TAMPEREEN TEKNILLINEN YLIOPISTO
TAMPERE UNIVERSITY OF TECHNOLOGY

JARI KUISTI

HÄMÄÄNNYTETYN JA HAITALLISEN TIETOLIIKENTEEN HAVAINNOINTI AVOIMEN LÄHDEKOODIN OHJELMISTOILLA

Diplomityö

Tarkastaja: professori Jarmo Harju
Tarkastaja ja aihe hyväksytty
Tieto- ja sähkötekniikan tiedekunta-neuvoston kokouksessa 14. tammikuuta 2015

TIIVISTELMÄ

JARI KUISTI: Hämäännytetyn ja haitallisen tietoliikenteen havainnointi avoimen lähdekoodin ohjelmistoilla
Tampereen teknillinen yliopisto
Diplomityö, 120 sivua, 18 liitesivua
toukokuu 2016
Tietotekniikan diplomi-insinöörin tutkinto-ohjelma
Pääaine: Communication systems and networks
Tarkastaja: professori Jarmo Harju

Avainsanat: tietoliikenteen hämäännyttäminen, pakettien syvätarkastus, haitallinen tietoliikenne, liikenteen luokittelu

Hämäännytetyllä tietoliikenteellä tarkoitetaan tietoliikenneprotokollan toiminnan tarkoituksenmukaista monimutkaistamista. Tavoitteena on hämäännyttää liikennettä tutkiva verkkosensori, jonka tarkoituksena on tunnistaa liikennöivä sovellus, eli suorittaa liikenteen luokitus. Verkkosensorit perustuvat tavallisesti pakettien syvätarkastukseen ja liikenteen tilastolliseen analyysiin. Syvätarkastuksen avulla tutkitaan IP-paketin hyötykuormaa. Tilastollinen analyysi pyrkii tutkimaan liikenteelle ominaisia piirteitä, kuten pakettikokoa tai pakettien välistä saapumisaikaa. Tässä diplomityössä tutkitaan hämäännytetyn ja haitallisen verkkoliikenteen tunnistamista.

Tämän diplomityön keskeisimmät tutkimuskysymykset ovat: Kykenevätkö avoimen lähdekoodin pakettien syvätarkastusohjelmistot luokittelemaan hämäännytettyä liikennettä? Kuinka hämäännytettyä liikennettä voidaan havaita? Kuinka muodostetaan liikennettä, jota on vaikea havaita?

Tutkimus suoritettiin sitä varten kehitetyssä suljetussa testausympäristössä, jossa generoitiin hämäännytettyä liikennettä sekä avoimen lähdekoodin hämäännyttämisohjelmistojen avulla, että haitallisten takaporttiohjelmistojen avulla. Haitallisista takaporttiohjelmistoista laadittiin liikenneanalyysi eri ilmaisohjelmistojen avulla. Testausympäristössä otettiin käyttöön kolme eri avoimen lähdekoodin pakettien syvätarkastusohjelmistoa, joiden avulla testattiin generoidun liikenteen luokittelua. Hämäännyttämis- ja takaporttiohjelmistojen muodostamasta liikenteestä laadittiin myös tilastollinen analyysi.

Tutkimuksen keskeisimpinä tuloksina havaittiin, että oletusasetuksilla DPI-kirjastot kykenevät luokittelemaan hämäännytettyä liikennettä. Kuitenkin pääosa hämäännytetystä liikenteestä luokiteltiin tuntematon-luokkaan, kuten alkuperäinen oletus olikin. Vääriä positiivisia luokituksia syntyi eniten säännöllisiin lausekkeisiin perustuvassa DPI-kirjastossa. Työn tuloksissa esitetään, kuinka osalle tuntemattomaksi luokitellusta liikenteestä voidaan kehittää omat protokolladekooderinsa. Selkeimmät tapaukset ovat ne, joiden hyötykuormasta voidaan lukea selväkielisiä merkkijonoja tai yhteydenmuodostuskättely on tunnistettavissa. Osaan esitetyistä hämäännytysmenetelmistä ei voida soveltaa pakettien syvätarkastusta, koska liikenne on salakirjoitettua ja yhteydenmuodostuskättelystä on vaikea havaita tunnistettavia piirteitä. Näihin menetelmiin voidaan hyödyntää tilastollista analyysiä. Esimerkiksi jaetun salaisuuden avulla salatun takaporttiohjelmiston liikenne on mahdollista tunnistaa pakettikokojakauman avulla. Tilastollisen analyysin tuloksia ei voida kuitenkaan yleistää kaikkiin menetelmiin, sillä hämäännytettävällä sovelluksella on mahdollisesti vaikutusta edellä esitettyihin eri jakaumiin.

ABSTRACT

JARI KUISTI: Detecting obfuscated and malicious network traffic using open source software
Tampere University of Technology
Master of Science Thesis, 120 pages, 18 Appendix pages
May 2016
Master's Degree Programme in Information Technology
Major: Communication systems and networks
Examiner: Professor Jarmo Harju

Keywords: traffic obfuscation, deep packet inspection, traffic classification, protocol obfuscation

Traffic obfuscation means obscuring the IP packet payload or traffic flow by removing easily identifiable properties of protocols. There are also obfuscation methods which mimic other applications. The best obfuscation methods combine several techniques. The easily identified properties are for example deterministic byte sequences and packet sizes. In traffic obfuscation, the aim is simply to confuse the network traffic sensor which tries to identify the running application. Often, these sensors are based on deep packet inspection and statistical analysis of the properties of network traffic. Deep packet inspection (DPI) inpects the IP packet payload. Statistical analysis aims to study the characteristic features of traffic, such as packet size and packet inter-arrival times (IAT). In both methods, the aim is to identify the application and classify the traffic. This thesis discusses the classification of obfuscated and malicious traffic.

In this thesis the main research questions were: Can open source DPI software classify obfuscated traffic? How to detect obfuscated traffic? How to form traffic, which is difficult to detect?

This research was carried out in a closed test environment, which was implemented especially for this study. The traffic used in this study was generated using open source obfuscation software and malicious backdoor software. The thesis presents behavioral analysis on one malicious backdoor software and traffic analysis of chosen obfuscation protocols. In the closed test environment, three open source DPI software were tested against the generated packet capture files. Statistical analysis was performed for chosen obfuscation protocols and malicious backdoor software.

The main findings in this study were that DPI libraries are able to classify obfuscated traffic. However, the bulk of the obfuscated traffic was classified as unknown. The DPI library which was based on regular expressions produced the most false positive classifications. With analysis of the obfuscation and backdoor sofware, we can present methods to write our own protocol dissectors for certain obfuscation applications and backdoor software. Clearest cases are the ones in which the payload can be read in plaintext or the handshake has certain identifiable features. DPI can not be applied for some of the presented obfuscation methods, because the traffic is encrypted and the handshake has no identifiable metrics due to uniform Diffie-Hellman key exchange. Statistical analysis is the only way to classify these applications. However, the results of statistical analysis can not be generalized, because the obfuscated application has potential impact for the packet size and IAT distributions.

ALKUSANAT

Tätä diplomityötä laatiessa kyberturvallisuus on noussut Suomessa merkittäväksi uutisoinnin aiheeksi. Kyberturvallisuudesta keskustellaan useissa eri medioissa ja tietoliikennetiedustelua koskevaa lakia valmistellaan. Edward Snowdenin paljastukset NSA:n toiminnasta käynnistivät kyberuutisoinnin aallon ja se tuntuu jatkuvan edelleen.

"Näkymätön" tai vaikeasti haivaittava tietoliikenne on muuttuvassa Internet-verkossa monia kiinnostava aihe. Viestinnän yksityisyydestä on käyty julkista keskustelua ja sen saavuttamiseksi pyritäänkin salakirjoittamaan liikennettä. Salakirjoituksen määrä on selvästi kasvanut Internetissä Snowdenin paljastuksien jälkeen. Salakirjoitus ei kuitenkaan aina riitä, jos salakirjoitettu liikenne halutaan jostain syystä estää. Tällöin ainoaksi keinoksi jää liikenteen hämäännyttäminen, jotta sitä ei havaittaisi lainkaan. Toistaiseksi liikenteen hämäännyttämiskeinot eivät ole kuitenkaan suuren yleisön tiedossa, mutta todennäköisesti näiden käyttö tulee jatkossa lisääntymään.

Turvallisuutta edistävät toiminnot voivat myös heikentää sitä. Liikenteen hämäännyttäminen mahdollistaa esimerkiksi verkkorikollisille tavan kätkeä omaa liikennettä. Tämän on yksi hyvä syy, miksi hämäännytettyä liikennettä tulisi kyetä tunnistamaan.

Edellä esitetyt seikat herättivät kiinnostuksen tämän diplomityön laatimiseksi. Työn edetessä kiinnostus syveni entisestään uusien havaintojen syntyessä. Suurimmat haasteet työn laadinnassa liittyivät ajankäytöön. Haluan kiittää erityisesti perhettäni Katria, Joonaa ja Miinaa jaksamisesta ja tuesta, jonka olen saanut heiltä koko opiskelujeni ajan. DI-opintojen suorittaminen Jyväskylästä käsin on vaatinut muun muassa vuokra-asunnon hankkimisen Tampereelta ja pitkiä poissaoloja kotoa. Esitän myös kiitokseni äidilleni Anna-Liisalle ja siskolleni Minnalle, jotka ovat tukeneet opiskeluani.

Lisäksi haluan kiittää Mikko Kunttua, Juhani Mesiäistä, Mikko Ojalaa ja Mikko Asikaista tuesta ja keskuteluista, joita olemme käyneet diplomityön aiheesta. Lopuksi kiitän työn tarkastajaa professori Jarmo Harjua tuesta ja ohjauksesta.

Tampereella, 10.4.2016

Jari Kuisti

SISÄLLYSLUETTELO

KUVALUETTELO

LYHENTEET JA MERKINNÄT

AES-CTR	engl. Advanced Encryption Standard, lohkosalausmenetelmä, jota voidaan käyttää mm. engl. Counter (CTR) -, eli laskurimoodissa.
APT	engl. Advanced Persistent Threat, kohdistettu hyökkäys, jossa operaation toimenpiteet on kohteen mukaan räätälöityjä.
BPF	engl. BSD Packet Filter, tarjoaa jalostamattoman rajapinnan OSI-mallin siirtokerrokselle, joka tukee pakettisuodatusta.
CDF	engl. Cumulative distribution function, eli kertymäfunktio on reaaliarvoisen satunnaismuuttujan todennäköisyyden jakautumista kuvaava funktio.
CPU	engl. Central Processing Unit, eli suoritin. Tietokoneen osa, joka suorittaa tietokoneohjelman sisältämiä konekielisiä käskyjä.
DCERPC	engl. Distributed Computing Environment/Remote Procedure Calls, etäproseduurikutsu. Tarkoittaa tietokoneohjelman tuottamaa menettelyä, jossa aliohjelma suoritetaan jaetun verkon jonkin toisen tietokoneen osoiteavaruudessa. Kutsu on koodattu ikään kuin se olisi normaali paikallinen proseduurikutsu. DCE-standardi on RPC:hen perustuva hajautusmenetelmien kokoelma.
D-H	Diffie-Hellman-avaimenvaihtoprotokolla. D-H:n avulla kaksi osapuolta voi sopia yhteisestä salaisuudesta turvattoman tietoliikenneyhteyden ylitse.
DHT	engl. Distributed Hash Table, hajautettu tiivistetaulu.
DNS	engl. Domain Name System, Internetin nimipalvelujärjestelmä.
DPI	engl. Deep Packet Inspection, pakettien syvätarkastus. Syvätarkastuksessa tutkitaan pakettien hyötykuormaa.
FTP	engl. File Transfer Protocol, TCP-protokollaa hyödyntävä tiedostonsiirtomenetelmä kahden tietokoneen välillä.
HTTP	engl. Hypertext Transfer Protocol, hypertekstin siirtoprotokolla.
HTTPS	(engl. Hypertext Transfer Protocol Secure) on HTTP-protokollan ja TLS/SSL-protokollan yhdistelmä jonka avulla HTTP-tiedonsiirto voidaan salakirjoittaa.
IAT	engl. packet interarrival time, pakettien välinen saapumisaika. Esim. asiakkaalta palvelimelle tai palvelimelta asiakkaalle.
IDPS	engl. Intrusion Detection and Prevention System, tunkeutumisen havaitsemis- ja -estojärjestelmä.
IDS	engl. Intrusion Detection System, tunketumisen havaitsemisjärjestelmä. Vanhentunut termi, sillä useat nykyiset järjestelmät kykenevät aktiivisesti puuttumaan haitalliseen liikenteeseen ja estämään sitä.
IMAP	engl. Internet Message Access Protocol, sähköpostien lukemiseen tarkoitettu protokolla, säilyttää viestit sähköpostipalvelimella.
JSON	engl. JavaScript Object Notation, yksinkertainen avoimen standardin tiedostomuoto tiedonvälitykseen.
KDD	engl. Knowledge Discovery in Databases, datan louhintaprosessi.
LVM	Liikenne- ja viestintäministeriö.

MAC	engl. Message Authentication Code, viestin todentamiseen käytettävä menetelmä, jossa tiivistefunktion ja salaisen avaimen kanssa muodostetaan tarkistuskenttä.
MLA	engl. Machine Learning Algorithm, koneoppimisalgoritmi.
5-tuple	monikkoavain, joka muodostuu viidestä muuttujasta: lähteen- ja kohteen IP-osoite, kuljetuskerroksen protokolla, lähde- ja kohdeporttinumero.
MPAA	engl. Motion Picture Association of America, elokuvateollisuuden etua ajava järjestö, joka edustaa kaikkia suuria yhdysvaltalaisia elokuvastudioita.
MPLS	engl. Multiprotocol Label Switching, menetelmä, jolla kuljetetaan esimerkiksi IP-paketteja runkoverkon solmujen kautta hyödyntämällä reitityksessä etikettejä (engl. label).
MSE	engl. Message Stream Encryption, bittorrent-protokollalle kehitetty hämäännytysmenetelmä.
MSN	engl. Microsoft Notification Protocol, Microsoftin kehittämä pikaviestintäprotokolla Microsoft Messenger service -palvelulle.
NetFlow	NetFlow on ominaisuus, joka alunperin esiteltiin Cisco Systemsin reitittimissä. NetFlown avulla voidaan kerätä IP-verkon liikennettä kun se saapuu tai lähtee reitittimen liitäntään. NetFlow-data sisältää ainoastaan paketin otsikkotiedot.
OSI-malli	engl. Open Systems Interconnection Reference Model, kuvaa tiedonsiirtoprotokollien yhdistelmän seitsemässä kerroksessa.
PCAP	engl. packet capture, libpcap-kirjaston tiedostoformaatti verkkoliikenteen kaappaukselle.
P2P-liikenne	engl. Peer to Peer, vertaisverkkoliikenne.
RAT	engl. Remote Access Trojan. Takaporttiohjelmisto, jonka avulla saadaan kohdejärjestelmän täydellinen hallinta tietoliikenneyhteyden ylitse.
SCP	engl. Secure Copy Protocol, perustuu SSH-protokollaan. SCP:n avulla voidaan siirtää salakirjoitettuna tiedostoja paikallisen- ja etätietokoneen välillä.
SHA	engl. Secure Hash Algorithm, kryptograafinen tiivistefunktio, yhdensuuntainen kuvaus, joka laskee sanomasta tarkistuskentän.
SMB	engl. Server Message Block, verkkoprotokolla, jota tyypillisesti käytetään tiedostojen jakamiseen Microsoft Windows -käyttöjärjestelmää käyttävien tietokoneiden välillä.
SMTP	engl. Simple Mail Transfer Protocol, TCP-pohjainen protokolla, jota käytetään viestien välittämiseen sähköpostipalvelimien kesken.
SNMP	engl. Simple Network Management Protocol, TCP/IP-verkkojen hallinnassa käytettävä tietoliikenneprotokolla.
SPI	engl. shallow packet inspection, matala pakettitarkastus, joka käyttää hyväkseen vain paketin otsikkotietoja.
SPID	engl. Statistical Protocol IDentification, liikenteen luokitteluun kehitetty koneoppimisalgoritmi.
SSH	engl. Secure Shell, salakirjoitettuun tietoliikenteeseen tarkoitettu protokolla, joka toimii TCP/IP-mallin siirto- ja sovelluskerroksen välissä.

SSL	engl. Secure Sockets Layer, salausprotokolla, jolla voidaan suojata Internet-sovellusten tietoliikenne IP-verkkojen yli, toimii TCP/IP-mallin siirto- ja sovelluskerroksen välissä.
TCP	engl. Transmission Control Protocol, kuljetuskerroksen tietoliikenneprotokolla, jolla luodaan luotettavia yhteyksiä tietokoneiden välille. Sisältää mm. vuonohjauksen- ja ruuhkanhallinnan algoritmit.
TLS	Transport Layer Security (TLS), aiemmin tunnettu nimellä Secure Sockets Layer (SSL). Liikenteen salakirjoitusprotokolla, joka tarjoaa viestinnän luottamuksellisuuden tietoverkon ylitse.
UDP	engl. User Datagram Protocol, siirtokerroksen yhteydetön protokolla, joka mahdollistaa informaation siirron, mutta ei varmista pakettien perille pääsyä.
URL	engl. Uniform Resource Locator, verkkosivun osoite.
VLAN	engl. Virtual LAN, eli virtuaalilähiverkko. Tekniikka, jossa fyysinen verkkosegmentti voidaan jakaa ethernet-kehyksessä olevan VLAN ID -arvon perusteella loogisiin osiin.
VoIP	engl. Voice over Internet Protocol. Tekniikka, jonka avulla ääntä voidaan siirtää reaaliaikaisesti IP-protokollaa käyttävän verkon välityksellä.
VPN	engl. Virtual Private Network, virtuaalinen erillisverkko. Tekniikan avulla voidaan muodostaa julkisen verkon ylitse ohjelmallisesti yksityinen verkko kahden tai useamman osapuolen välille.

1. JOHDANTO

"Sitä mitä et havaitse, et voi myöskään hakkeroida". Tämä on erään yhdysvaltalaisen yrityksen lennokas markkinointilause tietoliikenteen hämäännyttämistuotteelle. Tietoliikenne on varsin laaja käsite. Tässä sillä tarkoitetaan IP-protokollan päällä toimivia sovelluksia ja näiden sovellusten synnyttämää verkkoliikennettä. Hämäännytyksen tavoitteena on kätkeä liikenne siten, että sitä ei kyetä luokittelemaan esimerkiksi passiivisten verkkosensoreiden avulla.

Jos tietoliikenne voidaan "kätkeä" muun liikenteen sekaan tai se saadaan näkymättömäksi, on näillä menetelmillä varmasti potentiaalisia käyttäjiä monien eri motiivien vuoksi. Tietoturvallisuuden näkökulmasta esimerkiksi verkkorikolliset pyrkivät hyödyntämään hämäännyttämismenetelmiä erilaisten haittaohjelmien muodostamien komentokanavien kätkemiseksi. Toisaalta, tästä johtuen hämäännytetyn tietoliikenteen havaitsemiselle on myös selkeä tarve.

Tässä diplomityössä tutkitaan hämäännytetyn ja haitallisen tietoliikenteen havaitsemista avoimen lähdekoodin ohjelmistoilla. Tutkimuksen avulla pyritään vastaamaan kysymyksiin: kykenevätkö avoimen lähdekoodin pakettien syvätarkastusohjelmistot luokittelemaan hämäännytettyä liikennettä, kuinka hämäännytettyä liikennettä voidaan havaita ja kuinka muodostetaan hämäännytettyä liikennettä, jota on vaikea havaita. Tarkoituksena oli myös selvittää, kuinka paljon liikennettä luokitellaan väärin ja voidaanko tunkeutumisenestojärjestelmien sääntöjä hyödyntää tehokkaasti hämäännytetyn liikenteen havainnointiin.

Jotta liikenteen luokittelua tai havaitsemisen tarkkuuta voidaan testata, tarvitaan joukko laadukasta testausaineistoa. Hämäännytetyn liikenteen testausaineistoa on heikosti jaossa, joten testausaineiston laatimista varten generoitiin hämäännytettyä ja haitallista tietoliikennettä suljetussa testausympäristössä. Liikenteen generoinnissa hyödynnetiin avoimen lähdekoodin hämäännytysohjelmistoja sekä tunnettuja takaporttiohjelmistoja. Hämäännyttämisohjelmistoina käytettiin muun muassa obfsproxy- ja fteproxy-ohjelmistoja. Näistä ensiksi mainittu tarjoaa useita eri hämäännytysprotokollia. Takaporttiohjelmistoiksi valittiin monissa kohdistetuissa hyökkäyskampanjoissa käytetyt XtremeRAT- ja DarkComet RAT -ohjelmistot.

Lähtökohtainen oletus tässä diplomityössä muodostetulle hämäännytetylle liikenteelle oli se, että avoimen lähdekoodin pakettien syvätarkastusohjelmistot eivät kykene luokittelemaan hämäännytettyä sovellusliikennettä, olettaen, että varsinaista liikennöivää sovellusta ei tunnisteta, eikä hämäännytyksen suorittavaa protokollaa tunnisteta. Erityises-

ti yksi oleellinen kohde oli seurata väärien positiivisten luokitusten määrää, jolloin hämäännyttämisemenetelmä on onnistunut passiivista sensoria vastaan. "Tunnistamaton luokka" on syvätarkastusta suorittavan sensorin näkökulmasta parempi luokittelu hämäännytetylle liikenteelle.

Testausympäristössä muodostettu hämäännytetty ja haitallinen liikenne tallennettiin pakettikaappaustiedotoiksi. Pakettikaappaustiedostojen laadinnassa hyödynnettiin suodatusta, jotta voitiin varmistua, että tiedostot sisältävät vain tiettyä liikennettä. Liikenteen luokittelua testattiin ajamalla pakettikaappaustiedostot avoimen lähdekoodin pakettien syvätarkastusohjelmistojen läpi. Testattaviksi ohjelmistoiksi valittiin nDPI, libprotoident ja suricata.

Luvuissa 2-6 esitetään tutkimuksen kannalta tarvittava tausta. Luvuissa 2-3 käsitellään liikenteen luokittelun perusteita ja luokitteluun toteutettuja avoimen lähdekoodin ohjelmistoja. Luvussa 4-5 esitetään liikenteen hämäännyttämismenetelmien perusteita sekä liikenteen hämäännytystä varten toteutettuja avoimen lähdekoodin ohjelmistoja. Luvussa 6 keskitytään kohdistettujen hyökkäyksien toimintamalliin ja näissä hyödynnettäviin ohjelmistoihin. Teorian lisäksi kyseisessä luvussa esitetään molemmista RAT-ohjelmistoista luokittelua varten tarvittava liikenneanalyysi. Tähän sisältyy muun muassa DarkComet-ohjelmiston käyttäytymisanalyysi. Esitetyssä taustassa analysoidaan kirjallisuuden avulla hämäännytys- ja pakettien syvätarkastusohjelmistojen heikkouksia ja vahvuuksia. Analysoinnin avulla pyritään luomaan teoreettisia malleja tehokkaaseen hämäännytykseen ja hämäännytyksen havaitsemiseen.

Luvussa 7 käydään läpi aikaisemmin mainittu testausaineiston laadinta ja testausympäristön rakentaminen. Luku 8 keskittyy tulosten esittämiseen ja luku 9 on tutkimuksen yhteenveto.

2. PAKETTIEN SYVÄTARKASTUS JA LIIKENTEEN LUOKITTELU

Pakettien syvätarkastus (engl. Deep Packet Inspection, eli DPI) on menetelmä, jonka avulla voidaan tunnistaa liikennöivä sovellus ja sovelluskerroksen protokolla. Syvätarkastus eroaa perinteisestä pakettisuodatuksesta, siten että siinä tarkastellaan myös pakettien hyötykuormaa otsikkotietojen ja porttinumeroiden lisäksi. Syvätarkastuksella on monia tavoitteita, joita ovat esimerkiksi liikennöivän sovelluksen tunnistaminen sekä luokittelu, pakettien merkkaaminen ja merkkaamisen perusteella tunnistetun sovelluksen generoiman liikenteen muokkaaminen tai sen estäminen. (Corwin 2011.) Muokkaamisella tarkoitetaan esimerkiksi kyseessä olevan liikennetyypin rajoittamista kaistanleveyden suhteen tai sen prioriteetin heikentämistä reitittimien jonopuskureissa.

DPI-tutkimuksissa, joissa on tutkittu liikenteen tunnistamiskykyä, käytetään usein termiä liikenteen luokittelu (engl. traffic classification). Luokittelu oikeastaan sisältää edellä mainitun sovelluksen tuottaman liikenteen tunnistamisen ja sen nimeämisen. On myös erittäin olleellista todeta, että liikennettä voidaan luokitella muutenkin kuin pakettien syvätarkastuksen avulla.

Tietoliikenteen hämäännyttämisen tavoite on ohittaa tai kiertää liikenteen luokittelu. Jotta liikenteen hämäännyttämismenetelmiä voidaan suunnitella ja toteuttaa, tulee ymmärtää kuinka liikennettä luokitellaan. Tässä luvussa tarkastellaan liikenteen luokittelun periaatteita sekä tekniikoita. Tarkastelu keskittyy erityisesti pakettien syvätarkastuksen sekä tilastollisen analyysin perusteella tehtävään luokitteluun ja näiden taustalla oleviin tekniikoihin.

2.1 Syvätarkastuksen ja liikenteen luokittelun motiivit

Liikenteen luokittelua suorittavat muun muassa Internet-palveluntarjoajat eli tietoliikenneoperaattorit verkon kapasiteetin takaamiseksi. Tietynlaista liikennettä voidaan rajoittaa tai jopa estää. Esimerkiksi vertaisverkkoliikenne (P2P) on kasvanut viime vuosien aikana niin merkittävästi, että useat tietoliikenneoperaattorit rajoittavat sitä omissa verkoissaan (Singel 2007; Osborne 2014). P2P-verkoissa liikkuu laillisten tiedostojen lisäksi myös tunnetusti paljon tekijänoikeutta rikkovaa materiaalia. Tämän vuoksi esimerkiksi Motion Picture Association of America (MPAA), elokuvateollisuuden etua ajava järjestö, joka edustaa kaikkia suuria yhdysvaltalaisia elokuvastudioita ja International Federation of the Phonographic Industry, joka edustaa maailmanlaajuista levyteollisuutta ovat painostaneet tietoliikenneoperaattoreita rajoittamaan, seuraamaan ja es-

tämään vertaisverkkoliikennettä (Horten 2008; Kravets 2012). Yhdysvalloissa Internet-palveluntarjoajat ovat tehneet sopimuksen MPAA:n ja liittovaltion oman levy-yhtiöiden sekä ääniteteollisuuden etua ajavan järjestön kanssa (Recording Industry Association of America) tekijänoikeudellisen materiaalin jakamisen seurannasta. Seurannan avulla on muodostettu varoitusjärjestelmä, joka varoittaa asiakkaita, jos heidän liittymästään ladataan tai jaetaan tekijänoikeudellista digitaalista materiaalia. (Corwin 2011.) Samankaltainen järjestely on käytössä myös Iso-Britanniassa (Curtis 2014). Itse asiassa "tekijänoikeusteollisuus" on puhunut DPI-teknologian puolesta jo pitkään, jotta voidaan monitoroida tekijänoikeusloukkauksia (Horten 2008). Yhdysvalloissa on laadittu oma lakiesitys piratismin estämiseksi. Tämän tarkoituksena on saada viranomaisille lisää valtuuksia estää laittoman digitaalisen materiaalin jakaminen. Lakiesitys tunnetaan nimellä "Stop Online Piracy Act" (Pepitone 2012). Muun muassa näiden syiden vuoksi useimmat kaupalliset DPI-valmistajat tarjoavat tuotteissaan laillisen keskeytysrajapinnan ja datan takavarikoinnin mahdollisuuden. Kuitenkin vain harvat valmistajat haluavat julkisesti keskustella näistä ominaisuuksista, koska valvonnan vastustus ja kritiikki yksityisyyden menettämisestä on kasvanut. (Corwin 2011.) On selvää, että laillinen keskeytysrajapinta mahdollistaa myös muun rikollisen toiminnan tarkkailun IP-verkoissa, esimerkiksi Voice over Internet Protocol -puheluiden (VoIP) kuuntelu on mahdollista. Yhdysvalloissa vuonna 1994 laadittu telekommunikaation avunantolaki (engl. Communications Assistance for Law Enforcement Act) vaatii kaikilta Yhdysvaltojen tietoliikenneoperaattoreilta ja laitevalmistajilta, että järjestelmät suunnitellaan ja rakennetaan siten, että valtion virastoilla on mahdollisuus reaaliaikaiseen puheluiden, Internetin ja VoIP-liikenteen tarkailuun. (Corwin 2011.) Palveluntarjoajien liikenteen tarkkailuun on myös muita motiiveja. Heillä voi olla motiivina myös puhdas taloudellisen edun saavuttaminen mainonnan avulla, sillä DPI-teknologia mahdollistaa myös kohdistetun markkinoinnin (Corwin 2011). DPI:n avulla käyttäjien verkkokäyttäytymistä on mahdollista seurata.

Tietoliikenneoperaattoreiden lisäksi liikenteen luokittelua suorittavat eri valtiot sekä valtioiden tiedustelupalvelut. Valtiollisten toimijoiden motiivit liikenteen luokittelulle ovat luonnollisesti erilaiset kuin kaupallisilla tietoliikenneoperaattoreilla. Usein näille toimijoille ei riitä pelkkä liikenteen luokittelu, vaan myös pakettien hyötykuormaa halutaan analysoida. Tämän vuoksi käytössä on todennäköisemmin DPI-tekniikat, kuin liikenteen tilastollinen anayysi. On myös mahdollista, että molempia tekniikoita hyödynnetään samanaikaisesti. On esimerkiksi yleisesti tiedosssa, että luokittelun ja pakettien syvätarkastuksen avulla sananvapautta rajoittavat valtiot estävät kansalaisia pääsemästä tiettyihin Internet-palveluihin. Yleisesti myös tiedetään, että esimerkiksi Kiina sensuroi Internetiä DPI-tekniikoiden avulla (Horten 2008). Valtioiden tiedustelupalvelut haluavat taas suorittaa verkkotiedustelua terrorismin ja muiden valtiollisten uhkien havaitsemiseksi (Corwin 2011). Esimerkiksi Suomessa on juuri tätä kirjoittaessa laadittu tiedonhankintalakityöryhmän mietintö, jonka yhtenä keskeisenä sisältönä on verkkotiedusteluun liittyvän lainsäädännön muodostaminen (Nordström et al. 2015, 50). Sen johtopää-

töksissä esitetään, että Suomen tulisi harkita verkkotiedustelun käyttöönottoa (Kauhanen 2015). Mietinnön mukaan verkkotiedustelun toiminnan toteuttaminen edellyttää, että tiedusteluviranomaisella on pääsy tietoliikennekaapeleiden liityntäpisteisiin (Nordström et al. 2015, 50). Mietinnössä ei suoranaisesti esitetä tekniikoita, kuinka em. liityntäpisteistä tietoa kerätään, mutta siinä kuitenkin viitataan tietoliikennettä seuloviin automatisoituihin hakuehtoihin. Mietintöä laatinut työryhmä ei ollut yksimielinen. Liikenne- ja viestintäministeriö (LVM) jätti eriävän mielipiteen, jonka perusteluna on kotimaisen ja ulkomaisen tietoliikenteen erottelun vaikeus sekä tätä kautta puuttuminen kansalaisten perusoikeuksiin kuuluvaan yksityisyyden suojaan. LVM:n mukaan tiedustelun piirissä olisivat tahtomattaan kaikki Suomessa viestivät, myös viranomaiset, poliittiset toimijat, media ja yritykset. (Kauhanen 2015.)

Liikenteen luokittelua DPI:n avulla suoritetaan myös eri organisaatioiden ja yritysten verkoissa tietoturvallisuuden lisäämiseksi, sillä syvätarkastuksen avulla saadaan parempi näkyvyys omissa verkoissa liikkuviin tietoliikenneprotokolliin ja sovelluksiin. "Yritystasolla" DPI on pääosin tietoturvatyökalu, koska se mahdollistaa esimerkiksi tietoturvapoikkeamien havaitsemisen (Corwin 2011). DPI yhdistettynä palomuuriin sekä tunkeutumisenesto- ja havaitsemisjärjestelmään muodostaa verkon tietoturvan näkökulmasta tehokkaan ydistelmän.

Kuten edellä on osoitettu DPI-teknologia on laajasti käytössä eri ympäristöissä. Sandvine, joka on yksi DPI-laitteiston johtavista valmistajista, ilmoittaa asiakasmääräkseen yli 80 asiakasta yli 30:ssä eri maassa. Allot Communications arvioi, että yli 80% tason 1 operaattoreista käyttää verkoissaan DPI-tekniikkaa. (Corwin 2011.) Edellä on myös osoitettu, että pakettien syvätarkastus jakaa mielipiteitä ja se liitetään myös vahvasti yksityisyyden suojan loukkaamiseen. Usein DPI-teknologiaan kohdistuva kritiikki väittää sen olevan yksityisyyteen tunkeutumista (Corwin 2011). Horton (2008) kiteyttää asian seuraavasti: Me asetamme lain, joka hyväksyy ihmisten välisen yksityisen kommunikoinnin keskeyttämisen ja sisällön estämisen teollisuuden edun nimissä. Tässä otetaan kantaa itse asiassa vain Internet-palvelun tarjoajien DPI-toimintaan, mutta kuten Suomessakin juuri laadittu tiedonhankintalakityöryhmän mietintö verkkotiedustelusta osoittaa, myös valtioiden suorittama pakettien syvätarkastus aiheuttaa ihmisten keskuudessa eriäviä mielipiteitä sekä tunteita.

2.2 Liikenteen luokittelu ja sen perustekniikat

Kun Internet suunniteltiin, kaikki reitittimet ja muut verkon aktiivi- sekä turvalaitteet prosessoivat liikennettä pakettien otsikkotietojen mukaan. Tällöin esimerkiksi tietty Transmission Control Protocol - (TCP) tai User Datagram Protocol (UDP) -portti tarkoitti tiettyä sovellusta. Dynaamisten palveluiden ja dynaamisten porttinumeroiden jälkeen tuli tarve kehittää pakettien hyötykuorman analysointiin työkaluja, jotta pystyttiin havaitsemaan varsinainen sovellusprotokolla. (Danelutto et al. 2014, 92.)

Tarve sovellusprotokollan havaitsemiselle tuli monestakin syystä. Näitä syitä ja motiiveja on lueteltu edellä. Dynaamiset palvelut ja -porttinumerot olivat siis yksi ajava voima DPI-teknologian kehittämiseksi. Toinen merkittävä tekijä oli keino saada paremmin näkyvyyttä verkossa liikkuvaan dataan. Kuten jo luvun 2 esittelyssä todettiin, liikennettä voidaan kuitenkin luokitella muutoinkin kuin DPI-tekniikoiden avulla. Luokittelua voidaan toteuttaa myös liikenteen tilastollisen analyysin avulla, jonka etuna on se, että pakettien hyötykuormaa ei tarvitse käsitellä lainkaan. Tilastollinen analyysi perustuu tietyn sovelluksen tuottaman liikenteen tiettyihin tilastollisiin piirteisiin, kuten esimerkiksi pakettikokoon. Sovelluksen yhteyden muodostamiskättelyssä voidaan käyttää tietyn kokoisia paketteja tai pakettien välinen saapumisaika voi muodostaa tietynlaisen jakauman. Liikenteen luokittelu voidaan siis jakaa kahteen eri luokkaan: DPI ja liikenteen tilastollinen analyysi.

2.2.1 DPI-tekniikat

Liikenteen luokittelumallien tekninen toteutus voidaan edelleen jakaa omiin kategorioihin. Pakettien syvätarkastutekniikat, eli DPI-tekniikat voidaan perustekniikoiden osalta jakaa karkeasti kahteen eri kategoriaan:

- Protokolladekoodereita / analysointiliitännäisiä hyödyntävä tekniikka.
- Protokollalle ominaisten toistumakuvioiden etsintä. Tämä voidaan toteuttaa esimerkiksi säännöllisten lausekkeiden avulla, jolloin hyötykuormasta etsitään tiettyjä protokollalle tyypillisiä merkkijonoja (engl. regular-expression-based signature).

DPI-tekniikoista on hyvä huomioida, että esimerkiksi protokolladekooderit hyödyntävät omissa säännöissään myös protokollalle ominaisia staattisia piirteitä, eli osittain samoja menetelmiä, kuin liikenteen tilastollisessa analyysissä. Liikenteen luokittelu, joka perustuu täysin tilastolliseen analyysiin, on usein toteutettu teknisesti koneoppimisalgoritmien avulla. (engl. Machine Learning Algorithms eli MLA). Tästä on vielä hyvä terävöittää, että luokittelu voidaan toteuttaa tämän tekniikan avulla ilman pakettien hyötykuorman tarkastusta. Tilastollinen analyysi ei siis ole DPI-tekniikka.

Hyötykuorman analysointityökalujen kehittämiseen johtanut trendi edisti tunkeutumisen havaitsemis- ja -estojärjestelmien (IDPS-järjestelmien) esiintuloa. Näiden järjestelmien tarkoitus on tarkistaa paketit haitallisen liikenteen havaitsemiseksi. Liikennöivän sovelluksen tunnistaminen rikosten havaitsemiseksi sekä haitallisen liikenteen tunnistaminen ovat olleet DPI-teknologioiden kehittämisen ajavia voimia. (Danelutto et al. 2014, 92.)

DPI-tekniikoiden edeltäjänä on toiminut pakettisuodatus. Käytännössä suodatus voidaan jakaa kahteen eri tekniikkaan. Se voi tapahtua joko yksittäisten pakettien tasolla tai tietovirran eli liikennevuon tasolla. Pakettikohtaisessa eli tilattomassa suodatuksessa pudotetaan kaikki kielletyt paketit, kun taas tietovirtaan kohdistunut suodatus estää tai "ku-

ristaa" koko tietovirran, joka sisältää kiellettyjä paketteja. (Wiley 2014a.) Jälkimmäisessä tekniikassa seurataan verkkoistuntojen tilaa, eli TCP- tai UDP-istuntojen tilaa, jolloin molempiin suuntiin kulkevasta liikennevuosta eli istunnosta pidetään tilatietoa yllä. Tätä kutsutaan tilalliseksi pakettisuodatukseksi. Istuntotaulussa tyypillisesti pidetään yllä mm. lähdeosoitetta, lähdeporttia, kohdeosoitetta, kohdeporttia, tilaa ja aikaa. Kyseinen tietue luodaan, kun ensimmäinen paketti liikennevuosta havaitaan, mikäli tätä ei olla aikaisemmin jäljitetty. Muodostetun istunnon seuraavat paketit tarkistetaan istuntotaulua vasten, eikä sääntökantaa vasten kuten tilattomassa suodatuksessa. Tarkastus kohdistuu siis sarjaan paketteja, eikä yksittäisiin paketteihin. (Li et al. 2005.) Käytännössä tietovirtasuodattimille on tyypillistä, että näyte otetaan vain yhteyden käynnistävistä paketeista tai vaihtoehtoisesti otetaan tilastollisesti satunnaisia näytteitä paketeista. Kielletyt tietovirrat merkitään ja kieltomerkinnöistä pidetään yllä pysyvää tilatietoa. Yksinkertaistettuna suodattimen "kiertäminen" onnistuu siis lähettämällä paketteja, jotka eivät ole kiellettyjä. (Wiley 2014a.)

DPI-tekniikka periytyy tilallisesta pakettisuodatustekniikasta. Ne sisältävät paljon samoja toimenpiteitä, kuten esimerkiksi istuntojen havaitseminen ja niiden tilatiedon ylläpito. DPI-tekniikan tavoite on kuitenkin eri kuin perinteisessä pakettisuodatustekniikassa. Sen tavoitteena on tunnistaa liikennöivä sovellus tai sovellusprotokolla huolimatta käytettävästä siirtokerroksen porttinumerosta. Toisin sanoen tuottaa tarkasteltavasta liikenteestä luokitus. Jotta sovellusten tuottama liikennevuo voidaan identifioida, paketit luokitellaan joukoksi paketteja. Joukko muodostetaan paketeista, joilla on sama avain. Avain muodostuu paketin seuraavista attribuuteista: lähdeosoite, kohdeosoite, lähdeportti, kohdeportti ja kuljetuskerroksen protokolla, esimerkiksi TCP tai UDP. (Danelutto et al. 2014, 93.) Tätä kutsutaan myös nimellä "5-monikko" (engl. 5-tuple) ja tähän avaimeen on myös liikennevuon luokittelijat perinteisesti perustuneet (Rajahalme, Amante, Jiang & Carpenter 2011). Näitä yksittäisistä paketeista koostuvia joukkoja kutsutaan siis nimellä liikennevuo. Tyypillisesti liikennevuosta kertyvä tieto varastoidaan tiivistetauluun (engl. hash table). Käytännössä tämä mahdollistaa vuon uudelleen muodostamisen ja tunnistamisen. Danelutto et al. (2014, 93) esittävät DPI-järjestelmän pakettien käsittelyn sisältävän kuusi eri vaihetta. Esimerkiksi TCP-yhteyden prosessoinnissa vastaanotettaessa TCP-segmenttiä, joudutaan suorittamaan seuraavat toimenpiteet ja vaiheet:

- Vaihe 1: Pakettien otsikkotietojen avaus. Otsikkotietojen avulla saadaan purettua 5-monikko, joka kuvaa molempiin suuntiin kulkevaa liikennevuota. Paketit siis kuuluvat tiettyyn vuohon.
- Vaihe 2: Muodostetaan tiivistefunktion avulla 5-monikosta tiiviste. Tiivistettä tullaan vertaamaan tiivistetaulun alkiohin, jotta saadaan vastaanotettua pakettia vastaavan vuon tilatieto. Kun tiiviste tallennetaan tiivistetauluun, siihen liitetään myös vuon tilatieto.
- Vaihe 3: Haetaan taulu ja oikea alkio, jotta saavutetaan vuon nykyinen tila.

- Vaihe 4: Hallinnoidaan TCP-vuota. Jos segmentit ovat väärässä järjestyksessä, niin ne tallennetaan tulevaisuuden uudelleen koontia varten. Tämä toteutetaan tallentamalla pakettien hyötykuorma.
- Vaihe 5: Päätellään liikennöivä sovellusprotokolla segmentistä käyttämällä tiettyjä tarkistusmetodeja eli "tarkastajia". Jokaista tuettua protokollaa varten on omansa. Päättely tehdään käyttämällä aikaisemmin kerättyä informaatiota liikennevuosta ja analysoimalla nykyistä pakettia. Esimerkiksi Hypertext Transfer Protocol -liikenteen (HTTP) tunnistamiseen käytetään "HTTP-tarkastajaa" eli protokolladekooderia. Tämä voi etsiä paketin sisältä merkkijonoja, jotka edustavat HTTP-protokollan metodeja, kuten GET, PUT tai POST. Vakautta lisää, jos käytetään lisäksi kyseisen liikennevuon aikaisemmin vastaanotettua pakettia. Esimerkiksi voidaan tarkistaa HTTP-vastausta vastaava HTTP-pyyntö.
- Vaihe 6: Tunnistamisen jälkeen, pakettia voidaan jatkokäsitellä esimerkiksi poimimalla protokollaspesifistä metatietoa, kuten HTTP Uniform Resource Locator (URL), eli verkkosivun osoite ja käytetty verkkoselain.

Jos liikennevuon sovellusprotokolla on jo tunnistettu, niin vaiheet 4 ja 5 voidaan hypätä yli ja palata edelliseen tunnistukseen. (Danelutto et al. 2014, 93.)

Wiley (2014a) esittää saman asian hieman yleisemmällä tasolla. On olemassa kaksi yleistä luokittelutekniikkaa, joiden avulla voidaan päätellä onko paketti kielletty. Matala pakettitarkastus (engl. shallow packet inspection, eli SPI) käyttää hyväkseen vain paketin otsikkotietoja. (Wiley 2014a.) On hyvä huomioida, että lyhenne ja termi SPI on sama kuin tilallisen paketti suodatuksen (engl. stateful packet inspection, eli SPI). SPI-tekniikassa paketteja merkitään lähdeosoitteen, lähdeportin, kohdeosoitteen, kohdeportin ja pakettien pituuksien mukaan. Toinen luokittelutekniikka on edellä mainitun vastakohta eli pakettien syvätarkastus tai syvä pakettitarkastus eli DPI. Kuten nimi syvätarkastus viittaa, paketista on tarkoitus tarkistaa otsikkotietojen lisäksi muutakin. Tämä muu on paketin hyötykuorma. DPI-tekniikasta on ajateltiin aluksi, että se on resurssien näkökulmasta liian kallis ollakseen käytännöllinen. Nykyään se on kuitenkin laajasti käytössä. (Wiley 2014a.) Wileyn mukaan (2014a) DPI-tekniikoissa paketit käsitellään seuraavasti:

- tutkitaan paketin otsikkotiedot ja tutkitaan onko siinä jonkin ennalta tunnetun protokollan kättelydataa
- tutkitaan pakettien sisältö eli hyötykuorma kiellettyjen/etsittävien staattisten merkkijonojen osalta
- tutkitaan pakettien hyötykuormasta kiellettyjä/etsittäviä merkkijonomalleja.

Tunnistettaessa sovellusprotokollaa, kuten esimerkiksi HTTP-, SMTP- tai Skype-protokollaa, joudutaan usein tekemään yllä esitettyjä tarkistuksia pakettien hyötykuormaan. Protokollan sisältämän datan poiminnassa joudutaan suorittamaan suorituskyvyn näkökulmasta kalliita toimenpiteitä. Tyypillisesti tämän kaltainen pakettien prosessointi,

tai ainakin osa siitä, on implementoitu tähän toiminnallisuuteen varatulla laitteistolla. Toisaalta täysin ohjelmistopohjaiset ratkaisut ovat joustavampia ja taloudellisempia. (Danelutto et al. 2014, 93.) DPI-järjestelmien optimoinnista on tehty useita tutkimuksia, esimerkiksi Watson & Blox (2014) esittävät malleja, kuinka DPI-järjestelmän resurssien käytöstä saadaan joustavaa. Danelutto et al. (2014, 93) esittävät ohjelmistopohjaisen DPI-järjestelmän optimoimiseksi uutta kehysmallia, joka on nimeltään PEAFOWL. Tässä ideana on liikennevuon rinnakkainen prosessointi siten, että rinnakkaisille ohjelmistomoduuleille määritetään eri liikennevuoryhmiä, jotta ne voidaan käsitellä rinnakkain.

Edellä esitetyt DPI-tekniikat perustuvat luvun alussa esitettyihin protokolladekooderien avulla tehtävään analyysiin ja säännöllisten lausekkeiden merkkijonohavainnointiin. Danelutto et al. (2014) esittämässä esimerkissä on lisäksi metadatan kerääminen, jota useimmat kaupalliset DPI-järjestelmät nykyään hyödyntävät.

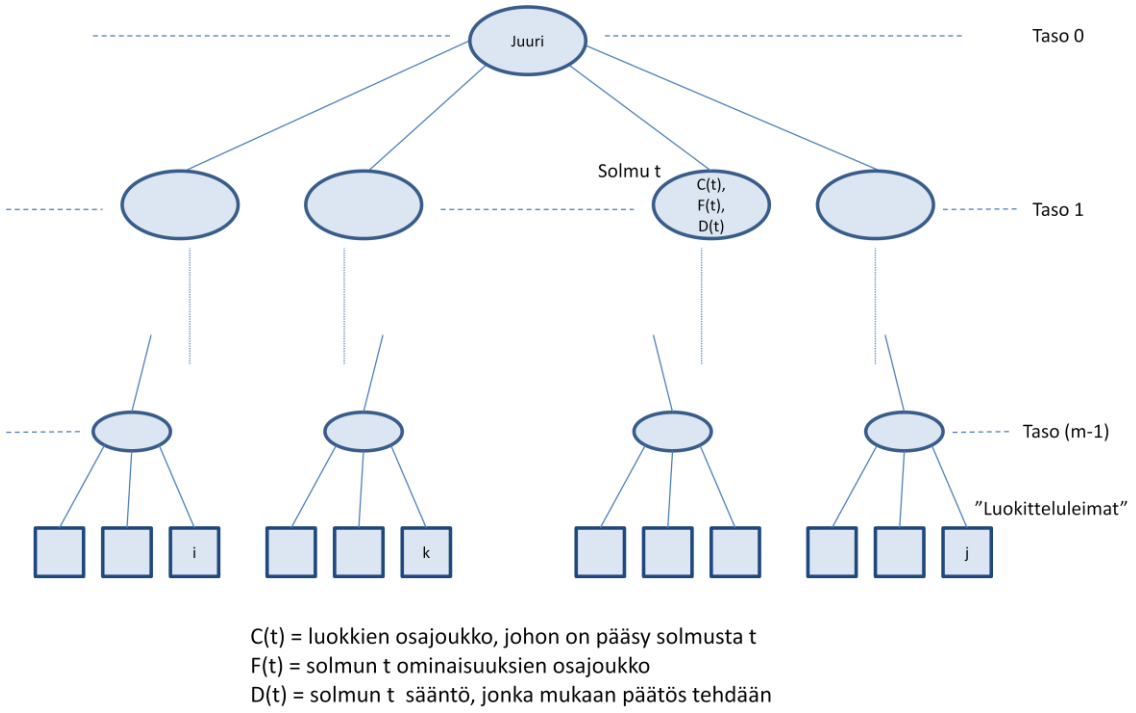
2.2.2 Tilastollisen analyysin hyödyntäminen liikenteen luokittelussa

Liikenteen tilastolliseen analyysiin perustuva luokittelu kehitettiin, koska paketin hyötykuorman analyysia edellä esitettyjen DPI-tekniikoiden avulla kritisoitiin (Deri, Martinelli, Bujlow & Cardigliano 2014). Pakettien syvätarkastusjärjestelmiä on kritisoitu lähinnä Internetin avoimuuden menettämisestä (Fuchs 2012, 18). Toinen merkittävä syy koneoppimismenetelmän kehittämiseksi oli protokolladekooderien käyttämä merkittävä laskentaresurssien käyttö tunnistustoimenpiteiden suorittamiseksi. Tilastollisessa koneoppimismenetelmässä sovellusprotokolla voidaan päätellä protokollan tilastollisten ominaisuuksien, kuten pakettikoon tilastollisen jakauman tai pakettien välisen saapumisajan perusteella. (Deri et al. 2014.) Tällöin ei tarvitse puuttua itse hyötykuormaan.

Luokittelussa käytetään koneoppimisalgoritmeja (engl. Machine Learning Algorithm, eli MLA), joita voidaan hyödyntää useissa eri tunnistamistoimenpiteissä. Käytännön esimerkkinä voidaan pitää monille yleisesti tunnettua digitaalikameroiden kasvojen- tai objektientunnistusmenetelmää (de Freitas 2013). Koneoppiminen tarkoittaa sitä, että kone oppii itse käyttämällä erilaisia tulosjoukkoja. Sen tavoitteena on suunnitella ja kehittää algoritmeja, jotka mahdollistavat järjestelmien hyödyntävän empiiristä dataa, kokemusta ja harjoitusdataa, joiden avulla järjestelmä pystyy itse muuntautumaan ja sopeuttamaan muutoksiin, jotka ilmenevät niiden ympäristössä. (Shubhangi, Sarika & Shital 2013.)

Kyseisten algoritmien käytöstä tietoliikenteen luokittelussa on tehty useita tutkimuksia (Jun, Shunyi, Yanqing & Zailong 2007; Hjelmvik & John 2010; Bujlow, Riaz & Pedersen 2012a; Shubhangi, Sarika, & Shital (2013). Esimerkiksi Jun et al. (2007) vertaili eri koneoppimisalgoritmeja keskenään liikenteen luokittelun täsmällisyyden, mallien muodostamisen ja harjoitusajan näkökulmasta. Näiden vertailukohteiden perusteella C4.5

päätöspuualgoritmi osoittautui tehokkaimmaksi. Päätöspuun avulla voidaan mallintaa peräkkäisiä päätösongelmia epätietoisuudessa. Sen avulla kuvataan tehtävät päätökset, mahdolliset tapahtumat ja lopputulokset, jotka seuraavat päätösten ja tapahtumien kombinaatioista. Tapahtumille määrätään todennäköisyydet ja jokaiselle lopputulokselle voidaan näin ollen laskea arvot. Päätöspuuanalyysin tavoitteena on tehdä "mahdollisimman" oikea ratkaisu. (TreePlan 2014.) Alla, kuvassa 1, on esitetty päätöspuun yleinen malli. Liikenteen luokittelun osalta päätöspuun avulla pyritään tunnistamaan sovellusprotokolla. Päätöspuut muodostetaan mahdollisten ominaisuuksien listasta sekä harjoittelutapauksista (Bujlow et al. 2012a). Toisin sanoen erittäin olennainen tekijä päätöspuun muodostamisessa on algoritmin "kouluttaminen" harjoitteludatalla. Käytännössä päätöspuun käyttäminen etenee kaksijakoisesti: päätöspuun oppiminen harjoitusesimerkeistä ja luokittelun päättely ennen näkemättömästä testausdatasta (Lewicki 2007).



Kuva 1. *Esimerkki päätöspuusta (Safavian & Landgrebe 1991)*

Edellä esitetyt DPI-tekniikat ovat käytössä laajasti eri kaupallisilla laitevalmistajilla tai avoimen lähdekoodin ohjelmistoissa. Näiden tekniikoiden lisäksi liikenteestä pyritään keräämään myös metatietoa tai muuta sisältöä. Tämä voi olla esimerkiksi pyydetty URL tai SMTP-runko. Usein sekä kaupalliset että avoimen lähdekoodin DPI-ohjelmistot sisältävät useamman edellä esitetyn tekniikan, joten ne eivät yleensä perustu vain yhteen em. menetelmään. Esimerkiksi avoimen lähdekoodin DPI-kirjasto libprotoident käyttää omissa säännöissään merkkijonoetsintää sekä pakettikoon vertailua, joka on periaatteessa yhdistelmä protokollan tunnusomaisten kaavojen ja tilastollisen tiedon hyväksi käyttämisestä sovellusprotokollan tunnistamisessa. Metadatan keräämisen osalta on kuitenkin havaittavissa ero kaupallisten ja avoimen lähdekoodin ohjelmistojen osalta, sillä

avoimen lähdekoodin ohjelmistoista ainoastaan nDPI käyttää analyysiliitännäisten lisäksi myös metadatan keräämistä. (Deri, Martinelli, Bujlow & Cardigliano 2014.) Tunnetut verkkolaitevalmistajat, kuten Cisco ja Juniper ovat kehittäneet omat DPI-tuotteensa. Esimerkiksi Cisco SCE Service Control Engine, Cisco NBAR Network-Based Application Recognition ja Juniper Junos Application Aware ovat tuotteita, joilla pyritään tunnistamaan verkossa liikennöivä sovellus. Muita tunnettuja DPI-valmistajia ja -tuotteita ovat: Procera Networks: Network Application Visibility Library (NAVL), Sandvine: Traffic Classification, Allot communications: Dynamic Actionable Recognition Technology (DART), ipoque: PACE 2.0 Protocol and application classification with metadata extraction ja Qosmos ixEngine. Suurimmat kaupalliset DPI-tuotteet on hyvä tunnistaa, mutta tässä tullaan keskittymään kuitenkin avoimen lähdekoodin ohjelmistoihin, joita käsitellään tarkemmin luvussa kolme.

2.3 DPI tunkeutumisen havaitsemis- ja -estojärjestelmissä

Kuten edellä todettiin, hyötykuorman analysointityökalujen kehittämiseen johtanut trendi edisti IDPS-järjestelmien esiintuloa. Näissä järjestelmissä hyödynnetään juuri DPI-tekniikkaa. Tunkeutumisen havainnointi suoritetaan tutkimalla paketin hyötykuormaa, josta pyritään päättelemään sisältääkö hyötykuorma haitallista koodia. Tämä päättely voidaan toteuttaa sormenjälkeen, anomaliaan tai protokollan "tunnettuun" tilaan perustuen. (Scarfone & Mell 2007.) Osa IDPS-järjestelmistä kykenee myös luokittelemaan liikennettä, mutta luokittelun taso on vaihteleva. Luokittelun vaihtelevuus johtuu tuettujen sovellusprotokollien määrästä. IDPS-järjestelmien suunnittelun lähtökohtana on ollut kuitenkin haitallisen liikenteen ja haitallisen ohjelmakoodin tunnistamisessa, joten niiden pääpaino ei ole ollut liikenteen luokittelussa. Näiden järjestelmien kehityskaari näyttäisi kuitenkin kulkevan suuntaan, jossa kaikki liikenne pyritään luokittelemaan. Tästä johtuen on hyvä selvittää näiden järjestelmien kyky luokitella liikennettä.

Vaikka IDPS-järjestelmien fokus ei ole suoranaisesti liikenteen luokittelussa, ovat nämä erittäin kiinnostavia liikenteen luokittelun näkökulmasta, kun pohditaan hämäännytetyn ja haitallisen liikenteen havainnointia. Monissa avoimen lähdekoodin IDPS-järjestelmissä, kuten Snort, Bro ja Suricata, sääntöjen kirjoittamiseen ei tarvita ohjelmointiosaamista. Sääntöjen avulla tutkitaan liikenteen hyötykuormaa ja laukaistaan mahdolliset hälytykset. Näiden laatimiseen on olemassa edellä mainittuihin järjestelmiin melko yksinkertainen syntaksi. Eräs kiinnostava kysymys onkin, voidaanko IDPS-sääntöjä hyödyntää tehokkaasti hämäännytetyn liikenteen havainnointiin. Usein myös haitallinen liikenne pyritään hämäännyttämään siten, että se näyttää joltain muulta liikenteeltä, mitä se itse asiassa on. Esimerkiksi kohdistetuissa hyökkäyksissä käytettyjen etähallintatyökalujen komentokanavaliikenne on usein hämäännytetty näyttämään normaalilta HTTP-liikenteeltä. Jos liikennettä ei tunnisteta HTTP-liikenteeksi, voidaan kyseinen hämäännytys havaita anomaliasäännön avulla. IDPS-järjestelmät sisältävät myös IP-osoiteiden maineeseen liittyvän säännöstön. Tämän avulla voidaan havaita

selvät maineeltaan huonot IP-osoitteet, kuten esimerkiksi tunnetut saastuneet tietokoneet, jotka toimivat roskapostin välittäjinä.

Tässä työssä esiteltävien DPI-kirjastojen pääasiallinen tarkoitus on luokitella liikennettä. Pakettien syvätarkastus, eli DPI on terminä varsin laaja, koska se käsittää paketin hyötykuorman tarkastelua eri näkökulmista. IDPS-ohjelmistot perustuvat DPI-tekniikkaan, mutta toisin kuin tässä opinnäytetyössä esiteltävät DPI-kirjastot, näiden ominaisuuksiin kuuluu esimerkiksi pakettien hyötykuorman tutkiminen, tietoliikenneistuntojen kokoaminen ja kokonaisten viestien sisällön esittäminen. Lähtökohtaisesti seuraavassa luvussa käsiteltävien DPI-kirjastojen yksi suunnittelun tavoite on ollut yksityisyyden säilyttäminen liikennettä tutkiessa. Tämä pyritään saavuttamaan tutkimalla mahdollisimman pientä määrää paketteja liikennevuota kohden.

