\% \tag{7.9}$$

$$\textit{False negative rate (FNR)} = \frac{FN}{TP + FN} = \frac{3}{1293} = 0.23\% \tag{7.10}$$

$$\textit{Tarkkuus (accuracy)} = \frac{TP + TN}{TP + TN + FP + FN} = \frac{1290 + 2997}{4292} = 99.88\% \tag{7.11}$$

$$\textit{Täsmällisyys (precision)} = \frac{TP}{TP + FP} = \frac{1290}{1293} = 99.77\% \tag{7.12}$$

7.3 Diffuusiokartta

Diffuusiokartoilla saaduista tuloksista tässä yhteydessä on myös tehty julkaisu otsikolla *Anomaly Detection from Network Logs using Diffusion Maps*. Tutkimus on julkaistu *EANN 2011* -konferenssissa [83].

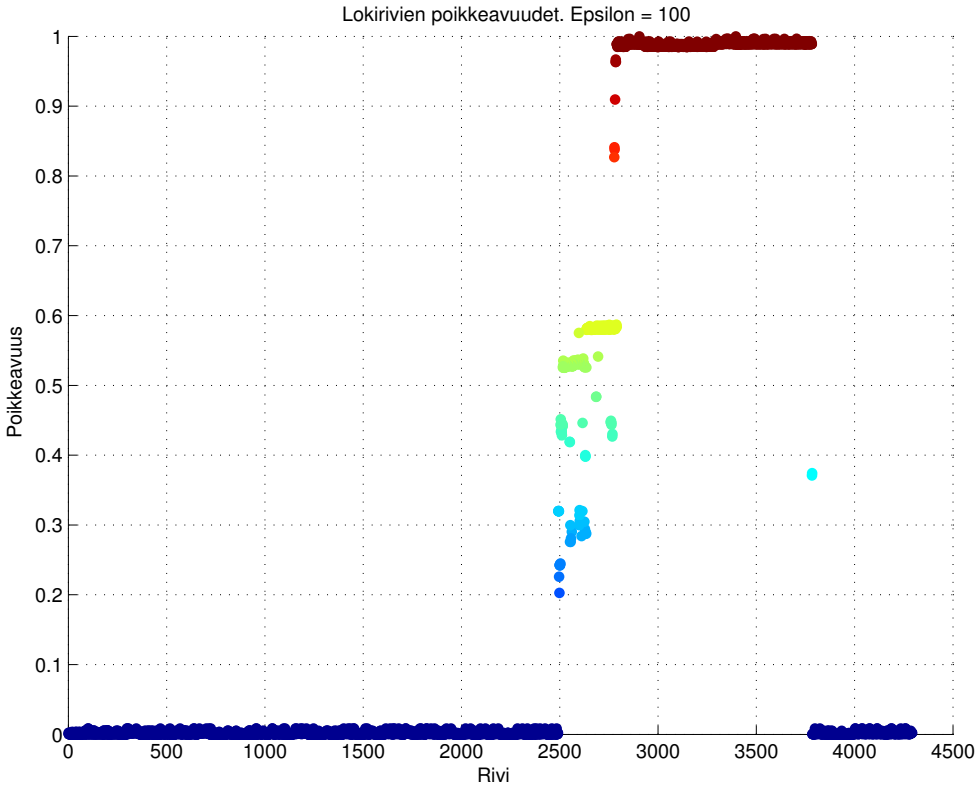
Tuomo Sipolan diffuusiokuvaussovellus soveltui hyvin testidatan analysointiin. Parhaaksi tulokseksi saatiin se, että kaikki hyökkäykset löytyivät ja yhtään väärää positiivista ei eksynyt joukkoon. Tulos oli lukujen valossa paras testatuista menetelmistä, mutta todennäköisesti myös muita menetelmiä kehittämällä oltaisiin päästy vieläkin parempaan tarkkuuteen. Hyökkäykset erottuivat testidatassa varsin selvästi ja todennäköisesti tärkeimmät piirteet erottuvat hyvin jo esikäsittelyvaiheessa. Koska kaikki hyökkäykset löytyivät ilman vääriä hälytyksiä, ei ole tarpeen piirtää erillisiä taulukoita tai laskea prosentteja tarkkuuksista. Sen sijaan käytetään useita kuvia esittelemään menetelmän toimintaa tarkemmin.

Kuvassa 7.1 on esitetty jokaisen lokirivin poikkeavuusluku. Se siis kuvaa sitä, kuinka poikkeavaksi rivi testidatan joukossa tulkitaan diffuusiokuvausmenetelmä perusteella. Kuvasta nähdään selvästi, että normaalit rivit ovat lähellä nollaa, kun taas useimmat hyökkäysrivit erottuvat todella selvästi. Osa poikkeavista riveistä on suhteellisen pieni poikkeavuusluku, mutta nekin erottuvat tarpeeksi selvästi jotta ne voidaan tulkita poikkeavuuksiksi.

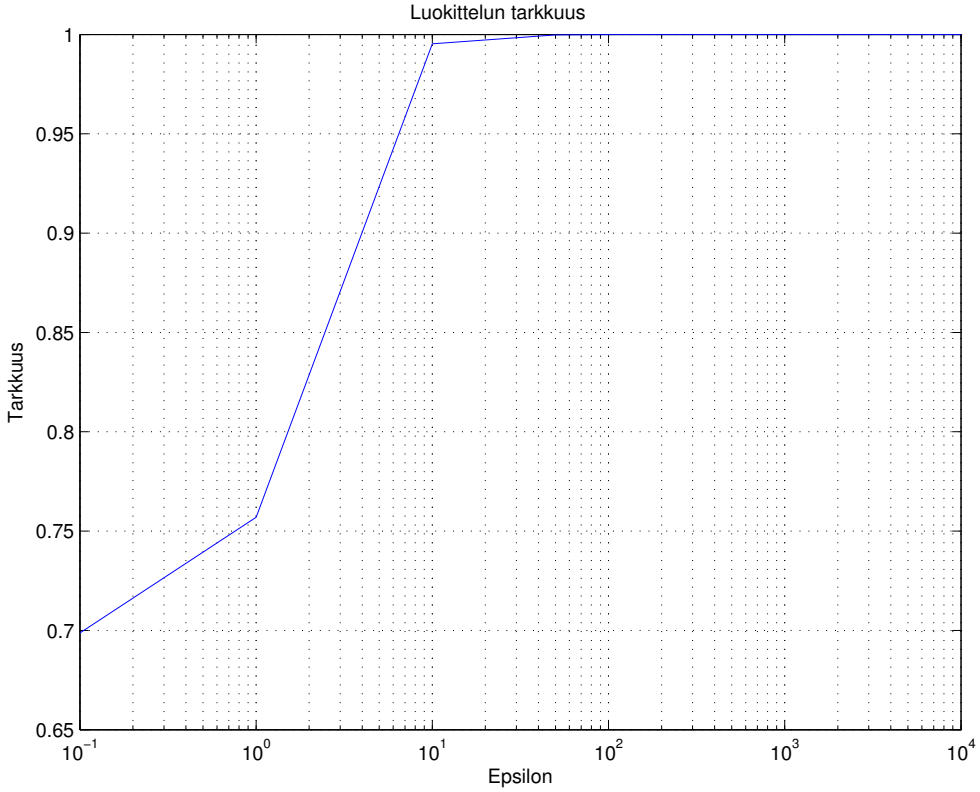
Menetelmän toimintaa voidaan vaikuttaa ϵ-parametrilla. Kuvassa 7.2 nähdään tarkkuus suhteutettuna parametrin arvoon. Arvolla 10^{-1} tarkkuus jäi 70 prosenttiin, mutta jo arvolla 10 päästiin selvästi yli yleensä vaaditun 95% tarkkuuden. Paras tulos löytyi parametrin arvolla 100, joka antoi siis tarkkuudeksi 100%. Parametria oltaisiin todennäköisesti voitu vielä kasvattaa tarkkuuden pysyessä edelleen samana, mikä on tärkeä havainto. Parametri voidaan ehkä kiinteästi asettaa melko suureen arvoon, jolloin tulokset ovat automaattisesti hyviä. Parametrin arvon muuttaminen manuaalisesti ei ole automaattisen järjestelmän toiminnassa tarkoituksenmukaista, vaan ihmisen toiminta on pystyttävä minimoimaan. Tämä vaatii kuitenkin lisätestejä.

Diffuusiokuvausmenetelmän periaatteena on siis tiivistää moniulotteisen datan piirteet siten, että ominaisuudet säilyvät vaikka ulottuvuuksien määrää pienennetään. Kuvassa 7.3 on diffuusiokartta, jossa näkyvät toisen ja kolmannen ominaisarvon mukaan piirretyt pisteet. Tässä näkyy selvästi, miten normaali liikenne asettuu lähes samaan pisteeseen. Ne on merkitty sinisellä värillä. Punaiset pisteet kuvaavat hyökkäysrivejä, jotka eivät asetu samaan pisteeseen yhtä tiheästi. Jokainen niistä on

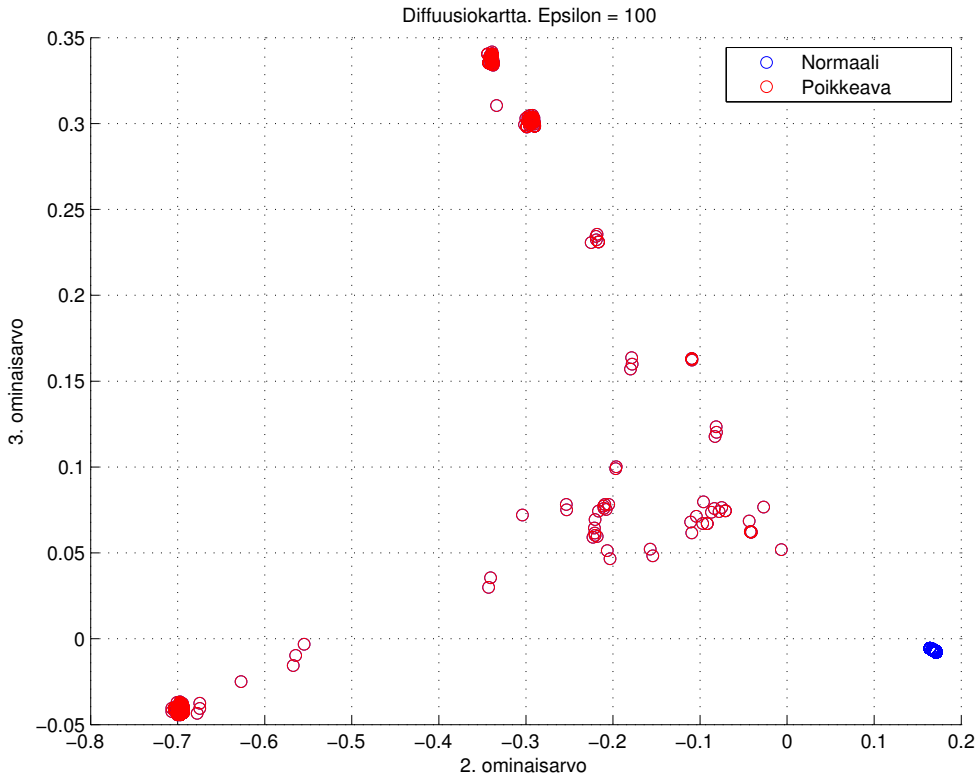
kuitenkin toisen ominaisarvon mukaan nollan alapuolella, joka jakaa testidatan selvästi ja yksikäsitteisesti kahtia. Lisäksi poikkeavuuksienkin joukossa on nähtävissä useampia tiheämpiä joukkoja, joihin on keskittynyt useampia lokirivejä. Niitä on kuitenkin enemmän kuin yksi, ja lisäksi monet hyökkäykset eivät ole kovin lähellä mitään olemassa olevaa suurempaa klusteria.