3. AVOIMEN LÄHDEKOODIN LIIKENTEEN LUOKITTELUOHJELMISTOT

Liikenteen luokittelua voidaan suorittaa pakettien syvätarkastuksen ja liikennevuon tilastollisten ominaisuuksien avulla. DPI-järjestelmiä on olemassa sekä kaupallisia että avoimeen lähdekoodiin perustuvia. Tässä työssä keskitytään avoimen lähdekoodin toteutuksiin useista eri syistä. Luonnollisesti avoimen lähdekoodin ohjelmistot ovat ilmaisia. Kaupalliset DPI-kirjastot ovat usein erittäin kalliita lisenssi- ja ylläpitokustannusten vuoksi. Kiinteä lisenssikustannus on yksi hinnoittelukeino, mutta kaupallisissa tuotteissa voidaan käyttää myös vuosittaista hinnoittelua, joka ei perustu kiinteään lisenssihintaan, vaan organisaation tuottamaan voittoon. (Deri et al. 2014.) Toinen hyvä peruste avoimen lähdekoodin käyttöön on sen läpinäkyvyys. Koodi on kaikkien saatavilla, joten jos siinä ilmenee turvallisuuspoikkeavuuksia, kuten esimerkiksi takaportteja tai muuta haitallista koodia, niin sen havaitseminen on selvästi helpompaa ja todennäköisempää kuin suljetussa kaupallisessa ohjelmistossa. Tätä kirjoittaessa NSA:n erilaiset tiedustelutapaukset ovat yleisesti tiedossa ja esimerkiksi on olemassa mm. spekulaatiota siitä, onko älypuhelimien käyttöjärjestelmissä takaportteja (IT-viikko 2014). Kolmas peruste avoimen lähdekoodin ohjelmiston käytölle on se, että tarvittaessa siihen voidaan tehdä muutoksia. Osaamistasosta riippuen siihen voidaan esimerkiksi kehittää omia tunnistamisalgoritmeja tai mukautettuja protokollatunnistimia. Kaupallisten tuotteiden osalta pyynnöt joudutaan tekemään aina valmistajalle. Tällöin prosessi saattaa venyä johtuen valmistajan omista aikatauluista ja muutospyynnön aiheuttamasta byrokratiasta. (Deri et al. 2014.) Tässä alaluvussa on tarkoituksena perehtyä eri avoimen lähdekoodin toteutuksiin, jotta tunnistetaan niiden suorituskyky tunnistaa sovelluskerroksen protokollia ja ennen kaikkea hämäännytettyä liikennettä. On myös tärkeää selvittää ilmaisten tuotteiden kyky, jotta ei maksettaisi turhasta.

Kuten todettu, hyötykuorman tarkastusta suoritetaan useista eri syistä. Sen tavoitteena on mm. tunnistaa sovelluskerroksen protokolla, liikennemallien havaitseminen ja metadatan poimiminen. Metadata voi olla esimerkiksi käyttäjänimen selvittäminen paketin hyötykuormasta. (Deri et al. 2014.) Olemassa olevien avoimen lähdekoodin DPI-toteutusten ja kaupallisten toteutusten välillä on näiden tavoitteiden välillä eroja. Esimerkiksi kaupalliset ratkaisut kuten ipoque PACE 2.0 (engl. Protocol and Application Classification with metadata Extraction), Qosmos ixEngine ja Procera Networks:n Network Application Visibility Library (NAVL) pyrkivät vastaamaan kaikkiin kolmeen edellä mainittuun tavoitteeseen. Monella avoimen lähdekoodin DPI-kirjastolla tavoite on kapeampi. Esimerkiksi libprotoident, UPC ja l7-filter DPI-kirjastot ovat rajanneet skaalaksi ainoastaan protokollatunnistamisen. (Deri et al. 2014.)

3.1 OpenDPI

OpenDPI on eräs varteenotettava avoimen lähdekoodin DPI-ohjelmisto. Nykyään sen avoin kehitys on lopetettu. OpenDPI perustui edellä mainittuun kaupalliseen ipoque PACE -ohjelmistoon ja oli siis tämän avoimen lähdekoodin versio. (OpenDPI.org.) Sen toiminta perustuu protokollan käyttäytymismalleihin ja tilastolliseen analyysiin. Tilastollinen analyysi on toteutettu laskemalla joidenkin tilastollisten ominaisuuksien keskiarvo, mediaani ja arvojen variaatio. Variaation avulla voidaan toteuttaa käyttäytymisanalyysia sekä selvittää liikennevuon entropiaa. (Khalife, Hajjar & Díaz-Verdejo 2013). Tilastollisista ominaisuuksista on vaikea löytää tarkempaa tietoa, koska OpenDPI:n kehitys on lopetettu ja siitä löytyvää dokumentaatiota on enää vähän tarjolla. Todennäköisesti nämä ovat kuitenkin samoja kuin aikaisemmin esitetyt ominaisuudet, kuten pakettikoko ja pakettien väliset saapumisajat. Käytännössä käyttäytymismallit, joihin varsinaista sovellusprotokollaa verrataan, on toteutettu ohjelmoimalla jokaista sovellusprotokollaa kohden oma protokolladekooderi. Shen & Huang (2012) käyttävät tästä nimitystä liitännäinen. OpenDPI-kirjasto on kirjoitettu C-kielellä ja se on jaettu kahteen pääkomponenttiin:

- Ydinkirjasto, joka on vastuussa "raakapakettien" käsittelystä, verkkokerroksen/kuljetuskerroksen dekoodaamisesta ja perusinformaation, kuten IP-osoitteen ja porttitiedon keräämisestä.
- Analysointiliitännäiset, jotka ovat vastuussa protokollan tunnistamisesta. OPenDPI tukee noin 100 eri protokollaa. (Deri et al. 2014.)

Käytännössä itse tunnistaminen tapahtuu siis analysointiliitännäisten avulla. Liitännäinen tutkii hyötykuorman sekä joskus paketin metadatan, jotta voidaan päätellä vastaako tarkasteltava liikennevuo tiettyä sovellusprotokollaa (Shen & Huang 2012). OpenDPI pyrkii yhdistämään ns. protokolla-allekirjoitukseen (engl. signature-based) ja tilastolliseen analyysiin perustuvat DPI-menetelmät. Yhdistämisen tavoitteena on parempi tunnistamistarkkuus ja pienempi ns. väärien positiivisten määrä. (Khalife et al. 2013.) OpenDPI:n tunnistamistehokkuudesta on tehty useita tutkimuksia (Khalife, Hajjar & Díaz-Verdejo 2011; Shen & Huang 2012; Khalife et al. 2013). Tulokset tunnistamistarkkuudesta ovat olleet erinomaisia, sillä tutkimuksissa ollaan päästy jopa 99% tarkkuuteen tutkittavan aineiston osalta.

3.2 nDPI

Koska suosittua OpenDPI-kirjastoa ei enää kehitetä, niin uusi DPI-kirjasto on laadittu tämän pohjalta. Tämä ohjelmisto on nDPI ja siitä on tullut yksi tämän hetken tunnetuimmista avoimen lähdekoodin DPI-kirjastoista. nDPI on OpenDPI-kirjaston ylijoukko, joka kuuluu osana ntop- ja nProbe-ohjelmistoihin. Se on julkaistu LGPL-lisenssin alaisuudessa ja kirjaston tavoitteena on laajentaa alkuperäistä OpenDPI-kirjastoa uusilla protokolladekoodereilla eli analysointiliitännäisillä, joita ei ole saatavilla kuin maksulli-

sessa OpenDPI-versiossa tai ipoque PACE -ohjelmistossa. (nDPI - Quick Start Guide 2013.)

nDPI-kirjasto perustuu samankaltaiseen suunnittelumalliin kuin OpenDPI. Ohjelmisto jakaantuu samankaltaisesti kahteen pääkomponenttiin: ydinkirjasto ja analysointiliitännäiset, mutta siinä on myös lukuisia erilaisia parannuksia verrattuna sen esi-isään. Tällä hetkellä nDPI tukee yli 170 protokollaa, mutta sitä voidaan myös laajentaa ajonaikana käyttämällä konfiguraatiotiedostoa. nDPI-kirjaston koodi on täysin vapaakäyntinen (engl. reentrant). Tämä tarkoittaa sitä, että nDPI-kirjastoon perustuvien sovellusten ei tarvitse käyttää lukkoja tai muita tekniikoita sarjallistamisoperaatioissa. Kirjaston alustaminen tehdään aluksi kerralla, jolloin vältytään ajonaikaiselta alustamiselta, kun uusi paketti pitää analysoida. nDPI olettaa, että kirjaston kutsuja on jo jakanut paketit liikennevoiksi (paketit jaettu virtuaalilähiverkkotunnisteen ja 5-monikon perusteella) ja dekoodannut paketit verkkokerrokselle saakka. Tämä tarkoittaa sitä, että kutsujan tulee käsitellä kaikki siirtokerroksen kapseloinnit eli otsikkotiedot, kuten esimerkiksi virtuaalilähiverkkotunniste (VLAN ID) ja Multiprotocol Label Switching -leima (MPLS). nDPI:n tehtävänä on dekoodata paketti verkkokerrokselta ylöspäin.

nDPI:n mukana on yksinkertainen testausohjelma nimeltään pcapReader.c. Testausohjelman avulla voidaan osoittaa, kuinka pakettien luokittelu implementoidaan liikennevoissa ja se tarjoaa myös toiminnallisuudet liikennevuoprosessointiin.

Protokollien analysointiliitännäiset rekisteröidään oletusprotokolla- ja porttitiedoilla. Esimerkiksi HTTP-analysointiliitännäiseen on määritelty TCP/80 ja Internetin nimipalvelujärjestelmän (DNS) analysointiliitännäiseen TCP/53 sekä UDP/53. Käytännössä tällä saavutetaan kaksi merkittävää etua:

- Paketti, joka kuuluu toistaiseksi luokittelemattomaan liikennevuohon tarkistutetaan kaikilla analysointiliitännäisillä alkaen todennäköisimmästä ensin. Esimerkiksi paketti, jossa kuljetuskerroksen protokollana on TCP ja porttinumerona 80, syötetään ensin HTTP-analysointiliitännäiselle. Jos tunnistamista ei voida tehdä, niin tämän jälkeen käydään läpi loput TCP-liitännäiset. UDP-liitännäisiä ei luonnollisesti käsitellä, koska paketti käyttää TCP:tä kuljetuskerroksen protokollana. Tällä ratkaisulla saadaan pienennettyä käytettävien liitännäisten keskimääräistä määrää sekä analysointiin kuluvaa aikaa, koska todennäköisin liitännäinen tarkistetaan ensimmäiseksi. On hyvä selkeyttää, että tämä optimointi ei estä havaitsemasta ei-standardin mukaista HTTP-liikennettä.
- Kun liikennevuo on luokittelematon (esimerkiksi kaikki liitännäiset on kokeiltu, mutta mikään ei ole täsmännyt), nDPI kykenee arvaamaan sovellusprotokollan tarkistamalla, onko jokin protokolla rekisteröity vuon käyttämälle protokolla-porttiyhdistelmälle. Vuo voi olla luokittelematon johtuen analysointiliitännäisen rajoitteista tai jos kaikki paketit vuosta eivät ole kulkeneet nDPI:n kautta. nDPI

aktivointi vuon käynnistyksen jälkeen on tyypillinen esimerkki siitä, että vuon kaikki paketit eivät ole kulkeneet nDPI:n kautta.

Protokollan tunnistamisen elinkaari voidaan jakaa eri osiin:

1. nDPI dekoodaa verkkokerroksen ja siirtokerroksen paketista.
2. Jos paketin protokollaa/porttinumeroa vastaava analysointiliitännäinen on rekisteröity, sitä kokeillaan ensin.
3. Jos mikään liitännäinen ei vastaa pakettia, nDPI-kirjastolla on kaksi vaihtoehtoa: tunnistaminen epäonnistuu, koska paketit ei koskaan tule täsmäämään tai se epäonnistuu kyseisen paketin kohdalta, mutta tulevat paketit voivat täsmätä. Ensimmäisessä tapauksessa tulevien pakettien ei oleteta enää kuuluvan samaan vuohon, kun taas jälkimmäisessä tapauksessa samaa analysointiliitännäistä testataan myös tuleviin paketteihin.
4. Protokollan tunnistaminen päättyy heti, kun analysointiliitännäinen täsmää tunnistettavan paketin kanssa. Riippuen sovellusprotokollasta, liikennevuon tunnistamisessa tarvitaan yleensä useampi kuin yksi paketti. Kuitenkin UDP:n päällä toimivia protokollia, kuten DNS, NetFlow ja Simple Network Management Protocol (SNMP), on mahdollista tunnistaa yhden paketin perusteella. Toisaalta BitTorrent-protokollan tunnistamiseen voidaan tarvita kahdeksan pakettia. nDPI:n osalta tunnistamisen nyrkkisääntönä voidaan pitää kahdeksan pakettia suuntaansa, eli yhteensä 16 paketin perusteella pitäisi kyetä tekemään tunnistamisen päätös. Päätös voi olla myös tuntematon protokolla.

Toisin kuin OpenDPI, nDPI kykenee tunnistamaan myös salattua liikennettä. Salatun liikenteen tunnistaminen perustuu yhteyden alustamisessa käytettävän kättelyn tai avaimien vaihdon dekoodaamiseen. nDPI sisältää dekooderin esimerkiksi Transport Layer Security (TLS) -protokollaa varten. TLS tunnettiin aiemmin nimellä Secure Sockets Layer (SSL) ja sen avulla voidaan salakirjoittaa TCP:n päällä oleva hyötykuorma. TLS-dekooderi eli analysointiliitännäinen poimii liikenteestä palvelimen isäntänimen, joka on niin kutsuttua metadataa. Näin voidaan merkitä tunnettuja palveluita palvelimen nimen perusteella. Esimerkiksi salattu liikenne palvelimelle, jonka nimi on api.twitter.com, tunnistetaan Twitter-sovellukseksi. Myös muuta metatietoa kerätään. Esimerkiksi eräs etsittävä metatieto on itse allekirjoitettu TLS-varmenne. TLS-protokollaa ja -varmenteita hyödynnetään muun muassa salakirjoitetussa HTTP-liikenteessä, eli Hypertext Transfer Protocol Secure (HTTPS) -liikenteessä. Tämä liikenne on perinteisesti epäsymmetristä, jossa asiakas lähettää pyyntöjä palvelimelle ja palvelin vastaa varsinaisella informaatiolla. Palvelimen tunnistamiseen käytettävä varmenne on usein allekirjoitettuja jonkin yleisesti tunnetun varmenteen myöntäjän toimesta. Näin ollen itse allekirjoitetuilla varmenteilla muodostetut symmetriset pitkäkestoiset TLS-yhteydet ovat usein merkki virtuaalisista erillisverkoista (VPN). Näitä kutsutaan SSL-VPN -tunneleiksi.

3.3 L7-filter

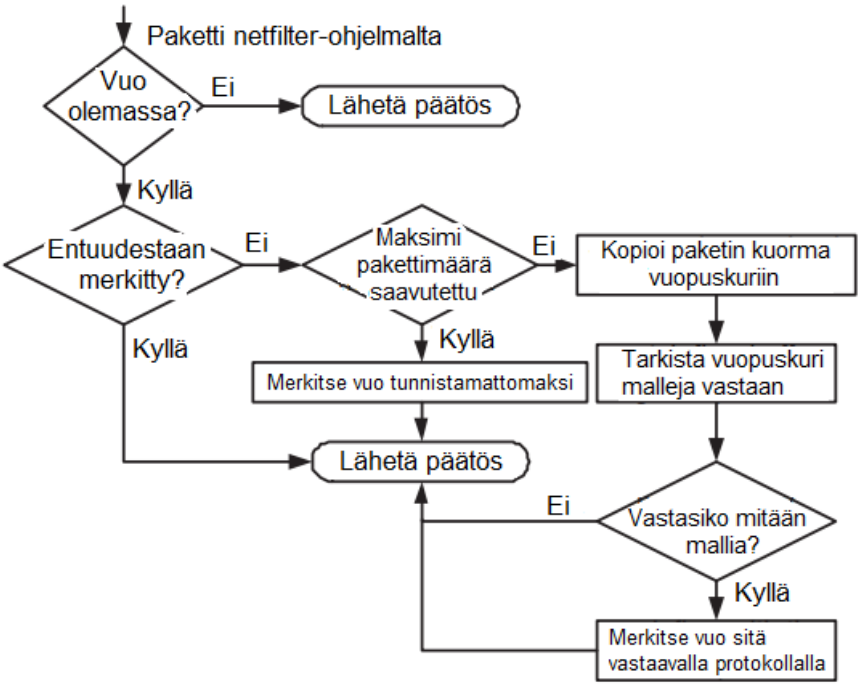
L7-filter -ohjelmiston toiminta perustuu tiettyjen kaavojen tai mallien etsimiseen liikenteen hyötykuormasta. Kaavoista tai malleista käytetään myös nimitystä "allekirjoitus" (engl. signature), koska protokollien otsikkotiedoissa on usein tiettyä kaavamaisuutta. Esimerkiksi HTTP-protokollasta voidaan havaita HTTP GET -pyyntö tai "HTTP/1.1 200 ok" ja FTP-protokollasta "220 ftp server ready", "230 User logged on, proceed" merkkijonoetsinnällä. L7-filter -ohjelmistossa allekirjoitusten etsintä toteutetaan kutsumalla POSIX regexec() -ohjelmointirajapintaa. Näin ollen etsintä tapahtuu säännöllisten lausekkeiden avulla. Jokaista protokollaa kohden ylläpidetään listaa malleista ja listan ylläpitoa edistää L7-filter-yhteisö. L7-filter on kehitetty kaksiosaiseksi siten, että itse ohjelmisto ja allekirjoitusten lista ovat erillisiä. Tämän vuoksi osia voidaan kehittää itsenäisesti. Allekirjoitukset ovat L7-filter -yhteisön yhteistyön tulos. Yhteisön kehittäjät ja tutkijat kirjoittavat malleja tietyille protokollille säännöllisten lausekkeiden avulla. (Shen & Huang 2012.)

Tällä hetkellä L7-filter -ohjelmiston ohjauksen ja kehittämisen vastuun on ottanut ClearFoundation (L7-filter | ClearFoundation). ClearFoundation tarjoaa kaksi erilaista Linux-versiota L7-filter -ohjelmistosta. Toinen toimii Linuxin ytimessä ja toinen Linuxin käyttäjätilassa. Näistä kernel-versio eli käyttöjärjestelmän ytimessä toimiva versio on yleisempi. Käyttöjärjestelmän ytimen ja L7-filter -ohjelmiston kommunikointi on toteutettu netlink-pistokkeiden päällä. (Shen & Huang 2012.)

L7-filter luottaa pakettien prosessoinnin ja yhteyden jäljittämisen osalta netfilter-runkoon. Ohjelman käynnistyksen yhteydessä muodostetaan kaksi säiettä. Ensimmäinen säie on l7 yhteyden jäljitin (engl. l7 connection tracker), joka vastaanottaa yhteyden jäljitystapahtumia käyttöjärjestelmän ytimeltä. Jäljitystapahtumiin kuuluvat mukaan myös mm. yhteyden alustus- ja purkamistapahtumat. Yhteyden jäljityssäie ylläpitää yhteystaulua luomalla siihen uusia yhteyksiä niiden alustamishetkellä ja poistamalla niitä sieltä yhteyden purkamisen jälkeen. Toinen säie on l7 jonojäljitin (engl. l7 queue tracker), joka vastaanottaa paketit käyttöjärjestelmän ytimeltä ja lähettää takaisin ytimen netfilter-ohjelmalle "päätöksiä". Päätökset ovat aina sallivia, koska L7-filter ei oletuksena pudota tai estä liikennettä. Jonojäljitin prosessoi paketin ainoastaan silloin, kun liikennevuo, johon paketti kuuluu on jo olemassa. Tämä tarkoittaa sitä, että yhteyden jäljittimen on pitänyt saada yhteyden alustamistapahtuma ennen jonojäljittimen prosessointia. Kuitenkaan näin ei tapahdu vuon ensimmäisen paketin osalta, koska netfilter ei kykene alustamaan yhteyttä ennen kuin se saa ensimmäisen paketin takaisin L7-filter -ohjelmistolta. Tämä ongelma ei oikeastaan koske TCP-yhteyksiä, sillä niiden ensimmäinen paketti, eli SYN-paketti ei sisällä yleensä hyötykuormaa. Toisaalta UDP-yhteydet usein sisältävät hyötykuormaa heti ensimmäisestä paketista lähtien, jolloin L7-filter ei kykene tekemään protokollan tunnistamista. L7-filter -ohjelmistosta löytyy kuitenkin laajennuksia, kuten L7-filter-U, jossa esimerkiksi tämä ongelma on korjattu. Ku-

vassa 2 on esitetty L7-filter -ohjelmiston pakettien prosessointi vuokaavion avulla. (Shen & Huang 2012.)

L7-filter -ohjelmisto rajoittaa tarkasteltavien pakettien määrää jokaista liikennevuota kohden. Käytännössä tämä on konfiguroitavissa oleva muuttuja nimeltään *maxpackets*. Jos protokollatarkistuksessa ei löydy vastaavuutta ja maxpackets-arvo saavutetaan, niin tarkastuksesta luovutaan. Useimmissa tapauksissa protokollan tietty kaavamaisuus ilmenee vuon ensimmäisissä paketeissa, jolloin pieni pakettimäärä vuon alusta riittää tunnistukseen. Tällä saavutetaan myös parempi suorituskyky, koska prosessoitavia paketteja on vain vähän suhteessa koko liikennevuohon. (Shen & Huang 2012.)



Kuva 2. *L7-filter-ohjelmiston paketin prosessointi (Shen & Huang 2012)*

3.4 Libprotoident

Libprotoident on DPI-kirjasto, joka sisältää C-kielisen ohjelmointirajapinnan. Se pyrkii mahdollisimman kevyeen tarkistukseen tarkistamalla neljä tavua paketin hyötykuormasta. Tällä saavutetaan pienempi levytilan eli varastoinnin tarve ja parempi yksityisyyden suoja, koska paketista tarvitaan vain vähän hyötykuormaa. (Alcock & Nelson 2012.) Koska libprotoident on kirjasto, tarvitaan ohjelma, joka käyttää ko. kirjastoa. Toisin sanoen verrattuna nDPI- ja L7-filter -ohjelmistoihin, periaatteessa tarvitaan ohjelmistokehitystä, jotta libprotoident-kirjastoa voidaan hyödyntää. Käytännössä kirjasto sisältää kuitenkin muutamia perustyökaluja, joilla voidaan demonstroida kirjaston kykyä tunnistaa protokollia ilman, että jouduttaisiin kehittämään koodia. Esimerkiksi mukana tuleva

lpi_protoident -työkalu pystyy raportoimaan tarkasteltavan liikennevuon protokollan, jonka libtrace pakettien kaappauskirjasto tarjoaa. (HowItWorks - Libprotoident 2012.)

Kehittäessä libprotoident-ohjelmia, käyttäjä on vastuussa pakettien lukemisesta lähteestä eli esimerkiksi verkkokortilta (Alcock & Nelson 2012). Lukeminen tapahtuu käyttämällä libtrace-kirjastoa. Jokainen paketti tulee määrätä kaksisuuntaiseen liikennevuohon ja paketin suunta tulee määrittää. (HowItWorks - Libprotoident 2012.) Paketin suunta voi saada arvot 0 tai 1. Kaikki sisään tuleva liikenne käyttää toista arvoa ja ulospäin lähtevä liikenne toista. (Alcock & Nelson 2012.) Arvoa ei ole siis etukäteen päätetty, vaan kehittäjä määrittää tämän. Lisäksi jokaisella kaksisuuntaisella liikennevuolla pitää olla lpi_data_t -rakenne, jota kirjaston kehittäjät kutsuvat "LPI data" -nimellä. LPI data alustetaan, kun kaksisuuntaista liikennevuota tutkitaan ensimmäisen kerran. Libflowmanager -ohjelmaa käytetään liikennevuon jäljittämiseen ja päätettyjen yhteyksien erääntymisten tarkkailuun. Jokainen paketti, jonka ohjelma lukee tulee välittää lpi_update_data -funktiolle paketin vuon LPI datan ja paketin suunnan kanssa. Lopuksi lpi_guess_protocol -funktiota voidaan käyttää liikennevuon sovellusprotokollan tunnistamiseen tai arvaamiseen, kuten funktion nimestäkin voidaan päätellä. Funktio käsittelee vuon LPI data -tietorakenteen ja palauttaa osoittimen protokollamoduuliin, jota kyseinen liikennevuo vastaa. (HowItWorks - Libprotoident 2012.) Käytännössä siis LPI data -tietorakenne on erittäin oleellinen komponentti tunnistamisen osalta. Se varastoi kaiken oleellisen informaation yksittäisestä kahdensuuntaisesta liikennevuosta, jota libprotoident tarvitsee päätelläkseen vuossa esiintyvän sovellusprotokollan. Tietorakenteeseen varastoidaan seuraavat tiedot:

- Ensimmäiset neljä tavua hyötykuormasta liikennevuon molempiin suuntiin.
- Ensimmäisen hyötykuormaa sisältävän paketin koko. Tämä tallennetaan myös molempiin suuntiin kulkevista paketeista. TCP/UPD/IP -otsikkojen kokoa ei oteta huomioon.
- Kuljetuskerroksen protokolla (esim. TCP, UDP, ICMP).
- Liikennevuon käyttämät porttinumerot.
- Liikennevuon käyttämät IP-osoitteet.
- Jos kyseessä on TCP-yhteys, ensimmäisen hyötykuormaa sisältävän paketin sekvenssinumero.
- Molempiin suuntiin kulkevan hyötykuorman tarkasteltu koko 32KB:n saakka. (HowItWorks - Libprotoident 2012.)

Sekvenssinumeroita ja tarkasteltua hyötykuorman kokonaiskokoa käytetään varmistamaan, että mahdolliset uudelleen järjestetyt TCP-segmentit on käsitellään oikein (Alcock & Nelson 2012). Libprotoident hylkää siis käytännössä suurimman osan liikennevuon segmenteistä tai paketeista, koska vain vuon ensimmäiset (ensimmäinen suuntaansa) hyötykuormaa sisältävät paketit merkitsevät (HowItWorks - Libprotoident 2012).

Kuten todettu protokollan määrittäminen tapahtuu kutsumalla lpi_guess_protocol -funktiota. Tällöin LPI data testataan tuettuja sovellusprotokollia vastaan kunnes vastaavuus löydetään. Jokainen tuettu protokolla on implementoitu erillisenä moduulina, jotka sisältävät mm. "vastaavuussääntöfunktion" (engl. rule matching function), jolla tarkistetaan protokollan mahdollinen vastaavuus sekä prioriteettiarvon kyseiselle protokollalle. Prioriteetti vaihtelee välillä 1 - 255. Pienempi arvo tarkoittaa korkeampaa prioriteettia. Prioriteetin avulla päätetään missä järjestyksessä vastaavuussääntöfunktiota ajetaan. Prioriteetti on määritetty sovelluksen yleisyyden ja suosion perusteella. Vastaavuussääntöfunktio palauttaa arvon "true", jos LPI data kohtaa sovelluksen ominaisuudet ja arvon "false", jos ominaisuudet eivät kohtaa. Funktio koostuu yhdestä tai useammasta säännöstä, joiden tulee "osua", jotta saadaan onnistunut tunnistaminen. On olemassa neljä erilaista sääntötyyppiä, joita protokollamoduuli voi hyödyntää sovellusprotokollan tunnistamisessa LPI data -tietorakenteesta. (HowItWorks - Libprotoident 2012.) Nämä ovat:

- Hyötykuormaosumat: Tämä on libprotoident-kirjaston yleisin sääntötyyppi. Liikennevuon molemmista suunnista kaapatusta neljän tavun hyötykuormasta etsitään tunnettuja protokollan kaavoja / malleja. Esimerkiksi BitTorrent-protokollan vastaavuussääntöfunktio etsii merkkijonokaavaa "0x13", "B", "i", "t".
- Hyötykuorman koko: Tämän säännön avulla verrataan ensimmäistä hyötykuormaa sisältävän paketin kokoa säännössä olevaan paketin kokoon. Esimerkiksi Skype-säännössä hyötykuorman koko yhteen suuntaan on 11 tavua. Jossain protokollissa, kuten esimerkiksi SMB-protokollassa, ensimmäiset neljä tavua sisältävät pituuskentän, jota voidaan verrata kyseisen paketin kokoa vasten.
- Porttinumero: Porttinumeroita käytetään usein selventämään tapauksia, joissa tarkasteltavan paketin hyötykuorma vastaa useaan sääntöön tai muuten heikkoon sääntöön, joka on taipuvainen vääriin positiivisiin tuloksiin. Tyypillisesti porttinumeroa hyödynnetään protokolliin, joilla on selkeästi määrätyt porttinumerot. Näitä ovat esim. DNS ja MySQL.
- IP-osoitteen vastaavuus: Vain harvoissa tapauksissa protokollan hyötykuorman neljä ensimmäistä tavua sisältää IP-osoitteen. Tämän vuoksi IP-osoitteet tallennetaan LPI data -tietorakenteeseen. Luonnollisesti tämä sääntö ei toimi, jos IP-osoitteet sanitoidaan pakettien kaappausprosessissa. (HowItWorks - Libprotoident 2012.)

Libprotoident ei tallenna mitään informaatiota onnistuneista osumista. Toisin sanoen onnistuneen tunnistamisen jälkeen ei tallenneta tietyn protokollan ja porttinumeron yhdistelmää, jotta tulevat tunnistamiset voitaisiin tehdä nopeasti tämän tallennetun tiedon perusteella. Tämä johtuu siitä, että IP-osoite ja porttinumeroyhdistelmän tilan tallennus on erittäin muisti-intensiivistä, varsinkin ruuhkaisilla linkeillä. Tämä tarkoittaa sitä, että

libprotoident kohtelee jokaista liikennevuota itsenäisesti. (HowItWorks - Libprotoident 2012.)

3.5 C5.0 ja SPID

C5.0 ja Statistical Protocol IDentification (SPID) ovat molemmat koneoppimisalgoritmeja. Molemmista on olemassa myös avoimen lähdekoodin toteutukset. Näiden algoritmien tavoitteena on hyödyntää empiiristä dataa, kokemusta ja harjoitusdataa, joiden avulla voidaan muuntautua ja sopeutua muutoksiin, jotka ilmenevät niiden ympäristössä. Käytännössä molemmat algoritmit tulee ensin kouluttaa harjoitusdatalla, ennen kuin niitä voi käyttää liikenteen luokitteluun. C5.0-algoritmi soveltuu monenlaisen tiedon luokitteluun ja datan louhintaan ja sitä on hyödynnetty tuloksellisesti myös tietoliikenteen luokittelussa ja sovellusprotokollan tunnistamisessa. C5.0-algoritmin soveltamisesta pakettien syvätarkastuksessa on tehty useita tutkimuksia (Bujlow, Riaz & Pedersen 2012a; Bujlow, Riaz & Pedersen 2012b; Shubhangi et al. 2013). Toisin kuin C5.0, SPID-algoritmi on kehitetty pelkästään pakettien syvätarkastusta varten. Molemmat algoritmit hyödyntävät tilastollista mittaamista sovellusprotokollan tunnistamisessa, joka eroaa staattisten kaavojen etsinnästä.

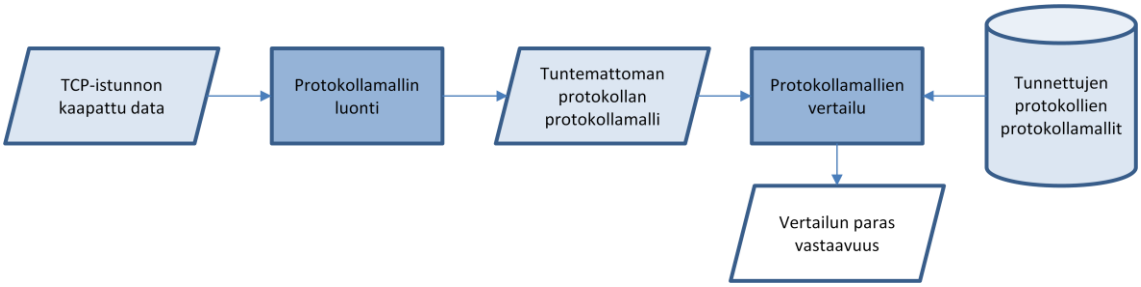
C5.0-algoritmi kehitettiin aikaisemmin mainitun ja laajasti käytetyn C4.5 algoritmin pohjalta. C5.0 on paranneltu versio ja sillä on useita etuja sekä parannuksia verrattuna sen edeltäjään. Näitä ovat mm. parempi tarkkuus, nopeampi suoritus ja pienempi muistin tarve. (RuleQuest Search 2012.) Kuten todettu, käytettäessä C5.0-algoritmia liikenteenluokittelijana, tulee se ensin kouluttaa harjoitusdatalla ennen kuin sitä voidaan käyttää verkossa. Kouluttaminen voidaan toteuttaa esimerkiksi tekstitiedoston avulla. Tekstitiedostosta kone oppii pakettien ominaisuuksia ja näitä ominaisuuksia verrataan "Knowledge Discovery in Databases (KDD) -datajoukkoon". (Shubhangi et al. 2013.) KDD tarkoittaa abstraktia tietämystä, joka on muodostettu raakadatasta tai aikaisemmasta tietämyksestä (ParisinLA project). Käytännössä tämä tarkoittaa datan louhintaprosessia. Liikenteenluokittelijan kouluttaminen koostuu siis useammasta vaiheesta. Shubhangi et al. (2013) kuvaavat kouluttamisprosessin seuraavasti: "Prosessissa KDD-louhinta suoritetaan aikaisemmin tuntemattomaan tietoon, joka arvioidaan tunnetulla tiedolla". Bujlow et al. (2012a) kouluttivat omassa tutkimuksessaan luokittelijan työasemilta kaapatun liikenteen avulla. Liikenne tallennettiin pcap-tiedostoina keskitetysti palvelimelle, jossa tiedostoista luotiin sovelluskohtaisesti liikennetilastot. Näiden tilastojen avulla luotiin luokittelusäännöt eli päätöspuut. Koska C5.0 perustuu sen edeltäjään C4.5:een, niin myös C5.0 on päätöspuualgoritmi. Yleinen algoritmi päätöspuiden rakentamiseen on seuraava:

- valitaan puun juureen parhaaksi osoittautunut attribuutti
- jaetaan opetusaineisto edellä mainitun attribuutin arvojen perusteella

- puun muodostamista jatketaan alipuista samalla periaatteella. (OHJ-2550 Tekoäly 2009.)

C5.0 luokittelija sisältää yksinkertaisen komentorivikäyttöliittymän, jonka avulla päätöspuut voidaan luoda (Bujlow et al. 2012a).

SPID-algoritmi on koneoppimismenetelmään perustuva algoritmi, joka suorittaa protokollatunnistuksen TCP- tai UDP-istunnosta vertaamalla tutkittavaa protokollamallia esilaskettuihin tunnettuihin protokollamalleihin. Jokainen esilaskettu protokollamalli pitää sisällään joukon "sormenjälkiä", joiden avulla tunnistus suoritetaan. Sormenjälki on jokin protokollalle tyypillinen tilastollinen ominaisuus. Tilastolliset ominaisuudet ovat mitatun määreen todennäköisyysjakaumia, joita ovat esimerkiksi sovelluskerroksen data tai liikennevuon tietty tilastollinen ominaisuus. Sormenjäljet luodaan tekemällä frekvenssianalyysi useista protokollan piirteistä, kuten sovelluskerroksen datan arvoista tai liikennevuon ominaisuuksista. Niiden data esitetään kahtena diskreettien "lokeroiden" taulukkona eli datamatriisina. Toinen taulukko sisältää "laskurilokerot" ja toinen "todennäköisyyslokerot". Datan avulla muodostetaan sormenjäljille todennäköisyysjakaumat. Samoin kuin libprotoident-kirjasto, SPID-algoritmi ei pidä kirjaa aikaisemmin tunnistetuista protokollista ja niiden porttinumeroista, vaan jokainen istunto luokitellaan itsenäisesti. (Hjelmvik & John 2010, s. 7.) Alla on havainnollistettu tiedon kulku SPID-toteutuksessa.



Kuva 3. *Tunnistamisprosessin tiedon kulku SPID-toteutuksessa (Hjelmvik & John 2010)*

SPID-sovelluksessa on yli kolmekymmentä ominaisuusmittaria (engl. AttributeMeter), joiden avulla mitataan sovelluskerroksen protokollien yleistä käyttäytymistä. Näiden avulla luodaan em. tunnistamisprosessissa protokollamalli. Jokaiselle tarkkailtavalle istunnolle luodaan protokollamalliolio istunnon muodostamisvaiheen aikana kolmivaiheisen kättelyn jälkeen. Olio sisältää joukon sormenjälkiä eli tilastollisia ominaisuuksia, jotka on mitattu tietyillä ominaisuusmittareilla. Jokaista istuntoon kuuluvaa pakettia, jossa on sovelluskerroksen dataa, kutsutaan havainnoksi. Näin ollen jokainen havainto syötetään kyseisen istunnon protokollamallioliolle. Havainnon vastaanottamisen aikana protokollamalli kasvattaa sormenjälkilaskureita vastaavasti kunkin ominaisuusmittarin mukaan. SPID-sovellus pyrkii tuottamaan protokollatunnistuksen mahdollisimman aikaisessa vaiheessa istuntoa. Tämän vuoksi nykyiset ominaisuusmittarit ottavat huomi-

oon vain istunnon kahdeksan ensimmäistä pakettia ja näistä tyypillisesti vähemmän kuin 32 tavua. (Hjelmvik & John 2010, s. 7-8.)

3.6 Suricata

Suricata on avoimen lähdekoodin tunkeutumisenesto- ja havaitsemisjärjestelmä, jonka kehityksestä vastaa voittoa tavoittelematon yhteisö nimeltään "Open Information Security Foundation" eli OISF. IDPS-järjestelmät suorittavat pakettien syvätarkastusta, joten on luonnollista ottaa tarkasteluun mukaan myös nämä järjestelmät. Suricatan ohella muita tunnettuja avoimen lähdekoodin tunkeutumisenesto- ja havaitsemisjärjestelmiä ovat Snort ja Bro. Tässä on kuitenkin tarkoitus tarkastella vain Suricata-järjestelmää, johtuen sen tietyistä eduista verrattuna kahteen edellä mainittuun järjestelmään. Suricata-, Snort- ja Bro-järjestelmiä on vertailtu useissa tutkimuksissa, esimerkiksi Day & Burns (2011), Albin & Rowe (2012), Pihelgas (2012) ja Mehra (2012). Tutkimuksissa on vertailtu eri ominaisuuksia, kuten esimerkiksi liikenteen läpäisykykyä, haitallisen kuorman tunnistamisen tarkkuutta, prosessorikuormaa ja muistin käyttöä. Kaikki edellä esitetyt tutkimukset eivät kuitenkaan vertaile kaikkia kolmea järjestelmää keskenään, mutta näistä voidaan vetää yhteenvetona liikenteen läpäisykyvyn ja tunnistustarkkuuden osalta tulos, jossa Suricata on tehokkain. Tosin tunnistustarkkuudessa on hyvin pienet erot verrattuna Snort-järjestelmään (Day & Burns 2011; Albin & Rowe 2012). Suorituskyky läpäisykyvyn osalta johtuu Suricatan monisäiearkkitehtuurista, jonka avulla paketit voidaan jakaa usealle säikeelle ja yksi säie voidaan asettaa yhdelle prosessoriytimelle. Juuri tämän ominaisuuden vuoksi Suricata kykenee nykyaikaisella raudalla 10Gbit/s läpäisykykyyn pudottamatta paketteja. (Leblond 2012.) Baena (2014) luettelee Suricatan eduiksi monisäietuen, yksilölliset sovelluskerroksen tunnistusmoduulit ja nopean HTTP-liikennevuon normalisoinnin sekä jäsentelyn.

Pakettien syvätarkastuksen osalta Suricata toimii periaatteeltaan samankaltaisesti kuin edellä esitetyt DPI-ohjelmistot. Liikenteestä poimitaan istunnot eli liikennevuot ns. monikkoperusteisesti (engl. tuple). Suricata kykenee myös dekoodaamaan vlan-informaation (engl. virtual lan). Syvätarkastus alkaa käytännössä dekoodaamalla paketista eri kerroksien kehysten otsikkotiedot ja ne kootaan uudelleen liikennevoiksi. Ethernet-, IP- ja TCP/UDP-kehyksen otsikkotietojen avulla muodostetaan monikkoavain, jonka avulla päätellään liikennevuo, johon paketti kuuluu. Jokaiseen liikennevuohon kuuluvat paketit uudelleen kootaan Suricata-järjestelmässä, jolloin saadaan kerättyä istunnon data kokonaisuudessaan ja voidaan rekonstruoida esimerkiksi koko HTTP-istunto. (Open Information Security Foundation 2010.)

Suricata käyttää sovelluskerroksen protokolladekoodereista nimitystä sovelluskerroksen jäsennin (engl. parser). Liikenne tunnistetaan ja luokitellaan näiden avulla. Suricata voidaan ryhmitellä näin ollen kuuluvaksi protokolladekoodereita käyttäviin järjestelmiin edellä esitettyjen DPI-tekniikoiden osalta. Dokumentaation mukaan Suricatassa on valmiina sovelluskerroksen dekooderit seuraaville protokollille: TLS, DCERPC, FTP,

SSH, SMTP, IMAP, MSN, SMB, DNS (UDP ja TCP) ja HTTP. Tämä on varsin pieni määrä verrattuna esimerkiksi nDPI-kirjastoon, joka tukee lähes 200 eri sovellusta. Suricata käyttää libhtp-kirjastoa HTTP-protokollaa varten. Se kykenee myös poimimaan siirretyt tiedostot esimerkiksi HTTP-liikennevuosta. Hyödyntämällä libmagic-kirjastoa, voidaan tunnistaa tiedostotyypit riippumatta merkitystä tiedostopäätteestä. (Inliniac 2011.)

Suricata tukee oletuksena vain murto-osan sovelluskerroksen protokollia verrattuna DPI-kirjastoihin. Siinä voidaan kuitenkin melko yksinkertaisesti luoda oma sääntö sovelluskerroksen protokollaa vastaan. Näiden sääntöjen tekemiseen voidaan hyödyntää Suricatan "liikennevuoavainsanoja" (engl. Flow-keywords). Nämä mahdollistavat sääntöjen kirjoittamisen, joka koskee useaa liikennevuon pakettia. Hälytys tai tapahtuma luodaan, kun esimerkiksi kaksi pakettia liikennevuosta omaa tietyn sisällön. (Open Information Security Foundation 2011.) Sääntöjen kirjoittaminen on selvästi helpompaa, kuin ohjelmakoodin kirjoittaminen. Näin ollen uuden sovellusprotokollatunnistimen lisääminen ei vaadi ohjelmointiosaamista, mikä luonnollisesti parantaa käytettävyyttä.

3.7 DPI-ohjelmistojen liikenteen luokittelukyky ja luokittelukyvyn tutkiminen

Laajaa tutkimusta hämäännytetyn eli obfuskoidun liikenteen tunnistamisesta eri avoimen lähdekoodin DPI-kirjastoilla ei ole tiedossa. Oletus on, että osa hämäännyttämismenetelmistä on avoimen lähdekoodin DPI-kirjastoja edellä, joten liikennöivää sovellusprotokollaa ei todennäköisesti tunnisteta tai liikenne luokitellaan oletuskonfiguraation avulla väärin. DPI-kirjaston tunnistamistarkkuus korostuu tämän vuoksi, koska sovellettaessa DPI-kirjastoa esimerkiksi organisaation liikenteen tarkkailuun tietoturvannäkökulmasta, voidaan sulkea ns. "varmat" tunnistamiset pois. Esimerkiksi tarkkailtaessa organisaation liikenneprofiilia, luokittelematon tai tunnistamaton sovellusprotokolla voi tarvita jatkoanalyysia. Liikenteen luokittelukyky on näin ollen tässä työssä tärkein kriteeri valittessa DPI-kirjastot hämäännytetyn liikenteen tunnistamisen tutkimiseen. Erityisesti oikea luokitus ja väärin tunnistamisen vähyys ovat merkitseviä. Väärä positiivinen arvo (engl. false positive) on selvästi haitallisempi, kuin tuntematon arvo. Tässä luvussa on tarkoituksena selvittää olemassa olevan kirjallisuuden perusteella luokittelukyvyltään parhaimmat avoimen lähdekoodin DPI-kirjastot. Näiden ohjelmistojen avulla on tarkoituksena selvittää hämäännytetyn liikenteen havaitsemista.

On olemassa tutkimuksia, jotka esittävät, että koneoppimismenetelmän avulla saavutetaan erittäin korkea tunnistamistarkkuus. (Deri et al. 2014.) Esimerkiksi Ubik & Žejdl (2010) ja Jun et al. (2007) esittävät, kuinka C4.5-koneoppimisalgoritmin avulla saavutetaan yli 95% tunnistamistarkkuus. Toisaalta käytännön testit tuotantoverkoissa ovat osoittaneet, että koneoppimisalgoritmien avulla voidaan luokitella liikennettä vain muutamaan kategoriaan. Esimerkiksi verrattuna protokolladekoodereihin, koneoppimisalgo-

ritmit ovat huonoja menetelmiä niille, jotka tarvitsevat hienojakoisen luokittelun. Jossain testeissä koneoppimisalgoritmit ovat tuottaneet korkean määrän vääriä tuloksia, jolloin näiden käyttäminen on hyödyllistä vain passiivisessa liikenneanalyysissä. Näin ollen koneoppimisalgoritmeja ei ole järkevää käyttää tapauksissa, joissa liikennettä joudutaan estämään tuloksien perusteella ja korkea luotettavuus on pakollista. (Deri et al. 2014.)

Pakettien syvätarkastus on toimenpide, jossa tarkastellaan pakettien hyötykuormaa tietyssä tarkastuspisteessä (Deri et al. 2014). Hyötykuorman tarkastelua suoritetaan eri syistä kuten edellä on esitetty. Tavoitteena voi olla sovellusprotokollan tunnistaminen, liikenteen toistumakuvioiden tunnistaminen ja metatiedon poimiminen, kuten esimerkiksi käyttäjätunnusten kerääminen. Kaupalliset DPI-kirjastot kuten ipoque PACE ja Qosmos ixEngine suorittavat näitä kaikkia toimenpiteitä, kun taas avoimen lähdekoodin libprotoident ja L7-filter rajoittuvat protokollan tunnistamiseen. Protokollan tunnistaminen voidaan implementoida käyttämällä toistumakuvioiden tunnistamista esimerkiksi säännöllisten lausekkeiden avulla tai siihen erikoistuneilla protokolladekoodereilla. Ensiksi mainittu lähestyminen on hidas johtuen säännöllisten lausekkeiden käyttämisestä sekä taipuvainen virheisiin johtuen seuraavista syistä (Deri et al. 2014.):

- Se ei rekonstruoi paketteja 6-monikkoavaimien avulla (VLAN, protokolla, IP/portti, lähde/kohde), jolloin jokainen paketti käsitellään ns. omana erillisenä "dokumenttina". Tämän vuoksi tunnistusta ei voida toteuttaa usean paketin sisältä. Toisin sanoen usean paketin yhdistämistä "yhdeksi dokumentiksi" ja tunnistuksen suorittamista tästä, ei tueta.
- Etsimällä toistumakuvioita ei-dekoodatusta hyötykuormasta voi johtaa vääriin tunnistuksiin. Esimerkiksi sähköposti, joka sisältää katkelman HTTP-yhteydestä voidaan sekoittaa web-liikenteeksi.

DPI-kirjaston valitsemiseen vaikuttaa luonnollisesti se, mihin käyttötarkoitukseen sitä on tarkoitus soveltaa. Esimerkiksi pelkästään passiiviseen pakettianalyysiin keskittyminen vähentää avain ominaisuuksien määrää verrattuna DPI-kirjastoon, jonka tulee kyetä myös aktiiviseen analyysiin sekä analyysin avulla suodatuspolitiikoiden toteuttamiseen.

Edellä esitettyjen SPID-algoritmin ja Suricata IDPS-ohjelmiston tunnistamisen luotettavuutta ei ole tutkittu yhdessä muiden DPI-kirjastojen kanssa. Itse asiassa, koska Suricata on puhdas IDPS-ohjelmisto, sitä ei ole vertailtu liikenteen luokittelussa lainkaan. Kehitettäessä organisaatiolle liikenteen luokittelu- ja tunnistamiskykyä, on hyvä kuitenkin huomioida mahdollisen DPI-kirjaston ja IDPS-ohjelmiston yhdistämisen edut. DPI-kirjaston tuottama tunnistamaton tulos voi olla merkki tarvittavasta jatkokäsittelystä esimerkiksi IDPS-ohjelmistolle.

Kirjallisuuden perusteella tehtävää arviointia edellä esitettyjen ohjelmistojen liikenteen luokittelukyvystä on haastava muodostaa, koska osaa niistä ei ole arvioitu lainkaan tai

arviointi on suoritettu eri tutkimuksissa eri liikennemateriaalin perusteella. Käytännössä osa tässä esitetyistä ohjelmistoista voidaan sulkea jo heti alussa pois. Esimerkiksi OpenDPI-kirjaston kehitys ja ylläpito on jo lopetettu, joten sen luokittelukykyä ei ole mielekästä enää selvittää. OpenDPI on ollut kuitenkin merkittävä avoimen lähdekoodin DPI-kirjasto ja sen luokittelukykyä on tutkittu aikaisemmin paljon. Tämän vuoksi esiteltäessä avoimen lähdekoodin DPI-kirjastoja, OpenDPI:tä ei voi sivuuttaa.

Arvioitaessa kirjallisuuden perusteella edellä esitettyjen DPI-kirjastojen liikenteen luokittelukykyä, on lähteessä Bujlow, Carela-Español & Barlet-Ros (2013) esitetty tutkimus varsin kattava. Tässä on tutkittu laajasti eri sovelluksien tuottamaa liikennettä useiden DPI-kirjastojen avulla. Kyseisessä tutkimuksessa ei kuitenkaan oteta kantaa hämäännytettyyn liikenteeseen. Tutkimuksessa käytetty testidata oli ns. todellista liikennettä. Testidataa ei ollut generoitu ohjelmistolla, vaan sen keräämisessä oli käytetty Aalborgin yliopistossa kehitettyä liikenteen keräystyökalua nimeltään "Volunteer-Based System" (VBS). Tämän ideana on kerätä liikennettä kohdekoneesta siten, että liikennöivän sovelluksen tiedot sidotaan sen tuottamaan liikennekaappaukseen, jotka edelleen lähetetään tietokantaan keskitetylle palvelimelle. Palvelimelta voidaan tämän jälkeen ajaa paketti kerrallaan liikenne DPI-ohjelmiston lävitse.

DPI-kirjastoilla on erilaiset tarkkuustasot tunnistamisen osalta. Esimerkiksi joku kirjasto voi tunnistaa HTTP-liikenteen Youtube-palvelusta HTTP-liikenteeksi, kun taas toinen kirjasto voi tunnistaa sen Youtube-liikenteeksi. Tämän vuosi on arvioitu liikenteen luokittelijan tunnistamisesta seuraavia asioita: oikea "tarkka" tunnistaminen, oikea "epätarkka" tunnistaminen, väärä tunnistus ja tuntematon. Kyseisen tutkimuksen luokittelukyvyn tulokset on esitetty alla olevassa taulukossa 1. (Bujlow et al. 2013.)

Taulukko 1. *Lähteessä Bujlow et al. 2013 esitetyt tulokset*

Luokittelija	% Oikea tarkka	% Oikea epätarkka	% Väärä	% Luokittelematon
PACE	63,33	30,48	0,15	6,05
OpenDPI	50,77	30,61	0,00	18,61
L7-filter-com	27,29	12,85	23,02	36,84
L7-filter-aut	36,14	6,65	5,76	51,45
L7-filter-all	25,42	0,84	48,15	25,58
L7-filter-sel	18,64	16,04	7,73	57,59
nDPI	82,73	5,48	1,17	10,61
Libprotoident	56,11	32,71	0,38	10,81
NBAR	31,17	33,33	0,88	34,61
UPC MLA	60,91	34,89	4,20	0,00

Kuten taulukosta 1 havaitaan, Bujlow et al. (2013) on tutkinut myös muita kuin avoimenlähdekoodin DPI-kirjastoja. Kaupallisista tuotteista mukana on ipoque PACE ja Cisco NBAR. Taulukossa alimpana oleva UPC koneoppimisalgoritmi (UPC MLA) perustuu puhtaasti C5.0 koneoppimisalgoritmiin. L7-filter -ohjelmistosta on mukana useita variantteja.

Laskettaessa oikeat tarkat ja oikeat epätarkat prosenttiosuudet yhteen, taulukon 1 tuloksista havaitaan, että selvästi parhaimmat avoimen lähdekoodin liikenteen luokittelijat ovat UPC MLA, Libprotoident ja nDPI. Tämä tulos tukee muita olemassa olevia tutkimuksia, joiden perusteella koneoppimismenetelmän avulla saavutetaan erittäin korkea tunnistamistarkkuus. Tästä huolimatta Deri et al. (2014) esittää, että jossain testeissä koneoppimisalgoritmit ovat tuottaneet korkean määrän vääriä tuloksia, jolloin näiden käyttäminen on hyödyllistä vain passiivisessa liikenneanalyysissä. Bujlow et al. (2013) ei tue tätä esittämää, sillä vääriä tuloksia on UPC MLA -algoritmilla hieman yli 4%. Toisaalta verrattuna nDPI- ja Libprotoident-kirjastoihin, vääriä havaintoja on selvästi enemmän. Koneoppimisalgoritmien haasteena on suuri laadukkaan koulutusdatan tarve (Bujlow 2014). Deri et al. (2014) esittää, että käytännön testit tuotantoverkoissa ovat osoittaneet, että koneoppimisalgoritmien avulla voidaan luokitella liikennettä vain muutamaan kategoriaan. Bujlow et al. (2013) kuitenkin osoittaa, että koneoppimisalgoritmi kykenee hienojakoiseen luokitteluun siinä missä muutkin avoimen lähdekoodin DPI-kirjastot. Tuloksissa tarkka tunnistamistulos on UPC MLA -algoritmilla ja Libprotoident-kirjastolla lähes sama.

Tässä esitettyjen avoimeen lähdekoodiin perustuvien DPI-kirjastojen tukema sovellusprotokollien määrä vaihtelee 100-250 välillä. On hyvä kuitenkin huomioida, että ilmoitettujen tuettujen sovellusprotokollien havaitsemisen taso voi olla hyvinkin vaihteleva eri DPI-kirjastojen välillä. Esimerkiksi L7-filter -ohjelmiston jokaista protokollaa koskevien toistumakuvioiden ylläpito ja päivittäminen on L7-filter yhteisön tukema ja nämä vaihtelevat laadun osalta. Sovellusprotokollaa koskevia julkaisuja saattaa esiintyä varsin harvoin suhteessa sovellusprotokollien muuttumiseen. (Shen & Huang 2012.) Bujlow et al. (2013) tukee tätä väittämää, sillä L7-filter ohjelmistojen eri variantit selviävät luokittelusta selvästi heikoiten.

nDPI tukee ainoana avoimen lähdekoodin DPI-kirjastona metadatan poimimista. Esimerkiksi Libprotoident analysoi liikenteestä vain 4 tavua molempiin suuntiin. Tällöin metadatan poiminta ei ole mahdollista, koska hyötykuorman tarkastelua ei jatketa 4 tavun jälkeen. Libprotoident-kirjasto ja C5.0 -algoritmiin perustuva UPC MLA ovat tunnistamistarkkuuden osalta hyvin lähellä toisiaan. Libprotoident on kuitenkin väärän luokittelun suhteen selvästi parempi kuin koneoppimisalgoritmi. Tuntematon luokitus on parempi vaihtoehto kuin väärä tunnistaminen, koska väärä tunnistaminen voi johtaa liikenteen havaitsemattomuuteen. Tämä tulee ongelmaksi esimerkiksi silloin kun pyritään havaitsemaan haittaohjelmien komentokanavia. Väärän tunnistamisen johdosta kyseistä liikennettä ei havaita.

Bujlow et al. (2013) päätyvät oman tutkimuksensa perusteella suosittelemaan avoimen lähdekoodin DPI-kirjastoista Libprotoident- ja nDPI-kirjastoja riippuen skenariosta, jossa ohjelmistoa tullaan käyttämään. Sinänsä suositus on mielenkiintoinen, koska C5.0-algoritmiin perustuvaa UPC MLA -ohjelmistoa ei suositella, vaikka sen liikenteen luokittelukyky on hyvä.

Edellä on esitetty olemassa olevan kirjallisuuden perusteella seikkoja, joiden perusteella hämäännytetyn tietoliikenteen havaitsemisen tutkimukseen valitaan DPI-kirjastoiksi nDPI ja Libprotoident. Tässä päädytään samaan ratkaisuun kuin Bujlow et al. (2013). Kolmanneksi tutkittavaksi ohjelmistoksi valitaan Suricata, jonka liikenteen luokittelukyvystä ei ole tutkimustietoa tiedossa. Koska tarkoituksena on tutkia myös haitallista hämäännytettyä liikennettä, on luonnollista valita mukaan myös IDPS-ohjelmisto. On selvää, että IDPS-ohjelmiston tulisi havaita haittaohjelmien komentokanavia paremmin kuin edellä esitettyjen DPI-kirjastojen.

4. HÄMÄÄNNYTETTY TIETOLIIKENNE JA SEN MENETELMÄT

Jotta hämäännytetty tietoliikenne kyetään tunnistamaan, on hyvä ymmärtää olemassa olevia hämäännyttämismenetelmiä. Tässä luvussa on tarkoitus perehtyä niiden toimintaperiaatteisiin ja toteuttamisessa käytettyihin algoritmeihin. Hämäännytetyllä tietoliikenteellä tarkoitetaan tietoliikenneprotokollan toiminnan tarkoituksenmukaista monimutkaistamista. Termi tunnetaan myös protokollaobfuskointina. Monimutkaistamisen tavoite on hämätä liikennettä tutkivaa laitetta siten, että se ei tunnista liikennöivää sovellusta tai tarkemmin ilmaistuna sovelluskerroksen protokollaa (Hjelmvik & John 2010). Protokollan tarkoituksenmukainen monimutkaistaminen voidaan toteuttaa eri tavoin.

Monimutkaistamisen tavoite eli hämäännytetty liikenne saavutetaan esimerkiksi poistamalla tai muuttamalla protokollan helposti tunnistettavia piirteitä. Tällaisia piirteitä ovat esimerkiksi tietyn protokollan deterministiset tavusekvenssit ja ennalta tunnetut pakettikoot. Käytännössä helposti tunnistettavien piirteiden muuttaminen tai poistaminen tapahtuu konvertoimalla data siten, että se näyttäisi olevan täysin satunnaista. (Hjelmvik & John 2010.) Tässä luvussa käsitellään hämäännyttämismenetelmien perusteita, eli toimintoja, kuinka hämäännytetty liikenne muodostetaan. Luvussa käytetään termejä hämäännyttäminen, hämäännyttämismenetelmä ja protokollaobfuskointi, joilla kaikilla tarkoitetaan tietoliikenneprotokollan tarkoituksenmukaista monimutkaistamista tai sen monimutkaistamista ja naamiointia joksikin muuksi. Yleisesti tästä tullaan käyttämään termiä: **tietoliikenteen hämäännyttäminen**.