Kuva 7.1: Lokirivien poikkeavuusluvut.



Kuva 7.2: Diffuusiokuvauksen tarkkuus.



Kuva 7.3: Diffuusiokartta.

8 Johtopäätökset

N-grammianalyysin tuloksista voidaan päätellä ainakin se, että poikkeavan liikenteen erottuminen tapahtuu melko vahvasti jo esikäsittelyvaiheessa. Koska n-grammien pisteiden yhteenlasku ei sinänsä sisällä uutta älykkyyttä, täytyy esikäsittelyssä valittujen parametrien olla riittävät testidatan tapauksessa. Tulokset olivat sinänsä hieman heikommat kuin muilla testatuilla menetelmillä, mutta menetelmän yksinkertaisuus vähentää turhaa laskentaa, ja tässä tapauksessa tulokset olivat tarpeeksi hyviä eli hyökkäykset löydettiin suhteellisen tarkasti. Toisaalta on otettava huomioon, että hyökkäykset liittyvät vahvasti toisiinsa. Niinpä kun löydetään niistä edes osa, voidaan helposti löytää myös muut samantyyppiset hyökkäykset lokidatasta. Mikäli mukana olisi ollut täysin toisentyyppisiä hyökkäyksiä, on mahdollista että ne olisivat jääneet normaalin datan joukkoon ja näin niitä ei olisi löydetty. Poikkeavuusluvun raja-arvon määrittäminen on myös haastavaa, sillä mitään yksiselitteistä arvoa sille ei välttämättä ole. Testausvaiheessa oikea arvo kyseiselle datalla löydettiin kokeilemalla. Todellisessa tilanteessa jossa täytyy analysoida paljon suurempia määriä tiedostoja ja dataa, ei tämän raja-arvon määrittäminen käsin ole ehkä mahdollista tai ainakaan helppoa. Niinpä jonkinlainen menetelmä arvon määrittämiseksi automaattisesti tarvitaan. Esimerkiksi jos hyökkäysrivejä olisi ollut vain muutamia kappaleita, todennäköisesti raja-arvon asettaminen keskiarvoon antaisi liian paljon vääriä positiivisia tunnistuksia. Suuremmalla arvolla voitaisiin saada tarkempia tuloksia ja silti löytää kaikki poikkeamat. Toisin sanoen käyttökelpoinen ja tarkkoja tuloksia antava raja-arvo riippuu paljon datan ominaisuuksista.

Tukivektorikone suoriutui analyysistä erittäin hyvin huolimatta siitä, että opetusdata ei ollut puhdasta. Tässä tapauksessa hyökkäysten erottuminen n-grammianalyysissä oli tarpeeksi selvää, ja normaali ja poikkeava liikenne eivät menneet sekaisin. Mikäli mukana olisi ollut erittäin paljon normaalia liikennettä muistuttavia hyökkäyksiä, on mahdollista että niiden joutuminen opetusdatan joukkoon olisi tehnyt analyysista huomattavasti vaikeampaa. Suurin ongelma tukivektorikoneen käyttämisessä on sama kuin n-grammianalyysissakin. Kernelifunktion parametrien C ja γ määrittäminen ei ole yksiselitteistä, vaikka aputyökaluja siihen löytyykin. Erityisesti se, että parametreja on kaksi kappaletta vaikeuttaa parhaiden arvojen löytä-

mistä. Myös tukivektorikoneen tapauksessa lopulta parhaat arvot löytyivät kokeilemalla. Arvot riippuvat todennäköisesti paljolti testattavana olevan datan ominaisuuksista. Lisäongelmia koituu siitä, että tukivektorikone vaatii opetusdatana normaalia liikennettä. Näin parametrien määrittäminen joudutaan tekemään opetusdatan perusteella, koska periaatteessa mitään tietoa testidatasta ei vielä siinä vaiheessa ole. Jos varsinainen testidata on ominaisuuksiltaan erilaista, voivat nämä parametrit olla varsin huonot. Tällöin parhaiden parametrien määrittäminen jää testausvaiheeseen, kuten tämänkin tutkimuksen tapauksessa. Tämä ei ole toivottavaa, vaan kaikki menetelmän toimivuuteen tapahtuvat asetukset ja valinnat pitäisi tapahtua jo opetusvaiheen aikana. Näin ohjelma voidaan jättää toimimaan testausvaiheessa automaattisesti. Ongelmaa ei välttämättä esiinny datassa, joka on ominaisuuksiltaan yhteneväistä. Käytössä ollut testidata on kuitenkin todella vaihtelevaa, sillä se sisältää lokirivejä monista eri verkkopalveluista. Niinpä esikäsittelyssä tapahtuvan klusteroinnin seurauksena saadaan suuri joukko tiedostoja, joiden välillä ei välttämättä ole juurikaan riippuvuutta. Niinpä nykyisellä esikäsittelyllä varsinaisen analyysivaiheen olisi tapahduttava automaattisesti. Yksi tärkeimmistä vaiheista jo ennen opetusvaihetta on skaalaus. Jos opetusdata on ominaisuuksiltaan hyvin erilaista kuin testidata, myös skaalaus opetusdatan mukaan voi aiheuttaa huonoja tuloksia. Tukivektorikone toimii opetusvaiheen jälkeen itsenäisesti ja nopeasti, joten mikäli opetusvaiheessa menetelmä saadaan opetettua tarpeeksi hyvin, siinä on paljon potentiaalia myös reaaliaikaiseksi järjestelmäksi. Sinänsä sen jälkeen kun opetusvaihe on suoritettu ei ole väliä tulevatko datapisteet analysoitaviksi yksittäin reaaliaikaisesti vai yhtenä massana jo kerätyn lokidatan muodossa. Niinpä ongelmat ja haasteet liittyvät lähinnä skaalaukseen ja opetusvaiheeseen.