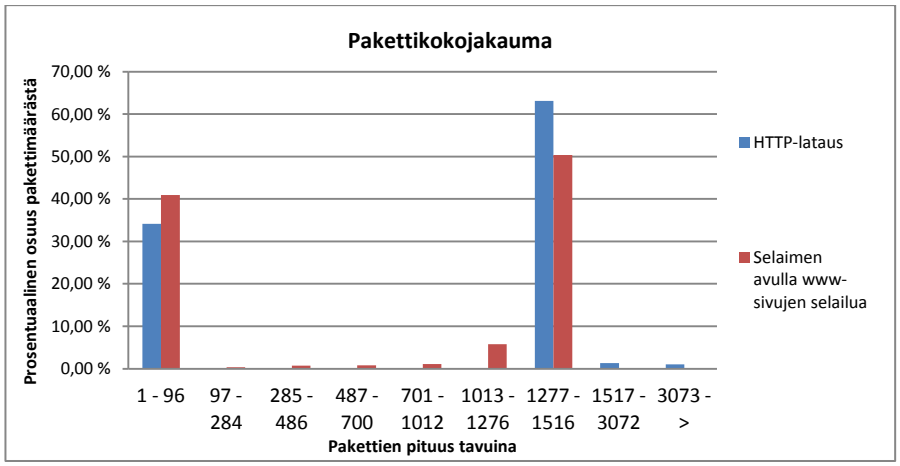
4.1 Liikenteen hyötykuorman ja liikennevuon tarkoituksenmukainen monimutkaistaminen

Protokollaobfuskoinnin yleinen tekniikka on salauksen käyttäminen datan satunnaistamiseksi. Salauksen avulla paketin hyötykuorma saadaan näyttämään satunnaiselta, lisäksi pakettikoko voidaan satunnaistaa lisäämällä satunnaista täytettä paketteihin. (Hjelmvik & John 2010.) Tällä pyritään hämäännyttämään pakettien syvätarkastusta sekä liikenteen tilastollista analyysia. Nämä ovat pääasiallisia tekniikoita, joiden avulla pyritään tunnistamaan sovelluskerroksen protokolla. Periaatteessa liikenteen hämäännyttäminen voidaan jakaa kahteen eri päämalliin: hämäännyttäminen poistamalla protokollalle tyyppilliset piirteet ja tyypillisten piirteiden poistamisen lisäksi liikenteen naamiointi joksikin muuksi liikenteeksi.

Avoimen lähdekoodin hämäännyttämisohjelmistoissa liikenteelle tyyppilliset piirteet pyritään kätkemään hyötykuorman hämäännyttämisen sekä liikennevuon hämäännyttämisen avulla (Hjelmvik & John 2010). Nämä menetelmät tukevat toisiaan, mutta vain harvat hämäännytysohjelmistot toteuttavat molemmat toiminnot. Käytännössä nykyiset avoimen lähdekoodin hämäännytysohjelmistot voidaan jakaa menetelmien mukaan ohjelmistoihin, jotka naamioivat liikenteen joksikin muuksi liikenteeksi ja ohjelmistoihin, jotka vain poistavat tyyppilliset piirteet, mutta eivät naamioi liikennettä näyttämään joltakin toiselta. Vain pienessä osassa suoritetaan sekä hyötykuorman että liikennevuon hämäännyttäminen. Naamiointi muuksi liikenteeksi suoritetaan tyypillisten piirteiden poistamisen jälkeen, mutta tässäkin harvoin suoritetaan molemmat toiminnot, joilla tyyppiliset piirteet poistetaan. Naamioinnissa voidaan käyttää esimerkiksi säännöllisiä lausekkeita, joiden avulla paketteihin lisätään haluttuja selväkielisiä merkkijonoja, jotta pakettien hyötykuormaa tutkiva ohjelmisto luokittelisi liikennevuon väärin.

Hyötykuorman hämäännyttämisessä tavoitteena on estää sovelluskerroksen protokollan tunnistamista tutkitun kuljetuskerroksen protokollan kuorman perusteella. Tyypillisesti kuorman hämäännyttäminen saavutetaan salauksen avulla. Kuorman sisältämä data saadaan näin satunnaiseksi. Liikennevuon hämäännyttämisessä pyritään tarkoituksenmukaisesti monimutkaistamaan liikennevuon tilastollisia ominaisuuksia, kuten esimerkiksi pakettikokoa ja saapuvien pakettien välistä aikaa eli ajastusta muutetaan. (Hjelmvik & John 2010.) Brumley ja Valkonen (2008) määrittelee liikenteen tarkoituksenmukaisen monimutkaistamisen siten, että obfuskointi toimii kääreenä sovelluskerroksen protokollalle. Kääreen tarkoitus on kätkeä itse sovelluskerroksen protokolla. Hyötykuorman tai liikennevuon tarkoituksenmukainen monimutkaistaminen satunnaistamisen avulla ei kuitenkaan ole aina riittävä toimenpide hämäämään liikennettä tutkivaa pakettien syvätarkastusta. Jos liikenne tulkitaan salakirjoitetuksi tai sen tunnistaminen ei onnistu, niin liikenne voidaan estää. Tämän vuoksi pelkästään sovellusprotokollan tunnistettavien piirteiden kätkeminen, ei välttämättä ole riittävä toimenpide. Tunnistamaton liikenne tulee myös saada näyttämään joltain toiselta liikenteeltä, joka on lähtökohtaisesti sallittu, esimerkiksi HTTP-liikenteeltä. Osa avoimen lähdekoodin obfuskointimenetelmistä toteuttaa vain hyötykuorman hämäännytyksen tai hyötykuorman ja liikennevuon hämäännytyksen. Ne eivät naamioi liikennettä näyttämään jonkin muun sovelluksen tuottamaksi liiketeeksi. Liikenteen naamiointi voidaan toteuttaa esimerkiksi muuttamalla liikennevuon tilastollisia ominaisuuksia, siten että obfuskoitu vuo muistuttaa esimerkiksi HTTP-liikenteen jakaumaa pakettikoon suhteen. Kuvassa 4 on esitetty testausympäristössä generoidun molempiin suuntiin kulkevan HTTP-liikenteen pakettikokojakaumat. Siniset pylväät esittävät Firefox-selaimen avulla ladatun FreeBSD.iso-tiedoston siirtämiseen tarvittavan verkkoliikenteen pakettikokojakaumaa ja punaiset pylväät esittävät ihmisen tuottaman verkon selausliikenteen pakettikokojakauman.

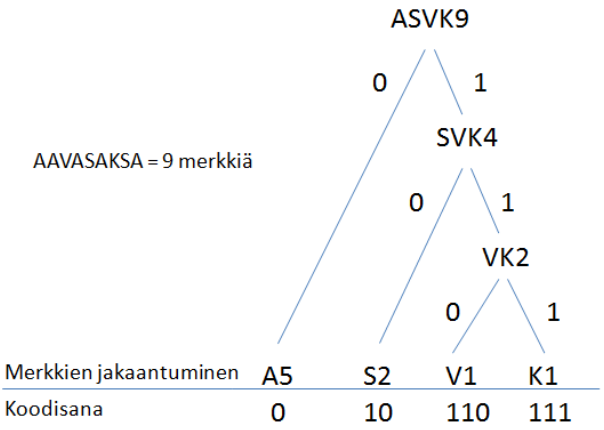
Edellä mainittu liikennevuon naamiointi voidaan toteuttaa esimerkiksi hyödyntämällä Huffman-koodausta. Tämä kuvataan tarkemmin seuraavassa luvussa.



Kuva 4. *HTTP-liikenteen jakauma pakettikoon suhteen*

4.2 Huffman-koodaus, entropia ja satunnaistaminen

Huffman-koodaus on tunnettu datan pakkausmenetelmä, joka perustuu eri mittaisiin koodisanoihin. Se on ns. optimi koodausmenetelmä, jonka avulla saadaan minimi redundanssi. Minimi toisto toteutetaan vaihtuva mittaisilla koodisanoilla, joiden pituus määräytyy esiintyvien merkkien todennäköisyyksien perusteella. Yleisimmin tai tiheimmin esiintyvät merkit koodataan lyhyimmillä koodisanoilla ja harvemmin esiintyvät merkit pidemmillä koodisanoilla. (Huffman 1952.) Kuvassa 5 on esitetty, kuinka lähtöjoukon eli sanan "AAVASAKSA" merkit kuvautuvat koodisanoiksi Huffman-koodauksessa. Useimmin esiintyvä eli todennäköisin merkki on esimerkissä "A" ja toiseksi todennäköisin merkki on "S". Merkki "A" koodataan lyhyimmällä bittijonolla, joka saa tässä esimerkissä arvon 0 ja "S" saa arvon 10. Vähiten esiintyvät merkit "V" ja "K" koodataan pisimillä bittijonoilla 110 ja 111. Käytännössä sanan "AAVASAKSA" siirtämiseen tarvittaisiin näin ollen 15 bittiä. Verrattaessa tulosta koodaustapaan, joka käsittelee merkkejä tavuina eli kahdeksan bittisinä alkioina, saadaan Huffman-koodauksen avulla sana "AAVASAKSA" siirrettyä lähes 5 kertaa tehokkaammin.



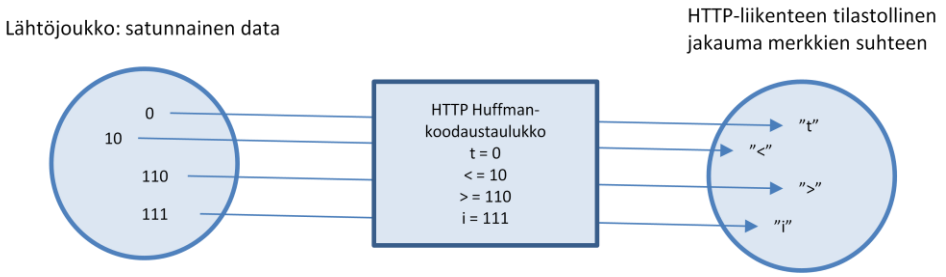
Kuva 5. *Lähtöjoukon kuvautuminen koodisanojen joukoksi Huffman-koodauksessa*

Taulukko, jonka avulla koodaus muodostetaan voidaan valita eri tavoin. Huffman-koodaus on ns. prefiksikoodi, joka tarkoittaa sitä, että mikään koodisana ei ole toisen prefiksi eli etuliite. Tämä tarkoittaa sitä, että koodi on aina yksiselitteisesti dekoodattavissa. (Ruckert 2005, s. 156.) Koska Huffman-koodaus on optimaalinen, sen avulla muodostettujen koodisanojen eli bittien sisältämä informaatio on myös optimaalinen. Toisin sanoen yhtä bittiä kohden informaation määrä on mahdollisimman korkea eli koodausprosessin entropia on korkea.

Informaatioteoriassa, joka on sovelletun matematiikan haara, entropian avulla voidaan mitata viestin sisältämän informaation määrää. Informaatioteoria pohjautuu mm. Claude E. Shannonin tutkimukseen: A Mathematical Theory of Communication, joka julkaistiin vuonna 1948. (Gray 2013.) Koodatun viestin sisältämä informaation määrä kasvaa bittiä kohden, kun viesti on pakattu siirtokanavalle. Mitä suurempi on informaation määrä yhtä bittiä kohden, sitä suurempi on myös viestin entropia. Huffman-koodauksen entropia on korkea, koska optimaalisessa pakkauksessa yksittäistä bittiä kohden informaation määrä kasvaa. Shannon tutki myös painetun englanninkielisen tekstin entropiaa vuonna 1951. Tällöin Shannon (1951) määritteli entropian seuraavasti: entropia on tilastollinen parametri, joka mittaa kuinka paljon informaatiota tuotetaan keskimäärin yhtä kirjainta kohden tietyn kielisessä tekstissä. Jos kieli muutetaan biteiksi tehokkaimmalla tavalla, entropia H on keskimääräinen bittien määrä, joka tarvitaan yhtä kirjainta kohden (Shannon 1951). Optimaalisessa pakkausmenetelmässä on mahdollisimman vähän toistoa, jonka seurauksena yhtä bittiä kohden informaation määrä kasvaa. Huffman-koodauksessa keskimääräinen bittien arvo kirjainta kohden on hyvin lähellä "parasta" entropiaa, itse asiassa Huffman-koodauksen osalta voidaan esittää, että $H \leq keskimääräinen\ koodin\ pituus \leq H + 1$, jossa H tarkoittaa entropiaa (Kulkarni 2002, s. 9).

Ruohonen (1999) määrittelee entropian epävarmuuden mittana, jolloin suuri entropia tarkoittaa epävarmempaa tulosta. Tästä johtuen korkea entropia viittaa myös satunnaisuuteen, joka on esimerkiksi salausalgoritmien tavoite. Salausalgoritmien tarkoituksena on järjestää viestin bitit mahdollisimman satunnaisesti, siten että alkiot (esim. tavujen arvot) noudattavat tilastollisesti tasaista jakaumaa (Duta, Mocanu, Vladescu & Gheorghe 2014, s. 31-32). Tästä päästään Ruohosen määrittelemään epävarmuuden mittaan, sillä satunnaislukuja on käytännössä mahdoton ennustaa. Satunnaislukujen sekvenssi omaa korkean entropian, mutta tämä ei välttämättä takaa kuitenkaan yksistään satunnaisuutta. Pakatun ja salatun informaation entropia voi olla lähes samat, mutta esimerkiksi gzip-ohjelmistolla pakattu tiedosto voi olla kuitenkin rakenteista (struktuuroitua), jolloin se ei ole satunnaista. (Lyda & Hamrock 2007.) You ja Tsai (1999) tutkivat Huffman-koodauksella pakatun datan satunnaisuutta. Kokeet osoittavat, että Huffman-koodaus tuottaa satunnaista dataa, jonka alkioiden tilastollinen jakauma on näin ollen tasainen.

Kuten edellä ja kuvassa 5 esitettiin, Huffman-koodaus tuottaa pakkauksen lisäksi merkkien tilastollisen jakauman. Näin ollen tietyn tilastollisen jakauman tuottamiseen voidaan soveltaa Huffman-koodausta. Toinen oleellinen tieto liikenteen hämäännyttämisen kannalta on edellä esitetty tulos, jossa todettiin, että Huffman-koodaus tuottaa satunnaista dataa. Tämä tarkoittaa sitä, että satunnaiseen dataan voidaan käänteisesti soveltaa Huffman-koodausta käyttämällä ennalta tunnettua kooditaulukkoa käänteisesti. (Wiley 2014a.) Satunainen data saadaan siis imitoimaan haluttua kohdemerkkijakaumaa. Sovellettaessa tätä verkkoliikenteeseen, pakettien hyötykuorma ensin salakirjoitetaan, jonka jälkeen siihen käytetään esimerkiksi HTTP-liikenteelle suoritettua Huffman-koodausta käänteisesti. Hyötykuormaan muodostuu samankaltainen merkkijakauma, kuin oikeassa HTTP-liikenteessä. Lopputuloksena syntyy siis hyötykuorma, joka imitoi HTTP-liikennettä, mutta sitä ei voida kuitenkaan tulkita oikein esimerkiksi verkkoselaimella. Kuvassa 6 on esitetty HTTP-liikenteen imitointi käänteisen Huffman-koodauksen avulla. Satunnaista bittijonoa käsitellään bitti kerrallaan, koska Huffman-koodisanat ovat vaihtelevan mittaisia. Bittien avulla muodostetaan koodaustaulukon mukaisesti merkkejä, joista muodostuu hyötykuormaan tavuja.



Kuva 6. *Käänteisen Huffman-koodauksen (dekoodaus) avulla muodostettava tilastollinen piirre*

Liikennevuon tai hyötykuorman satunnaistaminen ei välttämättä riitä hämäämään liikenteeseen aktiivisesti puuttuvaa sensoria. Esimerkiksi, jos pakettien syvätarkastusta suorittava laite toimii ns. "white-list" -moodissa, niin tunnistamaton tai kielletty liikenne estetään ja vain sallitut protokollat voivat liikennöidä. On selvää, että parhaaseen tulokseen hämäännyttämisen osalta päästään, jos molemmat, sekä liikennevuon tilastolliset piirteet että pakettien hyötykuorma voitaisiin naamioida jonkin muun sovellusprotokollan tuottamaksi. Käänteisen Huffman-koodauksen avulla saadaan muodostettua hämäännytettävään liikennevuohon jonkin kohdeprotokollan liikennevuon tilastollinen piirre, koska käänteisen koodauksen jälkeen hyötykuorman tavut muodostavat samanlaisen tilastollisen jakauman valitun kohdeprotokollan kanssa. Luonnollisesti toimenpiteen jälkeen hyötykuormaa ei voida kuitenkaan tulkita oikein. Usein liikennevuosta tarkistetaan vain ensimmäisten pakettien hyötykuorma, koska tunnistus pyritään suorittamaan mahdollisimman aikaisin. Jos tietty merkkijono löytyy ensimmäisestä paketista, tunnistus voidaan suorittaa kyseisestä paketista. Näin ollen käänteinen Huffman-koodaus yhdessä tunnettujen merkkijonojen lisäyksen kanssa muutamiin ensimmäisiin paketteihin voi muodostaa varsin tehokkaan hämäännytysmenetelmän.

5. AVOIMEN LÄHDEKOODIN HÄMÄÄNNYTTÄMISMENETELMÄT

Edellisessä luvussa esitetyt menetelmät ovat esimerkkejä tekniikoista, kuinka liikenteen hämäännyttäminen voidaan suorittaa. Tilastollisten piirteiden ja hyötykuorman hämäännyttämistä varten on toteutettu useita erilaisia avoimen lähdekoodin ohjelmistoja. Esimerkiksi bittorrent-protokollalle on kehitetty oma obfuskointimenetelmä. Tämä tunnetaan nimellä Message Stream Encryption (MSE). MSE menetelmästä käytetään myös nimitystä Protocol Header Encryption (PHE) ja se on saatavilla monelle eri P2P sovellukselle (Hjelmvik & John 2010). MSE kehitettiin tietoliikenneoperaattoreita vastaan, jotta nämä eivät kykenisi rajoittamaan tai estämään P2P liikennettä. (Brumley & Valkonen 2008.) Toinen yleisesti tunnettu hämäännyttämismenetelmä on Tor-projektin obfsproxy.

Tässä luvussa käsitellään tarkemmin avoimia tai avoimesti saatavilla olevia liikenteen hämäännyttämisohjelmistoja, sekä niissä esiintyviä liikenteen hämäännyttämistekniikoita ja -teknologioita.

5.1 MSE

MSE suunniteltiin tuomaan bittorrent-protokollalle turvallisuusominaisuuksia. Sen päätavoite on tarjota protokollaobfuskointia bittorrent-protokollalle, mutta sillä on myös muita tavoitteita, kuten tarjota liikennöiville osapuolille luottamuksellisuuden ja todennuksen. (Brumley & Valkonen 2008.) MSE tarjoaa satunnaiselta näyttävän protokollaotsikon sekä hyötykuorman, jonka avulla pyritään välttämään protokollan passiiviset tunnistamismenetelmät sekä liikenteen muokkaus. Jos hyötykuorman satunnaistamisessa käytetään RC4-salausalgoritmia, niin sen avulla saavutetaan myös kohtuullinen turvallisuus passiivista salakuuntelijaa vastaan. (VuzeWiki 2011.)

Viisivaiheinen MSE-kättely kahden liikennöivän osapuolen välillä alkaa Diffie-Hellman algoritmin avulla muodostettujen julkisten avainten vaihdolla. Julkisen avaimen lisäksi lähetetään 0-512 tavua satunnaista dataa täytteeksi, jotta avaimen vaihtoa ei tunnistettaisi protokollalle tyyppillisten piirteiden, kuten pakettikoon avulla. Diffie-Hellman-algoritmin eli D-H:n avulla voidaan muodostaa symmetrinen salausavain epäluotetun siirtokanavan ylitse siten, että kolmas osapuoli ei kykyene muodostamaan samaa salausavainta, vaikka salakuuntelisi siirtokanavaa. Tämän vuoksi D-H on erittäin yleinen salakirjoitusmenetelmä, joka on käytössä monissa sovelluksissa, joissa pyritään saavuttamaan luottamuksellisuus epäluotetun siirtotien ylitse. MSE-menetelmän D-H julkinen

avain ja symmetrinen salausavain muodostetaan kahden osapuolen A ja B välille seuraavasti:

- Valitaan yhteinen 768-bittinen alkuluku "P" ja kehitin "G", joka on aina kokonaisluku 2.
- A valitsee vähintään 128-bittisen yksityisen avaimen "Xa" ja B valitsee yksityisen avaimen "Xb". Yksityinen avain on satunnainen kokonaisluku.
- Tämän jälkeen voidaan laskea kummankin osapuolen julkiset avaimet:
    - Julkinen avain A: Ja = (G^Xa) mod P
    - Julkinen avain B: Jb = (G^Xb) mod P
- Liikennöivät osapuolet vaihtavat julkiset avaimensa siirtotien ylitse. Kumpkin osapuoli pystyy laskemaan nyt symmerisen salausavaimen (jaettu salaisuus), jonka avulla voidaan salata seuraavat kättelyn vaiheet:
    - D-H jaettu salaisuus: S = (Ja^Xb) mod P = (Jb^Xa) mod P

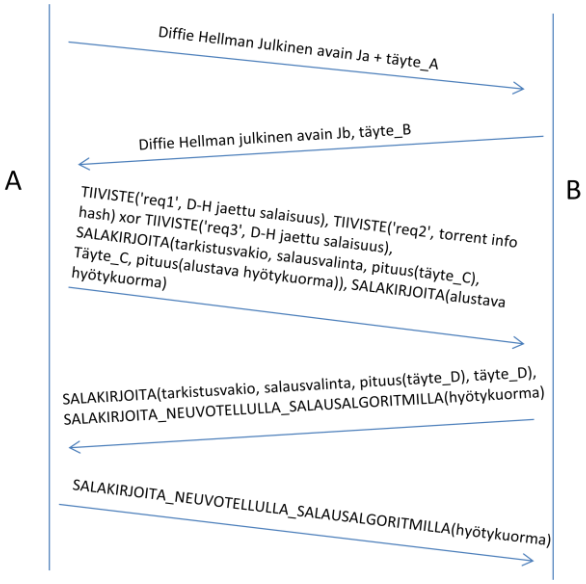
D-H:n avulla salakirjoitetaan "kryptokättelyn" (engl. crypto handshake) kolmas vaihe, joka on hyötykuorman salauksen neuvotteleminen. Jokaisen kättelyvaiheen paketteihin lisätään vaihtelevaa täytettä, jonka avulla pyritään hämäämään yksinkertaista pakettien pituusmallietsintää. Kättelyn kolmannen vaiheen salakirjoitus ei ole kovin selkeästi määritetty MSE:n spesifikaatiossa. Määrittely on seuraava:

- SALAKIRJOITA() hyödyntää RC4-algoritmia, joka käyttää yhtä seuraavista avaimista lähettäessä dataa:
    - A: TIIVISTE("avain_A", "D-H jaettu salaisuus", " torrent InfoHash").
    - B: TIIVISTE("avain_B", "D-H jaettu salaisuus", " torrent InfoHash"). (VuzeWiki 2011.)

Käytännössä RC4 tarvitsee symmetrisen avaimen, joten "avain_X" ei voi olla mahdollinen, koska sitä ei vaihdeta missään vaiheessa. Mahdolliset avaimet, joista tiiviste lasketan ovat siis D-H jaettu salaisuus ja torrent InfoHash-arvo. Hyödyntäessä MSE-menetelmää Bittorrent-protokollan kanssa, D-H todennus suoritetaan torrentin InfoHash-arvon avulla. MSE:n viisivaiheinen kättely on esitetty tarkemmin kuvassa 7.

MSE tarjoaa kaksi eri vaihtoehtoa protokollaobfuskoinnille: pelkkä protokollaotsikon satunnaistaminen tai koko liikennevuonsalaus RC4 algoritmilla. Luonnollisesti nämä vaihtoehdot vaikuttavat turvallisuuteen, nopeuteen ja itse hämäännyttämisen tasoon. Otsikkotiedon satunnaistamisessa tietokoneen suorittimen (CPU) kuorma on pienempi ja sovellustason protokollan käsittely nopeampaa verrattuna koko liikennevuon salaamiseen. Sovellustason protokollan tunnistaminen on kuitenkin helpompaa, koska itse hyötykuormaa ei salata. Koko liikennevuon salaaminen vaikeuttaa sovellustason protokollan passiivista tunnistamista, mutta nostaa CPU kuormaa. (VuzeWiki 2011.) On syytä tarkentaa, että edellä mainittu koko liikennevuon salaus tarkoittaa sovelluskerroksen salaamista eli liikennöivän sovellustason protokollan otsikkotiedon ja sovelluksen tuot-

taman hyötykuorman salaamista, ei OSI-mallin alempien kerroksien, kuten verkkokerroksen tai kuljetuskerroksen salaamista. Toisin sanoen bittorrent-protokollan tapauksessa IP-protokollan ja TCP- tai UDP-protokollan otsikkotiedot ovat selväkielisiä.



Kuva 7. *MSE:n viisivaiheinen kättely (VuzeWiki 2011)*

Edellä kuvattujen tekniikoiden avulla MSE kykenee siirtämään satunnaiselta näyttävää dataa. Yksinkertaistettuna tämä tapahtuu siis muuttamalla pakettikokoa ja pakettien prosessointiaikaa sekä käyttämällä salausalgoritmia. On hyvä terävöittää, että MSE-obfuskointimenetelmässä käytettävät kryptograafiset metodit ovat kuitenkin melko yksinkertaisia. Siinä ei ole kehittyneitä kuljetuskerroksen turvallisuusmekanismeja, kuten esimerkiksi TLS-protokollassa. Tämä johtuu siitä, että MSE:n tavoitteet eivät ole turvallisuuden maksimoinnissa, vaan sen tavoitteena on kyseistä menetelmää toteuttavien järjestelmien kuorman minimointi ja nopeat kryptograafiset metodit. (VuzeWiki 2011.) Näiden syiden vuoksi MSE-menetelmässä on myös useita haavoittuvuuksia. Brumley ja Valkonen (2008) ovat tutkineet menetelmän haavoittuvuuksia. Tutkimuksessa osoitetaan, kuinka esimerkiksi välimieshyökkäys on mahdollista toteuttaa heikon todennuksen vuoksi.

5.2 Obfsproxy ja muut Tor-projektille kehitetyt obfuskointiohjelmisot

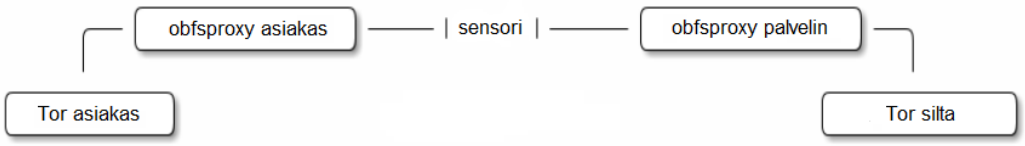
Obfsproxy on työkalu, joka alunperin kehitettiin Tor-verkolle. Se ei ole obfuskointiprotokolla, kuten MSE, vaan se on ohjelmisto, jonka avulla voidaan hämäännyttää liikennettä hyödyntämällä tiettyjä obfuskointiprotokollia. Sen alkuperäisenä tarkoituksena on ollut ohittaa liikenteen luokittelua suorittavat ohjelmistot muuntamalla Tor-liikennettä asiakaan ja sillan välillä (Tor Project a). Idea on sama kuin MSE-menetelmässä tai yleisesti liikenteen tarkoituksenmukaisessa monimutkaistamisessa. Tor-liikenne yritetään obfuskoida siten, että verkkosensori ei pysty tunnistamaan liikennöivää sovellusta tai

sovelluskerroksen protokollaa (Tor Project a). Hämäännytys saadaan aikaiseksi käyttämällä samankaltaisia menetelmiä kuten MSE:n tapauksessakin. Liikenteelle tyyppilliset piirteet pyritään kätkemään hyötykuorman hämäännyttämisen sekä liikennevuon hämäännyttämisen avulla. Tosin vain yksi obfsproxyn päällä toimivista obfuskointiprotokollista toteuttaa molemmat edellä mainitut hämäännyttämistoiminnot.

Obfsproxy toimii välityspalvelimena Tor-asiakkaan ja Tor-sillan välissä. Tor-asiakkaalla tulee olla myös Obfsproxy-asiakassovellus (kuva 8). Obfsproxy versio 0.2.13 tukee kolmea eri obfuskointimenetelmää eli obfuskointiprotokollaa, näitä kutsutaan myös kuljetusliitännäisiksi (engl. Pluggable Transports). (Tor Project a.) Itse asiassa tuettuja kuljetusliitännäisiä on yhteensä viisi, mutta vain kolme näistä tuottaa obfuskoinnin liikenteelle. Obfsproxyn tukemat kuljetusliitännäiset ovat:

- dummy
- b64
- obfs2
- obfs3
- scramblesuit.

Näistä kolme viimeksi mainittua ovat obfuskointiprotokollia. "Dummy-kuljetusliitännäinen" toimii ainoastaan välityspalvelimena asiakkaan ja palvelimen päissä. Tämä on tarkoitettu ainoastaan obfsproxyn testaamiseen. B64 suorittaa lähetettävälle datalle base64-koodauksen, joka ei täytä obfuskoinnin eli hämäännyttämisen määritelmää. Obfs2, obfs3 ja scramblesuit tuottavat obfuskointikerroksen TCP-protokollille.



Kuva 8. *Obfsproxy*

Obfsproxy-ohjelmiston lisäksi Tor-projektilla on myös muita obfuskointiohjelmia sekä obfuskointiprotokollia. Näitä on kehitetty useaan eri käyttötarpeeseen ja tätä kirjoittaessa Tor mainitsee esimerkiksi Format-Transforming Encryption -ohjelmiston (FTE), StegoTorus-ohjelmiston, SkypeMorph-ohjelmiston ja Dust-ohjelmiston. Näistä vain ensiksi mainittu on valmis ja muut ovat vielä kehitysvaiheessa. (Tor Bug Tracker & Wiki 2015a). Obfsproxy ja sen tukemat kuljetusliitännäiset eivät naamioi liikennettä näyttämään jonkin muun sovelluksen tuottamaksi liikenteeksi. Edellä esitetyt ohjelmistot eroavat obfsproxyn tukemista kuljetusliitännäisistä, sillä ne pyrkivät saamaan obfuskoitavan liikenteen näyttämään esimerkiksi HTTP-liikenteeltä tai Skype-videolta. Esimerkiksi FTE-asiakasohjelmiston avulla obfuskoitava liikenne salakirjoitetaan, jonka jälkeen pakettiin lisätään selväkielinen merkkijono, kuten "GET http://satunnainen.html

HTTP/1.1". FTE-palvelinohjelmisto vastaa lisäämällä obfuskoitavan liikenteen pakettiin selväkielisen merkkijonon "HTTP/1.1 200 ok", joka näyttää HTTP-palvelimen tuottamalta vastaukselta. (Tor Project a.) Tämä menetelmä on tehokas erityisesti säännöllisiä lausekkeita käyttäviin DPI-ohjelmistoihin, koska tunnistus tehdään näissä pakettien hyötykuormassa olevien merkkijonojen perusteella. FTE-ohjelmisto esitellään tarkemmin luvussa 5.3. Myös Dust-ohjelmisto esitetään tarkemmin luvussa 5.5, koska tässä voidaan hyödyntää muuttuvaa liikenneprofiilia obfuskoitavalle liikenteelle. Muuttuva liikenneprofiili tekee Dust-ohjelmistosta erityisen mielenkiintoisen hämäännyttämismenetelmän. Muihin edellä mainittuihin ohjelmistoihin ei perehdytä tarkemmin tässä työssä, koska niiden kehittäminen on vielä kesken.

Vaikka obfsproxy on alunperin kehitetty Tor-verkolle, sitä pystyy käyttämään myös muidenkin kuin Tor-sovellusten kanssa. Esimerkiksi SSH- ja OpenVPN-liikennettä voidaan obfuskoida sen avulla (Into the Void 2012; Othman 2013; OpenVPN Community 2013). Obfsproxy käyttää tapahtumapohjaista verkkomoottoria, joka on kirjoitettu Python-ohjelmointikielellä. Verkkomoottoria kutsutaan nimellä Twisted. Obfsproxy tarvitsee toimiakseen myös Pyptlib-kirjastoa joidenkin kuljetusliitännäisten ominaisuuksien vuoksi.

5.2.1 Obfs2-obfuskointiprotokolla

Kuten edellä todettiin, Obfsproxyn versio 0.2.13 tukee tällä hetkellä kolmea eri obfuskointimenetelmää eli obfuskointikerrosta. Tässä alaluvussa keskitytään obfs2-obfuskointiprotokollaan, joka on obfsproxy-ohjelmiston päällä ajettava "älä-näytämiltään" -kuljetusliitännäinen (Tor Bug Tracker & Wiki 2015b). Kaikki obfsproxyn päälle kehitetyt obfuskointiprotollat perustuvat samaan perusajatukseen. Näiden tarkoituksena ei siis ole naamioida liikenettä näyttämään jonkin muun sovelluksen liikenteeltä.

Obfs2-protokollan uhkamallissa on kuvattu protokollalle asetetut tavoitteet sekä asiat, jotka eivät ole olleet tavoitteena, kun protokollaa on suunniteltu. Näistä on hyvä tuoda esille muutama kohta, jota obfs2-protokolla ei toteuta. Se ei kykene esimerkiksi suojaamaan liikennevuon analyysiltä. Esimerkiksi tilastollisen analyysin avulla, jossa tutkitaan pakettikokoa tai ajastusta, tulisi kyetä luokittelemaan obfs2-protokolla. Se ei myöskään suojaa liikenteen entropian mittaukselta. (Kadianakis & Mathewson 2013.) Kuten aikaisemmin on esitetty korkea entropia merkitsee usein satunnaistettua, salakirjoitettua tai pakattua liikennettä.

Edellisen perusteella voidaan todeta, että obfs2 pyrkii hämäännyttämään vain liikenteen hyötykuorman, mutta ei liikennevuota. Koska ofbs2 kehitettiin ensimmäisenä obfsproxyn tukemista obfuskointiprotokollista, niin siinä ei ole osattu ottaa huomioon liikenteen luokittelun näkökulmasta tiettyjä asioita. Näistä yksi on protokollan suorittama yhteydenmuodostuskättely, jossa on tunnistettavat piirteet. Tämä on yksi syy, jonka

vuoksi obfs2-protokolla on luokiteltavissa. Edellä mainituista syistä johtuen voidaankin todeta, että obfs2 on vanhentunut. Obfs2-protokollan yhteydenmuodostuskättely esitetään yksityiskohtaisemmin alla. Tätä varten määritetään ensin muutama primitiivi:

- UINT32 = neljän tavun arvo big-endian muodossa
- taika-arvo (engl. macig value) = 0x2BF5CA7E
- S(A, m) on AES-CTR-128 salakirjoitus merkkijonosta m käyttäen avainta A
- tiiviste(x) on SHA256 hash-arvo x:stä
- MAC-funktio = MAC(s,x) = Tiiviste(s + x + s)
- obfuskointi_täyte = "Initiator obfuscation padding" tai "Responder obfuscation padding".

Viimeisimpänä mainittu "obfuskointi_täyte" on obfs2-ohjelmistoon kovakoodattu merkkijono. Yhteyden avaaja ja vastaaja käyttävät niille tarkoitettuja erillisiä merkkijonoja. Obfs2-kättely etenee seuraavasti sekä yhteyden avaajan että vastaanottajan puolesta (ts. kumpikin osapuoli suorittaa seuraavat toimenpiteet):

- Siemen (engl. seed) = generoidaan 16 tavua vahvaa satunnaisdataa.
- Täyteavain (engl. pad_key) = 16 ensimmäistä tavua MAC-funktiosta:
    - MAC(obfuskointi_täyte, siemen) = SHA256(obfuskointi_täyte + siemen + obfuskointi_täyte).
- Generoidaan satunnaisnumero PADLEN, joka saa arvon väliltä 0 - 8192.
- Lähetetään yhdistelmä arvoista:
    - siemen, S(täyteavain, UINT32(taika-arvo) , UINT32(PADLEN), PADLEN tavumäärä heikkoa satunnaisdataa).

Saatuaan "siemenen", kumpikin osapuoli johtaa toisen osapuolen täyteavaimen ja purkaa salauksen seuraavasta kahdeksasta tavusta. Jos "taika-arvo" ei täsmää tai PADLEN-arvo on suurempi kuin maksimi PADLEN, eli 8192, vastaanottava osapuoli sulkee yhteyden välittömästi. Muutoin tämä lukee jäljellä olevan PADLEN-arvon verran täytedataa ja hylkää nämä. (Tor Project b.)

Kättelyn jälkeen johdetaan lisäavaimet. Tässä "merkkijono" on jälleen kovakoodattu merkkijono riippuen liikennöivästä osapuolesta. Yhteyden avaaja käyttää merkkijonoa "Initiator obfuscated data" ja vastaanottaja "Responder obfuscated data". Lisäavaimet johdetaan seuraavasti A ja B osapuolen välillä (A = lähettäjä, B = vastaanottaja):

- A_salaisuus = MAC(A_merkkijono, A_siemen ja B_siemen)
- B_salaisuus = MAC(B_merkkijono, A_siemen ja B_siemen)
- A_avain = 16 ensimmäistä tavua arvosta "A_salaisuus"
- A_IV = 16 jälkimmäistä tavua arvosta "A_salaisuus"
- B_avain = 16 ensimmäistä tavua arvosta "B_salaisuus"
- B_IV = 16 jälkimmäistä tavua arvosta "B_salaisuus". (Tor Project b.)

"X_salaisuus-arvo" on avaimena vuosalauksessa, jonka avulla salakirjoitetaan viestit tästä eteenpäin. On hyvä huomioida, että AES CTR-moodissa muuntaa käytännössä lohkosalauksen vuosalaukseksi, koska se luo seuraavan avainvuolohkon laskurin ja käynnistysvektorin avulla (Housley 2004). "X_IV-arvo" toimii vuosalauksessa käynnistysvektorina. Kummallakin osapuolella on siis omat avaimet. Obfs2 tukee myös jaettua salaisuutta (engl. shared secret), jonka käyttö ei ole pakollista. (Tor Project b.) Käytettäessä jaettua salaisuutta, sen toiminta on mukana MAC-funktiossa seuraavasti:

- MAC(s,x) = Tiiviste^n(s + x + Tiiviste(jaettu_salaisuus) + s), jossa "tiiviste^n" on n iteraatiota SHA256-tiivistefunktiosta. Arvo "n" on tässä tapauksessa 100000. (Tor Project b.)

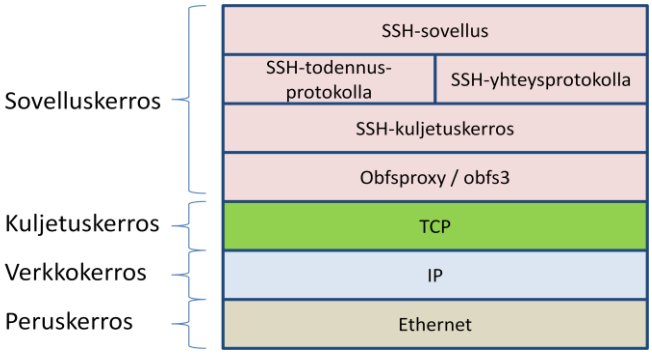
Obfs2-protokolla on kättelystä johtuen haavoittuvainen välimieshyökkäykselle ja itse asiassa pelkästään passiivinen liikenteen tarkkailija kykenee johtamaan avaimet, koska siemen-arvo lähetetään selväkielisenä verkon yli. Tällä hetkellä on olemassa useita esimerkkejä siitä, kuinka esimerkiksi python-koodin avulla johdetaan obfs2-salausavain. Eräs esimerkki esitellään Tor wikissä, jonka python-koodi esitetään liitteessä A. (Tor Bug Tracker & Wiki 2015b.) Luonnollisesti jaetun salaisuuden käyttäminen lisää yksityisyyttä protokollan toimintaan. Vaikka protokollan toiminta tunnettaisiin, on selvästi vaikeampaa saada salausavaimet selville, jos jaettua salaisuutta ei tunneta.

Obfs2 on myös haavoittuvainen "aktiiviselle luotaukselle" (engl. active probing). Aktiivinen luotaus tarkoittaa yksinkertaisuudessaan sitä, että esimerkiksi obfsproxy-palvelinohjelmiston socket-rajapinnasta luetaan 16 tavua, jonka avulla johdetaan edellä esitetty täyteavain. Täyteavaimen avulla avataan salaus seuraavasta neljästä tavusta ja jos tavut vastaavat taika-arvoa, on kyseessä obfs2-protokolla. Teoriassa palvelin tarvitsee asiakkaan ensin lähettämään kättelydataa, mutta tosiasiassa python-toteutus obfsproxy-ohjelmistosta ei tätä vaadi. Näin ollen voidaan käyttää skriptiä, joka vain lukee socket-rajapinnasta ja ei itse asiassa lähetä mitään. (Fifield 2015.) Fifield (2015) on laatinut esimerkki-ohjelman, jonka avulla voidaan suorittaa aktiivista luotausta. Ohjelma on esitety liitteessä B.

On havaittu, että Kiinan GFW eli "suuri palomuuri" kykenee estämään obfs2-protokollan avulla obfuskoidun liikenteen. Aktiivisen luotauksen perusteella tunnistetaan obfsproxy-palvelinohjelmisto ja estetään tähän lähetettävät TCP SYN -paketit. (Tor Project 2013.) Peruste luotaukselle voi olla esimerkiksi anomalia, joka havaitaan verkkoliikenteestä pakettien syvätarkastuksen avulla. Tälläinen anomalia eli poikkeavuus syntyy esimerkiksi siitä, jos naamioimatonta obfuskoitua liikennettä ajetaan HTTP-porttiin. HTTP-liikenne, joka ei sisällä protokollalle tyyppilisä ominaisuuksia, kuten HTTP GET -pyyntöä tai palvelimen vastausta, on selvä anomalia. Kiinasta käsin on havaittu on myös täysin satunnaista obfsproxy-ohjelmiston luotausta muun muassa TCP 80-, TCP 443- ja TCP 25 -portteihin (Tor Project 2013).

5.2.2 Obfs3

Obfs3, joka tunnetaan myös nimellä "The Threebfuscator", on paranneltu versio obfs2-protokollasta. Siinä on parannettu esimerkiksi kättelyssä tarvittavia menetelmiä, jotta sitä ei kyettäisi erottamaan täysin satunnaisesta bittijonosta. Kuten aikaisempi versio, myös obfs3 on obfuskointikerros TCP:tä käyttäville sovellusprotokollille. Tämä sekä aiempi versio obfs2, eivät varmista datan eheyttä, eivätkä tarjoa todennusta. Obfs3 ei myöskään kätke datan pituuksia, joten se ei edeltäjänsä tapaan hämäännytä liikennevuota. Näiden syiden vuoksi obfs3 soveltuu hyvin protokollien, kuten SSH tai TLS obfuskointikerrokseksi, koska nämä protokollat tarjoavat todennuksen. Kuvassa 9 havainnollistetaan obfuskointikerroksen ja SSH-protokollan sijainnit TCP/IP-kerrosmallissa. Obfuskointikerroksella on kaksi vaihetta. Ensimmäisessä vaiheessa osapuolet muodostavat salausavaimet, joiden avulla salakirjoitetaan symmetristen salausavainten vaihto. Toisessa vaiheessa vaihdetaan salakirjoitettua liikennettä. (Tor Project c.)



Kuva 9. *Obfuskointikerroksen ja SSH-protokollan sijainnit TCP/IP-kerrosmallissa*

Kuten MSE-menetelmässä, myös obfs3 käynnistää yhteyden neuvottelemalla Diffie-Hellman algoritmin avulla salausavaimet. Neuvottelu eroaa kuitenkin MSE:n käyttämästä D-H algorimista, sillä obfs3 käyttää räätälöityä D-H algoritmia, jota kutsutaan tasajakauma D-H:ksi (engl. uniform D-H). Tämän avulla pyritään muodostamaan kättely, jota ei kyetä erottamaan täysin satunnaisesta bittijonosta. Tasajakauma D-H käyttää RFC3526:n mukaista 1536-bittistä MODP-ryhmää, eli ryhmää 5. Tämä tarkoittaa sitä, että yhteinen valittava alkuluku P on 1536-bittinen. P toimii siis moduluksena eli renkaan kokona modulaariaritmetiikassa, jonka avulla lasketaan D-H:n julkinen avain ja jaettu salaisuus. (Tor Project c.)

Alla on esitetty tarkemmin tasajakauma D-H:n toiminta:

- Valitaan satunnainen 1536-bittinen luku ja asetetaan vähiten merkitsevä bitti nollaksi, jotta saadaan parillinen luku. Tämä luku toimii yksityisenä avaimena, joka A:n tapauksessa merkitään Xa. Näin ollen julkinen avain "Ja" lasketaan seuraavasti:
    - A: Ja = G^Xa (mod p)
    - B: Jb = G^Xb (mod p).

- Kun A:n julkinen avain lähetetään toiselle osapuolelle, valitaan satunnaisesti joko "Ja" tai "p-Ja", jolloin julkinen avain on merkityksettömän vähän erilainen verrattuna tasaisesti jakautuneeseen 1536-bittiseen merkkijonoon. B:n tapauksessa lähetetään luonnollisesti "Jb" tai "p-Jb".
- Jaettu salaisuus lasketaan samalla tavalla kuin MSE-menetelmässä. Tässä on kuitenkin hyvä huomioida seuraava:
    - (p-Jb)^Xa = Jb^Xa (mod p) ja (p-Ja)^Xb = Ja^Xb (mod p), koska Xa and Xb ovat parillisia. (Tor Project c.)

Räätälöityyn vaihtoehtoon on päädytty, koska perinteinen D-H algoritmi voidaan erottaa samanpituisista satunnaisista merkkijonoista. Tasajakauma D-H -protokollan avulla voidaan tuottaa julkiset avaimet, jotka näyttävät satunnaisilta merkkijonoilta. Tasajakauma D-H avaintenvaihdon jälkeen osapuolet luovat jaetun salaisuuden, jonka perusteella johdetaan symmetrinen salausavain yhteysistuntoa varten. Liikenteen salauksessa käytetään AES algoritmilla toteutettua lohkosalausmenetelmää CTR-moodissa ja salausavaimen pituus on 128-bittiä. (Tor Project c.)

Obfs3 pyrkii edellä esitetetyllä tavalla kätkemään yhteydenmuodostuskättelyn, jotta liikennettä ei kyettäisi luokittelemaan. Kun kättelyssä käytettävät paketit, eli tarkasteltava bittijono näyttää tasaisesti jakautuneelta, liikenne näyttää samalta kuin se olisi salakirjoitettu. Tässä tapauksessa on haastavaa päätellä salakirjoitustapa, koska D-H:n tunnusomaiset piirteet on saatu kätkettyä. Obfs3 ei ole obfs2-protokollan tapaan haavoittuvainen passiiviselle tunnistukselle tasajakauma D-H:n vuoksi, mutta se voidaan kuitenkin tunnistaa aktiivisen välimieshyökkäyksen tai aktiivisen luotauksen avulla. Ongelma on D-H:n avainten vaihdossa. Sen tulee näyttää tasaisesti satunnaiselta, kuten muunkin osan liikenteestä. Valittaessa alkulukua p, käytetään aliryhmää $\mathbb{Z}p^*$, joka on turvallisten alkulukujen ryhmä (engl. safe prime). Karkeasti vain noin puolet kokonaisluvuista nollan ja p:n välillä, kuuluvat ryhmään $\mathbb{Z}p^*$. Tarkkailtaessa avainten vaihtoa, on mahdollista, että sensori kykenee havaitsemaan, että lähetetyt luvut eivät ole oikeasti satunnaisia, vaan kuuluvat aina valitttuun aliryhmään. Tämän avulla sensori voi päätellä, että lähetetyt tavut eivät ole satunnaisia, vaan kyseessä on avaintenvaihto. (Tor Bug Tracker & Wiki 2015b.)

5.2.3 Scramblesuit

Edellä mainitut obfsproxyn tukemat obfuskointiprotokollat ovat haavoittuvaisia "aktiiviselle luotaukselle" (engl. active probing) tai ne voidaan tunnistaa liikennevuon allekirjoituksesta, eli liikennevuolle tyyppillisistä tilastollisista ominaisuuksista. Kuten todettu, pakettikoko on yksi näistä ominaisuuksita.

Scramblesuit-kuljetusliitännäinen kehitettiin taklaamaan juuri edellä esitetyt ongelmat. Kyseisiä ongelmia varten on kehitetty neljä ominaisuutta, joista hyötykuorman satun-

naistaminen sisältyy myös edellä esitettyihin obfuskointiprotokolliin eli kuljetusliitännäisiin. Pääominaisuudet ovat:

- hyötykuorman satunnaistaminen
- liikennevuon obfuskointi
- jaetun salaisuuden käyttäminen todennuksessa
- integrointi Tor-verkon obfuskointiekosysteemeihin. (Winter, Pulls & Fuss 2013.)

Obfs2- ja obfs3-protokollien tapaan, se toimii TCP-protokollan päällä omana obfuskointikerroksena. Koska scramblesuit on obfsproxy-ohjelmiston yksi kuljetusliitännäinen, se kykenee siirtämään mitä tahansa sovellusta, joka tukee SOCKS-välityspalvelinta. Tämä pätee kaikille kuljetusliitännäisille, jotka toimivat obfsproxyn kanssa, sillä obfsproxy sisältää SOCKS-välityspalvelimen.

Kuten useaan otteesseen on jo todettu, hyötykuorman satunnaistamisen avulla bittijonot näyttävät tarkkailijan näkökulmasta satunnaisilta. Tuloksena ei siis pitäisi syntyä ennalta arvattavissa olevia toistumakuvioita. Tämän tulisi hämäännyttää pelkästään säännöllisiin lausekkeisiin perustuvat DPI-kirjastot. Scramblesuit-kuljetusliitännäinen on ainoa obfsproxyn päällä toimiva obfuskointiprotokolla, joka obfuskoi myös liikennevuon. Se kykenee muuttaamaan ajastusta ja pakettikokoa. Jaetun salaisuuden avulla pyritään estämään aktiivinen luotaus. Palvelin vastaa vain pyyntöihin, jotka tulevat jaetun salaisuuden avulla todennetulta asiakkaalta. (Winter, Pulls & Fuss 2013.)

Scramblesuit-protokolla hyödyntää edellä esitettyjen obfs2- ja obfs3-protokollien tekniikoita saavuttaakseen edellä esitetyt tavoitteet. Tosin monia tekniikoita on hieman parannettu, jotta scramblesuit ei olisi haavoittuvainen passiiviselle tai aktiiviselle luotaukselle. Esimerkiksi se hyödyntää tasajakauma D-H:ta, jota on hieman muokattu paremmaksi, koska se ei suojaa alkuperäisessä muodossa aktiiviselta luotaukselta. Pelkkä anonyymi kättely ei siis riitä. Tämän vuoksi kättelyä on kehitetty siten, että siinä on mukana todennus. Todennus toteutetaan lisäämällä D-H:n julkisten avainten vaihtoon mukaan satunnainen täyte P ja MAC-arvo. MAC-arvo muodostetaan laskemalla SHA256-tiiviste viestistä ja jaetusta salaisuudesta k_B. Tor-liikenteen osalta tämä salaisuus jaetaan yhdessä Tor-sillan IP-osoitteen ja porttinumeron kanssa, joko sähköpostitse tai HTTPS-yhteyden kautta. Asiakas ja palvelin tunnistavat vastaanottaneensa koko kättelyviestin, kun viimeiseksi vastaanotetut tavut muodostavat oikean MAC-arvon edellä olevista tavuista. Salaisuus k_B voidaan käyttää uudelleen, koska sitä käytetään ainostaan avaimena MAC-funktiossa. Kättely muodostetaan siis käyttämällä tasajakauma D-H:ia, jotta saadaan satunnaiselta näyttävät julkiset avaimet ja näiden vaihtaminen todennetaan edellä esitetyllä tavalla. Peräkkäiset tasajakauma D-H-kättelyviestit, jotka hyödyntävät samaa kB-arvoa, näyttävät sensorin näkökulmasta erilaisilta. Mahdollinen toistohyökkäys estetään lisäämällä MAC-arvoon aikaleima E, joka on Unix-aika

jaettuna 3600:lla. Kättelyviestin MAC-arvo myös varastoidaan yhden tunnin ajan. (Winter, Pulls & Fuss 2013.)

Scramblesuit tarjoaa myös muita parannuksia, mutta kuten edellä on todettu se on ainoa obfsproxyn tukema obfuskointiprotokolla, joka mahdollistaa liikennevuon ominaisuuksien hämäännyttämisen. Kehityksen tavoitteena on ollut toteuttaa "kevyt-versio" liikennevuon obfuskoinnista, jonka avulla pienennetään luokittelun riskiä. Liikennevuon obfuskointi toteutetaan kaksijakoisesti. Pakettien pituuksia ja pakettien saapumisen välistä aikaa (engl. inter arrival times) pyritään muokkaamaan. Yleisesti käyttöjärjestelmän ytimen (engl. kernel) TCP-pino on vastuussa pakettien pituuksista. Jotta pakettien pituuksiin voidaan vaikuttaa käyttäjätilassa (engl. user space), joudutaan deaktivoimaan Naglen algoritmi. Algoritmi pyrkii estämään tarpeettomien pienten TCP-segmenttien lähetyksen. Luonnollisesti tämän poistaminen aiheuttaa "lisäkustannuksia" TCP:n tehokkuuteen. (Winter, Pulls & Fuss 2013.)

Pakettien pituuksien ja ajastuksen satunnaistaminen perustuu satunnaisesti generoituun diskreettiin todennäköisyysjakaumaan. Jakaumat muodostetaan valitsemalla ensin n kappaletta säiliöitä joukosta {1..100}. Jokainen säiliö on yhtä todennäköinen. Säiliöiden määrän määrittämisen jälkeen jokaiselle säiliölle merkitään arvo vastaavan todennäköisyyden kanssa, joka on väliltä]0,1]. Arvolla tarkoitetaan pakettikokoa tai viivästysaikaa ennen kuin paketti liipaistaan verkkoon. Luonnollisesti kaikkien säiliöiden todennäköisyyksien summa on 1. Kun tätä todennäköisyysjakaumaa sovelletaan pakettien pituuksien satunnaistamiseen, jokainen valittu säiliö todennäköisyysjakaumassa on yhtä todennäköinen ja säiliöiden arvot valitaan väliltä {21..1448}. (Winter 2014.) Luotua jakaumaa hyödynnetään satunnaisesti jokaiseen palaan sovellusdataa, jonka scramblesuit lähettää siirtolinjalle. Riippuen sovellettavasta pituudesta, pakettiin lisätään täytettä tai se pilkotaan. (Winter, Pulls & Fuss 2013.)

Ajastuksen satunnaistamisen mekanismi on sama kuin pakettien pituuksien muuntamisessa. Luodaan satunnaisjakauma, josta vedetään satunnaisia näytteitä, jotka toimivat parametrina lyhyille "nukkumiskutsuille" (engl. sleep calls). Näytteet eli säiliöiden arvot valitaan väliltä {0..0,01}. (Winter 2014.) Nukkumiskutsu suoritetaan ennen kuin paketti liipaistaan linjalle. Maksimi paketin viivästysarvo on tällöin 10ms. Tätä arvoa voidaan tosin muuttaa scramblesuit-koodissa. Luonnollisesti pakettien välistä saapumisaikaa ei voida lyhentää. On myös selvää, että kasvattamalla tätä aikaa, pienennetään linjan läpäisykykyä. (Winter, Pulls & Fuss 2013.)

5.3 Fteproxy

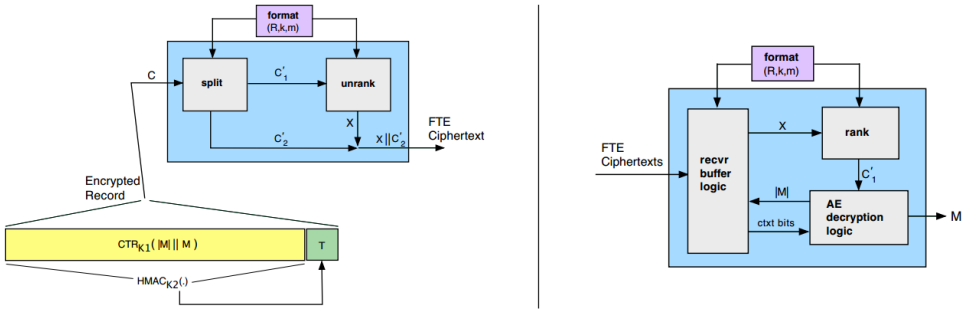
Fteproxy eli "Format-transforming encryption proxy" on obfuskointiohjelmisto, joka pyrkii naamioimaan sovelluksen liikenteen jonkin toisen sovelluksen liikenteeksi. Tähän viitataan ohjelmiston nimessäkin, joka suomennettuna tarkoittaa muodonmuutossalakirjoitusta. Naamiointi toteutetaan hyötykuorman obfuskoinnilla, jonka lisäksi hyöty-

kuormaan lisätään halutun sovelluksen tuottamia selväkielisiä merkkijonoja. Hyötykuorman obfuskoinnissa käytetään samankaltaisia menetelmiä, kuin edellä esitetyissä menetelmissä, eli salakirjoitusta. Tosin tässä käytettävä salakirjoitustapa pyrkii säilyttämään tietyn muodon.

Fteproxy on kokonainen obfuskointiohjelmisto, koska se sisältää välityspalvelimen ja fte-obfuskointikerroksen. Obfuskointikerros on toimintaperiaatteeltaan sukua "Format-preserving encryption" -menetelmälle, jossa ideana on salakirjoittaa viesti siten, että sen merkkimäärä ja muoto pysyy samanlaisena. (Dyer, Coull, Ristenpart & Shrimpton 2013.) Esimerkiksi luottokorttinumero salakirjoitetaan siten, että tuloksena syntyy yhtä pitkä numerosarja (Bellare, Ristenpart, Rogaway & Stegers 2009). Fte tarvitsee viestin salakirjoittamiseen symmetrisen avaimen K, muodon F ja viestin M. Muoto F määrittelee joukon L(F), jota kutsutaan F:n kieleksi. Salakirjoituksen ulostulona on salakirjoitettu viesti C, joka kuuluu L(F)-joukkoon.

Fte tukee säännöllisten lausekkeiden muotoa. Tämä mahdollistaa helpon muotojen, eli formaattien "ohjelmoinnin". Samalla se myös tuottaa tehokkaan menetelmän obfuskoinnin näkökulmasta, koska formaattia voidaan vaihtaa pienellä vaivalla. Jotta voitaisiin implentoida salakirjoitus E(K, F, M) säännölliselle lausekkeelle F, ensin salakirjoitetaan M standardinmukaisella salausmenetelmällä. Tämän avulla saadaan "välisalateksti" Y. Y:tä käsitellään kokonaislukuina $Z_{|L(F)|}$, joka on joukko kokonaislukuja nollasta kielen kokoon vähennettynä yhdellä. Tämän jälkeen formaatti muunnetaan soveltamalla koodausfunktiota, jonka tuloksena saadaan $Z_{|L(F)|} \rightarrow L(F)$. Jotta salakirjoitus voidaan purkaa, täytyy edellä esitetyn koodauksen olla bijektio. Luonnollisesti dekoodaus tapahtuu: $L(F) \rightarrow Z_{|L(F)|}$. (Dyer et al. 2013.)

Ohjelmistossa on oma tallennuskerros (engl. record layer), jonka tehtävänä on puskuroida, koodata, jäsentää ja dekoodata fteproxyn viestejä. Tallenuskerros tarvitsee toimiaikseen TCP-protokollan, eli se toimii ns. TCP-protokollan päällä. Se myös olettaa, että lähettäjä ja vastaanottaja ovat muodostaneet joukon avaimia, jotka ovat mahdollisesti johdettu yhdestä jaetusta salaisuudesta. Tallennuskerroksella formaatti F koostuu kolmesta muuttujasta: (R, k, m), jossa R tarkoittaa säännöllistä lauseketta. Arvo k on numero (k > 0), joka määrittelee merkkijonon pituuden, jota käytetään kielessä L(R). Kokonaisluku m ($m \geq 0$) hallinnoi muotoilettoman salatekstin bittien määrää. Kyseiset bitit lisätään fte-koodattujen viestien perään. Kun käytetään tiettyä pituutta k, saadaan sopiva tapa renderöidä helposti jäsenneltävää fte-salatekstiä. Arvon "m" avulla muodostettaan edullisesti lisäkapasiteettia tapauksissa, joissa haluttu kieli on mikä tahansa merkkijono, jolla on etuliitteenä L(R). Alla kuvassa 10, on esitetty fte:n tallennuskerroksen toiminnallisuudet. Vasemmalla puolella on lähettävä osapuoli ja oikealla vastaanottaja. (Dyer et al. 2013.)



Kuva 10. *Fte:n tallennuskerros (Dyer et al. 2013)*

Lähetyspäässä salakirjoitettu data syöteteään halkaisijamodulille (engl. split module), jolla on oma sisäinen puskurimuisti. Halkaisijamodulin tehtävänä on muodostaa kaksi merkkijonoa, toinen formatointia eli "unrank-modulia" (C'_1) varten ja toinen lähetetään sellaisenaan eteenpäin (C'_2). Unrank-modulille menevään merkkijonoon lisätään puskurimuistista merkkejä. Unrank-moduuli on komponentti, jonka avulla suoritetaan edellä esitetty salatekstin koodaus tiettyyn formaattiin. Varsinainen fte-salakirjoitus muodostetaan halkaisijamodulilta eteenpäin välitetystä salakirjoitetusta merkkijonosta (C'_2) ja unrank-modulilta tulevan formatoidun merkkijonon (X) yhdistelmästä. Vastaanottajan puolella formaatin dekoodaus salatekstiksi suoritetaan "rank-moduulin avulla".

Lyhyesti yhteenvedettynä voidaan todeta, että fte koodaa datan siten, että se vastaa mielivaltaista säännöllistä lauseketta. Tämä on tehokas tapa DPI-tekniikkaa vastaan, joka perustuu säännöllisiin lausekkeisiin. Fte naamioi liikenteen näyttämään joltain, joka kuuluu DPI:n näkökulmasta sallittuun kategoriaan. Usein sallittuun kategoriaan kuuluu HTTP-liikenne, joka sisältyy fte:n oletuskonfiguraatioon. Oletuksena voidaan liikennettä naamioida myös SSH-liikenteeksi. Alla esimerkki (kuva 11) liikennekaappauksesta, jossa SSH-liikenne on hämäännytetty HTTP-liikenteeksi Fteproxyn avulla. Tässä käytetään ohjelmiston oletus "manual-http-request"- ja "manual-http-response"-formaattia.

```
GET /ELfDpckdMzN1YFShX7OO82dNzo0LiEv2BjXcTxfLmFQmYCBvlN833Cs6L2Y18qdcupFChPyPSGtF60/5r9.WbVz8YK/
3FKDO7TFmTCxhnSKk.VDtK8Dx7PwsyRscmIIUiAo8.u7BU9uxb4ADtJuaOHoeqqdzVPZbKt6m3QoEZMPCaDSYZKBQVRgedkn0uYl3IW8PPZIwIx4tvXR7i5iMpAM5QA/
McTEuPIN6XmLFvKIrda HTTP/1.1

HTTP/1.1 200 OK
Content-Type: HH

D7...0...B.....^..s..3{?N]r..e...w.q?......pS.a9u.5g...F3Rl..........%..5...Aq....$..K......-..x&.}/../..O.G.T....d.z.DKc...Q..\...
2..w.s.eA.{..W%."e[..8`..I5.no.i..V1.y(..XKP..L....k...6.^Y.....88...'....c..m...|E..<y.HTTP/1.1 200 OK
Content-Type: H

OS......|2...%8@d.......)8_.Q..V!..RHP\8..&./.|5..9........   '....j...#a..
y..{...8(.F.d....I.9.6.]P....P.b?*?....&....&jH....BZ.i".\........f...#q....7....J.e.c..Z.}.K....<.C.R.CS...i.4..P...       .....Ql_..
1he...W...w. ..c.U.
......_.e........rvm.t..O3..f.c.......r4.;....
```

Kuva 11. *Fteproxyn avulla HTTP-liikenteeksi naamioitu SSH-liikenne*

Kuvasta 11 havaitaan (punainen teksti), kuinka HTTP-protokollan mukaiset GET ja HTTP/1.1 merkkijonot sisältyvät pyyntöön. Pyydetty url on muodostettu säännöllisen lausekkeen avulla. Sinisellä pohjalla oleva teksti on fteproxy-palvelimen vastaus. Ku-

vassa vastaukset eivät näy täydellisenä tilan säästämiseksi. Kuten edellä on todettu, säännöllisiin lausekkeisiin perustuva DPI etsii hyötykuormasta juuri tiettyjä merkkijonoja kuten "GET" tai "HTTP/1.1 200 OK". Omia formaatteja voidaan lisätä kohtuullisen helposti "fteproxy/defs/päivämäärä.json" -tiedostoon. Liitteessä C (Dyer 2014) on esimerkki Bittorrent-formaatin lisäyksestä.

5.4 Dust - pakettihämäännytysmenetelmä

Dust eroaa muista liikenteen hämäännyttämismenetelmistä, sillä sen avulla liikenne voidaan koodata annetun määritelmän tai parametrien mukaan (Schneier 2013). Ohjelmisto perustuu asiakas-palvelin -malliin (Wiley 2011), kuten muutkin tässä esitettävät hämäännyttämismenetelmät. Parametrit, kuinka liikennettä hämäännytetään, on mahdollista toteuttaa automaattisesti sen mukaan mitä liikennettä estetään verkkosensorin tai liikennettä suodattavan laitteen avulla ja mikä liikenne pääsee siitä läpi. (Schneier 2013.) Ohjelmiston kehittäjä Brandon Wiley on toteuttanut työkaluja estettävän liikenteen havainnointia varten. Dust-protokollan mukana tulee kaksi työkalua, joiden avulla voidaan havaita verkkosensorin läpäisevä liikenne. Wileyn (2014b) kehittämät työkalut ovat Replay ja Shaper. Replay on työkalu, joka etsii estettäviä tavumerkkijonoja. Työkalu on siis tarkoitettu pakettien syvätarkastusta vastaan, joka perustuu säännöllisiin lausekkeisiin. Shaper-työkalun tarkoitus on selvittää, kuinka liikenne tulisi muotoilla, jotta se läpäisee tilastollisen analyysin. (Wiley 2014b.) Periaatteessa näiden työkalujen avulla on mahdollista tehdä koneellisesti päätöksiä siitä, kuinka liikennettä tulisi muuntaa, jotta sitä ei tunnistettaisi. Wiley (2014b) ei kuitenkaan esitä tätä suoraan Dustin spesifikaatiossa. Käytännössä kyseisten työkalujen avulla lähetetään "tietyn näköistä" testiliikennettä, jos liikenne blokataan, liikenneprofiilia vaihdetaan toiseksi. Liikenneprofiilin muodostamiseen hyödynnetään pakettikoon sekä pakettien saapumisen välistä tilastollista jakaumaa. Dust kykenee hyödyntämään näitä työkaluja, joten periaatteessa voidaan sanoa, että ohjelmisto tarjoaa automaattisesti muuttuvan obfuskointikerroksen. Käytettävät liikenneprofiilit tulee olla kuitenkin etukäteen generoitu. Automaattisesti muuttuva obfuskointikerros luonnollisesti vaikeuttaa liikenteen tunnistamista verkkosensorissa. Tätä toiminnallisuutta ei ole luotu vielä toistaiseksi muihin tunnettuihin hämäännyytämismenetelmiin, joten Dust on tämän suhteen edelläkävijä.

Perusperiaatteeltaan Dust ei poikkea muista hämäännyttämismenetelmistä. Sen muodostamat paketit koostuvat salatuista tai satunnaisista kertakäyttöisistä tavuista, jotta niitä olisi mahdoton erottaa toisistaan ja muista satunnaisista paketeista. (Wiley 2011.) Poikkeavuuksia kuitenkin löytyy monista yksityiskohdista, joista esimerkkinä voidaan mainita mm. avaintenvaihto, jossa käytetään siirtolinjan ulkopuolella (engl. out-of-band half-handshake) tapahtuvaa puolikättelyä. Tässä liikennöivien osapuolien on saatava ensiksi siirtolinjan ulkopuolelta kutsu liittyä verkkoon. Tämä kutsu sisältää vastaanottajan IP-osoitteen ja julkisen avaimen. Lähettävä osapuoli voi tämän jälkeen päättää kättelyvaiheen lähettämällä siirtolinjan ylitse yksittäisen "intro-paketin", jonka perään voi-

daan lähettää useita datapaketteja. Datapaketit ovat salattu istuntoavaimella, joka lasketaan kättelyvaiheessa. Lyhyin mahdollinen Dust-keskustelu koostuu tämän vuoksi kahdesta siirtolinjan yli lähetetystä paketista: yksi intro-paketti ja yksi datapaketti. Dust-protokolla sallii näiden pakettien ketjutuksen yhteen, jotta ne mahtuvat yhteen UDP tai TCP pakettiin. (Wiley 2011.) Tämä estää ajastushyökkäykset (Wiley 2011) ja vaikeuttaa luonnollisesti pakettikokoanalyysia tai muuta tilastollista liikenneanalyysia, koska keskustelu voi koostua vain yhdestä paketista ja sen koko voi vaihdella merkittävästi.

Kuten edellä on todettu, sovelluksen toteuttaman yhteyden muodostuskättelyn perusteella voidaan usein tunnistaa liikennöivä sovellus. Tämän vuoksi kättelyyn on oleellista perehtyä tarkemmin. Yksityiskohtaisemmin esitettynä Dust-kättely toimii seuraavasti:

1. Palvelin luo tunnisteen ja jaetun salaisuuden, jonka jälkeen asiakkaalle lähetetään kutsupaketti siirtolinjan ulkopuolelta. Kutsu on suojattu salasanalla ja se sisältää palvelimen IP-osoitteen, julkisen avaimen, tunnisteen ja salaisuuden. Siirtolinjan ulkopuolelta lähetetyn kutsun mukana lähetetään myös salasana, jotta kutsu saadaan avattua. Salasana voidaan lähettää eri kanavaa pitkin kuin itse varsinainen kutsuviesti.
2. Asiakas päättää yhteyden muodostamiseen tarvittavan kättelyn lähettämällä intro-paketin kutsussa olevaan IP-osoitteeseen ja porttinumeroon. Intro-paketin alkuun on lisätty kertakäyttöinen ja satunnainen tunniste kutsupaketista. Paketti on salattu kutsussa olevalla salaisuudella ja se sisältää asiakaan julkisen avaimen.
3. Kun palvelin vastaanottaa paketin tuntemattomalta asiakkaalta, se olettaa sen olevan intro-paketti ja tarkistaa sen alusta tunnistearvon. Tarkistus toteutetaan vertaamalla tunnistetta sitä vastaavaan tallennettuun tunnisteeseen, joka sijaitsee palvelimella. Palvelin purkaa paketin jaetun salaisuuden avulla sekä vastaanottaa asiakkaan julkisen avaimen. Tämän jälkeen palvelin muodostaa jaetun istuntoavaimen. Se lisää istuntoavaimen listalle, jossa ylläpidetään listaa tunnetuista asiakkaista. Tunnettuun asiakkaaseen liitetään myös IP-osoite ja porttinumero josta intro-paketti lähetettiin. Tämä päättää julkisen avaimen vaihdon toisen vaiheen.
4. Kun avaintenvaihto on kokonaisuudessaan päätetty, asiakas ja palvelin voi aloittaa luottamuksellisen kommunikaation vaihtamalla salattuja datapaketteja. Kuten yllä jo esitettiin Dust pystyy yhdistämään intro- ja datapaketit TCP tai UDP pakettien sisällä. (Wiley 2011.)

Dustin arkkitehtuuri koostuu kolmesta eri kerroksesta, jotka ovat sovelluskerros, salauskerros ja muotoilukerros. Muotoilukerros ottaa tulosteen salauskerrokselta, joka siis muistuttaa täysin satunnaista tavuvirtaa ja muovaa sitä vastaamaan tavoitteena olevaa tilastollista mallia. (Wiley 2014a.) Muotoilukerroksen tavoitteena on muuntaa satunnainen tavuvirta muistuttamaan jotain tiettyä liikennetyyppiä, kuten esimerkiksi HTTP-liikennettä tai HTTPS-liikennettä. Tämä johtuu siitä, että täysin satunnaiselta näyttävä tavuvirta voi herättää liikennettä tarkkailevissa toimijoissa huomiota. Esimerkiksi kaikki salattu liikenne voidaan lähtökohtaisesti estää. (Wiley 2013.)

Eri liikennetyypeillä tai protokollilla on erilaisia ominaisuuksia, joiden perusteella voidaan rakentaa tilastollisia malleja. Näiden tilastollisten ominaisuuksien avulla voidaan tunnistaa sovelluskerroksen protokolla ja liikennöivä sovellus. Muotoilukerroksella on valmiina edellä mainittuja malleja. Jokainen liikenteen tilastollinen ominaisuus on esitetty tilastollisena todennäköisyysjakaumana. Tällaisia ominaisuuksia ovat esimerkiksi sisällön merkkien jakauma tai paketin koko. (Wiley 2014a.) Esimerkiksi HTTP-liikenteellä on tietty todennäköisyysjakauma pakettikoon suhteen (Wiley 2013). Muotoilukerros muuntaa liikenteen esittämään tiettyä kohdejakaumaa (Wiley 2014a).

Kuten luvussa 4.2 todettiin, käänteistä Huffman-koodausta voidaan hyödyntää liikenteen imitoinnissa muuntamalla satunnainen bittijono tietyn merkkijakauman mukaiseksi. Tämän avulla hyötykuorman sisältö imitoi jonkin toisen sovelluksen muodostamaa liikennettä. Tämän avulla ei voida kuitenkaan muodostaa haluttua pakettikokojakaumaa tai pakettien välisten saapumisaikojen jakaumaa. Esimerkiksi TCP-protokollaa hyödyntävien sovellusten tapauksessa, sovellus syöttää TCP-ohjelmalle dataa, mikä vastaa vuon ohjauksesta ja ruuhkanhallinnan toimenpiteistä. Pakettikoko ja pakettien välinen saapumisaika määräytyvät näin ollen TCP-ohjelman toimenpiteiden perusteella. Dustissa nämä jakaumat muodostaa muotoilukerros, joka tarvitsee toimiakseen oikeaa malliliikennettä tilastollisten piirteiden imitointia varten. Muotoilukerroksen liikenteen koodaus toimii periaatteeltaan seuraavasti:

- poimi imitoitavan liikennemallin yhteyden kesto → funktio: "duration()"
- ajetaan seuraavat toiminnot While-silmukassa (kulunut aika < liikennemallin yhteyden kesto)
    - poimi imitoitavasta liikennemallista pakettimäärä, joka tulee lähettää kuluneen ajan puitteissa → funktio: "packet_count(elapsed)"
    - tee seuraavaa jokaiselle lähetettävälle paketille
        - poimi liikennemallin paketin pituus → funktio: "packet_length()"
        - koodaa data lähetettäväksi, jotta saadaan tavumäärä → funktio: "encode(bytes)"
        - jos koodatun datan tavumäärä on pienempi, kuin liikennemallista saadun paketin pituus
            - täytä paketti satunnaisilla tavuilla → funktio: "add_padding(bytes, length)"
- sulje yhteys
- toista edellinen avaamalla uusi yhteys. (Wiley 2014a.)

6. HAITALLINEN LIIKENNE JA SEN HAVAINNOINTI

Haitallinen liikenne on varsin kattava käsite, joka tarkoittaa haitallisen koodin lähettämistä tietoliikenneyhteyden ylitse kohteeseen eri keinoin. Lähettäminen ei välttämättä tarkoita sitä, että hyökkääjä aktiivisesti lähettäisi haitallista koodia, vaan kohde voidaan eri keinoin ohjata lataamaan laitteelleen itse haitallinen koodi. Kohde voi olla tarkkaan valikoitu, tai kohteella ei ole mitään merkitystä. Tarkoituksena voi olla esimerkiksi haitallisen koodin levittäminen tietoverkoissa mahdollisimman moneen laitteeseen ja laitetyyppiin. "Hyökkääjät" voivat vaihdella valtiollisista toimijoista monikansallisiin rikollisryhmiin. Haitallisen liikenteen generoinnin tavoitteet voivat myös vaihdella merkittävästi. Tavoitteena voi olla esimerkiksi taloudellisen edun saaminen tietomurron tai palvelunestohyökkäyksen avulla. Toisessa äärilaidassa on taas esimerkiksi haktivismi, jonka tavoitteet voivat perustua jonkin yhteiskunnallisen muutoksen aikaansaamiseksi.

Yksi pakettien syvätarkastuksen tavoitteista on havaita haitallista liikennettä. Esimerkisi bot-verkkojen ja erityisesti niiden komentokanavien havaitseminen on pakettien syvätarkastuksen avulla mahdollista. Kuten edellä todettiin, on haitallinen liikenne erittäin laaja käsite. Siksi aihetta tullaan tässä rajaamaan koskemaan erityisesti kohdistettuja haittaohjelmahyökkäyksiä, joissa kohde on tarkkaan valittu. Painotus on valittu näiden hyökkäysten yleistymisen johdosta viime vuosina, mikä näkyy muuan muassa tietoturvayhtiöiden lisääntyneinä haittaohjelmaraportteina (Kyberturvallisuuskeskus 2014). Näiden hyökkäysten suorittajat ovat usein valtiollisia toimijoita, joilla on resurssit toteuttaa teknisesti edistyksellisiä hyökkäyksiä. Tässä luvussa keskitytään erityisesti kohdistetuissa hyökkäyksissä käytettyihin RAT-ohjelmistoihin, niiden tuottamaan komentoliikenteeseen ja sen havaitsemiseen verkkoliikenteestä. Tämän kaltainen liikenne on usein hämäännytetty.

6.1 Kohdistetut haittaohjelmahyökkäykset

Kohdistetutuissa haittaohjelmahyökkäyksissä kohde on tarkkaan valittu hyökkääjän motiivin perusteella. VAHTI 6/2009 määrittelee kohdistetun hyökkäyksen seuraavasti: *"Kohdistetun hyökkäyksen tavoitteena on jonkin tietyn tiedon kaappaaminen tai tietyn kohteen toiminnan haittaaminen"*. Näistä hyökkäyksistä käytetään myös nimitystä "edistynyt pysyvä uhka" (engl. advanced persistent threat, APT). Termi "edistynyt pysyvä uhka" ei ole vakiintunut suomenkielessä. Se kuitenkin kiteyttää hyvin nämä hyökkäykset. Kohdistetut hyökkäykset ovat usein teknisesti edistyneitä ja niiden tarkoituksе-

na on saavuttaa pitkäaikainen pysyvyys kohde järjestelmässä. Pysyvyyden avulla pyritään varmistamaan tavoitteiden saavuttaminen.

Kohdistettujen hyökkäyksien tavoitteet kohdistuvat usein sotilaalliseen toimintaan, politiikkaan tai taloudellisen edun saavuttamiseen. Hyökkäyksissä käytetyt haittaohjelmakomponentit eivät kuitenkaan välttämättä ole edistyneitä. Ne saattavat olla jo tunnettuja tekniikoita, mikä helpottaa niiden toteutusta. Termi edistynyt saa enemmänkin merkityksensä hyökkäyksen valmisteluista. Hyökkäyksen valmistelu sisältää mahdollisimman tarkan tiedon keräämisen kohteesta. Näissä hyökkääjällä saattaa olla käytössä eri tiedustelumenetelmiä tiedon keräämistä varten. Näitä ovat esimerkiksi signaalitiedustelumenetelmät, avoimien lähteiden tiedustelumenetelmät ja henkilötiedustelu. Tiedustelutiedon avulla pyritään vaikuttamaan mahdollisimman monen reitin kautta kohteeseen, jotta ensimmäinen "jalansija" saadaan muodostettua. Toisin sanoen hyökkääjä voi yhdistellä useita maalittamismenetelmiä saavuttaakseen ensimmäisen tavoitteensa, eli sisäänpääsyn kohdejärjestelmään. Haittaohjelmien osalta hyökkääjällä voi olla olemassa tuotantolinja, joten olemassa olevia ohjelmistoja voidaan tarvittaessa muuttaa tai vaihtaa niitä eri versioihin. (David 2014.)

Pitkäkestoisuus on ominaista kohdistetuille hyökkäyksille. Saavutettuaan pääsyn kohdejärjestelmään, hyökkääjä pyrkii säilyttämään reitin sen sisälle mahdollisimman pitkään. Informaation kerääminen tai toiminto, jota toteutetaan on maalitettu jo etukäteen edellä esitetyssä tiedusteluvaiheessa. Pysyvyyden saavuttamista varten hyökkääjä on laatinut etenemissuunnitelman, joka sisältää varmuuskopiointisuunnitelman ja suunnitelman mahdollisesta vetäytymisestä tietyn hyökkäysmenetelmän paljastuessa. Usein kohdistetuissa hyökkäyksissä käytetään hyväksi useita järjestelmähaavoittuvuuksia, jotta yhden paljastuessa pysyvyys kohdejärjestelmässä kyetään säilyttämään. Säilyvyys voidaan taata myös matalalla ja hitaalla lähestymisellä. (David 2014.) Tiedusteluvaihe on tässä erityisen merkittävä, koska se voi kestää riippuen kohteesta useita vuosia. Hitaalla ja matalalla lähestymisellä tarkoitetaan valmisteluja, joilla esimerkiksi kohdejärjestelmän käyttäjä pyritään "ohjaamaan" suorittamaan haitallista koodia, siten, että käyttäjä ja järjestelmä eivät havaitse tekevänsä mitään "väärää". Toisin sanoen aktiivinen vaikuttaminen suoraan käyttäjään ja kohdejärjestelmään pyritään minimoimaan.

6.1.1 Hyökkäysten kehittyminen ja APT-kampanjat

Kohdistettujen hyökkäysten kehityskaari on suuntautunut 1990-luvun verkkoperusteisista hyökkäyksistä asiakasperusteisiin hyökkäyksiin (David 2014). Tämä tarkoittaa sitä, että nykyään tunkeutuminen pyritään saavuttamaan verkossa toimivan asiakaslaitteen avulla, oli tämä sitten työasema tai älypuhelin. Kuten edellä on todettu hyökkäystä varten laaditaan etenemissuunnitelma siten, että hyödynnetään useita haavoittuvuuksia. Hyökkäyksen paljastuessa hyväksikäytetään toista haavoittuvuutta tai takaporttia, joka on jo ehditty rakentamaan heti järjestelmään päästyä. Usein alustavilla raporteilla paljastuneista kohdistetuista hyökkäyksistä on taipumus esittää, että kyseessä on yksi

hyökkäys tai yksittäinen operaatio, joka sisältää useita hyökkäystapahtumia. Kasvavassa määrin nykyään ollaan kuitenkin havaittu, että useimmat kohdistetut hyökkäykset ovat osa isompaa kampanjaa (Villeneuve & Bennett 2012). Näistä käytetään termiä APT-kampanja.

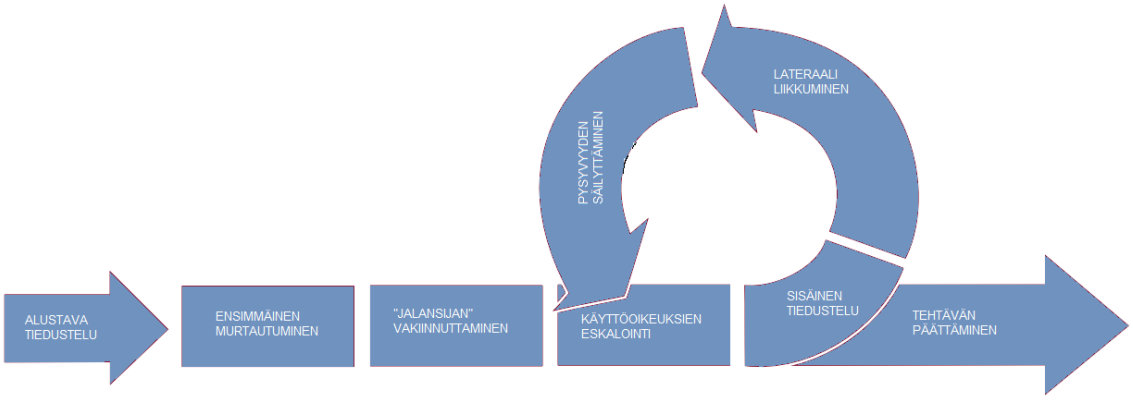
Yksi tunnetuimmista APT-kampanjoista on APT1. Itse asiassa APT1 on kiinalaisen ryhmän nimi, joka on toteuttanut kyberoperaatioita laajalle kohderyhmälle ainakin vuodesta 2006 alkaen. Se on toteuttanut vuosien 2006-2013 aikana kohdistettuja hyökkäyksiä vähintään 150:een eri kohteeseen. Kohteina on ollut eri teollisuuden organisaatioita. Määrällisesti hyökkäykset ovat kohdistuneet pääosin mm. ilmakehä- ja avaruusteollisuuteen, tietoliikenne- ja satelliittiteollisuuteen, kuljetusteollisuuteen sekä julkiseen hallintoon. Hyökkäykset ovat enimmäkseen suuntautuneet Yhdysvaltoihin ja niiden tavoitteena on ollut informaation varastaminen. Ryhmällä on ollut todennäköisesti Kiinan valtion tuki tai ainakin valtio on ollut tietoinen ryhmän toiminnasta. Yhdysvaltalaisen Mandiant kyberturvallisuusyrityksen analyysin mukaan APT1 on Kiinan kansan vapautusarmeijan osasto 61398, joka toimii signaalitiedustelujaoston alaisuudessa. APT1 on yksi noin kahdestakymmenestä Kiinassa toimivista ryhmistä, jotka toteuttavat APT-kampanjoita (Mandiant 2013).

APT1-ryhmä säilytti keskimäärin 356 vuorokauden ajan pääsyn kohdeverkkoon ja järjestelmiin. Pisin kohdistetuista hyökkäyksistä kesti lähes viisi vuotta. Varastettu informaatio on ollut varsin laaja-alaista ja se on sisältänyt mm. tuotekehitysdokumentteja, testaustuloksia, dokumentteja tuotantoprosesseista, liiketoimintasuunnitelmia ja korkea arvoisten työntekijöiden sähköposteja. Parhaimmillaan yhdestä kohteesta on varastettu informatiota 6,5 teratavua kymmenen kuukauden aikana. Luonnollisesti varastamisen kohteena on ollut myös kohdeverkon käyttäjätunnukset ja verkkotopologia. Näiden varastaminen on yksi elinehto sisäiselle tiedustelulle ja lateraaliselle liikehdinnälle kohdeverkkossa. Lateraalisen liikkeen avulla luodaan uusia takaportteja uusiin asikaslaitteisiin. Tämän avulla pyritään säilyttämään pysyvyys kohdeverkossa ja liikkumaan kohti todellista kohdetta sekä tavoitetta. (Mandiant 2013.)

6.1.2 Kohdistetun hyökkäyksen elinkaari

Kohdistetulla hyökkäyksellä on selkeä elinkaari, jonka esimerkiksi Mandiant on omassa APT1 analyysissään kuvannut. Hyökkäyksen elinkaareen sisältyviä vaiheita on osittain sivuttu jo edellä olevissa luvuissa. Elinkaari voidaan jakaa kahdeksaan eri vaiheeseen, jotka ovat: alustava tiedustelu, ensimmäinen murtautuminen, jalansijan vakiinnuttaminen, käyttöoikeuksien eskalointi, sisäinen tiedustelu, lateraali liikkuminen, pysyvyyden säilyttäminen ja tehtävän päättäminen. Kuvassa 12 on esitetty edellä mainitut vaiheet ja malli, kuinka osa vaiheista toteuttaa sykliä. Eri vaiheet on esitelty tarkemmin seuraavissa kappaleissa.

Kuten edellä on jo esitetty, kohdistettu hyökkäys vaatii esitöikseen tarkan tiedustelun. Sen elinkaari alkaa alustavasta tiedustelusta, jossa voidaan käyttää useita tiedustelumenetelmiä. Tämän vuoksi termiä "edistynyt" käytetään englanninkielisessä APT-termissä. APT1-ryhmän toteuttamissa hyökkäyksissä ensimmäinen murtautuminen kohdejärjestelmään on tapahtunut useasti kohdistetun kalasteluhyökkäyksen avulla (engl. spear phishing attack). Kohteena olevan organisaation henkilölle on lähetetty haitallista koodia sisältävä sähköposti. Henkilö on valittu alustavan tiedustelun tuloksena. Sähköposti on lähetetty kohteena olevan henkilön läheisen työtoverin tai esimiehen sähköpostiosoitteesta. Sähköposti on sisältänyt joko liitetiedoston, jonka suorittamalla haitallinen koodi asentuu suorittajan työasemalle tai linkin, jossa kohdekäyttäjä ohjataan lataamaan haitallinen koodi omalle työasemalleen. (Mandiant 2013.)



Kuva 12. *Kohdistetun hyökkäyksen elinkaari (Mandiant 2013)*

"Jalansija" vakiinnutetaan takaporttiohjelmiston avulla, joka asentuu haitallisen koodin suorittamisen johdosta. Takaporttiohjelmisto ottaa yhteyttä hyökkääjän komentopalvelimeen, josta käytetään myös nimitystä C2-palvelin. Tämä tulee englanninkielisestä termistä “command and control”. (Mandiant 2013.) Organisaation verkon reunalla oleva palomuuri voidaan ohittaa, koska hyökkäyksen kohde muodostaa komentoyhteyden. Usein ulospäin lähtevää liikennettä ei rajoiteta. Takaporttiohjelmisto voidaan myös injektoida kohdetyöasemassa esimerkiksi oletusselainprosessiin, jolloin mahdollinen asikaspalomuuri ei estä liikennettä. Komentoliikenne voidaan pakata tai salata sekä naamioida näyttämään se esimerkiksi HTTP-liikenteeltä, jolloin IDPS-järjestelmien on vaikea havaita sitä. Toisin sanoen komentoliikenne hämäännytetään.

APT1 on hyvä esimerkki siitä, kuinka aika ajoin hyökkäyksissä on käytetty tunnettuja ja julkisesti saatavilla olevia takaporttiohjelmistoja kuten Poison Ivy ja Gh0st RAT (Mandiant 2013). Näistä ohjelmistoista käytetään myös nimitystä "Remote Access Trojan" tai "Remote Access Tool" (RAT). Kuten todettu jo aikaisemmin, tekniikka ei ole aina edistynyttä. Toisaalta suurin osa APT1-ryhmän käyttämistä takaporttiohjelmistoista on ollut heidän itse tuottamia (Mandiant 2013).

Käyttöoikeuksien eskalointi sisältää uusien käyttäjätunnuksien ja salasanojen hakemisen kohdejärjestelmästä. Näiden avulla hyökkääjällä on mahdollisuus saada pääsy verkon eri resursseihin. Usein käytössä on julkisesti saatavilla olevia työkaluja, joilla voidaan ladata salasanojen tiivistesummia kohdejärjestelmästä. Tälläisia työkaluja ovat muun muassa "fgdump", "gsecdump", "lslsass", "mimikatz" ja "pass-the-hash toolkit". (Mandiant 2013.)

Sisäisessä tiedusteluvaiheessa hyökkääjä kerää informaatiota kohdejärjestelmästä. Murtautumisen kohteena olevasta asiakaslaitteesta kerätään muun muassa IP-konfiguraatio, käynnistetyt palvelut, käynnissä olevat prosessit, käyttäjätunnukset, käyttäjätunnukset, joilla on järjestelmävalvojan käyttöoikeudet, ja jaetut verkkoresurssit. Yhtenä tarkeänä tiedustelukohteena on eri palveluita tuottavat palvelimet, jotka pyritään myös listaamaan murtautumisen kohteena olevasta laitteesta. Tietojen avulla pyritään kartoittamaan verkkoympäristö mahdollisimman tarkasti, jotta hyökkäyksen tavoite saavutetaan. Sisäisen tiedustelutiedon keräämiseen käytetään usein käyttöjärjestelmän sisäänrakennettuja komentoja. Komentojen automatisointia varten voidaan laatia skriptejä. (Mandiant 2013.)

Todellisten käyttäjätunnusten ja salasanojen keräämisen jälkeen, hyökkääjän on mahdollista suorittaa sivuittaista eli lateraalia liikkumista kohdeverkossa herättämättä huomiota. Yksinkertaisuudessaan tämä tarkoittaa uusille laitteille murtautumista olemassa olevien tunnusten avulla. Tunnusten avulla voidaan myös hakea jaettuja verkkoresursseja, joihin ko. tunnuksilla on pääsyoikeudet. Komentoja voidaan suorittaa muissa järjestelmissä olemassa olevien järjestelmänvalvojan työkalujen avulla. Tälläisia työkaluja ovat esimerkiksi psexec tai Windowsin sisäänrakennettu ajastusohjelma "at.exe". Toimintaa on vaikea havaita, koska järjestelmänvalvojat käyttävät usein samoja työkaluja omassa työssään. (Mandiant 2013.)

Läsnäolon- tai pysyvyyden säilyttämisvaiheessa tunkeutuja suorittaa toimenpiteitä, joilla varmistetaan pitkäaikainen pääsy avainjärjestelmiin ulkoakäsin. APT1 on käyttänyt omissa hyökkäyksissään kolmea eri menetelmää:

- uusien takaporttiohjelmistojen asennus useisiin järjestelmiin
- kohteena olevan organisaation VPN-tunneleiden hyväksikäyttö varastetuilla tunnuksilla
- kohdeverkon tarjoamien web-portaalien hyväksikäyttö. Esimerkkinä voidaan mainita Microsoft Outlook Web Access. (Mandiant 2013.)

Saavuttaessaan tavoitteensa tai löytäessään mielenkiintoisia tiedostoja, hyökkääjä usein pakkaa tiedostot ennen niiden siirtämistä ulos kohdeverkosta. Yleinen pakkausmenetelmä on RAR. Pakatut tiedostot suojataan salasanalla. Monesti pakattujen tiedostojen koko on silti niin suuri, että ne joudutaa pilkkomaan. Esimerkiksi APT1 pilkkoi omissa hyökkäyksissään datan 200Mt:n paloihin. Pilkottujen tiedostojen lähettämiseen ulos

kohdeverkosta voidaan käyttää takaporttiohjelmistoissa olevaa FTP-asiakasohjelmaa. (Mandiant 2013.)

6.2 Komentokanavat takaporttiohjelmistoissa

Mandiant (2013) esittää APT1-raportissaan takaporttiohjelmistojen jakautuvan kahteen eri kategoriaan. Nämä ovat "sillanpää"-takaporttiohjelmistot (engl. beached backdoor) ja standardit takaporttiohjelmistot (engl. standard backdoors). Ensiksi mainitut ovat tyypillisesti toteutettu vähäisin ominaisuuksin. Niiden tarkoituksena on tarjota hyökkäjälle jalansija ja mahdollistaa joidenkin yksinkertaisten toimintojen toteuttaminen, kuten tiedostojen siirtäminen ja perusinformaation kerääminen kohdejärjestelmästä. Oleellisin toiminto näille on kuitenkin laukaista merkittävämpää kykyä, kuten standardin takaporttiohjelmiston asentaminen kohteeseen. (Mandiant 2013.) Merkittävä kyky voidaan saavuttaa esimerkiksi RAT-ohjelmistoilla, joilla on varsin kattavat ominaisuudet kohdejärjestelmän täydelliseen hallintaan.

Sillanpää-takaporttiohjelmistoista hyvä esimerkki on WEBC2-takaportti. Näissä uhrin järjestelmä suorittaa HTTP-pyyntöjä www-palvelimelle, joka toimii komentopalvelimena. Uhrin ja C2-palvelimen välille muodostuu näin ollen komentokanava. HTTP-pyyntöihin palautetut vastaukset eli web-sivut sisältävät komentoja, joita uhrin järjestelmä suorittaa. Komennot on upotettu tiettyihin HTML-tageihin, jotka uhrin tietokoneella oleva palvelinohjelmito tulkkaa komennoiksi. Ensimmäiset versiot havaituista WEBC2-takaporteista käyttivät HTML-sivun kommentteja komentoina. Eri varianttien kehittyessä komennot siirtyivät muiden HTML-tagien sisälle. (Mandiant 2013.)

Käyttämällä hyväksi aitoja HTTP-pyyntöjä ja HTML-sivuja, WEBC2-takaporttien muodostamien komentokanavien viestienvaihto on pyritty hämäännyttämään eli obfuskoimaan. HTTP:n ohella myös muita naamiointimenetelmiä on hyödynnetty. Esimerkiksi APT1 hyödynsi naamioimalla alla taulukossa 2 esitettyjen takaporttiohjelmistojen komentokanavia alla esitettyjen sovellusten liikenteeksi.

Taulukko 2. *Esimerkki APT1-kampanjan takaporttiohjelmien liikenteen naamiointikohteista (Mandiant 2013)*

TAKAPORTTIOHJELMISTO	LIIKENNE NAAMIOITU
MaCroMaIL	MSN Messenger
GLooxMaIL	Jabber/XMPP
CaLenDar	Gmail Calendar

Naamioinnin avulla pyritään hämäämään verkon puolustusta ja siihen tarkoitettuja automatisoituja työkaluja, kuten esimerkiksi IDPS-järjestelmiä. Komentokanavien liikenteen salakirjoittaminen tai pakkaaminen on myös eräs keino hämäännyttää liikennettä tutkivia laitteita. Asiakkaan ja palvelimen välisen liikenteen salakirjoittaminen voidaan toteuttaa monella tapaa, mutta tunnettuja salausmenetelmiä ovat ainakin SSL ja RC4.

APT1-kampanjassa käytettiin juurikin SSL-menetelmää ja luvussa 6.3.2 esiteltävä DarkComet-ohjelmisto käyttää RC4-algoritmia, joka on vuonsalausmenetelmä. SSL-menetelmä voidaan tunnistaa reaaliaikaisena, jos hyökkäyksessä käytetyt SSL-varmenteet ovat tiedossa (Mandiant 2013). Varmenne näkyy SSL-liikenteen kättelyssä selväkielisenä.

6.3 RAT-ohjelmistot

Edellä todettiin, että RAT-ohjelmistoja voidaan kutsua myös takaporttiohjelmistoiksi. Takaporttiohjelma mielletään usein ohjelmistoksi, mikä asentuessaan kohdejärjestelmään avaa kätketyn takaportin. Sen avulla säilytetään pääsy kohdejärjestelmään ja takaportin kautta voidaan antaa kohdejärjestelmälle etäkomentoja. Takaporttiohjelmisto voi myös johtaa terminä harhaan, sillä RAT-ohjelmistot sisältävät usein laajan valikoiman etähallintatyökaluja. Voidaan oikeastaan todeta, että pääsääntöisesti RAT-ohjelmistoilla kyetään toteuttamaan kaikki edellä esitetyt kohdistetun hyökkäyksen vaiheet. Kyseisille ohjelmistoille tyypillisiä ominaisuuksia ovat esimerkiksi:

- Graafinen käyttöliittymä ja helppokäyttöisyys.
- Valmiiksi sisäänrakennetut haavoittuvuuksien hyödyntämismenetelmät, joilla saavutetaan järjestelmänvalvojan oikeudet kohdejärjestelmässä.
- Yksinkertaisesti toteutettuja ja "kevyitä", mutta erittäin tehokkaita. Esimerkiksi Poison Ivy -palvelimen koko on vain 6 kilotavua.
- Työkalut sisäisen tiedustelun suorittamiseen sekä sivuttaisen liikkumisen suorittamiseen kohdeverkossa.
- Työkalut tiedon varastamiseen sekä monia muita työkaluja. (David 2014.)

RAT-ohjelmistoista tekee mielenkiintoisen myös sen, että niillä on tietty maine noviisien, eli skriptareiden (engl. script kiddie) "ohjelmistoleluina". Tämä johtunee siitä, että kyseiset työkalut ovat helppokäyttöisiä ja niiden käyttö ei edellytä laajaa teknistä osaamista. (David 2014.) Kuitenkin näitä on käytetty monissa monimutkaisissa kyberoperaatioissa, joista esimerkkeinä viime vuosilta:

- RSA tietomurto. Hyökkäyksessa saatiin SecurID-token-avaimia. Järjestelmää hyödynnetään muun muassa etäkäyttäjien tunnistamiseen. Varastettujen avaimien avulla hyökkääjien tavoitteena oli päästä käsiksi USA:n puolustusteollisuudelle järjestelmiä toimittavaan Lockheed Martin -yhtiön tietoihin. (Lehto 2011.)
- Nitro-hyökkäys, jossa kohteena oli kemian ja puolustusteollisuuden lisäksi muun muassa ihmisoikeusjärjestöjä ja autoteollisuus. Hyökkäys kohdistui ainakin 48 kemian teollisuuden yritykseen. Näiden joukossa oli 29 yritystä, jotka tutkivat ja kehittävät kemiallisia yhdisteitä ja kehittävät materiaaleja sotilasajoneuvoihin. (Digitoday 2011.)

- Molerats-kampanja, jossa hyökättiin Israelin, Palestiinan, Yhdysvaltojen ja Iso-Britannian valtion eri kohteisiin (Villeneuve, Haq & Moran 2013).
- APT1 – edellä esitetty kiinalainen APT-kampanja, jonka havaittiin olevan aktiivinen vielä vuonna 2014 (David 2014).

Edellä esitetyn perusteella on selvää, että järjestelmän sisällä havaittu RAT-ohjelmisto voi ilmaista kohdistetun hyökkäyksen tai APT-kampanjan kohteeksi joutumista. Tätä tukee myös se, että ohjelmisto on interaktiivinen, eli se vaatii toimiakseen oikean operaattorin tai toimijan. Tämä on tyypillistä myös APT-kampanjoille, sillä tehtävien suorittaminen vaatii aina ihmisen toimintaa. Esimerkiksi on osoittettu, että APT1-kampanjassa takaporttiohjelmistojen käyttäjinä on ollut useita eri henkilöitä. Henkilöt ovat saaneet selvästi käskyjä ylemmältä taholta, mitä toimenpiteitä heidän on täytynyt suorittaa ohjelmistoilla. (Mandiant 2013.) Käyttämällä yleisesti saatavilla olevia RAT-ohjelmistoja APT-kampanjoissa voidaan hämäännyttää forensiikkatutkimusta. Yleisen esiintyvyyden vuoksi ohjelmia on vaikea kytkeä tietyn tyyppisiin laajempiin APT-kampanjoihin tai tiettyihin hyökkääjiin.

RAT-ohjelmistot sisältävät usein välityspalvelimen. Välityspalvelimen avulla kohdejärjestelmän liikennettä voidaan lokittaa, jolloin kohdejärjestelmän liikennöinin lokitiedot on yksi mahdollinen varastettava informaatio. Tämän avulla voidaan suorittaa esimerkiksi sisäistä tiedustelua. Lokituksen lisäksi välityspalvelin mahdollistaa sen, että mahdollinen varastettava informaatio voidaan lähettää kohdejärjestelmästä haluttuun TCP-porttiin. (David 2014.) Jos lähetettävä liikenne pakatataan ja tämän lisäksi vielä salataan, niin on tätä äärimmäisen vaikea havaita. Periaatteessa välityspalvelimen avulla on siis mahdollista hämäännyttää liikennettä. Useimmissa RAT-ohjelmistoissa niiden komentoliikenne on kuitenkin jo valmiiksi hämäännytetty. Hämäännyttämismenetelmänä käytetään tiedonsiirron satunnaistamista ja sen tiettyyn porttiin lähettämistä. Naamiointi ei ole yleisesti käytössä. Poikkeuksena on XtremeRAT, jonka palvelin asentuessaan suorittaa HTTP GET -pyynnön asiakasohjelmistolle.

On selvää, että hyökkääjä haluaa kätkeä RAT-ohjelmistojen toiminnan kohdeverkossa, jotta kohdistetuille hyökkäyksille ominainen, mahdollisimman pitkäaikainen pysyvyys säilytetään. Luonnollisesti mahdollista paljastumista pyritään välttämään. Näiden syiden vuoksi RAT-ohjelmistojen liikenteen hämäännyttämiselle on selkeä motiivi. Seuraavissa alaluvuissa esitellään kaksi yleistä RAT-ohjelmistoa ja analysoidaan niiden tuottamaa verkkoliikennettä.

6.3.1 XtremeRAT

XtremeRAT on Poison Ivy -ohjelmiston kanssa yleisesti tunnettu RAT-ohjelmisto. Se on julkisesti saatavilla ja sitä on käytetty useissa APT-kampanjoissa. Esimerkkinä voidaan mainita edellä esitetty Molerats-kampanja. XtremeRAT-ohjelmistoa käytettiin myös kohdistetuissa hyökkäyksissä vuoden 2014 alussa Israelin puolustusministeriön

alaiseen laitokseen (engl. Civil Administration of Judea and Samaria). Kyseisissä hyökkäyksissä käytettiin kohdistettua kalastelua eli "spear phishing" –sähköposteja, jotka sisältivät haitallisen liitteen. Liitteen hyötykuormana oli XtremeRAT-palvelin.Viestien lähettäjäksi naamioitiin Israelin keskustiedustelupalvelu (Shin-Bet = Shabak), joka toimii Israelin armeijan alaisuudessa. Lähettäjän sähköpostiosoitteeksi oli valittu shabakreport@gmail.com. Liitteinä oli muun muassa viesti, joka koski entistä pääministeriä Ariel Sharonia. Osassa viesteistä liitteenä oli julkisesti saatavilla oleva keskustiedustelupalvelun raportti vuosikymmenen terroristihyökkäyksistä. (Raff 2014.)

Erityisesti Lähi-idästä olevien hyökkääjien on havaittu käyttävän XtremeRAT-ohjelmistoa. Näihin lukeutuu mukaan esimerkiksi Syyrian elektroninen armeija (SEA), jonka on väitetty olevan vastuussa useista edistyneistä hyökkäyksistä. Kohteina on ollut yhdysvaltalaisia mediaorganisaatiota, joista esimerkkinä New York Times. (Graham 2014.)

XtremeRAT on hyvä esimerkki siitä, kuinka voidaan hämäännyttää forensiikkatutkimusta käyttämällä yleisesti saatavilla olevaa RAT-ohjelmistoa. FireEye julkaisi www-sivuillaan vuoden 2014 alussa artikkelin, kuinka suurin osa XtremeRAT-ohjelman toiminnasta liittyi perinteiseen kyberrikollisuuteen. Artikkelin mukaan sitä on käytetty hyväksi roskapostin lähettämiseen ja tämän avulla on muun muassa ladattu kohdekoneisiin perinteinen "pankkitroijalainen", kuten Zeus. (Villeneuve & Bennett 2014.) Kuten todettu jo aikaisemmin, yleisesti saatavilla oleva "työkalu" saattaa tarjota lisätason turvallisuutta edistyneille hyökkääjille, koska tämä mahdollistaa heidän sekoittumisen muihin hyökkääjiin.

XtremeRAT sisältää asiakas- ja palvelinohjelmat. Asiakasohjelma toimii hyökkääjän koneella ja se sisältää graafisen käyttöliittymän. Käyttöliittymän avulla luodaan palvelin, joka toimitetaan asennusta varten hyökkäyksen kohteeseen, eli uhrille. Osassa tällä ohjelmalla toteutetuissa hyökkäyksissä on hyödynnetty nollapäivän haavoittuvuuksia, jonka hyötykuormana palvelinohjelmisto asennetaan. Nitro-kampanjassa hyödynnettiin Javassa olevaa nollapäivän haavoittuvuutta (David 2014). Tiettyä haavoittuvuutta ei kuitenkaan tarvitse välttämättä hyödyntää, jos uhri saadaan houkuteltua avaamaan suoritetteva tiedosto. Tässä voidaan hyväksi käyttää edellä esitettyä kohdistettua kalastelu-hyökkäystä. RAT-ohjelmiston toiminta perustuu käänteiseen yhteyden muodostamiseen. Hyökkääjän kone kuuntelee tiettyä TCP-porttia, johon uhrin kone ilmoittautuu, kun palvelinohjelma käynnistyy. Tämän jälkeen hyökkääjä voi suorittaa RAT-ohjelmiston tukemia komentoja uhrin koneella.

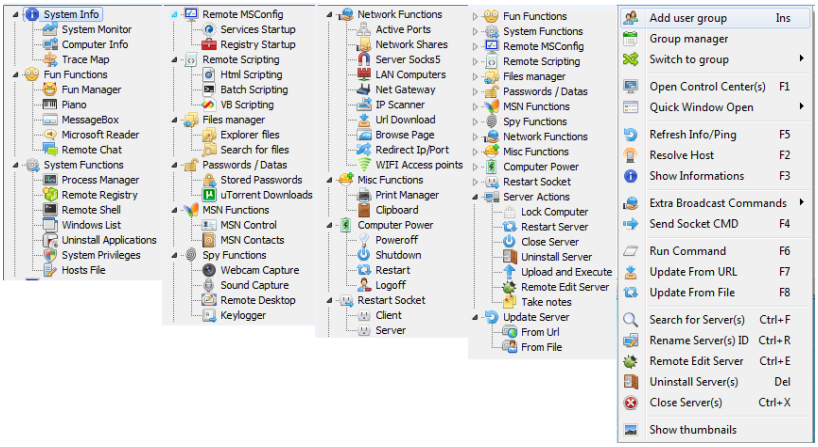
6.3.2 DarkComet

DarkComet RAT -ohjelmisto ei poikkea edellä esitetyistä RAT-ohjelmistoista. Sitä on hyödynnetty useissa APT-kampanjoissa, joista tunnetuin on Syyrian hallituksen kyberoperaatio protestoijia vastaan vuonna 2012. Hallitus hyödynsi ohjelmistoa "kapinaa" ja

kansannousua vastaan lähettämällä kohdistettuja sähköpostiviestejä sisältäen DarkComet-ohjelmiston. Ohjelmistoa levitettiin myös Facebookin avulla. Se lähetettiin pikaviestintätoiminnallisuuden avulla naamioituna turvallisuuspäivitykseksi yhdessä Facebook-ikonin kanssa. Hyödyntämällä muun muassa ohjelmiston tarjoamaa näppäilyn tallennusta, hallitus sai protestoijien Skype-tilejä haltuun. Tilien avulla se pystyi soluttautumaan kapinoitsijoiden verkostoon. DarkComet -ohjelmiston avulla Syyrian hallitus pystyi vakoilemaan useita kansannousun kannattajia ja teki myös monia pidätyksiä vakoilun perusteella. (Santinumi 2012.)

DarkComet ei juurikaan poikkea muista RAT-ohjelmistoista, vaan samoin kuin edellä esitetty XtremeRAT, se sisältää asiakas- ja palvelin-ohjelmistot. Hyökkääjä operoi asiakasohjelmiston kautta, jonka avulla konfiguroidaan palvelin. Palvelin voidaan toimittaa uhrin koneelle eri keinoin. Asentumisen jälkeen palvelin ilmoittautuu "soittamalla kotiin", eli ottamalla yhteyden verkon ylitse asiakasohjelmistoon. Asiakasohjelmiston avulla voidaan etähallita useita palvelimia. On selvää, että etähallinta muodostaa asiakkaan ja palvelimen välille komentokanavan. Toisin kun XtremeRAT, DarkComet salakirjoittaa kaiken liikenteen, joten sen komentokanava on selvästi vaikeammin havaittavissa. Koska kaikki viestit ovat salakirjoitettuja ensimmäisestä tavusta alkaen, joudutaan palvelinohjelmisto takaisin mallintamaan (engl. reverse engineering) viestien purkamista varten. Takaisin mallinuksen tavoitteena on selvittää salakirjoitusalgoritmi ja löytää tätä kautta ohjelmiston käyttämä salausavain.

Muiden RAT-ohjelmistojen tapaan DarkComet tarjoaa useita eri toiminnallisuuksia informaation varastamiseen ja vakoiluun. Tuettuja toiminallisuuksia ovat muun muassa tiedostojen etähallinta, työaseman etähallinta komentorivikehoitteella, etäskriptaus, välityspalvelin, Windows-palveluiden sekä -rekisterin etämanipulointi, etätyöpöytä näkymä ja web-kameran sekä mikrofoonin kaappaus. Ohjelmiston tarjoamat kaikki toiminallisuudet on esitetty kuvassa 13.



Kuva 13. *DarkComet-ohjelmiston toiminnallisuudet*

Tässä esiteltävä versio 5.3.1 on viimeinen täydet ominaisuudet sisältävä DarkComet-ohjelmisto. Ohjelmisto sai alkunsa vuonna 2008 ja sen tekijä ranskalainen Jean-Pierre

Lesueur lopetti ohjelmiston kehittämisen vuonna 2012 edellä esitettyjen Syyrian tapahtumien jälkeen. Viimeiseksi versioksi jäi kuitenkin 5.4.1, mutta tämä ei sisällä lainkaan palvelimen rakentajaa. Vaikka ohjelmistoa ei enää aktiivisesti kehitetä sen tekijän toimesta, ovat sen tajoamat toiminnallisuudet varmasti monille erilaisille toimijoille houkuttavia. Edellä esitetystä kuvasta 13 voidaan havaita, kuinka monipuolisesti DarkComet tarjoaa kohdejärjestelmän täydellisen hallinnan. Tästä johtuen voidaan olettaa, että DarkComet tulee säilymään vielä skriptaajien, kyberrikollisten ja kyberoperaatioita toteuttavien ryhmien työkalupakissa. (Fidelis Cybersecurity 2015.)

6.4 RAT-ohjelmistojen liikenneanalyysi

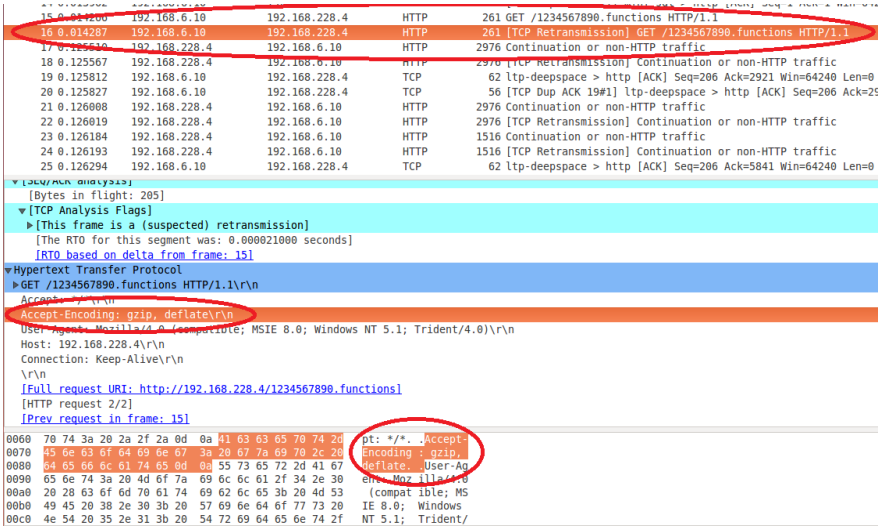
Takaporttiohjelmistojen kirjoittajat pyrkivät kätkemään haitallisen koodin kohdejärjestelmässä, jotta sitä ei havaittaisi ja toiminta voisi näin ollen jatkua mahdollisimman pitkään. On selvää, että hyökkääjä ei esimerkiksi halua kohdejärjestelmän haittaohjelmien torjuntaohjelmiston havaitsevan takaporttiohjelmistoa. APT-kampanjoissa käytetyissä RAT-ohjelmistoissa on ominaisuuksia ohjelman kätkemiseen kohdejärjestelmässä. Yleisesti tunnettuja menetelmiä ovat "pakkaajan" (engl. packer) tai "salakirjoittajan" (engl. crypter) käyttäminen. Näiden pääasiallinen tarkoitus on juurikin kätkeä haitallinen ohjelma haittaohjelmien torjuntaohjelmistolta. Pakatun tai salakirjoitetun ohjelmakoodin havaitseminen ei onnistu pelkästään vertaamalla olevassa olevia haittaohjelmien allekirjoituksia (engl. signature).

Kun kyse on takaporttiohjelmistosta, niin ohjelman toimintaperiaatteen mukaisesti sen kuitenkin jossain vaiheessa täytyy toimia tietokoneen keskusmuistissa selväkielisenä. Muutoin hyökkääjä ei kykene hyödyntämään takaporttiohjelmiston ominaisuuksia. Tämäkin on mahdollista vielä kätkeä, mutta jos kohdejärjestelmää halutaan hallita verkon yli, tulee sen "soittaa" C2-palvelimelle. Kuten luvussa 4 on esitetty, liikenteen hämäännyttämismenelmiä on monia. RAT-ohjelmistoissa liikenteen hämäännyttämisellä pyritään kätkemään asiakkaan ja palvelimen välinen liikenne muun liikenteen sekaan siten, että sitä ei havaittaisi. Seuraavissa alaluvuissa esitetään kahden edellä esitetyn RAT-ohjemiston liikenneanalyysi.

6.4.1 XtremeRAT 3.7 liikenneanalyysi ja komentokanavan havainnointi

Useilla sovelluksilla on tietyt tunnistettavat piirteet niiden lähettäessä liikennettä toisilleen. Usein tunnistaminen voidaan toteuttaa sovelluksen kättelyn perusteella, koska juuri yhteyden muodostaminen on yksilöllistä tietylle sovellukselle ja luonnollisesti salatun hyötykuorman perusteella on vaikea tehdä luokittelua. Koska tarkoituksena on selvittää haitallisen liikenteen havaitsemista, on oleellista selvittää RAT-ohjelmistojen suorittama yhteyden muodostamiskättely.

Tässä esiteltävällä ExtremeRAT 3.7 -ohjelmalla on käytännössä kaksi eri kättelyä. Kättely riippuu siitä missä vaiheessa ohjelman käyttöä ollaan. Kuten todettu, asennusvaiheessa tarvitaan uhrin toimenpiteitä. Luonnollisesti palvelinohjelmisto ei lähetä mitään, ennen kuin se on asentunut kohdekoneeseen. Asennusvaiheessa tapahtuva kättely tapahtuu palvelimen muodostaessa yhteyden asiakasohjelmistoon. Tästä voidaan käyttää nimitystä "asennuskättely". Palvelin suorittaa HTTP GET -pyynnön hyökkääjän koneelle palvelimen asetuksissa konfiguroituun TCP-porttiin. Pyynnössä lähetetään selväkielisenä palvelimen asetuksissa konfiguroitu "yhteyden" salasana. Oletussalasanana toimii 1234567890, jolloin pyyntö on muotoa "GET /1234567890.functions HTTP/1.1". Ohjelma pakkaa osan palvelimen ja asiakkaan välisestä tietoliikenteestä käyttäen deflate-algoritmia ja naamioi komennot ja komentokanavan HTTP-liikenteeksi. Kuvassa 14 on esitetty Wireshark-ohjelman kaappaus XtremeRAT-ohjelman kättelystä sen asennusvaiheessa.