Diffuusiokarttamenetelmä suoriutui testatuista menetelmistä parhaiten. Osasyynä voi tosin olla itse implementoija Tuomo Sipolan osallistuminen analyysin toteuttamiseen. Todennäköisesti vielä paremmilla parametrivalinnoilla myös tukivektorikone olisi päässyt samaan tulokseen. Diffusiokuvausmenetelmän etuna on se, että muutettavia parametreja on ainoastaan yksi kappale, ϵ. Tätä voidaan ajatella rajaarvona avain kuten n-grammianalyysinkin tapauksessa. Koska menetelmä pienentää datan dimensioita, suurikaan määrä sarakkeita ei pitäisi hidastaa analyysia merkittävästi. Tämä on suuri etu ja ehdoton vaatimus, koska varsinkin suuremmissa verkkopalveluissa datan määrä on valtava ja se kasvaa koko ajan. Kuten kuvasta 7.2 nähdään, suuremmilla parametrin ϵ arvoilla tulokset olivat todella hyviä. Kaikilla testatuilla arvoilla päästiin vähintään kohtuulliseen tunnistamistarkkuuteen, vaik-

kakin yleensä rajana hyvälle tulokselle pidetään 95 prosenttia. Testauksessa kuitenkin huomattiin se, että parametrin muuttaminen ei vaikuttanut tulokseen yhtä merkittävästi kuin muilla testatuilla menetelmillä. Niinpä oikean parametrin valitseminen oli helpompaa ja nopeampaa ja sitä kautta myös parametrin arvon valinnan automatisoiminen on helpompaa. Muut menetelmät olivat erittäin tarkkoja niille annettujen parametrien arvoista ja pienetkin muutokset saattoivat huonontaa tulosta merkittävästi. Lisäksi diffuusiokuvaus toimii täysin ohjaamattomasti eli mitään erillistä opetusdataa ei tarvita. Tämä on ennalta tuntemattomassa datassa tärkeä asia.

Tutkimuksessa kävi ilmi, että alkuperäisellä datalla ja erityisesti esikäsittelyllä on suurin vaikutus tuloksiin. Mikäli piirteet on valittu oikein ja esikäsittely toimii tarpeeksi hyvin, pystytään käyttämään useita eri menetelmiä ja saamaan niillä kaikilla hyviä tuloksia. Tämän hetkinen esikäsittely toimii sen perusteella, minkälaista dataa on ollut saatavilla. Useissa muissa tutkimuksissa datan piirteitä on ollut käytettävissä enemmänkin. Kuitenkin jo tästä rajallisesta määrästä informaatiota pystyttiin saamaan esiin datan ominaisuuksia suhteellisen hyvin. Eri kerroksilla tapahtuvat hyökkäykset jäävät totta kai löytymättä. Periaatteena on kuitenkin se, että matemaattiset analysointimenetelmät toimivat esikäsittelyn toiminnasta riippumatta. Toisin sanoen esikäsittely, jossa muutetaan data numeeriseen matriisimuotoon, on oma itsenäinen vaiheensa samoin kuin varsinainen analyysivaihekin. Esikäsittelyä voidaan tarvittaessa muuttaa, mutta tiedostoja pitäisi silti pystyä tutkimaan matemaattisesti. Toisaalta myös poikkeavuuksien analysointimenetelmä voidaan vaihtaa esimerkiksi neuroverkosta tukivektorikoneeseen muuttamatta esikäsittelyä. Todennäköisesti mikäli piirteitä ei esikäsittelyn jälkeen saada esiin, mikään analysointimenetelmä ei pysty havaitsemaan poikkeavuuksia kovin luotettavasti.

9 Yhteenveto

Web-palveluiden ja -sovellusten muuttuessa monimutkaisemmiksi myös niiden tietoturvan varmistaminen ja ylläpitäminen muuttuu haasteellisemmaksi. Palvelut ovat yhä helpommin suuren käyttäjäjoukon saatavilla, joten mahdollisia hyökkääjiä ja hyökkäystyyppejä on yhä enemmän. Monet hyökkäykset osaavat naamioitua normaaliksi liikenteeksi, ja toisaalta monia hyökkäyksiä ei osata vielä edes tunnistaa koska niitä ei ole ennen havaittu. Pelkkä palveluiden tarkka ohjelmoiminen ja tietoturva-aukkojen poistaminen ei riitä, sillä aina jää haavoittuvuuksia joita kukaan ei ole osannut ennakoida. Tietoturva ei voi aina olla jäljessä hyökkääjiä, vaan on keksittävä uusia lähestymistapoja joilla voidaan havaita hyökkäykset paremmin. Hyökkääjän tunnistamiseen käytetyt järjestelmät tarvitsevat parempia menetelmiä, joilla myös uudet ja tuntemattomat hyökkäykset saadaan joko estettyä tai ainakin havaittua. Poikkeavuuksien tunnistamisessa voidaan siis soveltaa tekoälyä ja koneoppimista. Näin järjestelmä sopeutuu dynaamisemmin uusiin uhkiin ja samalla pysytään samalla tasolla hyökkääjien kanssa. Tavoitteena on, että järjestelmä osaa havaita piilevät hyökkäykset jo paljon ennen kuin niistä on mitään varsinaista ennakkotietoa. Näin järjestelmistä saadaan turvallisempia ja tietoturvaa saadaan automatisoitua.