Kuva 14. *XtremeRAT-ohjelman "asennuskättely"*

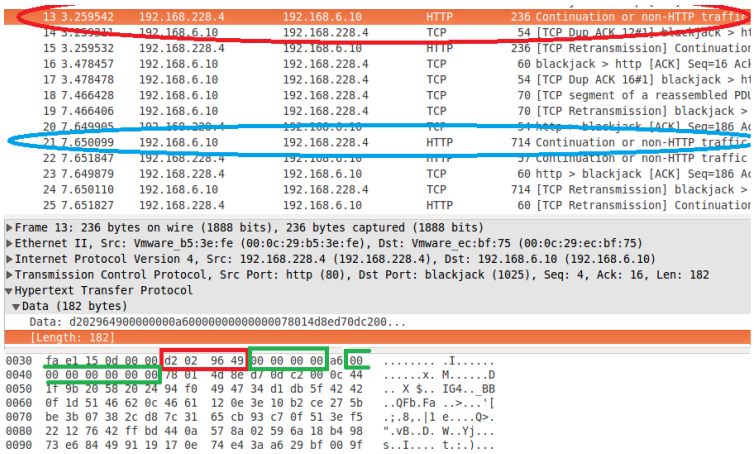
Toinen ja hieman erilainen kättely suoritetaan silloin, kun palvelinohjelma on jo asennettu ja esimerkiksi uhrin työasema käynnistetään uudelleen. Tällöin työaseman käynnistyessä palvelinohjelma käynnistää itsensä tietokoneen keskusmuistiin ja yrittää ottaa yhteyttä asiakasohjelmistoon. Tästä voidaan käyttää nimitystä "peruskättely". Denbow & Hertz (2012) esittävät XtremeRAT 3.6 -version peruskättelyn, joka ei juurikaan poikkea tässä esitettävästä uudemmasta 3.7 versiosta. Tässä esitettävän kättelyn selvittämisessä on käytetty pohjana Denbow & Hertz (2012) esitystä. Käytännössä kättely toimii seuraavasti:

- Palvelin ilmoittaa oman ohjelmistoversionsa → ascii = "my version|3.7".
- Asiakas vastaa kuittausviestillä → "58 0d 0a".
- Asiakas lähettää deflate-algoritmilla pakatun viestin, joka alkaa palvelimen asetuksiin konfiguroidulla salasanalla. Salasana on esitetty little endian -muodossa. Viesti sisältää myös täytteen ennen varsinaista viestiä. (Denbow & Hertz 2012.)

- Osa kättelyviesteistä on selväkielisiä, jolloin Wireshark-ohjelmalla voidaan lukea ascii-merkit, mutta osa viesteistä on pakattu tai salattu

Deflate-algoritmilla pakattu viesti voidaan avata esimerkiksi Gzip-ohjelmalla, joka hyödyntää samaa pakkausalgoritmia. Jotta viesti voidaan avata em. ohjelmalla, tulee tähän lisätä heksaeditorilla pakattua tiedostomuotoa vastaava otsikko. Otsikko voidaan lisätä hyödyntämällä heksaeditoria. Tiedoston alkuun tulee lisätä heksaluvut "\x1f\x8b\x08\x00\x00\x00\x00\x00".

Peruskättelyn kolmannessa viestissä, joka on kuvattu kuvassa 15, esitetään asiakkaan lähettämä 182 tavua sisältämä deflate-algoritmilla pakattu viesti. Kuvassa punaisella merkitty 13. paketti on kyseinen asiakkaan pyyntö, johon palvelin vastaa sinisellä merkityllä paketilla 21. Asiakkaan pyynnön hyötykuorma alkaa kuvassa punaisella suorakaiteella. Tämä sisältää edellä mainitun palvelimen asetuksiin konfiguroidun salasanan little endian -menetelmällä koodattuna. Vihreällä merkityt heksaluvut ovat viestin hyötykuormassa täytettä. Asiakkaan lähettämä pyyntö sekä palvelimen suorittama vastaus on avattu yksityiskohtaisemmin kuvan 15 alla.



Kuva 15. *ExtremeRAT-ohjelman peruskättelyn kolmas viesti*

Edellä mainitun ExtremeRAT-asiakkaan pyynnön hyötykuorma on kokonaisuudessaan esitetty alla heksalukuina:

```
d2 02 96 49 00 00 00 00 a6 00 00 00 00 00 00 00 78 01 4d 8e d7 0d c2 00
0c 44 1f 9b 20 58 20 24 94 f0 49 47 34 d1 db 5f 42 42 0f 1d 51 46 62 0c
46 61 12 0e 3e 10 b2 ce 27 5b be 3b 07 38 2c d8 7c 31 65 cb 93 c7 0f 51
3e f5 22 12 76 42 ff bd 44 0a 57 8a 02 59 6a 18 b4 98 73 e6 84 49 91 19
17 0e 74 e4 3a a6 29 bf 00 9f 2a 3d e2 ec 88 61 91 a4 c2 92 3e 53 d6 0c
c9 30 22 41 17 1b 8f 06 75 56 94 69 73 27 cd 9e 1b 79 06 e4 98 e8 bb 23
57 ed 0c 5d 79 f2 f0 95 ef cb 2d ad 2f 2c e9 5d a5 39 e2 b8 2e 62 62 4f
ec 6a 72 c4 b6 92 cc ef e6 0d 00 e0 29 69
```

Kuten esitetty kuvassa 15, viestin alku sisältää palvelimen asetuksiin konfiguroidun salasanan, joka on esitetty little endian -muodossa. Salasana näkyy yllä olevassa viestis-

sä punaisella. Little endian -koodaus tulee muuntaa big endian -muotoon. Tästä muunnoksesta saatu heksaluku muunnetaan vielä desimaaliluvuksi. (Denbow & Hertz 2012.) Muunnokset tapahtuvat seuraavasti:

```
d2 02 96 49 → 499602d2 → 1234567890
```

Merkkijono 1234567890 on hyökkääjän konfiguroima salasana palvelimelle. Tässä tapauksessa kyseinen merkkijono on myös oletussalasana. Kuten kuvassa 15, myös yllä olevassa kokonaisviestissä vihreällä merkityt luvut ovat täytettä. Lukujen välissä on arvo "a6", joka tarkoittaa heksalukuna varsinaisen viestin sisällön tavumäärää. Desimaaliluvuksi muunnettuna "a6" on 166. Näin ollen varsinainen viesti alkaa ensimmäiseltä riviltä seitsemännestätoista tavusta, jonka arvo on "78". Kun kyseisen viestin sisältö puretaan edellä esitetyillä keinoilla Gzip-ohjelman avulla, saadaan seuraavanlaiset merkkijonot:

```
maininfo������###�a"a"a
G7biEBL0Qhut2FgwrSaZPomeKU4p136JjVflXAY5T8dNMkHRz9qyDWCcnsx
90dd3e7e19b35baa54015d0b4a08f2d0
```

Ensimmäinen edellä esitetty merkkijono on asiakkaan "main info" -pyyntö palvelimelle. Asiakas pyytää palvelinta lähettämään itsestään tietyt "perustiedot", kuten muun muassa konfiguroitu palvelimen nimi, tietokoneen isäntänimi, kirjautunut käyttäjätunnus sekä tunnuksen taso, IP-osoite, käyttöjärjestelmä, palvelinlaitteen suorittimen tiedot, ExtremeRAT-ohjelman versio ja aikaleima. Tämä on esitetty tarkemmin alla. Toinen merkkijono on satunnainen merkkijono. Kolmas merkkijono on virheellisesti muodostettu md5-tiivistesumma sanasta ExtremeRAT.

Palvelimen vastaus "main info" -pyyntöön on myös pakattu deflate-algoritmilla, joten vastaus pitää purkaa samalla menetelmällä kuin pyyntökin. Palvelimen vastaus purettuna on seuraavanlainen:

```
maininfo������###�a"a"a
H����erversServer_IE8WINXP^IEUser(C00A56A9)Finland 192.168.6.10Admin
(x86)IE8WINXP IEUser -����Windows XP Professional (Build: 2600 - Servi
ce Pack: 3.0)  Intel(R) Core(TM) i7-3517U CPU @ 1.90GHz 1073192960
3.7P����2/10/2015 05:53:01
```

ExtremeRAT-ohjelmiston "peruskättelyn" neljän ensimmäisen viestin viestiketju on kuvattu alla. Viestiketjun kaksi ensimmäistä viestiä lähetetään selväkielisenä, mutta seuraavat viestit ovat pakattuja deflate-algoritmilla.

1. Palvelin lähettää viestin, jonka hyötykuormana on ohjelmiston versio → "my version|3.7".
2. Asiakas kuittaa viestin, jonka hyötykuormana on heksaluvut "58 0d 0a".
3. Asiakas lähettää viestin, jonka hyötykuormana on merkkijonot:

a. maininfo
b. G7biEBL0Qhut2FgwrSaZPomeKU4p136JjVflXAY5T8dNMkHRz9qyDWCcnsx
c. 90dd3e7e19b35baa54015d0b4a08f2d0

4. Palvelin vastaa asiakkaan main info -pyyntöön perustiedoilla:
   a. Palvelimen nimi → Servers = Server
   b. Palvelimen isäntänimi, kirjautunut käyttäjätunnus sekä käyttätunnuksen taso → IE8WINXP, IEUser, Admin
   c. Palvelimen IP-osoite → 192.168.6.10
   d. Palvelimen käyttöjärjestelmä → Windows XP Professional (Build: 2600 - ServicePack: 3.0)
   e. Palvelimen suorittimen tiedot → Intel(R) Core(TM) i7-3517U CPU @ 1.90GHz
   f. ExtremeRAT versio ja aikaleima → 3.7P, 22/10/2015 05:53:01

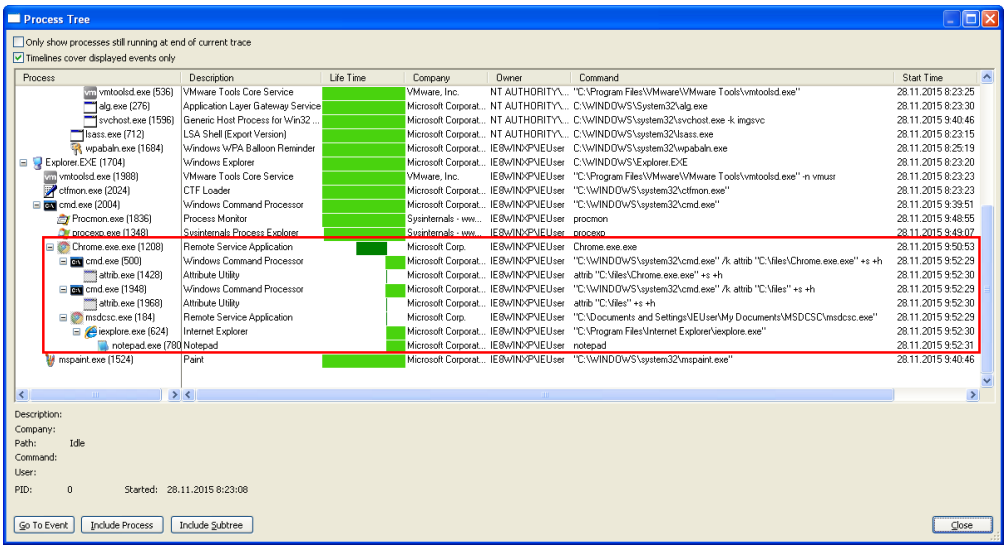
Edellä esitetyn perusteella voidaan todeta, että ExtremeRAT-ohjelmistolla on tunnistettavissa oleva kättely. Kättelyssä on selvät piirteet, joiden avulla ohjelmiston tunnistus verkkoliikenteen perusteella on mahdollista. On myös todettava, että tässä esitetty viestien purkumenetelmä ei toimi kaikkiin ohjelman lähettämiin viesteihin. Esimerkiksi hyökkääjän lähettäessä komentoja ja vastaanottaessa tiedostoja, on kyseinen purkumenetelmä näihin viesteihin kykenemätön. Nämä viestit eivät ole pakattuja, vaan todennäköisesti salattuja, koska viestien sisältö näyttäisi olevan täysin satunnaista tutkittaessa paketteja Wireshark-ohjelmalla. Tämä ei ole kuitenkaan oleellista, sillä ExtremeRAT-ohjelma voidaan tunnistaa esitetyn kättelyn perusteella verkkoliikenteen seasta esimerkiksi laatimalla IDPS-järjestelmään tai DPI-kirjastoon tätä varten oma sääntö.

6.4.2 DarkComet 5.3 analyysi ja komentokanavan havainnointi

Molemmat tässä esitellyt RAT-ohjelmistot sisältävät paljon yhtäläisyyksiä. Kumpikin ohjelmisto on kirjoitettu Pascal-ohjelmointikielellä ja molemmat pyrkivät hämäämään asiakkaan ja palvelimen välisen komentoliikenteen. Ohjelmistojen ominaisuudet ovat lähes samankaltaiset. Kumpaakin on käytetty viime vuosien aikana useissa APT-kampanjoissa, joista uusimmat löydökset DarkComet-ohjelmiston osalta ovat viime vuodelta (Neagu 2015).

Tarkasteltaessa DarkComet-ohjelmiston liikennettä asiakkaan ja palvelimen välillä, voidaan havaita selviä poikkeavuuksia verrattuna XtremeRAT-ohjelmistoon. Toisin kuin XtremeRAT, DarkComet salakirjoittaa kaiken liikenteen. Mitään viestejä ei lähetetä selväkielisenä. Tämä luonnollisesti aiheuttaa haasteita yrittäessä tunnistaa ja luokitella liikennettä. Reaaliaikainen luokittelu on vielä haastavampaa, jos liikennettä koskevaa palvelinohjelmistoa ei ole kyetty takaisinmallintamaan.

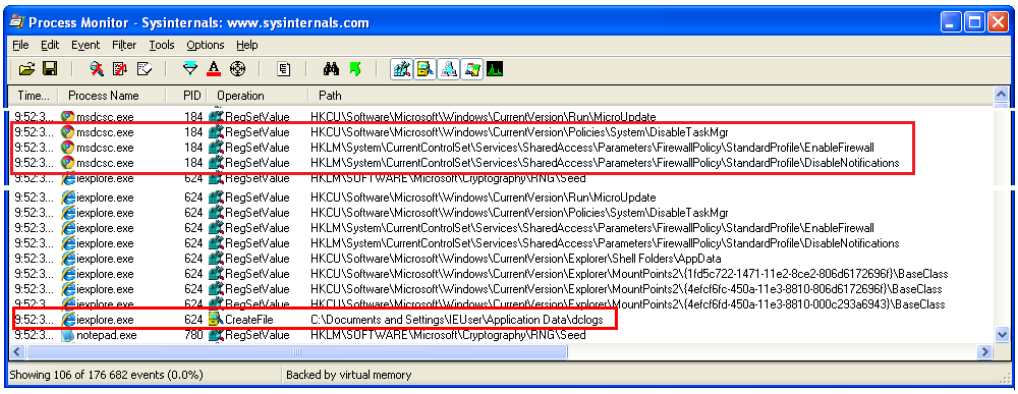
Tutkittaessa DarkComet-palvelinohjelmiston asentumista kohteeseen, havaitaan että varsinainen haitallinen koodi kopioidaan useaan eri prosessiin. Lisäksi erilaisia kirjoitustoimenpiteitä luodaan Windowsin rekisteriin. Näiden toimintojen avulla ohjelmisto pyrkii luomaan ja saavuttamaan pysyvyyden eli persistenssin kohdejärjestelmässä. Kuvassa 16 on esitetty palvelinohjelmiston toiminnot sen asentuessa kohdejärjestelmään.



Kuva 16. *DarkComet palvelimen "asennusprosessi" kohdejärjestelmässä*

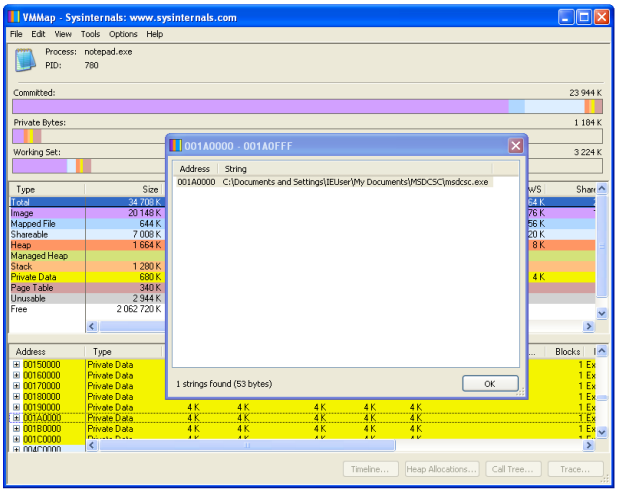
Palvelinohjelmisto on "naamioitu" Chrome.exe.exe -tiedostoksi. Kaksoispääte tiedostonimessä on yleinen tapa, sillä oletuksena Windows-käyttöjärjestelmän resurssien hallinta ei näytä tunnettuja tiedostopäätteitä. Kuvassa 16 Sysinternalsin process monitoring -ohjelma näyttää kuitenkin oikean tiedostopäätteen. Process monitoring -ohjelman avulla havaitaan, että DarkComet muuttaa "C:\files" -hakemiston ja "uhrin" toimesta suoritetun Chrome.exe.exe -tiedoston attribuutteja. Merkillä "+s" merkitään systeemiattribuutti, joka tarkoittaa sitä, että hakemisto tai tiedosto on käyttöjärjestelmän käyttämä ja sitä ei tulisi poistaa tai muuttaa. Toinen käytetty attribuuti "+h" piilottaa tiedoston tai hakemiston. Process monitoring -ohjelmalla havaitaan myös, että haitallinen koodi kopioidaan käyttäjän "My Documents\MSDCSC\msdcsc.exe-tiedostoon. Nimivalinta on varsin mielenkiintoinen, sillä ohjelmiston alkuperäinen tekijä Jean-Pierre Lesueur käyttää itsestään hakkerinimeä DarkCoderSc. Kopioinnin lisäksi haitallinen koodi injektoidaan iexplore.exe ja notepad.exe prosesseihin.

Seurattaessa process monitoring -ohjelmalla msdcsc-prosessin kirjoitustoimenpiteitä havaitaan, että Windowsin rekisteriarvoja muuttamalla kytketään pois päältä muun muassa tehtävien hallinta, Windowsin palomuuri ja automaattiset ilmoitukset. Lisäksi iexplorer-prosessi luo tiedoston nimeltään dclogs, johon kirjoitetaan DarkComet-ohjelmiston loki. Nämä kirjoitustoimenpiteet on havainnollistettu kuvassa 17.



Kuva 17. *DarkComet-palvelinohjelman kirjoitustoimenpiteitä*

Edellä mainittujen palveluiden pois päältä kytkeminen varmistetaan vielä iexplorer-prosessin avulla. Sulkemalla tärkeitä palveluita, kuten tehtävien hallinta, pyritään varmistamaan, että RAT-ohjelmistoa ei havaittaisi. Haitallisen koodin injektointi voidaan osoittaa Sysinternalsin vmmap-ohjelman avulla. Tämä ohjelma kykenee näyttämään ajossa olevien prosessien ascii merkkijonoja keskusmuistista. Alla on esimerkki notepad-prosessin keskusmuistissa olevista merkkijonoista. Kuvassa 18 havainnollistetaan vmmap-ohjelman löydökset. Muistista löytyy viittaus msdcsc.exe -tiedostoon.



Kuva 18. *Prosessin muistiavaruuden tutkiminen vmmap-ohjelmalla*

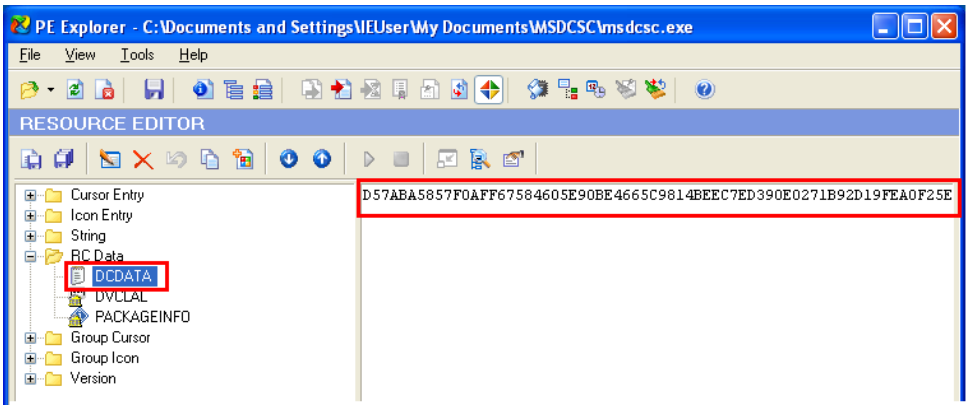
DarkComet-ohjelmisto salakirjoittaa kaiken verkkoliikenteen. Tämän vuoksi on oleellista tutkia itse ohjelmiston toimintaa, jotta voidaan purkaa sen toteuttama salakirjoitus. Salakirjoituksen purkamiseen tarvitaan palvelinohjelmiston takaisinmallinnusta. Takaisinmallinnus on haastavaa, koska siinä tarvitaan vahvaa symbolista konekieliosaamista. Takaisinkääntäjät (engl. disassembler) ovat ohjelmia, joiden avulla konekielinen ohjelma voidaan palauttaa assembler-koodiksi, eli symboliseksi konekieleksi. IDA, OllyDbg ja PE Explorer ovat esimerkkejä ohjelmistoista, joiden avulla voidaan suorittaa takaisinmallinnusta. DarkComet-ohjelmistoa on tutkittu varsin paljon, joten sen toiminnasta ja takaisinmallinuksesta löytyy useita raportteja. Esimerkisi Edwards (2012), Denbow & Hertz (2012) ja Fidelis Cybersecurity (2015) ovat analysoineet ja takaisin-

mallintaneet kyseisen RAT-ohjelmiston. Erityisesti Fidelis Cybersecurity (2015) on laatinut varsin kattavan raportin DarkComet-ohjelmistosta.

RAT-ohjelmiston palvelimen luonti suoritetaan graafisella käyttöliittymällä, jonka avulla määritetään useita palvelimen toimintaan liittyviä toimintoja, kuten esimerkiksi polku, johon palvelinohjelmisto asennetaan ja palvelimen tiedostonimi. Hyökkäjän on myös määritettävä IP-osoite tai DNS-nimi sekä TCP-porttinumero, johon palvelin ottaa asentumisen jälkeen yhteyttä. Oleellista tietoliikenteen salakirjoituksen avaamiseksi on kuitenkin salasana, jonka hyökkääjä määrittää palvelimen konfiguroinnissa. Kyseinen salasana on osa salausavainta, jonka avulla asiakkaan ja palvelimen välinen tietoliikenne salakirjoitetaan. Tietoliikenteen salausavain muodostetaan kahdesta osasta. Toinen osa avaimesta tulee edellä esitetystä hyökkääjän määrittämästä salasanasta ja toinen osa muodostuu tietystä "kovakoodatusta" salasanasta. Kovakoodattu salasana on erilainen eri ohjelmistoversioissa ja se toimii myös palvelimen salakirjoitetun konfiguraatiotiedoston salausavaimena. Palvelimen konfiguraatiossa määritettävä oletussalasana on lähes sama kuin XtremeRAT-ohjelmistossa. DarkComet-ohjelmistossa se on "0123456789".

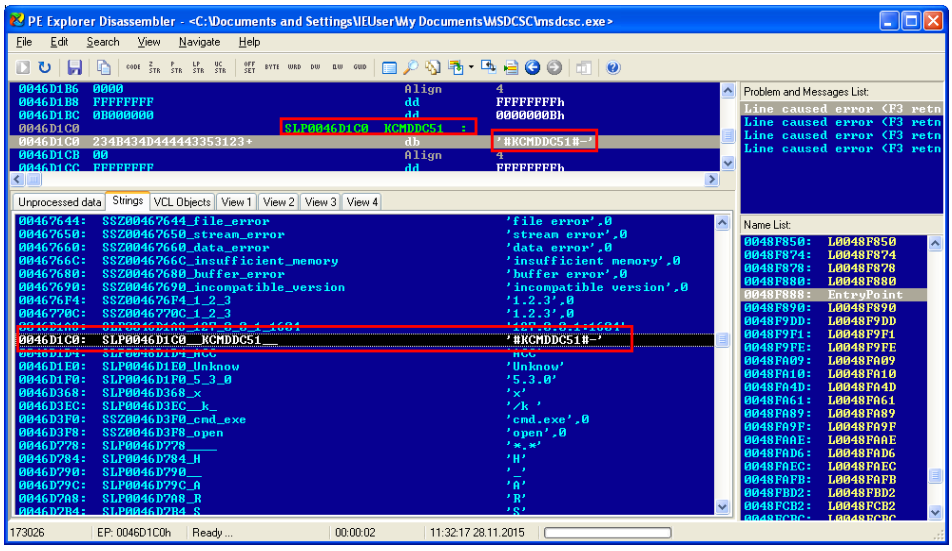
Palvelimen konfiguraatiossa määritetyn salasanan etsimiseen uhrin koneelta on useita tapoja. Koska DarkComet-ohjelmistoa on tutkittu paljon, on salasanan etsimiseen kehitetty jopa oma python-ohjelma. Kyseinen ohjelma on esitetty liitteessä D (Breenin 2014). Kuten todettu, salasana voidaan lukea DarkComet-palvelimen konfiguraatiotiedostosta, joka sijaitsee salakirjoitettuna uhrin tietokoneella olevassa suoritettavassa palvelintiedostossa tai sen suorittamisen jälkeen myös keskusmuistissa. Fidelis Cybersecurity (2015) esittää DarkComet-raportissaan konfiguraation sijainnin suorittettavasta palvelintiedostosta. Tämä voidaan havaita esimerkiksi PE Explorer -ohjelman resurssieditorilla (engl. resource editor), joka on esitetty kuvassa 19. Konfiguraatio sijaitsee msdcsc.exe-tiedostossa olevassa "DCDATA"-resurssissa, jonka sisältö näyttäisi olevan täysin satunnaista.

Konfiguraation salakirjoituksen avaamiseen tarvitaan edellä mainittua kovakoodattua salausavainta, joka toimii osana liikenteen salausavainta. Kovakoodattu salausavain löydetään esimerkiksi tutkimalla suoritettavan palvelinohjelmiston selväkielisiä merkkijonoja heksaeditorilla tai takaisinkääntäjällä. Toisaalta, johtuen tutkimuksen suosiosta, DarkComet-ohjelmiston käyttämät eri versioiden "kovakoodatut" salasanat ovat yleisesti tiedossa. Kuten todettu, salasana on erilainen ohjelmiston eri versioissa. Myös liitteessä D esitettävä DarkComet.py-ohjelma sisältää kaikki mahdolliset "kovakoodatut" salausavaimet, joita ajetaan vuorotellen konfiguraatiotiedostoa vasten. Konfiguraation salakirjoituksessa käytettävän salasanan tunnettavuus säästää merkittävästi aikaa liikenteen salakirjoituksessa käytettävän salasanan selvittämisessä, koska takaisinmallinnusta ei tarvitse toteuttaa.



Kuva 19. *DarkComet RAT -palvelimen konfiguraatiotiedoston sijainti*

Tässä esitettävän DarkComet 5.3 -ohjelmiston kiinteä salausavain on "#KCMDDC51#-890" ja käytettävä salausalgoritmi on RC4. (Fidelis Cybersecurity 2015). Kiinteä salausavain voidaan tarkistaa PE Explorer -ohjelmalla. Tarkistaminen onnistuu nopeasti etsimällä merkkijonovälilehdeltä tunnettua merkkijonoa. Kuvassa 20 esitetään PE Explorer-ohjelmalla havaittu merkkijono "#KCMDDC51#-" msdcsc.exe-tiedostosta. Merkkijono ei kuitenkaan toimi salausavaimena ilman "890"-jälkiliitettä. Jälkiliite on sama kaikissa DarkComet versioissa (Fidelis Cybersecurity 2015).



Kuva 20. *PE Explorer ohjelman takaisinmallinnus*

Kun kiinteä salausavain on selvitetty, salakirjoitettu DCDATA-resurssi voidaan avata. Tämä voidaan suorittaa liitteessä D esitetyn valmiin python-ohjelman avulla tai se voidaan avata yksinkertaisella python komennolla komentoriviltä (Fidelis Cybersecurity 2015). Jälkimmäisellä tavalla konfiguraatio puretaan selväkieliseksi seuraavasti:

```
python -c "from Crypto.Cipher import ARC4; print ARC4.new(\"#KCMDDC51#-
890\").decrypt(\"DCDATA_sisältö\".decode('hex'))"
```

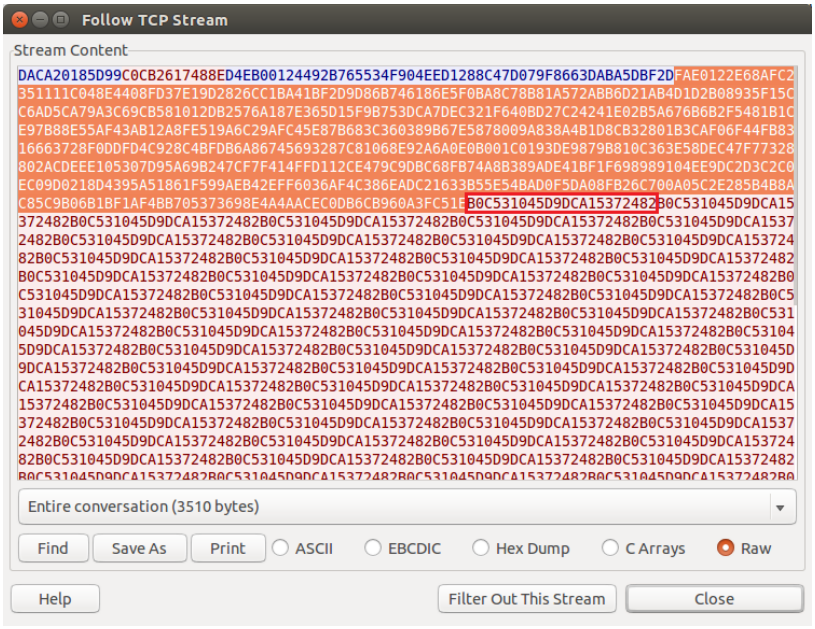
Molemmilla tavoilla tulokseksi saadaan selväkielinen konfiguraatio, joka sisältää myös edellä mainitun salasanan. Purettu konfiguraatio on esitetty liitteessä E, jossa etsittävä salasana on merkitty punaisella.

Kaikki hyökkääjän palvelinkomponentille tekemät asetukset voidaan havaita puretusta konfiguraatiosta. Salasanan löytäminen on kuitenkin oleellista, jos asiakkaan ja palvelimen välinen tietoliikenne halutaan nähdä selväkielisenä. Asiakkaan ja palvelimen välinen liikenne salakirjoitetaan salausavaimella, joka muodostuu aikaisemmin esitetyn kovakoodatun salasanan sekä hyökkääjän konfiguroiman salasanan yhdistelmänä. Tämä on osoitettu useissa DarkComet-ohjelmistoa koskevissa analyyseissä (Fidelis Cybersecurity 2015, Santinumi 2012, Denbow & Hertz 2012). Edellä mainitut analyysit on laadittu palvelinohjelman takaisinmallinuksen avulla.

Asiakkaan ja palvelimen yhteydenmuodostuskättely eroaa muutenkin kuin salauksen osalta verrattuna XtremeRAT-ohjelmistoon. DarkComet:n tapauksessa hyötykuorman lähettämisen aloittaa asiakas, eikä palvelin. Tämä poikkeaa perinteisestä takaporttiohjelmistojen liikennemallista, jossa palvelin "soittaa" kotiin, eli hyökkääjälle ja ilmoittaa itsestään. Tosin DarkComet-ohjelmiston tapauksessakin palvelin muodostaa ensin TCP-yhteyden, eli lähettää SYN-paketin asikkaalle. Fidelis Cybersecurity (2015) esitää raportissaan mallin, kuinka asiakaan ja palvelimen välinen salakirjoitettu liikenne voidaan purkaa selväkieliseksi python-ohjelmointikielen avulla. Tätä varten ei tarvitse välttämättä kirjoittaa ohjelmaa tiedostoksi, vaan sen yksinkertaisuuden vuoksi sitä voidaan ajaa suoraan komentoriviltä. Samaa menetelmää sovellettiin edellä DCDATA-sisällön purkamiseksi selväkieliseksi. Alla havainnollistetaan esimerkkikomento, jossa asiakkaan eli hyökkääjän ensimmäinen viesti puretaan selväkieliseksi. Varsinainen viesti esitetään heksadesimaalilukuna, joka pitää dekoodata ascii-muotoon. Viesti on esitetty alla olevassa esimerkissä vihreällä. Kuten edellä todetiin liikenteen salausavain on muotoa "#KCMDDC51#-8900123456789, joka on alla esitetty punaisella. Viestin purkua varten pythonin salauskirjastosta tulee kutsua ARC4-kirjasto.

```
python -c "from Crypto.Cipher import ARC4; print ARC4.new(\"#KCMDDC51#-
8900123456789\").decrypt(\"DACA20185D99\".decode('hex'))"
```

Kuvassa 21 esitetetään asiakkaan ja palvelimen välinen TCP-virta, joka sisältää myös edellä mainitun yhteydenmuodostuskättelyn. Kuvasta voidaan havaita liikennöivien osapuolien lähettämä hyötykuorma toisilleen. Sinisellä merkityt tavut ovat asiakaan lähettämiä ja punaisella merkityt tavut palvelimen lähettämiä. Oranssilla maalattu tavujono on palvelimen vastaus asiakkaan pyyntöön ilmoittaa perustiedot itsestään. Punaisella suorakaiteella merkitty tavujono havaitaan palvelimen vastauksen jälkeen toistuvasti. Tämä on palvelimen lähettämä "keepalive"-viesti, jolla TCP-yhteyttä pidetään yllä. Kaikki nämä viestit avataan tarkemmin kuvan alla.



Kuva 21. *DarkComet-asiakkaan ja -palvelimen välinen liikenne*

Edellä esitettyä python-komentoa soveltamalla saadaan purettua kaikki kuvan 21 esittämät viestit selväkielisenä. Asiakkaan (192.168.228.3) ja palvelimen (192.168.6.10) välinen viestintä on seuraavanlainen:

```
192.168.228.3     IDTYPE

192.168.6.10      SERVER

192.168.228.3     GetSIN 192.168.6.10|2756210

192.168.6.10      infoesDippa01|192.168.6.10 / [192.168.6.10]:80
                  |IE8WINXP/IEUser|2756210|8s|Windows XP Service Pack 3
                  [2600] 32 bit ( C:\ )|x||FI|Process Explorer -
                  Sysinternals: www.sysinternals.com
                  [IE8WINXP\IEUser]|{d8c22bc0-1471-11e2-93b8-
                  806d6172696f-3221903017}|29%|Finnish FI /  --
                  |28.11.2015 at 9:52:31|5.3.0

192.168.6.10      #KEEPALIVE#
```

Asiakkaan ja palvelimen välisen hyötykuormaa sisältävän viestinnän aloittaa asiakas, joka lähettää IDTYPE-kyselyn palvelimelle. Palvelin vastaa kuittausviestillä, jossa ilmoitetaan vastaajan olevan palvelin. Tämän jälkeen asiakas lähettää pyynnön, jossa palvelinta pyydetään lähettämään informaatio itsestään. Palvelimen viesti on hyvin samankaltainen XtremeRAT-ohjelmiston kanssa. Informaatio sisältää muun muassa palvelimen IP-osoitteen, tiedot käyttöjärjestelmästä, DarkComet-versio, aikaleima ja uhrilla ajossa oleva ohjelma.

Kuten edellä havainnollistettiin Sysinternals-työkalujen avulla, haitallinen koodi injektoidaan iexplorer-prosessiin, jolla pyritään kiertämään mahdollinen isäntäpalomuuri (engl. host firewall). Käytettäessä TCP-porttia 80 voidaan ohittaa organisaation verkon reunalla sijaitseva rautapalomuuri, koska lähes aina TCP 80 -portti eli HTTP-liikenne on ulospäin sallittu. DarkComet liikennettä on vaikea haivata, koska se on salakirjoitettu. Liikenteen reaaliaikaista tunnistusta varten symmetrinen avain tulisi olla tiedossa, jotta viestit voitaisiin purkaa esim. IDPS-järjestelmällä. Teoriassa reaaliaikainen havainnointi voidaan myös toteuttaa selvän anomalian avulla, jos käytetään TCP 80-porttia. DarkComet ei naamioi liikennettä, vaan se ainoastaan salaa sen. DarkComet-palvelin ei siis tee HTTP-liikenteelle ominaista GET-pyyntöä, eikä myöskaan asiakasohjelmisto suorita pyyntöön vastausta. Tämä siis poikkeaa HTTP-protokollalle normaalista toiminnasta. Toinen teoreettinen tunnistusmalli voidaan rakentaa asiakkaan ja palvelimen välisen viestinnän pakettikoosta. Pakettikokoon perustuvan analyysin avulla voidaan kehittää kyseisen ohjelmiston liikennemalli. Kuten aikaisemmin on perusteltu, usein ohjelmistojen muodostamat kättelyt ovat yksilöllisiä ja niissä on selvät tunnistettavat piirteet. Obfuskointimenetelmät, kuten obfs3 ja scramblesuit yrittävät peittää nämä piirteet. DarkComet-ohjelmiston asiakkaan ja palvelimen välisen viestinnän pakettikokokuvaaja on esitetty luvussa 8, jossa esitellään tämän työn tuloksia kokonaisuutena.

7. TESTAUSYMPÄRISTÖ, TESTIDATA JA TESTAUSTAPA

Liikenteen luokittelun tutkimuksessa yksi suurimmista ongelmista on julkisesti saatavilla olevien datakokoelmien saatavuus. Eri sovellusten tuottamasta liikenteestä ei ole julkisesti saatavissa olevaa luotettavaa datakokoelmaa johtuen yksityisyyteen liittyvistä seikoista tai siitä, että tutkijat eivät ole jakaneet käytettyä testidataa tutkimusyhteisölle. Tämä koskee erityisesti tilannetta, jolloin halutaan luokitella kaikki mahdollinen liikenne. Jotta tuloksesta voidaan olla varmoja, tulee olla käytössä ns. pohjatotuus (engl. ground-truth), johon automaattisen luokittelun tuloksia verrataan. On selvää, että kyseisen pohjatotuuden luominen on haasteellista silloin, kun luokiteltavien sovellusten määrä nousee useisiin satoihin.

Liikenteen hämäännyttämisen osalta tutkimus on keskittynyt pääosin uusien hämäännyttämismenetelmien DPI-läpäisykykyyn. Eri hämäännyttämismenetelmien vertailututkimuksia on vähän, jolla on myös suora yhteys valmiiden avoimien datakokoelmien saatavuuteen. Oikeastaan ainoa selkeä hämäännyttämismenetelmien vertailututkimus on tässäkin työssä viitattu "Breaking and Improving Protocol Obfuscation" (Hjelmvik & John 2010), joka pääasiassa keskittyy laadittuun SPID-algoritmiin ja sen liikenteen luokittelukykyyn. Hjelmvik & John (2010) eivät avoimesti tarjoa testausdatakokoelmaa muiden käyttöön.

Edellä esitetyistä syistä ja tässä tutkimuksessa käytettyjen hämäännyttämismenetelmien kokoelmasta johtuen, kaikki käytetty liikenne on luotu itse testiympäristössä. Itse luodulla datalla saavutetaan myös yksi tämän työn merkittävistä tavoitteista, joka on perehtyminen liikenteen hämäännyttämismenetelmiin ja niitä toteuttaviin avoimen lähdekoodin ohjelmistoihin. Tässä luvussa kuvatuissa testiympäristöissä on konfiguroitu käyttöön valitut hämäännyttämismenetelmät sekä liikenteen luokittelukirjastot. Näin saadaan muodostettua näkyvyys sekä hämäännyttämisen että liikenteen luokittelun näkökulmista.

Tässä luvussa esitetään perusteet testiliikenteen valinnalle ja generoinnille, testausympäristö, jossa tutkimus on toteutettu sekä liikenteen generointiin liittyvät toiminnot, konfiguraatiot ja rajoitteet. Lopuksi kuvataan itse testausmenetelmä. Esittely, konfiguraatiot ja testausmenetelmän kuvaukset on laadittu tarkasti mahdollistaen tulosten toistettavuuden.

7.1 Obfuskoidun testiliikenteen tuottamisen haasteet ja liikenteen kerääminen

Liikenteen generoinnin haasteena on usein laadukkaan datan generointi. Käytettäessä esimerkiksi koneellisesti luotua testiliikennettä, on mahdollista, että liikenne ei jäljittele ihmisen tuottamaa liikennettä. WWW-sivujen selailu ja tätä kautta verkkoselaimen tuottama liikenne on monesti epäsymmetristä siten, että selain hakee pyydetyn verkkosivun ja tämän sisältämät objektit, jonka jälkeen käyttäjä lukee sisältöä. Sisällön lukemisen aikana verkkoselaimen ja palvelimen välinen liikenne on monesti vähäistä tai sitä ei ole lainkaan. Riippuen verkkoselaimesta ja palvelimen toteutuksesta, palvelin voi pitää tilatietoa istunnosta evästeiden avulla tai pitää yhteyttä päällä "keepalive" -toiminnon avulla, joka on HTTP 1.1 -versiossa suoritetaan palvelimen päässä ja sen kesto on muutamia sekunteja. Liikenteen tuottaminen saattaa siis vaihdella sen mukaan kuinka kauan käyttäjä lukee sisältöä. Eri verkkoselaimet käsittelevät myös usein liikennettä eri tavoin, joten näiden tuottamassa liikenteessä on eroa (Zhioua & Langar 2014). WWW-sivujen selailu on vain yksi esimerkki, HTTP 1.1 -versiossa kaikki yhteydet ovat lähtökohtaisesti "persistent" -yhteyksiä, jolloin ei käytetä erillisiä "keepalive" -viestejä (Totty, Sayer, Reddy, Aggarwal & Gourley 2002), kun taas VPN-yhteyksissä näitä käytetään. Toisaalta, jos VPN-yhteyden sisällä ei liikennöidä, tunneli voidaan katkaista. Tällöin seuraavalla kerralla, kun liikennettä lähetetään, tunneli neuvotellaan uudestaan. On myös mahdollista, että tunneli pidetään yllä, vaikka liikennettä ei lähetettäisi. On selvää, että eri sovellukset ja niiden erilaiset konfiguraatiot tuottavat hyvin erilaista liikennettä. Testiliikenteen tuottamisessa haasteena on jäljitellä ihmisen tuottamaa dataa, sekä ottaa huomioon eri tuotteet sekä näiden konfiguraatiot, jotta mahdollisimman kaatava tulos saataisiin aikaiseksi.

Kuten edellä on todettu liikenteen hämäännyttäminen voidaan jakaa ylätasolla kahteen eri kategoriaan: sovellusprotokollan tyypillisten piirteiden poistaminen, jotta liikennettä ei voida luokitella tai liikenteen naamiointi joksikin toiseksi liikenteeksi siten, että se luokitellaan väärin. Kun liikennevuo pyritään naamioimaan toiseksi muuttamalla sen tilastollisia piirteitä, on hyvä huomioida yllä esitetyt seikat. Esimerkiksi Dust-protokolla tarvitsee "oikean" liikenteen kaappauksen naamiointia varten. Näiden syiden vuoksi tässä työssä on päädytty ihmisen tuottamaan testiliikenteeseen siten, että liikennettä ei generoida skriptien tai ohjelmakoodin avulla, vaan kaikki toiminnot suoritetaan siten, kuin ihminen ne suorittaisi. Liikenne tallennetaan pcap-tiedostoiksi Wireshark-ohjelman avulla ja pcap-tiedostot ajetaan uudelleen valittujen DPI-kirjastojen / luokittelijan lävitse. Pcap-tiedostojen avulla voidaan varmistua testaamisen toistettavuudesta. Niin kutsuttu pohjatotuus on melko yksinkertainen laatia, koska hämäännytettäviä sovelluksia on vähän. Pääpaino on kuitenkin hämäännytetyn liikenteen havaitsemisessa.

7.2 Testidatan rajoitteet ja hämäännytettävän liikenteen valinta

Tutkittaessa liikenteen hämäännyttämistä ja sen tunnistamista on otettava huomioon tietyt rajoitteet. Avoimesti saatavilla olevat hämäännyttämismenetelmät toimivat ilman ohjelmistokehitystä vain tiettyjen sovellusten kanssa. Esimerkiksi edellä esitetyt obfsproxy ja sen kuljetusliitännäiset toimivat sovellusten kanssa, jotka tukevat SOCKS 5 -välityspalvelinta. Python toteutus obfsproxy-ohjelmasta sisältää SOCKS 5 -välityspalvelimen. Fteproxy ei taas sisällä SOCKS 5 -välityspalvelinta, jolloin tämä pitää hankkia muuta kautta. MSE taas sisältyy suoraan useisiin Bittorrent asiakassovelluksiin. Syy siihen, miksi avoimen lähdekoodin hämäännyttämisohjelmistot toimivat lähtökohtaisesti tiettyjen sovellusten kanssa johtuu siitä, että tietyt sovellukset ovat Internet-sensuuria toteuttavissa maissa estetty. Näille ohjelmistoille on usein kehitetty suoraan tuki. Tällaisia ohjelmistoja tai protokollia ovat esimerkiksi OpenVPN tai SSH.

OpenVPN ja SSH eivät ole hämäännyttämismenetelmiä, vaan niiden tavoitteena on säilyttää liikennöivien osapuolien luottamuksellisuus tunneloinnin ja liikenteen salakirjoituksen avulla. Molempien sovellusten liikennettä on estetty esimerkiksi Iranissa ja Kiinassa (Into the Void 2012). Tämä on selvä merkki siitä, että sovelluksilla on yhteyden muodostusvaiheessa suoritettavassa kättelyssä selkeät piirteet tai muutoin tunnistettava liikennemalli.

Haitallisen liikenteen generointi on tässä selkeämpää, koska edellä esitettyjen ja valittujen RAT-työkalujen liikenteessä ei ole saman kaltaisesti eri variaation mahdollisuuksia kuin edellä esitetyissä hämäännyttämismenetelmissä. Tutkimuksessa esitetyt RAT-työkalut käyttävät salattua sekä pakattua liikennettä, jota voidaan ajaa tiettyyn TCP-porttiin. Kyseinen TCP-portti voidaan vaihtaa, mutta salausperusteita ei ole mahdollista vaihtaa käyttäjän toimesta.

Testidatan generointia varten ei ole mielekästä hämäännyttää kovin montaa eri sovellusta, koska pääpaino on tässä hämäännytetyn liikenteen havaitsemisessa ja sen tulkinnassa liikennettä tutkivan sensorin näkökulmasta. Avoimesti saatavien valmiiden obfuskointiohjelmistojen rajallisuus ja niiden suora tuki sovelluksille asettaa omat rajoitteensa obfuskoitavalle liikenteelle ja sovelluksille. RAT-ohjelmistojen osalta ei luonnollisesti tarvitse valita sovellusta, jonka liikenne hämäännytetään. Näistä syistä johtuen hämäännytettävät protokollat/sovellukset, joiden liikenne hämäännytetään ovat:

- OpenVPN
- SSH
- SCP
- HTTP
- Bittorrent.

OpenVPN- tai SSH-tunnellin sisällä voidaan ajaa mitä tahansa sovellusta. Näillä tunnelointitekniikoilla on omat tyypilliset piirteensä. Kuinka tunnelin sisällä ajettava sovellus vaikuttaa näiden tunneleiden muodostaman liikenteen tyypillisiin piirteisiin ei ole tämän tutkimuksen aiheena. Tarkoituksena on kuitenkin selvittää, kuinka eri obfuskointimenetelmät muuttavat tunnelointimenetelmän tai edellä valitun sovellusprotokollan pakettikokoa.

Testiliikennettä generoitaessa hämäännytetty liikenne ajettiin pääosin TCP-porttiin 80, jotta voitiin selvittää, suorittaako DPI-kirjastot väärän tunnistamisen porttinumeron ja satunnaistamisen perusteella. Tässä tapauksessa haluttiin siis havaita, että onko luokittelun tulos HTTP, vaikka ns. naamiointia ei toteutettaisikaan. Tarkoituksena oli myös selvittää hämäännyttämättömän liikenteen luokittelun tulos tapauksissa, joissa sovelluksen oletusporttinumero on vaihdettu joksikin toiseksi tai käytettäessä oletusporttia. Esimerkiksi OpenVPN-liikennettä ajettiin tarkoituksella sekä TCP- että UDP-protokollan päällä eri portteihin, mukaan lukien oletusportit. Käytetyt portit selviävät tarkemmin liikenteen luokittelun tulostaulukoista luvussa 8. Kaikki testausliikenne on esitelty yksityiskohtaisesti yhteenvetotaulukossa liitteessä G.

7.3 Testiympäristö

Testiliikenteen generointi toteutettiin kahdessa eri testiympäristössä. Toiseen ympäristöön konfiguroitiin liikenteen hämäännyttämismenetelmät ja toisessa ympäristössä luotiin haitallinen liikenne RAT-työkalujen avulla. Liikenteen hämäännyttämismenetelmät valittiin luvun 5 mukaisesti, poislukien Dust, joka oli vielä kehitysvaiheessa. Kuten aikaisemmin on esitetty, liikenteen obfuskointi voidaan jakaa kahteen eri pääkategoriaan:

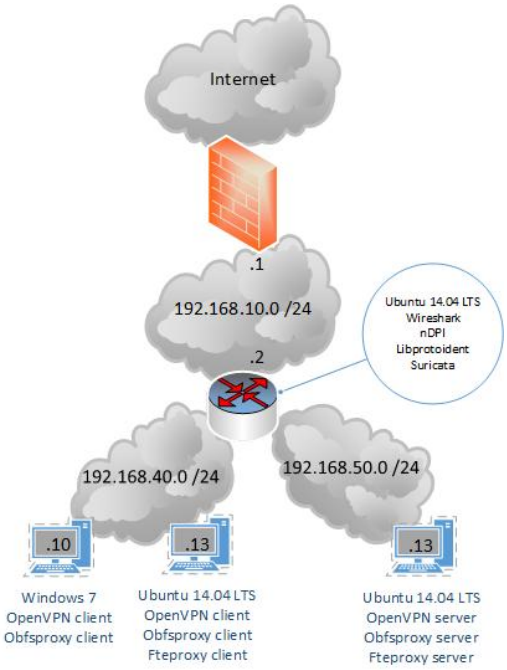
- Liikenteelle tai protokollalle tyypillisten piirteiden poistaminen
- Liikenteen tai protokollan naamioiminen joksikin toiseksi liikenteeksi

Liikenteen tyyppillisten piirteiden poistaminen voidaan suorittaa hyötykuorman hämäännyttämisellä ja liikennevuon hämäännyttämisellä. Vertailun aikaansaamiseksi, molemmista kategorioista valittiin hämäännyttämismenetelmiä. Protokollalle tyypillisten piirteiden poistamiseen valittiin obfsproxy, joka tukee obfs2-, obfs3- ja scramblesuit-obfuskointiprotokollia. Näistä scramblesuit hämäännyttää myös liikennevuon. Liikenteen naamiointiin valittiin Fteproxy, jonka avulla liikenne naamioitiin HTTP- ja SSH-protokollaksi. Fteproxy-hämäännyttämismenetelmän osalta myös HTTP-protokolla naamioitiin HTTP-protokollaksi, jonka avulla pyrittiin havaitsemaan mahdollinen luokittelun muutos. Tästä laadittiin myös pakettikokoanalyysi, sillä oletuksena oli, että pakettikokojakauma muuttuu. DPI-kirjastot luokittelevat liikenteen "tuntematon TCP-protokolla tai tuntematon UDP-protokolla tunnisteella, jos vastaavuutta ei löydetä tuetuista protokollamoduuleista.

Hämäännytetyn liikenteen generointia varten asennettiin virtuaaliympäristö, jossa virtuaalikoneina toimivat, 3 kpl Ubuntu 14.04 LTS -työasemia ja Windows 7 -työasema. Virtualisointi toteutettiin Vmware ESXi 5.5 -virtualisointikerroksen (engl. hypervisor) avulla HP ProLiant DL 380 G6 -palvelimeen. Palvelimen raudan kokoonpano oli seuraava:

- prosessori: Intel(R) Xeon(R) 2,266GHz (4 ydintä)
- keskusmuisti: 30GB
- massamuistin kapasiteetti 1,64TB
- 4 verkkokorttia.

Yksi Ubuntu-työasemista konfiguroitiin testiverkon reitittimeksi. Reitittimeen asennettiin wireshark-verkkoanalysaattoriohjelma, jonka avulla hämäännytetty liikenne kaapattiin pcap-tiedostoiksi. Kyseiseen reitittimeen asennettiin myös valitut DPI-kirjastot (nDPI ja libprotoident) sekä Suricata IDPS-ohjelmisto. Kuvassa 22 esitetään looginen verkkokuva testiympäristöstä.



Kuva 22. *Testausympäristön looginen verkkokuva*

Haitallinen testiliikenne generoitiin eri testausympäristössä kuin muu obfuskoitu liikenne. Tämä ympäristö oli samanakaltainen edellä esitetyn loogisen verkkokuvan kanssa. Erillisessä virtualisoidussa testausympäristössä suoritettiin luvussa 6 esitettyjen RAT-ohjelmistojen avulla erilaisia toimenpiteitä. Edellä mainitut toimenpiteet esitellään tarkemmin liitteesä F, jossa kuvataan yksityiskohtaisesti kaikki generoitu testausliikenne. RAT-ohjelmistojen testausympäristö luotiin VMware player -ohjelmiston avulla, jossa virtuaalikoneina toimivat Ubuntu 14.04 LTS -työasema, Windows 7 -työasema ja kaksi Windows XP -työasemaa. VMware player -ohjelmistoa ajettiin kannettavassa tietoko-

neessa. Ubuntu-työasema toimi tässä testiverkossa reitittimenä Windows-työasemien välillä. Kuten edellä, myös tässä verkossa liikenne kaapattiin Wireshark-ohjelman avulla pcap-tiedostoiksi.

7.4 Hämäännyttämismenetelmien konfiguraatiot

Hämäännyttämismenetelmät konfiguroitiin käyttöön kuvan 21 testiympäristössä. Verkon 192.168.40.0 Windows- ja Ubuntu-työasemiin asennettiin ja konfiguroitiin OpenVPN-asiakasohjelmistot sekä obfsproxy-asiakasohjelmistot. Verkossa 192.168.50.0 oleva Ubuntu-työasemaan asenettiin ja konfiguroitiin em. ohjelmistojen palvelin-versiot. Fteproxy-ohjelmisto asennettiin ja konfiguroitiin ainoastaan Ubuntu-työasemiin samalla asiakas-palvelin periaatteella kuin obfsproxy. Kuten todettu OpenVPN liikennettä voidaan ajaa obfsproxyn päällä, koska obfsproxy-ohjelmistossa on SOCKS5-välityspalvelin, jota OpenVPN tukee. Obfuskointiohjelmistoista käytettiin seuraavia versioita:

- obfsproxy 0.2.13
- fteproxy 0.2.18.

7.4.1 OpenVPN konfiguraatio

Tutkimuksessa käytettiin OpenVPN versiota 2.3.6. Ubuntu-linuxissa OpenVPN-palvelin konfiguroitiin toimimaan oletusportissa 1194 TCP-protokollan päällä. Palvelimella luotiin esijaetut TLS-avaimet, joiden avulla varmistetaan, että VPN-tunnelin neuvottelu aloitetaan vain entuudestaan luotettujen osapuolten välillä. Tämä tuo lisäturvaa palvelunestohyökkäyksiä vastaan, koska paketit ilman TLS-avainta hylätään. Tunnistamaton asiakas ei kykene tällöin luomaan suoritin- ja salauskuormaa palvelinta vastaan, vaan "roskaliikenne" voidaan pudottaa ajoissa (Hardening OpenVPN 2013).

Tunnelin osapuolten tunnistamista varten luotiin X.509 -varmenteet, joita käytetään osapuolten autentikoinnissa. X.509 -varmenteet luotiin Easy-RSA varmenteiden hallintaympäristön avulla, joka tulee OpenVPN-ohjelmiston mukana. Tämän avulla generoitiin oma varmenneympäristö, johon sisältyy oma varmentaja eli CA. Palvelimelle ja asiakkaille luotiin 2048-bitin RSA-avaimet, jotka varmenneympäristön varmentaja allekirjoitti. Kyseisen varmenneympäristön avulla luodaan myös Diffie-Hellman -avaimet symmetrisen salauksen sopimista varten. Tunnelin salausalgoritmina (symmetrinen salaus) käytettiin AES-256-CBC -algoritmia ja pakettien eheyden tarkistuksessa SHA256-algoritmia. Liitteessä G on esimerkki asiakkaan OpenVPN -konfiguraatiosta. Obfsproxyn kanssa OpenVPN-konfiguraatioon lisätään SOCKS-konfiguraatio. Tämä esitetään tarkemmin alla omassa luvussaan.

7.4.2 Obfsproxy konfiguraatio

Obfsproxy ohjelmistosta asennettiin sen python versio. Tutkimuksessa käytettiin Obfsproxy versiota 0.2.13. Tätä varten tarvittiin Ubuntu-linuxiin python-ympäristö. Tämä asennetaan komennolla:

```
apt-get install python2.7
```

Obfsproxy voitiin tämän jälkeen asentaa Ubuntu-linuxiin pythonin omalla pip-pakettien hallintaohjelmalla:

```
pip install obfsproxy
```

Käännety Obfsproxy-ohjelma asentuu hakemistoon:

```
/usr/local/bin
```

Obfsproxy-ohjelmistoa voidaan käyttää suoraan komentoriviltä sekä asiakkaan, että palvelimen päässä. Käytettäessä obfsproxy-ohjelmistoa OpenVPN-ohjelmiston kanssa, tulee tämä huomioida myös OpenVPN konfiguraatiossa. OpenVPN-asiakkaan konfiguraatioon tulee lisätä seuraavat rivit:

```
socks-proxy-retry
socks-proxy 127.0.0.1 1050
```

Esimerkissä lähtevä OpenVPN-liikenne ohjataan SOCKS-proxy -ohjelmalle, joka muuttaa lähdeportiksi 1050. Proxy-ohjelma myös vastaanottaa paluupaketit ja lähettää ne edelleen OpenVPN-asiakasohjelmalle.

Palvelimen päässä Obfsproxy tarvitsee käynnistysparametreiksi käytettävän obfuskointiprotokollan, porttinumeron, johon välityspalvelin ohjaa sille saapuvan liikenteen, lokitustason ja tarvittaessa esijaetun avaimen. Esijaettu avain on ASCII-muodossa. Obfsproxy käynnistetään obfs2 obfuskointiprotokollan ja esijaetun salaisen avaimen kanssa palvelimella seuraavasti:

```
obfsproxy --log-min-severity=info obfs2 --dest=127.0.0.1:1194 --shared-
secret=this_is_shared_secret server 0.0.0.0:80
```

Käynnistys palvelimella voidaan suorittaa myös ilman esijaettua avainta, jolloin komento on seuraava:

```
obfsproxy --log-file=obfsproxy.log --log-min-severity=info obfs2 --
dest=127.0.0.1:1194 server 0.0.0.0:80
```

Obfsproxyn käynnistäminen palvelimella obfs3 obfuskointiprotokollan avulla suoritetaan komennolla:

```
obfsproxy --log-min-severity=debug obfs3 --dest=127.0.0.1:1194 server
0.0.0.0:80
```

Käytettäessä obfsproxy-ohjelmaa scramblesuit obfuskointiprotokollan avulla, palvelimelle laaditaan lyhyt käynnistysohjelma, eli bash-skripti. Bash-skripti tarkoittaa sitä, että skripti on kirjoitettu bash-komentotulkille. Tämä siksi, koska komentoriviltä käynnistäminen kaikkien tarvittavien käynnistysparametrien avulla venyy merkkien osalta melko pitkäksi, joten on selkeämpää käyttää tähän omaa ohjelmaa. Yksi tarvittavista käynnistysparametreista on esijaettu avain. Scramblesuit-protokolla tarvitsee esijaetun avaimen, jonka avulla voidaan varmistua liikennöivistä osapuolista. Ohjelma tiputtaa tunnistamattomat paketit. Esijaettu avain tulee olla Base32-koodattu. Tämän luomiseksi voidaan käyttää tähän omaa ohjelmaa, joka esimerkiksi Ubuntu-linuxissa on "Base32 encoder/decoder". Base32-koodauksen avulla voidaan koodata satunnaista binääri-dataa ASCII-tekstiksi. Esijaettu avain tulee muuttaa ensin binäärimuotoon esim. heksadesimaaliluvuiksi, jonka jälkeen se voidaan koodata Base32-muotoon. Muutoin tarvittavat parametrit ohjelman käynnistämiseen ovat samankaltaiset kuin käytettäessä obfs2- tai obfs3-protokollia Alla esimerkki edellä mainitusta käynnistys-skriptistä:

```
#!/bin/bash
python /usr/local/bin/obfsproxy \
--log-min-severity=debug \
  --data-dir=/tmp/scramblesuit-server \
scramblesuit \
  --password=32POWUWZDOLDWNBYLIIEIAIS3QE23BHO \
--dest 127.0.0.1:1194 \
server 0.0.0.0:80
```

Obfsproxy-asiakkaan käynnistys ei juurikaan eroa palvelimen käynnistyksestä. Asiakkaan päässä ei tietysti tarvitse konfiguroida portti-ohjausta välityspalvelimelta sovelluspalvelimelle. Tässä riittää, että konfiguroidaan välityspalvelin ja käytettävä lähdeporttinumero. Alla esimerkki Obfsproxy-asiakasohjelman käynnistyskomennosta käytettäessä obfs2-protokollaa:

```
obfsproxy --log-file=obfsproxy.log --log-min-severity=info obfs2 --
shared-secret=this_is_shared_secret socks 127.0.0.1:1050
```

Muiden protokollien (obfs3 ja scramblesuit) kanssa asiakkaan käynnistys tapahtuu samalla analogialla:

```
obfsproxy --log-file=obfsproxy.log --log-min-severity=debug obfs3
socks 127.0.0.1:1050
```

```
obfsproxy --log-min-severity=debug scramblesuit --password
32POWUWZDOLDWNBYLIIEIAIS3QE23BHO socks 127.0.0.1:1050
```

7.4.3 Fteproxy konfiguraatio

Fteproxy-ohjelma on toteutettu myös python-ohjelmointikielellä. Tämän vuoksi sen asennus tapahtuu samalla tavalla kuin obfsproxy-ohjelmiston. Asennus suoritetaan pythonin omalla pakettienhallinta ohjelmalla:

```
pip install fteproxy
```

Fteproxy-ohjelmistoa voidaan käyttää suoraan komentoriviltä sekä asiakkaan, että palvelimen päässä. Ohjelman käynnistäminen tapahtuu samalla periaatteella kuin obfsproxy-ohjelmiston käynnistyskin. Käynnistysparametreina annetaan sekä palvelin että asiakkaan päässä toimintamoodi, Fteproxy-palvelimen IP-osoite ja Fteproxy-palvelimen porttinumero. Lisäksi palvelimen päässä määritetään välityspalvelimen IP-osoite sekä käytettävä porttinumero. Alla esimerkit palvelimen ja asiakkaan käynnistyskomennoista liikenteen naamioimiseksi HTTP-liikenteeksi:

```
fteproxy --mode server --server_ip 192.168.50.13 --server_port 8080 --
proxy_ip 127.0.0.1 --proxy_port 8888

fteproxy --mode client --server_ip 192.168.50.13 --server_port 8080
```

Oletuksena hämäännytetty liikenne naamioidaan HTTP-liikenteeksi. Kuten todettu, Fteproxy tukee valmiina myös SSH- ja SMB-protokollia, joten liikenne voidaan naamioida myös näiksi, ilman omaa erillistä kehitystä. Muutettaessa oletusnaamiointia, palvelimen ja asiakkan käynnistyskomentoihin tulee lisätä lähtevä ja saapuva sovellusprotokolla:

```
--upstream-format XXX --downstream-format YYY
```

Fteproxy ei sisällä Obfsproxy-ohjelmiston tapaan sisäänrakennettua SOCKS -välityspalvelinta, joten se pitää asentaa sekä käynnistää erikseen. Helpoin tapa on käyttää tähän OpenSSH-palvelinta. Käyttämällä OpenSSH-palvelinta SOCKS-välityspalvelin voidaan luoda Fteproxy-palvelimella yksinkertaisesti esimerkiksi komennolla:

```
ssh -D 8888 username@localhost
```

7.5 RAT-ohjelmistot

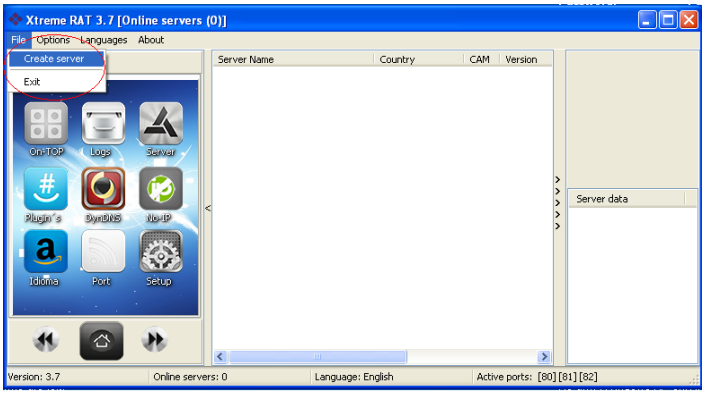
Haitallisen liikenteen osalta tässä työssä keskityttiin kahteen eri RAT-ohjelmistoon, joiden valinta perusteltiin tarkemmin luvussa 6. Nämä ohjelmistot ovat:

- XtremeRAT 3.7
- DarkComet 5.3.1.

Valintaan vaikutti ohjelmistojen yleisyyden lisäksi myös oleellisesti suoritettavan ohjelmakoodin löytyminen. Näiden ohjelmistojen osalta tavoitteena oli havaita mahdollinen väärä luokittelu. Toisin sanoen, luokitteleeko DPI-kirjastot ja IDPS-ohjelmisto liikenteen HTTP-liikenteeksi. Tarkoituksena oli myös selvittää Suricatan osalta haitallisen liikenteen tunnistamista. Kyseiset RAT-ohjelmistot eivät ole kovin uusia, joten sormenjälki tai allekirjoitus näistä pitäisi olla tunnettu.

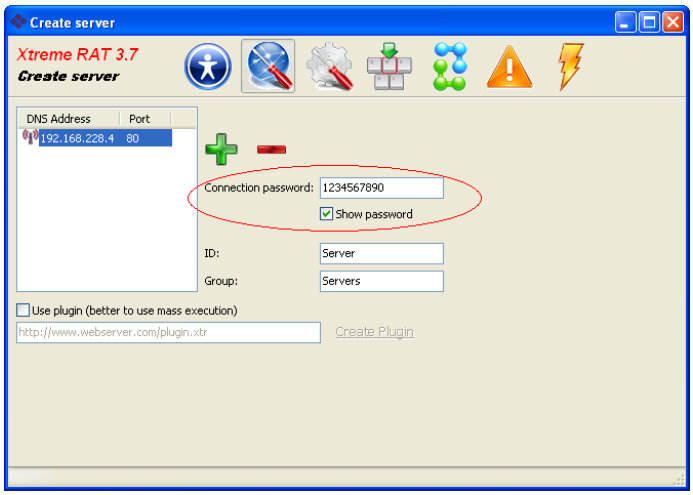
RAT-ohjelmistot, joista löytyi valmiit asennuspaketit, asennettiin Windows-työasemiin. XtremeRAT-ohjelmisto asennettiin Windows XP -työasemaan ja DarkComet Windows 7 -työasemaan. Tähän yksinkertaisena perusteena oli se, että valittu versio XtremeRAT-ohjelmistosta ei toiminut Windows 7 -käyttöjärjestelmässä. Molempien RAT-ohjelmistojen palvelinohjelmistot suoritettiin "uhrin" Windows XP -työasemassa.

RAT-ohjelmistojen asennus tapahtui hyödyntämällä valmista asennusohjelmaa. Asennusohjelman avulla asennetaan asiakasohjelmisto, jonka avulla laadittiin varsinainen palvelinohjelma. Palvelinohjelma on ohjelma, joka suoritetaan uhrin tietokoneella. Molempien RAT-ohjelmistojen asiakasohjelman asentaminen on hyvin saman kaltainen, joten tässä esitellään vain XtremeRAT-ohjelman asentaminen. Asentaminen suoritetaan kummankin RAT-ohjelmiston tapauksessa käyttämällä asennusvelhoa. Asennusvelhon avulla käyttäjä opastetaan vaihe vaiheelta luomaan palvelinohjelma. Kuvassa 23 on esimerkki XtremeRAT-ohjelmiston asennusvelhon käynnistämisestä.



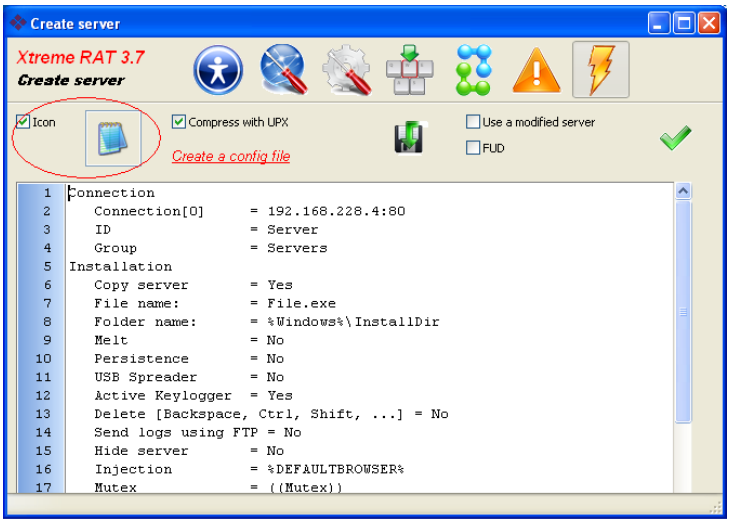
Kuva 23. *XtremeRAT asennusvelhon käynnistys*

Asennusvelhon toisessa vaiheessa määritetään hyökkääjän IP-osoite ja TCP-porttinumero, johon uhrin kone muodostaa yhteyden. Tässä vaiheessa määritetään myös luvussa 6 esitetty salasana, joka on oletuksena "1234567890". Yhteyden muodostamisvaiheessa uhrin kone tekee selväkielisen HTTP GET -pyynnön ./1234567890-tiedostoa varten. Pyyntö tehdään asennusvelhossa määritettyyn TCP-porttiin. Jos oletussalasana muutetaan, pelkästään merkkijonoa "HTTP GET ./1234567890" etsimällä ei tietenkään havaita mitään. Kuvassa 24 on esitetty IP-osoitteen, TCP-porttinumeron ja salasanan määrittely.



Kuva 24. *Hyökkääjän IP-osoitteen ja TCP-porttinumeron, sekä salasanan määrittäminen*

Kun asennusvelho on suoritettu loppuun, palvelimen asennuksesta saadaan yhteenveto. Yhteenvedossa on kuvattu palvelimelle valitut asetukset. Kuten luvussa 6 esitettiin, palvelinohjelma voidaan esimerkiksi injektoida oletusselaimeen. Tämän etuna on se, että verkkoselain on pääsääntöisesti sallittujen ohjelmien listalla asiakaspalomuureissa. Muita asetusvaihtoehtoja ovat mm. palvelinohjelman kätkeminen, lokien lähettäminen määritellylle FTP-palvelimelle, "USB-levittäjän" käyttöönotto, palvelinohjelman sulattaminen ajon jälkeen ja aktiivisen näppäinpainalluksien tallentimen määrittäminen. Kuvassa 25 on esitetty yhteenveto palvelinohjelmassa käytetyistä asetuksista.



Kuva 25. *XtremeRAT-palvelimen asetusten yhteenveto*

Molempien RAT-ohjelmistojen käyttäminen on tehty varsin helpoksi graafisten käyttöliittymien avulla. Ohjelmien käyttämiseen ei tarvita syvää tietotekniikan osaamista, vaan niiden suunnittelussa on selvästi otettu huomioon käytettävyyteen ja helppokäyttöisyyteen liittyviä näkökohtia.

7.6 DPI-kirjastojen ja Suricatan konfiguraatiot

Luvun 3 vertailun perusteella valitut DPI-kirjastot ja Suricata asennetiin omaan testausympäristöön, joka kuvattiin aiemmin. Ohjelmistot asennettiin Ubuntu-linux käyttöjärjestelmään, joka toimi kyseisen testiympäristön yhdyskäytävänä asiakas- ja palvelinverkkojen välillä. Kaikki liikenne, sekä obfuskoitu että selväkielinenkin, kulki tämän yhdyskäytävän kautta. DPI-kirjastot asennetiin kohdejärjestelmään lähdekoodista kääntäen ja Suricatan asennuksessa käytettiin Ubuntu-linuxin paketointiohjelmistoa, eli apt-ohjelmaa. Testattavat ohjelmistoversiot olivat:

- nDPI 1.5.2
- Libprotoident 2.0.7
- Suricata 2.0.9.

Tutkimuksessa käytetty liikenne generoitiin testiympäristössä ja tallennettiin pakettikaappaustiedostoiksi, eli pcap-tiedostoiksi Wireshark-ohjelmalla. Wireshark-ohjelma tallentaa oletuksena versiosta 1.8 alkaen kaapatun liikenteen pcapng-muotoon (engl. PCAP Next Generation Dump File Format). Pcap-tiedostoja syntyi useita, jolloin testaaminen DPI-kirjastoja ja IDPS-ohjelmaa vasten oli mielekästä toteuttaa massa-ajona.

7.6.1 nDPI asennus ja konfiguraatio

nDPI:n asennus lähdekoodista tapahtui Github-palvelun kautta. Github-palvelu on web-pohjainen versiohallintajärjestelmä, jonka kautta voidaan asentaa eri versioita halutusta ohjelmistosta. Git-hakemisto, jossa sijaitsee ohjelmiston lähdekoodit, voidaan "kloonata" komennolla:

```
git clone https://github.com/ntop/nDPI.git
```

Tämän jälkeen nDPI versio 1.5.2 asennettiin oletusasetuksilla kääntäen lähdekoodista:

```
cd nDPI
./configure –with-pic
make
```

nDPI-ohjelmaan sisältyy ndpiReader -testausohjelma, jolla voidaan testata kirjastoa. Testausohjelma tukee pcap-tiedostoja, jonka avulla pcap-tiedostot voidaan ajaa DPI-kirjaston läpi. Testiympäristössä luotujen pcap-tiedostojen uudelleen ajaminen nDPI-kirjaston läpi tapahtui hyödyntäen tätä ohjelmaa. Pcap-tiedosto voidaan ajaa em. työkalulla seuraavasti:

```
ndpiReader -i tiedosto.pcap (TAI)
ndpiReader -i tiedosto.pcap -d
```

Alempi esimerkki, jossa on mukana "-d" optio, on oleellinen tämän tutkimuksen kannalta. Tämän avulla otetaan poiskäytöstä nDPI:n protokolla-arvaustoiminto. Kun liikennevuo on tunnistamaton, nDPI suorittaa arvauksen, joka perustuu tunnettuun tai rekisteröityyn kuljetuskerroksen protokollaan ja porttinumeroon (Deri et al. 2014). IANA eli "The Internet Assigned Numbers Authority" on vastuussa kyseisen rekisterin (engl. Service Name and Transport Protocol Port Number Registry) ylläpidosta. nDPI:n tapauksessa tunnistamaton liikennevuo tarkoittaa sitä, että kaikkia protokolladekoodereita on kokeiltu liikennettä vastaan.

Edellä esitetyt komennot palauttavat luokittelun tuloksen komentoriville. Koska pcap-tiedostoja oli käytössä kymmeniä, niiden suorittamiseksi käytettiin yksinkertaista skriptiä, joka tulee nDPI-ohjelman mukana. Tulokset tallennettiin yksilöllisiin tekstitiedostoihin, jotka nimettiin ajetun pcap-tiedoston mukaan. Liitteessä H on esitetty kaikki tässä työssä hyödynnetyt massa-ajoskriptit. nDPI:n tulostiedostot sisältävät verkkoliikennetilastoja ajetusta pcap-tiedostosta ja tunnistetut sovellusprotokollat. Verkkoliikenteestä esitetään mm. liikennevoiden määrä, tilastot pakettien koosta, hylätty tavumäärä, IP-pakettien määrä ja nDPI:n liikenteen läpäisykyky. Liitteessä I on esitetty esimerkki nDPI:n tulostiedostosta.

7.6.2 Libprotoident asennus ja konfiguraatio

Libprotoident-ohjelman asennus vaatii toimiakseen libtrace-, libpacketdump-, libflowmanager- ja libwandevent-kirjastot. Libtrace-paketti tarjoaa kirjastot liikenteen seurantaan. Libtrace sisältää mm. libpacketdump-kirjaston, joka pystyy jäsentämään sekä näyttämään paketin sisällön ihmiselle luettavaan muotoon. Tulos on hyvin samankaltainen tcpdump-ohjelman tuottaman tuloksen kanssa. (WAND Network Research Group a.) Libflowmanager on kirjasto, joka yhdistää yksittäiset paketit liikennevuoksi ja ylläpitää taulukkoa aktiivisista liikennevoista, sekä raportoi erääntyneistä liikennevoista tietyn joutokäyntiajan jälkeen. Se myös tarjoaa ohjelmointirajapinnan (API) TCP-pakettien uudelleenjärjestämiseen, jos paketit ovat saapuneet väärässä järjestyksessä. Tyypillisesti paketit pyritään tallentamaan kronologisessa järjestyksessä, jolloin epäjärjestyksessä saapuvat paketit voivat aiheuttaa ongelmia. (WAND Network Research Group b.)

Libwandevent on kirjasto, joka tarjoaa ohjelmointirajapinnan tapahtumapohjaisiin ohjelmiin. Toisin sanoen kyseinen kirjasto tarjoaa työkalut tapahtumien käsittelyyn. Tapahtuma on tässä tapauksessa esimerkiksi paketin saapuminen tai usean paketin saapuminen millä tahansa ohjelman suoritushetkellä. Kehittäjän pitää rekisteröidä tapahtumat ja kehittää tapahtumien tunnistamista varten funktiot. Libwandevent-kirjasto tarjoaa em. funktioille ohjelmakoodin tapahtuman hallintaa varten ja tarkistustoiminnot onko tapahtuma esiintynyt. Kehittäjä voi näin ollen keskittyä itse tapahtumiin, eikä siihen onko tapahtuma esiintynyt. (WAND Network Research Group c.)

Libprotoident-ohjelman asentaminen ei ollut niin suoraviivainen kuin nDPI:n asennus. Johtuen edellä esitetyistä riippuvuuksista ennen varsinaista ohjelman asennusta em. riippuvuuksien asentaminen täytyi toteuttaa ensin. Riippuvuudet löytyivät seuraavista paketeista:

- libtrace3-dev
- libpacketdump3-dev
- libflowmanager-2.0.4
- libwandevent-2.0.0.

Riippuvuuksien asennuksen jälkeen Libprotoident-ohjelman asentaminen suoritettiin samalla tavalla kääntäen lähdekoodista kuin nDPI-ohjelman. Lähdekoodi "kloonattiin" Github-palvelun kautta. Liitteessä J on esitetty ohjelman asentamisessa käytetyt komennot siinä järjestyksessä, kuin niitä tarvittiin.

Kuten nDPI, myös Libprotoident sisältää työkalun DPI-kirjaston testaamiseen komentoriviltä. Testausohjelma on nimeltään lpi_protoident ja se tukee pcap-tiedostoja. Pcapng-tiedostoja ei tueta, joten tässä muodossa olevat tiedostot joudutaan konvertoimaan. Konvertointi voidaan suorittaa Wireshark-ohjelman mukana tulevalla editcap-ohjelmalla. Lpi_protoident ohjelmaa ajetaan komentoriviltä alla olevan esimerkin mukaan.

```
lpi_protoident tiedosto.pcap
```

Koska työkalua voidaan ajaa komentoriviltä, voidaan pcap-tiedostojen massa-ajoa varten hyödyntää samaa skriptiä (liite H) kuin nDPI:n tapauksessa. Lpi_protoident on lisätty polkuun, joten aikaisemmin esitettyä skriptiä joudutaan muuttamaan vain READER-muuttujan osalta. Lisäksi suoritettavat verkkoliikenteen kaappaustiedostot ovat pcap-muodossa. Alla on esitetty tarvittava muutos.

```
READER=lpi_protoident
PCAPS=`cd pcap; /bin/ls *.pcap`
```

Lpi_protoident-ohjelman tuottama tulos ei ole ihmiselle niin selkeässä esitysmuodossa kuin ndpiReader-ohjelman tuottama tulos. Lpi_protoident ei myöskään tuota perustilastoa verkkoliikenteestä nDPI:n tapaan. Tulos sisältää ainoastaan liikenteen luokittelun. Liitteessä K on esimerkki Lpi_protoident-ohjelman tuottamasta luokittelutuloksesta.

7.6.3 Suricata asennnus ja konfiguraatio

Suricata IDPS-ohjelmiston asennus toteutettiin Ubuntun pakettienhallintaohjelmalla. Asennusta varten Suricata-ohjelmiston säilö (engl. repository) tuli lisätä asennettavalle Ubuntu-linuxille. Asennus suoritettiin seuraavasti:

```
add-apt-repository ppa:oisf/suricata-stable
```

```
apt-get update
apt-get install suricata
```

Varsinainen Suricatan konfiguraatiotiedosto on Yaml-formaatissa. Konfigurointi tapahtuu muokkaamalla tätä tiedostoa, joka sijaitsee oletusasennuksessa hakemistossa "/etc/suricata/suricata.yaml". Yaml on eräs datan tallennusmuoto, joka on sekä ihmiselle, että tietokoneelle tehokkaassa esitysmuodossa (Ben-Kiki, Evans & döt Net 2009). Suricata tarjoaa komennon, jolla siltä voi tarkistaa konfiguraatiotiedostossa käyttöönotetut sovelluskerroksen protokollat. Tarkistus voidaan suorittaa seuraavasti:

```
/etc/suricata# suricata -c suricata.yaml --list-app-layer-protos
```

Oletuksena tuetut ja konfiguroidut sovelluskerroksen protokollat ovat: HTTP, FTP, SMTP, TLS, SSH, IMAP, MSN, SMB, DCERPC ja DNS. Suricata ilmoittaa sovelluskerroksen protokollien löydöksistä eve.json -tiedostoon, joka sijaitsee /var/log/suricata -hakemistossa.

Kuten Yaml, myös JavaScript Object Notation (JSON) on datan esitystapamuot. JSON perustuu JavaScript-ohjelmointikielen osajoukkoon, mutta se on silti ohjelmointikielestä riippumaton tiedostomuoto. (json.org.) Vaikka JSON-muotoinen data on sekä ihmiselle että tietokoneelle "selkeä" lukuista, on datan selkiyttämiseksi suositeltua käyttää jäsennystyökalua. Eräs tälläinen työkalu on jq. Jq-ohjelman avulla voidaan suodataa, pilkkoa ja muuntaa strukturoitua dataa.

Koska Suricata on IDPS-ohjelmisto, tulee tähän ladata uusimmat tunkeutumisen havaitsemissäännöt (engl. Intrusion Detection System rules, IDS rules). Säännöt ovat ns. sormenjälkiä tai allekirjoituksia (engl. signature) haitallisesta liikenteestä. Suricataan tarjolla olevia sääntökokoelmia on useita, joista osa on maksullisia. Yksi käytetyimipiä sääntökokoelmia on Emerging Threats -sääntökokoelma. Tämä kokoelma on vapaasti jaettu, joten se on ilmainen. Kyseiseen sääntökokoelmaan tulee noin viikon välein päivityksiä. Päivitykset on helpoin ladata Oinkmaster-ohjelman avulla, jonka konfiguraatiotiedostoon lisätään URL, josta päivitykset sääntökokoelmaan haetaan. Alla esimerkki sääntöjen päivityskomennosta:

```
oinkmaster -C /etc/oinkmaster.conf -o /etc/suricata/rules
```

Testattaessa esimerkiksi RAT-ohjelmistojen havaitsemista, on hyvä varmistua siitä, että käytössä on uusin sääntökokoelma. Suricataan voidaan laatia myös omia sääntöjä. Suricatan sääntöjä laaditaan samalla syntaksilla kuin Snort IDS -järjestelmässä. Edellä esitetty Emerging Threats -sääntökokoelma on itse asiassa tuotettu juurikin Snortille, mutta se toimii myös suoraan Suricatassa. Omat säännöt laaditaan local.rules -tiedostoon, joka sijaitsee "/etc/suricata/rules" -hakemistossa. Alla on esimerkki itse laadituista säännöistä XtremeRAT-ohjelman havaitsemiseen. Kyseiset säännöt suorittavat saman asian, eli pyrkivät havaitsemaan XtremeRAT-ohjelman "asennuskättelyn". Uhrin kone "ilmoittautuu" suorittamalla TCP-porttiin 80 " HTTP GET /1234567890.functions" -merkkijonon

hyökkääjän koneella. Kuten alla on esitetty, IDS-sääntö voidaan luoda joko suoraan heksadesimaalilukujen avulla, tai ASCII-merkistön avulla tekstinä. Alla olevissa säännöissä ei kuitenkaan otetaan huomioon käytettävää porttinumeroa, vaan ainoastaan viestin sisällössä oleva merkkijono. Säännöt ovat vain esimerkkinä, todellisuudessa ne eivät ole kovin hyviä, koska sisältö "/1234567890" on ohjelmiston oletussalasana, joka on vaihdettavissa hyökkääjän toimesta. Kuten todettu, osa XtremeRAT-ohjelman liikenteestä on selväkielistä, jolloin IDS-säännöt on melko suoraviivaisesti toteutettavissa. Yleisempi ja tätä kautta myös parempi tapa olisi etsiä "my version|3.*" -merkkijonoa tai asiakkaan kuittausviestiä "58 0d 0a". Salakirjoittettu liikenne on sääntöjen osalta selvästi haastavampaa, sillä salakirjoitus tulisi kyetä purkamaan ennen kuin hyötykuormasta etsitään tiettyjä merkkijonoja.

```
alert tcp any any -> any any (msg:"Possible RAT  exploit"; content:"|2F
31 32 33 34 35 36 37 38 39 30 2E 66 75 6E 63 74 69 6F 6E 73|";sid:1;
rev:1;)
#
alert tcp any any -> any any (msg:"Possible  Xtremerat exploit"; con
tent:"/1234567890.functions";sid:2;rev:1;)
```

Kuten edellä esitetyissä DPI-kirjastoissa, on myös Suricatassa oma toiminallisuus järjestelmän testaamiseen. Versiosta 1.4 alkaen Suricata kykenee kuuntelemaan Unix-socket -rajapintaa ja vastaanottamaan tätä kautta komentoja käyttäjältä. Suricatan asennuksen mukana tulee suricatasc-skripti. Sen avulla päästään vuorovaikuttamaan socket-rajapinnan kanssa sekä saavutetaan olemassa oleva käskykokoelma, jonka socket-rajapinta tarjoaa. Toisin sanoen Suricatasc-skripti on asiakasohjelma, jolla voidaan hyödyntää kyseistä socket-rajapintaa. Socket-rajapinnan kautta Suricata-ohjelmistolle voidaan ajaa suoraan pcap-tiedostoja ja kyseisestä tiedostosta tehdyt havainnot voidaan tallentaa omaan yksilölliseen hakemistoonsa. (Open Information Security Foundation 2014.) Edellä mainittua socket rajapintaa voidaan hyödyntää seuraavasti:

```
suricata -c suricata.yaml --unix-socket
suricatasc
```

Yllä esitettyjen komentojen avulla päästään vuorovaikutustilaan socket-rajapinnan kanssa sekä voidaan hyödyntää sen tarjoamaa käskykokoelmaa. Pcap-tiedosto voidaan ajaa Suricata-ohjelmassa socket-rajapinnan kautta alla olevan esimerkin mukaisesti.

```
pcap-file /pcap-tiedosto/DarkComet_liikenne.pcapng
/tuloshakemisto/suricata_tulos1
```

Kyseinen komento voidaan suorittaa massa-ajona, jolloin kaikki hakemistossa olevat pcap-tiedostot ajetaan Suricata-ohjelmistolle. Komennot voidaan ajaa peräkkäin socket-rajapinnan kautta, jolloin ne voidaan syöttää listana.

Toinen tapa ja myös helpompi tapa on hyödyntää suoraan Suricatan komentorivikomentoa käsitellä pcap-tiedostoja. Komentoriville syötettävien komentojen avulla on helpompi luoda skripti, eli lyhyt ohjelma, joka suorittaa massa-ajon pcap-tiedostoista. Tä-

hän voidaan hyödyntää jo aikaisemmin esitettyä skriptiä, jota käytettiin ndpiReader- ja Lpi_protoident-ohjelmien kanssa. Kyseinen skripti on esitetty liitteessä H. Alla esimerkki Suricatan komennosta, jolla voidaan suoraan lukea pcap-tiedosto ja tallentaa tulokset tiettyyn hakemistoon.

```
Suricata -c konfiguraatiotiedosto -r hakemisto/tiedosto.pcap -l \
/hakemisto2/ -k "ei-pakettien-tarkistussumma-tarkistusta"
```

Liitteessä H esitetyn skriptin avulla saadaan tulokset yksilöllisiin jokaista pcap-tiedostoa vastaaviin hakemistoihin. Riippuen Suricatan konfiguraatiosta, se luo oletusasetuksin hakemistoon useita lokitiedostoja. Näitä ovat eve.json -tiedoston lisäksi fast.log, http.log, stats.log , suricata-start.log sekä unified2.alert -tiedostot. Unified2.alert tiedostot sisältävät lokin binäärimuodossa hakemistossa, josta se voidaan jatkokäsitellä Barnyard-ohjelmalla. Tämä nopeutaa Suricatan toimintaa, koska se ei tarvitse huolehtia datan formatoinnista, eikä sen tietokantaan tallentamisesta, jonka Barnyard-ohjelma suorittaa. Barnyard voi esimerkiksi tallentaa unified2-tiedostot suoraan MySQL-tietokantaan, lähettää ne Squil-ohjelmalle tai suorittaa tallennuksen monella muulla ulostulovaihtoehdolla eri ohjelmille.

Koska Suricata ei ole liikenteenluokittelija, vaan IDPS-järjestelmä, se tuottaa monenlaista lokia. Verrattuna nDPI- ja Libprotoident-kirjastoihin, liikenteen luokittelun tulos ei ole niin suoraviivaisesti saatavilla Suricatasta kuin em. DPI-kirjastoista. DPI-kirjastot tuottavat melko selkeät tulostiedostot edellä esitettyjen testausohjelmien avulla. Kuten liitteistä I ja K voidaan havaita, on näidenkin selkeydessä selvästi eroa nDPI:n hyväksi. Suricatassa liikenteen luokittelu on vain yksi ominaisuus, joten liikenteen luokittelu lisätään yhtenä havaintona lokitietoon, joka kirjataan mm. eve.json -tiedostoon JSON-tiedostomuodossa. Luokittelun tulos pitää suodattaa tästä tiedostosta, jossa on paljon muutakin lokitietoa, kuin pelkästään sovellusprotokollan tunnistustieto. Jotta Suricatan liikenteen luokittelun tulos saadaan järkevästi näkyviin kymmenistä pcap-tiedostosta, laadittiin tähän tarkoitukseen oma skripti. Skripti suodattaa liikenteen luokitustiedon kaikesta Suricatan luomasta lokitiedosta. JSON-tiedoston lukemiseen käytettiin edellä esitettyä jq-ohjelmaa. Jq-ohjelman avulla JSON-tiedostosta voidaan hakea tiettyä merkkojonoa, joka tässä tapauksessa on "event_type". Suricata ilmoittaa sovelluskerroksen protokollan havaitsemisen omana tapahtumanaan. Tapahtuma ilmenee eve.json tiedostossa seuraavasti:

```
"event_type":"http","src_ip":"192.168.40.10","src_port":50995...
```

Jq-ohjelma palauttaa hakuavaimena käytetyn merkkijonon arvon, joka eve.json tiedostossa tarkoittaa yllä olevan esimerkin mukaisesti arvoa "http". Tarvittavan skriptin laadintaan hyödennytetään aikaisemmin esitettyä pcap-tiedostojen massa-ajoskriptiä tietyin muutoksin. Soveltamalla em. skriptiä saadaan aikaiseksi ohjelma, joka suodattaa sovelluskerroksen protokollat Suricatan lokista ja tallentaa ne omaan tiedostoon. Skripti on esitetty liitteessä H.

Skriptin suorittamisen jälkeen saadaa tulokseksi "protocols.txt" -tiedosto, jossa on listattuna Suricatan havaitsemat tapahtumat. Kuten edellä esitetty, tapahtumiin sisältyy tuettujen sovelluskerroksen protokollien luokittelu. Alla esimerkki suodatetusta tiedosta protocols.txt tiedostossa.

```
582181 "alert"
16 "http"
14 "fileinfo"
9 "dns"
1 "tls"
```

Tiedosta havaitaan tiettyjen tapahtumien määrä, sekä niiden tyyppi. Esimerkin pcap-tiedostosta on havaittu kolme tuettua sovelluskerroksen protokollaa: http, dns ja tls.

7.7 Testiliikenteen ja hämäännytetyn liikenteen generointi sekä tuloksien tuottaminen valittujen ohjelmistojen avulla

Testiliikennettä generoitiin useita gigatavuja. Kaiken kaikkiaan liikennettä kertyi 16,1 gigatavua. Tosin kaikkea kaapattua liikennettä ei hyödennetty tuloksissa, koska osaa liikenteestä ei suodatettu siten, että niissä olisi mukana vain tutkittava liikenne. Liikenne kaapattiin Wireshark-ohjelman avulla pcap-tiedostoiksi ja liikenne suodatettiin BSD Packet filter (BPF) -suodattimien avulla. BPF on unix-järjestelmiin kehitetty pakettisuodatin, jonka avulla voidaan seuloa tiettyjen suodattimien avulla liikennettä. Liikenteen kaappauksessa kopioidut paketit siirtyvät verkkoliitännän siirtokerroksen laiteajureilta suodattimille. Pakettien kopioinnin suorittaa oma ohjelmistomoduuli, jota kutsutaan verkkohaaroittimeksi (engl. network tap). Liikenteen seulominen toteutetaan käyttöjärjestelmän ydintilassa, jolloin suodatetut paketit eivät päädy käyttäjätilan monitorointiohjelmistoille lainkaan. Paketti voidaan pudottaa tällä tavoin mahdollisimman aikaisessa vaiheessa, jolloin käsittelystä tulee tehokasta. (McCanne & Jacobson 1993.)

BPF-suodattimien avulla laaditut pcap-tiedostot sisältävät vain haluttua liikennettä. Haluttu liikenne tarkoittaa tässä tapauksessa sitä, että ne sisältävät vain tietyn sovellusprotokollan tuottaman liikenteen. Haluttu liikenne todennettiin Wireshark-ohjelman avulla. Liikennettä generoitiin sekä selväkielisenä että obfuskoituna. Käytännössä tarkoituksena oli aluksi luoda selväkielistä HTTP-liikennettä, sekä salattua SSH-, Secure Copy Protocol (SCP) - ja OpenVPN-liikennettä ilman obfuskointia. Tämän tarkoituksena oli selvittää DPI-kirjastojen sekä Suricatan osalta, että toteuttaako ohjelmistot niiden dokumentaatiossa luvattuja ominaisuuksia. Tämän jälkeen HTTP-, SSH-, SCP- ja OpenVPN-liikennettä obfuskoitiin edellä esitettyjen ja valittujen hämäännyttämismenetelmien avulla. Tavoitteena oli havaita, muuttuuko luokittelun tulos obfuskoinnin avulla ja kuinka paljon DPI-kirjastot ja Suricata tuottavat vääriä positiivisia luokitteluja.

RAT-ohjelmistojen liikenne generoitiin suorittamalla uhrin tietokoneella palvelinohjelmisto ja hyödyntämällä asiakasohjelmiston komentoja uhrin tietokoneen etäkäyttöön

saamiseksi. Tällä tavoin saatiin aikaiseksi komentokanavaliikennettä, joka sisälsi yhteydenmuodostuskättelyn sekä varsinaisia komentoja, joiden avulla asiakas komentaa uhrin tietokonetta. Kuten tässä työssä on osoitettu, kyseisten ohjelmistojen kättelyt ovat tunnistettavissa, mutta salauksen tai pakkauksen johdosta niitä ei ole helppo havaita. Myöskään komentojen havaitseminen ei ole yksinkertaista salauksen vuoksi.

Kaikki liikenne, joka tässä työssä generoitiin, on esitelty tarkemmin liitteen F liikenteen kuvaustaulukossa. Taulukossa on yhteenveto kaikista pcap-tiedoistoista. Taulukon sarakkeissa kuvataan, kuinka pcap-tiedosto on tuotettu, mitä BPF-suodatinta on käytetty, kun liikennettä on kaapattu sekä tuotetun pcap-tiedoston koko.

Bittorrent-protokollan osalta oli vaikea luoda etukäteen BPF-suodatinta, koska käytössä on UDP- ja TCP-protokolla, sekä useita IP-osoitteita ja porttinumeroita. Näin ollen Bittorrent-protokollan liikennekaappauksista suodatettiin kaikki muu liikenne pois Wireshark-ohjelman omilla suodattimilla jälkikäteen. Wireshark ei oletuksena osaa dekoodata tässä työssä käytetyn Bittorrent Vuze -asiakkaan UDP-liikennettä, eikä UDP:n päällä toimivaa hajautettua tiivistetaulutoiminnallisuutta (engl. Distributed Hash Table, DHT). DHT-toiminallisuuden avulla bittorrent-asiakas voi etsiä ladattavia "tiedostopaloja" yhdensuuntaisen tiivistearvon avulla muilta vertaisilta. Jotta Wireshark-ohjelmassa saadaan muodostettua selkeä suodatin DHT-toiminallisuudelle, protokolla pitää erikseen merkitä dekoodattavaksi Vuze-DHT-protokollaksi. Bittorrentin ohella myös kaikki muut pcap-tiedostot läpikäytiin ja tarkistettiin Wireshark-ohjelmalla, ennen niiden ajamista liikenteen luokittelijoille. Pohjatotuus on näin ollen muodostettu BPF-suodattimen ja Wireshark-ohjelman avulla.

Pcap-tiedostot ajettiin massa-ajona liikenteen luokittelijoille liitteessä H ja aiemmin esitettyjen bash-skriptien avulla. DPI-kirjastojen osalta tulokset luokittelusta saatiin jokaista pcap-tiedoston nimeä vastaavaan tekstitiedostoon. Suricatan osalta tulokset tallennettiin pcap-tiedostonimeä vastaavaan hakemistoon. Suricatan liikenteen luokittelutulokset suodatettiin lokitiedoista omalla skriptillä, joka on myös esitetty liitteessä H. Liikenteen luokittelun tulokset ja analyysi hämäännytetystä liikenteestä esitetään tarkemmin seuraavassa luvussa.

8. LUOKITTELUN TULOKSET JA HÄMÄÄNNYTETYN LIIKENTEEN ANALYYSI

Tämän tutkimuksen pääasiallisena tarkoituksena oli selvittää eri hämäännyttämismenelmien havaitsemista avoimen lähdekoodin ohjelmistoilla. Hämäännyttämismenetelmiksi valittiin useita avoimen lähdekoodin ohjelmistoja, jotka on pääosin kehitetty Tor-yhteisölle.

Tämän lisäksi tutkittiin takaporttiohjelmistojen komentokanavien muodostusta ja näiden asiakas- ja palvelinohjelmistojen liikenteen hämäännyttämistä. Takaporttiohjelmistojen osalta kohderajauksena oli RAT-ohjelmistot, jotka ovat usein käytössä kohdistetuissa hyökkäyksissä. RAT-ohjelmistoista valittiin kaksi tunnettua ohjelmistoa, jotka ovat olleet mukana viime vuosina useissa kohdistetuissa hyökkäyksissä ja APT-kampanjoissa.

Käytännössä liikenteen luokittelua varten on olemassa useita avoimen lähdekoodin ohjelmistoja. Liikenteen luokittelua voidaan suorittaa pakettien syvätarkastuksen eli DPI:n avulla. Useat avoimen lähdekoodin DPI-ohjelmat ovat DPI-kirjastoja, joissa ei ole varsnaista käyttöliittymää. Kirjastojen avulla DPI-toiminallisuus voidaan integroida omiin sovelluksiin. Useissa kirjastoissa on kuitenkin mukana testaustyökalu, jonka avulla kirjaston luokittelukykyä voidaan tutkia. Liikenteen luokittelusta on tehty useita tutkimuksia, mutta hämäännyttämismenetelmien havaitsemisesta ei ole tiedossa kattavaa tutkimusta. Tässä työssä DPI-kirjastoiksi valittiin aikaisemmissa liikenteen luokittelututkimuksissa menestyneet ohjelmat. DPI-kirjastojen lisäksi tunkeutumisen havaitsemiseen ja estoon kehitetyt järjestelmät suorittavat pakettien syvätarkastusta. Näitä ei ole suoranaisesti kehitetty liikenteen luokitteluun ja tätä kautta sovellukerroksen protokollien tunnistamiseen, mutta osaan näistä on kuitenkin kirjoitettu ns. protokolladekoodereita, joten niillä voi suorittaa luokittelua ainakin auttavasti. Tämän vuoksi tutkimukseen otettiin valittujen DPI-kirjastojen lisäksi myös yksi avoimen lähdekoodin IDPS-järjestelmä.

Tässä luvussa tullaan esittämään valittujen DPI-kirjastojen ja IDPS-ohjelmiston hämäännytetyn liikenteen luokittelu- eli tunnistuskyky. Tuloksissa esitetään pakettikoon ja pakettien välisen saapumisajan perusteella tehty analyysi hämäännyttämismenetelmistä. Analyysin avulla tunnistetaan hämäännytetyn ja normaalin liikenteen tilastollisten piirteiden erot. Tarkoituksena oli selvittää voidaanko näiden avulla muodostaa liikennemallit hämäännyttämismenetelmille.

8.1 Liikenteen luokittelu

Hämäännytetyn liikenteen generointi, pcap-tiedostojen luonti ja menetelmät, kuinka liikenteen luokittelua selvitettiin, kuvattiin luvussa 7. Painotus luokittelun tutkimisessa oli hämäännyttämismenetelmissä ja haitallisessa hämäännytetyssä liikenteessä. Lähtökohtainen oletus olikin, että hyödyntämällä hämäännyttämismenetelmää ns. "oikeaa" liikennöivää sovellusta ei tunnisteta tai liikennettä tutkiva ohjelmisto luokittelee sovellusprotokollan väärin.

Hämäännytetyn liikenteen lisäksi tutkittavaksi mukaan otettiin normaali HTTP-liikenne, salakirjoitettu SSL/TLS-liikenne (HTTPS), salakirjoitettu SSH-liikenne ja salakirjoitettu OpenVPN-liikenne. OpenVPN-liikennettä ajettiin eri TCP- ja UDP-portteihin, jonka avulla selvitettiin kykenevätkö liikenteen luokittelua suorittavat ohjelmistot tunnistamaan salakirjoitettua liikennettä, jos sitä ei ajeta oletusporttiin. DPI-kirjastojen dokumentaatioissa painotetaan juurikin tätä ominaisuutta. Näiden kirjastojen suunnittelun lähtökohtana on ollut siirtokerroksen porttinumeroiden dynaamisuus.

8.1.1 Hämäännyttämätön liikenne

Taulukossa 3 on esitetty normaalin liikenteen (hämäännyttämättömän liikenteen) luokittelun tulokset. Punaisella merkityt luokitukset ovat vääriä tunnistuksia (engl. false positive), vihreällä merkityt ovat oikeita tunnistuksia ja mustalla on tuntematon luokittelu. Koska Suricata ei tue OpenVPN-sovellusprotokollaa, eikä se näin ollen kykene luokitteleemaa liikennettä, sen luokitus on merkitty myös mustalla. nDPI:n osalta selvitettiin luokittelu normaalisti "arvaus"-toiminnon (engl. protocol guess) ja ilman arvausta, eli "-d" option kanssa.

Taulukko 3. *"Normaalin" liikenteen luokittelu*

LIIKENNE / HÄMÄÄNNYTYSMENETELMÄ	LIIKENTEEN LUOKITTELIJAT JA NIIDEN TUOTTAMA LUOKITUS			
	nDPI	**nDPI -d**	**Libprotoident**	**Suricata**
HTTP-liikenne 1	HTTP*	HTTP*	HTTP	HTTP
HTTP-liikenne 2	HTTP	HTTP	HTTP	HTTP
HTTPS (SSL)	SSL	SSL	HTTPS	TLS
SSH + SCP (TCP 22)	SSH	SSH*	SSH	SSH
HTTP-liikenne 2 + OpenVPN, UDP 50010	Unknown	Unknown	Unknown_UDP	Ei tunnistusta
HTTP-liikenne 2 + OpenVPN, TCP 443	SSL	Unknown	OpenVPN	Ei tunnistusta
HTTP-liikenne 2 + OpenVPN, TCP 1194	OpenVPN	Unknown	OpenVPN	Ei tunnistusta
SSH + SCP + OpenVPN, TCP 1194	OpenVPN	Unknown	OpenVPN	Ei tunnistusta

Kuten taulukosta 3 havaitaan, kaikki tutkimukseen valitut ohjelmistot luokittelevat salakirjoittetun TLS/SSL ja SSH liikenteen oikein. Myös selväkielinen HTTP-liikenne luo-

kitellaan täysin oikein. Tähtimerkki tuloksen perässä tarkoittaa sitä, että liikenteenkaappaustiedostosta on havaittu hienojakoisemmin esim. palveluntuottajia, kuten google, youtube ja twitter tai oikean tunnistamisen lisäksi jotain muuta sovellusprotokollaa, mitä ei pitäisi olla kaappaustiedostossa.

OpenVPN-liikenteen osalta tulos on varsin mielenkiintoinen. Tutkimuksessa käytettyjen DPI-kirjastojen dokumentaation mukaan OpenVPN-protokollan pitäisi olla tunnistettavissa, eli ohjelmistojen tulisi tunnistaa tämä, vaikka sovelluskerroksen oletusporttinumero vaihdetaan toiseksi. Mikään ohjelmistoista ei tunnista OpenVPN-liikennettä, joka ajetaan UDP:n päällä porttiin 50010. Liikenne luokitellaan tuntemattomaksi. nDPI tunnistaa väärin tai luokittelee tuntemattomaksi liikenteen joka ajetaan TCP 443 -porttiin. Tämä johtunee TLS-autentikaatiosta, jota hyödynnetän OpenVPN-liikenteen kättelyssä.

Libprotoident luokittelee TCP 443 -porttiin ajetun liikenteen täysin oikein. nDPI luokittelee ilman "protokolla-arvausta" TCP 1194 -oletusporttiin ajetun OpenVPN-liikenteen tuntemattomaksi. Tämä on varsin erikoinen tulos, sillä nDPI:ssä on oma protokolladekooderi OpenVPN-liikennettä varten. Tässä tutkitaan mm. 1194 lähde- ja kohdeportit. "Arvaustoimintoa" käytetään vasta sen jälkeen, kun kaikki protokolladekooderit on käyty läpi, joten jostain syystä liikennevuo ei ole "osunut" sitä vastaavaan protokolladekooderiin.

8.1.2 Bittorrent-, MSE- ja fteproxy-liikenteen luokittelu

Taulukossa 4 esitetään normaalin bittorrent-liikenteen sekä fteproxy- ja MSE- menetelmillä hämäännytetyn liikenteen luokittelun tulokset. Bittorrent-liikenteen luokitusta testattiin HTTP-trackerin kautta haetuilla vertaisilla ja käyttämällä DHT-ominaisuutta tiedostopalojen etsimiseen. Fteproxyn osalta liikennettä naamioitiin HTTP- ja SSH-liikenteeksi. Taulukon 4 liikennesarakkeessa suluissa oleva teksti tarkoittaa sovellusprotokollaa joksi liikenne on naamioitu.

Alla esitettävän taulukon 4 ensimmäisellä kahdella rivillä esitetään tulokset naamioidusta HTTP-liikenteestä. Tässä HTTP-liikenne on naamioitu HTTP-liikenteeksi. Tämän tarkoituksena oli selvittää tuleeko naamioinnista "tuntematon-luokitus" (engl. unknown). Pyrkimyksenä oli myös selvittää, kuinka oikean HTTP-liikenteen pakettijakauma eroaa fteproxyn avulla hämäännytetystä HTTP-liikenteestä. Jos pakettijakauma on hyvin erilainen, on tämä selvä anomalia, jonka avulla voidaan päätellä, että liikenne ei ole HTTP-liikennettä. HTTP-liikenteen tulos, joka on hämäännytetty HTTP-liikenteeksi, on merkitty violetilla tekstillä.

Tuloksista voidaan havaita, kuinka nDPI luokittelee option "-d" kanssa kaiken hämäännytetyn liikenteen tuntematon kategoriaan. Tämä on hyvä tulos, sillä luokittelun kannalta "tuntematon" on parempi tulos kuin väärä luokittelu. Väärän luokittelun tuloksena

liikenne pääsee mahdollisesta kontrollista läpi, joka ei tietysti ole DPI-kirjastojen näkökulmasta tavoitteena.

Taulukko 4. *Fteproxy hämäännytyksen sekä bittorrent-liikenteen ja MSE-hämäännytyksen luokitus*

LIIKENNE / HÄMÄÄNNYTYSMENETELMÄ	LUOKITUS			
	nDPI	**nDPI -d**	**Libprotoident**	**Suricata**
HTTP curl / Fteproxy, TCP 8080 (HTTP)	HTTP Proxy	Unknown	HTTP	HTTP
HTTP-liikenne 2 / Fteproxy, TCP 80 (HTTP)	HTTP	Unknown	HTTP	HTTP
HTTP-liikenne 2 / Fteproxy TCP 8080 (SSH)	HTTP Proxy	Unknown	SSH	Ei tunnistusta
SSH-liikenne / Fteproxy, TCP 80 (HTTP)	HTTP	Unknown	HTTP	HTTP
SSH-liikenne / Fteproxy, TCP 8080 (HTTP)	HTTP	Unknown	HTTP	HTTP
Bittorrent	Bittorrent *	Bittorrent *	Bittorrent *	P2P BitTorrent *
Bittorrent DHT	Bittorrent *	Bittorrent *	Bittorrent_UDP *	P2P Vuze BT UDP*
Bittorrent DHT / MSE	Bittorrent *	Bittorrent *	Bittorrent_UDP *	P2P Vuze BT UDP*
Bittorrent / MSE	Bittorrent *	Bittorrent *	Bittorrent_UDP*	P2P Vuze BT UDP*

HTTP-liikenne, joka on naamioitu SSH-liikenteeksi tai SSH-liikenne, joka on naamioitu HTTP-liikenteeksi, luokitellaan libprotoident-kirjaston toimesta väärin. Tämä oli oikeastaan oletettavissa, koska fteproxy on juurikin suunniteltu säännöllisiä lausekkeita käyttäviä DPI-kirjastoja vastaan. Perusasetuksilla ajettaessa nDPI luokittelee HTTP-liikenteen, joka on naamioitu SSH-liikenteeksi, HTTP proxyksi. Luokittelu tulee todennäköisesti "arvaustoiminnon" perusteella, koska obfuskoitu liikenne ei osu protokolladekoodereihin, jolloin lopuksi "arvataan" kuljetuskerroksen protokollan ja porttinumeron perusteella, että liikennöivä sovellus on HTTP proxy. Periaatteessa nDPI:n luokittelu on oikein, eli alkuperäinen liikkenne on HTTP-liikennettä, mutta jos liikenne olisi ajettu johonkin muuhun tunnettuun porttiin, olisi tulos varmasti toinen. Tämän vuoksi luokittelun tulos on myös merkitty violetilla. Obfuskoidun SSH-liikenteen osalta nDPI:n tulos voidaan analysoida samalla tavalla. Liikenne ei osu protokolladekoodereihin, jolloin TCP 80 -portin perusteella luokitellaan liikenne HTTP-liikenteeksi.

Suricata ei tunnista väärin HTTP-liikennettä, joka on naamioitu SSH-liikenteeksi. Suricatassa on oma protokolladekooderi SSH-liikennettä varten, mutta obfuskoitu liikenne ei osu tähän dekooderiin, joten hämäännytys ei Suricatan osalta onnnistu. SSH-liikenne, joka on naamioitu HTTP-liikenteeksi osuu Suricatan HTTP-dekooderiin, koska liikenne tallentuu "http.log" -tiedostoon. Tämän perusteella tehdään siis väärä luokittelu. Suricataan olisi kuitenkin helppo luoda anomalia-sääntö tätä liikennettä varten, sillä käytettävää www-selainta (engl. user-agent) ei tunnisteta. Tästä alla esimerkki, joka on poimittu Suricatan http.log -tiedostosta.

```
<useragent unknown> [**] 192.168.40.13:38959 -> 192.168.50.13:80
```

Bittorrent-protokollan osalta tulos on yllättävä. Sekä selväkielinen, että hämäännytetty liikenne luokitellaan oikein kaikkien sensori-ohjelmistojen toimesta. Alkuoletuksena oli, että selväkielinen ja hämäännytetty bittorent-liikenne havaitaan selväkielisen HTTP GET -pyynnön avulla, joka suoritetaan tracker-palvelimelle. Tämän vuoksi wireshark-ohjelman avulla suodatettiin kaksi liikennekappausta siten, että ne sisältävät vain DHT:n avulla suoritetun palojen etsinnän. Toinen selväkielinen ja toinen näistä MSE:n avulla hämäännytetty. Kyseinen liikenne on pelkästään UDP-protokollan päällä ajettavaa bittorrent-liikennettä. Tämä ei kuitenkaan vaikuttanut luokittelun tulokseen, mikä on varsin mielenkiintoinen havainto. Bittorrent-liikenne on siis täysin tunnistettavissa tässä esitettyjen DPI-kirjastojen ja Suricatan toimesta. Suricatan osalta on hyvä huomioida se, että siinä ei ole omaa protokolladekooderia bittorrent-protokollalle, vaan tämän tunnistus suoritetaan sormenjälkisäännön avulla. Bittorrent-liikenteen tunnistus ilmoitetaan näin ollen hälytyslokissa, eli fast.log -tiedostossa.

8.1.3 Obfsproxyn avulla hämäännytetyn liikenteen luokittelu

Taulukossa 5 esitetetään Obfsproxy-ohjelmiston tukemien obfs2-, obfs3- ja scramblesuit obfuskointiprotokollien luokittelu.

Taulukko 5. *Obfs2, obfs3 ja scramblesuit luokittelu*

LIIKENNE / HÄMÄÄNNYTYSMENETELMÄ	LUOKITUS			
	nDPI	**nDPI -d**	**Libprotoident**	**Suricata**
OpenVPN (HTTP-2) / Obfsproxy obfs2, TCP 80	HTTP	Unknown	Unknown_TCP	Ei tunnistusta
OpenVPN (HTTP-2) / Obfsproxy obfs2 + esijaettuavain, TCP 80	HTTP	Unknown	Unknown_TCP	Ei tunnistusta
OpenVPN (HTTP-2) / Obfsproxy obfs3, TCP 80	HTTP	Unknown	Unknown_TCP	Ei tunnistusta
OpenVPN (HTTP-2) / Obfsproxy scramblesuit, TCP 80	HTTP	Unknown	Unknown_TCP	Ei tunnistusta
OpenVPN (SSH+SCP) / Obfsproxy obfs2, TCP 80	HTTP	Unknown	Unknown_TCP	Ei tunnistusta
OpenVPN (SSH+SCP) / Obfsproxy obfs2 + esijaettuavain, TCP 80	HTTP	Unknown	Unknown_TCP	Ei tunnistusta
OpenVPN (SSH+SCP) / Obfsproxy obfs3, TCP 80	HTTP	Unknown	Unknown_TCP	Ei tunnistusta
OpenVPN (SSH+SCP) / Obfsproxy scramblesuit, TCP 80	HTTP	Unknown	Unknown_TCP	Ei tunnistusta

Käytännössä ainoastaan nDPI suorittaa väärän luokituksen perusasetuksilla. Tämä ei itse asiassa ole yllättävää johtuen ohjelmiston suorittamasta arvaustoiminnosta. Tuloksien perusteella voidaan selvästi todeta, että em. obfuskointiprotokollien avulla hämäännytetty liikenne ei osu minkään sensorin protokolladekoodereihin, jonka perusteella kaikki luokittelevat obfuskoidun liikenteen tuntemattomaksi. Suricatan osalta voidaan todeta, että obfuskoitua liikennettä ei tunnisteta väärin, sillä Suricata ei luokittele liikennettä lainkaan, jos se ei osu sen tukemiin sovellusprokolliin. Obfuskoidusta liikenteestä jää jälki ainoastaan Suricatan tilasto-lokiin, josta havaitaan, että Suricata on tutkinut liikennettä.

Tulos on hyvä, koska sensorit eivät suorita väärää positiivista tunnistusta. Kun otetaan huomioon mahdollinen liikenteen obfuskointi, on väärä positiivinen tunnistus selvästi vaarallisempi, kuin tunnistamaton luokittelu.

8.1.4 Haitallisen liikenteen luokittelu

Viimeiseksi, taulukossa 6 on kuvattu hämäännytetyn haitallisen liikenteen luokittelu. Haitallinen liikenne rajattiin tässä tutkimuksessa APT-kampanjoissa yleisesti käytettyihin RAT-ohjelmistoihin. RAT-ohjelmistoiksi valittiin XtremeRAT 3.7 ja DarkComet 5.3. Rajaus ja valinta on kuvattu edellä tarkemmin luvussa 6.