Tutkimuksen tarkoituksena oli löytää palvelinlokitiedostoista poikkeavaa tietoliikennettä ja sitä kautta hyökkäyksiä ja muuta haitallista liikennettä. Käytetystä testidatasta löydettiin suurin osa hyökkäyksistä kaikilla testatuilla menetelmillä, joten tutkimustavoitteet täytettiin. Samalla huomattiin se, että esikäsittelyllä on ehkä tärkein merkitys poikkeavuuksien löytämisessä. Jos piirteet erottuvat hyvin jo esikäsitellyssä datassa, on helppoa analysoida sitä erilaisilla menetelmillä.

Edelleen haasteena on järjestelmän muuttaminen reaaliaikaiseksi tai ainakin nopeammaksi. Tällä hetkellä pystytään lähinnä havaitsemaan jo tapahtuneet hyökkäykset. Tätäkin tietoa voidaan käyttää näiden havaittujen hyökkäyksien löytämisessä jatkossa, mutta olisi tärkeää havaita uudet hyökkäykset mahdollisimman nopeasti, mieluiten reaaliajassa kun ne ovat tapahtumassa. Jos tähän tavoitteeseen päästään, on mahdollista myös sisällyttää järjestelmään jonkinlaista hyökkäystä estävää toiminnallisuutta jolla voidaan pysäyttää hyökkäys ja rajoittaa sen tekemiä

vahinkoja. Tätä ennen on kuitenkin valittava hyvä analysointimenetelmä ja kehitettävä esikäsittelijästä reaaliaikainen versio. Siihenkin on vielä matkaa, ja olemassa olevat ja taatusti toimivat reaaliaikajärjestelmät ovat jääneet lähinnä sääntöpohjaisiksi suodatinohjelmiksi, jotka osaavat havaita vain tarkasti määriteltyjä ja tunnettuja hyökkäyksiä.

Jatkotutkimuksen osalta jäi vielä paljon muutakin tehtävää. Itse lokitustasoa voitaisiin muuttaa tarkemmaksi. Näin saataisiin enemmän tietoa ja siten tarkempia tuloksia. Myös uudenlaisia hyökkäyksiä voitaisiin tunnistaa, koska osa hyökkäyksistä ei erotu HTTP-pyynnöstä. Niinpä olisi tärkeää saada tietoa esimerkiksi yksittäisen käyttäjän sessiotiedoista. Näin yhdestä sessiosta voidaan esimerkiksi koostaa yksi datarivi ja siten ottaa huomioon käyttäjän kaikki toimet yhtenä kokonaisuutena. Mikäli näin yritettäisiin toteuttaa hyökkäyksiä, jotka vaativat useita itsessään sinänsä normaaleja toimintoja, näiden toimintojen yhdessä muodostamia hyökkäyksiä pystyttäisiin havaitsemaan. Lisäksi tietoa voitaisiin kerätä eri kerroksilta, ja näin saada täysin erilaista informaatiota ja havaita uusia hyökkäyksiä. Tällä hetkellä lokiformaatti on melko rajoittunut, joten suuri osa hyökkäyksistä voi jäädä jo senkin takia havaitsematta. Datan määrä kuitenkin kasvaa, mikäli lokitustasoa nostetaan tarkemmaksi. Itse kerätty data on tunkeilijan havaitsemiseen liittyvän ketjun ensimmäinen lenkki, ja jos tietoa ei pystytä heti keräämään tarpeeksi, ei hyökkäyksiä pystytä havaitsemaan vaikka esikäsittely ja analysointi hoidettaisiin hyvin. Niinpä tärkeimmät jatkokehitysajatukset liittyvätkin juuri kerättyyn dataan ja sen formaattiin sekä esikäsittelyyn. Mikäli poikkeavat piirteet erottuvat esikäsittelyn jälkeen, ne pystytään kyllä havaitsemaan useillakin analysointimenetelmillä varsin helposti.

Tässä tutkimuksessa diffuusiokarttamenetelmää käytettiin lähinnä jo olemassa olevan lokidatan klusterointiin, mutta sitä voitaisiin laajentaa oppivaksi reaaliaikaiseksi menetelmäksi, joka pystyy myös analysoimaan uusia datapisteitä ja tulkitsemaan ne heti joko normaaliksi tai poikkeavaksi liikenteeksi. Tämän onkin yksi seuraavista kehityshaasteista, sillä pelkkä lokianalysaattori ei riitä, vaan tarvitaan myös reaaliaikaista poikkeavuuksien tunnistamista. Diffuusiokartat toimivat kuitenkin erittäin lupaavasti ja menetelmä tarjoaa monia mahdollisuuksia jatkokehitykseen. Lisäksi erittäin suurena etuna on se, että jatkotutkimus voidaan suorittaa yhteistyössä Tuomo Sipolan kanssa, joten myös muutoksia itse algoritmiin on helpompaa toteuttaa.

Esikäsittelijä ei myöskään toimi reaaliaikaisesti, joten sen suunnitteleminen uusiksi on ajankohtaista. Sinänsä olisi mahdollista pienin muutoksin muokata esikäsit-

telijä sellaiseksi, että sille voidaan syöttää yksittäisiä pisteitä ja ne voitaisiin lisätä jo olemassa oleviin datamatriiseihin. Tässä tapauksessa tarvitaan tietoa siitä, mikä n-vastaa mitäkin saraketta eri tiedostoissa. Näin voidaan aluksi tutkia jokaisen jo aineistossa esiintyneen n-grammin esiintymistiheydet ja lisätä ne olemassa oleviin sarakkeisiin. Mikäli lokirivissä on aikaisemmin esiintymättömiä n-grammeja, niille voidaan lisätä omat sarakkeet tiedostoon. Jos samalla on tarkoitus käyttää reaaliaikaista poikkeavuuksien tunnistamisjärjestelmää, tämä aiheuttaa myös ongelmia, koska tiedostojen sarakemäärät eivät pysy enää vakiona. Jo nyt esikäsittely tuottaa eri kokoisia tiedostoja, mutta reaaliaikaiseksi muuttamisen jälkeen jokaisen tiedoston sarakemäärä alkaisi kasvaa datan lisääntyessä. Tällaisen datan analysointi ei ole yksiselitteistä, joten todennäköisesti esikäsittely on reaaliaikaista toteutusta varten mietittävä uudelleen. Yksi vaihtoehto on pitää esikäsittelijän toimintaperiaate samana, mutta varata jokaiseen tiedostoon kiinteästi sarake jokaiselle n-grammille. Näin tiedostoista tulee kuitenkin erittäin suuria, sillä sarakkeita pitäisi olla kiinteästi 2-grammien tapauksessa 256^2 kappaletta.

Tutkimus antoi jonkinlaista kuvaa siitä, miten poikkeavuuksia voidaan löytää palvelindatan joukosta. Kuten aina, kysymyksiä ja ongelmia nousee esiin ehkä enemmän kuin vastauksia. Yllä kuvatut jatkotutkimusta koskevat ajatukset ja ongelmat takaavat sen, että työtä tulee aihealueen parissa riittämään vielä pitkään. Mikäli tutkimus aiheen parissa etenee hyvin, saatetaan jonain päivänä nähdä tehokas ja reaaliaikainen analysointijärjestelmä, joka tunnistaa uudet ja tunnetut hyökkäykset verkkoliikenteestä nopeasti ja tarkasti aiheuttamatta haittaa verkolle tai liikaa vääriä hälytyksiä. Koneoppimisen tuominen mukaan alalle tulee viemään monimutkaisten ja avointen web-palveluiden tietoturvan aivan uudelle tasolle. Se onkin lähes välttämätöntä, sillä pelkällä manuaalisella analysoinnilla ja tarkalla ohjelmoinnilla ei enää saada toteutettua tarpeeksi turvallisia järjestelmiä.

Lähteet

[1] C. Kruegel, F. Valeur, and G. Vigna, *Intrusion detection and correlation: challenges and solutions*. Springer Verlag, 2005.

[2] D. Gollmann, "Computer security," *Wiley Interdisciplinary Reviews: Computational Statistics*, 2006.

[3] R. Slade, *Dictionary of Information Security*. Syngress Media Inc, 2006.

[4] M. Asaka, A. Taguchi, and S. Goto, "The Implementation of IDA: An Intrusion Detection Agent System," *Proceedings of the 11th Annual FIRST Conference on Computer Security Incident Handling and Response (FIRST'99)*, 1999.

[5] V. Engen, *Machine learning for network based intrusion detection*. PhD thesis, Bournemouth University, 2010.

[6] 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.

[7] K. Kendall, "A Database of Computer Attacks for the Evaluation of Intrusion Detection Systems," Master's thesis, Massachusetts Institute of Technology, 1999.

[8] "Social engineering reloaded." `http://www.symantec.com/connect/articles/social-engineering-reloaded`. Viitattu 04.03.2011.

[9] T. Jagatic, N. Johnson, M. Jakobsson, and F. Menczer, "Social phishing," *Communications of the ACM*, vol. 50, no. 10, pp. 94–100, 2007.

[10] "Krp: Kalastelijoiden nordea-potti kohoaa 60 000 euroon." `http://www.digitoday.fi/page.php?page_id=66&news_id=200518646`. Viitattu 04.03.2011.

[11] K. G. McClure Stuart, Scambray Joel, *Hacking Exposed (5th Ed.)*. McGraw-Hill Professional Publishing, 2005.

[12] K. Czosseck, C. Geers, *Virtual Battlefield: Perspectives on Cyber Warfare*. IOS Press, 2009.

[13] Y. Dai, X. Zou, and Y. Pan, *Trust and Security in Collaborative Computing Vol. 2*. World Scientific, 2008.

[14] J. Mirkovic and P. Reiher, "A taxonomy of DDoS attack and DDoS defense mechanisms," *ACM SIGCOMM Computer Communication Review*, vol. 34, no. 2, pp. 39–53, 2004.

[15] Z. Bankovic, S. Bojanic, O. Nieto-Taladriz, and A. Badii, "Increasing Detection Rate of User-to-Root Attacks Using Genetic Algorithms," in *Emerging Security Information, Systems, and Technologies, 2007. SecureWare 2007. The International Conference on*, pp. 48–53, IEEE, 2007.

[16] A. Patcha and J. Park, "An overview of anomaly detection techniques: Existing solutions and latest technological trends," *Computer Networks*, vol. 51, no. 12, pp. 3448–3470, 2007.

[17] R. Kemmerer and G. Vigna, "Intrusion detection: a brief history and overview," *Computer*, vol. 35, no. 4, pp. 27–30, 2002.

[18] D. Denning, "An intrusion-detection model," *Software Engineering, IEEE Transactions on*, vol. SE-13, no. 2, pp. 222 – 232, 1987.

[19] J. Ryan, M. Lin, and R. Miikkulainen, "Intrusion detection with neural networks," in *Advances in neural information processing systems*, pp. 943–949, Morgan Kaufmann Publishers, 1998.

[20] S. Axelsson, "Research in Intrusion-Detection Systems A Survey," 1998. Technical report, University of Technology, Göteborg.

[21] S. Smaha, "Haystack: An intrusion detection system," in *Fourth Aerospace Computer Security Applications Conference*, vol. 44, 1988.