Taulukko 6. *Hämäännytetyn haitallisen liikenteen luokittelu*

LIIKENNE / HÄMÄÄNNYTYSMENETELMÄ	LUOKITUS			
	nDPI	**nDPI -d**	**Libprotoident**	**Suricata**
DarkComet RAT 5.3.1	HTTP	Unknown	Unknown_TCP	Ei tunnistusta
XtremeRAT 3.7 (asennus + komennot)	HTTP	HTTP *	Unknown_TCP	HTTP
XtremeRAT 3.7 (komennot)	HTTP	Unknown	Unknown_TCP	Ei tunnistusta

Suricata ei tunnista haitallista liikennettä uusimmilla "emerging threats" -tunnisteilla. Tämä on erikoinen löydös, sillä kyseiset RAT-ohjelmistot ovat tunnettuja ja niistä pitäisi olla olemassa selvät allekirjoitukset verkkoliikenteen osalta. Esimerkiksi luvussa 6 esitettiin, kuinka osa XtremeRAT-ohjelmiston viesteistä on selväkielisiä. Kuitenkaan versiota 3.7 ei tunnisteta. DarkComet-ohjelmiston osalta tilanne on haastavampi, koska kaikki liikenne on salakirjoitettua. XtremeRAT-ohjelmiston asennuskättely havaitaan Suricatan HTTP-lokista, jonne tulee tiedot HTTP GET -pyynnöstä ja asiakkaan vastauksesta tähän pyyntöön. Tästä ei kuitenkaan muodosteta erillistä hälytystä. Suricata ei siis tulkitse liikennettä haitalliseksi. Alla esimerkkirivi Suricatan http.log -tiedostosta:

```
05/20/2015-14:56:41.332762 192.168.228.4 [**] /1234567890.functions
[**] Mozilla/4.0 (compatible; MSIE 8.0; Windows NT 5.1; Trident/4.0)
[**] 192.168.6.10:1113 -> 192.168.228.4:80
```

Perusasetuksilla nDPI luokittelee arvaustoiminnon avulla kaiken haitallisen liikenteen HTTP-liikenteeksi. Tämä johtuu TCP 80 -porttinumerosta. XtremeRAT 3.7 (asennus + komennot) -pakettikaappaustiedosto sisältää 162 liikennevuota. Tässä pakettikaappaustiedostossa on mukana ns. asennuskättely, jossa suoritetaan HTTP GET -pyyntö. Tästä tiedostosta 160 liikennevuota on arvattu HTTP-liikenteeksi. Ainoastaan kaksi liikennevuota, jotka ovat HTTP GET -pyynnöt "/1234567890.functions" -hakemistoon luokitellaan protokolladekooderin avulla. Kun nDPI ajetaan perusasetuksilla, tulostiedostoon tulee rivi, joka kertoo onko luokitus suoritettu arvaustoiminnon avulla. Tämä esitetään liitteessä I, jossa arvausten määrä näkyy liikennetilastojen alimmalla rivillä. Tähän saadaan vahvistus, kun nDPI-kirjastoa ajetaan "-d" option kanssa, joka sammuttaa arvaustoiminnon. Kyseisestä XtremeRAT pakettikaappaustiedostosta tunnistetaan protokolla-

dekooderin avulla kaksi liikennevuota ja loput liikennevoista on tunnistamattomia. Tämän vuoksi edellä esitettyyn taulukkoon 6, on merkitty luokittelun kohdalle asteriski, eli tähtimerkki. Kaappauksesta tunnistetaan siis varmasti HTTP ja muu liikenne on tunnistamatonta. DarkComet- ja XtremeRAT-liikenne (komennot) luokitellaan "-d" option kanssa tuntematon-kategoriaan. Libprotoident luokittelee kaiken haitallisen liikenteen tuntematon-kategoriaan.

8.1.5 Luokittelun yhteenveto

Yhteenvetona kaikista tuloksista voidaan todeta, että pääosin hämäänyttämätön liikenne, johon lasketaan taulukon 3 liikennekaappaukset ja taulukon 4 bittorrent-liikenne, luokitellaan oikein. Ainoastaan OpenVPN-liikenteen kanssa oli ongelmia. Bittorrentin osalta, MSE-hämäännyttämismenetelmällä ei ollut vaikutusta luokituksen tulokseen. Hämäännytyksestä huolimatta, bittorrent-liikenne tunnistettiin oikein. **Hämäännyttämättömän liikenteen** osalta selvästi parhaiten selvisi **libprotoident**-ohjelmisto, joka luokitteli lähes kaiken liikenteen oikein. Ainoastaan OpenVPN-liikenne UDP-portiin 50010 jäi luokittelematta täysin oikein.

Hämäännytetyn liikenteen osalta **nDPI ilman arvaustoimintoa** selviytyi luokittelusta parhaiten. Tämän asetuksen avulla nDPI luokittelee kaiken fteproxyn avulla obfuskoidun liikenteen tuntematon kategoriaan, kun libprotoident suorittaa vääriä positivia luokituksia. Obfsproxyn osalta DPI-kirjastojen luokittelussa ei ollut eroja. Kumpikaan kirjastoista ei suorittanut vääriä positiivisia luokituksia. Sama pätee myös pääosin haitalliseen liikenteeseen. nDPI suorittaa HTTP-luokituksen XtremeRAT-ohjelmiston asennuskättelystä, muu liikenne kyseisestä pcap-tiedostosta luokitellaan tuntematon-kategoriaan. Yhteenvetotaulukko liikenteen luokittelusta, jossa on esitetty kaikki edellä esitetyt luokittelun tulokset, esitetään liitteessä L.

Suricata selviää luokittelusta hyvin suhteessa sen protokolladekoodereihin, eli käytännössä kaikki sovellukset, joihin Suricatassa on tuki, luokitellaan oikein. Luvussa 2 esitettyyn kysymykseen, voidaanko IDPS-sääntöjä hyödyntää tehokkaasti hämäännytetyn liikenteen havainnointiin, ei edellä suoritettu luokittelun tulos suoraan vastaa. Esimerkiksi XtremeRAT-liikenne voidaan tunnistaa IDS-säännön avulla, koska siinä on selväkielisiä merkkijonoja. Osa viesteistä on kuitenkin pakattuja ja tämä aiheuttaa Suricatan hyötykuorman lukemiselle selvän haasteen. XtremeRAT hyödyntää viestien pakkaamiseen gzip-pakkausta. Pakattaessa deflate-algoritmilla viestejä, tuloksena syntyy satunnainen tavujono. Vaikka pakattaisiin sama merkkijono, joka pakkauskerralla syntyy eri tulos. Näin ollen pakatun viestin sisältöä ei voida hyödyntää IDS-säännössä suoraan. Suricatassa on tuki deflate-algoritmin purkamiseen, mutta tuki on ainoastaan HTTP-liikenteelle. HTTP-palvelimet pakkaavat usein lähettämänsä datan, jotta voidaan säästää verkon kaistaa. Suricatassa HTTP-liikenteen dekoodaamiseen käytetään libhtp-kirjastoa, joka siis sisältää tuen deflate-algoritmin purkamiseen. Ongelma tulee kuitenkin siitä, että Suricata ei dekoodaa XtremeRAT-liikennettä HTTP-liikenteeksi, jolloin libhtp-

kirjastoa ei myöskään käytetä. Teoriassa yksinkertainen tapa läpäistä IDS-järjestelmä olisi vain pakata komentoliikenne. Suricatassa on tuki lua-ohjelmointikielelle, jonka avulla voidaan myös kirjoittaa sääntöjä. Lua sisältää deflate-modulin, joten yksi mahdollisuus olisi kehittää oma sääntö luan avulla, joka tarkastaisi kaikesta liikenteestä mahdollisen pakkauksen.

Suricata esittää dokumentaatiossaan esimerkkisääntöjä anomalioiden havaitsemiseen verkkoliikenteestä. Useasti haittaohjelmien komentoliikenne on naamioitu HTTP-liikenteeksi ja kuten luvussa 6 esitettiin, myös haittaohjelman binääri kopioidaan selainprosessiin. Muutamat anomalian havaitsemiseen esittettävät sääntöesimerkit koskevatkin juuri HTTP-liikennettä. Tällaisia ovat esimerkiksi säännöt (Open Information Security Foundation 2015):

```
alert tcp any any -> any ![80,8080] (msg:"SURICATA HTTP but not tcp
port 80, 8080"; flow:to_server; app-layer-protocol:http; sid:2271001;
rev:1;)

alert tcp any any -> any 80 (msg:"SURICATA Port 80 but not HTTP";
flow:to_server; app-layer-protocol:!http; sid:2271002; rev:1;)
```

Näistä ensiksi esitetty koskee HTTP-liikennettä, joka ei hyödynnä TCP-porttia 80 tai porttia 8080. Jälkimmäinen sääntö koskee nimenomaan liikennettä, jota ajetaan TCP-porttiin 80, mutta sitä ei tunnisteta HTTP-liikenteeksi. Käytännössä tämän yksinkertaisen säännön avulla voidaan tunnistaa verkkoliikenteestä anomalia, mikäli hyödynnetään tässä esitettyjä haittaohjelmia tai obfuskointimenetelmiä ja käytettävänä porttina on TCP 80. Ongelmana säännössä on kuitenkin se, että se ei toimi, kuten sen kuuluisi. Todennäköisesti ongelma Suricatan ohjelmakoodissa, sillä sääntö ei toimi myöskään Suricatan uusimmassa versiossa 3.0, joka julkaistiin vuoden 2016 tammikuun lopussa. Kyseistä sääntöä vasten toteutettiin useita testejä, mutta mikään tässä generoiduista pcap-tiedostoista ei aiheuttanut hälytystä. Vika on todennäköisesti käytettävässä negaatiossa (!http), sillä hälytys luodaan, jos testattavana datana on HTTP-liikenne ja negaatio poistetaan. Toisin sanoen, Suricatan sovellukserroksen dekoodaus HTTP-liikenteelle toimii oikein. Anomalia-sääntö voitaisiin luoda myös pakettien pituuksien avulla, mutta tätä ei tässä työssä testattu.

8.2 Hämäännyttämismenetelmien liikenneanalyysi

Liikenteen luokittelua voidaan suorittaa myös tilastollisen analyysin avulla. Luvuissa 2 ja 3 on kuvattu, kuinka tilastollista analyysiä suoritetaan. Pääasiassa luokittelussa käytetään pakettikokojakaumaa ja pakettien välisten saapumisaikojen jakaumaa. Tilastollinen analyysi ei ole pakettien syvätarkastusta, koska siinä ei tarvitse lainkaan tutkia yksittäisten pakettien hyötykuormaa.

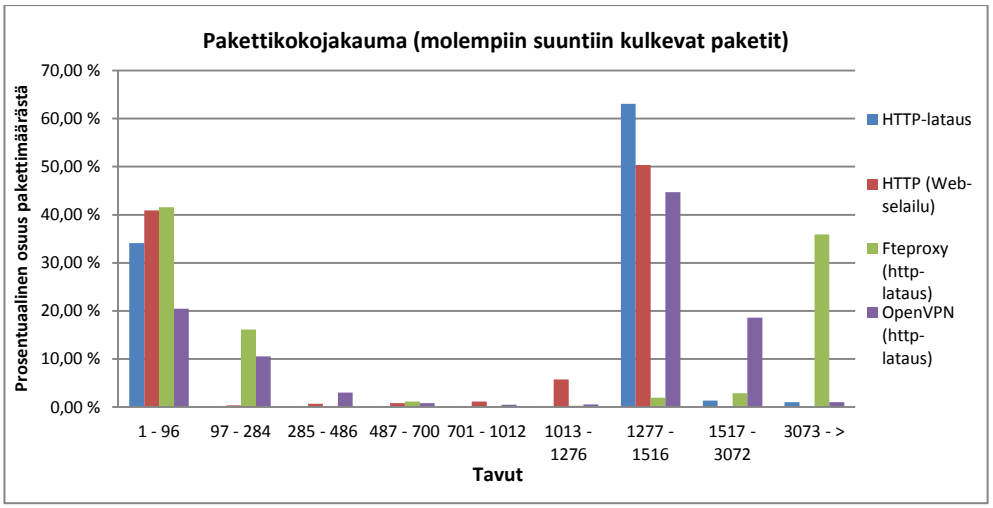
Tässä alaluvussa on tarkoituksena esittää eri hämäännyttämismenetelmien edellä esitetyt jakaumat. Jakaumat on muodostettu generoiduista pcap-tiedostoista. Jakaumia verrataan

alkuperäisiin, ilman hämäännytyskerrosta oleviin jakaumiin. Tämän perusteella voidaan päätellä mahdollinen eroavuus alkuperäisen liikenteen ja hämäännytetyn liikenteen välillä. Luokituksen määrittäminen pakettijakauman perusteella voi olla haastavaa, mutta anomalian havaitseminen ei niinkään. Toisin sanoen, jos hämäännytettyä liikennettä ajetaan TCP 80 -porttiin ja sen pakettikokojakauma poikkeaa merkittävästi oikean HTTP-liikenteen pakettikokojakaumasta, on tämä selvä anomalia. On hyvä myös huomioida, että tässä esitetyistä obfuskointimenetelmistä, ominaisuuksien perusteella ainoastaan scramblesuit-obfuskointiprotokolla hämäännyttää liikennevuon. Muut suorittavat vain hyötykuorman hämäännyttämisen.

Hämäännyttämismenetelmän tunnistamisesta alla esitettävien tilastollisten analyysien avulla ei voida tehdä kovin tarkkoja päätelmiä, koska pakettikoko ja pakettien välinen saapumisaika voi vaihdella riippuen sovelluksesta, jota pyritään hämäännyttämään. Pakettikokoanalyysi esitetään histogrammina ja kertymäfunktiona (CDF). Kertymäfunktion avulla nähdään tulosten käyttäytyminen helposti yhdellä silmäyksellä. Pakettien väliset saapumisajat esitetään kertymäfuntiona.

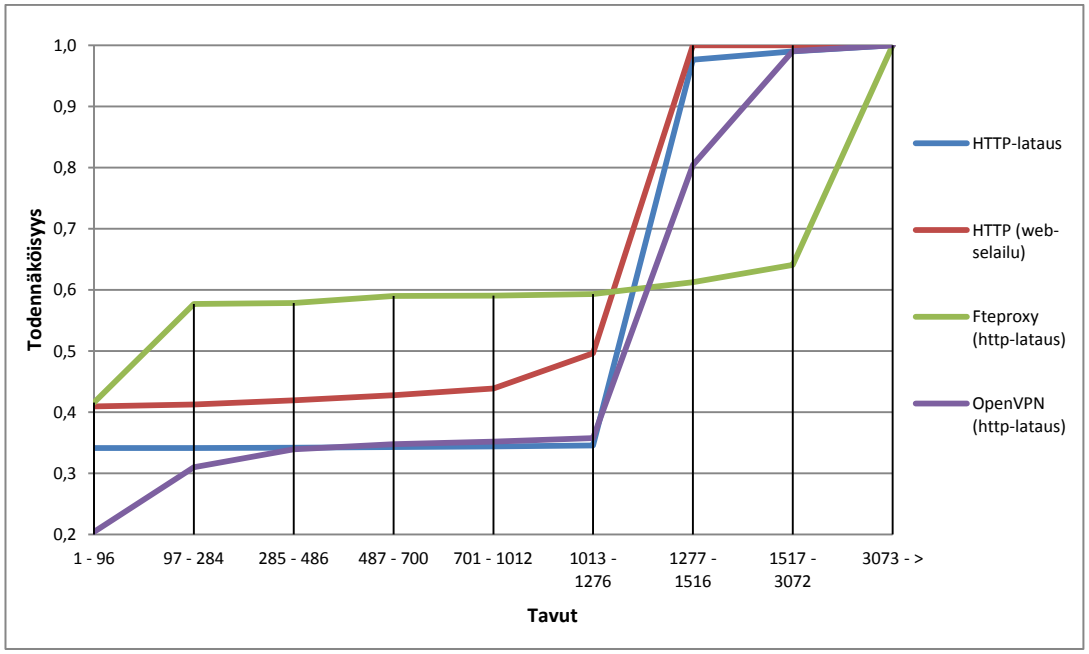
8.2.1 HTTP, Fteproxy- ja OpenVPN-liikenteen pakettikokojakaumat

Kuvissa 26 ja 27 on esitetty neljän tässä työssä generoidun pcap-tiedoston pakettikokojakaumat. Kuvassa 26 sinisellä olevat pylväät esittävät HTTP-liikennettä, jonka avulla on ladattu ftp.funet.fi -palvelusta "FreeBSD-10.1-RELEASE-amd64-bootonly.iso" levykuva. Levykuvan koko on 231Mt. Verrokiksi on lisätty web-selaimen avulla generoitu HTTP-liikenne (punainen pylväs), jossa selaimen avulla on vierailtu eri www-sivustoilla. FreeBSD-lataus on suoritettu OpenVPN-yhteyden avulla (violetti pylväs) sekä fteproxyn avulla hämäännytettynä HTTP-liikenteeksi (vihreä pylväs).



Kuva 26. *Pakettikokojakauma: HTTP, Fteproxy ja OpenVPN*

HTTP-latauksen pakettijakauma jakautuu pääosin kahteen eri tavuväliin. Suurin osa paketeista on 1-96 tavua tai 1277-1516 tavua. Samoin web-selaimen avulla www-sivustojen selaus jakaantuu myös kahteen eri tavuväliin. Tosin web-selailussa reilu 5 prosenttiyksikköä on paketteja väliltä 1013-1276 tavua. HTTP-lataus ja web-selailu ovat pakettikokojakaumaltaan hyvin samankaltaiset. Tämä selittyy sillä, että HTTP-palvelin lähettää 1500 tavun paketteja, joka on ethernet-kehyksen maksimi siirtoyksikkö (MTU-arvo), kun taas pienet paketit ovat TCP-protokollalle tyyppilisiä kuittauksia. HTTP:n osalta tulos noudattaa lähes Internet-liikenteen pakettikokojakaumaa, jossa noin 40% paketeista on alle 100 tavua ja noin 40% paketeista on väliltä 1300-1500 tavua (Center for Applied Internet Data Analysis 2010).



Kuva 27. *Pakettikokojakauma: HTTP-, Fteproxy- ja OpenVPN-liikenne kertymäfunktiona (CDF)*

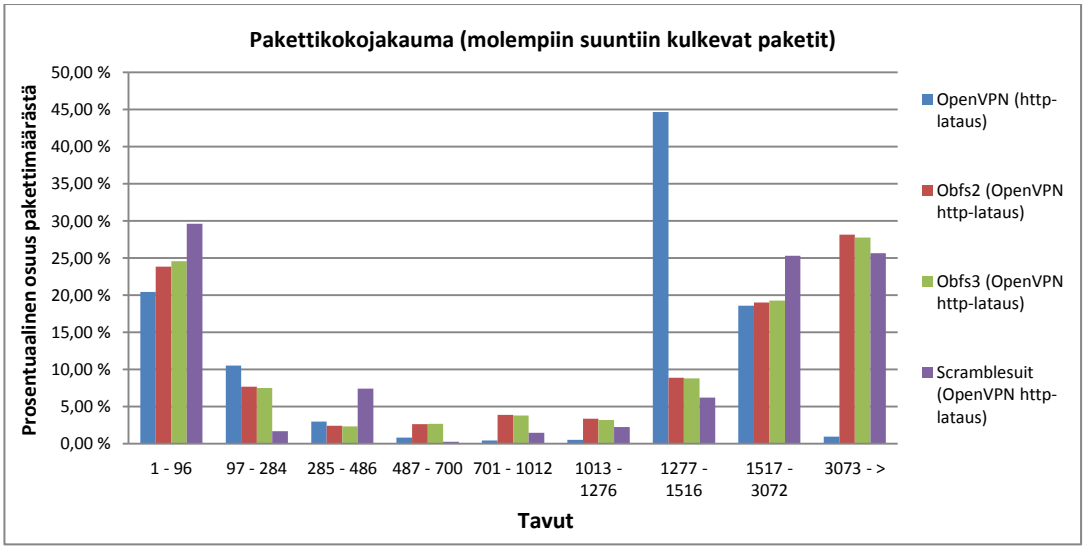
Fteproxyn ja OpenVPN-liikenteen pakettikokojakaumat poikkeavat selvästi normaalista HTTP-liikenteestä. Fteproxyn avulla hämäännytetyn liikenteen pakettijakauma sisältää pääosin pieniä paketteja, jotka osuvat välille 1-96 ja erittäin isoja paketteja, jotka ovat kooltaan 3073 tavua tai enemmän. Noin 15 prosenttia paketeista osuu välille 97-284 tavua. Fteproxyn pakettikokojakauma ei juurikaan sisällä paketteja väliltä 1277-1516. OpenVPN-liikenne sisältää paketteja väleiltä 1-96, 97-284, 1277-1516 ja 1517-3072. Edellä esitetyn pakettikokojakauman perusteella voidaan selvästi erottaa OpenVPN-, Fteproxy- ja HTTP-liikenne toisistaan.

Jakaumat ovat keskenään erilaisia, joka ei sinänsä ole mikään yllätys. Pakettikoko muuttuu normaalista HTTP-liikenteestä, koska molemmat fteproxy ja OpenVPN salakirjoittavat liikenteen. Toimenpide kasvattaa pakettien kokoa, koska salakirjoituksessa käytetään täytettä. Tämä ei kuitenkaan selitä kokonaan pakettikoon kasvua, koska pelkkä täyte ei voi selittää pakettikoon moninkertaistumista.

Fteproxyn osalta moninkertaistuminen johtunee obfuskointikerroksesta, joka lähetyspäässä syöttää TCP-ohjelmalle hämäännytettyä dataa. Luvussa 5.3 esitetty halkaisijamodulin datan puskurointi ja merkkijonojen yhdistäminen kasvattanee TCP-ohjelmalle tulevaa datan määrää. Fteproxy lisää myös paketteihin säännöllisten lausekkeiden avulla muodostetun "kielen" ja HTTP-otsikon, joka sisältää GET-metodin. Palvelin lisää viesteihin em. kielen ja "HTTP/1.1 200 OK" merkkijonon. Toisaalta on selvää, että nämä toimenpiteet myös pienentävät linjalle siirrettävää hyötykuorman määrää.

8.2.2 Obfsproxyn tukemien obfuskointiprotokollien pakettikokojakaumat

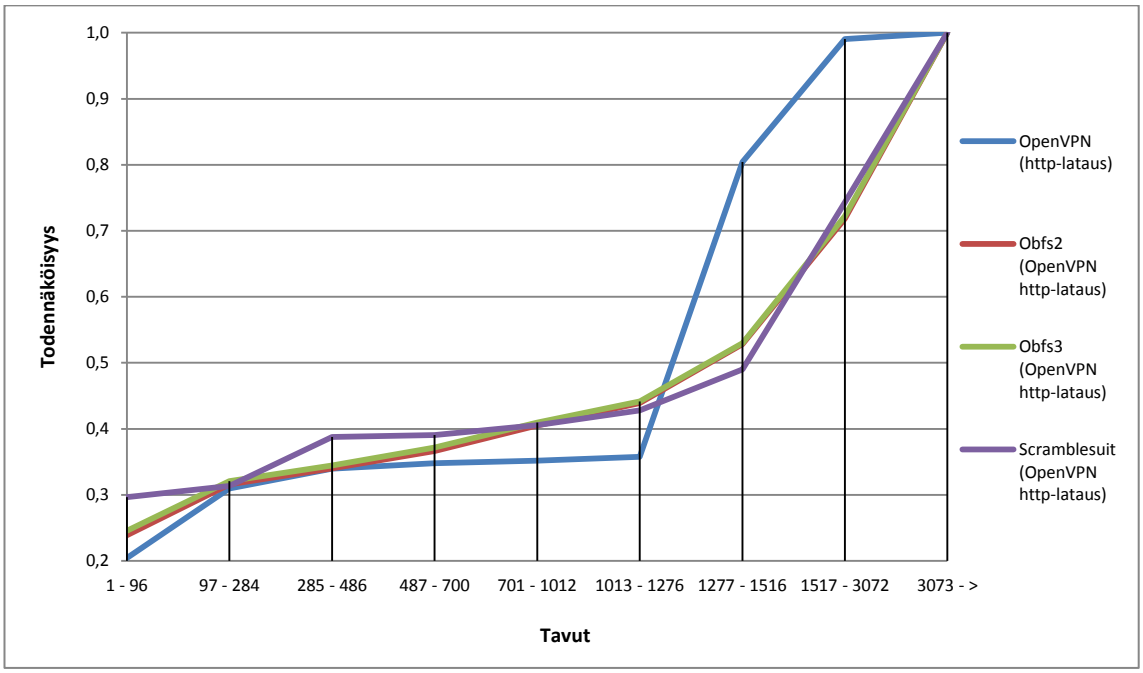
Kuvissa 28 ja 29 esitetään normaalin ja obfsproxyn avulla hämäännytetyn OpenVPN-liikenteen pakettikokojakaumat. Tässä esitetty OpenVPN-liikenne muodostettiin suorittamalla edellä esitetty HTTP-lataus OpenVPN-palvelimen kautta. OpenVPN-liikenne on jo lähtökohtaisesti pakettikokojakaumaltaan erilainen verrattuna HTTP-liikenteen jakaumaan. Tällä on myös vaikutus obfuskoidun liikenteen pakettikokojakaumaan, koska edellä esitetty HTTP-lataus on ensin salattu OpenVPN-sovelluksen avulla, jonka jälkeen tämä liikenne on hämäännytetty obfsproxy-ohjelmiston obfuskointiprotokollilla. Tämän vuoksi kuvissa 28 ja 29 verrataan normaalin ja hämäännytetyn OpenVPN-liikenteen pakettikokojakaumia, koska hämäännytyksen kohteena on sama sovellus.



Kuva 28. *Pakettikokojakauma: normaali sekä hämäännytetty OpenVPN*

Tässäkin normaali OpenVPN-liikenne erottuu selvästi hämäännytetystä liikenteestä. OpenVPN-liikenne jakaantuu pääosin kolmeen tavuväliin. Tavuvälillä 1277-1516 havaitaan selkeä piikki pakettien osalta, jota ei esiinny obfuskoidussa liikenteessä. OpenVPN-liikenteessä ei juurikaan esiinny paketteja, jotka ovat 3073 tavua tai enemmän. Kun obfuskoitua liikennettä verrataan normaaliin OpenVPN-liikenteeseen pakettikokojakauman osalta, voidaan havaita selvä anomalia.

Se, onko hämäännytetyn liikenteen pakettikokojakauma suhteessa samanlainen riippumatta hämäännytettävästä sovelluksesta, ei selviä tästä. Teoriassa suhde pitäisi olla samankaltainen, ainakin obfs2- ja obfs3-protokollan osalta, koska nämä eivät hämäännytä liikennevuota. Kuvan 28 kertymäfunktio osoittaa, kuinka juuri edellä mainittujen obfuskointiprotokollien pakettikokojakauma on lähes identtinen, kun obfuskoitavana sovelluksena on OpenVPN. Erot obfs2- ja obfs3-protokollan osalta ovat jokaisella tavuvälillä alle yksi prosenttiyksikköä.



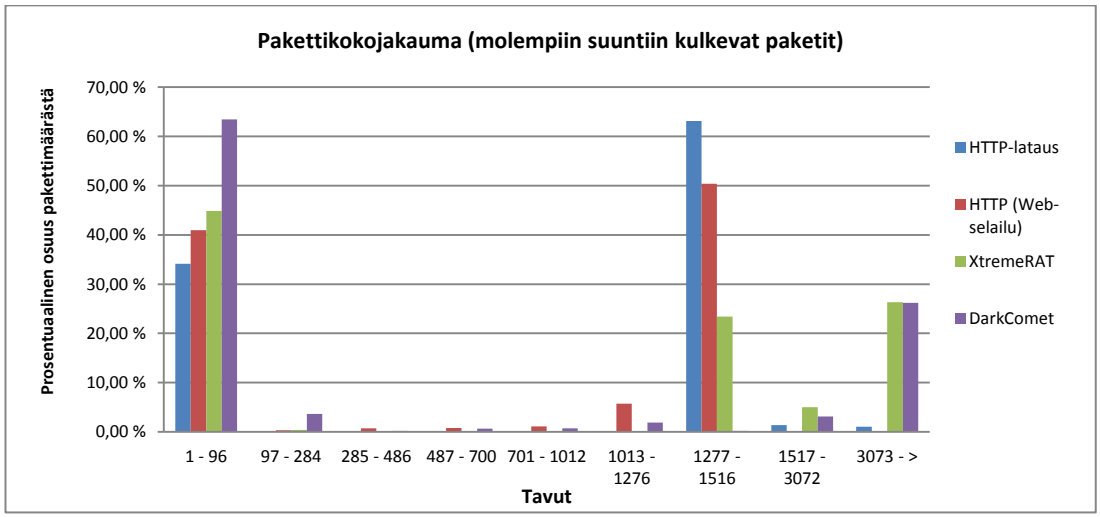
Kuva 29. *Pakettikokojakauma: normaali sekä hämäännytetty OpenVPN-liikenne kertymäfunktiona*

Spesifikaation mukaan ainoastaan scramblesuit-obfuskointiprotokolla hämäännyttää liikennevuon pakettikoon ja pakettien välisen saapumisajan suhteen. Kuvasta 28 kuitenkin havaitaan, että kaikkien obfuskointiprotokollien pakettikokojakauma on hyvin samankaltainen. Eroja toki on, esimerkiksi väleillä 1-96, 285-486 ja 1517-3072 on scramblesuit liikenteessä noin 5 prosenttiyksikköä enemmän paketteja, kuin muilla obfuskointiprotokollilla. Toki ero scramblesuit-protokollan ja obfs-protokollien välillä on selkeämpi, kuin obfs-protokollien välillä keskenään. Tulos on kuitenkin erilainen, mitä Tor esittää omalla wiki-sivustollaan (Tor Bug Tracker & Wiki 2015b). Tässä scramblesuit-protokollan pakettikokojakauman erot ovat selvempiä verrattuna obfs3-protokollaan.

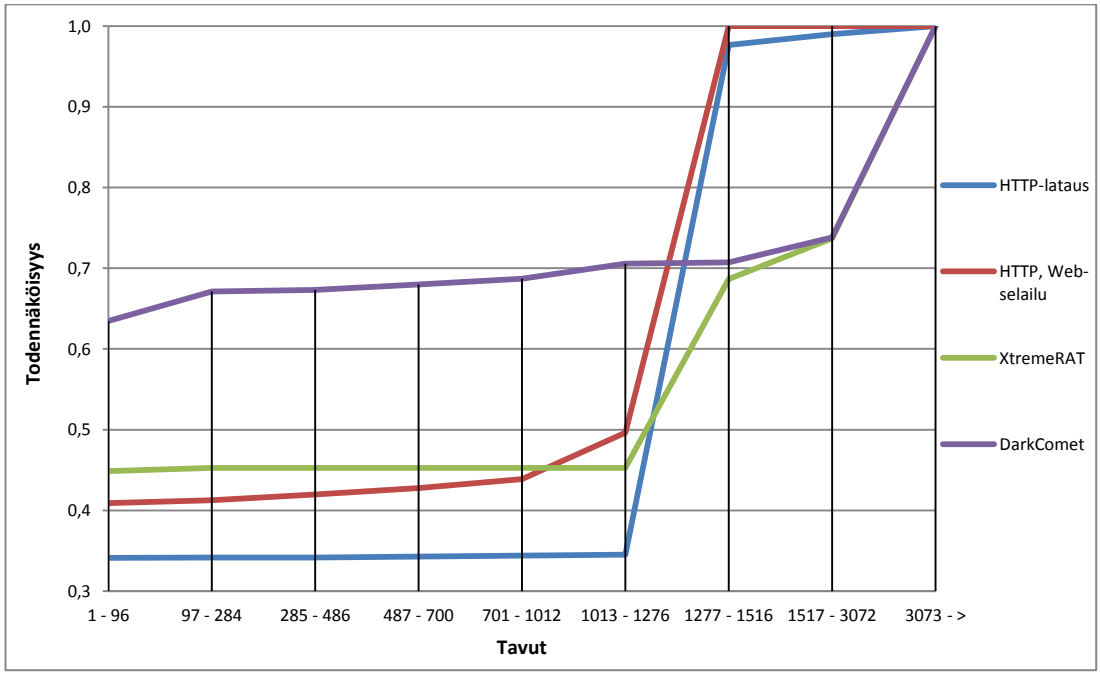
8.2.3 Haitallisen liikenteen pakettikokojakaumat

Kuvissa 30 ja 31 on esitetty normaalin HTTP-liikenteen ja haitallisen liikenteen pakettikokojakaumat. Kuvasta 30 havaitaan, kuinka haitallisen liikenteen pakettikokojakaumat ovat erilaisia verrattuna normaaliin HTTP-liikenteeseen. DarkComet-ohjelmiston liikenteestä lähes 60% sisältää pieniä paketteja ja noin 30% liikenteestä on suuria yli 1517

tavua sisältäviä paketteja. DarkComet-liikenne ei juurikaan sisällä paketteja väliltä 1277-1516, kun taas XtremeRAT-liikenne sisältää niitä yli 20%. Kuvasta 31 havaitaan, kuinka XtremeRAT-liikenne muistuttaa enemmän normaalia HTTP-liikennettä (web-selain), kuin DarkComet. Tämä sisältää pieniä paketteja ja paketteja väliltä 1277-1516. Selvä poikkeama verrattuna HTTP-liikenteeseen on kuitenkin siinä, että suuria yli 1517 tavua sisältäviä paketteja on noin 30% liikenteestä. Yli 3073 tavua sisältäviä paketteja on suhteessa lähes saman verran molempien RAT-ohjelmistojen liikenteessä.



Kuva 30. *Normaalin HTTP-liikenteen ja haitallisen liikenteen pakettikokojakaumat*



Kuva 31. *Normaalin HTTP-liikenteen ja haitallisen liikenteen pakettikokojakauma kerty-mäfunktiona (CDF)*

Edellä esitetyn perusteella voidaan todeta, että jos haitallista liikennettä ajetaan standardiin HTTP-porttiin (TCP 80), voidaan liikenne erottaa pakettikokojakauman perusteella selvästi normaalista HTTP-liikenteestä. Myös haitallinen liikenne on keskenään pakettikoon suhteen erilaista. XtremeRAT-liikenteestä viidesosa sisältää paketteja väliltä 1277-1516, kun DarkComet ei juurikaan sisällä näitä paketteja.

Tor-verkolle kehitettyjen obfuskointimenetelmien avulla hämäännytetystä liikenteestä on haastava muodostaa yleistä profiilia pakettijakauman perusteella, koska se saattaa olla hyvinkin erilainen riippuen obfuskoitavasta sovelluksesta. Haitallisen liikenteen osalta jakaumaan ei pitäisi tulla muutoksia, koska siinä obfuskoidaan aina samaa sovellusta, joka tässä tapauksessa on RAT-ohjelmisto. Tästä johtuen, teoriassa haitallinen liikenne voitaisiin profiloida pakettikokojakauman perusteella.

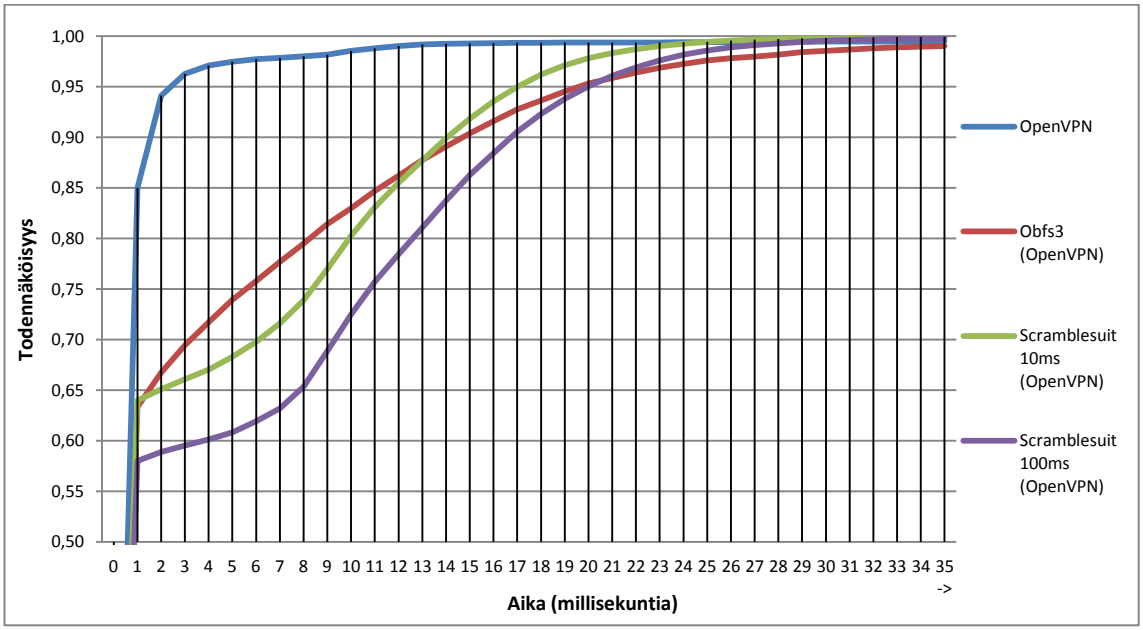
8.2.4 Analyysi pakettien välisen saapumisajan perusteella

Kuten aikaisemmin on esitetty, liikenteen profilointia voidaan suorittaa pakettien välisen saapumisajan perusteella (engl. packet interarrival time, IAT). IAT:n perusteella voidaan muodostaa jakauma, jonka avulla voidaan luokitella sovelluksen liikenne. Scramblesuit-protokolla on ainoa tässä työssä esitetty obfuskointiprotokolla, joka spesifikaation mukaan hämäännyttää liikennevuon pakettien pituuden ja IAT:n suhteen.

Obfsproxy-ohjelmistoon sisältyvä scramblesuit-protokolla ei obfuskoi oletuksena pakettien välistä saapumisaikaa. IAT:n obfuskointi hidastaa merkittävästi liikenteen läpäisikykyä, jonka vuoksi se on poiskytketty. Toimintoa ei voi myöskään käynnistää komentoriviargumenttien avulla. Jotta pakettien välisten saapumisaikojen obfuskointi saadaan käyttöön, joudutaan tutustumaan scramblesuit-koodiin. Obfsproxyn sisältämät obfuskointiprotokollat on kirjoitettu python-kielellä. Scramblesuit hyödyntää "const.py" nimistä tiedostoa, jonka avulla voidaan muuttaa tiettyjä protokollan parametreja. Yksi näistä parametreista on IAT:n obuskointi, joka otetaan käyttöön muuttamalla totuusarvoa. Luvussa 5 esitettiin malli, kuinka scramblesuit-protokolla hyödyntää pakettien välisen saapumisajan obfuskointia. Protokollan spesifikaation mukaan paketteja viivästytetään ennen siirtolinjalle siirtämistä. Viivästyttäminen tapahtuu muodostetun satunnaisjakauman mukaisesti siten, että aika vaihtelee välillä 0-100ms. Kuitenkin varsinaisessa toteutuksessa maksimi viivästysaika on asetettu kymmeneen millisekuntiin. Tätä arvoa voidaan kuitenkin muokata const.py-tiedostossa. Jotta tässä tutkittavaan jakaumaan saatiin eroja, scramblesuitin avulla obfuskoitua liikennettä generoitiin kahdella eri maksimi viivästysarvolla. Nämä olivat 10ms ja 100ms.

Scramblesuit-protokollan avulla hämäännytettiin OpenVPN-liikennettä. Tämän vuoksi tässä tutkittiin normaalin sekä hämäännytetyn OpenVPN-liikenteen pakettien välisiä saapumisaikoja. Hämäännytetystä liikenteestä valittiin scramblesuit-protokollan lisäksi obfs3-protokolla, joka ei hämäännytä liikennevuota. Obfs3-protokollasta saatiin näin ollen hyvä verrokki, jonka pakettien välistä saapumisaikajakaumaa voidaan verrata al-

kuperäiseen OpenVPN-liikenteeseen ja scramblesuit-protokollan avulla hämäännytettyyn liikenteeseen. Alla, kuvassa 32 esitetään pakettien välinen saapumisaikajakauma asiakkaalta palvelimelle kertymäfunktiona (CDF).



Kuva 32. *Pakettien välinen saapumisaika (asiakkaalta palvelimelle)*

Analysoitaessa tulosta on hyvä huomioda testiympäristö, jossa pakettikaappaukset on generoitu. Obfsproxy-asiakas ja -palvelin ovat kumpikin virtuaaliympäristössä, joten saapumisajat ovat hyvin lyhyitä. Kun scramblesuit-protokollaa verrataan obfs3-protokollaan, joka ei obfuskoi liikennevuota, on ero selvästi havaittavissa. Jakaumat poikkeavat selvästi toisistaan jo kahden millisekunnin jälkeen. Esimerkiksi noin 75% obfs3 pakettien välisistä saapumisajoista ajoittuu 6 millisekunnin sisälle, kun taas 10ms maksimi viivästyksellä noin 70% scramblesuit pakettien välisestä saapumisajasta ajoittuu tähän aikaikkunaan. Kuvasta 31 havaitaan myös, että scramblesuit-liikenteen (10ms ja 100ms) jakaumat ovat keskenään profiililtaan samankaltaiset.

Normaali OpenVPN-liikenne erottuu IAT-jakauman perusteella hämäännytetystä liikenteestä selvästi. OpenVPN-liikenteen osalta lähes 95% saapumisajoista on kaksi millisekuntia tai vähemmän. Kuvan 31 esittämästä jakaumasta voidaan todeta, että siinä esitetyillä soveluksilla on erilaiset IAT-jakaumat. Jakaumien perusteella voidaan erottaa normaali OpenVPN-liikenne obfuskoidusta liikenteestä sekä obfuskoitu liikenne keskenään toisistaan. Tämän tuloksen perusteella ei voida kuitenkaan yleistää mitään. Pakettien välisten saapumisaikojen jakauma voi riippua obfuskoitavasta sovelluksesta. Tämän vuoksi ei voida esimerkiksi todeta, että obfuskointiprotokollalla olisi aina samankaltainen IAT-jakauma riippumatta obfuskoitavasta sovelluksesta. Kun obfuskoitavana sovelluksena on OpenVPN, niin scramblesuit-protokolla näyttäisi kuitenkin synnyttävän samankaltaisen profiilin omaavan IAT-jakauman, oli maksimi viivästysaika sitten 10ms tai 100ms. Toistettaessa scramblesuitin avulla obfuskoitua liikennettä, IAT-jakauma ei

näyttäisi muuttuvan. Teoriassa näin ei pitäisi olla, koska spesifikaation mukaan scramblesuit luo satunnaisjakauman, jonka perusteella pakettien välistä saapumisaikaa obfuskoidaan. Näin ollen scramblesuit-liikenteen pakettien IAT-jakauma pitäisi olla joka kerta erilainen.

8.3 Protokolladekoodereiden laatiminen esitetyn kirjallisuuden ja tulosten perusteella

Tuloksien ja esitetyn kirjallisuuden perusteella voidaan toteuttaa obfs2-protokollalle ja XtremeRAT-liikenteelle protokolladekooderit, joita voidaan hyödyntää passiivisessa sensorissa. Nämä voidaan toteuttaa pelkän hyötykuorman tarkastelun avulla. Teoriassa "sormenjälki" XtremeRAT-ohjelmistosta voidaan toteuttaa myös tässä esitetyn tilastollisen analyysin avulla.

Obfs2-protokollan tunnistamiseen käytettävän protokolladekooderin algoritmi on seuraavanlainen:

- Tarkista 16 tavua ensimmäisen paketin hyötykuormasta (siemenarvo).
- Laske tiivistesumma funktiosta sha256("Initiator obfuscation padding" + siemenarvo + "Initiator obfuscation padding").
- Tiivistesumman 16 ensimmäistä tavua on avain.
- Käytä avainta paketin seuraaviin tavuhin ja jos seuraavat neljä tavua on taikaarvo (engl. magic value) 2bf5ca7e, on liikenne obfs2-protokollan tuottamaa.

XtremeRAT-liikenteen luokittelu voitaisiin toteuttaa seuraavan algoritmin avulla:

- Etsi paketin hyötykuormasta merkkijonoa "my version|3.*".
- Sisältääkö vastaus heksa-arvon "58 0d 0a"?
- Jos ensimmäiset kohdat täsmäävät, suorita deflate-algoritmin purku seuraaville paketeille ja etsi merkkijonoa "maininfo" ja "Server".

DarkComet-liikenteelle on selvästi vaikeampaa toteuttaa protokolladekooderi tai luokitus, koska liikenne on heti ensimmäisestä tavusta alkaen salakirjoitettu. Hyötykuorman tarkastelun avulla voidaan ainoastaan muodostaa luokitus oletusavaimia hyödyntäen. Hyötykuorman avulla ei voida myöskään luokitella obfs3- ja scramblesuit-protokollaa, koska näiden kättelyssä hyödynnetään tasajakauma-D-H:a. Fteproxyn luokittelu on myös hyötykuorman avulla haasteellista, koska merkkijonot hyötykuormassa ovat satunnaisia. Ainoaksi mielekkääksi tavaksi toteuttaa luokitus näille sovelluksille on liikennevuon analyysi.

XtremeRAT, DarkComet, obfs2, obfs3 ja fteproxy eivät hämäännytä liikennevuota. Kun verrataan pakettikokojakauman perusteella tässä esitettyä aineistoa keskenään, voidaan jakaumien perusteella erottaa sovellukset toisistaan. Toisaalta fteproxyn ja obfsproxyn

tukemien obfuskointiprotokollien pakettikokojakaumat eivät ole keskenään verrattavissa, koska hämäännytettävät sovellukset eivät ole samat. Obfsproxyn kanssa käytettiin OpenVPN-sovellusta, jonka avulla tunneloitiin HTTP-liikennettä.

Fteproxyn pakettikokojakauma on hyvin erilainen verrattuna normaalin HTTP-liikenteen jakaumaan. Fteproxyn naamioidessa liikennettä oletusasetuksilla HTTP-liikenteeksi, se tuottaa satunnaisen merkkijonon GET-pyyntöön. Yksi mahdollinen tekijä tunnistamiseen voisi olla myös entropian laskeminen merkkijonoille. Usein pyydettävät linkit kuitenkin sisältävät jonkin kielen. Monille kielille on tyypillistä, että ne sisältävät tiettyjä merkkejä, kuten esimerkiksi vokaaleja paljon. Tällöin entropia on selvästi matalampi kuin satunnaisissa merkkijonoissa. Lisäksi satunnaisesti tuotettujen merkkijonojen entropian vaihtelu ei ole suurta, koska aina generoidaan entropialtaan korkea merkkijono.

Työssä toteutettujen pakettikokojakaumien perusteella voidaan tunnistaa vain tietty liikenne. Toisin sanoen, esimerkiksi fteproxyn avulla hämäännytetty HTTP-liikenne voidaan tunnistaa. Toisaalta SSH-liikenne, joka on hämäännytetty fteproxyn avulla HTTP-liikenteeksi, omaa erilaisen pakettikokojakauman. Pakettikokojakauma on riippuvainen hämäännytettävästä sovelluksesta. DarkComet-ohjelmiston osalta tilanne on selkeämpi, koska "hämäännytettävä" sovellus on aina sama. Sama pätee myös XtremeRAT-ohjelmistoon.

Scramblesuit pyrkii hämäännyttämään liikennevuon tilastolliset piirteet, eli pakettien pituudet ja IAT:n. Tämän vuoksi pakettien pituuksien ja pakettien välisten saapumisaikojen perusteella ei voida teoriassa tehdä tilastollista analyysia. Tästä kuitenkin havaittiin, että scramblesuit tuottaa profiililtaan samankaltaisen jakauman toistettaessa hämäännytys OpenVPN-liikenteelle. Sama testi eri maksimi viivästysparametreilla (10ms ja 100ms) näytti tuottavan profiililtaan samankaltaisen jakauman. Pakettien välisten saapumisaikojen obfuskointi heikentää liikenteen läpäisykykyä merkittävästi, joten se on kytketty oletuksena pois päältä. Käytännössä pakettien välisten saapumisaikojen perusteella voitaisiin toteuttaa luokittelua, koska todennäköisesti ominaisuus on suurimmasta osasta toteutuksista poiskytkettynä.

9. YHTEENVETO

Tämän työn tarkoitus oli tutkia hämäännytetyn ja haitallisen liikenteen luokittelua olemassa olevien avoimen lähdekoodin ohjelmistojen avulla. Työssä perehdyttiin kirjallisuuden perusteella liikenteen luokittelun ja hämäännytttämisen eri menetelmiin. Teorian ja aikaisemman tutkimustiedon perusteella valikoitiin työssä käytetyt avoimen lähdekoodin luokitteluohjelmistot sekä avoimen lähdekoodin hämäännyttämisohjelmistot. Kirjallisuus toimi myös käytännön tukena tässä työssä toteutettujen haitallisisten ohjelmistojen liikenne- ja käyttätymisanalyysissä. Työn empiiriinen osuus toteutettiin suljetussa testausympäristössä, jossa hämäännytettyä ja haitallista liikennettä generoitiin eri ohjelmistojen avulla. Liikenteen luokittelua mitattiin testausympäristössä generoidun liikenteen avulla. Lisäksi hämäännytettyä ja haitallista liikennettä analysoitiin pakettitien pituuksien ja pakettien saapumisten välisen ajan perusteella (packet interarrival time). Työn tärkeimmiksi tutkimuskysymyksiksi muodostuivat:

- Kykenevätkö avoimen lähdekoodin pakettien syvätarkastusohjelmistot luokittelemaan hämäännytettyä liikennettä?
- Kuinka hämäännytettyä liikennettä voidaan havaita?
- Kuinka muodostetaan liikennettä, jota on vaikea havaita?

Alakysymyksinä olivat mm. kuinka paljon suoritetaan vääriä liikenteen luokitustuloksia ja voidaanko IDS-sääntöjä hyödyntää tehokkaasti hämäännytetyn liikenteen havainnointiin.

Keskeisimpinä tuloksina liikenteen luokittelun osalta olivat:

- Hämäänyttämätön liikenne luokiteltiin pääosin oikein.
- Eri bittorrent-liikennne luokiteltiin oikein ja MSE-hämäännyttämismenetelmällä ei ollut vaikutusta luokituksen tulokseen.
- Suurin osa hämäännytetystä liikenteestä luokiteltiin tuntematon-luokkaan. Erityisesti säännöllisiä lausekkeita hyödyntäviä DPI-kirjastoja vastaan toteutettu fte-proxy kykeni hämäännyttämään libprotoident-kirjaston.
- Haitallisen liikenteen luokittelussa nDPI suoritti HTTP-luokituksen Xtreme-RAT-ohjelmiston asennuskättelystä. Myös Suricata dekoodasi kyseiset paketit HTTP-liikenteeksi. DarkComet-liikenne luokiteltiin tuntemattomaksi kaikkien DPI-ohjelmistojen toimesta. Suricata ei suorittanut oletuksena "emerging threats" -säännöstön avulla hälytystä kummankaan RAT-ohjelmiston liikenteestä.

Oletusasetuksilla DPI-kirjastot kykenevät luokittelemaan MSE-protokollan avulla hämäännytettyä liikennettä. Muiden obfuskointiprotokollien avulla hämäännytetty liikenne luokiteltiin pääosin tuntematon luokkaan, kuten alkuperäinen oletus olikin. Vääriä positiivisia luokituksia syntyi eniten säännöllisiin lausekkeisiin perustuvassa DPI-kirjastossa, eli libprotoident-ohjelmistossa. Toisaalta tämä selvisi parhaiten hämäännyttämättömän liikenteen luokittelusta.

Tässä työssä esitettyjä hämäännyttämismenetelmiä kyetään luokittelemaan, mikäli näitä varten kehitetään omat protokolladekooderit. Selkeimmät tapaukset ovat ne, joiden hyötykuormasta voidaan lukea selväkielisiä merkkijonoja ja yhteydenmuodostuskättely on tunnistettavissa. Esimerkiksi esitettyjen analyysien avulla obfs2-protokollan ja Xtreme-RAT-ohjelmiston tunnistamisesta voidaan luoda omat algoritmit. Obfs2-protokollan tapauksessa on mahdollista luokitella myös hämäännytetty sovellus, koska liikenteen salausavain voidaan johtaa passiivisen salakuuntelun avulla.

Obfs3- ja scramblesuit-protokollien luokittelu selvästi haastavinta, kun tarkastellaan avoimen lähdekoodin obfuskointimenetelmiä. Molempien protokollien kättelyn havaitseminen on tehty haastavaksi tasajakauma-D-H:n avulla. Koska käytössä on D-H-avaintenvaihtomenetelmä, on salausavaimia käytännössä mahdoton johtaa passiivisen salakuuntelun avulla. Kyseisten protokollien luokittelukeinoksi jää oikeastaan liikennevuon tilastolliset ominaisuudet, jotka scramblesuit pyrkii hämäännyttämään.

Tässä työssä laadittujen pakettikokojakaumien ja IAT-jakaumien perusteella voidaan avoimen lähdekoodin hämäännyttämismenelmien osalta tehdä johtopäätöksiä vain testattuun aineistoon verraten. Näitä tuloksia ei voida yleistää, koska testiaineisto on suppea johtuen obfuskoitavien sovelluksien määrästä. Verrattaessa testiaineiston häämäännyttämättömään liikenteeseen, oli fteproxy, obfs- ja scramblesuit-protokollien pakettikokojakaumat erotettavissa ja lisäksi ne olivat keskenään erilaiset. Toisaalta fteproxyn ja obfsproxyn tukemia obfuskointimenetelmiä ei voida verrata keskenään, koska obfsproxyn liikennettä tunneloitiin OpenVPN-sovelluksen kanssa. Obfs-protokollien pakettikokojakaumat ovat erotettavissa muista obfuskointimenetelmistä, mutta niiden pakettikokojakaumat ovat lähes täsmälleen samanlaiset keskenään. Tämä pätee silloin, kun obfuskoitavana sovelluksena on OpenVPN. Se, kuinka paljon obfuskoitava sovellus vaikuttaa jakaumiin, ei selviä tässä työssä. Jos obfuskoitavana sovelluksena olisi jokin muu sovellus kuin OpenVPN, on mahdollista, että pakettijakauma on erilainen. Pakettikokojakauman perusteella on siis mahdollista tunnistaa obfs-protokollien avulla hämäännytetty OpenVPN-liikenne. Obfs2- ja obfs3-protokolla voidaan erottaa toisistaan hyötykuorman tarkistuksen avulla. Obfs2-protokollan hyötykuormasta voidaan lukea neljä tavua pitkä taika-arvo, joka varmistaa luokituksen. Myös fteproxy voidaan tunnistaa, tässä esitetyn pakettikokojakauman perusteella silloin, kun hämäännytettävänä sovelluksena on HTTP. Fteproxyn pakettikokojakauma oli hyvin erilainen verrattuna normaalin HTTP-liikenteen jakaumaan.

IAT-jakauman osalta voidaan verrata vain OpenVPN, obfs3- ja scramblesuit-protokollien jakaumia. Näiden osalta tulos on sama pakettikokojakauman kanssa. Jakaumat ovat keskenään erilaiset. Toisaalta scramblesuit-protokollan avulla hämäännytetty OpenVPN-liikenne tuotti profiililtaan samankaltaisen IAT-jakauman toistettaessa liikennettä eri IAT-viivästysparametreilla. Scramblesuit-protokollan avulla tulisi kuitenkin tehdä lisätestejä, jotta voitaisiin todeta korrelaatio IAT-jakaumassa.

Haitallista liikennettä tarkasteltaessa on XtremeRAT-ohjelmiston luokittelu suoraviivaista johtuen sen lähettämistä selväkielisistä viesteistä. Myös ohjelmiston lähettämät deflate-algoritmilla pakatut viestit voidaan dekoodata tässä työssä esitetyin keinoin. DarkComet-liikenteen luokittelu on selvästi haastavampaa, koska liikenne on salakirjoitettu esijaetun avaimen avulla. Liikenne on siis kättelystä alkaen salakirjoitettua, jolloin tunnitettavia piirteitä ei juurikaan ole. Liikennettä voitaisiin luokitella yksinkertaisesti oletussalausavaimien avulla, jolloin liikenne havaittaisiin, kun oletussalausavainta ei vaihdeta. Tämä ei kuitenkaan ole kovin kestävä ratkaisu. DarkComet-liikenteen osalta, kuten oikeastaan kaiken salakirjoitetun liikenteen osalta, on mielekästä tutkia vain liikennevuon tilastollisia piirteitä. Pakettikokojakauman perusteella DarkComet-liikenne voidaan erottaa muista tässä työssä esitetyistä sovelluksista. Tämä pätee myös XtremeRAT-ohjelmistoon, vaikkakin luokitus voidaan tuottaa varmemmin tarkastelemalla hyötykuormaa. Eniten DarkComet-liikennettä muistutti pakettikokojakauman perusteella fteproxy-liikenne (HTTP). Kummankin RAT-ohjelmiston osalta pakettikokojakauman tulos voidaan yleistää, koska liikenteen tuottaa aina sama sovellus, kyseinen RAT-ohjelmisto.

Anomalian tunnistusta IDS-järjestelmän avulla voidaan hyödyntää hämäännytetyn liikenteen etsintään, mutta sen avulla ei voida tietenkään tunnistaa ja luokitella hämäännytysmenetelmää. Tämä pätee myös salakirjoitettuun liikenteeseen. DPI-kirjastojen tapaan myös IDS-järjestelmään voidaan kuitenkin kirjoittaa omat protokolladekooderit obfuskointiprotokollia varten. Suricatassa tähän voidaan hyödyntää lua-laajennusta.

Jos pakettien hyötykuorma ja liikennevuo on obfuskoitu jonkin todellisen liikenteen perusteella, on luokittelu erittäin haastavaa. Tässä diplomityössä esitelty Dust-protokolla pyrkii toteuttamaan liikennevuon hämäännyttämisen jonkin todellisen liikenteen perusteella. Liikennevuo obfuskoidaan siten, että se saa pakettikokojakauman ja IAT-jakauman esimerkiksi oikean HTTP-liikenteen perusteella. Dust obfuskoi myös hyötykuorman, mutta se ei naamioi tätä toiseksi sovellukseksi fteproxyn tapaan. Naamioitaessa hyötykuorma fte:n tapaan, on luokitus erittäin haastavaa. Luonnollisesti hyötykuorman naamiointi pitäisi toteuttaa saman sovelluksen mukaisesti, kuin liikennevuo on naamioitu. Teoriassa, yhdistämällä Dust-protokollan ja fteproxyn parhaat ominaisuudet, voidaan kätkeä alkuperäisen sovelluksen tuottama liikenne siten, että hämäännytysmenetelmää ja alkuperäistä sovellusta on erittäin vaikea luokitella. Haasteeksi tämänkaltaisissa hämäännyttämismenetelmissä nousee varmasti protokollan suorituskyky, sillä obfuskointikerros aiheuttaa varsinaisen hyötykuorman pienentämistä jokaista pa-

kettia kohden ja liikennevuon obfuskointi vaikuttaa liikenteen läpäisykykyyn. Esimerkiksi scramblesuit-protokollassa liikennevuon obfuskointi oletuksena kytketty pois päältä, johtuen merkittävästä liikenteen läpäisykyvyn heikentymisestä.

Hämäännytetyn liikenteen havaitsemisen tai sen luokittelun tutkimusta voitaisiin jatkaa näiden tulosten perusteella vielä edelleen. Hämäännytysmenetelmiä varten voitaisiin kehittää tässä työssä esitettyjä protokolladekoodereita, joiden avulla selvitettäisiin hämäännytetyn liikenteen havaitsemista riippumatta hämäännytetystä sovelluksesta. Lisäksi tilastollisen analyysin käyttämistä hämäännytetyn liikenteen luokittelussa tulisi tutkia lisää, sillä sen merkitys tulee jatkossa kasvamaan hyötykuorman salakirjoittaminen kasvun myötä.

LÄHTEET

Alcock, S., & Nelson, R. (2012). Libprotoident: Traffic Classification Using Lightweight Packet Inspection. WAND Network Research Group, Tech. Rep. [WWW]. [Viitattu: 21.1.2015]. Saatavissa: http://wand.net.nz/sites/default/files/lpi.pdf.

Bellare, M., Ristenpart, T., Rogaway, P., & Stegers, T. (2009). Format-preserving encryption. Selected Areas in Cryptography: 16th Annual International Workshop, SAC 2009, Calgary, Alberta, Canada, August 13-14, 2009, Revised Selected Papers, s. 295-312.

Ben-Kiki, O., Evans, C., & döt Net, I. (2009). YAML Ain't Markup Language (YAML™) Version 1.2. [WWW]. [Viitattu: 20.11.2015]. Saatavissa: http://www.yaml.org/spec/1.2/spec.html#Introduction.

Bujlow, T., Riaz, T., & Pedersen, J. M. (2012a). A method for classification of network traffic based on C5. 0 Machine Learning Algorithm. Computing, Networking and Communications (ICNC), 2012 International Conference on, Maui, HI, s. 237-241.

Bujlow, T., Riaz, T., & Pedersen, J. M. (2012b). Classification of HTTP traffic based on C5. 0 Machine Learning Algorithm. In Computers and Communications (ISCC), 2012 IEEE Symposium on, Cappadocia, s. 882-887.

Bujlow, T., Carela-Español, V., & Barlet-Ros, P. (2013). Comparison of Deep Packet Inspection (DPI) Tools for Traffic Classification. Universitat Politècnica de Catalunya. Department of Computer Architecture (DAC). Tech. Rep, version 3. [WWW]. [Viitattu: 15.1.2015]. Saatavissa: http://personals.ac.upc.edu/pbarlet/reports/dpi.upctech2013.pdf

Bujlow, T. (2014). Usefulness of the results - a forgotten evaluation metric of traffic identification tools. [WWW]. [Viitattu: 2.2.2015]. Saatavissa: http://luca.ntop.org/sgr2014/docs/dpi_presentation_pisa.ppt.

Center for Applied Internet Data Analysis. (2010). [WWW]. [Viitattu: 3.2.2015]. Saatavissa: https://www.caida.org/research/traffic-analysis/pkt_size_distribution/graphs.xml.

Corwin, E. H. (2011). Deep Packet Inspection: Shaping the Internet and the Implications on Privacy and Security. Information Security Journal: A Global Perspective. Vol.20(6), s. 311-316.

Curtis, S. (2014). Illegal downloading: four strikes and then... nothing. The Telegraph. [WWW]. [Viitattu: 23.2.2015]. Saatavissa: http://www.telegraph.co.uk/technology/news/10979918/Illegal-downloading-four-strikes-and-then...-nothing.html.

Danelutto, M., Deri, L., De Sensi, D., & Torquati, M. (2014). Deep Packet Inspection on Commodity Hardware using FastFlow. Parallel Computing: Accelerating Computational Science and Engineering (CSE). Vol.25, s. 92-93.

David, G. (2014). ITKST48 - Advanced Persistance Threat, luentomateriaali. Jyväskylän yliopisto.

de Freitas, N. (2013). Decision trees for classification. Course taught in 2013 at UBC by Nando de Freitas. [WWW]. [Viitattu: 13.2.2015]. Saatavissa: https://www.youtube.com/watch?v=-dCtJjlEEgM

Denbow, S., & Hertz, J. (2012). Pest control: taming the rats. Matasano Security. [WWW]. [Viitattu: 7.11.2015]. Saatavissa: http://www.steptoecyberblog.com/files/2012/11/PEST-CONTROL1.pdf

Deri, L., Martinelli, M., Bujlow, T., & Cardigliano, A. (2014). nDPI: Open-source high-speed deep packet inspection. In Wireless Communications and Mobile Computing Conference (IWCMC), 2014 International, Nicosia, s. 617-622.

Digitoday. (2011). Nitro vakoili kemian yrityksiä 4 kuukautta myös Suomessa. [WWW]. [Viitattu: 13.4.2015]. Saatavissa: http://www.digitoday.fi/tietoturva/2011/11/01/nitro-vakoili-kemian-yrityksia-4-kuukautta-myos-suomessa/201115846/66.

Duta, C. L., Mocanu, B. C., Vladescu, F. A., & Gheorghe, L. (2014). Randomness evaluation framework of cryptographic algorithms. International Journal on Cryptography and Information Security (IJCIS). Vol. 4(1), s. 31-49.

Dyer, K. P., Coull, S. E., Ristenpart, T., & Shrimpton, T. (2013). Protocol misidentification made easy with format-transforming encryption. In Proceedings of the 2013 ACM SIGSAC conference on Computer & communications security, s. 61-72.

Dyer, K. (2014). Fteproxy issues: --upstream-format XXX --downstream-format YYY #164. [WWW]. [Viitattu: 12.12.2015]. Saatavissa: https://github.com/kpdyer/fteproxy/issues/164.

Fidelis Cybersecurity. (2015). Looking at the Sky for a Dark Comet. Fidelis Threat Advisory # 1018. [WWW]. [Viitattu 6.11.2015]. Saatavissa: https://www.fidelissecurity.com/sites/default/files/FTA_1018_looking_at_the_sky_for_a_dark_comet.pdf

Fifield, D. 2015. Obfs2-version.nse (detection for obfs2 obfuscation transport). [WWW]. [Viitattu 3.12.2015]. Saatavissa: http://seclists.org/nmap-dev/2015/q1/82.

Fuchs, C. (2012). Implications of Deep Packet Inspection (DPI) Internet Surveillance for Society. The Privacy & Security Research Paper Series. Vol. 1, s.18. [WWW]. [Viitattu 23.4.2015]. Saatavissa: http://www.projectpact.eu/privacy-security-research-paper-series/%231_Privacy_and_Security_Research_Paper_Series.pdf.

Graham, R.J. (2014). XtremeRAT Malware Targets Israeli Government Agency. RobertJGraham.com, Archive for January, 2014. [WWW]. [Viitattu: 10.11.2015]. Saatavissa: http://robertjgraham.com/?m=201401%2Fpage%2F3&paged=8

Gray, R. M. (2013). Entropy and information theory. First Edition, Corrected. Springer-Verlag, New York. Prologue x. [WWW]. [Viitattu: 10.11.2015]. Saatavissa: http://www-ee.stanford.edu/~gray/it.pdf.

Hardening OpenVPN. 2013. [WWW]. [Viitattu: 12.3.2015]. Saatavissa: https://community.openvpn.net/openvpn/wiki/Hardening.

Hjelmvik, E., & John, W. (2010). Breaking and improving protocol obfuscation. Department of Computer Science and Engineering, Chalmers University of Technology. Technical Report No. 2010-05. [WWW]. [Viitattu: 17.2.2015]. Saatavissa: https://www.iis.se/docs/hjelmvik_breaking.pdf .

Horten, M. (2008). Briefing Paper - Deep packet inspection, copyright and the Telecoms Package - DRAFT. Communications and Media Research Institute, University of Westminster. PhD Research – The Political Battle for Online Content in the European Union.

Housley, R. (2004). Using advanced encryption standard (aes) counter mode with ipsec encapsulating security payload (esp). s. 2. [WWW]. [Viitattu: 26.10.2015]. Saatavissa: https://tools.ietf.org/html/rfc3686#section-2.1.

HowItWorks - Libprotoident. (2012). Libprotoident wiki. WAND research group, University of Waikato Computer Science Department. [WWW]. [Viitattu: 24.3.2015]. Saatavissa: https://secure.wand.net.nz/trac/libprotoident/wiki/HowItWorks.

Huffman, D. A. (1952). A method for the construction of minimum redundancy codes. Proceedings of the IRE. Vol.40(9), s. 1098-1101.

Inliniac. (2011). File extraction in Suricata. [WWW]. [Viitattu: 13.10.2015]. Saatavissa: http://blog.inliniac.net/2011/11/29/file-extraction-in-suricata/.

Into the Void. (2012). Bypassing censorship devices by obfuscating your traffic using obfsproxy. [WWW]. [Viitattu: 4.2.2015]. Saatavissa: http://www.void.gr/kargig/blog/tag/obfsproxy/

IT-viikko. (2014). Stallman: Joka kännykässä on takaportti vakoojalle. [Viitattu: 19.2.2015]. Saatavissa: http://www.itviikko.fi/ihmiset-ja-ura/2014/02/12/stallman-joka-kannykassa-on-takaportti-vakoojalle/20142075/7

json.org. Introducing JSON. [WWW]. [Viitattu: 20.11.2015]. Saatavissa: http://json.org/

Jun, L., Shunyi, Z., Yanqing, L., & Zailong, Z. (2007). Internet Traffic Classification Using Machine Learning. Communications and Networking in China, 2007. CHINACOM '07. Second International Conference on, Shanghai, 2007, s. 239-243.

Kadianakis, G., & Mathewson, N. (2013). Obfs2 threat model. [WWW]. [Viitattu: 18.11.2015]. Saatavissa: https://github.com/isislovecruft/obfsproxy/blob/master/doc/obfs2/obfs2-threat-model.txt.

Kauhanen, A-L. (2015). Työryhmä ehdottaa armeijalle ja poliisille lupaa verkkotiedusteluun – viestintäministeriö vastustaa kiivaasti. Helsingin Sanomat. [WWW]. [Viitattu: 29.3.2015]. Saatavissa: http://www.hs.fi/kotimaa/a1420864745560

Khalife, J. M., Hajjar, A., & Díaz-Verdejo, J. (2011). On the Performance of OpenDPI in Identifying P2P Truncated Flows. In AP2PS 2011, The Third International Conference on Advances in P2P Systems, s. 79-84.

Khalife, J. M., Hajjar, A., & Díaz-Verdejo, J. (2013). Performance of OpenDPI in identifying sampled network traffic. Journal of Networks. Vol.8(1), s. 71-81.

Kulkarni, S.R. (2002). Information, Entropy, and Coding. Lecture Notes for ELE201 Introduction to Electrical Signals and System. Princeton University. Chapter 8, s. 9. [WWW]. [Viitattu: 27.10.2015]. Saatavissa: http://www.princeton.edu/~cuff/ele201/kulkarni_text/information.pdf.

Kyberturvallisuuskeskus. (2014). Kohdistettujen haittaohjelmahyökkäyksien uhka on otettava vakavasti. Viestintävirasto. Raportti, s. 4. [WWW]. [Viitattu: 15.10.2015]. Saatavissa: https://www.viestintavirasto.fi/attachments/tietoturva/Kohdistetut_haittaohjelmahyokkaykset_uhka_otettava_vakavasti_raportti_28082014.pdf.

L7-filter | ClearFoundation. ClearFoundation. [WWW]. [Viitattu: 15.4.2015]. Saatavissa: http://l7-filter.clearfoundation.com/#community.

Leblond, É. (2012). Suricata, to 10Gbps and beyond. [WWW]. [Viitattu: 22.4.2015]. Saatavissa: https://home.regit.org/2012/07/suricata-to-10gbps-and-beyond/.

Lehto, T. (2011). RSA: SecurID-hyökkäyksen takana valtio. Tivi. [WWW]. [Viitattu: 18.10.2015]. Saatavissa: http://www.tivi.fi/Arkisto/2011-10-12/RSA-SecurID-hyökkäyksen-takana-valtio-3141161.html.

Lewicki, M. S. (2007). 15-381 Artificial Intelligence: Representation and Problem Solving. School of Computer Science, Carnegie Mellon University. [WWW]. [Viitattu: 18.10.2015]. Saatavissa: http://www.cs.cmu.edu/afs/cs/academic/class/15381-s07/www/slides/041007decisionTrees1.pdf.

Li, X., Ji, Z. Z., & Hu, M. Z. (2005). Stateful Inspection firewall session table processing. Information Technology: Coding and Computing, 2005. ITCC 2005. International Conference on, 2005. Vol. 2, s. 615-620.

Lyda, R., & Hamrock, J. (2007). Using Entropy Analysis to Find Encrypted and Packed Malware. In IEEE Security & Privacy. Vol. 5(2), s. 40-45.

Mandiant. (2013). APT1:Exposing One of China's Cyber Espionage Units. Mandiant_APT1_Report.pdf. [WWW]. [Viitattu: 3.11.2015]. Saatavissa: http://intelreport.mandiant.com/Mandiant_APT1_Report.pdf.

McCanne, S., & Jacobson, V. (1993). The BSD packet filter: A new architecture for user-level packet capture. USENIX Association. Proceedings of the USENIX Winter 1993 Conference Proceedings on USENIX Winter 1993, s. 2-2. [WWW]. [Viitattu 6.3.2015]. Saatavissa: http://www.tcpdump.org/papers/bpf-usenix93.pdf.

nDPI - Quick Start Guide. (2013). Open and Extensible GPLv3 Deep Packet Inspection Library. [WWW]. [Viitattu 2.3.2015]. Saatavissa:http://www.ntop.org/wp-content/uploads/2013/12/nDPI_QuickStartGuide.pdf.

Neagu, A. (2015). Security Alert: Infamous DarkComet RAT Used In Spear Phishing Campaigns. Heimdal Security. [WWW]. [Viitattu 6.11.2015]. Saatavissa: https://heimdalsecurity.com/blog/darkcomet-rat-phishing-campaigns/

Nordström, H., Laitinen, K., Lundelin, M., Herrala, J., Sjöblom, J., Honkanen, K., Nurminen, M. (2015). Suomalaisen tiedustelulainsäädännön suuntaviivoja. Tiedonhankintalakityöryhmän mietintö, s. 50. [WWW]. [Viitattu 30.1.2015]. Saatavissa: http://www.defmin.fi/files/3016/Suomalaisen_tiedustelulainsaadannon_suuntaviivoja-.pfd.

OHJ-2550 Tekoäly. (2009). Tampereen teknillinen yliopisto. Luentokalvot, Koneoppiminen. [WWW]. [Viitattu 12.2.2015]. Saatavissa: http://www.cs.tut.fi/kurssit/OHJ-2550/AI09-9.pdf.

Open Information Security Foundation. (2010). Flow and Stream handling. Suricata User Guide, Suricata Rules. [WWW]. [Viitattu 20.10.2015]. Saatavissa: https://redmine.openinfosecfoundation.org/projects/suricata/wiki/Suricatayaml.

Open Information Security Foundation. (2011). Flow-keywords. Suricata User Guide, Suricata Rules. [WWW]. [Viitattu 20.10.2015]. Saatavissa: https://redmine.openinfosecfoundation.org/projects/suricata/wiki/Flow-keywords.

Open Information Security Foundation. (2014). Interacting via Unix Socket. Suricata wiki. Suricata User Guide. [WWW]. [Viitattu: 10.11.2015]. Saatavissa: https://redmine.openinfosecfoundation.org/projects/suricata/wiki/Interacting_via_Unix_Socket.

Open Information Security Foundation. (2015). Protocol Anomalies Detection. Suricata wiki. Suricata User Guide. [WWW]. [Viitattu: 20.2.2016]. Saatavissa: https://redmine.openinfosecfoundation.org/projects/suricata/wiki/Protocol_Anomalies_Detection.

OpenVPN Community 2013. Traffic Obfuscation, wiki. [WWW]. [Viitattu 19.2.2015]. Saatavissa: https://community.openvpn.net/openvpn/wiki/TrafficObfuscation.

OpenVPN wiki, TrafficObfuscation, 2013. [WWW]. [Viitattu 19.2.2015]. Saatavissa: https://community.openvpn.net/openvpn/wiki/TrafficObfuscation.

Osborne, C. (2014). Does your ISP throttle BitTorrent traffic? Find out. CNET. [WWW]. [Viitattu 4.2.2015]. Saatavissa: http://www.cnet.com/news/does-your-isp-throttle-bittorrent-traffic-find-out/.

Othman, D. (2013). Bypassing censorship by using obfsproxy and openVPN , SSH Tunnel, 2013. Dlshad Othman's personal blog. [WWW]. [Viitattu 20.2.2015]. Saatavissa: http://dlshad.net/?p=135.

ParisinLA project. Prototype Action Recommending Information System. USC - University of Southern California. [WWW]. [Viitattu 13.3.2015]. Saatavissa: http://www.usc.edu/dept/ancntr/Paris-in-LA/Analysis/discovery.html.

Pepitone, J. (2012). SOPA explained: What it is and why it matters. CNN Money. [WWW]. [Viitattu 23.1.2015]. Saatavissa: http://money.cnn.com/2012/01/17/technology/sopa_explained/.

Raff, A. (2014). Xtreme RAT Strikes Israeli Organizations Again. Seculert blog. [WWW]. [Viitattu: 20.4.2015]. Saatavissa: http://www.seculert.com/blog/2014/01/xtreme-rat-strikes-israeli-organizations-again.html.

Rajahalme, J., Amante, S., Jiang, S., & Carpenter, B. (2011). IPv6 flow label specification. RFC 6437. IETF, 2011, s.2. [WWW]. [Viitattu 21.1.2015]. Saatavissa: https://tools.ietf.org/html/rfc6437.

Ruckert, M. (2005). Understanding MP3: Syntax, Semantics, Mathematics and Algorithms. Springer Verlag, New York. s.156.

RuleQuest Search. (2012). Is See5/C5.0 Better Than C4.5? [WWW]. [Viitattu 27.3.2015]. Saatavissa: http://rulequest.com/see5-comparison.html.

Safavian, S. R., & Landgrebe, D. (1991). A survey of decision tree classifier methodology. In IEEE Transactions on Systems, Man, and Cybernetics. Vol. 21(3), s.660-674.

Ruohonen, K. (1999). Koodaus- ja informaatioteoria. TTKK:n matematiikan kurssien Koodausteoria ja Informaatioteoria luentomoniste. [WWW]. [Viitattu 12.3.2015]. Saatavissa: http://math.tut.fi/~ruohonen/KIT.pdf.

Santinumi, A. (2012). DarkComet Analysis – Understanding the Trojan used in Syrian Uprising. Infosec institute. [WWW]. [Viitattu: 5.11.2015]. Saatavissa: http://resources.infosecinstitute.com/darkcomet-analysis-syria/.

Scarfone, K., & Mell, P. (2007). Guide to intrusion detection and prevention systems (idps). NIST special publication 800-94, s.2-3 - 2-5. [WWW]. [Viitattu 16.3.2015]. Saatavissa: http://csrc.nist.gov/publications/nistpubs/800-94/SP800-94.pdf.

Schneier, B. (2013). Schneier on Security: Evading Internet Censorship. [WWW]. [Viitattu 21.4.2015]. Saatavissa: https://www.schneier.com/blog/archives/2013/08/evading_interne.html.

Shen, C., & Huang, L. (2012). On Detection Accuracy of L7-filter and OpenDPI. Networking and Distributed Computing (ICNDC), 2012 Third International Conference on, Hangzhou, China, 2012, s.119-123.

Shubhangi, G., Sarika, R., & Shital, G. (2013). Implementation of Network Traffic Classification by Using MLA. International Journal of Scientific and Research Publications 2013, Vol.3(5), s.578-580.

Singel, R. (2007). Comcast Sued Over BitTorrent Blocking. Wired. [WWW]. [Viitattu 28.4.2015]. Saatavissa: http://www.wired.com/2007/11/comcast-sued-ov/.

Tor Bug Tracker & Wiki. (2015a). Pluggable Transports. [WWW]. [Viitattu 28.11.2015]. Saatavissa: https://trac.torproject.org/projects/tor/wiki/doc/PluggableTransports.

Tor Bug Tracker & Wiki. (2015b). A Childs Garden Of Pluggable Transports. [WWW]. [Viitattu 3.12.2015]. Saatavissa: https://trac.torproject.org/projects/tor/wiki/doc/AChildsGardenOfPluggableTransports.

Tor Project (a). Obfsproxy. [WWW]. [Viitattu 14.3.2015]. Saatavissa: https://www.torproject.org/projects/obfsproxy.html.en.

Tor Project (b). Obfs2 protocol specification. Obfs2 (the twobfuscator). [WWW]. [Viitattu 14.3.2015]. Saatavissa:https://gitweb.torproject.org/pluggable-transports/obfsproxy.git/tree/doc/obfs2/obfs2-protocol-spec.txt.

Tor Project (c). Obfs3 protocol specification. Obfs3 (the threebfuscator). Pluggable transport for obfuscated traffic. [WWW]. [Viitattu 14.3.2015]. Saatavissa: https://gitweb.torproject.org/pluggable-transports/obfsproxy.git/tree/doc/obfs2/obfs2-protocol-spec.txt.

Tor Project. (2013). GFW actively probes obfs2 bridges. [WWW]. [Viitattu 22.11.2015]. Saatavissa: https://trac.torproject.org/projects/tor/ticket/8591.

Totty, B., Sayer, M., Reddy, S., Aggarwal, A., & Gourley, D. (2002). HTTP: The Definitive Guide. O'Reilly Media, Inc, s.97-99.

Obfsproxy. N.d. Obfsproxy. [WWW]. [Viitattu: 16.1.2015]. Saatavissa: https://www.torproject.org/projects/obfsproxy.html.en.

TreePlan. (2014). Chapter 14 Introduction to Decision Tree p.157. TreePlan Software, Free Book Chapters for TreePlan. [WWW]. [Viitattu: 27.1.2015]. Saatavissa: http://www.treeplan.com/chapters/introduction-to-decision-trees.pdf.

Ubik, S., & Žejdl, P. (2010). Evaluating Application-Layer Classification Using a Machine Learning Technique over Different High Speed Networks. Systems and Networks Communications (ICSNC), 2010 Fifth International Conference on, Nice, s.387-391.

VAHTI 6/2009. (2009). Kohdistetut hyökkäykset. Valtiovarainministeriönjulkaisuja, s.9. [WWW]. [Viitattu: 3.10.2015]. Saatavissa: https://www.vahtiohje.fi/c/document_library/get_file?uuid=c423fe00-d6d4-4bdf-99dc-1f7ac8ecf977&groupId=10128&groupId=10229.

Villeneuve, N., & Bennett, J. (2012). Detecting APT Activity with Network Traffic Analysis. Trend Micro Incorporated Research Paper, s.1. [WWW]. [Viitattu: 12.11.2015]. Saatavissa: http://www.trendmicro.com/cloud-content/us/pdfs/security-intelligence/white-papers/wp-detecting-apt-activity-with-network-traffic-analysis.pdf.

Villeneuve, N., Haq, T., & Moran, N. (2013). Operation Molerats: Middle East Cyber Attacks Using Poison Ivy. FireEye. FireEye Blogs, Threat Research. [WWW]. [Viitattu: 12.11.2015]. Saatavissa: https://www.fireeye.com/blog/threat-research/2013/08/operation-molerats-middle-east-cyber-attacks-using-poison-ivy.html.

Villeneuve, N., & Bennett, J.T. (2014). XtremeRAT: Nuisance or Threat? FireEye. FireEye Blogs, Threat Research. [WWW]. [Viitattu: 14.11.2015]. Saatavissa: https://www.fireeye.com/blog/threat-research/2014/02/xtremerat-nuisance-or-threat.html.

WAND Network Research Group. (a). Libtrace. The University of Waikato. [WWW]. [Viitattu: 17.11.2015]. Saatavissa: http://research.wand.net.nz/software/libtrace.php.

WAND Network Research Group. (b). Libflowmanager. The University of Waikato. [WWW]. [Viitattu: 17.11.2015]. Saatavissa: http://research.wand.net.nz/software/libflowmanager.php.

WAND Network Research Group. (c). Libwandevent. The University of Waikato. [WWW]. [Viitattu: 17.11.2015]. Saatavissa: http://research.wand.net.nz/software/libwandevent.php.

Watson, B. W., & Blox, I. P. (2014). Elastic deep packet inspection. Cyber Conflict (CyCon 2014), 2014 6th International Conference On, Tallinn, s. 241-253.

Wiley, B. (2011). Dust: A blocking-resistant internet transport protocol. Technical report. [WWW]. [Viitattu: 30.4.2015]. Saatavissa: http://blanu. net/Dust. pdf.

Wiley, B. (2013). Defeating Protocol Classifying Internet Filters with Dust. ACM at the University of Illinois at Urbana-Champaign. [WWW]. [Viitattu: 30.4.2015]. Saatavissa: https://www.youtube.com/watch?v=IfLh3tr2amk.

Wiley, B. (2014a). Dust Specification, Dust v2, Polymorphic Protocol Engine for Circumventing Filters. [WWW]. [Viitattu: 3.5.2015]. Saatavissa: https://github.com/blanu/Dust/blob/e9515bc862654e4a41491cc54bb3d10a7f0000dd/docs/DustSpecification.pdf.

Wiley, B. (2014b). Dust-tools. [WWW]. [Viitattu: 3.5.2015]. Saatavissa: https://github.com/blanu/Dust-tools.

Winter, P., Pulls, T., & Fuss, J. (2013). Scramblesuit: A polymorphic network protocol to circumvent censorship. Proceedings of the 12th ACM Workshop on Workshop on Privacy in the Electronic Society, Berlin, s.213-224.

Winter, P. (2014). ScrambleSuit Protocol Specification. [WWW]. [Viitattu: 20.11.2015]. Saatavissa: https://github.com/NullHypothesis/scramblesuit/blob/master/doc/scramblesuit-spec.txt.

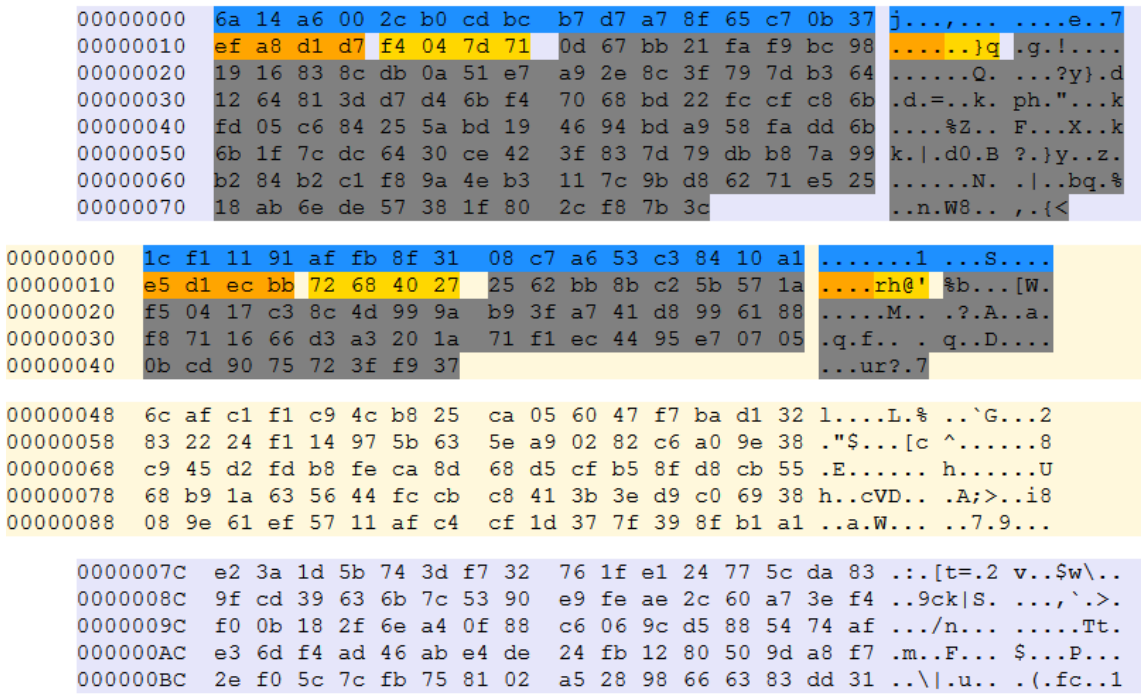
You, Y. P., & Tsai, S. C. (1999). On The Random Property of Compressed Data via Huffman Coding. In Proceedings of 1999 National Computer Symposium (NCS '99), Vol. 1, s.418-420.

Zhioua, S., & Langar, M. (2014). Traffic Analysis of Web Browsers. CEUR Workshop Proceedings. Proceedings of the Formal Methods for Security Workshop. Vol. 1158, s.20-33.

LIITE A: OBFS2-KÄTTELYN WIRESHARK-ESIMERKKI JA OBFS2-VIESTIEN PURKU PYTHON-KOODIN AVULLA (TOR BUG TRACKER & WIKI 2015B)

Wireshark-esimerkissä:

- sinisellä pohjalla on siemen-arvo (palvelin = resp_seed / asiakas = init_seed)
- oranssilla pohjalla on merkitty salakirjoitetu "taika-arvo" (engl. magic value)
- keltaisella pohjalla on merkitty salakirjoitettu PADLEN-arvo
- tumman harmaalla pohjalla on merkitty salakirjoitettu täyte



Asiakkaan siemen-arvo voidaan lukea suoraan pakettikaappauksesta, joka tässä tapauksessa on 1cf11191affb8f3108c7a653c38410a1. Siemen-arvosta voidaan johtaa avain ja käynnistysvektoriarvo AES-CTR-moodille. Tämä voidaan tehdä alla olevan python-esimerkin avulla:

```
>>> init_seed = "1cf11191affb8f3108c7a653c38410a1".decode("hex")
>>> resp_seed = "6a14a6002cb0cdbcb7d7a78f65c70b37".decode("hex")
>>> init_secret = sha256("Initiator obfuscated data" + init_seed + resp_seed
+ "Initiator obfuscated data").digest()
>>> init_key = init_secret[0:16]
>>> init_ctr = init_secret[16:32]
>>>  aes  =  AES.new(init_key,  AES.MODE_CTR,  counter=Counter.new(128,  ini-
tial_value=long(init_ctr.encode("hex"), 16)))
>>> ciphertext = "6cafc1f1c94cb825ca056047f7bad132".decode("hex")
>>> print aes.decrypt(ciphertext).encode("hex")
16030100ef010000eb0303923472ed41
```

LIITE B: OBFS2 AKTIIVINEN LUOTAUS-SKRIPTI (FIFIELD 2015)

```
local nmap = require "nmap"
local shortport = require "shortport"
local openssl = require "openssl"

author = "David Fifield"
license = "Same as Nmap--See http://nmap.org/book/man-legal.html"
categories = {"version"}

--- @output
-- PORT      STATE SERVICE VERSION
-- 43172/tcp open  obfs2

local function sha256(x)
  return openssl.digest("sha256", x)
end

local function h(x)
  return sha256(x)
end

local function mac(s, x)
  return h(s .. x .. s)
end

local function is_obfs2(data)
  local MAGIC_VALUE = "\x2b\xf5\xca\x7e"

  if data:len() < 20 then
    return false
  end

  local resp_seed = data:sub(1, 16)
  local resp_secret = mac("Responder obfuscation padding", resp_seed)
  local resp_key = resp_secret:sub(1, 16)
  local resp_iv = resp_secret:sub(17, 32)

  local magic = openssl.decrypt("aes-128-ctr", resp_key, resp_iv, da-
ta:sub(17, 20))
  -- print(string.byte(magic, 1), string.byte(magic, 2), string.byte(magic,
3), string.byte(magic, 4))

  return magic == MAGIC_VALUE
end

portrule = function(host, port)
  return port.protocol == "tcp" and port.state == "open"
    and not(shortport.port_is_excluded(port.number, port.protocol))
    and nmap.version_intensity() >= 9
end

function action(host, port)
  local status, err, s

  s = nmap.new_socket()
  status, err = s:connect(host, port)
```

```
  if not status then
    return
  end

  status, data = s:receive_bytes(20)
  if not status then
    s:close()
    return
  end
  s:close()

  if is_obfs2(data) then
    port.version.name = "obfs2"
    nmap.set_port_version(host, port)
  end
end
```

LIITE C: BITTORRENT-FORMAATIN LISÄYS FTEPROXY-OHJELMISTOON (DYER 2014)

```
{
    "bittorrent-request": {
        "regex":"^(\\x13BitTorrent protocol|GET
/announce\\?(info_hash|peer_id|ip|port|uploaded|downloaded|left|event)).*$"
    },
    "bittorrent-response": {
        "regex": "^(\\x13BitTorrent protocol|GET
/announce\\?(info_hash|peer_id|ip|port|uploaded|downloaded|left|event)).*$"
    }
}
```

LIITE D: DARKCOMET RAT -OHJELMISTON KONFIGURAATION DEKOODERI (BREEN 2014)

```
#!/usr/bin/env python
'''
DarkComet Rat Config Decoder
'''

__description__ = 'DarkComet Rat Config Extractor'
__author__ = 'Kevin Breen http://techanarchy.net'
__version__ = '0.1'
__date__ = '2014/03/15'

import sys
import string
from struct import unpack
try:
      import pefile
except ImportError:
      print "Couldnt Import pefile. Try 'sudo pip install pefile'"
from optparse import OptionParser
from binascii import *

def rc4crypt(data, key):
    x = 0
    box = range(256)
    for i in range(256):
        x = (x + box[i] + ord(key[i % len(key)])) % 256
        box[i], box[x] = box[x], box[i]
    x = 0
    y = 0
    out = []
    for char in data:
        x = (x + 1) % 256
        y = (y + box[x]) % 256
        box[x], box[y] = box[y], box[x]
        out.append(chr(ord(char) ^ box[(box[x] + box[y]) % 256]))

    return ''.join(out)

def v51_data(data, enckey):
      config = {"FWB": "", "GENCODE": "", "MUTEX": "", "NETDATA": "",
"OFFLINEK": "", "SID": "", "FTPUPLOADK": "", "FTPHOST": "", "FTPUSER": "",
"FTPPASS": "", "FTPPORT": "", "FTPSIZE": "", "FTPROOT": "", "PWD": ""}
      dec = rc4crypt(unhexlify(data), enckey)
      dec_list = dec.split('\n')
      for entries in dec_list[1:-1]:
            key, value = entries.split('=')
            key = key.strip()
            value = value.rstrip()[1:-1]
            clean_value = filter(lambda x: x in string.printable, value)
            config[key] = clean_value
            config["Version"] = enckey[:-4]
      return config
```

```
def v3_data(data, key):
      config = {"FWB": "", "GENCODE": "", "MUTEX": "", "NETDATA": "",
"OFFLINEK": "", "SID": "", "FTPUPLOADK": "", "FTPHOST": "", "FTPUSER": "",
"FTPPASS": "", "FTPPORT": "", "FTPSIZE": "", "FTPROOT": "", "PWD": ""}
      dec = rc4crypt(unhexlify(data), key)
      config[str(entry.name)] = dec
      config["Version"] = enckey[:-4]

      return config

def versionCheck(rawData):
      if "#KCMDDC2#" in rawData:
            return "#KCMDDC2#-890"

      elif "#KCMDDC4#" in rawData:
            return "#KCMDDC4#-890"

      elif "#KCMDDC42#" in rawData:
            return "#KCMDDC42#-890"

      elif "#KCMDDC42F#" in rawData:
            return "#KCMDDC42F#-890"

      elif "#KCMDDC5#" in rawData:
            return "#KCMDDC5#-890"

      elif "#KCMDDC51#" in rawData:
            return "#KCMDDC51#-890"
      else:
            return None

def configExtract(rawData, key):
      config = {"FWB": "", "GENCODE": "", "MUTEX": "", "NETDATA": "",
"OFFLINEK": "", "SID": "", "FTPUPLOADK": "", "FTPHOST": "", "FTPUSER": "",
"FTPPASS": "", "FTPPORT": "", "FTPSIZE": "", "FTPROOT": "", "PWD": ""}

      pe = pefile.PE(data=rawData)
      rt_string_idx = [
      entry.id for entry in
      pe.DIRECTORY_ENTRY_RESOURCE.entries].index(pefile.RESOURCE_TYPE['RT_RCD
ATA'])
      rt_string_directory =
pe.DIRECTORY_ENTRY_RESOURCE.entries[rt_string_idx]
      for entry in rt_string_directory.directory.entries:
            if str(entry.name) == "DCDATA":

                  data_rva = en-
try.directory.entries[0].data.struct.OffsetToData
                  size = entry.directory.entries[0].data.struct.Size
                  data = pe.get_memory_mapped_image()[data_rva:data_rva+size]
                  config = v51_data(data, key)

            elif str(entry.name) in config.keys():

                  data_rva = en-
try.directory.entries[0].data.struct.OffsetToData
                  size = entry.directory.entries[0].data.struct.Size
                  data = pe.get_memory_mapped_image()[data_rva:data_rva+size]
                  dec = rc4crypt(unhexlify(data), key)
```

```
                  config[str(entry.name)] = filter(lambda x: x in
string.printable, dec)
                  config["Version"] = key[:-4]
      return config

def configClean(config):
      try:
            newConf = {}
            newConf["FireWallBypass"] = config["FWB"]
            newConf["FTPHost"] = config["FTPHOST"]
            newConf["FTPPassword"] = config["FTPPASS"]
            newConf["FTPPort"] = config["FTPPORT"]
            newConf["FTPRoot"] = config["FTPROOT"]
            newConf["FTPSize"] = config["FTPSIZE"]
            newConf["FTPKeyLogs"] = config["FTPUPLOADK"]
            newConf["FTPUserName"] = config["FTPUSER"]
            newConf["Gencode"] = config["GENCODE"]
            newConf["Mutex"] = config["MUTEX"]
            newConf["Domains"] = config["NETDATA"]
            newConf["OfflineKeylogger"] = config["OFFLINEK"]
            newConf["Password"] = config["PWD"]
            newConf["CampaignID"] = config["SID"]
            newConf["Version"] = config["Version"]
            return newConf
      except:
            return config

def run(data):
      versionKey = versionCheck(data)
      if versionKey != None:
            config = configExtract(data, versionKey)
            config = configClean(config)

            return config
      else:
            return None

if __name__ == "__main__":
      parser = OptionParser(usage='usage: %prog inFile outConfig\n' +
__description__, version='%prog ' + __version__)
      (options, args) = parser.parse_args()
      if len(args) > 0:
            pass
      else:
            parser.print_help()
            sys.exit()
      try:
            print "[+] Reading file"
            fileData = open(args[0], 'rb').read()
      except:
            print "[+] Couldn't Open File {0}".format(args[0])
      print "[+] Searching for Config"
      config = run(fileData)
      if config == None:
            print "[+] Config not found"
            sys.exit()
      if len(args) == 2:
            print "[+] Writing Config to file {0}".format(args[1])
            with open(args[1], 'a') as outFile:
```

```
                for key, value in sorted(config.iteritems()):
                      outFile.write("Key: {0}\t Value:
{1}\n".format(key,value))

      else:
            print "[+] Printing Config to screen"
            for key, value in sorted(config.iteritems()):
                  print "   [-] Key: {0}\t Value: {1}".format(key,value)
            print "[+] End of Config"
```

LIITE E: PURETTU DARKCOMET KONFIGURAATIO-TIEDOSTO

```
#BEGIN DARKCOMET DATA --
PWD={0123456789}
MUTEX={DC_MUTEX-J26F2FA}
SID={Dippa01}
FWB={1}
NETDATA={192.168.228.3:80}
GENCODE={2UatFPGUnZGo}
INSTALL={1}
COMBOPATH={7}
EDTPATH={MSDCSC\msdcsc.exe}
KEYNAME={MicroUpdate}
EDTDATE={16/04/2007}
PERSINST={1}
MELT={0}
CHANGEDATE={0}
DIRATTRIB={4}
FILEATTRIB={4}
FAKEMSG={1}
MSGTITLE={Welcome}
MSGCORE={57656C636F6D6520746F204461726B436F6D6574205241542E0D0A496620796F7520
7365652074686973206D6573736167652C206974206D65616E732074686520737475622073756
36365737366756C6C792072756E7320616E6420796F752077696C6C20617065616172200D0A696E
20746865206D617374657220757365722206C6973742E0D0A}
MSGICON={64}
SH1={1}
SH3={1}
SH5={1}
CHIDEF={1}
CHIDED={1}
PERS={1}
OFFLINEK={1}
#EOF DARKCOMET DATA --
```

LIITE F: TYÖSSÄ GENEROIDUN TESTAUSLIIKENTEEN KUVAUS

LIIKENNE / HÄMÄÄNNYTYSMENETELMÄ	LIIKENTEEN KUVAUS	BPF SUODATIN	PCAP-TIEDOSTON KOKO
HTTP-liikenne 1	Windows työaseman selainliikenne (Firefox), ihmisen tuottama verkkoselailu eri web-sivuilla	tcp port http	38,4 Mt
HTTP-liikenne 2	Verkkoselaimen (Firefox) avulla HTTP-lataus ftp.funet.fi -sivustolta. Ladattu tiedosto: freeBSD bootonly.iso	tcp port http	474 Mt
HTTPS (SSL)	Selaimella käynti verkkopankin web-sivustolla	tcp port 443	79 Kt
SSH + SCP (TCP 22)	SSH-yhteyden muodostaminen Linux-palvelimelle, shell-komentojen käyttö ja SCP:n avulla word-dokumentin lataus	tcp port 22	888 Kt
HTTP curl / Fteproxy, TCP 8080	Curl ohjelman avulla HTTP-sivun lataus, obfuskoitu Fteproxyn avulla. Naamiointimenetelmänä käytetty oletusprotokollaa: HTTP. Liikenne ohjattu TCP-porttiin 8080	tcp port 8080	1,84 Mt
HTTP-liikenne 2 / Fteproxy, TCP 80	freeBSD bootonly.iso -tiedoston lataus verkkoselaimen avulla, obfuskoitu Fteproxyn avulla.	tcp port http	444 Mt
HTTP-liikenne 2 / Fteproxy, TCP 8080	Liikenne ohjattu TCP-porttiin 80. Liikenne naamoitu oletusprotokollan mukaan: HTTP freeBSD bootonly.iso -tiedoston lataus verkkoselaimen avulla, obfuskoitu Fteproxyn avulla. Liikenne ohjattu TCP-porttiin 8080. Liikenne naamoitu oletusprotokollan mukaan: HTTP	tcp port 8080	441 Mt
SSH / Fteproxy, TCP 80	SSH-yhteyden muodostus testiverkon Linux-tietokoneiden välillä ja shell-komentojen käyttö, obfuskoitu Fteproxyn avulla, liikenne ohjattu TCP-porttiin 80. Liikenne naamoitu oletusprotokollan mukaan: HTTP	tcp port http	597 Kt
SSH / Fteproxy, TCP 8080	SSH-yhteyden muodostus testiverkon Linux-tietokoneiden välillä ja shell-komentojen käyttö, obfuskoitu Fteproxyn avulla, liikenne ohjattu TCP-porttiin 8080. Liikenne naamoitu oletusprotokollan mukaan: HTTP	tcp port 8080	442 Kt

Bittorrent	Bittorrent Vuze client. Ubuntu iso-tiedoston lataus (vain osa tiedostosta)	Wireshark	2,01 Gt
Bittorrent / MSE	Bittorrent Vuze client. Liikenne obfuskoitu MSE:n avulla. Ubuntu iso-tiedoston lataus (vain osa tiedostosta)	Wireshark	1,08 Gt
Bittorrent vuze-DHT	Bittorrent Vuze client. Ubuntu iso-tiedoston lataus (vain osa tiedostosta). Ainoastaan UDP DHT-paketit	Wireshark	218 Mt
Bittorrent vuze-DHT / MSE	Bittorrent Vuze client. Ubuntu iso-tiedoston lataus (vain osa tiedostosta). Ainoastaan UDP DHT-paketit. Obfuskoitu MSE:n avulla	Wireshark	114 Mt
HTTP-liikenne 2 + OpenVPN, UDP 50010	freeBSD bootonly.iso -tiedoston lataus verkkoselaimen avulla, salattu OpenVPN-ohjelman avulla. Liikenne ohjattu UDP-porttiin 50010	udp port 50010	10,2 Mt
HTTP-liikenne 2 + OpenVPN, TCP 443	freeBSD bootonly.iso -tiedoston lataus verkkoselaimen avulla, salattu OpenVPN-ohjelman avulla. Liikenne ohjattu TCP-porttiin 443	tcp port 443	213 Mt
HTTP-liikenne 2 + OpenVPN, TCP 1194	freeBSD bootonly.iso -tiedoston lataus verkkoselaimen avulla, salattu OpenVPN-ohjelman avulla. Liikenne ohjattu OpenVPN-oletusporttiin: TCP 1194	tcp port 1194	523 Mt
SSH + SCP + OpenVPN, TCP 1194	SSH-yhteyden muodostaminen Linux-palvelimelle, shell-komentojen käyttö ja SCP:n avulla word-dokumentin lataus. Liikenne salattu lisäksi OpenVPN-ohjelman avulla. Liikenne ohjattu OpenVPN-oletusporttiin: TCP 1194	tcp port 1194	1,19 Mt
OpenVPN (HTTP-liikenne 2) Obfsproxy obfs2, TCP 80	freeBSD bootonly.iso -tiedoston lataus verkkoselaimen avulla. OpenVPN-liikenne obfuskoitu Obfsproxy-ohjelman avulla (obfs2). Liikenne ohjattu TCP-porttiin 80	tcp port http	306 Mt
OpenVPN (HTTP-liikenne 2) Obfsproxy obfs2 + esijaet-tuavain, TCP 80	freeBSD bootonly.iso -tiedoston lataus verkkoselaimen avulla. OpenVPN-liikenne obfuskoitu Obfsproxy-ohjelman avulla (obfs2). Liikenne ohjattu TCP-porttiin 80. Käytössä esijaettu salausavain	tcp port http	304 Mt
OpenVPN (HTTP-liikenne 2) Obfsproxy obfs3, TCP 80	freeBSD bootonly.iso -tiedoston lataus verkkoselaimen avulla. OpenVPN-liikenne obfuskoitu Obfsproxy-ohjelman avulla (obfs3). Liikenne ohjattu TCP-porttiin 80	tcp port http	305 Mt
OpenVPN (HTTP-liikenne 2) Obfsproxy scramblesuit, TCP 80	freeBSD bootonly.iso -tiedoston lataus verkkoselaimen avulla. OpenVPN-liikenne obfuskoitu Obfsproxy-ohjelman avulla	tcp port http	323 Mt

	(scramblesuit). Liikenne ohjattu TCP-porttiin 80		
OpenVPN (SSH+SCP) Obfsproxy obfs2, TCP 80	SSH-yhteyden muodostaminen Linux-palvelimelle, shell-komentojen käyttö ja SCP:n avulla word-dokumentin lataus. OpenVPN-liikenne obfuskoitu Obfsproxy-ohjelman avulla (obfs2). Liikenne ohjattu TCP-porttiin 80	tcp port http	1,12 Mt
OpenVPN (SSH+SCP) Obfs-proxy obfs2 + esijaettuavain, TCP 80	SSH-yhteyden muodostaminen Linux-palvelimelle, shell-komentojen käyttö ja SCP:n avulla word-dokumentin lataus. OpenVPN-liikenne obfuskoitu Obfsproxy-ohjelman avulla (obfs2). Liikenne ohjattu TCP-porttiin 80. Käytössä esijaettu salausavain	tcp port http	1,12 Mt
OpenVPN (SSH+SCP) Obfsproxy obfs3, TCP 80	SSH-yhteyden muodostaminen Linux-palvelimelle, shell-komentojen käyttö ja SCP:n avulla word-dokumentin lataus. OpenVPN-liikenne obfuskoitu Obfsproxy-ohjelman avulla (obfs3). Liikenne ohjattu TCP-porttiin 80.	tcp port http	1,18 Mt
OpenVPN (SSH+SCP) Obfsproxy scramblesuit, TCP 80	SSH-yhteyden muodostaminen Linux-palvelimelle, shell-komentojen käyttö ja SCP:n avulla word-dokumentin lataus. OpenVPN-liikenne obfuskoitu Obfsproxy-ohjelman avulla (scramblesuit). Liikenne ohjattu TCP-porttiin 80.	tcp port http	2,35 Mt
DarkComet RAT 5.3.1	DarkComet RAT -ohjelman palvelinosan asennus uhrin koneelle. Etäkomentojen antaminen uhrin koneelle. Ohjelmiston tarjoamien työkalujen käyttö: etäistunto komentorivikehotteella, työpöydän kaappaus etänä, tiedostojen siirto ja "keylloggerin" käyttö	tcp port http	6,22 Mt
Xtreme RAT 3.7	DarkComet RAT -ohjelman palvelinosan asennus uhrin koneelle. Etäkomentojen antaminen uhrin koneelle. Ohjelmiston tarjoamien työkalujen käyttö: etäistunto komentorivikehotteella, työpöydän kaappaus etänä, tiedostojen siirto ja "keylloggerin" käyttö	tcp port http	14,8 Mt

LIITE G: OPENVPN ASIAKASKONFIGURAATIO

```
client
dev tun
proto tcp
remote 192.168.50.13 1194
resolv-retry infinite
nobind
user nobody
group nogroup
persist-key
persist-tun
tls-auth ta.key 1
cipher AES-256-CBC
keysize 256
auth SHA256
comp-lzo
verb 3
keepalive 10 120
```

LIITE H: PCAP-TIEDOSTOJEN MASSA-AJO JA SOVELLUS-PROTOKOLLAN SUODATUS SURICATAN LOKISTA

ndpireader massa-ajo

```
#!/bin/bash
READER=../example/ndpiReader
PCAPS=`cd pcap; /bin/ls *.pcapng`
# käy läpi pcap-tiedostot hakemistossa ja
# luo tulostiedosto, jos sitä ei ole
build_results() {
    for f in $PCAPS; do
          [ ! -f result/$f.txt ] && $READER -i pcap/$f > result/$f.txt
    done
}
build_results
```

Suricata massa-ajo

```
READER=suricata
PCAPS=`cd /mnt/hgfs/wireshark_captures/; /bin/ls *.pcapng`

build_results() {
    for f in $PCAPS; do
        echo $f
        # create_dir $f
        mkdir -p "/mnt/hgfs/Jaettu_hakemisto/$f"
        $READER -c suricata.yaml -r \
        "/mnt/hgfs/wireshark_captures/${f}" \
        -l " /mnt/hgfs/Jaettu_hakemisto/${f}/" -k none
done
}
build_results
```

Sovellusprotokollan suodatus Suricatan lokista

```
DIRS=`cd /mnt/hgfs/Jaettu_hakemisto/; /bin/ls -d *.pcapng`

build_results() {
    for f in $DIRS; do
        echo $f
        cd "/mnt/hgfs/Jaettu_hakemisto/$f"
        less eve.json | jq '.event_type' | sort | uniq -c | sort -nr >\
        protocols.txt
done
}
build_results
```

LIITE I: NDPI TULOSTIEDOSTO

```
Using nDPI (r1.5.2
(e4923a589c04584a2faa7d9ce35e9875f41b1153:20150521)) [1
thread(s)]
Reading packets from pcap file pcap/XtremeRAT_liikenne.pcapng...
Running thread 0...

nDPI Memory statistics:
     nDPI Memory (once):      91.46 KB
     Flow Memory (per flow):  1.92 KB
     Actual Memory:           1.93 MB
     Peak Memory:             1.93 MB

Traffic statistics:
     Ethernet bytes:        15460726      (includes ethernet
CRC/IFC/trailer)
     Discarded bytes:       0
     IP packets:            9955          of 9955 packets total
     IP bytes:              15221806      (avg pkt size 1529
bytes)
     Unique flows:          162
     TCP Packets:           9955
     UDP Packets:           0
     VLAN Packets:          0
     MPLS Packets:          0
     PPPoE Packets:         0
     Fragmented Packets:    0
     Max Packet size:       52580
     Packet Len < 64:       6520
     Packet Len 64-128:     306
     Packet Len 128-256:    622
     Packet Len 256-1024:   267
     Packet Len 1024-1500:  613
     Packet Len > 1500:     1627
     nDPI throughput:       393.35 K pps / 4.55 Gb/sec
     Traffic throughput:    10.66 pps / 129.32 Kb/sec
     Traffic duration:      934.038 sec
     Guessed flow protos:   160


Detected protocols:
     HTTP                packets: 9955         bytes: 15221806
flows: 162


Protocol statistics:
     Acceptable               15221806 bytes
```

LIITE J: LIBPROTOIDENT ASENNUS

Libprotoident asennuksessa käytetyt komennot suoritusjärjestyksessä:

```
git clone https://github.com/wanduow/libprotoident.git
cd libprotoident
touch NEWS AUTHORS
libtoolize
aclocal
autoconf
autoheader
automake --add-missing
./configure
make
sudo make install
```

LIITE K: LIBPROTOIDENT TULOSTIEDOSTO

```
Unknown_TCP 192.168.228.4 192.168.6.10 80 1113 6 1432123001.125
2017340 410 1c666da5 .fm. 2920 47455420 GET. 205
Unknown_TCP 192.168.228.4 192.168.6.10 80 1114 6 1432123001.138
183960 410 1c666da5 .fm. 2920 47455420 GET. 205
Unknown_TCP 192.168.228.4 192.168.6.10 80 1115 6 1432123002.884
366 1394 580d0a00 X... 3 6d797665 myve 15
Unknown_TCP 192.168.228.4 192.168.6.10 80 1116 6 1432123008.567
370 1394 580d0a00 X... 3 6d797665 myve 15
Unknown_TCP 192.168.228.4 192.168.6.10 80 1117 6 1432123014.193
366 1394 580d0a00 X... 3 6d797665 myve 15
Unknown_TCP 192.168.228.4 192.168.6.10 80 1119 6 1432123019.818
370 1390 580d0a00 X... 3 6d797665 myve 15
Unknown_TCP 192.168.228.4 192.168.6.10 80 1120 6 1432123025.457
370 1394 580d0a00 X... 3 6d797665 myve 15
Unknown_TCP 192.168.228.4 192.168.6.10 80 1121 6 1432123031.287
362 1390 580d0a00 X... 3 6d797665 myve 15
Unknown_TCP 192.168.228.4 192.168.6.10 80 1122 6 1432123036.911
366 1394 580d0a00 X... 3 6d797665 myve 15
Unknown_TCP 192.168.228.4 192.168.6.10 80 1124 6 1432123042.646
366 1398 580d0a00 X... 3 6d797665 myve 15
Unknown_TCP 192.168.228.4 192.168.6.10 80 1125 6 1432123048.287
366 1394 580d0a00 X... 3 6d797665 myve 15
Unknown_TCP 192.168.228.4 192.168.6.10 80 1126 6 1432123054.021
366 1394 580d0a00 X... 3 6d797665 myve 15
Unknown_TCP 192.168.228.4 192.168.6.10 80 1127 6 1432123059.770
366 1394 580d0a00 X... 3 6d797665 myve 15
Unknown_TCP 192.168.228.4 192.168.6.10 80 1129 6 1432123065.506
366 1398 580d0a00 X... 3 6d797665 myve 15
No_Payload 192.168.228.4 192.168.6.10 80 1130 6 1432123071.115 0
0 00000000 .... 0 00000000 .... 0
No_Payload 192.168.228.4 192.168.6.10 80 1130 6 1432123071.535 0
0 00000000 .... 0 00000000 .... 0
No_Payload 192.168.228.4 192.168.6.10 80 1130 6 1432123072.081 0
0 00000000 .... 0 00000000 .... 0
No_Payload 192.168.228.4 192.168.6.10 80 1131 6 1432123077.239 0
0 00000000 .... 0 00000000 .... 0
No_Payload 192.168.228.4 192.168.6.10 80 1131 6 1432123077.771 0
0 00000000 .... 0 00000000 .... 0
No_Payload 192.168.228.4 192.168.6.10 80 1131 6 1432123078.316 0
0 00000000 .... 0 00000000 .... 0
No_Payload 192.168.228.4 192.168.6.10 80 1133 6 1432123083.475 0
0 00000000 .... 0 00000000 .... 0
No_Payload 192.168.228.4 192.168.6.10 80 1133 6 1432123084.004 0
0 00000000 .... 0 00000000 .... 0
No_Payload 192.168.228.4 192.168.6.10 80 1133 6 1432123084.552 0
0 00000000 .... 0 00000000 .... 0
No_Payload 192.168.228.4 192.168.6.10 80 1134 6 1432123089.707 0
0 00000000 .... 0 00000000 .... 0
No_Payload 192.168.228.4 192.168.6.10 80 1134 6 1432123090.130 0
0 00000000 .... 0 00000000 .... 0
```

LIITE L: LIIKENTEEN LUOKITTELUN YHTEENVETOTAULUKKO

LIIKENNE / HÄMÄÄNNYTYSMENETELMÄ	LIIKENTEEN LUOKITTELIJAT JA NIIDEN TUOTTAMA LUOKITUS			
	nDPI	**nDPI -d**	**Libprotoident**	**Suricata**
HTTP-liikenne 1	HTTP*	HTTP*	HTTP	HTTP
HTTP-liikenne 2	HTTP	HTTP	HTTP	HTTP
HTTPS (SSL)	SSL	SSL	HTTPS	TLS
SSH + SCP (TCP 22)	SSH	SSH*	SSH	SSH
HTTP-liikenne 2 + OpenVPN, UDP 50010	Unknown	Unknown	Unknown_UDP	Ei tunnistusta
HTTP-liikenne 2 + OpenVPN, TCP 443	SSL	Unknown	OpenVPN	Ei tunnistusta
HTTP-liikenne 2 + OpenVPN, TCP 1194	OpenVPN	Unknown	OpenVPN	Ei tunnistusta
SSH + SCP + OpenVPN, TCP 1194	OpenVPN	Unknown	OpenVPN	Ei tunnistusta
	nDPI	**nDPI -d**	**Libprotoident**	**Suricata**
HTTP curl / Fteproxy, TCP 80 (HTTP)	HTTP Proxy	Unknown	HTTP	HTTP
HTTP-liikenne 2 / Fteproxy, TCP 80 (HTTP)	HTTP	Unknown	HTTP	HTTP
HTTP-liikenne 2 / Fteproxy TCP 8080 (SSH)	HTTP Proxy	Unknown	SSH	Ei tunnistusta
SSH-liikenne / Fteproxy, TCP 80 (HTTP)	HTTP	Unknown	HTTP	HTTP
SSH-liikenne / Fteproxy, TCP 8080 (HTTP)	HTTP	Unknown	HTTP	HTTP
Bittorrent	Bittorrent *	Bittorrent *	Bittorrent *	P2P BitTorrent *
Bittorrent DHT	Bittorrent *	Bittorrent *	Bittorrent_UDP *	P2P Vuze BT UDP*
Bittorrent DHT / MSE	Bittorrent *	Bittorrent *	Bittorrent_UDP *	P2P Vuze BT UDP*
Bittorrent / MSE	Bittorrent *	Bittorrent *	Bittorrent_UDP*	P2P Vuze BT UDP*
	nDPI	**nDPI -d**	**Libprotoident**	**Suricata**
OpenVPN (HTTP-2) / Obfsproxy obfs2, TCP 80	HTTP	Unknown	Unknown_TCP	Ei tunnistusta
OpenVPN (HTTP-2) / Obfsproxy obfs2 + esijaettuavain, TCP 80	HTTP	Unknown	Unknown_TCP	Ei tunnistusta
OpenVPN (HTTP-2) / Obfsproxy obfs3, TCP 80	HTTP	Unknown	Unknown_TCP	Ei tunnistusta
OpenVPN (HTTP-2) / Obfsproxy scramblesuit, TCP 80	HTTP	Unknown	Unknown_TCP	Ei tunnistusta
OpenVPN (SSH+SCP) / Obfsproxy obfs2, TCP 80	HTTP	Unknown	Unknown_TCP	Ei tunnistusta
OpenVPN (SSH+SCP) / Obfsproxy obfs2 + esijaettuavain, TCP 80	HTTP	Unknown	Unknown_TCP	Ei tunnistusta
OpenVPN (SSH+SCP) / Obfsproxy obfs3, TCP 80	HTTP	Unknown	Unknown_TCP	Ei tunnistusta
OpenVPN (SSH+SCP) / Obfsproxy scramblesuit, TCP 80	HTTP	Unknown	Unknown_TCP	Ei tunnistusta
	nDPI	**nDPI -d**	**Libprotoident**	**Suricata**
DarkComet RAT 5.3.1	HTTP	Unknown	Unknown_TCP	Ei tunnistusta
XtremeRAT 3.7 (asennus + komennot)	HTTP	HTTP *	Unknown_TCP	HTTP
XtremeRAT 3.7 (komennot)	HTTP	Unknown	Unknown_TCP	Ei tunnistusta