[22] M. Sebring, E. Shellhouse, and E. Mary, "Expert systems in intrusion detection: A case study," in *11th National Computer Security Conference: proceedings, 17-20 October, 1988*, p. 74, National Bureau of Standards, National Computer Security Center, 1988.

[23] P. Porras and P. Neumann, "EMERALD: Event monitoring enabling responses to anomalous live disturbances," in *Proceedings of the 20th National Information Systems Security Conference*, pp. 353–365, Citeseer, 1997.

[24] T. Verwoerd and R. Hunt, "Intrusion detection techniques and approaches," *Computer communications*, vol. 25, no. 15, pp. 1356–1365, 2002.

[25] A. Labbi, *Handbook of Integrated Risk Management for E-Business : Measuring, Modeling, and Managing Risk*. J. Ross Publishing, Incorporated, 2003.

[26] H. Debar, M. Dacier, and A. Wespi, "Towards a taxonomy of intrusion-detection systems," *Computer Networks*, vol. 31, no. 8, pp. 805–822, 1999.

[27] P. Porras and A. Valdes, "Live traffic analysis of TCP/IP gateways," in *NDSS*, 1998.

[28] S. Kumar and E. Spafford, "A pattern matching model for misuse intrusion detection," in *Proceedings of the 17th national computer security conference*, vol. 10, p. 11, Baltimore, MD, USA: NIST National Institute of Standards and Technology/National Computer Security Center, 1994.

[29] A. Lazarevic, L. Ertoz, V. Kumar, A. Ozgur, and J. Srivastava, "A comparative study of anomaly detection schemes in network intrusion detection," in *Proceedings of the Third SIAM International Conference on Data Mining*, pp. 25–36, 2003.

[30] K. Scarfone and P. Mell, "Guide to intrusion detection and prevention systems (idps)," *NIST Special Publication*, vol. 800, p. 94, 2007.

[31] N. Ye, *Secure Computer and Network Systems: Modeling, Analysis and Design*. Wiley Publishing, 2008.

[32] "Snort." `http://www.snort.org/`. Viitattu 07.02.2011.

[33] M. Roesch *et al.*, "Snort-lightweight intrusion detection for networks," in *Proceedings of the 13th USENIX conference on System administration*, pp. 229–238, Seattle, Washington, 1999.

[34] J. Beale, A. Baker, J. Esler, T. Kohlenberg, and S. Northcutt, *Snort: IDS and IPS toolkit*. Syngress Media Inc, 2007.

[35] V. Chandola, A. Banerjee, and V. Kumar, "Anomaly detection: A survey," *ACM Computing Surveys (CSUR)*, vol. 41, no. 3, pp. 1–58, 2009.

[36] N. Hubballi, S. Biswas, and S. Nandi, "Layered Higher Order N-grams for Hardening Payload Based Anomaly Intrusion Detection," in *Availability, Reliability, and Security, 2010. ARES'10 International Conference on*, pp. 321–326, IEEE, 2010.

[37] Q. Tran, H. Duan, and X. Li, "One-class support vector machine for anomaly network traffic detection," *China Education and Research Network (CERNET), Tsinghua University, Main Building*, vol. 310, 2004.

[38] O. Depren, M. Topallar, E. Anarim, and M. Ciliz, "An intelligent intrusion detection system (IDS) for anomaly and misuse detection in computer networks," *Expert systems with Applications*, vol. 29, no. 4, pp. 713–722, 2005.

[39] K. Hwang, M. Cai, Y. Chen, and M. Qin, "Hybrid intrusion detection with weighted signature generation over anomalous Internet episodes," *IEEE Transactions on Dependable and Secure Computing*, pp. 41–55, 2007.

[40] M. Rash, *Intrusion prevention and active response: deploying network and host IPS*. Syngress Media Inc, 2005.

[41] "Snortsam - A Firewall Blocking Agent for Snort." `http://www.snortsam.net/`. Viitattu 10.03.2011.

[42] "fwsnort: Application Layer IDS/IPS with iptables." `http://www.cipherdyne.org/fwsnort/`. Viitattu 10.03.2011.

[43] C. Cortes and V. Vapnik, "Support-vector networks," *Machine learning*, vol. 20, no. 3, pp. 273–297, 1995.

[44] C. Hsu, C. Chang, C. Lin, *et al.*, "A practical guide to support vector classification," 2003.

[45] B. Schölkopf, J. Platt, J. Shawe-Taylor, A. Smola, and R. Williamson, "Estimating the support of a high-dimensional distribution," *Neural computation*, vol. 13, no. 7, pp. 1443–1471, 2001.

[46] S. Mukkamala, G. Janoski, and A. Sung, "Intrusion detection using neural networks and support vector machines," in *Proceedings of IEEE international joint conference on neural networks*, vol. 1702, 2002.

[47] "Darpa intrusion detection data sets." http://www.ll.mit.edu/mission/communications/ist/corpora/ideval/data/index.html. Viitattu 18.03.2011.

[48] "Tcpstat." http://www.frenchfries.net/paul/tcpstat/. Viitattu 18.03.2011.

[49] J. Freeman and D. Skapura, *Neural networks: algorithms, applications, and programming techniques*. Addison Wesley Longman Publishing Co., Inc. Redwood City, CA, USA, 1991.

[50] M. Arbib, *The handbook of brain theory and neural networks*. The MIT Press, 2003.

[51] T. Kohonen, "The self-organizing map," *Proceedings of the IEEE*, vol. 78, no. 9, pp. 1464–1480, 2002.

[52] "Bibliography of som papers." http://www.cis.hut.fi/research/som-bibl/. Viitattu 28.03.2011.

[53] A. Höglund, K. Hätönen, and A. Sorvari, "A computer host-based user anomaly detection system using the self-organizing map," *Neural Networks, IEEE - INNS - ENNS International Joint Conference on*, vol. 5, p. 5411, 2000.

[54] M. Ramadas, S. Ostermann, and B. Tjaden, "Detecting anomalous network traffic with self-organizing maps," in *Recent Advances in Intrusion Detection*, pp. 36–54, Springer, 2003.

[55] F. González and D. Dasgupta, "Anomaly detection using real-valued negative selection," *Genetic Programming and Evolvable Machines*, vol. 4, no. 4, pp. 383–403, 2003.

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

[57] B. Nadler, S. Lafon, R. Coifman, and I. Kevrekidis, "Diffusion maps - a probabilistic interpretation for spectral embedding and clustering algorithms," *Principal Manifolds for Data Visualization and Dimension Reduction*, pp. 238–260, 2007.

[58] M. Damashek, "Gauging similarity with n-grams: Language-independent categorization of text," *Science*, vol. 267, no. 5199, p. 843, 1995.

[59] G. David, *Anomaly Detection and Classification via Diffusion Processes in Hyper-Networks*. PhD thesis, Tel-Aviv University, 2009.

[60] R. Maxion and K. Tan, "Benchmarking anomaly-based detection systems," in *dsn*, p. 623, Published by the IEEE Computer Society, 2000.

[61] K. Wang and S. Stolfo, "Anomalous payload-based network intrusion detection," in *Recent Advances in Intrusion Detection*, pp. 203–222, Springer, 2004.

[62] "Codered ii." `http://www.symantec.com/security_response/writeup.jsp?docid=2001-080421-3353-99`. Viitattu 04.04.2011.

[63] K. Wang, J. Parekh, and S. Stolfo, "Anagram: A content anomaly detector resistant to mimicry attack," in *Recent Advances in Intrusion Detection*, pp. 226–248, Springer, 2006.

[64] B. Bloom, "Space/time trade-offs in hash coding with allowable errors," *Communications of the ACM*, vol. 13, no. 7, pp. 422–426, 1970.

[65] W. Wang and R. Battiti, "Identifying intrusions in computer networks with principal component analysis," 2006.

[66] M. Shyu, S. Chen, K. Sarinnapakorn, and L. Chang, "A novel anomaly detection scheme based on principal component classifier," in *Proceedings of the IEEE foundations and new directions of data mining workshop*, Citeseer, 2003.

[67] "Kdd cup 1999 data." `http://kdd.ics.uci.edu/databases/kddcup99/kddcup99.html`. Viitattu 03.05.2011.

[68] Y. Bouzida, F. Cuppens, N. Cuppens-Boulahia, and S. Gombault, "Efficient intrusion detection using principal component analysis," in *Proceedings of the 3ème Conférence sur la Sécurité et Architectures Réseaux (SAR), Orlando, FL, USA*, Citeseer, 2004.

[69] D. Goldberg, *Genetic algorithms in search, optimization, and machine learning*. Addison-wesley, 1989.

[70] W. Li, "Using genetic algorithm for network intrusion detection," in *Proceedings of the United States Department of Energy Cyber Security Group 2004 Training Conference, Kansas City, Kansas*, pp. 24–27, Citeseer, 2004.

[71] J. Kim, P. Bentley, U. Aickelin, J. Greensmith, G. Tedesco, and J. Twycross, "Immune system approaches to intrusion detection–a review," *Natural computing*, vol. 6, no. 4, pp. 413–466, 2007.

[72] S. Forrest, A. Perelson, L. Allen, and R. Cherukuri, "Self-nonself discrimination in a computer," in *Research in Security and Privacy, 1994. Proceedings., 1994 IEEE Computer Society Symposium on*, pp. 202–212, IEEE, 1994.

[73] P. D'haeseleer, "An immunological approach to change detection: Theoretical results," in *csfw*, p. 18, Published by the IEEE Computer Society, 1996.

[74] D. Heckerman, "A tutorial on learning with Bayesian networks," *Innovations in Bayesian Networks*, pp. 33–82, 2008.

[75] C. Kruegel, D. Mutz, W. Robertson, and F. Valeur, "Bayesian event classification for intrusion detection," 2003.

[76] N. Ye, M. Xu, and S. Emran, "Probabilistic networks with undirected links for anomaly detection," in *IEEE Systems, Man, and Cybernetics Information Assurance and Security Workshop*, pp. 175–179, 2000.

[77] J. McHugh, "Testing intrusion detection systems: A critique of the 1998 and 1999 DARPA intrusion detection system evaluations as performed by Lincoln Laboratory," *ACM Transactions on Information and System Security*, vol. 3, no. 4, pp. 262–294, 2000.

[78] M. Mahoney and P. Chan, "An analysis of the 1999 DARPA/Lincoln Laboratory evaluation data for network anomaly detection," in *Recent Advances in Intrusion Detection*, pp. 220–237, Springer, 2003.

[79] "Log files - apache http server." `http://httpd.apache.org/docs/current/logs.html`. Viitattu 06.04.2011.

[80] "Hypertext transfer protocol – http/1.1 / request." `http://www.w3.org/Protocols/rfc2616/rfc2616-sec5.html`. Viitattu 06.04.2011.

[81] J. Lehtonen and K. Siljander, "Web-palvelut ja niihin kohdistuvan poikkeavan tietoliikenteen analyysi käyttäen diffuusiokuvausmenetelmää ," Master's thesis, Jyväskylän yliopisto, 2010.

[82] C.-C. Chang and C.-J. Lin, "Libsvm – a library for support vector machines." http://www.csie.ntu.edu.tw/~cjlin/libsvm/. Viitattu 12.04.2011.

[83] T. Sipola, A. Juvonen, and J. Lehtonen, "Anomaly detection from network logs using diffusion maps," in *Engineering Applications of Neural Networks*, vol. 363 of *IFIP Advances in Information and Communication Technology*, pp. 172–181, Springer Boston, 2011.